A task-oriented priority queue for telephone switch design

by

Randall George Martens

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Mechanical and Industrial Engineering
University of Manitoba
Winnipeg, Manitoba

ISBN 0-612-13342-7

Canada

Name _____

Dissertation Abstracts International and *Masters Abstracts International* are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

Operations Research
SUBJECT TERM

0796
SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology ... 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology .. 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics ... 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Nom _____

Dissertation Abstracts International est organisé en catégories de sujets. Veuillez s.v.p. choisir le sujet qui décrit le mieux votre thèse et inscrivez le code numérique approprié dans l'espace réservé ci-dessous.

_____ ☐☐☐ U·M·I

SUJET — CODE DE SUJET

Catégories par sujets

HUMANITÉS ET SCIENCES SOCIALES

COMMUNICATIONS ET LES ARTS
Architecture 0729
Beaux-arts 0357
Bibliothéconomie 0399
Cinéma 0900
Communication verbale 0459
Communications 0708
Danse 0378
Histoire de l'art 0377
Journalisme 0391
Musique 0413
Sciences de l'information 0723
Théâtre 0465

ÉDUCATION
Généralités 515
Administration 0514
Art 0273
Collèges communautaires 0275
Commerce 0688
Économie domestique 0278
Éducation permanente 0516
Éducation préscolaire 0518
Éducation sanitaire 0680
Enseignement agricole 0517
Enseignement bilingue et multiculturel 0282
Enseignement industriel 0521
Enseignement primaire. 0524
Enseignement professionnel 0747
Enseignement religieux 0527
Enseignement secondaire 0533
Enseignement spécial 0529
Enseignement supérieur 0745
Évaluation 0288
Finances 0277
Formation des enseignants 0530
Histoire de l'éducation 0520
Langues et littérature 0279
Lecture 0535
Mathématiques 0280
Musique 0522
Orientation et consultation 0519
Philosophie de l'éducation 0998
Physique 0523
Programmes d'études et enseignement 0727
Psychologie 0525
Sciences 0714
Sciences sociales 0534
Sociologie de l'éducation 0340
Technologie 0710

LANGUE, LITTÉRATURE ET LINGUISTIQUE
Langues
- Généralités 0679
- Anciennes 0289
- Linguistique 0290
- Modernes 0291

Littérature
- Généralités 0401
- Anciennes 0294
- Comparée 0295
- Médiévale 0297
- Moderne 0298
- Africaine 0316
- Américaine 0591
- Anglaise 0593
- Asiatique 0305
- Canadienne (Anglaise) 0352
- Canadienne (Française) 0355
- Germanique 0311
- Latino-américaine 0312
- Moyen-orientale 0315
- Romane 0313
- Slave et est-européenne 0314

PHILOSOPHIE, RELIGION ET THÉOLOGIE
Philosophie 0422
Religion
- Généralités 0318
- Clergé 0319
- Études bibliques 0321
- Histoire des religions 0320
- Philosophie de la religion 0322

Théologie 0469

SCIENCES SOCIALES
Anthropologie
- Archéologie 0324
- Culturelle 0326
- Physique 0327

Droit 0398
Économie
- Généralités 0501
- Commerce-Affaires 0505
- Économie agricole 0503
- Économie du travail 0510
- Finances 0508
- Histoire 0509
- Théorie 0511

Études américaines 0323
Études canadiennes 0385
Études féministes 0453
Folklore 0358
Géographie 0366
Gérontologie 0351
Gestion des affaires
- Généralités 0310
- Administration 0454
- Banques 0770
- Comptabilité 0272
- Marketing 0338

Histoire
- Histoire générale 0578
- Ancienne 0579
- Médiévale 0581
- Moderne 0582
- Histoire des noirs 0328
- Africaine 0331
- Canadienne 0334
- États-Unis 0337
- Européenne 0335
- Moyen-orientale 0333
- Latino-américaine 0336
- Asie, Australie et Océanie 0332

Histoire des sciences 0585
Loisirs 0814
Planification urbaine et régionale 0999
Science politique
- Généralités 0615
- Administration publique 0617
- Droit et relations internationales 0616

Sociologie
- Généralités 0626
- Aide et bien-être social 0630
- Criminologie et établissements pénitentiaires 0627
- Démographie 0938
- Études de l' individu et de la famille 0628
- Études des relations interethniques et des relations raciales 0631
- Structure et développement social 0700
- Théorie et méthodes. 0344
- Travail et relations industrielles 0629

Transports 0709
Travail social 0452

SCIENCES ET INGÉNIERIE

SCIENCES BIOLOGIQUES
Agriculture
- Généralités 0473
- Agronomie. 0285
- Alimentation et technologie alimentaire 0359
- Culture 0479
- Élevage et alimentation 0475
- Exploitation des pâturages 0777
- Pathologie animale 0476
- Pathologie végétale 0480
- Physiologie végétale 0817
- Sylviculture et faune 0478
- Technologie du bois 0746

Biologie
- Généralités 0306
- Anatomie 0287
- Biologie (Statistiques) 0308
- Biologie moléculaire 0307
- Botanique 0309
- Cellule 0379
- Écologie 0329
- Entomologie 0353
- Génétique 0369
- Limnologie 0793
- Microbiologie 0410
- Neurologie 0317
- Océanographie 0416
- Physiologie 0433
- Radiation 0821
- Science vétérinaire 0778
- Zoologie 0472

Biophysique
- Généralités 0786
- Medicale 0760

SCIENCES DE LA TERRE
Biogéochimie 0425
Géochimie 0996
Géodésie 0370
Géographie physique 0368
Géologie 0372
Géophysique 0373
Hydrologie 0388
Minéralogie 0411
Océanographie physique 0415
Paléobotanique 0345
Paléoécologie 0426
Paléontologie 0418
Paléozoologie 0985
Palynologie 0427

SCIENCES DE LA SANTÉ ET DE L'ENVIRONNEMENT
Économie domestique 0386
Sciences de l'environnement 0768
Sciences de la santé
- Généralités 0566
- Administration des hôpitaux 0769
- Alimentation et nutrition 0570
- Audiologie 0300
- Chimiothérapie 0992
- Dentisterie 0567
- Développement humain 0758
- Enseignement 0350
- Immunologie 0982
- Loisirs 0575
- Médecine du travail et thérapie 0354
- Médecine et chirurgie 0564
- Obstétrique et gynécologie 0380
- Ophtalmologie 0381
- Orthophonie 0460
- Pathologie 0571
- Pharmacie 0572
- Pharmacologie 0419
- Physiothérapie 0382
- Radiologie 0574
- Santé mentale 0347
- Santé publique 0573
- Soins infirmiers 0569
- Toxicologie 0383

SCIENCES PHYSIQUES
Sciences Pures
Chimie
- Généralités 0485
- Biochimie 487
- Chimie agricole 0749
- Chimie analytique 0486
- Chimie minérale 0488
- Chimie nucléaire 0738
- Chimie organique 0490
- Chimie pharmaceutique 0491
- Physique 0494
- PolymÈres 0495
- Radiation 0754

Mathématiques 0405
Physique
- Généralités 0605
- Acoustique 0986
- Astronomie et astrophysique 0606
- Électronique et électricité 0607
- Fluides et plasma 0759
- Météorologie 0608
- Optique 0752
- Particules (Physique nucléaire) 0798
- Physique atomique 0748
- Physique de l'état solide 0611
- Physique moléculaire 0609
- Physique nucléaire 0610
- Radiation 0756

Statistiques 0463

Sciences Appliqués Et Technologie
Informatique 0984
Ingénierie
- Généralités 0537
- Agricole 0539
- Automobile 0540
- Biomédicale 0541
- Chaleur et thermodynamique 0348
- Conditionnement (Emballage) 0549
- Génie aérospatial 0538
- Génie chimique 0542
- Génie civil 0543
- Génie électronique et électrique 0544
- Génie industriel 0546
- Génie mécanique 0548
- Génie nucléaire 0552
- Ingénierie des systèmes 0790
- Mécanique navale 0547
- Métallurgie 0743
- Science des matériaux 0794
- Technique du pétrole 0765
- Technique minière 0551
- Techniques sanitaires et municipales 0554
- Technologie hydraulique 0545

Mécanique appliquée 0346
Géotechnologie 0428
Matières plastiques (Technologie) 0795
Recherche opérationnelle 0796
Textiles et tissus (Technologie) 0794

PSYCHOLOGIE
Généralités 0621
Personnalité 0625
Psychobiologie 0349
Psychologie clinique 0622
Psychologie du comportement 0384
Psychologie du développement 0620
Psychologie expérimentale 0623
Psychologie industrielle 0624
Psychologie physiologique 0989
Psychologie sociale 0451
Psychométrie 0632

A TASK-ORIENTED PRIORITY QUEUE FOR TELEPHONE SWITCH DESIGN

BY

RANDALL GEORGE MARTENS

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

Acknowledgements

The author would like to express his gratitude for the guidance and help he has received from Dr. Attahiru Sule Alfa over the course of this thesis. The author would also like to thank the members of his examining committee, with special thanks to Dr. W. Kinsner for his suggestions and comments. The author would like to acknowledge the financial assistance the University of Manitoba has provided via a graduate fellowship. Financial assistance was also made through a grant from Bell-Northern Research, Ottawa. Special thanks to Marvin Krym of Bell-Northern Research for his suggestions and comments. Lastly, the author wishes to express his appreciation for the help he received from his parents.

