Fall 2002

Simulation and close-to-optimal algorithm for the static load balancing of a network of heterogeneous processors

Mark G. Steiner
New Jersey Institute of Technology

ABSTRACT

SIMULATION AND CLOSE-TO-OPTIMAL ALGORITHM FOR THE STATIC LOAD BALANCING OF A NETWORK OF HETEROGENEOUS PROCESSORS

by
Mark G. Steiner

A close-to-optimal linear programming-based algorithm for the static load balancing of a network of heterogeneous processors is described and implemented. Experimental results suggest that the amount of time required by the implementation of the algorithm to balance the loads of the servers as a function of the number of servers has polynomial complexity.

SIMULATION AND CLOSE-TO-OPTIMAL ALGORITHM FOR THE STATIC LOAD BALANCING OF A NETWORK OF HETEROGENEOUS PROCESSORS

by
Mark G. Steiner

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

January 2002

4.2.3 Thesis Committee Form

STUDENT NAME: Mark G. Steiner

ID # 152-70-2492

THESIS TITLE: Simulation and Close-to-Optimal Algorithm for the Static Load Balancing of a Network of Heterogeneous Processors

PROPOSAL NUMBER:

SEMESTER IN WHICH THESIS WILL BE COMPLETED: Fall 2001

DATE: December 20, 2001

SIGNATURES:

THESIS ADVISOR:

DATE: 12/12/01

COMMITTEE MEMBER: [illegible]

DATE: 5/14/02

COMMITTEE MEMBER: [illegible]

DATE: 5/14/02

BIOGRAPHICAL SKETCH

Author: Mark G. Steiner

Degree: Master of Science

Date: January 2002

Date of Birth:

Place of Birth:

Undergraduate and Graduate Education:

- Master of Science in Computer Science
  New Jersey Institute of Technology, Newark, NJ, 2002

- Bachelor of Arts in Chemistry
  Rutgers College, Rutgers University, New Brunswick, NJ, 1987

Major: Computer Science

Presentations, Publications and Patents:

Helen M. Armstrong, Richard Beresis, Joung L. Goulet, Mark A. Holmes, Xingfang Hong, Sander G. Mills, William H. Parsons, Peter J. Sinclair, Mark G. Steiner, Frederick Wong, Dennis M. Zaller, "Preparation of pyrimidine derivatives as Src-family protein tyrosine kinase inhibitor compounds", patent no. WO 2001000213.

Meng-Hsin Chen, Mark G. Steiner, Stephen E. DeLaszlo, Arthur A. Patchett, Matt S. Anderson, Sheryl A. Hyland, H. Russell Onishi, Lynn L. Silver and Christian R. H. Raetz, "Carbohydroxamido-oxazolidines: antibacterial agents that target lipid a biosynthesis", Bioorganic and Medicinal Chemistry Letters, pp. 313-318, vol. 9, 1999.

Meng-Hsin Chen, Mark G. Steiner, Stephen E. DeLaszlo, Arthur A. Patchett, Matt S. Anderson, Sheryl A. Hyland, H. Russell Onishi, Lynn L. Silver and Christian R. H. Raetz, "Carbohydroxamido-oxazolidines as lipid a biosynthesis inhibitors: Target on antibacterial", 216th National Meeting of the American Chemical Society, Boston, Massachusetts, August 23-27, 1998.

Meng-Hsin Chen, Mark G. Steiner, Arthur A. Patchett, K. Cheng, L. Wei, W. W.-S. Chan, B. Butler, T. M. Jacks and R. G. Smith, "Analogs of the orally active growth hormone secretagogue L-162,752", Bioorganic and Medicinal Chemistry Letters, pp. 2163-2169, vol. 6, 1996.

A. E. Weber, M. G. Steiner, P. A. Krieter, A. E. Colletti, J. R. Tata, T. A. Halgren, R. G. Ball, J. J. Doyle, T. W. Schorn, R. A. Stearns, R. R. Miller, P. K. S. Siegl, W. J. Greenlee and A. A. Patchett, "Highly potent, orally active diester macrocyclic human renin inhibitors", Journal of Medicinal Chemistry, p. 3755, vol. 35, 1992.

A. E. Weber, M. G. Steiner, J. R. Tata, J. J. Doyle, T. W. Schorn, P. K. S. Siegl, W. J. Greenlee and A. A. Patchett, "Highly potent, orally active diester macrocyclic human renin inhibitors", 202th American Chemical Society Meeting, New York, NY, August 26-30, 1991.

A. E. Weber, M. G. Steiner, L. Yang, D. S. Dhanoa, J. R. Tata, T. A. Halgren, P. K. S. Siegl, W. H. Parsons, W. J. Greenlee and A. A. Patchett, "Highly potent, orally active P_2-$P_1^{|}$ linked macrocyclic human renin inhibitors", in "Peptides: Chemistry and Biology; Proceedings of the Twelfth American Peptide Symposium, June 16-21, 1991, Cambridge, Massachusetts", J. E. Rivier and J. A. Smith, eds., Leiden: ESCOM Science Publishers, p. 749.

A. E. Weber, M. G. Steiner, D. S. Dhanoa, K. J. Fitch, J. J. Doyle, R. J. Lynch, T. A. Halgren, P. K. S. Siegl, W. H. Parsons, W. J. Greenlee and A. A. Patchett, "Design and synthesis of macrocyclic renin inhibitors", 200th American Chemical Society Meeting, Washington, D.C., August 26-30, 1990.

DEDICATED TO

Marvin E. Steiner, my father, *in memoriam*

ACKNOWLEDGMENT

Thanks are given to Dr. Verkhovsky for his guidance throughout this work. Special thanks are given to Dr. James A. M. McHugh and the other members of my thesis committee.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Problem Statement

The recent advances in computer hardware and networking technologies have made distributed computer systems very attractive. By allocating jobs among the processors in a network and then executing the jobs simultaneously, the time required to complete the jobs can be decreased. The process of assigning jobs to the processors in a way that keeps processor utilization as even as possible is known as load balancing. The goal of this project is to implement a near-optimal, static algorithm that distributes jobs among heterogeneous processors in a network.

1.2 Previous Works

Load balancing policies can be categorized as being either static or dynamic [1]. When the workload is known at the start, static load balancing strategies can be used that distribute the work across the processors before execution. Static load balancing algorithms consider the utilization experienced by each processor in the network, and assign to these processors loads that are inversely proportional to processor utilization. By doing this, all processors would be able to finish at nearly the same time. Static load balancing algorithms are often referred to as "initial placement algorithms" since the distribution of jobs to host processors occures prior to the start of execution and once a host has been selected and the job started, the job is not reallocated to another host. This results in lower placement costs and minimal communication delays since the run-time events experienced by the host processors are ignored.

In contrast to static load balancing algorithms, dynamic load balancing algorithms consider the run-time events that influence the state of a host and the state of the system overall [1]. These algorithms are suitable for cases where the work load is not exactly known ahead of time or the composition of the work load has changed during execution [2]. In these systems, the host processors and the work load are monitored to ensure that the current work load allocation is consistent with near equal processor utilization among the hosts [3]. Since loads on processors must be reallocated at run time, dynamic load balancing policies are more complicated than static policies and require more computation to determine the allocation of loads. Due to the increased communication required among processors, the network interconnection topology greatly influences the overall performance of a dynamic load balancing process [4]. In some cases with a large number of processors, a saturation effect can be observed where adding additional processors to the network results in an increase in the performance of the system that is far less than linear due to the need to communicate with additional processors and the increase in overall network traffic [5, 6, 7, 8]. The communicational and computational overheads of dynamic policies may be substantial enough to negate many of the benefits of dynamic load balancing systems [9, 10, 11]. Furthermore, the rapid advancement of processor capacity has amplified the effect of network communication time resulting in decreased processor utilization [12]. The best solution may be a hybrid of static and dynamic load balancing [12, 13, 14].

These static and dynamic load balancing systems can be further categorized as either centralized or distributed [15]. In centralized systems, one node in the network is responsible for allocating the workload among the remaining processing nodes in the system [16, 17]. In decentralized systems, several or all the processing nodes are responsible for the workload allocation by either accepting work from another node or transferring work to another node [18].

Load balancing systems can be categorized as either optimal or sub-optimal based upon the degree to which they achieve their objective functions. Examples of the objective functions are the minimization of job completion time, maximization of system resources or maximization of system throughput [1]. Systems that are static and sub-optimal have been extensively studied and many heuristics have been used. These include graph theory [5, 19, 20, 21, 22, 23], list scheduling [24, 25, 26, 27, 28, 29], bin-packing [7, 30], genetic algorithms [31,32, 33] and others [34, 35, 36, 37, 38, 39]. It has been shown that the problem of optimal, static load balancing is NP complete [40, 41] and many optimal, static load balancing algorithms have been proposed. These algorithms use techniques such as branch and bound [42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52], integer programming [6, 53], shortest path graphs [54, 55, 56, 57, 58], non-linear optimization [59] and others [60, 61, 62,63, 64].

1.3 Approximate Algorithm for Client-Server Load Balancing [38]

This algorithm deals with a decision making support of load balancing, i.e., how to distribute jobs among servers in such a way that the latest response will be as small as possible. In more general terms, this problem can be formulated as a one-stage scheduling problem where m jobs must be assigned to n machines in such a way that the latest job will be finished as early as possible. Several settings of the problem and a corresponding algorithm for decision making are shown below.

1.3.1 Notations and Data

Users: i=1,2,…,m;

Servers: k=1,2,…,n;

Processing capacity of the k-th server (in MB/sec) $= c_k$;

Query of the i-th user (in MB) $= q_i$;

Current backlog at k-th server (in MB) $= b_k$;

Current delay at k-th server (in sec) $= d_k$; then $d_k := b_k / c_k$. (1)

1.3.2 Quantitative Model

It is assumed that every query is non-dividable. This is plausible when every query is small.

Let z_{ik} be a portion of the i-th query sent to the k-th server. Then all variables must satisfy the following constraints: $0 \le z_{ik} \le 1$; $z_{i1} + z_{i2} + .. + z_{in} = 1$; (2)

For every k=1,..,n holds the inequality

$$(q_1 z_{1k} + .. + q_m z_{mk}) / c_k + sign(z_{1k} + .. + z_{mk}) d_k \le t ; \qquad (3)$$

{here $sign(x)$=0 if x=0; $sign(x)$=1 if x>0 else $sign(x)$= -1};

and $$z_{11}^2 + .. + z_{1n}^2 + z_{21}^2 + .. + z_{2n}^2 + .. + z_{m1}^2 + .. + z_{mn}^2 \ge m . \qquad (4)$$

All variables z_{ik} must be selected in such a way that the latest-time response for all queries will be minimal. This combinatorial problem is *NP*-hard. Indeed, although there is no explicit requirement that every z_{ik} variable is either 0 or 1, the constraints (2) and (4) hold if and only if all variables z_{ik} *are* 0-1 variables. This means that the entire i-th query is either sent to the k-th server or not sent at all to this server.

Theorem: The variables satisfy all the constraints (1) and (3), if and only if every variable is either equal to 0 or 1.

Proof: It follows from the observation that every variable satisfies the inequalities

$$1/4 \geq z_{ik}(1-z_{ik}) \geq 0 \qquad (5)$$

and the equality holds if and only if the variable is 0-1 integer. The right inequality implies that for every i=1,..,m $\quad 1 = z_{i1}+..+z_{in} \geq z_{i1}^2+..+z_{in}^2 \quad$ (6)

and all the equalities hold if and only if every variable is 0-1 integer. Finally, the inequality (4) holds if and only if every variable is 0-1 integer.

1.3.3 Geometric Interpretation

The variables in the system (2)-(4) must satisfy 2n linear constraints (2) and one quadratic constraint (4). The latter represents a sphere of radius $\sqrt{m}$. This problem can be divided onto m separate problems: for a fixed $1 \leq i \leq m$, find and every k=1,..,n, find $0 \leq z_{ik} \leq 1$; $\quad z_{i1}+z_{i2}+..+z_{in}=1$ and $z_{i1}^2+..+z_{in}^2=1$. It is not hard to visualize it for the case where m=n=2: for every i=1,2 one must find points of intersection of the straight line and circumference with the radius=1.

1.3.4 Examples

In the following considerations it is assumed that the queries q_i are large and as a result, divisible into parcels: $\{u_{i1},\ldots,u_{in}\}$, where u_{ik} is a portion of the i-th user's query sent to the k-th server. All queries must be processed:

for every i=1,2,…,m, the equality holds $\quad u_{i1}+..+u_{in}=q_i \quad$ (7)

All queries sent to k-th server must be processed: the following inequality holds for the k-th server $t \ge (u_{1k} + .. + u_{mk} + b_k)/c_k$. (8)

Comment: The backlogs at some servers can be so large that there is no reason to send to them new queries.

Let P be a set of all servers participating in the processing of the current queries. Then for all $k \notin P$ $u_{ik} = 0$. (9)

Assignment criterion: To assign all queries in such a way that all of them will be processed in minimal time t.

Criterion of optimization {minimax performance}:

$$\min_{u_{ik}} \max_{k \in P} (c_k t - b_k) . \quad (10)$$

Property of optimal solution: Let $Q := q_1 + .. + q_m$. (11)

If $t > (u_{1k} + .. + u_{mk} + b_k)/c_k$, then for all i=1,..,m $u_{ij} = 0$ (12)

Let $x^+ = x$ if $x \ge 0$, otherwise $x^+ = 0$. Then t can be determined from the equation:

$$(c_1 t - b_1)^+ + .. + (c_n t - b_n)^+ = Q \quad (13)$$

This is a combinatorial problem with worst-case complexity of $O(2^n)$.

1.3.4.1 Example 1

Let $q_1 = q_2 = 3; q_3 = 2.5; q_4 = 1.5$; $b_1 = 40; b_2 = 28; b_3 = 53.3$; $c_1 = .3; c_2 = .25; c_3 = .15$.

Then t=23.64 and for every k $a_k = c_k t - b_k$: $a_1 = 3.5; a_2 = 4.16; a_3 = 2.34$. The following Table 1 shows all u_{ik} assignments: $u_{11} = 3$; $u_{21} = .5$; $u_{22} = 2.5$ and so on:

Table 1

$a_k \setminus q_i$	3	3	2.5	1.5
3.5	3	.5	0	0
4.16	0	2.5	1.66	0
2.34	0	0	.84	1.5

Special case: If for all *k*=1,2,..,*s* holds $d_k = d$, then $a_k := Qc_k / p_s$.

Here $p_s := c_1 + .. + c_s$. (14)

1.3.4.2 Example 2

The queries, processing capacities of the servers and the backlogs for *k*=1,2 are the same as in Example 1. The only difference now is that $b_3 = 200$. In this case only the first two servers will be used to process these queries. Thus *t*=27.91; $a_1 = 4.775; a_2 = 5.225; a_3 = 0$. Finally, Table 2 provides the solution.

Table 2

$a_k \setminus q_i$	3	3	2.5	1.5
4.775	3	1.775	0	0
5.225	0	1.225	2.5	1.5
0	0	0	0	0

1.3.5 Distributed Assignment

$$u_{ik} := q_i a_k / Q \qquad (15)$$

This assignment splits every query into *n* assignments/requests, i.e., the total number of assignments is equal *ms*. It will be demonstrated that in general case the total number of assignments can be reduced to at most $m + n - 1$.

1.3.5.1 Saturation Algorithm

1. Let $Q := q_1 + .. + q_m$; let d_k be current delay for k-th server and t be time required to process all queries by all servers and let

   $d_1 < d_2 < .. < d_n < d_{n+1} = \infty$ (16)

2. $p_1 := c_1;\ r_1 := b_1$ (17)

3. for s=1,2,..,n do $p_s := p_{s-1} + c_s;\ r_s := r_{s-1} + b_s;\ w_s := d_{s+1} r_s - p_s$; (18)

4. $\{ w_0 := 0;\ w_n := \infty \}$; if $w_{s-1} < Q \le w_s$ (19)

   then $t := (Q + r_s) / p_s$ (20)

5. $P = \{1,2,..,s\}$; {the set of participating servers}; (21)

6. for k=1,s do $a_k := (t - d_k) c_k = c_k t - b_k$ (22)

Justification of the algorithm: If (19) holds, then t must satisfy the equation

$$(c_1 t - b_1) + .. + (c_s t - b_s) = Q \tag{23}$$

1.3.5.2 The Assignment Algorithm

```
Step1: Assign $j := 1; r := 1$;
Step2: $u_{jr} := \min(q_j, a_r)$;
Step3: if $q_j < a_r$ then $j := j + 1; a_r := a_r - u_{jr}$;
       {the case $q_m \le a_r, r < s$ is impossible since $\sum_{k \in P} a_k = Q$};     goto step2;
       if $q_j > a_r$ then $r := r + 1; q_j := q_j - u_{jr}$;
       {the case $q_j \ge a_n, j < s$ is impossible because $\sum_{k \in P} a_k = Q$}; goto step2;
       if $q_j = a_r$ and $\{ j \ne m;\ r \ne s \}$ then $j := j+1$; $r := r+1$;     goto step2;
       if $q_m = a_s$ then stop.
```

Comment: There are at most $m!n!$ different assignments each having $m+n-1$ positive components [65]. These are the basic solutions. This additional set of possible optimal solutions allows usage of a second criterion of optimization. It is not difficult to demonstrate that there are at least m-n+1 positive job assignments such that each must be processed by one server only.

1.3.6 Two-criteria Problem and its Algorithm

In this case, the clients must pay a fee f_{ik} for every MB of a request sent from the i-th client to the k-th server. Then the total payment by all clients is equal

$$F:= f_{11}u_{11}+..+f_{1n}u_{1n}+f_{21}u_{21}+..+f_{2n}u_{2n}+..+f_{m1}u_{m1}+..+f_{mn}u_{mn}\,. \qquad (24)$$

In this case the *second criterion* of optimization selects such non-negative values of all control variables u_{ik} that minimize the total fee F in (24), satisfies the constraints (4) and for every k=1,..,s, holds the equation $u_{1k}+..+u_{mk}=a_k\,.$ (25)

1.3.6.1 Numerical Example

The values of all inputs { q_i ,b_k ,c_k ,f_{ik} } are provided in Table 3.

Table 3

(b_k, c_k) \ q_i	q_1=10	q_2=12	q_3=15	q_4=18	q_5=22	q_6=23	q_7=27
(b_1,c_1)=(8,4)	f_{11}=5	f_{21}=6	f_{31}=1	f_{41}=3	f_{51}=2	f_{61}=4	f_{71}=2
(b_2,c_2)=(6,2)	f_{12}=5	f_{22}=4	f_{32}=4	f_{42}=1	f_{52}=1	f_{62}=2	f_{72}=1
(b_3,c_3)=(12,3)	f_{13}=3	f_{23}=1	f_{33}=5	f_{43}=6	f_{53}=1	f_{63}=5	f_{73}=3
(b_4,c_4)=(12,2)	f_{14}=2	f_{24}=2	f_{34}=1	f_{44}=5	f_{54}=5	f_{64}=7	f_{74}=1

Then Q=127 and the delays are $d_1=2; d_2=3; d_3=4; d_4=6$. By the Saturation Algorithm

(17)-(22) $$p_1=4; p_2=6; p_3=9; p_4=11;\ r_1=8; r_2=14; r_3=26; r_4=38; \qquad (26)$$

$$w_1 = 20; w_2 = 50; w_3 = 147. \tag{27}$$

Since $50<Q<147$, $P=\{1,2,3\}$ and the earliest time when all queries will be processed is $t=(127+26)/9=17$. Hence, the optimal loads are equal:

$$a_1 = 60; a_2 = 28; a_3 = 39; a_4 = 0 \tag{28}$$

Table 4 shows the sum of all loads is equal to Q.

Table 4

$a_k \ \backslash \ q_i$	q_1=10	q_2=12	q_3=15	q_4=18	q_5=22	q_6=23	q_7=27
$a_1 = 60$	**10**	**12**	**15**	**18**	**5**	0	0
$a_2 = 28$	0	0	0	0	**17**	**11**	0
$a_3 = 39$	0	0	0	0	0	**12**	**27**

Hence the total communication fee is equal

F=50+72+15+54+10+17+22+60+81=381.

There are 30240 basic solutions how to distribute the queries among the servers. It is easy to check that the assignments in Table 5 satisfy all the constraints:

Table 5

$a_k \ \backslash \ q_i$	q_1=10	q_2=12	q_3=15	q_4=18	q_5=22	q_6=23	q_7=27
$a_1 = 60$	0	0	**15**	0	**5**	**13**	**27**
$a_2 = 28$	0	0	0	**18**	0	**10**	0
$a_3 = 39$	**10**	**12**	0	0	**17**	0	0

Yet, the total communication fee for the assignments in Table 5 is substantially smaller:

F=15+10+42+54+18+20+30+12+17=218,

i.e., the initial assignments provided in the Table 4 require a 75% larger communication fee, than the optimal assignments. Actually, this set of assignments is also optimal from the point of view of the *second* criterion.

Thus, from Table 5, we determined the following load balancing which is optimal from the points of view of *both* criteria:

- the 1st job is to be processed by the 3rd server;
- the 2nd job by the 3rd server;
- the 3rd job by the 1st server;
- the 4th job by the 2nd server;
- the 5th job by the 1st *and* the 3rd servers;
- the 6th job by the 1st *and* the 2nd servers;
- and finally, the 7th job is to be processed by the 1st server.

1.3.6.2 Analysis of Complexity

1. The saturation algorithm complexity is $O(m+n\log n)$.
2. The assignment algorithm complexity is $O(m+n)$.
3. The minimal-fee-assignment algorithm requires in average $m+n$ iterations. Every iteration requires $O(mn)$ operations. Thus the complexity of this algorithm is of order $O[(m+n)mn]$.
4. Hence the overall complexity of the bi-criterial solution is $O[(m+n)mn]$. This complexity can be reduced if instead of the exact minimal communication-fee solution we apply an algorithm that seeks a solution with "reasonably small" communication fees.

1.3.6.3 Approximate Algorithm

*Step*1: Sort all values f_{ik} in ascending order $\{$complexity $O[(mn\log(mn)]\}$.

*Step*2: Let f_{jr} be the smallest fee; $u_{jr} := \min(q_j, a_r)$; {after this step the dimension of the problem is reduced: either the *j*-th query will be completely assigned to the *k*-th server or the *r*-th server load will be used to process the j-th query}.

*Step*3: repeat Step2 until all queries will be assigned to all servers.; {this step will be applied at most $m+n-1$ times}.

It is easy to see that the overall complexity of this algorithm is O[(*mn*log(*mn*)]. This time is required to sort the set of the specific fees $\{f_{11},..,f_{mn}\}$ in ascending order. However, they need to be presorted only once. Hence the average complexity of the approximation algorithm for a long period of operation is substantially smaller. If the approximate algorithm is applied to the problem with the inputs given in Table 3, the resulting solution is: $u_{31} = 15; u_{42} = 18; u_{52} = 10; u_{23} = 12; u_{53} = 12; u_{71} = 27; u_{13} = 10; u_{61} = 18; u_{63} = 5$

This solution requires a communication fee *F*=248, which is about 13.76% worse than the minimal fee required by solution provided in Table 5.

CHAPTER 2

LOAD BALANCING ALGORITHM IMPLEMENTED IN THIS DOCUMENT

The centralized, static load balancing algorithm developed by B.Verkhovsky [66] and implemented in this document, uses linear programming techniques [67]. As Section 2.1 explains, the problem of finding an optimal solution is combinatorial in nature with complexity $O(2^n)$. This problem is a set of 2^n linear programming (LP) problems. Because of the difficulty in obtaining an optimal solution, an approximate algorithm was implemented in this document instead. It is described in Sections 2.1 through 2.5 below.

2.1 Quantitative Model of the Load Balancing Algorithm

Let: m be the number of jobs

n be the number of processors

a_i be the amount of the jth job to be processed, where $j = 1,2,\ldots, m$

b_k be the current backlog at the kth processor, where $k = 1,2,\ldots, n$

(the backlog, or initial delay, is the amount of time that the processor requires to complete jobs that were assigned to that processor prior to the start of the load balancing algorithm)

c_{jk} be the processing capacity of the kth processor for the jth job

t_k be the time-span of the kth processor (time-span is the amount of time a processor requires to complete the job loads assigned to it in addition to the backlog of the processor)

x_{jk} be the amount of the jth job that is assigned to the kth processor

(an instance of x_{jk} is also referred to as a load in this document)

A be the set of active processors (processors designated to accept load x_{jk})

$\overline{A}$ be the set of inactive processors (processors not designated to accept load)

Comment: All jobs may be measured in "natural" job units. For example, voice jobs can be measured in minutes of conversation, video jobs can be measured in pixels, text jobs can be measured in pages, etc. All t_k are measured in the same time units (milliseconds, seconds, etc.). All c_{jk} are measured in units of time units/"natural" job units, where the time units are the same for all c_{jk}, for example, milliseconds/pixel for video jobs or milliseconds/page for text jobs.

Then: The objective function is to select all $x_{jk} \geq 0$ such that

$\sum_{k \in A} x_{jk} = a_j$, $\sum_{j=1}^{n} c_{jk}x_{jk} + b_k \leq t$ for $\forall k \in A$, $b_k \geq t$ for $\forall k \in \overline{A}$ and make t as small as possible.

From this quantitative model, it follows that the problem is a set of many linear programming (LP) problems since there are 2^n combinations of how to select the set A of active processors.

2.2 Description of a "cycle" and Redistribution Variables

The loads on the processors are balanced when $t_k = t$ for $\forall k \in A$. If $t_g < t$, then x_{jg} must be increased to balance the loads. Since $\sum_{k \in A} x_{jk}=a_j$, this implies a load x_{jk} must be decreased. Again because of constraint $\sum_{k \in A} x_{jk}=a_j$, this implies that load x_{fk} must be increased which again implies a load x_{fz} must be decreased and $g,z \in A$. Loads are added to and removed from processors unitil $t_k = t$ for $\forall k \in A$. In this document, the amount of loads given to and removed from processors that satisfies these constraints are referred to as "redistribution variables". The graph formed by all of the redistribution variables is referred to as a "cycle" in this document. Table 6 below shows a simple scenario where $t_3<t$ and r_1 amount of job 1 is to be given to the newly activated processor 3 to increase t_3 so that $t_3'=t'$. Here, r_1 and r_2 are the redistribution variables. A "+" sign indicates that part of this job is to be added to this processor and a "-" sign indicates that load is to be removed from this processor. The entries "$+r_1$", "$-r_1$", "$+r_2$" and "$-r_2$" constitute the redistribution cycle. Only $x_{jk} > 0$ are shown.