Contents

List of Figures

List of Tables

Abstract

In the development of a telephone switch, it is necessary to assess the performance of its operating system. Performance analysis can be done using the model developed in this thesis. The model gives the expected sojourn times for the various tasks associated with call processing. The model itself is a single server priority queueing model, where the call processing is broken up into ordered sets of prioritized tasks. The arrival process is Poisson, and the service time distribution is general (independent of the arrival process). The queueing discipline is a modified first-come first-served (FCFS) service discipline with preemption (preemptive resume). Initially, this modification allowed tasks to jump to the head of the queue if a similar task was receiving service when it arrived. When this was found to affect sojourn times, the modification was extended to allow tasks to jump and join similar tasks regardless of where they are in the queue and whether or not these tasks were being served. It was found that those tasks with larger service times and higher arrival rates, jumped ahead more frequently and subsequently enjoyed a reduction in time spent waiting for service. The sojourn times given by the model compared favourably with times found by simulation. The model was further extended to allow tasks to fork into multiples of other task types upon completion of service. Both the modified FCFS service discipline and forking were needed to model the actual behaviour of tasks.

Finally, the system design problem was formulated as a combinatorial optimization problem by assigning costs to the waiting times of the tasks. The optimization problem was solved using the simulated annealing method. The best solutions were found within 3 to 20 minutes of execution time, requiring 300 temperatures and an epoch of 50. A small value for the maximum allowable increase in the objective function at termination was found to be best (0.02 to 0.05 percent of the objective function). Lastly, the values for the two parameters responsible for searching the neighbourhood structure respond differently to different problems, and the best values must be found by experimentation.

Chapter 1

Introduction

1.1 Problem Description

In the development of a telephone switch, it is useful to know how long it will take to process the various tasks associated with call processing. It is necessary to group the call processing functions into tasks, and to assign each task a priority, since the functions vary considerably in their urgency for execution. A model that gives the sojourn times for the tasks is very useful, because it characterizes processing delay and will indicate whether or not a particular system design meets quality of service standards. An example of a quality of service standard is dial tone delay. This delay must not exceed a certain amount. Simulation can be used to obtain sojourn times, but it is very time consuming.

A brief description of the general steps in establishing (and ending) a telephone call further details call processing. The first step is the detection of an incoming call (an unhooked receiver). The digits are then received

and translated to determine the terminating terminal (call destination). A route for the call is then found (if a clear line exists) and the speech path is set up. The next step is to apply ringing. When an answer has been detected, a call has been established. It may be necessary to record the time for billing purposes. When the call is finished, a disconnection is detected and the speech path is taken down. It may again be necessary to record the time. The whole process of establishing and ending a telephone call requires the execution of many different functions. It is these functions (portions of system software) that form the tasks.

The processing required to establish and disconnect a single call will be referred to as a job. Not every job that a switch processes, will require the execution of exactly the same functions. The result is a number of job types or classes. Just how a job is broken up into tasks, and what priorities these tasks are given, greatly influences sojourn times. The system must be designed to give preferential treatment to more important tasks without seriously delaying others. The number of functions, possible priority assignments, and job types creates a very large number of possible system designs. For example, a single job with ten functions generates approximately 5.5 billion different system configurations. To illustrate how functions can be grouped, consider four functions 1, 2, 3, and 4 (given in the order in which they must be executed). When grouping the functions into tasks, the order of execution of the functions must be maintained. There are three ways of grouping these

functions into two tasks: 1 234, 12 34, and 123 4 (functions in a task sharing a common underline). Other ways of grouping these functions would involve one, three, or four tasks. When you add to the number of task structures, the fact that each task structure can have many priority assignments, the number of system designs becomes very large. Adding multiple job classes makes the problem even larger. Later (in Chapter 4), a heuristic is presented that searches the problem space for good designs.

System designs are evaluated on the basis of delay characteristics. Two important measurements of delay are job transit time and start of service delay. Job transit time is the total amount of time a job spends in the queueing system. Start of service delay is the amount of time a job waits before initial processing begins. Start of processing delay is very important to certain kinds of jobs. One of these kinds of jobs is a message that flows through a packet switch [15]. Some of these messages have contention protocols, requiring that an acknowledgement be returned to the message source within a certain period of time. If an acknowledgement is not returned in time, the message source times out and retries the transmission. It is for this reason, that the first task of these jobs are usually given a high priority. There is another type of task which should be given a high priority. This is a task which has a greater effect on the user of the system. Sometimes the delay experienced by the user of the system, depends only on the delay a particular task has and not the entire job. There are tasks which should be given a low

priority. In a packet switch there are tasks (like statistics gathering) which can be done in what is called a background mode. A message may depart the system long before tasks being performed in a background mode on its behalf are complete. Sometimes several grades of service for a particular job are offered, resulting in different job classes. These job classes do not differ with respect to task structure, but differ only in priority assignment. Jobs with higher grades of service will have higher priority assignments. In this case, the designer would most likely assign priorities which would achieve a reasonable balance in the service given to the various classes. For example, experimental results for a two class problem found in the literature [16] show that it is possible to considerably decrease the start of service delay experienced by one class, while increasing only slightly the message transit time of both classes.

1.2 Solution Method: The Analytical Model

The switch can be modeled as a single server queueing system, since the call processing is done on a single processor. The arrival process of the jobs is assumed to follow the Poisson process, with all of the tasks in a job arriving with that job. This assumption permits the use of the PASTA property (Poisson Arrivals See Time Averages). This is an important property in forming the equations for the sojourn time. A different arrival process would make the problem much more difficult to represent mathematically. In addition to

this, the Poisson process does closely follow the actual arrival process. Many people in industry and other researchers in this area, use this same arrival process. The model considers multiple job types or classes. A general service time distribution is allowed for each of the tasks in a job (independent of the arrival process). This is not a limitation: any service time distribution can be used. The server tends to the tasks in a preemptive resume fashion. In preemptive priority systems, an arriving high priority task interrupts the service of a lower priority task and obtains service for itself. With preemptive resume, service of the lower priority task is resumed from the interruption point. Preemption is necessary, because of the urgency for execution some of the tasks have.

The model can further be described by the use of an example to illustrate the operation of the queueing system. The example will involve two classes of jobs. The first job type will have two tasks, with the first task having a priority of one and the second a priority of two. The second job type will also have two tasks, the first having a priority of one and the second a priority of three (see Figures 1.1 and 1.2). The convention concerning priority that will be used in this work, is that the task with the lower number has the higher priority. For example, a priority of one (the lowest number) is higher than a priority of two. The service system consists of a collection of priority queues (one for each distinct priority). This example will create three queues; one for tasks with a priority of one, one for tasks which have a priority of two,

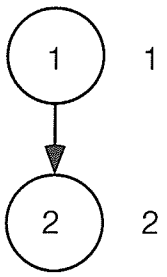

Figure 1.1: First Job Class

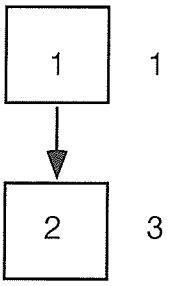

Figure 1.2: Second Job Class

and one for tasks that have a priority of three. The first stage in Figure 1.3 shows these queues at a point in time when the system is empty. The server (which is idle) is denoted by a triangle.

When a job arrives, its first task is given ready-for-service status and joins the appropriate priority queue, where it will wait its turn for service. The second stage shows the priority queues, after a job of the first type has arrived and its first task has been given ready-for-service status and has joined the priority one queue. Tasks from the first job class will be denoted by a circle. Even though the second task of this job is in the queueing system, it will not join a priority queue until it is given ready-for-service status. This, it will not receive until the first task has completed service. The tasks within a job must be serviced in order. As this one task receives service, another job of the first type arrives, followed by a job of the second type. The third stage shows the queues after the first task's of these jobs have received ready-for-service status and joined the queue. Tasks from the second job class will

be denoted by a square. A short time later, another job of the first type arrives. When its first task joins the queue, it will position itself behind the two tasks of its type. This means jumping ahead of a task which has arrived before it. This modification to the first-come first-served service discipline is needed to portray actual software execution of the operating system within the switch. A task will only jump to join a similar task (or group of similar tasks). Similar tasks are tasks which not only come from the same job type, but they must come from the same part of that job. If, when a task joins a priority queue, there are no similar tasks, it will position itself at the end of the queue. This applies to all the priority queues. The fourth stage shows the queues after this last task has positioned itself behind its group.

Once the first task of a job has completed service, the job's second task is given ready-for-service status and joins the priority queues. The fifth stage shows the queues right after the first task to receive service has been completed. The priority two task will have to wait until the priority one queue is empty before it will receive service, since the server will always tend to the nonempty queue of the highest priority. Low priority tasks will eventually receive service, providing the total traffic intensity is less than one. It may happen that the first task in an arriving job has a higher priority than the task currently being serviced. When this happens, the task being serviced is preempted and remains in a preempted state until work on all higher priority tasks has been completed. Service on the preempted task is resumed from

the point of interruption. It is possible, to have a number of tasks (one from each queue) in a preempted state simultaneously.

The first tasks for the second and last job to arrive to the queueing system will then receive service, resulting in the priority queues being as depicted in the sixth stage. When the first task of the job of the second type completes service, the job's second task will join the priority three queue, and the server will move over to the priority two queue (see the seventh stage). When service to the second task of the first job (to arrive) has been completed, the job is finished. The eighth stage shows the queues right after this has occurred.

The approach used to obtain the mean waiting times is presented in the next chapter. It is similar to that used by Daigle [16] for a strictly first-come first-served (FCFS) priority queueing model with preemption and multiple job types. This type of queueing model will later be compared to the model developed in this work. Earlier, Daigle and Houstis [13] solved a problem similar to [16] for a single job type. Simon [42] extended the model of [13] to include a range of preemption (from non-preemptive to pure preemptive) with IPF (Interrupted Processes First) strategy, forking, and bulk arrivals. After [16], Daigle [17] solved the multiple job type problem for non-preemption. Paterok [38] extended the model of [16] to include a range of preemption with STRICT strategy, forking, and bulk arrivals. Burakowski [4] also solved the multiple job type problem for a range of preemption with IPF strategy. An alternative approach is to divide the job into time slices

(instead of tasks), and to give the job a priority and allow it to change. Schrage and Ruschitzka have done some work in this area, to name just a couple. Schrage [41] used a round-robin queueing system, where jobs are given a certain amount of uninterrupted service and then placed into a lower priority queue if more service is needed. In this work, a job's priority is repeatedly lowered, creating many priority queues. Ruschitzka [40] uses a processor-sharing system, where a job's priority is defined by the difference between the time it has been in the system and an arbitrary function of its attained service. None of these models are able to mimic the jumping of tasks that occurs in the priority queues. The contribution of this work is the modification of the FCFS service discipline for a priority queueing model with preemption, multiple job types, and forking.

The organization of the remainder of the thesis is as follows. In the first section of Chapter 2, the effect of jumping is examined by developing a queueing model with a simplified modification to the FCFS service discipline. In the second section of Chapter 2, a queueing model is developed with a more general modification to the FCFS service discipline. The model is further extended to allow tasks to fork into multiples of other task types in Chapter 3. In Chapter 4 the system design problem is formulated as a combinatorial optimization problem and solved using the simulated annealing method. Conclusions and a summary of the thesis is given in Chapter 5.

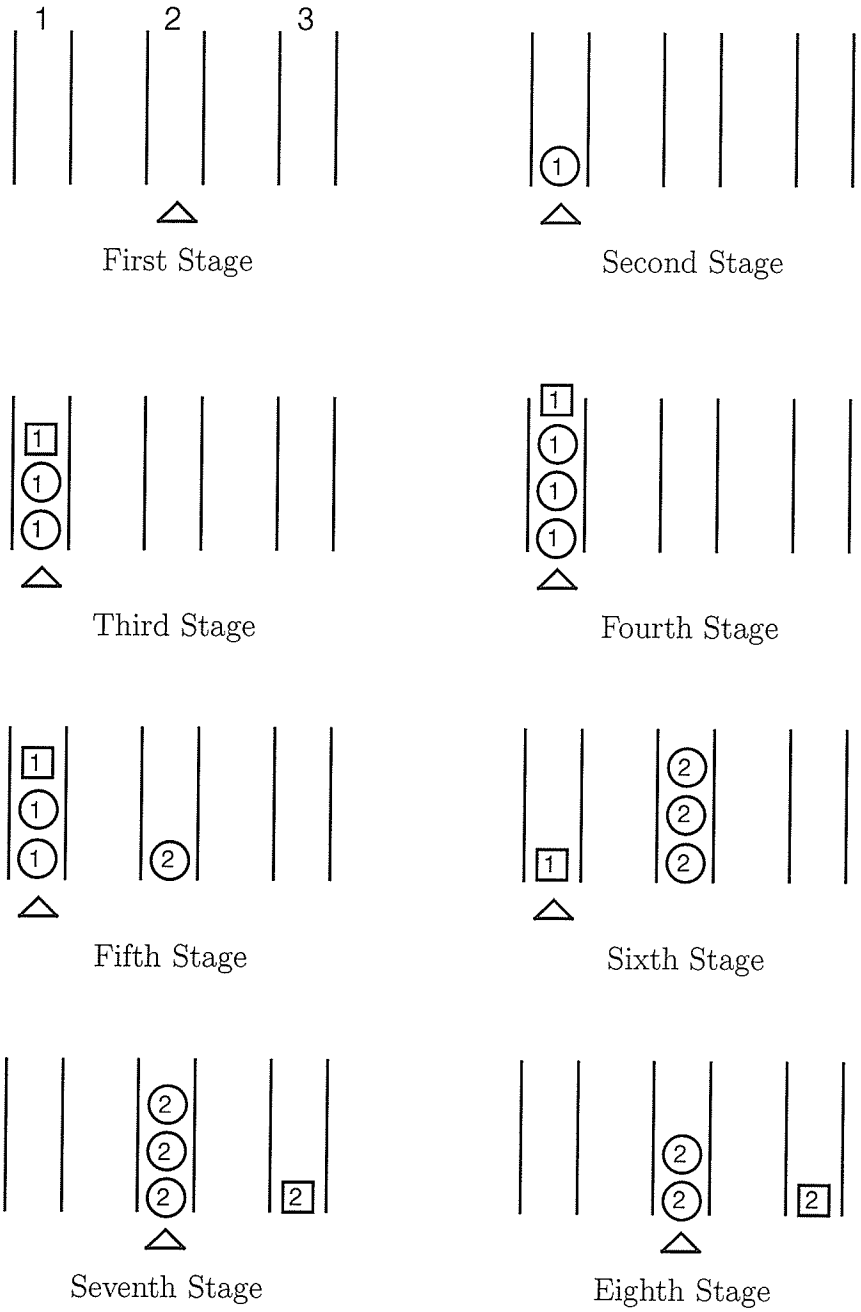

Figure 1.3: Priority Queueing Example

Chapter 2

Modified FCFS Priority Queues

2.1 Single Jumping

2.1.1 Introduction

In order to determine what effect jumping has on the sojourn time of tasks, a queueing model was developed that allowed certain tasks to jump to the front of the queue. The certain tasks are those that enter the priority queue to find the server tending to a similar task. In this situation, the arriving task will jump ahead of dissimilar tasks that are waiting in the queue. The only way a task can enter a priority queue when the server is busy, is by the arrival of a job (with its first task immediately joining the appropriate queue). Therefore, a job's first task will be the only one which will have the opportunity to jump other tasks. It has already been mentioned, that these tasks will normally have a high priority. For the model, it will be assumed

that these tasks have a priority of one (the highest value). If this is not assumed, there could be jumping in other priority queues. This would cause a problem, since the probability of finding a given task in service at the time of arrival, becomes difficult. The difficulty lies in the fact that these tasks can be preempted. It is known how to calculate the probability of these tasks being found either in service or a preempted state, but the probability of these tasks being found in service only would need to be found. The mathematics of the model build upon the foundation of Daigle's model [16], since it lent itself to be further developed into modeling this type of queueing discipline.