Table 6

processor number					t_k
1			$x_{1,3}$ - r_2	$x_{1,4}$	$t_1=t$
2	$x_{2,1}$ - r_1	$x_{2,2}$	$x_{2,3}$ $+r_2$		$t_2=t$
3	+ r_1				$t_3<t$

Table 7 shows another possible scenario.

Table 7

processor number					t_k
1	$x_{1,1}$ - r_2		$x_{1,3}$	$x_{1,4}$	$t_1=t$
2	$x_{2,1}$ - r_1	$x_{2,2}$			$t_2=t$
3	$+r_1+r_2$				$t_3<t$

When the redistribution variables are applied, that is added or subtracted from their respective loads, all resulting x_{jk} must be positive. If this constraint is violated and x_{qw} <0, all redistribution variables must be scaled linearly such that the resulting value of x_{qw} = 0 when the scaled redistribution variables are applied. In this, case there are now $m+n$-2 load entries since x_{qw} = 0 and x_{qw} is dropped from the basic set of variables. Since there must be $m+n$-1 load entries and $t_l < t$ another $x_{j,k}$ must be increased and therefore another redistribution cycle must be found. This is repeated until $t_k = t$ for $\forall k$ E A.

2.3 Selection of x_{jk}

When $t_q < t$ for k E A, a new job must be found for processor q to work on where x_{jq}=0. That is, processor q has not yet been assigned part of job j. x_{jq} is selected as follows:

Find CR_j for all j = 1 to m where x_{jq}=0 and

$$\text{CR}_j = c_{jq}a_j \,/\, \left(\sum_{k=1}^{q-1} c_{jk}x_{jk}\right)$$

(CR is an abbreviation for Capacity Ratio: the ratio of c_{jq} and the weighted average capacity for job j for the other active processors).

Selection criteria: Choose min(CR_j) if a cycle exists when x_{jq} is introduced as a basic variable. Else, exclude job j and choose the next least CR. Repeat until a cycle is found.

2.4 A Small Numerical Example

The following is a small example that goes through the steps and of the algorithm. The notations described in Sections 2.1, 2.2 and 2.3 are used. Table 8 represents the data given to the load balancing algorithm as input.

Table 8

	a_j	2	15	4	17	
b_k						t_k
1		$c_{1,1}$: 9	$c_{2,1}$: 4	$c_{3,1}$: 4	$c_{4,1}$: 3	1
5		$c_{1,2}$: 1	$c_{2,2}$: 3	$c_{3,2}$: 6	$c_{4,2}$: 2	5
15		$c_{1,3}$: 4	$c_{2,3}$: 1	$c_{3,3}$: 3	$c_{4,3}$: 2	15
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

The first two processors are activated and all of the job loads are distributed to them via a knapsack algorithm. Activated processors are indicated by *. All other processors are inactive, meaning they are not yet designated to receive job load. The resulting load distribution is shown in Table 9 below.

Table 9

	a_j	2	15	4	17	
b_k						t_k
1 *		$c_{1,1}$: 9	$c_{2,1}$: 4 $x_{2,1}$: 9.86	$c_{3,1}$: 4 $x_{3,1}$: 4	$c_{4,1}$: 3	56.43
5 *		$c_{1,2}$: 1 $x_{1,1}$: 2	$c_{2,2}$: 3 $x_{2,2}$: 5.14	$c_{3,2}$: 6	$c_{4,2}$: 2 $x_{4,2}$: 17	56.43
15		$c_{1,3}$: 4	$c_{2,3}$: 1	$c_{3,3}$: 3	$c_{4,3}$: 2	15
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

Since the delay (15) of the next inactive processor is less than the time (56.43) of the previously balanced processors, activate the next processor. Then a job is found for the newly activated processor 3 to work on. The job is selected as described in Section 2.3. The Capacity Ratios are calculated as:

$$CR_1= 4*2/((9*0)+(1*2))= 4$$

CR_2= 1*15/((4*9.86)+(3*5.14))= 0.273
CR_3= 3*4/((4*4)+(6*0))= 0.75
CR_4= 2*17/((3*0)+(2*17))= 1

Since the ratio is minimum for column 2, part of job 2 will be given to processor 3, the newly activated processor. A redistribution cycle is found where r load of job 2 will be given to processor 3, r_1 load will be removed from processor 2 and r_2 load will be removed from processor 1. Since a_2 is constant, $r = r_1 + r_2$. This redistribution cycle is shown in Table 10.

Table 10

	a_j	2	15	4	17	
b_k						t_k
1 *		$c_{1,1}$: 9	$c_{2,1}$: 4 $x_{2,1}$:9.86-r_2	$c_{3,1}$: 4 $x_{3,1}$: 4	$c_{4,1}$: 3	56.43
5 *		$c_{1,2}$: 1 $x_{1,1}$: 2	$c_{2,2}$: 3 $x_{2,2}$:5.14-r_1	$c_{3,2}$: 6	$c_{4,2}$: 2 $x_{4,2}$: 17	56.43
15 *		$c_{1,3}$: 4	$c_{2,3}$: 1 +r_1+r_2	$c_{3,3}$: 3	$c_{4,3}$: 2	15
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

A system of linear equations involving the redistribution variables is constructed and solved with the condition that when the redistribution variables are applied to their respective loads, t_k for all active processors will become equal. This time is represented as t in the system of linear equations below.

$$0r_1 + -4r_2 - t = -56.43$$
$$-3r_1 + 0r_2 - t = -56.43$$
$$1r_1 + 1r_2 - t = -15.0$$

When the system is solved, r_1 = 8.72, r_2 = 6.54 and t = 30.26. If these redistribution variables (r_1 and r_2) are applied, that is r_1 load of job 2 is removed from processor 2 and r_2 load of job 2 is removed from processor 1, $x_{2,2}$ becomes 5.14-8.72 = -3.58. Since negative loads are not feasible, the redistribution variables must be scaled linearly by a multiple so that when the redistribution variables are applied, the load that would have become negative ($x_{2,2}$) becomes instead exactly zero. The scaling multiplier (SM) is calculated as SM= $x_{2,2}/ r_1 = 5.14/8.72 = 0.58945$. Scaling r_1 and r_2 by SM gives

$$r_{1\ \text{scaled}} = r_1 * 0.58945 = 5.14$$
$$r_{2\ \text{scaled}} = r_2 * 0.58945 = 3.86$$

These scaled redistribution variables are then applied to their respective loads resulting in the state shown in Table 11.

Table 11

	a_j	2	15	4	17	
b_k						t_k
1 *		$c_{1,1}$: 9	$c_{2,1}$: 4 $x_{2,1}$: 6.00	$c_{3,1}$: 4 $x_{3,1}$: 4	$c_{4,1}$: 3	41.00
5 *		$c_{1,2}$: 1 $x_{1,1}$: 2	$c_{2,2}$: 3 $x_{2,2}$: 0.00	$c_{3,2}$: 6	$c_{4,2}$: 2 $x_{4,2}$: 17	41.00
15 *		$c_{1,3}$: 4	$c_{2,3}$: 1 $x_{2,3}$: 9.00	$c_{3,3}$: 3	$c_{4,3}$: 2	24.00
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

The table entry $x_{2,3}$: 0.00 will be omitted to make the table easier to read and to indicate that it has been remove from the basic solution. Since t_3 is still less than that of the other active processors, another $x_{j,3}$ must be increased. This is done as in the previous case except that job 2 is now excluded since the $x_{2,3}$ is no longer zero. Using the previously calculated CR$_j$, CR$_j$ is minimum for j= 3. However, part of this job cannot be allocated to the processor 3 since a redistribution cycle cannot be found that includes all processors. Processor 1 can give load to processor 3 but processor 2 cannot transfer its load to either processor 1 or processor 3. Thus, at this point, processor 3 cannot be given part of job 3.

Excluding j=2,3, CR$_j$ is minimum for j= 4. A load redistribution cycle can be found including $x_{4,3}$ and is shown in Table 12.

Table 12

b_k	a_j	2	15	4	17	t_k
1 *		$c_{1,1}$: 9	$c_{2,1}$: 4 $x_{2,1}$:6.00-r_2	$c_{3,1}$: 4 $x_{3,1}$: 4	$c_{4,1}$: 3	41.00
5 *		$c_{1,2}$: 1 $x_{1,1}$: 2	$c_{2,2}$: 3 $x_{2,2}$: 0.00	$c_{3,2}$: 6	$c_{4,2}$: 2 $x_{4,2}$: 17-r_1	41.00
15 *		$c_{1,3}$: 4	$c_{2,3}$: 1 $x_{2,3}$:9.0+r_2	$c_{3,3}$: 3	$c_{4,3}$: 2 +r_1	24.00
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

Once again, system of linear equations involving the redistribution variables is constructed and solved with the condition that when the redistribution variables are applied to their respective loads, t_k=t for all active processors.

$$\begin{aligned}
0r_1 &+ -4r_2 - t = -41.00\\
-2r_1 &+ 0r_2 - t = -41.00\\
2r_1 &+ 1r_2 - t = -24.00
\end{aligned}$$

When the system is solved, r_1 = 3.78, r_2 = 1.89 and t = 33.44. Unlike the previous case, if these redistribution variables are applied, that is r_1 load of job 4 is removed from processor 2 and r_2 load of job 2 is removed from processor 1, all loads will still be positive. Therefore, scaling the redistribution variables is not required. They can be applied directly resulting in a state shown in Table 13 below.

Table 13

	a_j	2	15	4	17	
b_k						t_k
1 *		$c_{1,1}$: 9	$c_{2,1}$: 4 $x_{2,1}$: 4.11	$c_{3,1}$: 4 $x_{3,1}$: 4	$c_{4,1}$: 3	33.44
5 *		$c_{1,2}$: 1 $x_{1,1}$: 2	$c_{2,2}$: 3 $x_{2,2}$: 0.00	$c_{3,2}$: 6	$c_{4,2}$: 2 $x_{4,2}$: 13.22	33.44
15 *		$c_{1,3}$: 4	$c_{2,3}$: 1 $x_{2,3}$: 10.89	$c_{3,3}$: 3	$c_{4,3}$: 2 $x_{4,3}$: 3.78	33.44
45		$c_{1,4}$: 2	$c_{2,4}$: 2	$c_{3,4}$: 1	$c_{4,4}$: 5	45

All of the times for the active processors are now equal. Since the time (45) for the next processor to be added is not less than t (33.44), the processor is not added and is left inactive and the load balancing is complete.

2.5 Procedural Description of the Algorithm and Program

The following section describes the implementation of the load balancing algorithm. The algorithm is implemented in source code file "loadbalancer.cpp".

1). BEGIN: The algorithm begins by first creating a data_arrays object. The creation of this object allocates memory for arrays to contain the initial delays of each processor, the size of each job, the capacity for each processor to do each job and the time for each processor to complete the jobs allocated to it.

2). The elements of the data_arrays object are then initializes with input data read from a text file.

3). The elements of the data_arrays object are the sorted in increasing order of the initial delay of each processor. At this point, there are no processors that are activated in the system. An activated processor is one that may be given load. Load is defined as part of a job.

4). The first 2 processors (those with the smallest initial delays) in the system are activated and their times are balanced via a knapsack algorithm.

5). Test: Is there a processor in the system which has not yet been activated and is the initial delay for this processor less than the time of the previously balanced processors?

If the result of this test is true, go to 6). If false go to 16).

6). Activate the next processor.

7). Test: Is the time of the newly activated processor less than the time of the previously balanced processors?

If the result of the test is true, go to 8). If false, go to 5).

8). Find a job for the newly activated processor to work on that creates a cycle.

9). Test: Was a job found for the newly activated processor that creates a cycle?

If result of test is true: go to 10). If result of test is false go to 15).

10). Build a redistribution graph that represents the redistribution cycle that will give load of the job found for the newly activated processor.

11). Given the above redistribution graph, build a system of linear equations and solve the system so that all of the activated processors will have equal time. Solve the system of linear equations to get the redistribution variables.

12). Test: If the redistribution variables are applied, will a load become negative?

If result of test is true, go to 13). If false go to 14).

13). Scale the solution so that when it is applied, the load that would have become negative, becomes instead exactly zero. This is a linear scaling and all redistribution variables are scaled by the same multiple. Then apply the scaled solution. Go to 7).

14). Apply the solution of redistribution variables. Go to 7).

15). A node_not_found or cycle_not_found exception was caught indicating that a redistribution cycle could not be found. Set the return value to indicate the type of exception. Go to 17).

16). Load balancing completed successfully. Set the return value to indicate success.

17). Create a text file and write the relevant elements contained in data_arrays to this file. This includes the load array, capacity array, initial delay array, and the time array.

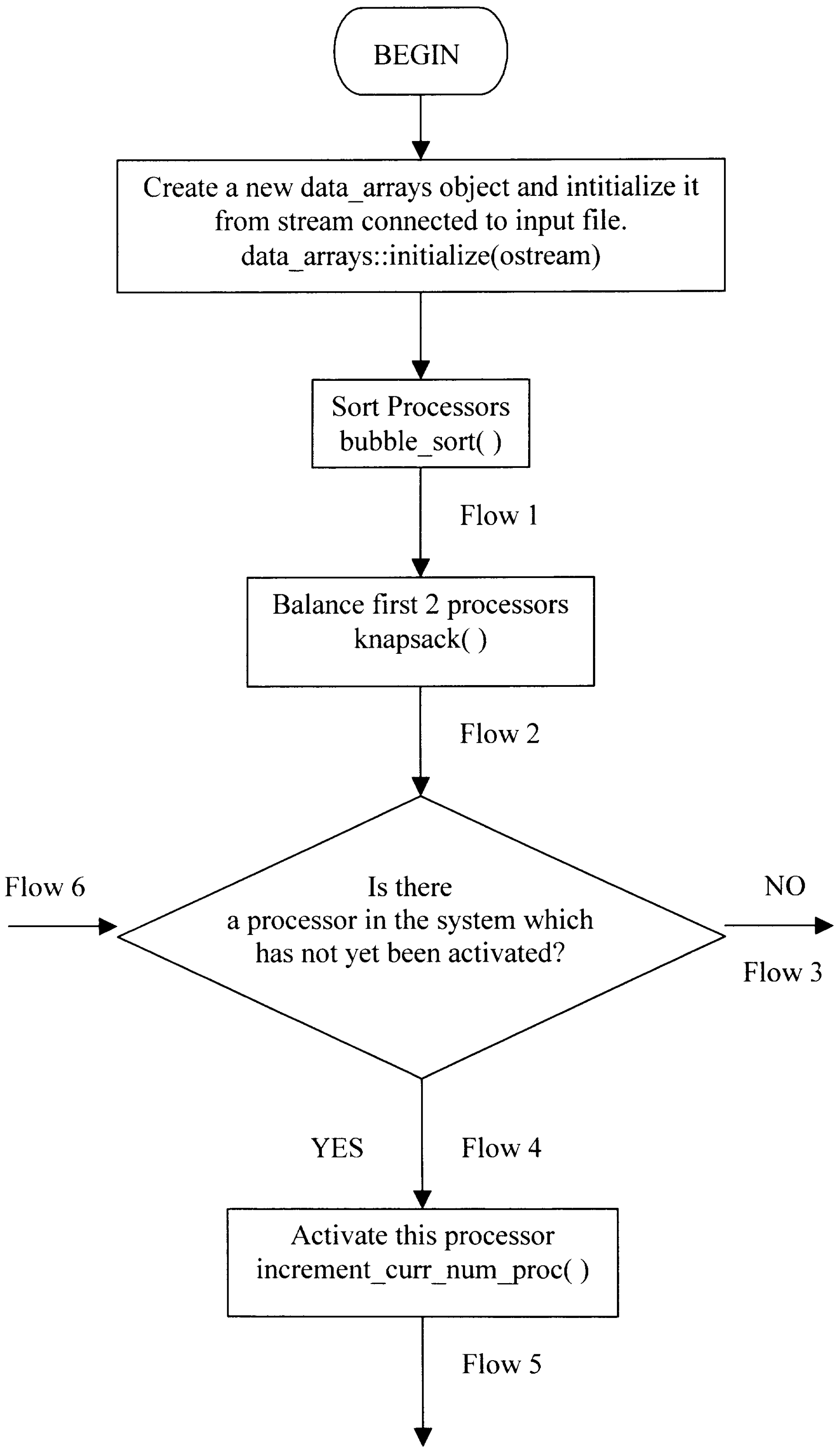
18). Return the return value indicated level of success.

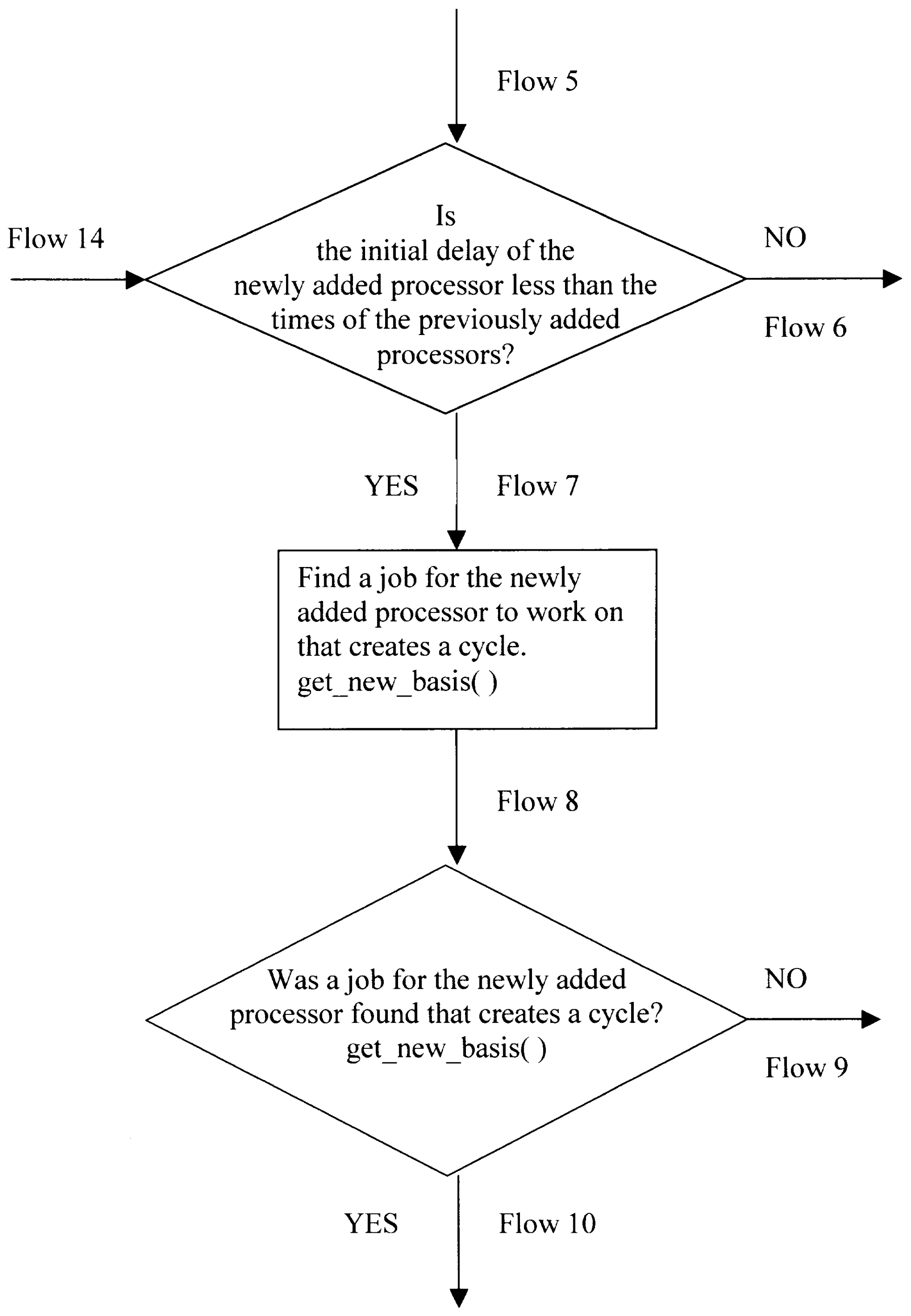
19). END

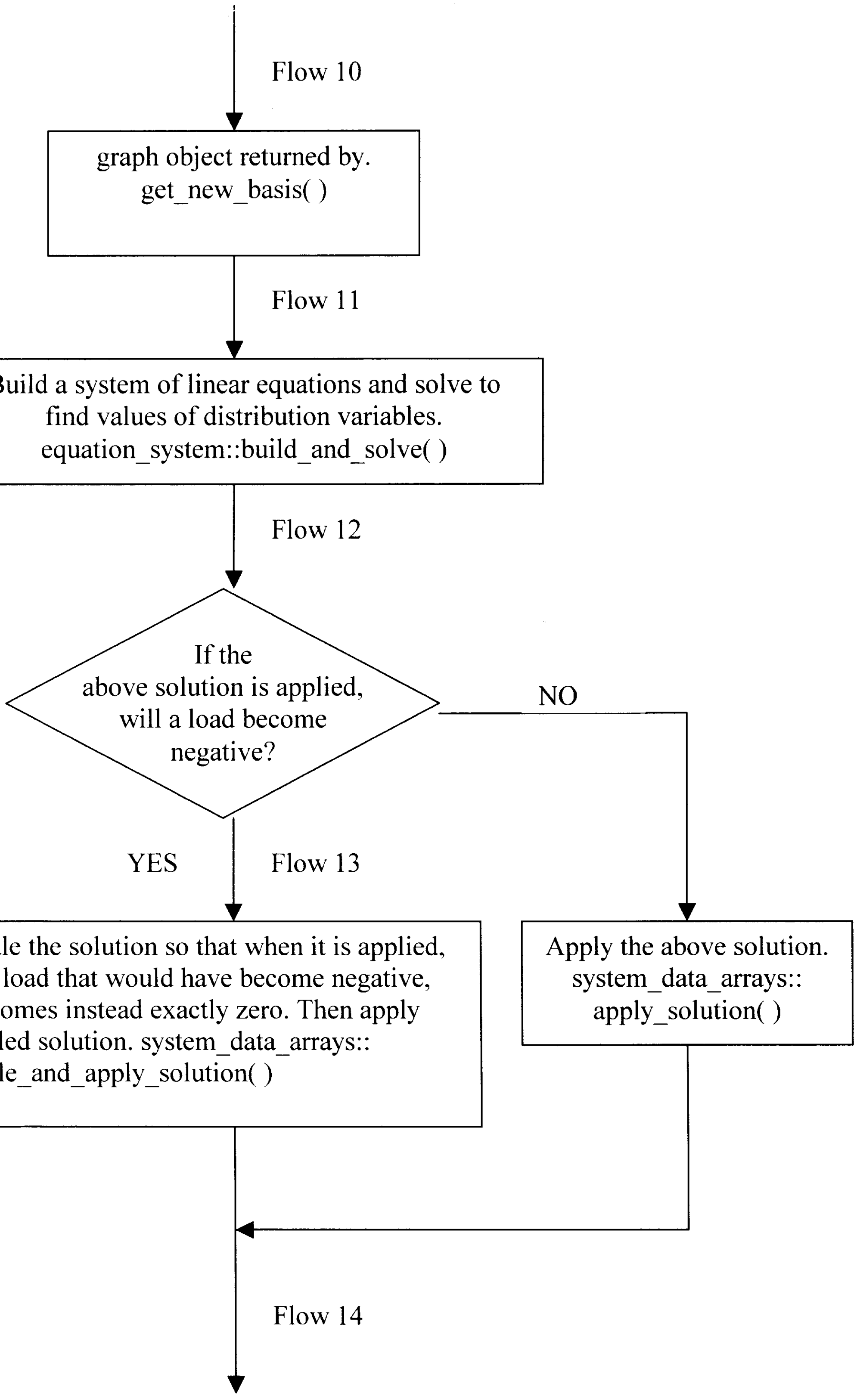
2.6 Flow Chart for the Load Balancing Program

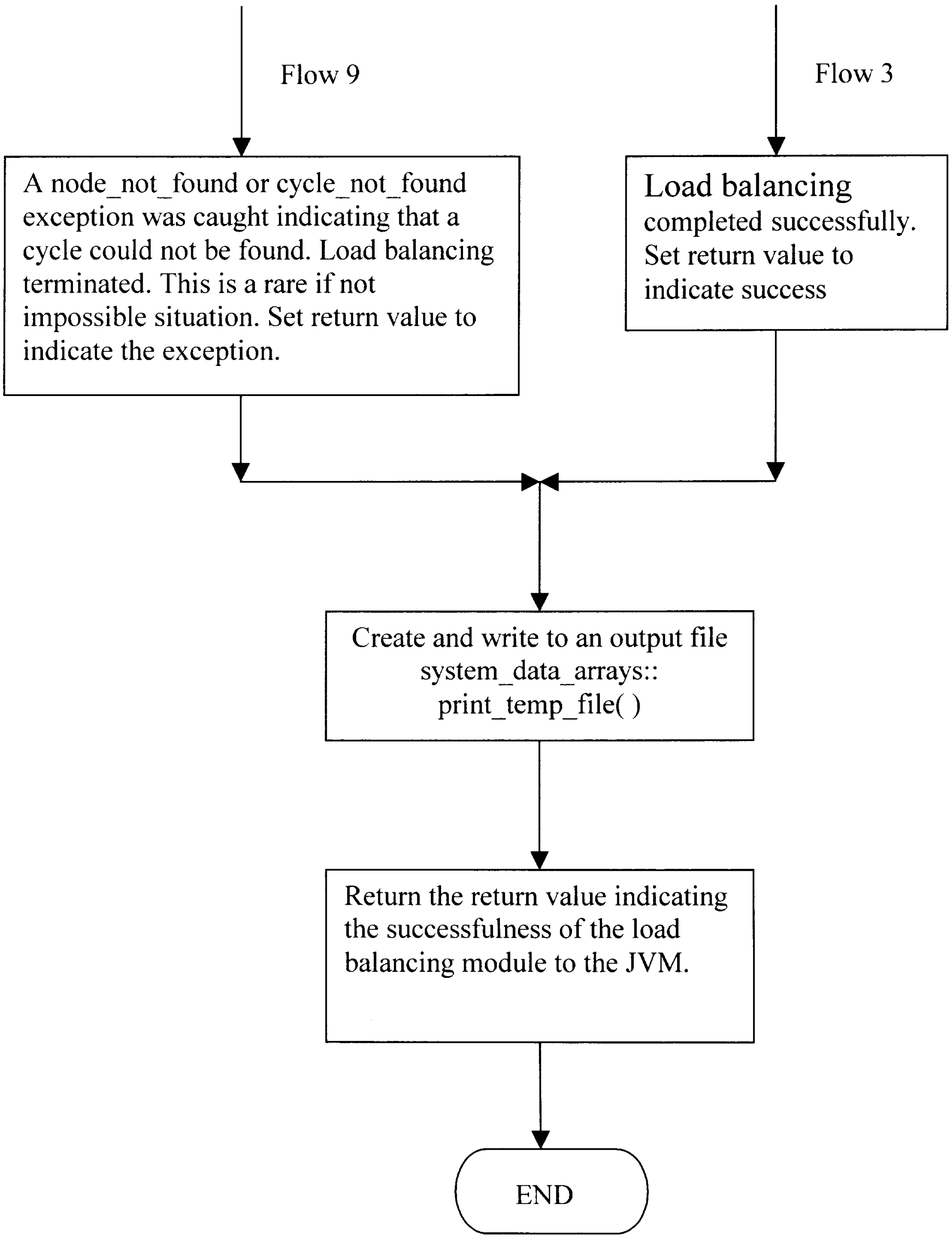
The flow of control of the load balancing algorithm is also shown in Flow Chart 1. This flowchart also shows the flow of control and function calls in "loadbalancer.cpp" which is at the highest level of source code in the load balancing program.

Flow Chart 1


BEGIN
Create a new data_arrays object and intitialize it from stream connected to input file. data_arrays::initialize(ostream)
Sort Processors bubble_sort( )
Flow 1
Balance first 2 processors knapsack( )
Flow 2
Flow 6
Is there a processor in the system which has not yet been activated?
NO
Flow 3
YES
Flow 4
Activate this processor increment_curr_num_proc( )
Flow 5


Flow 5
Is the initial delay of the newly added processor less than the times of the previously added processors?
Flow 14
NO
Flow 6
YES
Flow 7
Find a job for the newly added processor to work on that creates a cycle. get_new_basis( )
Flow 8
Was a job for the newly added processor found that creates a cycle? get_new_basis( )
NO
Flow 9
YES
Flow 10


Flow 10
graph object returned by.
get_new_basis( )
Flow 11
Build a system of linear equations and solve to find values of distribution variables.
equation_system::build_and_solve( )
Flow 12
If the above solution is applied, will a load become negative?
NO
YES
Flow 13
Scale the solution so that when it is applied, the load that would have become negative, becomes instead exactly zero. Then apply scaled solution. system_data_arrays:: scale_and_apply_solution( )
Apply the above solution. system_data_arrays:: apply_solution( )
Flow 14

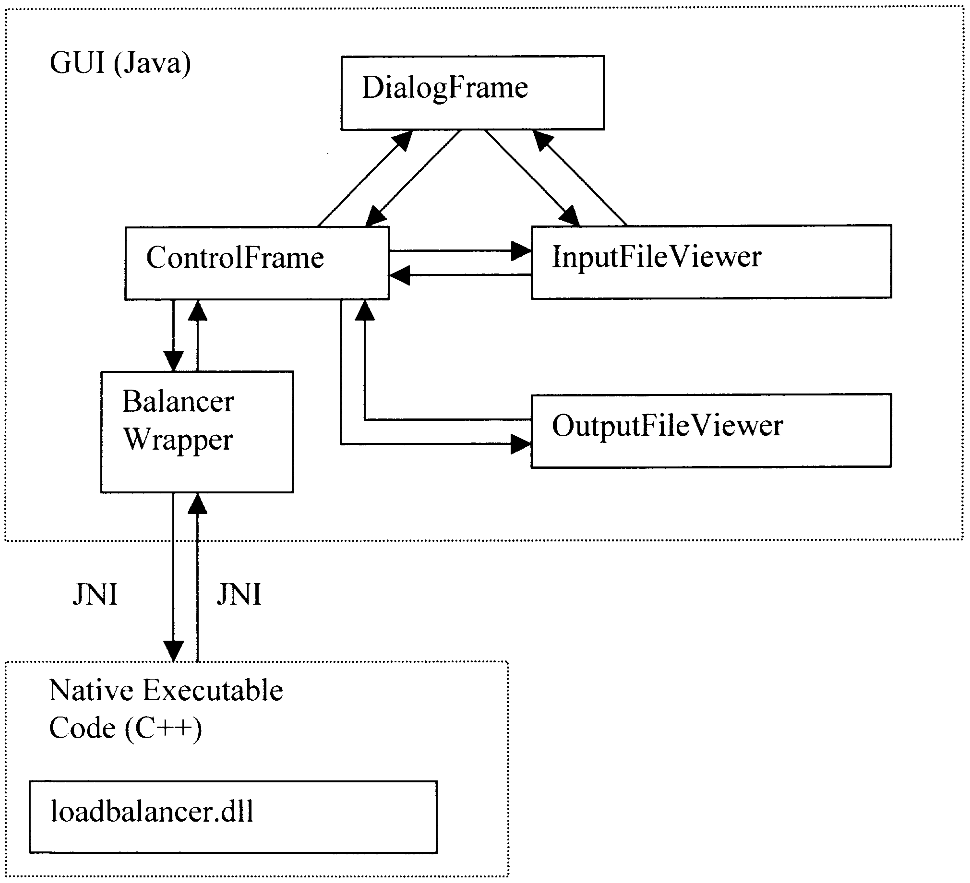

Flow 9
Flow 3
A node_not_found or cycle_not_found exception was caught indicating that a cycle could not be found. Load balancing terminated. This is a rare if not impossible situation. Set return value to indicate the exception.
Load balancing completed successfully. Set return value to indicate success
Create and write to an output file system_data_arrays:: print_temp_file( )
Return the return value indicating the successfulness of the load balancing module to the JVM.
END

CHAPTER 3

STRUCTURE OF THE SYSTEM

The system was designed in two parts, a graphical user interface (GUI) implemented in Java and the load balancing algorithm (loadbalancer.dll) implemented in C++. The GUI calls the native executable code via Java Native Interface (JNI). The structure of the system is shown in Diagram 1 below.

3.1 Diagram 1



3.2 Description of System Components

3.2.1 Graphical User Interface

ControlFrame: The ControlFrame is JFrame that controls the functioning of the entire system. It appears to the user when the user starts the system. It contains 6 buttons to provide the user with the following functionality:

1. "Create a New Data File": Selecting this button creates a DialogFrame that prompts the user for the number of processors and jobs.
2. "View and/or Edit a Data File": Selecting this button brings up JFileChooser that allows a user to select the file he wants to edit. The data in the file then appears in an InputFileViewer for editing.
3. "Run Balancer": Selecting this button brings up JFileChooser that allows a user to select the input file he wants to run the load balancing algorithm on. After balancing, the results appear in an OutputFileViewer.
4. "View an Output File": Selecting this button brings up JFileChooser that allows a user to select the output file he wants to edit. The data in the file then appears in an OutputFileViewer for viewing.
5. "Get Average Run Time": Selecting this button creates a DialogFrame that prompts the user for the number of processors and jobs. 6 input files are automatically generated using random numbers. The balancer is automatically run on these files. The run times of these 6 runs are reported as well as the average run time in a JFrame.
6. "Exit": Terminates the program.

The ControlFrame also contains a pull down menu for setting the preferred directories where input and output files are stored. These preferences are automatically saved to a text file "preferences.txt" so that the preferences are multi-session persistent. These preferences are automatically loaded whenever a ConrolFrame is created.

DialogFrame: The DialogFrame is a JFrame that contains JTextAreas allowing the user to enter the number of jobs and processors to create a new InputFileViewer or for getting the average run time. User entered values < 1 generate error messages and allow the user to correct the values.

InputFileViewer: This is a JFrame that contains a scrollable JTable that is color coded for initial delays, job size and capacities. The user can enter or edit values for the initial delays (backlogs), size of jobs and processor capacities. If the user fails to fill in a value or enters a value less than one, an error message appears and allows the user to correct his mistake. If the user does not want to enter numbers manually, he can select the "Random Numbers" button. This will fill in all table entries with random integers in the following ranges:

Job Size: 1 to 100

Initial Delays: 1 to 50

Processor Capacities: 1 to 10

The user can then select the "Save As", "Save" or "Close" buttons that follow the standard GUI conventions. The InputFileViewer also contains a pull down menu for setting the color preferences of the JTable. These preferences are automatically saved to a text file "preferences.txt" so that the preferences are multi-session persistent. These preferences are automatically loaded whenever an InputFileViewer is created.

OutputFileViewer: This is a JFrame that contains a scrollable JTable that is color coded for initial delays, job size, capacities, time-span and load distribution. It allows the user to view the results of running the balancer on an input file. The user can then select the "Don't Show Cap", "Save" or "Close" buttons. If the user selects the "Don't Show Cap" button, the capacities disappear making the load distribution even easier to read. The label on the button then changes to "Show Cap" and when the button is selected the capacities reappear. This can be repeated as often as desired. The "Save" and "Close" buttons follow the standard GUI conventions. If the OutputFileViewer was created automatically following the balancing of an input file, the OutputFileViewer suggests to the user to name the output file while saving as the name of the input file with the extension ".out" automatically appended. Of course, the user can provide his own name and/or extension but the default mechanism makes it extremely easy to relate output files with the input files that lead to their generation. The OutputFileViewer also contains a pull down menu for setting the color preferences of the JTable. These preferences are automatically saved to a text file "preferences.txt" so that the preferences are multi-session persistent. These preferences are automatically loaded whenever an OutputFileViewer is created.

BalancerWrapper: This wrapper class contains the signature of the native method which is the actual load balancer. Selecting the "Run Balancer" button of the ControlFrame creates a BalancerWrapper instance from which the native method can be called. The native method is called through Java Native Interface (JNI). This class is invisible to the user.

3.2.2 Native Code, loadbalancer.dll

The load balancing algorithm is implemented in C++ and compiled to a dynamically-linked library (dll) that is executable on machines running any version of the Microsoft Windows operating system. This dll contains only one function that matches the JNI required signature so that the GUI can access this function via JNI.

CHAPTER 4

EXPERIMENTS

Experiments were conducted to determine the run time (the amount of time the CPU is allocated while executing "loadbalancer.dll") as a function of the number of jobs and processors. 30 experiments (runs) were conducted on each combination of 4 to 10 processors and 4 to 10 jobs. In the case of 4 processors and 4 jobs, the input data was created by selecting the "Random Numbers" button on the InputFileViewer screen. The results of these experiments are shown in Section 4.1.1. For all other combinations of jobs and processors, the average run time was obtained by selecting the "Get Average Time" button on the ControlFrame. This was repeated 5 times for each combination since each selection of "Get Average Time" conducts 6 experiments. The average run time for each set of 30 experiments was then calculated and entered in Table 14 in Section 4.1.2. In all cases, the initial delays range from 1 to 50, the job sizes range from 1 to 100 and the capacities range from 1 to 10. All input data was automatically generated randomly by the program. All experiments were conducted on a Dell Inspiron 8000 laptop computer with Intel Pentium 3 CPU running at 850 MHz with 320 MB RAM and a 100 MHz system bus. The operating system was Microsoft Windows Me.

4.1 Complete Listing, 4 Processors and 4 Jobs

In these tables, "Job Size"=a_j, "Delays"= b_k, "Time"=t_k, "L"=x_{jk} and "C"=c_{jk} using the notations described in Section 2.1.

Run 1:

Job Size	92	56	74	89

Delay	Capacities (C) and Loads (L)				Time
27	C: 7	C:5	C: 4 L: 26.88	C: 8	134.50
21	C: 4 L: 14.38	C: 1 L: 56	C: 10	C: 5	134.50
24	C: 1 L: 77.63	C: 1	C: 1 L: 32.88	C: 8	134.50
17	C: 7	C: 2	C: 2 L: 14.25	C: 1 L: 89	134.50

Balancing time: 0.148 milliseconds

Run 2:

Job Size	31	11	54	99

Delay	Capacities (C) and Loads (L)				Time
16	C: 4	C: 3 L: 11	C: 3	C: 6 L: 22.89	186.34
47	C: 7 L: 1.53	C: 10	C: 2 L: 54	C: 7 L: 2.95	186.34
40	C: 3 L: 29.47	C: 3	C: 7	C: 2 L: 73.17	186.34
39	C: 5	C: 7	C: 6	C: 8	186.34

Balancing time: 0.214 milliseconds

Run 3:

Job Size	37	29	36	54

Delay	Capacities (C) and Loads (L)				Time
40	C: 1 L: 37	C: 9	C: 6	C: 2 L: 50.27	177.53
3	C: 9	C: 9 L: 19.39	C: 8	C: 9	177.53
46	C: 9	C: 8 L: 9.61	C: 5 L: 8.69	C: 3 L: 3.74	177.53
41	C: 2	C: 10	C: 5 L: 27.31	C: 3	177.53

Balancing time: 0.247 milliseconds

Run 4:

Job Size	92	6	13	89

Delay	Capacities (C) and Loads (L)				Time
17	C: 6	C: 3	C: 5 L: 13	C: 2 L: 17.70	117.39
20	C: 10 L: 2.61	C: 7	C: 7	C: 1 L: 71.30	117.39
16	C: 2 L: 50.70	C: 10	C: 6	C: 8	117.39
28	C: 2 L: 38.70	C: 2 L: 6	C: 4	C: 6	117.39

Balancing time: 0.242 milliseconds

Run 5:

Job Size	18	8	17	27

Delay	Capacities (C) and Loads (L)				Time
47	C: 8	C: 3	C: 8 L: 7.03	C: 6	103.21
25	C: 6	C: 3	C: 5 L: 4.95	C: 5 L: 10.70	103.21
45	C: 1 L: 18	C: 6	C: 8 L: 5.03	C: 7	103.21
30	C: 2	C: 1 L: 8	C: 10	C: 4 L: 16.30	103.21

Balancing time: 0.171 milliseconds

Run 6:

Job Size	25	87	26	45

Delay	Capacities (C) and Loads (L)				Time
17	C: 8 L: 13.64	C: 10	C: 8	C: 8	126.11
11	C: 4	C: 3 L: 23.37	C: 6	C: 1 L: 45	126.11
39	C: 7 L: 11.36	C: 5 L: 1.52	C: 7	C: 4	126.11
38	C: 5	C: 1 L: 62.11	C: 1 L: 26	C: 3	126.11

Balancing time: 0.258 milliseconds

Run 7:

Job Size	21	19	28	8

Delay
24
24
41
37

Capacities (C) and Loads (L)			
C: 9	C: 3	C: 6 L: 5.55	C: 5 L: 5.97
C: 2 L: 21	C: 9	C: 3 L: 7.06	C: 6
C: 7	C: 5	C: 3 L: 15.39	C: 8
C: 7	C: 2 L: 19	C: 6	C: 6 L: 2.03

Balancing time: 0.175 milliseconds

Time
87.17
87.17
87.17
87.17

Run 8:

Job Size	50	63	17	33

Delay
26
6
38
36

Capacities (C) and Loads (L)			
C: 1 L: 50	C: 6 L: 14.96	C: 9	C: 9
C: 7	C: 4 L: 27.20	C: 3 L: 17	C: 6
C: 9	C: 6 L: 20.84	C: 8	C: 5 L: 0.55
C: 8	C: 9	C: 7	C: 4 L: 32.45

Balancing time: 0.225 milliseconds

Time
165.79
165.79
165.79
165.79

Run 9:

Job Size	70	96	79	38

Delay
24
31
5
6

Capacities (C) and Loads (L)			
C: 5	C: 2 L: 26.41	C: 2 L: 63.02	C: 7
C: 3	C: 5 L: 3.97	C: 10	C: 4 L: 38
C: 1 L: 70	C: 6	C: 8 L: 15.98	C: 3
C: 4	C: 3 L: 65.62	C: 10	C: 6

Balancing time: 0.231 milliseconds

Time
202.856
202.856
202.856
202.856

Run 10:

Job Size	51	8	92	27

Delay	Capacities (C) and Loads (L)				Time
25	C: 3	C: 7	C: 3 L: 63.56	C: 4	215.68
9	C: 7 L: 21.82	C: 8	C: 10	C: 2 L: 27	215.68
17	C: 6	C: 6	C: 7 L: 28.38	C: 8	215.68
32	C: 6 L: 29.19	C: 1 L: 8	C: 10 L: 0.05	C: 9	215.68

Balancing time: 0.274 milliseconds

Run 11:

Job Size	29	81	48	65

Delay	Capacities (C) and Loads (L)				Time
43	C: 9	C: 3	C: 7	C: 4 L: 48.46	236.83
17	C: 1 L: 29	C: 8	C: 3 L: 48	C: 9 L: 5.20	236.83
1	C: 6	C: 5 L: 31.29	C: 5	C: 7 L: 11.34	236.83
38	C: 5	C: 4 L: 49.71	C: 8	C: 10	236.83

Balancing time: 0.154 milliseconds

Run 12:

Job Size	70	23	60	39

Delay	Capacities (C) and Loads (L)				Time
4	C: 6 L: 7.94	C: 5	C: 1 L: 60	C: 3	111.63
4	C: 7	C: 10	C: 8	C: 3 L: 35.88	111.63
50	C: 1 L: 49.14	C: 4	C: 2	C: 4 L: 3.12	111.63
1	C: 5 L: 12.93	C: 2 L: 23	C: 9	C: 10	111.63

Balancing time: 0.242 milliseconds

Run 13:

Job Size	7	53	6	55

Delay	Capacities (C) and Loads (L)				Time
40	C: 4	C: 5 L: 10.21	C: 9	C: 9 L: 3.83	125.57
33	C: 1 L: 7	C: 2 L: 42.79	C: 7	C: 10	125.57
39	C: 7	C: 7	C: 7	C: 4 L: 21.64	125.57
25	C: 6	C: 4	C: 2 L: 6	C: 3 L: 29.52	125.57

Balancing time: 0.242 milliseconds

Run 14:

Job Size	41	46	68	57

Delay	Capacities (C) and Loads (L)				Time
16	C: 7	C: 5 L: 28.96	C: 10	C: 10	160.79
2	C: 2 L: 13.10	C: 9 L: 14.73	C: 9	C: 2	160.79
4	C: 7	C: 9 L: 2.30	C: 2 L: 68	C: 7	160.79
48	C: 2 L: 27.90	C: 10	C: 7	C: 1 L: 57	160.79

Balancing time: 0.258 milliseconds

Run 15:

Job Size	5	28	49	27

Delay	Capacities (C) and Loads (L)				Time
8	C: 2 L: 3.61	C: 1 L: 28	C: 5	C: 1 L: 27	70.22
11	C: 7	C: 4	C: 2 L: 29.61	C: 2	70.22
40	C: 5 L: 1.39	C: 7	C: 5 L: 4.65	C: 5	70.22
26	C: 6	C: 3	C: 3 L: 14.74	C: 7	70.22

Balancing time: 0.203 milliseconds

Run 16:

Job Size	93	23	7	60

Delay	Capacities (C) and Loads (L)				Time
13	C: 6 L: 14.10	C: 5 L: 23	C: 2	C: 9	212.58
19	C: 5 L: 38.72	C: 7	C: 7	C: 7	212.58
17	C: 5 L: 15.12	C: 8	C: 3	C: 2 L: 60	212.58
5	C: 8 L: 25.07	C: 8	C: 1 L: 7	C: 4	212.58

Balancing time: 0.237 milliseconds

Run 17:

Job Size	36	19	46	26

Delay	Capacities (C) and Loads (L)				Time
7	C: 9	C: 9	C: 8 L: 11.56	C: 1 L: 26	125.68
27	C: 3	C: 10	C: 8 L: 12.34	C: 2	125.68
10	C: 1 L: 36	C: 1 L: 19	C: 6 L: 10.11	C: 1	125.68
18	C: 4	C: 7	C: 9 L: 11.96	C: 5	125.68

Balancing time: 0.154 milliseconds

Run 18:

Job Size	26	27	89	65

Delay	Capacities (C) and Loads (L)				Time
5	C: 8	C: 8	C: 3 L: 56.15	C: 7	173.45
10	C: 9	C: 2 L: 16.72	C: 8	C: 2 L: 65	173.45
37	C: 9 L: 9.45	C: 5 L: 10.28	C: 7	C: 10	173.45
25	C: 5 L: 16.55	C: 3	C: 2 L: 32.85	C: 9	173.45

Balancing time: 0.275 milliseconds

Run 19:

Job Size	45	36	99	12

Delay	Capacities (C) and Loads (L)				Time
28	C: 8 L: 19.47	C: 4	C: 4	C: 3	183.76
14	C: 8	C: 3 L: 23.59	C: 1 L: 99	C: 3	183.76
29	C: 5 L: 1.57	C: 7 L: 12.41	C: 9	C: 5 L: 12	183.76
40	C: 6 L: 23.96	C: 7	C: 7	C: 5	183.76

Balancing time: 0.253 milliseconds

Run 20:

Job Size	67	15	7	68

Delay	Capacities (C) and Loads (L)				Time
35	C: 2	C: 8	C: 6	C: 9 L: 16.49	183.40
1	C: 7	C: 3 L: 15	C: 3 L: 7	C: 9 L: 12.93	183.40
37	C: 3	C: 3	C: 2	C: 5 L: 29.28	183.40
42	C: 1 L: 67	C: 6	C: 5	C: 8 L: 9.30	183.40

Balancing time: 0.214 milliseconds

Run 21:

Job Size	46	22	89	85

Delay	Capacities (C) and Loads (L)				Time
48	C: 4	C: 9 L: 3.52	C: 6	C: 2 L: 36.88	153.44
16	C: 4	C: 10	C: 6	C: 3 L: 45.81	153.44
9	C: 8	C: 3 L: 18.48	C: 1 L: 89	C: 9	153.44
43	C: 2 L: 46	C: 8	C: 5	C: 8 L: 2.31	153.44

Balancing time: 0.209 milliseconds

Run 22:

Job Size	18	50	31	66

Delay	Capacities (C) and Loads (L)				Time
42	C:	C: L: 22.62	C:	C:	200.31
15	C:	C:	C:	C: L: 46.33	200.31
19	C:	C: L: 1.92	C: L: 31	C: L: 19.67	200.31
1	C: L: 18	C: L: 25.46	C:	C:	200.31

Balancing time: 0.192 milliseconds

Run 23:

Job Size	38	81	85	61

Delay	Capacities (C) and Loads (L)				Time
43	C: 7 L: 3.19	C: 9	C: 3 L: 51.69	C: 8	220.42
35	C: 9	C: 2	C: 10 L: 0.24	C:3 L: 61	220.42
22	C: 3	C: 10	C: 6 L: 33.07	C: 2	220.42
35	C: 3 L: 34.81	C: 1 L: 81	C: 9	C: 8	220.42

Balancing time: 0.28 milliseconds

Run 24:

Job Size	64	36	82	26

Delay	Capacities (C) and Loads (L)				Time
37	C: 3	C: 4 L: 7.02	C: 6	C: 4 L: 26	169.08
42	C: 10	C: 5 L: 9.02	C: 1 L: 82	C: 7	169.08
15	C: 3 L: 51.36	C: 8	C: 2	C: 8	169.08
48	C: 8 L: 12.64	C: 1 L: 19.96	C: 1	C: 9	169.08

Balancing time: 0.226 milliseconds

Run 25:

Job Size	48	74	18	59

Delay	Capacities (C) and Loads (L)				Time
40	C: 3 L: 38	C: 4	C: 9	C: 5	154
16	C: 7	C: 6 L: 17	C: 2 L: 18	C: 5	154
49	C: 6 L: 10.00	C: 5 L: 9	C: 8	C: 9	154
47	C: 4	C: 1 L: 48	C: 5	C: 1 L: 59	154

Balancing time: 0.247 milliseconds

Run 26:

Job Size	38	66	78	62

Delay	Capacities (C) and Loads (L)				Time
10	C: 9	C: 3	C: 5 L: 46.95	C: 6	244.74
19	C: 7 L: 32.25	C: 9	C: 6	C: 6	244.74
38	C: 7 L: 5.75	C: 4	C: 1 L: 31.05	C: 5 L: 27.09	244.74
8	C: 10	C: 2 L: 66	C: 9	C: 3 L: 34.91	244.74

Balancing time: 0.269 milliseconds

Run 27:

Job Size	11	15	13	76

Delay	Capacities (C) and Loads (L)				Time
13	C: 2 L: 4.13	C: 1 L: 15	C: 3 L: 13	C: 4	75.27
20	C: 9	C: 1	C: 9	C: 9 L: 6.14	75.27
12	C: 7	C: 2	C: 10	C: 1 L: 63.27	75.27
22	C: 2 L: 6.87	C: 9	C: 3	C: 6 L: 6.59	75.27

Balancing time: 0.253 milliseconds

Run 28:

Job Size	87	44	66	81

Delay	Capacities (C) and Loads (L)				Time
4	C: 8 L: 19.33	C: 4 L: 44	C: 2	C: 6	334.63
27	C: 10 L: 30.76	C: 9	C: 9	C: 5	334.63
48	C: 8 L: 5.45	C: 6	C: 2	C: 3 L: 81	334.63
17	C: 8 L: 31.45	C: 9	C: 1 L: 66	C: 8	334.63

Balancing time: 0.215 milliseconds

Run 29:

Job Size	83	11	9	5

Delay	Capacities (C) and Loads (L)				Time
11	C: 9 L: 12.17	C: 7	C: 1	C: 4 L: 5	140.49
37	C: 7 L: 11.64	C: 2 L: 11	C: 5	C: 5	140.49
48	C: 2 L: 46.24	C: 9	C: 7	C: 6	140.49
2	C: 10 L: 12.95	C: 3	C: 1 L: 9	C: 8	140.49

Balancing time: 0.182 milliseconds

Run 30:

Job Size	47	95	11	25

Delay	Capacities (C) and Loads (L)				Time
15	C: 9	C: 6 L: 21.20	C: 9	C: 4	142.19
42	C: 8 L: 1.92	C: 1 L: 73.80	C: 1 L: 11	C: 3	142.19
42	C: 6 L: 12.53	C: 9	C: 3	C: 1 L: 25	142.19
12	C: 4 L: 32.55	C: 6	C: 8	C: 3	142.19

Balancing time: 0.302 milliseconds

Average time = 6.792/30= 0.2264 milliseconds

4.2 Tabulated Data for all Experiments

Table 14: Table entries have units of milliseconds/run.