2.1.2 Sojourn Times

Sojourn times are calculated by tagging an arbitrary arriving job, and computing each tagged task's average sojourn time. This is made possible, by the equivalence of what a Poisson arrival sees and what is seen by a random observer [47]. Further, it has been shown that the fraction of arrivals that see a process in some state, is equal to the fraction of time the process is in that state [48]. This leads to the conclusion that the average delays experienced by the tagged tasks are equal to those experienced by a task in stochastic equilibrium. Little's result is used to describe equilibrium conditions, resulting in a linear system of equations from which sojourn time can be determined.

Consider a system with C classes of jobs, where every job in class c

$(1 \leq c \leq C)$ has T_c tasks and a priority vector $P_c = (P_{c1}, P_{c2}, ..., P_{cT_c})$ such that P_{ci} is the priority of task i in job c. The service time for task ci (x_{ci}) is drawn independently from the distribution B_{ci}. Let the term t_{c0} denote the arrival time of the tagged class c job, and the term t_{ci} denote the time at which service of the tagged task ci is completed. The expected sojourn time for task ci, S_{ci}, is defined to be the expected value of the quantity $t_{ci} - t_{c0}$. The expected sojourn time for a task, is found by summing up delays due to other tasks arriving prior to t_{c0}, at t_{c0}, and after t_{c0}. The following equations describe the effect tasks dj have on the tagged task ci.

The expected delay to task ci due to early arrivals (tasks arriving prior to t_{c0}) is denoted $S_{E_{ci}}$, and is equal to the time required to service those early tasks that received service during the time interval $(t_{c0},\ t_{ci}]$. This is determined by first finding the expected number of early tasks dj that have been served during this time period. It is the number of tasks that are found in the system at t_{c0}, subtracted by the number of tasks that are left in the system at t_{ci}. Since $S_{E_{ci}}$ is the delay to task ci, this difference must be multiplied by the expected service time for task dj. In addition, a term must be included to take into account the possibility that the arriving task ci will find one of these tasks either in service or in a preempted state (the term subtracts the service that this task has already received). Summing the effect of all of the task types, results in the following equation:

$$S_{E_{ci}} = \sum_{dj; E_{c0;dj} \neq E_{ci;dj}} (E_{c0;dj} - E_{ci;dj})\bar{x}_{dj} + \delta_{dj}(r_{dj} - \bar{x}_{dj}) \qquad (2.1)$$

where $E_{c0;dj}$ = the expected number of tasks dj found in the system by the arriving tagged class c job.
$E_{ci;dj}$ = the expected number of tasks dj that are left in the system at time t_{ci}.
$\bar{x}_{dj}$ = the expected service time of task dj.

δ_{dj} = the probability that an arbitrary arriving job finds a task dj either in service or interrupted.
r_{dj} = the mean residual life for task dj.

The value of $E_{c0;dj}$ is equal to $\lambda_d S_{dj}$ due to Little's result (λ_d being the mean arrival rate for class d jobs), for $1 \leq c \leq C$, $1 \leq d \leq C$, and $1 \leq j \leq T_d$. A value for δ_{dj} can be determined through the use of the expression

$$\delta_{dj} = \frac{\rho_{dj}}{1 - \sum_{g=1}^{C} \sum_{k=1}^{m_{g;dj}} \rho_{gk}} \tag{2.2}$$

where $m_{g;dj}$ = the index of the last task in a job of this class that runs during an interruption of a task dj. If its value is equal to zero, then the sum is also zero.
ρ = the traffic intensity ($\lambda\bar{x}$).

This equation is the result of Little's result and the equation for the average completion time (from [44] pp. 64–65). The average completion time referred to here, is the amount of time that task dj is either in service or is in a preempted state. For the single class problem, that time is equal to $\frac{\bar{x}_j}{1-\lambda\sum_{k=1}^{m_j}\bar{x}_k}$. Since this dissertation deals with multiple classes, tasks from the other classes must be taken into account. The result, is an average completion time which is equal to $\frac{\bar{x}_{dj}}{1-\sum_{g=1}^{C}\lambda_g\sum_{k=1}^{m_{g;dj}}\bar{x}_{gk}}$. The value for δ_{dj} is also the average number of tasks dj which are either in service or in a preempted state at an arbitrary point in time. This is due to the fact that there can

only be zero or one of these tasks in such a state, at any given point in time. Thus, Little's result can be used to relate δ_{dj} to the average completion time, resulting in Equation 2.2.

A value for r_{dj} can be found by using the equation $r_{dj} = E[x_{dj}^2]/2\bar{x}_{dj}$, where the first and second moments are known from the input data. The equation for r_{dj} is widely known and can be found in [30] (page 173). The value of $E_{ci;dj}$ can be found using the following recursive equation and an initial condition.

$$E_{ci;dj} = \begin{cases} E_{c,i-1;dj} & P_{dj} > P_{ci} \\ E_{c,i-1;d,j-1} & P_{dj} = P_{ci} & i \neq 1 \\ E_{c,i-1;d,j-1} & P_{dj} = P_{ci} & i = 1,\ j = 1,\ c = d \\ E_{c,i-1;d,j-1} & P_{dj} = P_{ci} \neq P_{d,j-1} & i = 1,\ j \neq 1 \\ \delta_{ci}(E_{c,i-1;dj} - \lambda_d \bar{x}_{dj}) & P_{dj} = P_{ci} & i = 1,\ j = 1,\ c \neq d \\ \delta_{ci}(E_{c,i-1;dj} - \lambda_d \bar{x}_{dj}) & & \\ +(1-\delta_{ci})E_{c,i-1;d,j-1} & P_{dj} = P_{ci} = P_{d,j-1} & i = 1,\ j \neq 1 \\ E_{ci;d,j-1} & P_{dj} < P_{ci} \end{cases} \tag{2.3}$$

The initial condition is $E_{ci;d0} = 0$ for $1 \leq c \leq C$, $0 \leq i \leq T_c$, and $1 \leq d \leq C$. The task $d0$ is an imaginary task of job d, and it runs at the instant of job arrival. This task has the system's highest priority and requires zero service time, so there are never any of them in the queue at a point in time (e.g. t_{ci}).

The term $E_{ci;dj}$ in Equation 2.3 recurses to either $E_{c,i-1;dj}$, $E_{c,i-1;d,j-1}$, or $E_{ci;d,j-1}$. The justification for doing so will be given in the next section, where a proof is given for a similar (but not identical) equation (Equation 2.7). What is important here, is to discuss the conditions for which these various

terms apply and any additional terms that are needed. The discussion involves the part of the equation where $P_{dj} = P_{ci}$. This is a requirement for jumping. If the two tasks are not in the same priority queue, jumping will not take place. When dealing with the delay due to early arrivals, it must be taken into account that the tagged task can (on occasion) jump over tasks dj which have arrived before it. It has already been mentioned, that the jumping task must be the first task of a particular job. Since task ci will be doing the jumping in this case, no jumping can take place when $i \neq 1$. The equation $E_{ci;dj} = E_{c,i-1;d,j-1}$ for $P_{dj} = P_{ci}$ is applicable when $i \neq 1$. There are two situations which can arise when $i = 1$ and $E_{ci;dj} = E_{c,i-1;d,j-1}$ is still applicable. One of these occasions is when $j = 1$ and $c = d$. Jumping will not occur in this situation, because the two tasks being considered are identical. It is important and necessary to state this, because the algorithm takes into account the effect that every task has on a given task. Even those tasks which are in the same job as the task under consideration (including the task which is the same as the one being considered) are taken into account. In other words, the influence ci tasks have on the tagged task ci is taken into account. The second occasion is when $j \neq 1$ and $P_{dj-1} \neq 1$. For task dj to be jumped, it must find itself between two ci tasks. This would never happen since $P_{dj-1} > 1$. Task dj would only enter the priority one queue when it would be empty (otherwise task $dj - 1$ would not obtain service). For all other situations, when $i = 1$ (and $P_{dj} = P_{ci}$) one of two other equations must

be used.