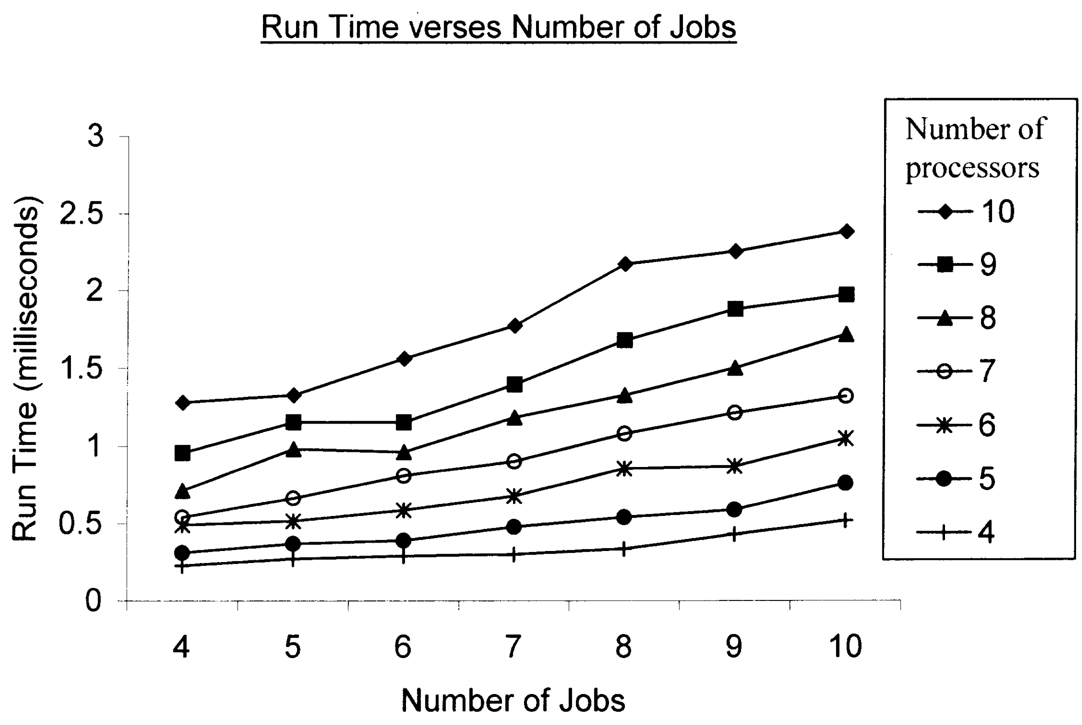
		Number of Jobs						
		4	5	6	7	8	9	10
Number of Processors	4	0.226	0.271	0.289	0.300	0.335	0.429	0.517
	5	0.311	0.367	0.390	0.478	0.541	0.587	0.760
	6	0.490	0.515	0.586	0.677	0.855	0.871	1.048
	7	0.541	0.663	0.809	0.901	1.079	1.212	1.319
	8	0.713	0.981	0.962	1.183	1.327	1.502	1.717
	9	0.956	1.152	1.151	1.397	1.680	1.881	1.971
	10	1.278	1.327	1.562	1.773	2.175	2.255	2.383

CHAPTER 5

RESULTS OF EXPERIMENTS

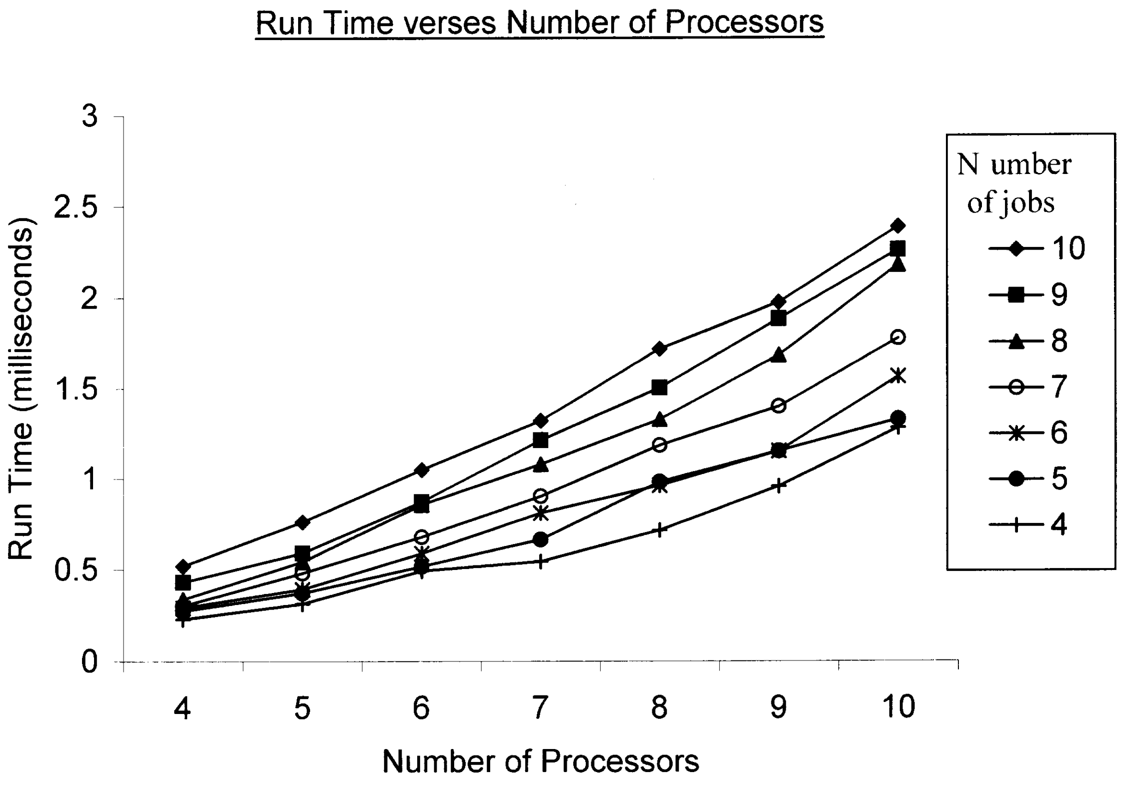
5.1 Run Time as Function of the Number of Jobs

Graph 1



5.2 Run Time as a Function of the Number of Processors

Graph 2



CHAPTER 6

CONCLUSIONS

Graph 1 shows that in general, the run time increases with the number of jobs. This is expected since many of the functions in the implementation require traversing through entire arrays containing the capacities and loads assigned to each processor. The size of these arrays, and therefore the time to traverse them, is proportional to the number of jobs. The slope of Graph 1 and is not very smooth. This suggests that the run time data points are greatly influenced by the input data that was generated randomly for each experiment. This is evident by the fact that the balancing time for the experiment with 8 processors and 6 jobs was actually less than that for the experiment with 8 processors and 5 jobs. The erratic slope of this graph makes it difficult to estimate the general complexity with respect to the number of jobs.

Graph 2 shows that in general, the run time increases with the number of processors. This is expected since many of the functions in the implementation require traversing through entire arrays of sizes that are proportional to the number of processors. It appears that in general the slopes of the graph of run time as a function of the number of processors are increasing. Since the times are not increasing drastically, it is hopeful that this suggests that T(m) has polynomial complexity.

APPENDIX A

C++ SOURCE CODE

This Appendix contains the source code and header files required to create the dynamically linked library (dll) "loadbalancer.dll". The exact procedure in Appendix C can be followed to compile the dll.

A.1 BalancerWrapper.h

```
// BalancerWrapper.h

/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class BalancerWrapper */

#ifndef _Included_BalancerWrapper
#define _Included_BalancerWrapper
#ifdef __cplusplus
extern "C" {
#endif
#undef BalancerWrapper_NO_ERRORS
#define BalancerWrapper_NO_ERRORS 0L
#undef BalancerWrapper_FILE_NOT_FOUND
#define BalancerWrapper_FILE_NOT_FOUND 1L
#undef BalancerWrapper_CYCLE_NOT_FOUND
#define BalancerWrapper_CYCLE_NOT_FOUND 2L
#undef BalancerWrapper_NODE_NOT_FOUND
#define BalancerWrapper_NODE_NOT_FOUND 3L
/*
 * Class:     BalancerWrapper
 * Method:    runBalancer
 * Signature: (Ljava/lang/String;)I
 */
JNIEXPORT jint JNICALL Java_gui_BalancerWrapper_runBalancer
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
#endif
#endif
```

A.2 stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#if
!defined(AFX_STDAFX_H__01EA0F80_BFFC_4CB0_9046_7476C0879FE8__INCLUDED_)
#define AFX_STDAFX_H__01EA0F80_BFFC_4CB0_9046_7476C0879FE8__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


#define THRESH .0001
// TODO: reference additional headers your program requires here
#include <stddef.h>
#include <stdlib.h>
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <math.h>
#include <iomanip.h>
#include <time.h>
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately
// before the previous line.

#endif //
!defined(AFX_STDAFX_H__01EA0F80_BFFC_4CB0_9046_7476C0879FE8__INCLUDED_)
```

A.3 data_arrays.h

```
// data_arrays.h

#ifndef _DATA_ARRAYS
#define _DATA_ARRAYS

#include "../stdafx.h"
#include "from_to_node.h"

class data_arrays
{
public:

    // Constructor
    data_arrays(int rows, int cols);

    // Copy constructor
    data_arrays(data_arrays &m);

    // Destructor
    ~data_arrays();

    // Accessor functions
    int get_rows();
    int get_columns();
    int get_curr_num_proc();
    float** get_cap_array();
    float** get_load_array();
    float* get_time_array();
    float* get_delay_array();

    void initialize(ifstream &fin);
    void bubble_sort();
    void knapsack();
    void* get_new_basis();
    int check_solution(from_to_node* from_to_array, float* x);
    void apply_solution(from_to_node* from_to_array,  float* x);
    void scale_and_apply_solution(from_to_node* from_to_array,
        float* x);
    void increment_curr_num_proc();
    void print_temp_file(int number_of_lin_solve,
                         double millisec_per_iteration);


private:
    int rows;
    int columns;
    int curr_num_proc;

    float** cap_array;
    float** load_array;

    float* job_array;
    float* delay_array;
    float* time_array;

    int* original_order;

    void parse_row(int rw, int col);
    void parse_col(int rw, int col);

    float calc_av_cap(int row, int col);
    float calc_cap_ratio(float cap, float av_cap);

    int get_index_most_neg(from_to_node* from_to_array, float* x);
};

#endif
```

A.4 eq_system.h

```
// eq_system.h

// This class builds and solves a system of linear equations.
// Gaussian Elimination functions are dervived from those that appear
// in D.R. Brooks, C Programming: The Essentials for Engineers and
// Scientists, pp. 387-390, Springer-Verlag, 1999.

#ifndef _EQ_SYSTEM
#define _EQ_SYSTEM

#include "../stdafx.h"
#include "data_arrays.h"
#include "graph.h"

class eq_system
{
public:
    eq_system(data_arrays* m);
    void build_and_solve(graph* g);
    float* get_x();
    ~eq_system();


private:
    data_arrays* mat;        // pointer to data_arrays
    int num_rows;            // = number of variables
    int num_cols;            // = number of variables +1 for b column
    float** A;               // b is last column of A
    float* x;                // x contains the solutions
    int row;
    int col;

    int fill_data_arrays(graph* g);
    int triangularize();
    int backsolve();
};

#endif
```

A.5 exceptions.h

```
// exceptions.h

#ifndef _EXCEPTION
#define _EXCEPTION

class Exception
{
};

class FileNotFoundException : public Exception
{
};

class CycleNotFoundException : public Exception
{
};

class NodeNotFoundException : public Exception
```

```
{
};

#endif
```

A.6 from_to_node.h

```
// from_to_node.h

#ifndef _FROM_TO_NODE
#define _FROM_TO_NODE

#include "../stdafx.h"


struct from_to_node
{
    // these are rows to be used in building the eq_system.
    // The first row in the system has an index of 0
    int col;
    int from_row;   // The row in the data_arrays that contains from
    int to_row;     // The row in the data_arrays that contains to

};

#endif
```

A.7 graph.h

```
// graph.h

#ifndef _GRAPH
#define _GRAPH

#include "../stdafx.h"
#include "data_arrays.h"
#include "from_to_node.h"

class graph
{
public:
    graph(data_arrays* m);
    int build_graph(int row, int col);
    bool cycle_found();
    from_to_node* get_from_to_array();
    ~graph();

private:
    data_arrays* mat;
    int cur_index;
    int num_vars;
    int vars_allocated;
    from_to_node* from_to_array;
    bool assignment_OK(int new_from_row, int new_to_row);
    bool rows_not_yet_connected(int new_from_row, int new_to_row);
    int parse_column(int row, int col);
    int parse_row(int row, int col);

};

#endif
```

A.8 loadbalancer.cpp

```
// loadbalancer.cpp

#include <jni.h>
#include "stdafx.h"
#include "headers/data_arrays.h"
#include "headers/exceptions.h"
#include "headers/eq_system.h"
#include "headers/graph.h"
#include "BalancerWrapper.h"

// To get a more accurate reading of the run time of the program, the
// load balancing algorithm is repeated NUM_ITERATIONS. The time for
// each iteration is calculated by dividing the time for all the
// iterations by NUM_ITERATIONS.

#define NUM_ITERATIONS 500

// The function called by the Java program. Jpath is the path of the
// input data file selected by the user. JNIEXPORT jint JNICALL
// Java_gui_BalancerWrapper_runBalancer(JNIEnv * env, jobject obj,
// jstring jpath)
{
    jint returnValue= 0;
    jboolean isCopy;

    // Convert jstring object "jpath" to char[]. Returns a pointer
    // to a NULL-terminated sequence of ASCII bytes. If successfull,
    // iscopy will be set to JNI_TRUE.
    const char* path= env->GetStringUTFChars(jpath, &isCopy);

    try
    {
        // For keeping track of the amount of CPU time that the
        // program requires to complete execution.
        clock_t start_clock, end_clock;

        // Open stream to input data file
        ifstream fin(path);
        if(!fin)
        {
            throw new FileNotFoundException();
        }

        char buf[10];
```

```
fin >> buf; // remove "data" from stream
fin >> buf; // remove "file" from stream

int num_proc, num_jobs;
fin >> num_proc;    // The number of processors in the system
fin >> num_jobs;    // The number of jobs in the system

// Create a system_data_arrays. Every iteration will use a new
// COPY of this object.
data_arrays* mm= new data_arrays(num_proc, num_jobs);

// Initialize data members from input stream
mm->initialize(fin);

// Close the stream
fin.close();

// Start keeping track of the run time of the program
start_clock= clock();

// To get a more accurated reading of the run time,
// repeat balancing algorithm for NUM_ITERATIONS.
for(unsigned long x=0; x<NUM_ITERATIONS; x++)
{
    // Make a new copy of the original system_data_arrays
    data_arrays *m= new data_arrays(*mm);

    // Sort the rows of the system_data_arrays in order of
    // increasing initial delay of processors.
    m->bubble_sort();

    // A counter for the number of linear equation
    // systems that must be solved for each iteration.
    int number_of_lin_solve= 0;

    // Distribute load to the first 2 processors.
    m->knapsack();

    try
    {
        // Repeat while there is a processor that is not yet
        // been allocated any load by the system.
        while(m->get_curr_num_proc()<m->get_rows())
        {
            // Enter that processor into the system so that it
            // can be given load.
            m->increment_curr_num_proc();

            // Repeat while the time of the newly added
            // processor is less than the previously added
            // processors.
            while(((m->get_time_array()[0]) -
            (m->get_time_array()[(m->get_curr_num_proc()-1)]))
            > THRESH)
            {
                // Find a distribution cycle. If one is found,
                // return a graph that represents the cycle.
                // Else, throws a CycleNotFoundException or
                // a NodeNotFoundException.
                graph*g= (graph*)m->get_new_basis();

                // Use the graph of the distribution cycle to
                // build a system of linear equations. Solve
                // the system by Gaussian Elimination.
                eq_system* es= new eq_system(m);
                es->build_and_solve(g);

                number_of_lin_solve++;

                // Check the solution of the system of linear
                // equations so that if applied, no load will
                // become negative.
                if( (m->check_solution(g->get_from_to_array(),
                    es->get_x())) < 0)
                {
                    // No load will go negative so apply the
                    // solution
                    m->apply_solution(g->get_from_to_array(),
                        es->get_x());
                }
                else
                {
                    // A load will go negative, therefor scale
                    // the solution so that when applied, one
                    // of the loads will become exactly zero.
                    // Now no loads will become negative.
                    m->scale_and_apply_solution(
                        g->get_from_to_array(), es->get_x());
                }
                delete es;
                delete g;
            } // end while
        } // end while
    } // end try
    catch(CycleNotFoundException* cnfe)
    {
        returnValue = 2;
        delete cnfe;
    }
    catch(NodeNotFoundException* nnfe)
    {
        returnValue = 3;
        delete nnfe;
    }

    if(x < (NUM_ITERATIONS -1))
    {
        delete m;
    }
    else
    {
        end_clock= clock();
        double clock_diff= (double)(end_clock - start_clock);
        double millisec_per_iteration=
                    (clock_diff/CLOCKS_PER_SEC) * 1000/NUM_ITERATIONS;

                // Create file "temp.txt" to be read and displayed
                // by Java GUI.
                m->print_temp_file(number_of_lin_solve,
                                   millisec_per_iteration);
                delete m;
            }
        } // end for

        delete mm;
    }// end try
    catch(FileNotFoundException* fnfe)
    {
        returnValue = 1;
        delete fnfe;
    }

    if(isCopy == JNI_TRUE)
    {
        // Inform the JVM to release the memory required by jpath.
        env->ReleaseStringUTFChars(jpath, path);
    }

    return returnValue;
}
```

A.9 data_arrays.cpp

```
// data_arrays.cpp

#include "../stdafx.h"
#include "../headers/data_arrays.h"
#include "../headers/exceptions.h"
#include "../headers/graph.h"


data_arrays::data_arrays(int rws, int cols)
{
    rows = rws;
    columns = cols;
    curr_num_proc=0;

    job_array= new float[columns];
    delay_array= new float[rows];
    time_array= new float[rows];
    original_order= new int[rows];

    cap_array= new float*[rows];
    load_array= new float*[rows];

    int i;
    for(i=0; i<rows; i++)
    {
        cap_array[i]= new float[columns];
```

```
            load_array[i]= new float[columns];
            original_order[i]= i;
        }
}


data_arrays::data_arrays(data_arrays &m)
{
        rows = m.rows;
        columns = m.columns;
        curr_num_proc=m.curr_num_proc;

        job_array= new float[columns];
        delay_array= new float[rows];
        time_array= new float[rows];
        original_order= new int[rows];

        cap_array= new float*[rows];
        load_array= new float*[rows];

        for(int r=0; r<rows; r++)
        {
            delay_array[r]= m.delay_array[r];
            time_array[r]= m.time_array[r];

            cap_array[r]= new float[columns];
            load_array[r]= new float[columns];
            original_order[r]= m.original_order[r];

            for(int c=0; c<columns; c++)
            {
                cap_array[r][c]= m.cap_array[r][c];
                load_array[r][c]= m.load_array[r][c];
            }
        }

        for(int cc=0; cc<columns; cc++)
        {
            job_array[cc]= m.job_array[cc];
        }
}


void data_arrays::initialize(ifstream &fin)
{
        // jobs, init del cap
        int x, y;
        for(x=0; x<columns; x++)
        {
            fin >> job_array[x];
        }
        for(x=0; x<rows; x++)
        {
            fin >> delay_array[x];
            time_array[x]= delay_array[x];
        }
        for(x=0; x<rows; x++)
        for(y=0; y<columns; y++)
        {
            fin >> cap_array[x][y];
            load_array[x][y]= 0.0;
        }
}


void data_arrays::bubble_sort()
{
    for(int i=1; i<rows; i++)
        for(int j=1; j<rows; j++)
            if(delay_array[j-1] > delay_array[j])
            {
                float temp= delay_array[j-1];
                delay_array[j-1]= delay_array[j];
                delay_array[j]= temp;

                temp= time_array[j-1];
                time_array[j-1]= time_array[j];
                time_array[j]= temp;

                float *temp_ptr= cap_array[j-1];
                cap_array[j-1]= cap_array[j];
                cap_array[j]= temp_ptr;

                int temp_int= original_order[j-1];
                original_order[j-1]= original_order[j];
                original_order[j]= temp_int;
            }
}


void data_arrays::print_temp_file(int number_of_lin_solve,
                             double millisec_per_iteration)
{

    ofstream os = ("temp.txt");
    if(!os)
    {
        throw FileNotFoundException();
    }

    os << "output file ";

    os << rows << " ";       // print number of processors
    os << columns << " ";    // print number of jobs


    // print job sizes
    int r, c, orig_order_index;
    for(c=0; c<columns; c++)
    {
        os << job_array[c] << " ";
    }

    // print initial delays
    for(orig_order_index=0; orig_order_index<rows; orig_order_index++)
    {
        for(r=0; r<rows; r++)
        {
            if(orig_order_index == original_order[r])
            {
                os << delay_array[r] << " ";
                break;
            }
        }
    }

    // print capacity and load data
    for(orig_order_index=0; orig_order_index<rows; orig_order_index++)
    {
        for(r=0; r<rows; r++)
        {
            if(orig_order_index == original_order[r])
            {
                for(c=0; c<columns; c++)
                {
                    os << cap_array[r][c] << " ";
                    os << load_array[r][c] << " ";
                }
                break;
            }
        }
    }

    // print the time data
    for(orig_order_index=0; orig_order_index<rows; orig_order_index++)
    {
        for(r=0; r<rows; r++)
        {
            if(orig_order_index == original_order[r])
            {
                os << time_array[r] << " ";
                break;
            }
        }
    }

    os << number_of_lin_solve << " ";
    os << millisec_per_iteration << " ";

    os.close();
}


data_arrays::~data_arrays()
{
    for(int i=0; i<rows; i++)
    {
        delete []cap_array[i];
        delete []load_array[i];
    }
    delete []cap_array;
    delete []load_array;
    delete []job_array;
    delete []delay_array;
    delete []time_array;
    delete []original_order;
}


void data_arrays::knapsack()
{
    // Give all load to first row
    int x;
    for(x=0; x<columns; x++)
    {
        load_array[0][x]= job_array[x];
        time_array[0]= (time_array[0]) +
            ((load_array[0][x])*(cap_array[0][x]));
    }
    curr_num_proc++;
    curr_num_proc++;

    // find column to redistribute by iterating through columns

    float cap1;
    float cap2;
    float cr_ratio;
    float max_ratio;
    int curr_max_index;

    while(((time_array[0]) - (time_array[1])) > THRESH)
    {
        int curr_col_index = 0;
        max_ratio = 0; // Initialize out of range
        for(curr_col_index=0; curr_col_index<columns; curr_col_index++)
        {
            cap1 = cap_array[0][curr_col_index];
            cap2 = cap_array[1][curr_col_index];
            cr_ratio = cap1/cap2;
            if( (cr_ratio > max_ratio) &&
                ( load_array[0][curr_col_index] > THRESH ) )
            {
                max_ratio = cr_ratio;
                curr_max_index = curr_col_index;
            }
        }

        float time1 = time_array[0];
        float time2 = time_array[1];

        cap1 = cap_array[0][curr_max_index];
        cap2 = cap_array[1][curr_max_index];

        float ld = load_array[0][curr_max_index];
        float safe_dist=(time1 - time2)/(cap1 + cap2);

        if(safe_dist < ld)
        {
            load_array[0][curr_max_index]= ld - safe_dist;
            load_array[1][curr_max_index]= safe_dist;
            time_array[0]= time_array[0] - ((safe_dist)*(cap1));
            time_array[1]= time_array[1] + ((safe_dist)*(cap2));
        }
        else
        {
            load_array[0][curr_max_index]= 0;
            load_array[1][curr_max_index]= ld;
            time_array[0]= time_array[0] - ((ld)*(cap1));
            time_array[1]= time_array[1] + ((ld)*(cap2));
        }
    }
}


// Calculates the weighted average of the capacities of the OTHER
// nodes in a column given a node (from) in the column
float data_arrays::calc_av_cap(int row, int col)
{
    float av_cap = 0;
    float product = 0;
    float divisor = 0;

    for(int curr_row=0; curr_row<row; curr_row++)
    {
        if( (load_array[curr_row][col]) > 0 )
        {
            product = product + ( (load_array[curr_row][col])*
                                  (cap_array[curr_row][col]) );
            divisor = divisor + (load_array[curr_row][col]);
        }
    }
    if( (divisor > 0) && (product > 0) )
    {
        av_cap = product/divisor;
    }

    return av_cap;
}


float data_arrays::calc_cap_ratio(float cap, float av_cap)
{
    return cap/av_cap;
}


/* Returns a void* (pointing to a graph).
   Otherwise, throws exceptions */
void* data_arrays::get_new_basis()
{
    int last_row= curr_num_proc -1;
    int curr_col;
    graph* g= NULL;
```

```
int num_cols= 0;
int* col_index_array= new int[columns];

for(curr_col=0; curr_col<columns; curr_col++)
{
    if(load_array[last_row][curr_col] ==0 )
    {
        col_index_array[num_cols]= curr_col;
        num_cols++;
    }
}


float* rel_cap_array= new float[columns];
if(num_cols>0)
{
    for(int d=0; d<num_cols; d++)
    {
        curr_col= col_index_array[d];
        rel_cap_array[d] =
            calc_cap_ratio( (cap_array[last_row][curr_col]),
                calc_av_cap(last_row, curr_col) );
    }
}
else
{
    delete[] col_index_array;
    delete[] rel_cap_array;
    throw new NodeNotFoundException();
}

if(num_cols > 1)
{
    // Bubble sort the list
    for(int i=1; i<num_cols; i++)
    {
        for(int j=1; j<num_cols; j++)
        {
            if(rel_cap_array[j-1] > rel_cap_array[j])
            {
                float temp= rel_cap_array[j-1];
                rel_cap_array[j-1]= rel_cap_array[j];
                rel_cap_array[j]= temp;

                int tempint= col_index_array[j-1];
                col_index_array[j-1]= col_index_array[j];
                col_index_array[j]= tempint;
            }
        }
    }
}


// find a cycle
for(int s=0; s<num_cols; s++)
{
    g= new graph(this);
```

```
            g->build_graph(last_row, col_index_array[s]);
            if(g->cycle_found())
            {
                break;
            }
            else
            {
                delete g;
            }
        }

        if(g==NULL)
        {
            delete[] col_index_array;
            delete[] rel_cap_array;
            throw new CycleNotFoundException();
        }

        delete[] col_index_array;
        delete[] rel_cap_array;

        return g;
    }


    int data_arrays::check_solution(from_to_node* from_to_array, float* x)
    {
        // from_to_array is from_to_array in graph
        // f_t_a_size is size of from_to_array (number of variables -1 (L))
        // x is solution array in eq_system

        int f_t_a_size= curr_num_proc -1;

        int ret_val= -1;
        // return index in from_to_array and x where Load will be <= 0
        // if the solutions were applied

        for(int i=0; i<f_t_a_size; i++)
        {
            if(((
               load_array[from_to_array[i].from_row][from_to_array[i].col])
               - x[i]) <= 0)
                ret_val = i;
        }

        return ret_val;
    }


    void data_arrays::apply_solution(from_to_node* from_to_array, float* x)
    {
        int f_t_a_size= curr_num_proc -1;
        int i;
        for(i=0; i<f_t_a_size; i++)
        {
            int frm_rw= from_to_array[i].from_row;
            int to_rw= from_to_array[i].to_row;
        int cl= from_to_array[i].col;

        load_array[frm_rw][cl]= load_array[frm_rw][cl] - x[i];
        load_array[to_rw][cl]= load_array[to_rw][cl] + x[i];

        time_array[frm_rw]= time_array[frm_rw] -
                        ((cap_array[frm_rw][cl])*(x[i]));
        time_array[to_rw]= time_array[to_rw] +
                        ((cap_array[to_rw][cl])*(x[i]));
    }
}


int data_arrays::get_index_most_neg(from_to_node* from_to_array,
    float* x)
{
    int index_most_neg= -1;            // initialize out of range
    float min_scale_factor= 1.1f;      // initialize out of range
    int fta_size= curr_num_proc -1;

    for(int i=0; i<fta_size; i++)
    {
        float load=
           load_array[from_to_array[i].from_row][from_to_array[i].col];

        if((load - x[i] <= 0) && (load/x[i] < min_scale_factor))
        {
            min_scale_factor= load/x[i];
            index_most_neg= i;
        }
    }
    return index_most_neg;
}


// Use this function since otherwise a load once became -.000001 (or
// something like that) instead of exactly zero.
void data_arrays::scale_and_apply_solution(from_to_node* from_to_array,
    float* x)
{
    int f_t_a_size= curr_num_proc -1;
    int index_most_neg = get_index_most_neg(from_to_array, x);
    float scale_factor = (load_array[from_to_array[index_most_neg]
        .from_row][from_to_array[index_most_neg].col])
            /x[index_most_neg];

    for(int i=0; i<f_t_a_size; i++)
    {
        int frm_rw= from_to_array[i].from_row;
        int to_rw= from_to_array[i].to_row;
        int cl= from_to_array[i].col;

        if(i == index_most_neg)
        {
            // Give and take the whole load (ld), no need to scale
            // this one, doing so may introduce a very small error
            float ld= load_array[frm_rw][cl];
            load_array[frm_rw][cl]= 0;
            load_array[to_rw][cl]= load_array[to_rw][cl] + ld;

            time_array[frm_rw]= time_array[frm_rw] -
                            ((cap_array[frm_rw][cl])*(ld));
            time_array[to_rw]= time_array[to_rw] +
                            ((cap_array[to_rw][cl])*(ld));
        }
        else
        {
            // Scale and apply all others
            x[i]= x[i]*scale_factor;

            load_array[frm_rw][cl]= load_array[frm_rw][cl] - x[i];
            load_array[to_rw][cl]= load_array[to_rw][cl] + x[i];

            time_array[frm_rw]= time_array[frm_rw] -
                            ((cap_array[frm_rw][cl])*(x[i]));
            time_array[to_rw]= time_array[to_rw] +
                            ((cap_array[to_rw][cl])*(x[i]));
        }
    }
}


int data_arrays::get_rows()
{
    return rows;
}


int data_arrays::get_columns()
{
    return columns;
}


int data_arrays::get_curr_num_proc()
{
    return curr_num_proc;
}


void data_arrays::increment_curr_num_proc()
{
    curr_num_proc++;
}


float** data_arrays::get_load_array()
{
    return load_array;
}


float** data_arrays::get_cap_array()
```

```
{
    return cap_array;
}


float* data_arrays::get_time_array()
{
    return time_array;
}


float* data_arrays::get_delay_array()
{
    return delay_array;
}
```

A.10 eq_system.cpp

```
// eq_system.cpp

// This class builds and solves a system of linear equations.
// Gaussian Elimination functions are dervived from those that appear
// in D.R. Brooks, C Programming: The Essentials for Engineers and
// Scientists, pp. 387-390, Springer-Verlag, 1999.

#include "../stdafx.h"
#include "../headers/eq_system.h"


eq_system::eq_system(data_arrays *ma)
{
    mat= ma;
    num_rows= mat->get_curr_num_proc();
    num_cols= num_rows +1;

    // Allocate a array of pointers to float pointers
    A = new float*[num_rows];

    // Allocate rows of data_arrays A
    for(int i=0; i<num_rows; i++)
    {
        A[i]= new float[num_cols];
    }

    // Initialize A
    for(int m=0; m<num_rows; m++)
        for(int n=0; n<num_cols; n++)
            A[m][n]= 0;

    // Allocate and initialize x (solution vector)
    x = new float[num_cols-1];
    for(int y=0; y<num_cols-1; y++)
        x[y]= 0;

    row= 0;
    col= 0;
}


eq_system::~eq_system()
{
    for(int i=0; i<num_rows; i++)
    {
        delete[] A[i];
        A[i]= NULL;
    }
    delete x;
    x= NULL;
}


float* eq_system::get_x()
{
    return x;
}


void eq_system::build_and_solve(graph* g)
{
    fill_data_arrays(g);
    triangularize();
    backsolve();
}


int eq_system::fill_data_arrays(graph* g)
{
    from_to_node* f_t_a = g->get_from_to_array();

    int i;

    // Iterate through the from to array
    for(i=0; i< num_cols-2; i++)
    {
        int fr= f_t_a[i].from_row;
        int tr= f_t_a[i].to_row;
        int cl= f_t_a[i].col;
        float** ca= mat->get_cap_array();
        A[fr][i] = (-1)*(ca[fr][cl]);
        A[tr][i] = ca[tr][cl];
        ca= NULL;
    }

    // Iterate through rows of A and data_arrays to fill in L (-1)
    // and T
    for(i=0; i<num_rows; i++)
    {
        A[i][num_cols-2]= -1;
        A[i][num_cols-1]= (-1)*(mat->get_time_array()[i]);
    }

    f_t_a= NULL;
    return 0;
}


int eq_system::triangularize()
{
    int pivot_row;
    float divide_by;
    float* temp;
    int i;
    int j;

    for (row=0; row < num_rows; row++)
    {
        /* Search for pivot row. */
        if (row < num_rows-1)
        {
            pivot_row=row;
            for (i=row+1; i < num_rows; i++)
                if (fabs(A[i][row]) > fabs(A[pivot_row][pivot_row]))
                    pivot_row=i;

            /* Swap pivot row if required. */
            if (pivot_row != row)
            {
                temp = A[row];
                A[row]= A[pivot_row];
                A[pivot_row]= temp;
            }
        }

        /* Divide by pivot. */
        divide_by=A[row][row];
        for (i=row; i < num_cols; i++)
        {
            if(A[row][i] != 0)
            {
                A[row][i]=A[row][i]/divide_by;
            }
        }

        /* Reduce the (row)th column to 0. */
        if (row < (num_rows-1))
        {
            for (i=row+1; i < num_rows; i++)
            {
                for (j=row+1; j < num_cols; j++)
                {
                    if( ((A[row][j]) != 0) && ((A[i][row]) != 0) )
                    {
                        A[i][j]=A[i][j]-((A[row][j])*(A[i][row]));
                    }
                }
                A[i][row]=0.0;
            }
        }
    }
    return 0;
}


int eq_system::backsolve()
{
    /* Backsolve for solutions. */
    x[num_rows-1]=A[num_rows-1][num_cols-1];
    for (row=num_rows-2; row >= 0; row--)
    {
        x[row]=A[row][num_cols-1];
        for (int i=row+1; i < num_rows; i++)
        {
            if( (A[row][i] != 0) )
            {
                x[row]=x[row]-A[row][i]*x[i];
            }
        }
    }

    return 0;
}
```

A.11 graph.cpp

```
// graph.cpp

#include "../stdafx.h"
#include "../headers/graph.h"

graph::graph(data_arrays* m)
{
    mat= m;
    cur_index = 0;
    num_vars = mat->get_curr_num_proc() -1;
    vars_allocated= 0;

    // Use this array to build the eq_system
    from_to_array = new from_to_node[num_vars];
}


graph::~graph()
{
    mat= NULL;
    delete[] from_to_array;
}


int graph::build_graph(int row, int col)
{
    parse_column(row, col);
    parse_row(row, col);
    return vars_allocated;
```

```
}


from_to_node* graph::get_from_to_array()
{
    return from_to_array;
}


// returns true if a problematic cycle does not exist
bool graph::assignment_OK(int new_frm_row, int new_to_row)
{
    bool ass= true;

    // iterate through the from_to_array
    for(int i= 0; i<vars_allocated; i++)
    {
        if(((from_to_array[i].to_row==new_frm_row)&&
            (from_to_array[i].from_row==new_to_row)) ||
           ((from_to_array[i].from_row == new_frm_row) &&
            (from_to_array[i].to_row == new_to_row)))
        {
            ass= false;
        }
    }
    return ass;
}


bool graph::rows_not_yet_connected(int new_from_row, int new_to_row)
{
    bool ass= true;

    // Iterate through the from_to_array
    for(int i=0; i<vars_allocated; i++)
    {
        if(from_to_array[i].from_row==new_from_row)
        {
            ass= false;
        }
    }
    return ass;
}


int graph::parse_column(int row, int col)
{
    int num_var = 0;

    int curr_row;
    if(vars_allocated < num_vars)
    {
        for(curr_row=row-1; curr_row>=0; curr_row--)
        {
            if(((mat->get_load_array()[curr_row][col]) > 0 ) &&
                (assignment_OK(curr_row, row)) &&
                (rows_not_yet_connected(curr_row, row)))
                {
                    // Assign a variable to the source node
                    from_to_array[cur_index].from_row= curr_row;
                    from_to_array[cur_index].col= col;

                    // Add the variable to the destination node
                    from_to_array[cur_index].to_row= row;
                    cur_index++;
                    num_var++;
                    vars_allocated++;

                    parse_row(curr_row, col);
                }
            }
        }

        if(vars_allocated < num_vars)
        {
            for(curr_row=row+1; curr_row<(mat->get_curr_num_proc());
                curr_row++)
            {

                if(((mat->get_load_array()[curr_row][col]) > 0 ) &&
                    (assignment_OK(curr_row, row)) &&
                    (rows_not_yet_connected(curr_row, row)))
                {
                    // Assign a variable to the source node
                    from_to_array[cur_index].from_row= curr_row;
                    from_to_array[cur_index].col= col;

                    // Add the variable to the destination node
                    from_to_array[cur_index].to_row= row;
                    cur_index++;
                    num_var++;
                    vars_allocated++;

                    parse_row(curr_row, col);
                }
            }
        }
        return num_var;
    }


    int graph::parse_row(int row, int col)
    {
        int num_cols = 0;
        int curr_col;

        if(vars_allocated < num_vars)
        {
            for(curr_col=col-1; curr_col>=0; curr_col--)
            {
                if( (mat->get_load_array()[row][curr_col]) > 0 )
                {
                    parse_column(row, curr_col);
                    num_cols++;
                }
            }
        }

        if(vars_allocated < num_vars)
        {
            for(curr_col=col+1; curr_col<mat->get_columns(); curr_col++)
            {
                if( (mat->get_load_array()[row][curr_col]) > 0 )
                {
                    parse_column(row, curr_col);
                    num_cols++;
                }
            }
        }
        return num_cols;
}


/********* WARNING ******* WARNING **************/
// this function will only work when load is being
// added to the last row because of code below see **** NOTE ****
bool graph::cycle_found()
{
    bool found = true;
    int i=0;
    while( (found==true) && (i<num_vars) )
    {
        // row i is not yet found in from_to_array
        found = false;
        for(int indx= 0; indx<num_vars; indx++)
        {
            if((from_to_array[indx].from_row == i) ||
               (from_to_array[indx].to_row == i) )
            {
                // row i is found as a to_row or a from_row
                found = true;
            }
        }
        i++;
    }

    /**** NOTE ****/
    // make sure all rows except the last one are represented as
    // from_rows
    i= 0;
    bool another_found;
    for(i=0; i<num_vars; i++)
    {
        another_found= false;
        for(int y=0; y<num_vars; y++)
        {
            if(from_to_array[y].from_row == i)
            {
                another_found = true;
            }
        }
    }

    if( (found==true) && (another_found==true) )
    {
        found= true;
    }
    else
    {
        found= false;
    }


    return found;
}
```

APPENDIX B

GUI SOURCE CODE

This Appendix contains all of the source code for the graphical user interface. The exact procedure given in Appendix C can be used to compile these files.

B.1 BalancerWrapper.java

```
// BalancerWrapper.java

// This is a wrapper class simply used to load and run the dynamically
// linked library (dll) that does the actual load balancing.

package gui;

public class BalancerWrapper
{
    public final static int NO_ERRORS = 0;
    public final static int FILE_NOT_FOUND = 1;
    public final static int CYCLE_NOT_FOUND = 2;
    public final static int NODE_NOT_FOUND = 3;

    static
    {
        System.loadLibrary("loadbalancer");
    }

    public native int runBalancer(String inputFilePath);
}
```

B.2 ControlFrame.java

```
// ControlFrame.java

// This is the central class of the GUI. It provides the user with
// many options and is the first window the user will see.

package gui;

import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JFileChooser;
import javax.swing.JOptionPane;
import javax.swing.filechooser.*;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;


public class ControlFrame extends JFrame implements DialogListener
{
    JButton newFileButton;
    JButton editFileButton;
    JButton runBalancerButton;
    JButton veiwOutputFileButton;
    JButton getAverageTimeButton;
    JButton exitButton;

    DialogFrame df= null;

    Properties p;

    String homeDir;
    String inputHomeDir;
    String outputHomeDir;

    final JFileChooser inputFC;
    final JFileChooser outputFC;

    File currInputFile;
    File currOutputFile;

    int NUM_RUNS= 6;

    JFrame timeDisplayFrame;

    public ControlFrame()
    {
        super("Load Balancer");

        // Create the file choosers
        inputFC = new JFileChooser();
        inputFC.setFileFilter(new MyFileFilter("txt"));
        outputFC= new JFileChooser();
        outputFC.setFileFilter(new MyFileFilter("out"));

        loadPreferences();

        // Create the help menu components for setting preferences
        JMenuBar menuBar;
        JMenu menu, subMenu;
        JMenuItem subMenuItem1, subMenuItem2, subMenuItem3;

        // Create the menu bar.
        menuBar = new JMenuBar();
        setJMenuBar(menuBar);
```

```
// Build the first menu.
menu = new JMenu("Help");
menu.setMnemonic(KeyEvent.VK_H);
menu.getAccessibleContext().setAccessibleDescription(
        "Help");
menuBar.add(menu);

// build submenu
subMenu = new JMenu("Set Preferences");
menu.add(subMenu);

// subMenuItems
subMenuItem1= new JMenuItem("Set input file directory");
subMenu.add(subMenuItem1);
subMenuItem2= new JMenuItem("Set output file directory");
subMenu.add(subMenuItem2);
subMenuItem3= new JMenuItem("Restore default directories");
subMenu.add(subMenuItem3);

subMenuItem1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        setInputFileDirectory();
        repaint();
    }});

subMenuItem2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        setOutputFileDirectory();
        repaint();
    }});

subMenuItem3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        setToDefaultDirectories();
        repaint();
    }});


newFileButton = new JButton("Create a New Data File");
newFileButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae)
    {
        df= new DialogFrame(ControlFrame.this);
    }
});

editFileButton= new JButton("Veiw and/or Edit a Data File");
editFileButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
        int returnVal =
            inputFC.showOpenDialog(ControlFrame.this);

        if (returnVal == JFileChooser.APPROVE_OPTION)
```

```
            {
                currInputFile = inputFC.getSelectedFile();
                if(isDataFile(currInputFile))
                {
                    InputFileViewer ifv=
                        new InputFileViewer(currInputFile,
                            ControlFrame.this);
                }
                else
                {
                    JOptionPane.showMessageDialog(ControlFrame.this
                        ,currInputFile.getName() + " is not a valid
                        data file","Message",
                        JOptionPane.ERROR_MESSAGE);
                }
            }
            else
            {
                // cancelled by user
            }
        }
    });

    runBalancerButton= new JButton("Run Balancer");
    runBalancerButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae)
        {
            try
            {
                int returnVal =
                    inputFC.showDialog(ControlFrame.this, "Run");
                inputFC.setApproveButtonText("Open");

                if (returnVal == JFileChooser.APPROVE_OPTION)
                {

                    currInputFile = inputFC.getSelectedFile();
                    if(isDataFile(currInputFile))
                    {

                        BalancerWrapper bw= new BalancerWrapper();
                        int balancerStatus=
                           bw.runBalancer(currInputFile.getPath());
                        if(balancerStatus ==
                            BalancerWrapper.NO_ERRORS)
                        {
                            //default title and icon
                            JOptionPane.showMessageDialog(
                                ControlFrame.this,
                                "Balancer worked successfully");
                            OutputFileViewer ofv=
                                new OutputFileViewer(
                                    new File("temp.txt"),
                                    ControlFrame.this);
                        }
                        else if(balancerStatus ==
                            BalancerWrapper.FILE_NOT_FOUND)
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(
                    ControlFrame.this,
                    "A FileNotFound exception occured
                    while balancing", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
            else if(balancerStatus ==
                BalancerWrapper.CYCLE_NOT_FOUND)
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(
                    ControlFrame.this,
                    "A CycleNotFound exception occured
                    while balancing", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
                OutputFileViewer ofv=
                    new OutputFileViewer(
                        new File("temp.txt"),
                        ControlFrame.this);
            }
            else if(balancerStatus ==
                BalancerWrapper.NODE_NOT_FOUND)
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(
                    ControlFrame.this,
                    "A NodeNotFound exception occured
                    while balancing", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
                OutputFileViewer ofv=
                    new OutputFileViewer(
                        new File("temp.txt"),
                        ControlFrame.this);
            }
            else
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(
                    ControlFrame.this,
                    "A non-specified exception occured
                    while balancing", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
        else
        {
            //custom title, error icon
            JOptionPane.showMessageDialog(
                ControlFrame.this,
                currInputFile.getName() + " is not a
                valid data file","Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }
    else
```

```
                {
                    // cancelled by user
                }
            }
            catch(Exception e)
            {
                JOptionPane.showMessageDialog(ControlFrame.this,
                    "Exception ocurred in actionPerformed().",
                    "Error Message", JOptionPane.ERROR_MESSAGE);
                PrintStream errorPS;
                try
                {
                    errorPS= new PrintStream(new FileOutputStream(
                                 new File("errors.txt")));
                    e.printStackTrace(errorPS);
                    errorPS.close();
                }
                catch(IOException ioe)
                {
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        "A stream to errors.txt could not be
                        opened.", "Error Message",
                        JOptionPane.ERROR_MESSAGE);
                }
            }
            catch(Error e)
            {
                //e.printStackTrace(errPS);
                JOptionPane.showMessageDialog(ControlFrame.this,
                    "DLL or function could not be found.",
                    "Error Message", JOptionPane.ERROR_MESSAGE);
            }
        }
    });

    veiwOutputFileButton= new JButton("Veiw an Output File");
    veiwOutputFileButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae) {
            int returnVal =
                outputFC.showOpenDialog(ControlFrame.this);

            if (returnVal == JFileChooser.APPROVE_OPTION)
            {
                currOutputFile = outputFC.getSelectedFile();
                if(isOutputFile(currOutputFile))
                {
                    OutputFileViewer ofv=
                        new OutputFileViewer(currOutputFile,
                                             ControlFrame.this);
                }
                else
                {
                    //custom title, error icon
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        currOutputFile.getName() + " is not a valid
                        output file", "Message",
                        JOptionPane.ERROR_MESSAGE);
                }
            }
            else
            {
                // cancelled by user
            }
        }
    });

    getAverageTimeButton= new JButton("Get Average Run Time");
    getAverageTimeButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ae)
        {
            df= new DialogFrame(ControlFrame.this);
            df.setButtonText("get time");
        }
    });

    exitButton= new JButton("Exit");
    exitButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent ae)
        {
            System.exit(0);
        }
    });


    addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
        System.exit(0); } });

    getContentPane().setLayout(new GridLayout(0,1));
    getContentPane().add(new JLabel(""));
    getContentPane().add(newFileButton);
    getContentPane().add(editFileButton);
    getContentPane().add(runBalancerButton);
    getContentPane().add(veiwOutputFileButton);
    getContentPane().add(getAverageTimeButton);
    getContentPane().add(exitButton);

    setSize(275,275);
    setLocation(300, 200);
    setVisible(true);
}

public void dialogActionPerformed(DialogEvent de)
{
    if(de.getSource().equals("OK"))
    {
        int numProc= df.getNumberOfProcessors();
        int numJobs= df.getNumberOfJobs();
        if((numProc > 0) && (numJobs > 0))
        {
            df.dispose();
            InputFileViewer ifv= new InputFileViewer(
                numProc, numJobs, ControlFrame.this);
        }
    }
    else if(de.getSource().equals("get time"))
    {
        int numProc= df.getNumberOfProcessors();
        int numJobs= df.getNumberOfJobs();
        if((numProc > 0) && (numJobs > 0))
        {
            df.dispose();
            float totalTime= 0.0f;
            int successfulRuns= NUM_RUNS;
            timeDisplayFrame= new JFrame(Integer.toString(numProc)
                +" Proc " + Integer.toString(numJobs) +" Jobs");
            timeDisplayFrame.getContentPane().setLayout(new
                GridLayout(0,1));

            JLabel runTimeLabel[]= new JLabel[NUM_RUNS];
            for(int i=0; i<NUM_RUNS; i++)
            {
                File tempInFile= new File("tempin.txt");
                generateRandomFile(numProc, numJobs, tempInFile);
                // run balancer
                BalancerWrapper bw= new BalancerWrapper();
                int balancerStatus=
                    bw.runBalancer(tempInFile.getPath());
                if(balancerStatus == BalancerWrapper.NO_ERRORS)
                {
                    float time= getTime(new File("temp.txt"));
                    totalTime+= time;
                    runTimeLabel[i]= new JLabel(
                        "Run " + (i+1)+ ": " + time + " ms");
                    timeDisplayFrame.getContentPane().add(
                        runTimeLabel[i]);
                    File t= new File("temp.txt");
                    t.delete();
                }
                else if(balancerStatus ==
                    BalancerWrapper.FILE_NOT_FOUND)
                {
                    //custom title, error icon
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        "A FileNotFound exception occured while
                        balancing", "Error Message",
                        JOptionPane.ERROR_MESSAGE);
                    runTimeLabel[i]= new JLabel(
                        "FileNotFound exception");
                    timeDisplayFrame.getContentPane().add(
                        runTimeLabel[i]);
                    successfulRuns--;
                }
                else if(balancerStatus ==
                    BalancerWrapper.CYCLE_NOT_FOUND)
                {
                    //custom title, error icon
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        "A CycleNotFound exception occured while
                        balancing", "Error Message",
                        JOptionPane.ERROR_MESSAGE);
                    runTimeLabel[i]= new JLabel(
                        "CycleNotFound exception");
                    timeDisplayFrame.getContentPane().add(
                        runTimeLabel[i]);
                    successfulRuns--;
                }
                else if(balancerStatus ==
                    BalancerWrapper.NODE_NOT_FOUND)
                {
                    //custom title, error icon
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        "A NodeNotFound exception occured while
                        balancing", "Error Message",
                        JOptionPane.ERROR_MESSAGE);
                    runTimeLabel[i]= new JLabel(
                        "NodeNotFound exception");
                    timeDisplayFrame.getContentPane().add(
                        runTimeLabel[i]);
                    successfulRuns--;
                }
                else
                {
                    //custom title, error icon
                    JOptionPane.showMessageDialog(
                        ControlFrame.this,
                        "A non-specified exception occured while
                        balancing", "Error Message",
                        JOptionPane.ERROR_MESSAGE);
                    runTimeLabel[i]= new JLabel(
                        "non-specified exception");
                    timeDisplayFrame.getContentPane().add(
                        runTimeLabel[i]);
                    successfulRuns--;
                }
                tempInFile.delete();
            }// end for
            float averageTime= totalTime/successfulRuns;
            timeDisplayFrame.getContentPane().add(new JLabel(
                "Average Time: "+averageTime + " milliseconds
                (ms)"));
            timeDisplayFrame.pack();
            timeDisplayFrame.setLocation(200, 100);
            timeDisplayFrame.setVisible(true);
        }
    }
    else if(de.getSource().equals("cancelButton"))
    {
        df.dispose();
    }
    else if(de.getSource().equals("windowClosed"));
    {
        df.dispose();
    }
}

float getTime(File fl)
{
    float returnValue= 0.0f;
    try
    {
        BufferedReader br = new BufferedReader(new FileReader(fl));

        String dataString= br.readLine();
        StringTokenizer st= new StringTokenizer(dataString);

        try
        {
            st.nextToken();  // remove "output"
            st.nextToken();  // remove "file"
            int np= Integer.parseInt(st.nextToken());
            int nj= Integer.parseInt(st.nextToken());

            for(int i=0; i<nj; i++)
            {
                st.nextToken();
            }

            for(int i=0; i<np; i++)
            {
                st.nextToken();
            }

            for(int i=0; i<np; i++)
            {
                for(int j=0; j<nj; j++)
                {
                    st.nextToken();
                    st.nextToken();
                }
            }
            // end capacity and load data

            // get times data
            for(int i=0; i<np; i++)
            {
                st.nextToken();
            }
            // end of times data

            // get other number of linear systems solved
            st.nextToken();

            // get time
            returnValue= Float.parseFloat(st.nextToken());
        }
        catch(NoSuchElementException nsee)
        {
            //custom title, error icon
            JOptionPane.showMessageDialog(ControlFrame.this,
                "NSEException caught in OutputFileVeiwer
                 constructor ", "Message",
                 JOptionPane.ERROR_MESSAGE);
            }
            br.close();
        }
        catch(IOException ioe)
        {
            //custom title, error icon
            JOptionPane.showMessageDialog(ControlFrame.this,
                "IOException caught in getTime(File)",
                "Message", JOptionPane.ERROR_MESSAGE);
        }
        catch(Error e)
        {
            //custom title, error icon
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Error caught in getTime(File)",
                "Message", JOptionPane.ERROR_MESSAGE);
        }
        return returnValue;
    }

    void generateRandomFile(int numProc, int numJobs, File f)
    {
        try
        {
            PrintWriter pw = new PrintWriter(
                             new FileOutputStream(f));

            pw.print("data file ");
            pw.print(numProc);
            pw.print(" ");
            pw.print(numJobs);
            pw.print(" ");

            for(int i=0; i< numJobs; i++)
            {
                pw.print(generateRandomNumber(100));
                pw.print(" ");
            }

            for(int i=0; i<numProc; i++)
            {
                pw.print(generateRandomNumber(50));
                pw.print(" ");
            }

            for(int i=0; i<numProc; i++)
            {
                for(int j=0; j<numJobs; j++)
                {
                    pw.print(generateRandomNumber(10));
                    pw.print(" ");
                }
            }
            pw.close();
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "An IOException was caught while writing the data
                file", "Message", JOptionPane.ERROR_MESSAGE);
            dispose();
        }
    }

    int generateRandomNumber(int max)
    {
        int returnNumber= 0;
        while(returnNumber < 1)
        {
            returnNumber= Math.round((float)(Math.random()*max));
        }
        return returnNumber;
    }

    boolean isDataFile(File file)
    {
        boolean returnValue= false;
        try
        {
            BufferedReader br = new BufferedReader(
                new FileReader(file));