The first of the two equations is for when $j = 1$ and $c \neq d$. If no jumping were allowed, $E_{ci;dj}$ would equal zero ($E_{ci;d0} = 0$ for any value of i). Even when jumping is allowed, the number of early tasks dj that are left in the system at time t_{ci} is zero most of the time. It is the odd time ($100\delta_{ci}$ percent of the time to be exact) that jumping occurs and all the tasks dj that the arriving job found in the queue are left at time t_{ci}. The expected number of tasks dj that are left in the system at time t_{ci}, is the probability of jumping (δ_{ci}) multiplied by the expected number of tasks at time t_{c0} (the arrival time of the tagged task) which are waiting to be served. The expected number of tasks that are waiting to be served at time t_{c0} is equal to $\lambda_d(S_{dj} - \bar{x}_{dj})$. The expected number of tasks dj that are left in the system at time t_{ci} is then equal to $\delta_{ci}\lambda_d(S_{dj} - \bar{x}_{dj})$. Since $E_{c0;dj} = \lambda_d S_{dj}$ the equation can be rewritten as $E_{ci;dj} = \delta_{ci}(E_{c,i-1;dj} - \lambda_d \bar{x}_{dj})$. The second of the two equations is for when $j \neq 1$ and $P_{d,j-1} = 1$. The first part of this equation is similar to the previous equation in that $100\delta_{ci}$ percent of the time the queueing system acts like $P_{dj} > P_{ci}$. However, the rest of the time $(1 - \delta_{ci})$ the result is not zero, but $E_{c,i-1;d,j-1}$. The result is the equation $E_{ci;dj} = \delta_{ci}(E_{c,i-1;dj} - \lambda_d \bar{x}_{dj}) + (1 - \delta_{ci})E_{c,i-1;d,j-1}$.

The expected delay to task ci due to arrivals at t_{c0} is denoted $S_{A_{ci}}$, and can be calculated with the simple summation

$$S_{A_{ci}} = \sum_{k=1}^{i} \bar{x}_{ck}. \tag{2.4}$$

The expected delay to task ci due to late arrivals (tasks arriving after t_{c0}) is denoted $S_{L_{ci}}$, and is equal to the time required to service late arriving tasks that received service during the time interval $(t_{c0},\ t_{ci}]$. The delay can be calculated with the following equation.

$$S_{L_{ci}} = \sum_{d=1}^{C} \sum_{j=1}^{T_d} L_{ci;dj} \bar{x}_{dj} \tag{2.5}$$

Where $L_{ci;dj}$ = the expected number of tasks dj from the late arrivals that are serviced by time t_{ci}.

Like $E_{ci;dj}$, the value of $L_{ci;dj}$ can be determined by the use of a recursive equation and a couple of initial conditions.

$$L_{ci;dj} = \begin{cases} L_{c,i-1;dj} & P_{dj} > P_{ci} & \\ L_{c,i-1;d,j-1} & P_{dj} = P_{ci} & j \neq 1 \\ L_{c,i-1;d,j-1} & P_{dj} = P_{ci} & i = 1,\ j = 1,\ c = d \\ L_{c,i-1;d,j-1} & P_{dj} = P_{ci} \neq P_{c,i-1} & i \neq 1,\ j = 1 \\ \delta_{dj}(L_{ci;d,j-1} - \lambda_d \bar{x}_{ci}) & P_{dj} = P_{ci} & i = 1,\ j = 1,\ c \neq d \\ \delta_{dj}(L_{ci;d,j-1} - \lambda_d \bar{x}_{ci}) & & \\ +(1 - \delta_{dj}) L_{c,i-1;d,j-1} & P_{dj} = P_{ci} = P_{c,i-1} & i \neq 1,\ j = 1 \\ L_{ci;d,j-1} & P_{dj} < P_{ci} & \end{cases} \tag{2.6}$$

The first initial condition is $L_{c0;dj} = 0$ for $1 \leq c \leq C$, $1 \leq d \leq C$, and $0 \leq j \leq T_d$. The second is $L_{ci;d0} = \lambda_d S_{ci}$ for $1 \leq c \leq C$, $1 \leq i \leq T_c$, and $1 \leq d \leq C$. Since task $d0$ is a special task (it is "serviced" immediately upon arrival), those that arrive before the tagged task has completed service (and after the tagged job has arrived) will be served. This term ($L_{ci;d0}$) is the mean number of class d job arrivals during $(t_{c0}, t_{ci}]$.

The term $L_{ci;dj}$ in Equation 2.6 recurses to either $L_{c,i-1;dj}$, $L_{c,i-1;d,j-1}$, or $L_{ci;d,j-1}$. The justification for doing so will be given in the next section, where

a proof is given for a similar (but not identical) equation (Equation 2.11). What will be given here, is a discussion on the conditions for which these various terms apply and any additional terms that are needed. Again, the discussion need only involve the part of the equation where jumping will take place ($P_{dj} = P_{ci}$). When dealing with the delay due to late arrivals, it must be taken into account that these arrivals can (on occasion) jump the tagged task while it waits in the queue. In this case, it is the task dj which will be doing the jumping. Again, the jumping task must be the first task of a job. This results in no jumping taking place when $j \neq 1$. The equation which states this is $L_{ci;dj} = L_{c,i-1;d,j-1}$ when $P_{dj} = P_{ci}$ and $j \neq 1$. This equation should also be used on two occasions when $j = 1$. The first occasion is when $i = 1$ and $c = d$. Jumping will not occur in this situation, because the two tasks being considered are identical. The second occasion is when $i \neq 1$ and $P_{c,i-1} \neq 1$. For task ci to be jumped, it must find itself between two dj tasks. This would never happen since $P_{c,i-1} > 1$. Task ci would only enter the priority one queue when it would be empty (otherwise task $c, i-1$ would not obtain service). For all other situations when $j = 1$ (and $P_{dj} = P_{ci}$), one of two other equations must be used.

The first of the two equations is for when $i = 1$ (and $c \neq d$). If no jumping were allowed, $L_{ci;dj}$ would equal zero ($L_{c0;dj} = 0$ for any value of j). When jumping is allowed, the tagged task can be jumped at any time from its arrival to the time when it begins to receive service. This time

period is given by $S_{ci} - \bar{x}_{ci}$. To obtain the expected number of tasks dj that can jump the tagged task during this time period, one has to multiply the time period by the arrival rate (λ_d) and by the probability that the arriving task will find a dj receiving service (δ_{dj}). The resulting equation is $L_{ci;dj} = \delta_{dj}\lambda_d(S_{ci} - \bar{x}_{ci})$. Using the equation $L_{ci;d0} = \lambda_d S_{ci}$, the result can be written as $L_{ci;dj} = \delta_{dj}(L_{ci;d,j-1} - \lambda_d \bar{x}_{ci})$. The second of the two equations is for when $i \neq 1$ and $P_{c,i-1} = 1$. The first part of this equation is similar to the previous equation, but an additional part is needed. The additional part has to do with the contribution to $L_{ci;dj}$ that dj makes when jumping is not occurring ($100(1 - \delta_{dj})$ percent of the time). This additional part is $(1 - \delta_{dj})L_{c,i-1;d,j-1}$. The addition is needed, because the term $L_{c,i-1;d,j-1}$ is no longer zero (like it is when $i = 1$). The whole equation is $L_{ci;dj} = \delta_{dj}(L_{ci;d,j-1} - \lambda_d \bar{x}_{ci}) + (1 - \delta_{dj})L_{c,i-1;d,j-1}$.

The expected sojourn time for task ci is given in the equation $S_{ci} = S_{E_{ci}} + S_{A_{ci}} + S_{L_{ci}}$. Applying this equation to each task in every job ($1 \leq c \leq C$ and $1 \leq i \leq T_c$) gives a linear system of equations of the form $S = aS + b$. The number of the equations is equal to the number of unique tasks in the queueing system. The unknowns are solved for by using the Jacobi iteration method [26]. This method was fast (returning results in 1 or 2 seconds), and it had no difficulty solving problems with 20 or so task types (a large problem as far as this particular problem is concerned). The matrix S is a column vector (S_{11}, S_{12}, $\cdots$, S_{1T_1}, S_{21}, $\cdots$, S_{2T_2}, $\cdots$, S_{C1}, $\cdots$, S_{CT_C}),

while a and b are matrices (b is also a column vector) of known quantities.

2.1.3 Numerical Examples

Two queueing disciplines were tried on an example problem, where one allowed jumping and the other did not. This was done to determine the effect that jumping had on task and job sojourn times. A number of runs were done on a system containing two jobs, where the arrival rates of the jobs were varied. However, the ratio of the arrival rates of the two jobs remained constant. Job 2 had ten times the arrival rate of Job 1 for every run. When jumping was allowed, the sojourn time of the first task of the job with the higher arrival rate was lower than it would have been had jumping not been allowed (see Figure 2.1). The difference is small, only because of the numbers used in this particular example. Different numbers were not used, because it would have made the differences in following figure much larger. This larger difference would have required a change of scale (along the y axis) to plot properly. This would have made a comparison between these two figures (Figure 2.1 and Figure 2.2) difficult. The specific values for the traffic intensity are also not that important. Higher values of traffic intensity will give larger differences in sojourn time, but it is the relative differences that are important. The opposite of what happened to the job with the higher arrival rate, is seen happening to the job with the lower arrival rate (see Figure 2.2).

The sojourn time for this job's first task was higher than it would have been had jumping not been allowed. As the arrival rates increase, so does

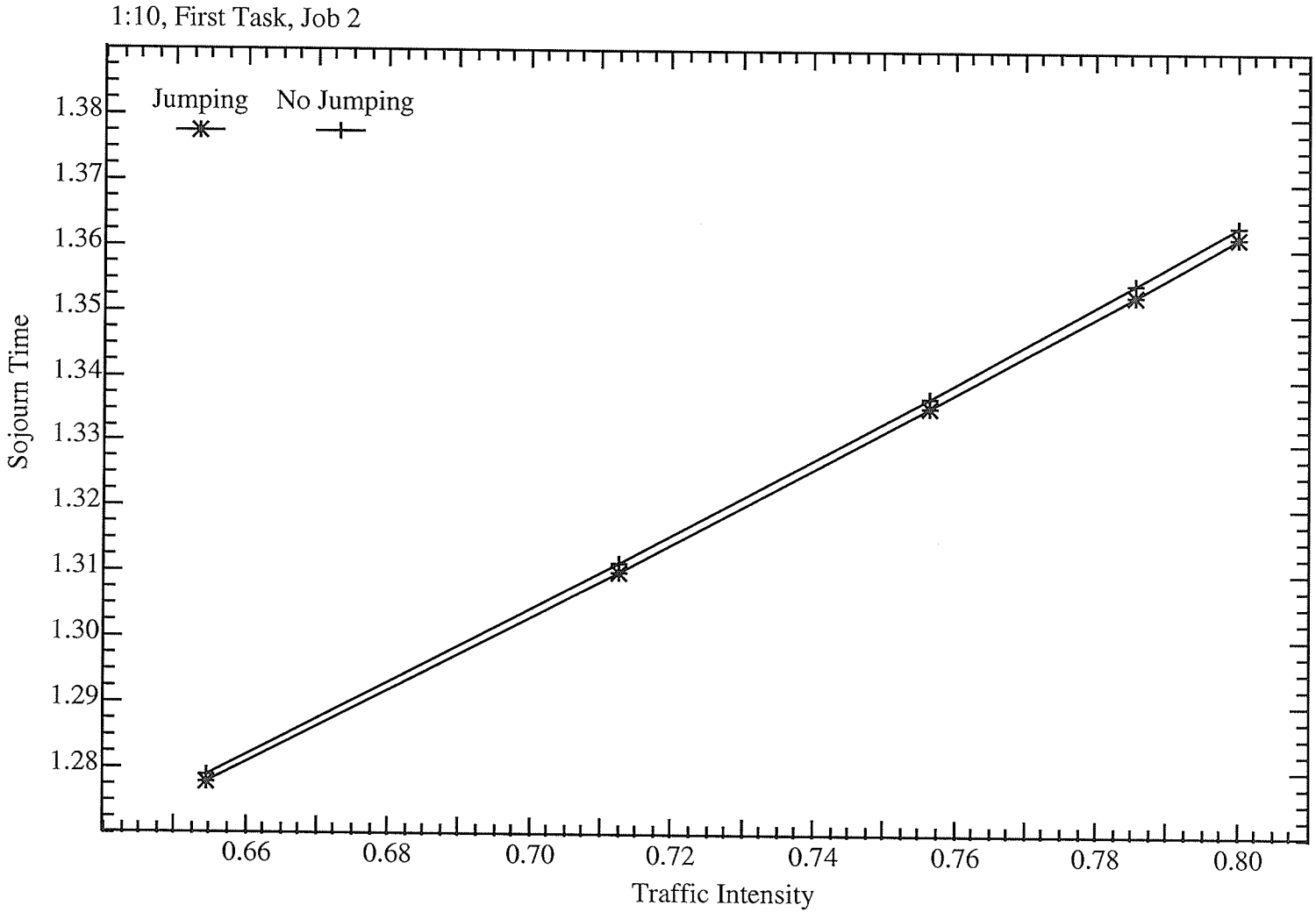

Figure 2.1: Sojourn time of the first task in Job 2

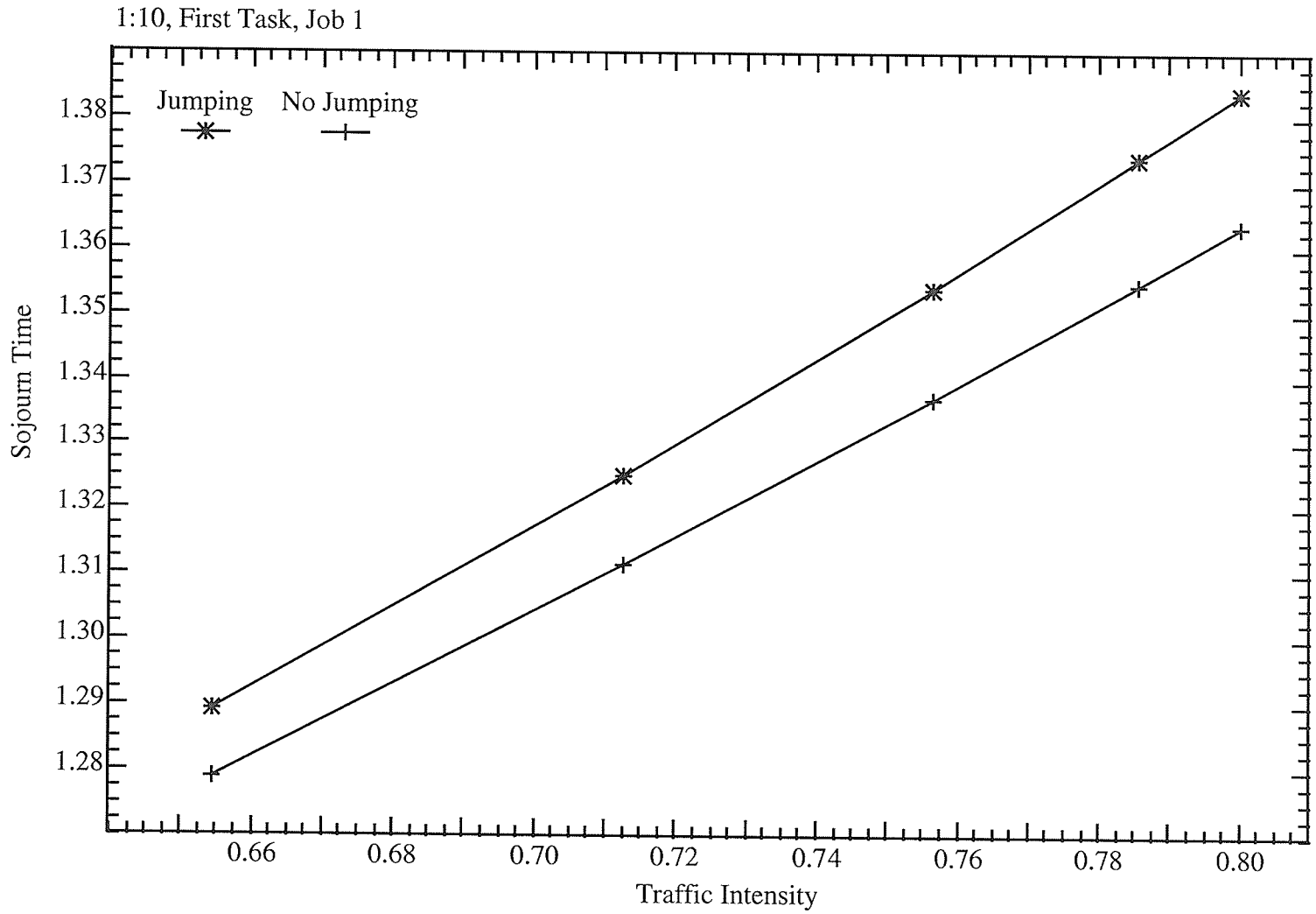

Figure 2.2: Sojourn time of the first task in Job 1

the benefit to the frequently arriving job's task and the loss to the seldom arriving job's task. It is obvious from the two figures, that the gain of the benefiting task does not equal the loss of the losing task. If the ratio of the arrival rates is 2 to 1, the benefiting task gains one half of what the losing task loses. This is because there are two tasks (from two jobs) gaining for every task which loses. There is conservation of work, as one job class gains what the other loses. In the figures, the difference of ten to one is evident when a comparison between the first two figures is made. The arrival rate is not the only factor to influence the sojourn time of the first task. The service time of the first task also determines the size of the benefit or loss. A job would benefit by having a larger service time for the first task; since this would create a longer time period for its tasks to jump. A longer time period would mean more tasks jump, lowering the time the task spends waiting to be served. Differences in service times affect the sojourn time in a manner similar to that for differences in arrival rates. All other things being equal, a job having twice the service time for the first task than that of another job, will benefit one half of what the first task of the other job will lose.

As far as message transit time is concerned, neither of the two jobs experienced a gain or a loss when changes to arrival rates or service times were made. Figures 2.3 and 2.4 show this to be true for a change in the arrival rate. Notice that the line in Figure 2.3 is slightly curved, and the line in Figure 2.4 is considerably curved. Both show the effect of low priority tasks.

For the example problem, the priority of the first tasks in each job were equal to one. The straight lines in Figures 2.1 and 2.2, show that the increased sojourn time is due to the increased traffic of the priority one tasks. Figures 2.3 and 2.4 involve the other tasks in the jobs. Both jobs had three tasks, with the second and third tasks having priorities of two and one (respectively) for Job 1 and three and one (respectively) for Job 2. When the traffic intensity is increased, it is the low priority tasks that suffer. The priority three task in Job 2, feels the increased traffic of the priority one, two, and three tasks. This is seen in the dramatic rise of the sojourn time with traffic intensity in Figure 2.4. The rise in Figure 2.3 is slower, because its lowest priority task has a priority of two.

The results also show a benefit to having the modified FCFS queueing discipline. This benefit is that frequently arriving messages that have a contention protocol, are less likely to time out and require retransmission.

2.2 Multiple Jumping

2.2.1 Introduction

The differences in the sojourn times between the two queueing disciplines that were seen in the previous section, has prompted further work in this area. The queueing model presented in this section differs from the one in the previous section, in that the modification to the first-come first-served service discipline is different. Here, tasks are allowed to jump and join similar