            String dataString= br.readLine();
            StringTokenizer st= new StringTokenizer(dataString);

            try
            {
                if( (st.nextToken().equalsIgnoreCase("data")) &&
                    (st.nextToken().equalsIgnoreCase("file")))
                {
                    returnValue= true;
                }
            }
            catch(NoSuchElementException nsee)
            {
                //custom title, error icon
                JOptionPane.showMessageDialog(ControlFrame.this,
                    "Exception caught in isDataFile(" +
                    file.getName()+")", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
            br.close();
        }
        catch(IOException ioe)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "IOException caught in isDataFile()",
                "Error Message", JOptionPane.ERROR_MESSAGE);
        }

        return returnValue;
```

```
}

boolean isOutputFile(File file)
{
    boolean returnValue= false;
    try
    {
        BufferedReader br = new BufferedReader(
            new FileReader(file));

        String dataString= br.readLine();
        StringTokenizer st= new StringTokenizer(dataString);

        try
        {
            if( (st.nextToken().equalsIgnoreCase("output")) &&
                (st.nextToken().equalsIgnoreCase("file")))
            {
                returnValue= true;
            }
        }
        catch(NoSuchElementException nsee)
        {
            //custom title, error icon
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Exception caught in isOutputFile(" +
                file.getName()+")", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
        br.close();
    }
    catch(IOException ioe)
    {
        JOptionPane.showMessageDialog(ControlFrame.this,
            "IOException caught in isOutputFile()",
            "Error Message", JOptionPane.ERROR_MESSAGE);
    }

    return returnValue;
}

void loadPreferences()
{
    p= new Properties();
    try
    {
        p.load(new FileInputStream(
            "gui"+File.separator+"preferences.txt"));

        if((inputHomeDir=p.getProperty("inputHomeDir"))!=null)
        {
            File file= new File(inputHomeDir);
            inputFC.setCurrentDirectory(file);
        }
        else
        {
            File temporary= new File(
                    "gui"+File.separator+"temporary.txt");
                JFileChooser jfc= new JFileChooser(
                    temporary.getParentFile());
                jfc.changeToParentDirectory();
                inputFC.setCurrentDirectory(jfc.getCurrentDirectory());
                temporary.delete();
                jfc= null;
                currInputFile= inputFC.getCurrentDirectory();
            }

            if((outputHomeDir=p.getProperty("outputHomeDir"))!=null)
            {
                File file= new File(outputHomeDir);
                outputFC.setCurrentDirectory(file);
            }
            else
            {
                File temporary= new File(
                    "gui"+File.separator+"temporary.txt");
                JFileChooser jfc= new JFileChooser(
                    temporary.getParentFile());
                jfc.changeToParentDirectory();
                outputFC.setCurrentDirectory(
                    jfc.getCurrentDirectory());
                temporary.delete();
                jfc= null;
                currOutputFile= outputFC.getCurrentDirectory();
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Your preferences were not loaded.\n" +
                "See help menu -> Set Preferences.",
                "Error Message", JOptionPane.ERROR_MESSAGE);
        }
    }

    void setInputFileDirectory()
    {
        File temporary= new File("gui"+File.separator+"temporary.txt");
        JFileChooser jfc= new JFileChooser(temporary.getParentFile());
        jfc.changeToParentDirectory();
        temporary.delete();

        jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        int returnVal = jfc.showDialog(
            ControlFrame.this, "Set input dir.");

        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            inputFC.setCurrentDirectory(jfc.getSelectedFile());
            currInputFile= inputFC.getCurrentDirectory();
            p= new Properties();
            try
            {
                p.load(new FileInputStream(
                "gui"+File.separator+"preferences.txt"));
            p.setProperty("inputHomeDir",
                inputFC.getCurrentDirectory().getPath());
            p.store(new FileOutputStream(
                "gui/preferences.txt"), "Preferences");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Error occured while setting input directory
                preference.\n" +
                "The file \"gui\\preferences.txt\"
                could not be found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }
    else
    {
        // cancelled by user
    }
    jfc= null;
}

void setOutputFileDirectory()
{
    File temporary= new File("gui"+File.separator+"temporary.txt");
    JFileChooser jfc= new JFileChooser(temporary.getParentFile());
    jfc.changeToParentDirectory();
    temporary.delete();

    jfc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
    int returnVal = jfc.showDialog(ControlFrame.this,
        "Set output dir.");

    if (returnVal == JFileChooser.APPROVE_OPTION)
    {
        outputFC.setCurrentDirectory(jfc.getSelectedFile());
        currOutputFile= outputFC.getCurrentDirectory();
        p= new Properties();
        try
        {
            p.load(new FileInputStream(
                "gui"+File.separator+"preferences.txt"));
            p.setProperty("outputHomeDir",
                outputFC.getCurrentDirectory().getPath());
            p.store(new FileOutputStream("gui"+File.separator+
                "preferences.txt"), "Preferences");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Error occured while setting output directory
                preference.\n" + "The file \"gui\\preferences.txt\"
                could not be found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }
```

```
        else
        {
            // cancelled by user
        }
        jfc= null;
    }

    void setToDefaultDirectories()
    {
        File temporary= new File("gui"+File.separator+"temporary.txt");
        JFileChooser jfc= new JFileChooser(temporary.getParentFile());
        jfc.changeToParentDirectory();
        inputFC.setCurrentDirectory(jfc.getCurrentDirectory());
        outputFC.setCurrentDirectory(jfc.getCurrentDirectory());
        temporary.delete();
        jfc= null;
        currInputFile= inputFC.getCurrentDirectory();
        currOutputFile= outputFC.getCurrentDirectory();
        p= new Properties();
        try
        {
            p.load(new FileInputStream(
                "gui"+File.separator+"preferences.txt"));
            p.setProperty("inputHomeDir",
                inputFC.getCurrentDirectory().getPath());
            p.setProperty("outputHomeDir",
                outputFC.getCurrentDirectory().getPath());
            p.store(new FileOutputStream("gui"+File.separator+
                "preferences.txt"), "Preferences");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(ControlFrame.this,
                "Error occured while setting default directory
                preference.\n" + "The file \"gui\\preferences.txt\"
                could not be found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}
```

B.3 DialogEvent.java

```
// DialogEvent.java

// This is a simple class used in communication with user.

package gui;

public class DialogEvent
{
    String src;

    DialogEvent(String source)
    {
        src= source;
    }

    public String getSource()
    {
        return src;
    }
}
```

B.4 DialogFrame.java

```
// DialogFrame.java

// This is a simple class used in communication with user.

package gui;

import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JTextField;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

public class DialogFrame extends JFrame
{
    int numberOfProcessors= 0;
    int numberOfJobs= 0;

    JTextField procField;
    JTextField jobField;

    JButton okButton;
    JButton cancelButton;

    ControlFrame cf;

    public DialogFrame(ControlFrame parentFrame)
    {
        super("Enter Data");

        cf= parentFrame;

        //in the constructor for a JFrame subclass:
```

```
JMenuBar menuBar;
JMenu menu, submenu;
JMenuItem menuItem;

//Create the menu bar.
menuBar = new JMenuBar();
setJMenuBar(menuBar);

//Build the first menu.
menu = new JMenu("Help");
menu.setMnemonic(KeyEvent.VK_H);
menu.getAccessibleContext().setAccessibleDescription(
    "Help");
menuBar.add(menu);

//a group of JMenuItems
menuItem = new JMenuItem(
    "How to Enter Data", KeyEvent.VK_T);
menuItem.setAccelerator(
    KeyStroke.getKeyStroke(KeyEvent.VK_1,
    ActionEvent.ALT_MASK));
menuItem.getAccessibleContext().setAccessibleDescription(
    "How to enter data correctly");
menuItem.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        // Do something here
    }});
menu.add(menuItem);


getContentPane().setLayout(new GridLayout(4,2));
getContentPane().add(new JLabel(" "));
getContentPane().add(new JLabel(" "));
getContentPane().add(new JLabel("Number of Processors: "));
procField= new JTextField(15);
procField.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        numberOfProcessors=
            Integer.parseInt(procField.getText());
    }});

getContentPane().add(procField);
getContentPane().add(new JLabel("Number of Jobs: "));
jobField= new JTextField(15);
jobField.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        numberOfJobs= Integer.parseInt(jobField.getText());
    }});

getContentPane().add(jobField);
getContentPane().add(new JLabel(" "));

JPanel buttonPanel= new JPanel(new GridLayout(1,2));
okButton= new JButton("OK");
```

```
okButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae)
    {
        try
        {
            numberOfProcessors=
                Integer.parseInt(procField.getText());
            numberOfJobs= Integer.parseInt(jobField.getText());
            if((numberOfProcessors<1)&&(numberOfJobs<1))
            {
                JOptionPane.showMessageDialog(DialogFrame.this,
                    "Number of Processors & Jobs must be
                    greater than 0", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
            else if(numberOfProcessors<1)
            {
                JOptionPane.showMessageDialog(DialogFrame.this,
                    "Number of Processors must be greater than
                    0", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
            else if(numberOfJobs<1)
            {
                JOptionPane.showMessageDialog(DialogFrame.this,
                    "Number of Jobs must be greater than 0",
                    "Error Message",JOptionPane.ERROR_MESSAGE);
            }
            else
            {
                cf.dialogActionPerformed(new
                    DialogEvent(okButton.getText()));
            }
        }
        catch(Exception e)
        {
            numberOfProcessors= 0;
            numberOfJobs= 0;
            JOptionPane.showMessageDialog(DialogFrame.this,
                "Data entered incorrectly. See Help -> How to
                Enter Data.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }});

cancelButton= new JButton("Cancel");
cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e)
    {
        numberOfProcessors= 0;
        numberOfJobs= 0;
        cf.dialogActionPerformed(new
            DialogEvent("cancelButton"));
    }});

buttonPanel.add(okButton);
buttonPanel.add(cancelButton);
```

```
        getContentPane().add(buttonPanel);

        setLocation(250, 230);
        setVisible(true);
        pack();

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                numberOfProcessors= 0;
                numberOfJobs= 0;
                cf.dialogActionPerformed(new
                    DialogEvent("windowClosed"));
            }
            });
    }

    public int getNumberOfProcessors()
    {
        return numberOfProcessors;
    }

    public int getNumberOfJobs()
    {
        return numberOfJobs;
    }

    public void setButtonText(String txt)
    {
        okButton.setText(txt);
    }
}
```

B.5 DialogListener.java

```
// DialogListener.java

// This is a simple interface used in communication with user.

package gui;

public interface DialogListener
{
    public void dialogActionPerformed(DialogEvent de);
};
```

B.6 InputFileViewer.java

```
// InputFileViewer.java

// This class enables the user to create, save, view and edit input
// data files.
```

```
package gui;

import javax.swing.JTable;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.TableCellEditor;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JFileChooser;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableColumn;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.TableModelListener;
import javax.swing.event.TableModelEvent;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import javax.swing.WindowConstants;
import javax.swing.JColorChooser;
import java.io.*;
import java.util.*;


public class InputFileViewer extends JFrame
{
    boolean dataHasChanged= false;

    boolean named;

    File readFile= null;
    ControlFrame parentCF;

    int numberOfJobs;
    int numberOfProc;

    JTable dataTable;

    Object[][] data;

    String[] colNames;
    String[] procNames;

    int InJobSizeColor;
    int InDelayColor;
    int InCapColor;

    JLabel jobSizeLabel;
    JLabel delayLabel;
JLabel capLabel;

JButton randNumButton;



public InputFileViewer(int numProc, int numJobs,
    ControlFrame parent)
{
    super("Not Yet Named");

    numberOfProc= numProc;
    numberOfJobs= numJobs;
    parentCF= parent;

    named = false;

    loadColorProperties();

    data= new Object[numberOfProc+1][numberOfJobs+1];
    for(int f=0; f<numberOfProc+1; f++)
    {
        for(int g=0; g<numberOfJobs+1; g++)
        {
            data[f][g]= new String();
        }
    }

    initTableHeaders();
    makeTheFrame();

    pack();
    setVisible(true);
}


public InputFileViewer(File file, ControlFrame parent)
{
    super(file.getName());
    readFile= file;
    parentCF= parent;

    named = true;

    loadColorProperties();

    try
    {
        BufferedReader br = new BufferedReader(
            new FileReader(readFile));

        String dataString= br.readLine();
        StringTokenizer st= new StringTokenizer(dataString);

        try
        {
            st.nextToken();  // remove "data"
            st.nextToken();  // remove "file"
            numberOfProc= Integer.parseInt(st.nextToken());
            numberOfJobs= Integer.parseInt(st.nextToken());

            data = new Object[numberOfProc+1][numberOfJobs+1];
            data[0][0]= new String();

            for(int i=1; i< numberOfJobs+1; i++)
            {
                data[0][i]= st.nextToken();
            }

            for(int i=1; i<numberOfProc+1; i++)
            {
                data[i][0]= st.nextToken();
            }

            for(int i=1; i<numberOfProc+1; i++)
            {
                for(int j=1; j< numberOfJobs+1; j++)
                {
                    data[i][j]= st.nextToken();
                }
            }
        }
        catch(NoSuchElementException nsee)
        {
            JOptionPane.showMessageDialog(InputFileViewer.this,
                "A NoSuchElementException was caught while reading
                the data file", "Message",
                JOptionPane.ERROR_MESSAGE);
            dispose();
        }
        br.close();
    }
    catch(IOException ioe)
    {
        JOptionPane.showMessageDialog(InputFileViewer.this,
            "An IOException was caught while reading the data
            file", "Message",
            JOptionPane.ERROR_MESSAGE);
        dispose();
    }

    initTableHeaders();
    makeTheFrame();

    pack();
    setVisible(true);
}


void initTableHeaders()
{
    colNames = new String[numberOfJobs+1];
    colNames[0]= " ";
    for(int y=1; y<numberOfJobs+1; y++)
    {
            colNames[y]= "Job " + (y);
        }
        procNames = new String[numberOfProc+1];
        procNames[0]= " ";
        for(int b=1; b<numberOfProc+1; b++)
        {
            procNames[b]= "Proc " + (b);
        }
    }

    void makeTheFrame()
    {
        /************** Start of menu code ***********************/
        // Create the help menu components for setting preferences
        JMenuBar menuBar;
        JMenu menu, subMenu;
        JMenuItem subMenuItem1, subMenuItem2, subMenuItem3;

        // Create the menu bar.
        menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        // Build the first menu.
        menu = new JMenu("Help");
        menu.setMnemonic(KeyEvent.VK_H);
        menu.getAccessibleContext().setAccessibleDescription(
            "Help");
        menuBar.add(menu);

        // build submenu
        subMenu = new JMenu("Set Colors");
        menu.add(subMenu);

        // subMenuItems
        subMenuItem1= new JMenuItem("Set Job Size Color");
        subMenu.add(subMenuItem1);
        subMenuItem2= new JMenuItem("Set Initial Delay Color");
        subMenu.add(subMenuItem2);
        subMenuItem3= new JMenuItem("Set Capacity Color");
        subMenu.add(subMenuItem3);

        subMenuItem1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent evt)
            {
                Color newColor= JColorChooser.showDialog(
                    InputFileViewer.this, "Choose Job Size Color",
                    new Color(InJobSizeColor));
                if(newColor != null)
                {
                    InJobSizeColor= newColor.getRGB();
                    jobSizeLabel.setForeground(newColor);
                    repaint();
                    storeColorProperties();
                }
            }});

        subMenuItem2.addActionListener(new ActionListener(){
```

```
    public void actionPerformed(ActionEvent evt)
    {
        Color newColor= JColorChooser.showDialog(
            InputFileViewer.this, "Choose Initial Delay
            Color", new Color(InDelayColor));
        if(newColor != null)
        {
            InDelayColor= newColor.getRGB();
            delayLabel.setForeground(newColor);
            repaint();
            storeColorProperties();
        }
    }});

subMenuItem3.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        Color newColor= JColorChooser.showDialog(
            InputFileViewer.this, "Choose Capacity Color",
            new Color(InCapColor));
        if(newColor != null)
        {
            InCapColor= newColor.getRGB();
            capLabel.setForeground(newColor);
            repaint();
            storeColorProperties();
        }

    }});
/******************** End of menu code ********************/

ColorLegendPanel lcp = new ColorLegendPanel();

getContentPane().add(lcp, BorderLayout.NORTH);
getContentPane().add(new DataPanel(), BorderLayout.CENTER);

// This code makes the buttons and the panels that hold them
JButton saveAsButton= new JButton("Save As ");
JButton saveButton= new JButton("  Save  ");
JButton closeButton= new JButton(" Close ");
MultipleButtonListener MBL= new MultipleButtonListener();
saveAsButton.addActionListener(MBL);
saveButton.addActionListener(MBL);
closeButton.addActionListener(MBL);
randNumButton.addActionListener(MBL);
JPanel jp1= new JPanel(new GridLayout(1, 3));
jp1.add(saveAsButton);
jp1.add(saveButton);
jp1.add(closeButton);
JPanel jp2= new JPanel(new BorderLayout());
jp2.add(jp1, BorderLayout.EAST);

getContentPane().add(jp2, BorderLayout.SOUTH);

setLocation(125, 75);

setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE );
```

```
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e)
            {
                if(dataTable.isEditing())
                {
                    TableCellEditor tce= dataTable.getCellEditor(
                        dataTable.getEditingRow(),
                        dataTable.getEditingColumn());
                    tce.stopCellEditing();
                }

                if(dataHasChanged)
                {
                    int n = JOptionPane.showConfirmDialog(
                        InputFileViewer.this,
                        "You have made changes that have not yet\n" +
                        "been saved. Do you want to continue closing?",
                        "Continue Closing?",
                        JOptionPane.OK_CANCEL_OPTION);

                    if(n == JOptionPane.CANCEL_OPTION)
                    {
                        /* do nothing */
                    }
                    else
                    {
                        dispose();
                    }
                }
                else
                {
                    dispose();
                }
            }
            });
    }


    void writeToFile(File file)
    {
        try
        {
            PrintWriter pw = new PrintWriter(
                            new FileOutputStream(file));

            pw.print("data file ");
            pw.print(numberOfProc);
            pw.print(" ");
            pw.print(numberOfJobs);
            pw.print(" ");

            for(int i=1; i< numberOfJobs+1; i++)
            {
                pw.print(dataTable.getValueAt(0, i));
                pw.print(" ");
            }
            for(int i=1; i<numberOfProc+1; i++)
            {
                pw.print(dataTable.getValueAt(i, 0));
                pw.print(" ");
            }

            for(int i=1; i<numberOfProc+1; i++)
            {
                for(int j=1; j< numberOfJobs+1; j++)
                {
                    pw.print(dataTable.getValueAt(i, j));
                    pw.print(" ");
                }
            }
            pw.close();
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(InputFileViewer.this,
                "An IOException was caught while writing the data
                file", "Message",
                JOptionPane.ERROR_MESSAGE);
            dispose();
        }

    }

    boolean allFieldsAreValid()
    {
        boolean retVal = true;

        for(int i=0; i<numberOfProc+1; i++)
        {
            for(int j=0; j< numberOfJobs+1; j++)
            {
                if( (!((i==0)&&(j==0))) &&
                    ((String)dataTable.getValueAt(i, j)).equals("") )
                {
                    retVal= false;
                }
            }
        }

        return retVal;
    }



    class ColorLegendPanel extends JPanel
    {
        ColorLegendPanel()
        {
            setLayout(new BorderLayout());

            JPanel LabelHolderPanel= new JPanel(new GridLayout(0,1));

            jobSizeLabel= new JLabel(" Job Size ");
        jobSizeLabel.setForeground(new Color(InJobSizeColor));
        jobSizeLabel.setFont(
            jobSizeLabel.getFont().deriveFont(14.0f));
        delayLabel= new JLabel(" Initial Delays ");
        delayLabel.setForeground(new Color(InDelayColor));
        delayLabel.setFont(delayLabel.getFont().deriveFont(14.0f));
        capLabel= new JLabel(" Capacities ");
        capLabel.setForeground(new Color(InCapColor));
        capLabel.setFont(capLabel.getFont().deriveFont(14.0f));

        LabelHolderPanel.add(new JLabel(" "));
        LabelHolderPanel.add(jobSizeLabel);
        LabelHolderPanel.add(delayLabel);
        LabelHolderPanel.add(capLabel);
        LabelHolderPanel.add(new JLabel(" "));

        add(LabelHolderPanel, BorderLayout.WEST);

        // Make and add a Jpanel for the random number button

        JPanel buttonHolderPanel= new JPanel(new BorderLayout());
        randNumButton= new JButton("Random Numbers");
        buttonHolderPanel.add(randNumButton, BorderLayout.NORTH);
        add(buttonHolderPanel, BorderLayout.EAST);
    }
}

class DataPanel extends JPanel implements TableModelListener
{
    DataPanel()
    {
        setLayout(new BorderLayout());

        dataTable = new JTable(data, colNames)
        {
            public TableCellRenderer getCellRenderer(int row,
                                                      int column)
            {
                return new MyTableCellRenderer();
            }
        };
        dataTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        dataTable.setRowHeight(dataTable.getRowHeight());
        int rtWidth= ((numberOfJobs+1)*51) +5;
        int rtHeight= ((numberOfProc+1)*17) +5;
        dataTable.setPreferredScrollableViewportSize(
            new Dimension(rtWidth, rtHeight));

        TableColumn column = null;
        for (int w = 0; w < numberOfJobs+1; w++)
        {
            column = dataTable.getColumnModel().getColumn(w);
            column.setPreferredWidth(50);
        }

        column = dataTable.getColumnModel().getColumn(0);
        // Set table header renderer with SDK v1.2 or v1.3
```

```
            column.setHeaderRenderer(new TableCellRenderer(){
                public Component getTableCellRendererComponent(
                    JTable table, Object value, boolean isSelcted,
                    boolean hasFocus, int row ,int column)
                {
                    JPanel jp= new JPanel();
                    return jp;
                }
                });
            column= null;

            //Create the scroll pane and add the table to it.
            JScrollPane scrollPane = new JScrollPane(dataTable);
            scrollPane.setRowHeaderView(new ProcPanel());

            add(scrollPane, BorderLayout.CENTER);

            dataTable.getModel().addTableModelListener(DataPanel.this);
        }

        public void tableChanged(TableModelEvent e)
        {
            dataHasChanged= true;
        }
    }

    class ProcPanel extends JPanel
    {
        JLabel temp = null;
        ProcPanel()
        {
            JPanel innerPanel= new JPanel(
                new GridLayout(numberOfProc+1,1));
            temp= new JLabel(" ");
            temp.setBorder(BorderFactory.createEmptyBorder());
            innerPanel.add(temp);
            for(int i=1; i<numberOfProc+1; i++)
            {
                 temp= new JLabel(" Proc " + (i) + " ");
                 temp.setBorder(BorderFactory.createEmptyBorder());
                 innerPanel.add(temp);
            }
            setLayout(new BorderLayout());
            add(innerPanel, BorderLayout.NORTH);
        }
    }

    class MultipleButtonListener implements ActionListener
    {
        public void actionPerformed(ActionEvent ae)
        {
            if(dataTable.isEditing())
            {
                TableCellEditor tce= dataTable.getCellEditor(
                    dataTable.getEditingRow(),
                    dataTable.getEditingColumn());
                tce.stopCellEditing();
```

```
}

String temp= ae.getActionCommand();
if(temp.equals("Save As "))
{
    if(allFieldsAreValid())
    {
        parentCF.inputFC.setSelectedFile(new File(
            parentCF.inputFC.getCurrentDirectory(), " "));

        int returnVal = parentCF.inputFC.showSaveDialog(
            InputFileViewer.this);

        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            File file = parentCF.inputFC.getSelectedFile();
            if(file != null)
            {
                String filename = file.getName();
                int i = filename.lastIndexOf('.');
                if(i>0 && i<filename.length()-1)
                {
                    // an extension exists
                    writeToFile(file);
                    setTitle(file.getName());
                }
                else
                {
                    // an extension does not exist
                    File newFile= new File(
                        file.getPath()+".txt");
                    writeToFile(newFile);
                    parentCF.inputFC.setSelectedFile(
                        newFile);
                    setTitle(newFile.getName());
                    file.delete();
                }
            }

            dataHasChanged= false;
            parentCF.inputFC.rescanCurrentDirectory();
            named= true;
        }
    }
    else
    {
        JOptionPane.showMessageDialog(InputFileViewer.this,
            "Incomplete data entry.\nAn integer or float
            must be entered\nin each colored field",
           "Message", JOptionPane.ERROR_MESSAGE);
    }
}
else if(temp.equals("  Save  "))
{
    if(allFieldsAreValid())
    {
        if(named == true)
```

```
{
    //default icon, custom title
    int n = JOptionPane.showConfirmDialog(
        InputFileViewer.this, "The file "+
        getTitle() +" already exists.\n"
        + "Do you want to over-write the file?\n"
        + "If not, select No and then Save As\n",
        "Over-write file?",
        JOptionPane.YES_NO_OPTION);

    if(n == JOptionPane.YES_OPTION)
    {
        writeToFile(readFile);
        dataHasChanged= false;
        parentCF.inputFC.rescanCurrentDirectory();
    }
}
else
{
    // named= false

    parentCF.inputFC.setSelectedFile(new File(
        parentCF.inputFC.getCurrentDirectory(),
        " "));

    int returnVal=parentCF.inputFC.showSaveDialog(
        InputFileViewer.this);

    if (returnVal == JFileChooser.APPROVE_OPTION)
    {
        File file =
            parentCF.inputFC.getSelectedFile();
        if(file != null)
        {
            String filename = file.getName();
            int i = filename.lastIndexOf('.');
            if(i>0 && i<filename.length()-1)
            {
                // an extension exists
                writeToFile(file);
                setTitle(file.getName());
            }
            else
            {
                // an extension does not exist
                File newFile= new File(
                    file.getPath()+".txt");
                writeToFile(newFile);
                parentCF.inputFC.setSelectedFile(
                    newFile);
                setTitle(newFile.getName());
                file.delete();
            }
        }

        dataHasChanged= false;
        parentCF.inputFC.rescanCurrentDirectory();
```

```
                    named= true;
                }
            }
        }
        else
        {
            JOptionPane.showMessageDialog(InputFileViewer.this,
                "Incomplete data entry.\nAn integer or float
                must be entered\nin each colored field",
                "Message", JOptionPane.ERROR_MESSAGE);
        }
    }
    else if(temp.equals(" Close "))
    {
        if(dataHasChanged)
        {
            int n = JOptionPane.showConfirmDialog(
                InputFileViewer.this,
                "You have made changes that have not yet\n" +
                "been saved. Do you want to continue closing?",
                "Continue Closing?",
                JOptionPane.OK_CANCEL_OPTION);

            if(n == JOptionPane.CANCEL_OPTION)
            {
                /* do nothing */
            }
            else
            {
                dispose();
            }
        }
        else
        {
            dispose();
        }
    }
    else if(temp.equals("Random Numbers"))
    {
        // generate random numbers for jobs, between 1 and 100
        for(int i=1; i< numberOfJobs+1; i++)
        {
            dataTable.setValueAt(Integer.toString(
                generateRandomNumber(100)), 0, i);
        }

        // generate random numbers for initial delays,
        // between 1 and 50
        for(int i=1; i<numberOfProc+1; i++)
        {
            dataTable.setValueAt(Integer.toString(
                generateRandomNumber(50)), i, 0);
        }

        // generate random numbers for capacities,
        // between 1 and 10
        for(int i=1; i<numberOfProc+1; i++)
```

```
                {
                    for(int j=1; j< numberOfJobs+1; j++)
                    {
                        dataTable.setValueAt(Integer.toString(
                            generateRandomNumber(10)), i, j);
                    }
                }
            }
        }
    }

    class MyTableCellRenderer implements TableCellRenderer
    {
        public Component getTableCellRendererComponent(JTable table,
                Object value, boolean isSelcted, boolean hasFocus,
                int row ,int column)
        {
            JTextArea jta= new JTextArea();

            if(column == 0 && row > 0)
            {
                jta.append(value.toString());
                jta.setBackground(new Color(InDelayColor));
            }
            else if(column > 0 && row == 0)
            {
                jta.append(value.toString());
                jta.setBackground(new Color(InJobSizeColor));
            }
            else if(column > 0 && row > 0)
            {
                jta.append(value.toString());
                jta.setBackground(new Color(InCapColor));
            }
            else
            {
                jta.setBackground(
                    InputFileViewer.this.getBackground());
            }

            return jta;
        }
    }

    void loadColorProperties()
    {
        Properties p= new Properties();
        try
        {
            p.load(new FileInputStream("gui"+File.separator+
                "preferences.txt"));
            if( p.getProperty("InJobSizeColor") != null)
            {
                InJobSizeColor= Integer.parseInt(p.getProperty(
                    "InJobSizeColor"));
            }
            else
            {
                InJobSizeColor = (Color.blue).getRGB();
            }
            if( p.getProperty("InDelayColor") != null)
            {
                InDelayColor= Integer.parseInt(p.getProperty(
                    "InDelayColor"));
            }
            else
            {
                InDelayColor = (Color.green).getRGB();
            }
            if( p.getProperty("InCapColor") != null)
            {
                InCapColor= Integer.parseInt(p.getProperty(
                    "InCapColor"));
            }
            else
            {
                InCapColor = (Color.orange).getRGB();
            }
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(InputFileViewer.this,
                "Error occured while loading color preferences.\n" +
                "The file \"gui\\preferences.txt\" could not be
                found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }

    void storeColorProperties()
    {
        Properties p= new Properties();
        try
        {
            p.load(new FileInputStream("gui"+File.separator+
                "preferences.txt"));
            p.setProperty("InJobSizeColor",
                Integer.toString(InJobSizeColor));
            p.setProperty("InDelayColor",
                Integer.toString(InDelayColor));
            p.setProperty("InCapColor", Integer.toString(InCapColor));
            p.store(new FileOutputStream("gui"+File.separator+
                "preferences.txt"), "Preferences");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(InputFileViewer.this,
                "Error occured while storing color preferences.\n" +
                "The file \"gui\\preferences.txt\" could not be
                found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }
    int generateRandomNumber(int max)
    {
        int returnNumber= 0;
        while(returnNumber < 1)
        {
            returnNumber= Math.round((float)(Math.random()*max));
        }
        return returnNumber;
    }
}
```

B.7 LoadBalancer.java

```
// LoadBalancer.java

// This class simply creates a ControlFrame. Instantiation of this
// class starts the GUI program.

package gui;

import java.io.*;

public class LoadBalancer
{
    public static void main(String args[])
    {
        ControlFrame cf= new ControlFrame();
    }
}
```

B.8 MyFileFilter.java

```
// MyFileFilter.java

// This class is a custom FileFilter used by JfileChoosers to select
// input and output files.

package gui;

import javax.swing.filechooser.FileFilter;
import java.util.Hashtable;
import java.util.Enumeration;
import java.io.*;
import java.util.*;


public class MyFileFilter extends FileFilter
{

    private Hashtable filters = null;
    private String description = null;
    private String fullDescription = null;
    private boolean useExtensionsInDescription = true;
```

```
public MyFileFilter()
{
    this.filters = new Hashtable();
}

public MyFileFilter(String extension)
{
    this(extension,null);
}

public MyFileFilter(String extension, String description)
{
    this();
    if(extension!=null) addExtension(extension);
    if(description!=null) setDescription(description);
}

public boolean accept(File f)
{
    if(f != null)
    {
        if(f.isDirectory())
        {
            return true;
        }
        String extension = getExtension(f);
        if(extension != null &&
            filters.get(getExtension(f)) != null)
        {
            return true;
        }
    }
    return false;
}


public String getExtension(File f)
{
    if(f != null)
    {
        String filename = f.getName();
        int i = filename.lastIndexOf('.');
        if(i>0 && i<filename.length()-1)
        {
            return filename.substring(i+1).toLowerCase();
        }
    }
    return null;
}

public void addExtension(String extension)
{
    if(filters == null)
    {
        filters = new Hashtable(5);
    }
```

```
        filters.put(extension.toLowerCase(), this);
        fullDescription = null;
    }



    public String getDescription()
    {
        if(fullDescription == null)
        {
            if(description == null || isExtensionListInDescription())
            {
                fullDescription =
                    description==null ? "(" : description + " (";
                Enumeration extensions = filters.keys();
                if(extensions != null)
                {
                    fullDescription +=
                        "." + (String) extensions.nextElement();
                    while (extensions.hasMoreElements())
                    {
                        fullDescription +=
                            ", " + (String) extensions.nextElement();
                    }
                }
                fullDescription += ")";
            } else
            {
                fullDescription = description;
            }
        }
        return fullDescription;
    }


    public void setDescription(String description)
    {
        this.description = description;
        fullDescription = null;
    }


    public void setExtensionListInDescription(boolean b)
    {
        useExtensionsInDescription = b;
        fullDescription = null;
    }


    public boolean isExtensionListInDescription()
    {
        return useExtensionsInDescription;
    }
}
```

B.9 OutputFileViewer.java

```
// OutputFileViewer.java

// This class allows users to view and save output files containing
// the results of load balancing.

package gui;


import javax.swing.JTable;
import javax.swing.JScrollPane;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.BorderFactory;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JFileChooser;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableCellEditor;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableColumn;
import javax.swing.JTextArea;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.SwingUtilities;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;
import javax.swing.WindowConstants;
import javax.swing.JColorChooser;


public class OutputFileViewer extends JFrame
{
    boolean named;

    int numberOfJobs;
    int numberOfProc;

    JTable resultTable;
    JPanel rowHeaderPanel;
    ResultPanel rp;
    JPanel buttonPanel;

    ThreeStringHolder[][] resultData;

    String[] jobColNames;

    boolean showCapacities;

    JButton showCapButton;
JButton saveButton;

JLabel numLinSolveLabel;
JLabel milliSecLabel;
Dimension showCapButtonDim;

ControlFrame parentControlFrame;
File readFile= null;

JLabel ldl;
JLabel delayValueLabel[];

int OutJobSizeColor;
int OutDelayColor;
int OutCapColor;
int OutTimeSpanColor;
int OutLoadColor;
int ImplCapColor;

JLabel jobSizeLabel;
JLabel delayLabel;
JLabel capLabel;
JLabel loadLabel;
JLabel timeSpanLabel;



public OutputFileViewer(File file, ControlFrame parentCF)
{
    super(file.getName());
    parentControlFrame= parentCF;
    readFile= file;

    loadColorProperties();

    showCapacities= true;

    if(readFile.getName().equals("temp.txt"))
    {
        named= false;
    }
    else
    {
        named = true;
    }

    try
    {
        BufferedReader br = new BufferedReader(
            new FileReader(readFile));

        String dataString= br.readLine();
        StringTokenizer st= new StringTokenizer(dataString);

        try
        {
        st.nextToken();  // remove "output"
st.nextToken();  // remove "file"
numberOfProc= Integer.parseInt(st.nextToken());
numberOfJobs= Integer.parseInt(st.nextToken());

// Make String[] to be used as the table header
jobColNames= new String[numberOfJobs+1]; // +1 for times
for(int i=0; i<numberOfJobs; i++)
{
    jobColNames[i]= "Job "+ (i+1) +";" +st.nextToken()+";";
}
jobColNames[numberOfJobs]= "Time;Span;";
// end get table header data

// *** Start of code to build rowHeaderPanel ***
// Build the panel with the delays
JPanel delayPanel= new JPanel(new BorderLayout());
JPanel innerPanel= new JPanel(new GridLayout(0,1));

delayValueLabel= new JLabel[numberOfProc];
for(int i=0; i<numberOfProc; i++)
{
    innerPanel.add(new JLabel(" "));
    delayValueLabel[i]= new JLabel(
        "  " + st.nextToken() + "     ");
    delayValueLabel[i].setForeground(
        new Color(OutDelayColor));
    innerPanel.add(delayValueLabel[i]);
}
delayPanel.add(innerPanel, BorderLayout.NORTH);
// end panel with delays

// Build a panel to hold delayPanel
rowHeaderPanel= new JPanel(new GridLayout(1,0));
rowHeaderPanel.add(delayPanel);
// *** End of code to build rowHeaderPanel ***


// get capacity and load data
resultData=new ThreeStringHolder
    [numberOfProc][numberOfJobs + 1]; // +1 for times
for(int i=0; i<numberOfProc; i++)
{
    for(int j=0; j< numberOfJobs; j++)
    {
        resultData[i][j]= new ThreeStringHolder();
        resultData[i][j].setFirstString(st.nextToken());
        resultData[i][j].setSecondString(st.nextToken());
        resultData[i][j].setThirdString("");
    }
}
// end capacity and load data

// get times data
for(int i=0; i<numberOfProc; i++)
{
    resultData[i][numberOfJobs]= new ThreeStringHolder();
    resultData[i][numberOfJobs].setFirstString(
                st.nextToken());
            resultData[i][numberOfJobs].setSecondString("");
            resultData[i][numberOfJobs].setThirdString("");
        }
        // end of times data

        // get other data
        numLinSolveLabel= new JLabel(
            "Number of Linear Systems Solved: " + st.nextToken());
        milliSecLabel= new JLabel(
            "Balancing Time: " + st.nextToken() + " milliseconds");
        numLinSolveLabel.setFont(
            numLinSolveLabel.getFont().deriveFont(18.0f));
        numLinSolveLabel.setFont(
            numLinSolveLabel.getFont().deriveFont(Font.ITALIC));
        milliSecLabel.setFont(
            milliSecLabel.getFont().deriveFont(18.0f));
        milliSecLabel.setFont(
            milliSecLabel.getFont().deriveFont(Font.ITALIC));
        // end get other data

        makeTheFrame();
        setLocation(50, 75);
        pack();
        setVisible(true);

    }
    catch(NoSuchElementException nsee)
    {
        //custom title, error icon
        JOptionPane.showMessageDialog(OutputFileViewer.this,
            "NSEException caught in OutputFileVeiwer constructor ",
            "Message", JOptionPane.ERROR_MESSAGE);
    }
    br.close();
    }
    catch(IOException ioe)
    {
        //custom title, error icon
        JOptionPane.showMessageDialog(OutputFileViewer.this,
            "IOException caught in OutputFileVeiwer constructor ",
            "Message", JOptionPane.ERROR_MESSAGE);
    }
    catch(Error e)
    {
        //custom title, error icon
        JOptionPane.showMessageDialog(OutputFileViewer.this,
            "Error caught in OutputFileVeiwer constructor ",
            "Message", JOptionPane.ERROR_MESSAGE);
    }
}

void makeTheFrame()
{
    /************** Start of menu code *********************/
    // Create the help menu components for setting preferences
    JMenuBar menuBar;
```

```
JMenu menu, subMenu;
JMenuItem subMenuItem1, subMenuItem2, subMenuItem3,
    subMenuItem4, subMenuItem5;

// Create the menu bar.
menuBar = new JMenuBar();
setJMenuBar(menuBar);

// Build the first menu.
menu = new JMenu("Help");
menu.setMnemonic(KeyEvent.VK_H);
menu.getAccessibleContext().setAccessibleDescription(
        "Help");
menuBar.add(menu);

// build submenu
subMenu = new JMenu("Set Colors");
menu.add(subMenu);

// subMenuItems
subMenuItem1= new JMenuItem("Set Job Size Color");
subMenu.add(subMenuItem1);
subMenuItem2= new JMenuItem("Set Initial Delay Color");
subMenu.add(subMenuItem2);
subMenuItem3= new JMenuItem("Set Load Color");
subMenu.add(subMenuItem3);
subMenuItem4= new JMenuItem("Set Time Span Color");
subMenu.add(subMenuItem4);
subMenuItem5= new JMenuItem("Set Capacity Color");
subMenu.add(subMenuItem5);

subMenuItem1.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        Color newColor= JColorChooser.showDialog(
            OutputFileViewer.this,
            "Choose Job Size Color",
            new Color(OutJobSizeColor));
        if(newColor != null)
        {
            OutJobSizeColor= newColor.getRGB();
            jobSizeLabel.setForeground(newColor);
            repaint();
            storeColorProperties();
        }
    }});

subMenuItem2.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent evt)
    {
        Color newColor= JColorChooser.showDialog(
            OutputFileViewer.this,
            "Choose Initial Delay Color",
            new Color(OutDelayColor));
        if(newColor != null)
        {
            OutDelayColor= newColor.getRGB();
```

```
                delayLabel.setForeground(newColor);
                for(int i=0; i<numberOfProc; i++)
                {
                    delayValueLabel[i].setForeground(newColor);
                }
                repaint();
                storeColorProperties();
            }
        }});

    subMenuItem3.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent evt)
        {
            Color newColor= JColorChooser.showDialog(
                OutputFileViewer.this,
                "Choose Load Color",
                new Color(OutLoadColor));
            if(newColor != null)
            {
                OutLoadColor= newColor.getRGB();
                loadLabel.setForeground(newColor);
                ldl.setForeground(newColor);
                repaint();
                storeColorProperties();
            }
        }});

    subMenuItem4.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent evt)
        {
            Color newColor= JColorChooser.showDialog(
                OutputFileViewer.this,
                "Choose Time Span Color",
                new Color(OutTimeSpanColor));
            if(newColor != null)
            {
                OutTimeSpanColor= newColor.getRGB();
                timeSpanLabel.setForeground(newColor);
                repaint();
                storeColorProperties();
            }

        }});

    subMenuItem5.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent evt)
        {
            Color newColor= JColorChooser.showDialog(
                OutputFileViewer.this,
                "Choose Capacity Color",
                new Color(OutCapColor));
            if(newColor != null)
            {
                OutCapColor= newColor.getRGB();
                capLabel.setForeground(newColor);
                repaint();
                storeColorProperties();
        }
    }});

/******************** End of menu code ******************/

ColorLegendPanel lcp = new ColorLegendPanel();

getContentPane().add(lcp, BorderLayout.NORTH);
getContentPane().add(rp=new ResultPanel(),
    BorderLayout.CENTER);

// This code makes the buttons and the panels that hold them
showCapButton= new JButton("Dont Show Cap");
showCapButtonDim= new Dimension(showCapButton.getSize());
saveButton= new JButton("  Save  ");
JButton closeButton= new JButton(" Close ");
MultipleButtonListener MBL= new MultipleButtonListener();
showCapButton.addActionListener(MBL);
saveButton.addActionListener(MBL);
closeButton.addActionListener(MBL);
buttonPanel= new JPanel(new GridLayout(1, 4));
buttonPanel.add(showCapButton);
if(named == false)
{
    buttonPanel.add(saveButton);
}
buttonPanel.add(closeButton);
JPanel jp2= new JPanel(new BorderLayout());
jp2.add(buttonPanel, BorderLayout.EAST);

getContentPane().add(jp2, BorderLayout.SOUTH);

addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e)
    {
        if(named == false)
        {
            int n = JOptionPane.showConfirmDialog(
                OutputFileViewer.this,
                "Just a reminder: \n" +
                "This output file has not yet been saved.\n" +
                "Click yes if you want to save it.
                Otherwise,\n" + "click No.\n",
                "Save this Output file?",
                JOptionPane.YES_NO_OPTION);

            if(n == JOptionPane.YES_OPTION)
            {
                String suggestedName= new String(
                   parentControlFrame.currInputFile.getName());
                StringTokenizer st= new
                    StringTokenizer(suggestedName, ".");
                try
                {
                    suggestedName= st.nextToken() + ".out";
                }
                catch(NoSuchElementException nsee)
                {
                }
                String suggestedDir= ParentControlFrame
                    .outputFC.getCurrentDirectory().getPath();
                String suggestedPath= suggestedDir
                    +File.separator+ suggestedName;
                File suggestedFile= new File(suggestedPath);
                parentControlFrame.outputFC.setSelectedFile(
                    suggestedFile);

                int returnVal =
                    parentControlFrame.outputFC.showSaveDialog(
                    OutputFileViewer.this);

                if (returnVal == JFileChooser.APPROVE_OPTION)
                {
                    File file = ParentControlFrame
                        .outputFC.getSelectedFile();
                    if(file.exists())
                    {
                        file.delete();
                    }

                    String filename = file.getName();
                    int i = filename.lastIndexOf('.');
                    if(i>0 && i<filename.length()-1)
                    {
                        // an extension exists
                        readFile.renameTo(file);
                        setTitle(readFile.getName());
                    }
                    else
                    {
                        // an extension does not exist
                        File newFile= new File(
                            file.getPath()+".out");
                        if(newFile.exists())
                        {
                            newFile.delete();
                        }
                        readFile.renameTo(newFile);
                        parentControlFrame.outputFC
                            .setSelectedFile(newFile);
                        setTitle(readFile.getName());
                    }

                    parentControlFrame.outputFC
                        .rescanCurrentDirectory();
                    buttonPanel.remove(saveButton);
                    named= true;
                    validate();
                }
            }
        }
        if( (new File("temp.txt")).exists() )
        {
            new File("temp.txt").delete();
            }
            dispose();
        }
        });
}

class ColorLegendPanel extends JPanel
{
    ColorLegendPanel()
    {
        setLayout(new BorderLayout());

        JPanel LabelHolderPanel= new JPanel(new GridLayout(0,1));

        jobSizeLabel= new JLabel(" Job Size ");
        jobSizeLabel.setForeground(new Color(OutJobSizeColor));
        jobSizeLabel.setFont(
            jobSizeLabel.getFont().deriveFont(14.0f));
        delayLabel= new JLabel(" Initial Delays ");
        delayLabel.setForeground(new Color(OutDelayColor));
        delayLabel.setFont(delayLabel.getFont().deriveFont(14.0f));
        capLabel= new JLabel(" Capacities ");
        capLabel.setForeground(new Color(OutCapColor));
        capLabel.setFont(capLabel.getFont().deriveFont(14.0f));
        loadLabel= new JLabel(" Load ");
        loadLabel.setForeground(new Color(OutLoadColor));
        loadLabel.setFont(loadLabel.getFont().deriveFont(14.0f));
        timeSpanLabel= new JLabel(" Time Span ");
        timeSpanLabel.setForeground(new Color(OutTimeSpanColor));
        timeSpanLabel.setFont(
            loadLabel.getFont().deriveFont(14.0f));

        LabelHolderPanel.add(new JLabel(" "));
        LabelHolderPanel.add(jobSizeLabel);
        LabelHolderPanel.add(delayLabel);
        LabelHolderPanel.add(loadLabel);
        LabelHolderPanel.add(timeSpanLabel);
        LabelHolderPanel.add(capLabel);

        add(LabelHolderPanel, BorderLayout.WEST);
    }
}

class ResultPanel extends JPanel
{
    ResultPanel()
    {
        setLayout(new BorderLayout());

        ldl= new JLabel("Load Distribution", JLabel.CENTER);
        ldl.setForeground(new Color(OutLoadColor));
        ldl.setFont(ldl.getFont().deriveFont(18.0f));
        ldl.setFont(ldl.getFont().deriveFont(Font.ITALIC));

        JPanel jp1= new JPanel(new GridLayout(0,1));
        jp1.add(numLinSolveLabel);
        jp1.add(milliSecLabel);
        MyTableModel myModel= new MyTableModel();
        resultTable = new JTable(myModel);
        resultTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        resultTable.setRowHeight(
            (resultTable.getRowHeight() * 2)+1);
        resultTable.setGridColor(this.getBackground());

        int rtWidth= ((numberOfJobs+1)*76) + 0;
        int rtHeight= ((numberOfProc)*34) + 0;
        resultTable.setPreferredScrollableViewportSize(
            new Dimension(rtWidth, rtHeight));

        TableColumn column = null;
        for (int w = 0; w < numberOfJobs +1; w++)
        {
            column = resultTable.getColumnModel().getColumn(w);
            column.setCellRenderer(
                new TextAreaTableCellRenderer());
            column.setMinWidth(70);
            // Set table header renderer with SDK v1.2 or v1.3
            column.setHeaderRenderer(new MyHeaderRenderer());
        }

        // Set table header renderer with SDK v1.3 only
        //resultTable.getTableHeader().setDefaultRenderer(
            new MyHeaderRenderer());

        //Create the scroll pane and add the table to it.
        JScrollPane resultScrollPane =
            new JScrollPane(resultTable);
        resultScrollPane.setRowHeaderView(rowHeaderPanel);

        add(ldl, BorderLayout.NORTH);
        add(resultScrollPane, BorderLayout.CENTER);
        add(jp1, BorderLayout.SOUTH);
    }
}

class MultipleButtonListener implements ActionListener
{
    public void actionPerformed(ActionEvent ae)
    {
        String temp= ae.getActionCommand();
        if(temp.equals("Dont Show Cap"))
        {
            showCapacities= !showCapacities;
            showCapButton.setText("    Show Cap      ");
            showCapButton.setSize(showCapButtonDim);
            capLabel.setVisible(false);
            repaint();
        }
        else if(temp.equals("    Show Cap      "))
        {
            showCapacities= !showCapacities;
            showCapButton.setText("Dont Show Cap");
            showCapButton.setSize(showCapButtonDim);
    capLabel.setVisible(true);
    repaint();
}
else if(temp.equals("  Save  "))
{
    if(named == false)
    {
        String suggestedName= new String(
            ParentControlFrame
                .currInputFile.getName());
        StringTokenizer st= new StringTokenizer(
            suggestedName, ".");
        try
        {
            suggestedName= st.nextToken() + ".out";
        }
        catch(NoSuchElementException nsee)
        {
        }
        String suggestedDir= ParentControlFrame
            .outputFC.getCurrentDirectory().getPath();
        String suggestedPath=
            suggestedDir +File.separator+ suggestedName;
        File suggestedFile= new File(suggestedPath);
        parentControlFrame.outputFC.setSelectedFile(
            suggestedFile);

        int returnVal = parentControlFrame.outputFC
            .showSaveDialog(OutputFileViewer.this);

        if (returnVal == JFileChooser.APPROVE_OPTION)
        {
            File file = parentControlFrame
                .outputFC.getSelectedFile();
            if(file.exists())
            {
                file.delete();
            }

            String filename = file.getName();
            int i = filename.lastIndexOf('.');
            if(i>0 && i<filename.length()-1)
            {
                // an extension exists
                readFile.renameTo(file);
                setTitle(readFile.getName());
            }
            else
            {
                // an extension does not exist
                File newFile= new File(
                    file.getPath()+".out");
                if(newFile.exists())
                {
                    newFile.delete();
                }
                readFile.renameTo(newFile);
```

```
                    parentControlFrame.outputFC
                        .setSelectedFile(newFile);
                    setTitle(newFile.getName());
                }

                parentControlFrame.outputFC
                    .rescanCurrentDirectory();
                buttonPanel.remove(saveButton);
                named= true;
                validate();
            }
            else
            {
                //Save command cancelled by user.
            }
        }
    }
    else if(temp.equals(" Close "))
    {
        if(named == false)
        {
            int n = JOptionPane.showConfirmDialog(
                OutputFileViewer.this,
                "Just a reminder: \n" +
                "This output file has not yet been saved.\n" +
                "Click yes if you want to save it.
                Otherwise,\n" + "click No.\n",
                "Save this Output file?",
                JOptionPane.YES_NO_OPTION);

            if(n == JOptionPane.YES_OPTION)
            {
                String suggestedName= new String(
                   parentControlFrame.currInputFile.getName());
                StringTokenizer st=
                    new StringTokenizer(suggestedName, ".");
                try
                {
                    suggestedName= st.nextToken() + ".out";
                }
                catch(NoSuchElementException nsee)
                {
                }
                String suggestedDir= ParentControlFrame
                    .outputFC.getCurrentDirectory().getPath();
                String suggestedPath= suggestedDir
                    +File.separator+ suggestedName;
                File suggestedFile= new File(suggestedPath);
                parentControlFrame.outputFC.setSelectedFile(
                    suggestedFile);

                int returnVal = parentControlFrame.outputFC
                    .showSaveDialog(OutputFileViewer.this);

                if (returnVal == JFileChooser.APPROVE_OPTION)
                {
                    File file = parentControlFrame
```

```
                        .outputFC.getSelectedFile();
                    if(file.exists())
                    {
                        file.delete();
                    }