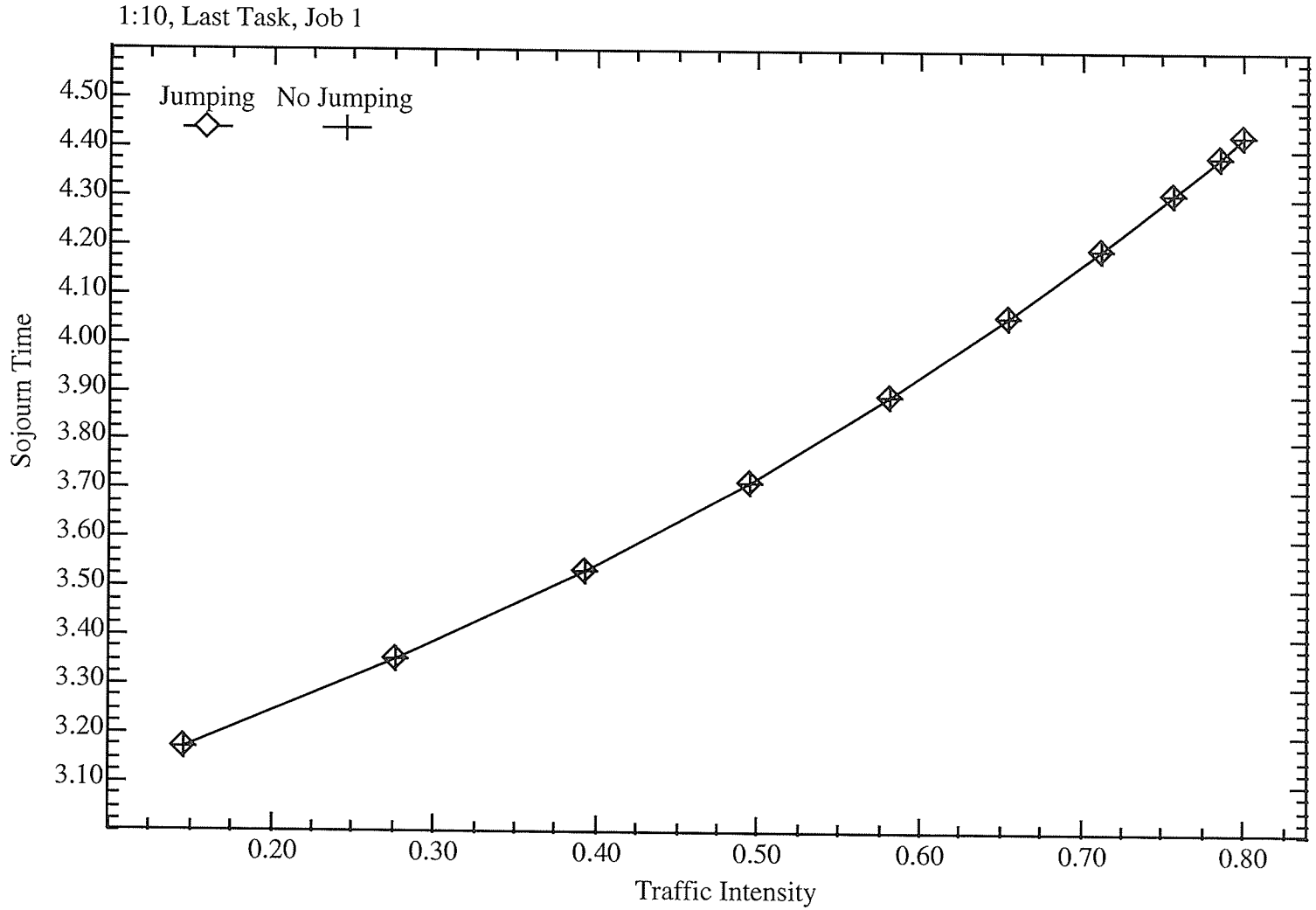

Figure 2.3: Sojourn time of the last task in Job 1

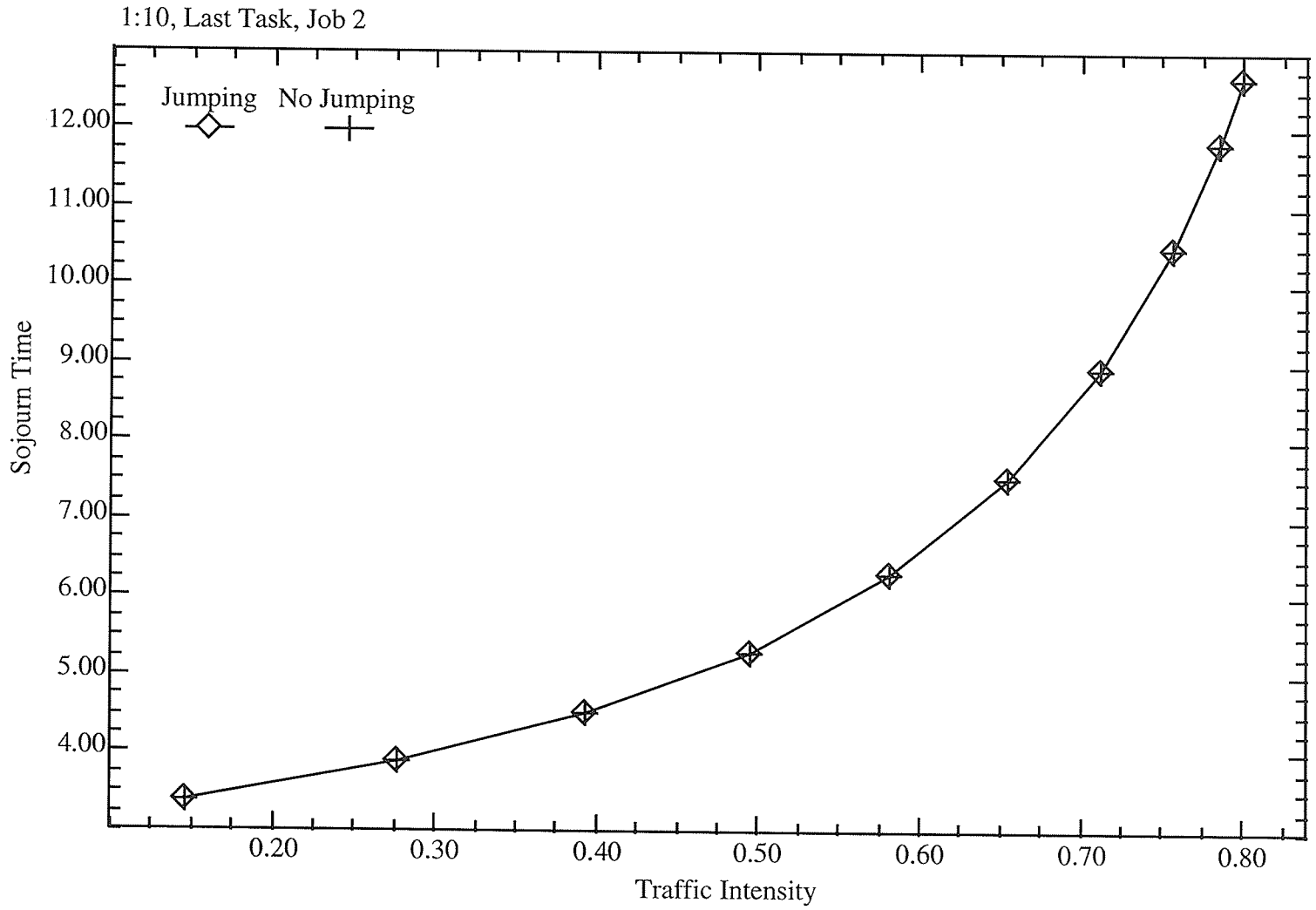

Figure 2.4: Sojourn time of the last task in Job 2

tasks wherever they exist in the queue. This means that tasks are going to jump, even when the server is not servicing a task of its type. It also means that tasks are going to be jumping in queues other than the priority one queue. The end result is a grouping of task types in each priority queue. To illustrate see Figure 2.5a. Task A arrives to find other tasks of its type in the

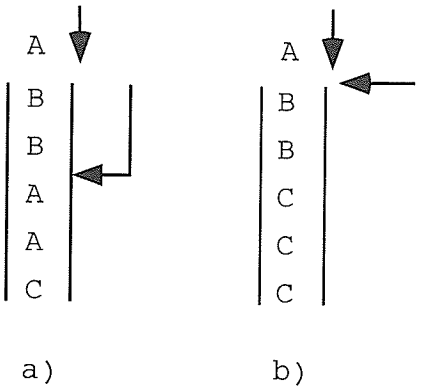

Figure 2.5: Task A joins the queue.

queue and jumps ahead to join them. It will position itself behind the group of tasks of its type. In doing so, it is jumping ahead of tasks from different groups, that arrived before it (even though they are of the same priority). If there are no tasks of its type, the task will position itself at the end of the queue. In Figure 2.5b, task A does not find any tasks of its type and has to join the back of the queue (forming a group of its own). Again, similar tasks are tasks which not only come from the same job type, but they must also come from the same part of that job. This model more closely models the

actual movement of tasks in system software.

2.2.2 Sojourn Times

Portions of the mathematics for this model are identical to portions for the model in the previous section. So in order to avoid repetition, only those portions which differ will be discussed. The first difference is in the equation for $E_{ci;dj}$, which will be derived from a random variable equation.

Remaining Tasks: Random Variable Equation

Two random variables must be defined.

$X_{c0;dj}$ = the number of early tasks dj found in the system by the arriving tagged class c job.
$X_{ci;dj}$ = the number of early tasks dj that are left in the system at time t_{ci}.

Proposition 1

The equation relating the two variables, is the following:

$$X_{ci;dj} = \begin{cases} X_{c,i-1;dj} & P_{dj} > P_{ci} \\ X_{c,i-1;d,j-1} & P_{dj} = P_{ci} \quad ci = dj \\ & \quad \text{or} \quad P_{ci} < P_{c,i-1} \\ & \quad \text{or} \quad P_{dj} < P_{d,j-1} \\ X_{c,i-1;d,j-1} \mid \text{jumping does not occur} & P_{dj} = P_{ci} \quad ci \neq dj \\ & \quad \text{and} \quad P_{ci} > P_{c,i-1} \\ & \quad \text{and} \quad P_{dj} > P_{d,j-1} \\ X_{c,i-1;dj} \mid \text{jumping occurs} & P_{dj} = P_{ci} \quad ci \neq dj \\ & \quad \text{and} \quad P_{ci} > P_{c,i-1} \\ & \quad \text{and} \quad P_{dj} > P_{d,j-1} \\ X_{ci;d,j-1} & P_{dj} < P_{ci} \end{cases} \tag{2.7}$$

The initial condition $X_{ci;d0} = 0$ for $1 \leq c \leq C$, $0 \leq i \leq T_c$, and $1 \leq d \leq C$.

Proof