                    String filename = file.getName();
                    int i = filename.lastIndexOf('.');
                    if(i>0 && i<filename.length()-1)
                    {
                        // an extension exists
                        readFile.renameTo(file);
                        setTitle(readFile.getName());
                    }
                    else
                    {
                        // an extension does not exist
                        File newFile= new File(
                            file.getPath()+".out");
                        if(newFile.exists())
                        {
                            newFile.delete();
                        }
                        readFile.renameTo(newFile);
                        parentControlFrame.outputFC
                            .setSelectedFile(newFile);
                        setTitle(readFile.getName());
                    }

                    parentControlFrame.outputFC
                        .rescanCurrentDirectory();
                    buttonPanel.remove(saveButton);
                    named= true;
                    validate();
                }
            }
            else
            {
                // NO_OPTION. ReadFile is not named so it
                // is equal to temp.txt. Delete temp.txt.
                readFile.delete();
            }
        }
        if( (new File("temp.txt")).exists() )
        {
            new File("temp.txt").delete();
        }
        dispose();
    }
  }
}

class TextAreaTableCellRenderer implements TableCellRenderer
{
    public Component getTableCellRendererComponent(
        JTable table, Object value, boolean isSelcted,
        boolean hasFocus, int row ,int column)
        {
            JPanel jp = new JPanel(new GridLayout(0,1));
            JTextArea jta1= new JTextArea();
            JTextArea jta2= new JTextArea();

            jp.add(jta1);
            jp.add(jta2);

            if(column < numberOfJobs)
            {
                if(showCapacities)
                {
                    jta1.append(
                        ((ThreeStringHolder)value).getFirstString());
                    jta1.setForeground(new Color(OutCapColor));
                }
                String temp=
                    ((ThreeStringHolder)value).getSecondString();
                if( !(temp.equals("0")) )
                {
                    jta1.append(" ");
                    jta2.setForeground(new Color(OutLoadColor));
                    jta2.append(temp);
                }
            }
            else
            {
                jta1.append(" ");
                jta2.setForeground(new Color(OutTimeSpanColor));
                jta2.append(
                    ((ThreeStringHolder)value).getFirstString());
            }
            return jp;
        }
    }

    class MyHeaderRenderer implements TableCellRenderer
    {
        public Component getTableCellRendererComponent(JTable table,
                Object value, boolean isSelcted, boolean hasFocus,
                int row ,int column)
        {
            JPanel jp= new JPanel(new GridLayout(2,1));
            JLabel jl1= new JLabel();
            JLabel jl2= new JLabel();
            try
            {
                StringTokenizer st= new StringTokenizer(
                    (String)value, ";");
                jl1.setText(st.nextToken());
                jl2.setText(st.nextToken());
            }
            catch(NoSuchElementException e)
            {
            }
            jp.add(jl1);
            jp.add(jl2);
        if(column == numberOfJobs)
        {
            jl1.setForeground(new Color(OutTimeSpanColor));
            jl2.setForeground(new Color(OutTimeSpanColor));
        }
        else
        {
            jl1.setForeground(new Color(OutJobSizeColor));
            jl2.setForeground(new Color(OutJobSizeColor));
        }

        return jp;
    }
}


class ThreeStringHolder
{
    String firstString;
    String secondString;
    String thirdString;

    public void setFirstString(String s)
    {
        firstString= s;
    }
    public void setSecondString(String s)
    {
        secondString= s;
    }
    public void setThirdString(String s)
    {
        thirdString= s;
    }
    public String getFirstString()
    {
        return firstString;
    }
    public String getSecondString()
    {
        return secondString;
    }
    public String getThirdString()
    {
        return thirdString;
    }
    public String toString()
    {
        // Default when a renderer is not specified
        // Placed in a JTextFeild by default
        return "Default";
    }
}

class MyTableModel extends AbstractTableModel
{
    public int getColumnCount() {
            return jobColNames.length;
        }

        public int getRowCount() {
            return resultData.length;
        }

        public String getColumnName(int col) {
            return jobColNames[col];
        }

        public Object getValueAt(int row, int col) {
            return resultData[row][col];
        }

        public Class getColumnClass(int c) {
            return getValueAt(0, c).getClass();
        }

        public boolean isCellEditable(int row, int col)
        {
            return false;
        }
    }

    void loadColorProperties()
    {
        Properties p= new Properties();
        try
        {
            p.load(new FileInputStream(
                "gui"+File.separator+"preferences.txt"));
            if( p.getProperty("OutJobSizeColor") != null)
            {
                OutJobSizeColor= Integer.parseInt(p.getProperty(
                    "OutJobSizeColor"));
            }
            else
            {
                OutJobSizeColor = (Color.blue).getRGB();
            }
            if( p.getProperty("OutDelayColor") != null)
            {
                OutDelayColor= Integer.parseInt(p.getProperty(
                    "OutDelayColor"));
            }
            else
            {
                OutDelayColor = (Color.green).getRGB();
            }
            if( p.getProperty("OutCapColor") != null)
            {
                OutCapColor= Integer.parseInt(p.getProperty(
                    "OutCapColor"));
            }
            else
            {
                OutCapColor = (Color.orange).getRGB();
            }
            if( p.getProperty("OutTimeSpanColor") != null)
            {
                OutTimeSpanColor= Integer.parseInt(p.getProperty(
                    "OutTimeSpanColor"));
            }
            else
            {
                OutTimeSpanColor = (Color.blue).getRGB();
            }
            if( p.getProperty("OutLoadColor") != null)
            {
                OutLoadColor= Integer.parseInt(p.getProperty(
                    "OutLoadColor"));
            }
            else
            {
                OutLoadColor = (Color.red).getRGB();
            }

        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(OutputFileViewer.this,
                "Error occured while loading color preferences.\n" +
                "The file \"gui\\preferences.txt\" could not be
                found.", "Error Message",
                JOptionPane.ERROR_MESSAGE);
        }
    }

    void storeColorProperties()
    {
        Properties p= new Properties();
        try
        {
            p.load(new FileInputStream(
                "gui"+File.separator+"preferences.txt"));
            p.setProperty("OutJobSizeColor",
                Integer.toString(OutJobSizeColor));
            p.setProperty("OutDelayColor",
                Integer.toString(OutDelayColor));
            p.setProperty("OutCapColor",
                Integer.toString(OutCapColor));
            p.setProperty("OutLoadColor",
                Integer.toString(OutLoadColor));
            p.setProperty("OutTimeSpanColor",
                Integer.toString(OutTimeSpanColor));
            p.store(new FileOutputStream("gui"+File.separator+
                "preferences.txt"), "Preferences");
        }
        catch(Exception e)
        {
            JOptionPane.showMessageDialog(OutputFileViewer.this,
                "Error occured while storing color preferences.\n" +
                "The file \"gui\\preferences.txt\" could not be
```

```
                    found.", "Error Message",
                    JOptionPane.ERROR_MESSAGE);
            }
        }
}
```

B.10 Preferences.txt

This text file contains the users preferred colors for table cells and home directories where input and output files are stored as selected in the program.

```
// preferences.txt

#Preferences
#Tue Dec 04 16:33:21 EST 2001
OutCapColor=-13434880
ImplCapColor=-3355648
OutDelayColor=-6750157
InJobSizeColor=-16750849
OutLoadColor=-65536
OutTimeSpanColor=-16738048
inputHomeDir=C\:\\project\\arraybased\\notoptimal\\4proc4jobs\\input
InCapColor=-26368
InDelayColor=-13369549
outputHomeDir=C\:\\project\\arraybased\\notoptimal\\4proc4jobs\\output
OutJobSizeColor=-16776961
```

APPENDIX C

COMPILING AND BUILDING THE SYSTEM

C.1 Compiling the Java GUI

The following steps can be used to create the graphical user interface (GUI) for the load balancing system. Sun's Java Development Kit version 1.2.2 (JDK 1.2.2) and later can be used.

1). Create the required directories and packages.

A). Create a directory named "LBsystem".

B). Create a subdirectory in "LBsystem" named "gui".

C).It is suggested but not required that you create sub directories in "LBsystem" named "input" and "output" for storing input and output files.

2). Place the files in the proper directories.

A). Place the following files in the directory (Java package) named "gui":

LoadBalancer.java, ControlFrame.java, MyFileFilter.java, BalancerWrapper.java, DialogEvent.java, DialogListener.java, DialogFrame.java, InputFileViewer.java, OutputFileViewer.java and preferences.txt.

3). Compile the GUI.

A). Get a command line prompt (MS-DOS prompt). Change the current working directory to "LBsystem".

B). To compile the package "gui", issue the command without the quotation marks:

```
"javac gui/*.java"
```

C.2 Compiling loadbalancer.dll

The following describes the exact steps that can be followed to create the dll using Microsoft Visual C++ version 6.0. If another compiler is used, follow that compiler's instructions.

1). Create a new dll workspace.

A). Open Microsoft Visual C++ version 6.0

B). From the menu, select "File" then select "New".

C). Under the "Projects" menu, select "Win32 Dynamic-Link Library" and enter "loadbalancer" under "Project name". Select "Create new workspace" checkbox. Then click "OK" button.

D). When prompted "What kind of DLL would you like to create", select "An empty DLL project" checkbox. Click the "Finish" button.

E). When informed "An empty DLL project will be created for you", click the "OK" button. Visual C++ should now have created a directory named "loadbalancer" for you.

2) Place the files in the proper directories so they can be found by the compiler.

A). Place the following files in directory "loadbalancer":

StdAfx.h, BalancerWrapper.h, and loadbalancer.cpp

B). In the directory "loadbalancer", create a subdirectory named "headers" for header files. Place the following files in the directory named "headers":

data_arrays.h, eq_system.h, exceptions.h, from_to_node.h, graph.h

C). In the directory "loadbalancer", create a subdirectory named "source" for source files. Place the following files in the directory named "source":

data_arrays.cpp, eq_system.cpp, graph.cpp

3). Add the files to the project.

A). If Microsoft Visual C++ version 6.0. is still open, this step, 3).A, can be skipped. If it was closed open Visual C++ and from the menu select "File". Then select "Recent Workspaces". Then select *pathname*\loadbalancer.

B). In the pull down menu in Visual C++, select "Project" then select "Add To Project". Finally, select "Files".

C). In the pop up widow, select files BalancerWrapper.h, loadbalancer.cpp, and StdAfx.h. Click the "OK" button.

D). In the pull down menu in Visual C++, select "Project" then select "Add To Project". Finally, select "Files".

E). In the pop up widow, open (double click) the directory "headers". Select all the files in that directory and then click the "OK" button.

D). In the pull down menu in Visual C++, select "Project" then select "Add To Project". Finally, select "Files".

E). In the pop up widow, open (double click) the directory "source". Select all the files in that directory and then click the "OK" button.

4). Compile the dll.

A). In the pull down menu in Visual C++, select "Build" then select "Build loadbalancer.dll".

C.3 Building the System

In the subdirectory “Debug” in the directory “loadbalancer”, copy the file “loadbalancer.dll” to the directory “LBsystem”. The system is now operational.

APPENDIX D

RUNNING THE PROGRAM

D.1 Starting the Java GUI

Get a command line prompt (MS-DOS prompt). Change the current working directory to "LBsystem". To start the program, issue the case-sensitive command without the quotation marks:

"Java gui/LoadBalancer"

A ControlFrame window titled "Load Balancer" should now be visible.

D.2 Setting Preferred Directories (optional)

To set the directories where the program looks for input and output files as default, select "Help". Then select "Set Preferences" and then select "Set input file directory". Select, but do not open, the directory where input files will be stored. It is suggested that the directory "input" is selected. Then click the "Set input dir." button. Repeat the similar steps to set the directory where output file are stored.

D.3 Creating a New Data File

In the window named "Load Balancer", select the "Create New Data File" button. In the window that appears named "Enter Data", enter the number of processors and jobs. Then select the "OK" button. A window named "Not Yet Named" should appear. In this table, enter data manually by selecting and editing each and every box in the table or select "Random Numbers" to generate random data automatically. Select the "Save" button and enter the name you choose for your input file. Then select the "Save" button. Finally, select the "Close" button.

D.4 Editing a Data File

In the "Load Balancer" window, select the "View and/or Edit a Data File" button. Choose the name of the data file you want to edit and select the "Open" button. Select the desired table entry to be edited by double clicking on the table cell. Edit the data entry and then select the "Save" or "Save As" button following standard GUI convention.

D.5 Running the Load Balancing Algorithm on a Data File

In the "Load Balancer" window, select the "Run Balancer" button. Choose the name of the data file you want to use and then select the "Run" button. When a status message appears, select the "OK" button. A window named "temp.txt" will appear showing the results of load balancing. It is optional to save this file. If the "Save" button is selected, notice that it is automatically suggested by the program to save the output file as the same name as the input file. The file extensions differentiate input and output files. Of course any name can be chosen. Select the "Save" button. Finally, select the "Close" button.

D.6 Viewing an Output File

In the "Load Balancer" window, select the "View an Output File" button. Choose the name of the output file you want to view and select the "Open" button. Select the "Close" button when finished viewing.

D.7 Finding Average Run Time

In the “Load Balancer” window, select the “Get Average Run Time” button. Enter the number of processors and jobs and select the “get time” button. A window will appear with the balancing time of 6 experiments and the average time for the 6 experiments. Select the“X” button (close) in the upper right-hand corner when finished viewing

D.8 Terminating the Program

In the “Load Balancer” window, select the “Exit” button. The “X” button can also be selected with the same result

.

GLOSSARY OF TERMS

Backlog: The amount of time it will take processor k to finish any work assigned to it prior to the start of load balancing algorithm. Often referred to b_k in this document. Also called Initial delays.

Balancing Time: The time that CPU (central processing unit) requires to distribute the jobs. This will not include the time required to read from or write to the text files.

Cycle: A graph comprised of all the redistribution variables such that at least one load on every active processor is increased or decreased. Also called a redistribution cycle. See section 2.2.

Initial Delay: Also called backlog. See "backlog".

Input Text File: A text file containing the number of processors, the number of jobs, Processor Capacities, Initial Delays and the Size of Jobs to be balanced. It is read as input to the load balancing algorithm which is implemented in "loadbalacerdll.dll".

Job Completion Time: The time required for processor k to finish the work allocated to it by the Load Balancing Program. Often written in this document as t_k.

Load: The portion of job j that is assigned processor k. Often written in this document as x_{jk}.

Load Balancing Program: The executable program that will determine the Load Distribution.

Load Distribution: The set of all partial amounts of each job that is assigned to each processor.

LP: Linear Program

Output text File: The file containing the results of the Load Balancing Program that is created by "loadBalancerdll.dll" and is read for display by the GUI.

Processor Capacity: The amount of time required by a processor to complete 1 unit of the job.

Redistribution Cycle: Also called "Cycle". See "Cycle".

Redistribution Variable: The amount of a job that is to be removed from or added to a load. See section 2.2.

Size of Job: The total number of job units of a given job that will be distributed by the Load Balancing Program. Often written in this document as a_{jk}.

REFERENCES

[1] T.L. Casavant and J.G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems", *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141-154, Feb. 1988.

[2] M. Nuttall and M. Sloman, "Workload characteristics for process migration and load balancing", *Proceedings of the 17th International Conference on Distributed Computing Systems*, pp. 133-140, 1997.

[3] P. Kreuger and N. G. Shivarati. "Adaptive location policies for global scheduling", *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 432-444, June 1994.

[4] P.K.K Loh. Wen Jing Hsu, Cai Wentong, and N. Sriskanthan. "How network topology affects dynamic loading balancing", *IEEE Parallel and Distributed Technology: Systems and Applications*, vol. 4, no. 3, pp. 25-35, Fall 1996.

[5] S. Selvakumar and C.S.R. Murthy, "Static task allocation for heterogeneous distributed computing systems", *TENCON '91.1991 IEEE Region 10 International Conference on EC3-Energy, Computer, Communication and Control Systems*, vol. 3, pp. 190 –193, 1991.

[6] W.W. Chu, L.T. Holloway, M.T. Lan, and K. Efe, "Task allocation in distributed data processing," *IEEE Computer*, vol. 13, no. 11, pp. 57-69, November 1980.

[7] C.M. Woodside and G. G. Monforton, "Fast allocation of processes in distributed and parallel systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 164-174, Feb. 1993.

[8] C.-I.H Chen and V. Cherkassky, "Task allocation and reallocation for fault tolerance in multicomputer systems", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 30, no. 4, pp. 1094-1104, Oct.1994.

[9] Y. Zhang, H. Kameda, and S.-L Hung, "Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems", *IEEE Proceedings on Computers and Digital Techniques*, vol. 144, no. 2, pp. 100-106, March 1997.

[10] Y. Zhang, K. Hakozaki, H. Kameda, and K. Shimizu, "Comparison of dynamic and static load-balancing strategies in heterogeneous distributed systems", *Proceedings of the 28th Annual Simulation Symposium*, 1995, pp. 332-340, 1995.

[11] Y.Zhang, H. Kameda, and K. Shimizu, "A comparison of adaptive and static load balancing strategies by using simulation methods", *SICICI '92. Proceedings of the Singapore International Conference on Intelligent Control and Instrumentation*, vol. 2, pp. 1162-1167, 1992.

[12] U. Hofmann and M. Krajewski, "Quasi-static load balancing in local area networks", *Proceedings of the 20th Conference on Local Computer Networks,* pp. 254-263, 1995.

[13] R.D. Dietz, T.L. Casavant, T.E. Scheetz, T.A. Braun, and M.S. Andersland, "Modeling the impact of run-time uncertainty on optimal computation scheduling using feedback", Proceedings of the 1997 International Conference on Parallel Processing, pp. 481 –488, 1997.

[14] Boon Ping Gan; Yoke Hean Low; Jain, S.; Turner, S.J.; Wentong Cai; Wen Jing Hsu; and Shell Ying Huang, "Load balancing for conservative simulation on shared memory multiprocessor systems", *PADS 2000. Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation*, pp. 139-146, 2000.

[15] E. Luque, A. Ripoll, A. Cortes, and T. Margalef, "A distributed diffusion method for dynamic load balancing on parallel computers", *Proceedings. Euromicro Workshop on Parallel and Distributed Processing*, pp. 43-50, 1995.

[16] Hwa-Chun Lin and C.S. Raghavendra, "A dynamic load-balancing policy with a central job dispatcher (LBC)", *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 148-158, February 1992.

[17] F. Bonomi and A. Kumar, "Adaptive optimal load balancing in a nonhomogeneous multiserver system with a central job scheduler", *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1232-1250, Oct. 1990.

[18] T.T.Y Suen and J.S.K. Wong, "Efficient task migration algorithm for distributed systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 4, July 1992, pp. 488-499.

[19] V.M. Lo, "Heuristic algorithms for task assignment in distributed systems", *IEEE Transactions on Computers*, vol. 37, no. 11, Nov. 1988, pp. 1384 –1397.

[20] H. Stone, "Multiprocessor scheduling with the aid of network flow algorithms", *IEEE Transactions on Software Engineering"*, Jan. 1977, pp. 85-93.

[21] J.M. Kumar, L.M. Patnaik, and A. Das, "Load balancing algorithms for an extended hypercube", *IEEE Proceedings on Computers and Digital Techniques*, vol. 141, no. 5, Sept. 1994, pp. 298-306.

[22] L. Borzemski, "Load balancing in parallel and distributed processing of tree-based multiple-task jobs", *Proceedings of the Euromicro Workshop on Parallel and Distributed Processing*, 1995, pp. 98 –105.

[23] A.A. Khan, C.L. McCreary, and M.S. Jones, "A comparison of multiprocessor scheduling heuristics", *Proceedings of the International Conference on Parallel Processing*, 1994.

[24] A. Radulescu, and A.J.C. van Gemund, "Fast and effective task scheduling in heterogeneous systems", *Proceedings. 9th Heterogeneous Computing Workshop*, 2000, pp. 229 –238.

[25] J.-J. Hwang, Y.-C. Chow, F.D. Anger, and C.-Y. Lee, "Scheduling precedence graphs in systems with inter-processor communication times, *SIAM Journal of Computing*, vol. 18, pp. 244-257, April 1989.

[26] G.-L. Park, B. Shirazi, J. Marquis, and H. Choo, "Decisive path scheduling: a new list scheduling method", *Proceedings of the International Conference on Parallel Processing*, 1997.

[27] A. Radulescu and A.J.C. van Gemund, "FLB: fast load balancing for distributed-memory machines", *Proceedings of the International Conference on Parallel Processing*, 1999.

[28] A. Radulescu and A.J.C. van Gemund, "On the complexity of list scheduling algorithms for distributed-memory systems", *Proceedings of the ACM International Conference on Supercomputing*, 1999.

[29] M.-Y. Wu and D.D. Gajski, "Hypertool: a programming aid for message-passing systems", *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 7, pp. 330-343, July 1990.

[30] E.G. Coffman Jr., M.R. Garey, and D.S. Johnson, "An application of bin-packing to multiprocessor scheduling", *SIAM Journal of Computing*, vol. 7, pp. 1-17, February 1978.

[31] I. Ahmad, M.K. Dhodhi, and A. Ghafoor, "Task assignment in distributed computing systems", *Proceedings of the 1995 IEEE Fourteenth Annual International Phoenix Conference on Computers and Communications*, 1995, pp. 49 –53.

[32] T. Chockalingham and S. Arankumar, "A randomized heuristic for the mapping problem: the genetic approach", *Parallel Computing*, vol. 18, pp. 1157-1165, 1992.

[33] L. Wang, H.G. Siegel, V.P. Roychowdhury, and A.A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach", *Journal of Parallel and distributed Computing*, vol. 47, pp. 8-22, 1997.

[34] A. Mehta, "Experiments with client/sever load balancing algorithm", Masters Project, New Jersey Institute of Technology, Spring 2001 (Professor B. Verkhovsky, advisor).

[35] Min-You Wu, Wei Shu, and H. Zhang, "Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems", *Proceedings. 9th Heterogeneous Computing Workshop*, 2000, pp. 375–385.

[36] T.D. Braun, H.J. Siegal, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, Bin Yao, D. Hensgen, and R.F. Freund, "A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems", *Proceedings. Eighth Heterogeneous Computing Workshop*, 1999, pp. 15 –29.

[37] A.K. Sarje and G. Sagar, "Heuristic model for task allocation in distributed computer systems", *IEEE Proceedings Computers and Digital Techniques*, vol. 138, no. 5, Sept. 1991, pp. 313 –318.

[38] B. Verkhovsky, "Clients-servers load balancing", manuscript, New Jersey Institute of Technology, Spring 2001.

[39] R. Leland and B. Hendrickson, "An empirical study of static load balancing algorithms", *Proceedings of the Scalable High-Performance Computing Conference*, 1994, pp. 682 –685.

[40] R.L. Graham, "Bounds on multiprocessing timing anomalies", *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416-429, March 1969.

[41] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.J. Rinnooykan, "Optimization in deterministic sequencing and scheduling: a survey", *Annals of Discrete Mathematics*, vol. 5, pp. 287-326, 1979.

[42] P.Y.R. Ma, E.Y.S. Lee, and M. Tsuchiya, "A task allocation model for distributed computing systems", *IEEE Transactions*, C-31, pp. 41-47, 1982.

[43] M.S.Chern, G.H. Chen, and P. Liu, "An LC branch and bound algorithm for the module assignment problem", *Information Processing Letters*, vol. 32, pp. 61-71, July 1989.

[44] R. Correa and A. Ferreira, "On the effectiveness of synchronous parallel branch and bound algorithms", *Parallel Processing Letters*, vol. 5, no. 3, pp. 375-386, 1995.

[45] R.M. Karp and Y. Zhang, "Randomized parallel algorithms for backtrack search and branch and bound computation", *Journal of the ACM*, vol. 40, no. 3, pp. 765-789, July 1993.

[46] W.H. Kohler and K. Steiglitz, "Characterization and theoretical comparison of branch and bound algorithms for permutation problems", *Journal of the ACM*, vol. 21, pp. 140-156, January 1974.

[47] V. Kumar, K. Ramesh, and V.N. Rao, "Parallel best-first search of state-space graphs: a summary of results", *Proceedings of the Seventh International Conference on Artificial Intelligence*, vol. 1, pp. 122-127, August 1988.

[48] N.R. Mahapatra and S. Dutt, "Scalable global and local hashing strategies for duplicate pruning in parallel A* graph search", *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 7, pp. 738-756, July 1997.

[49] D.R. Smith, "Random trees and the analysis of branch and bound procedures", *Journal of the ACM*, vol. 31, no. 1, pp. 163-188, January 1984.

[50] P.C. Chang and Y.S. Jiang, "A state-space search approach for parallel processor scheduling problems with arbitrary precedence relations", *European Journal of Operations Research*, vol. 77, pp. 208-223, 1994.

[51] G.-H. Chen and J.-S. Yu, "A branch and bound with underestimates algorithm for the task assignment problem with precedence constraint", *Proceedings of the International Conference for Distributed Computing Systems*, pp. 494-501, 1990.

[52] H.-C. Chou and C.-P. Chung, "Optimal microprocessor task scheduling using dominance and equivalence relations", *Computers Operations Research*, vol. 21, no. 4, pp. 463-475, 1994.

[53] S.H. Hosseini, B. Litow, M. Malkawi, J. McPherson, and K. Vairavan, "System theory modeling and performance analysis of a distributed load balancing algorithm", *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, 1989, vol. 1, 1990, pp. 648 -652.

[54] D. Towsley, "Allocating programs containing branches and loops within a multiple processor system", *IEEE Transactions on Software Engineering*, vol. 12, pp. 1018-1034, 1986.

[55] C.C. Price and U.W. Pooch, "Search technique for non-linear multiprocessor scheduling problem", *Naval Research Logistics Quarterly,* vol. 29, pp. 213-233, 1982.

[56] S.H. Bokhari, "A shortest tree algorithm for optimal assignment across space and time in a distributed computer system", *IEEE Transactions on Software Engineering*, vol. 7, pp. 583-589, 1981.

[57] Jie Li and H. Kameda, "A decomposition algorithm for optimal static load balancing in tree hierarchy network configurations", *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, May 1994, pp. 540 –548.

[58] I. Ahmad and Yu-Kwong Kwok, "Optimal and near-optimal allocation of precedence-constrained tasks to parallel processors: defying the high complexity using effective search techniques", *Proceedings. 1998 International Conference on Parallel Processing*, 1998, pp. 424 –431.

[59] Xueyan Tang and S.T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers", *Proceedings. 2000 International Conference on Parallel Processing*, 2000, pp. 373 –382.

[60] B. Verkhovsky, "Algorithm for optimal multi-job time-cost load balancing", keynote address, *The Int'l Symp. on System Integration, InterSymp.* 2001, Baden-Baden, Germany, July-Aug.2001.

[61] S.C. Dafermos and F.T. Sparrow, "The traffic assignment problem for a general network", *J. Res. National Bureau of Standards-B*, vol. 73B, no. 2, pp. 91-118, 1969.

[62] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: an approach to store-and-forward communication network design", *Networks*, vol. 3, pp. 97-133, 1973.

[63] C. Kim and H. Kameda, "Optimal static load balancing of multi-class jobs in a distributed computer system", *Proceedings of the 10th International Conference on Distributed Computing Systems*, 1990, pp. 562 –569.

[64] A.N. Tantawi and D. Towsley, "A general model for optimal static load balancing in star network configurations", *Performance '84*, E. Gelenbe, Ed., Elsevier Science (North-Holland), pp. 277-291, 1984.

[65] Boris S. Verkhovsky, "Numerical methods for solution of linear programming problems", *Econ. and Math. Methods*, 5, 1965.

[66] B. Verkhovsky, Lecture Notes, Fall Semester, 2000.