The verification of Equation 2.7 is as follows. When $P_{dj} > P_{ci}$ no tasks dj will be serviced during the time interval $(t_{c,i-1},\ t_{ci}]$. That is to say that the number of early tasks dj that are present at time $t_{c,i-1}$, are also present at time t_{ci}. As a result, $X_{ci;dj} = X_{c,i-1;dj}$. When $P_{dj} = P_{ci}$, there are two possibilities: either the task ci will jump over tasks dj, or it will not. If it does not jump, any task dj with a priority equal to that of task ci and which has not yet joined the ready-for-service queue by time $t_{c,i-1}$ will still be in the system at time t_{ci}. These tasks will have predecessor tasks $[d, j-1]$ that are still in the system at time $t_{c,i-1}$. The number of predecessor tasks in the system at this time, will equal the number of tasks dj that are still in the system at time t_{ci}. As a result, $X_{ci;dj} = X_{c,i-1;d,j-1}$. If $ci = dj$, the two tasks are similar and jumping will not occur. Two other conditions will ensure that jumping will not occur are $P_{ci} < P_{c,i-1}$ and $P_{dj} < P_{d,j-1}$. In both cases, these tasks (ci and dj, respectively) enter empty queues. In order for jumping to be possible, the tagged task ci must find a task (or group of tasks) dj waiting, when it enters the queue (as well as a group of its own type to jump to). If any one of these three conditions are true, jumping will not occur (more than one can be true for a particular ci and dj). If $ci \neq dj$, $P_{ci} > P_{c,i-1}$, and $P_{dj} > P_{d,j-1}$, there is a chance that tagged task ci will jump over tasks dj. When this happens, no tasks dj will be serviced during

the time interval $(t_{c,i-1},\ t_{ci}]$. This situation is similar to what happens when $P_{dj} > P_{ci}$. It has been shown that in this case, $X_{ci;dj} = X_{c,i-1;dj}$. This portion of the equation will produce a decrease in the sojourn time of the tagged task, to account for the fact that the tagged task need not always wait for earlier tasks dj to be serviced before receiving service itself. This only occurs with certain configurations of the groups of task types in the queue. That is, the arriving task ci must find one or more tasks of its type in front of earlier tasks dj, when it arrives. If this does not happen, jumping will not occur and $X_{ci;dj} = X_{c,i-1;d,j-1}$ (the reasoning for which has already been discussed). When $P_{dj} < P_{ci}$, only those tasks dj whose predecessor tasks $[d, j-1]$ have not been serviced by time t_{ci} will still be in the system at t_{ci}. The number of tasks $d, j-1$ which have not been serviced by time t_{ci} is equal to $X_{ci;d,j-1}$. The result is $X_{ci;dj} = X_{ci;d,j-1}$. □

The only possibilities not covered by Equation 2.7 are when $P_{ci} = P_{c,i-1}$ and/or when $P_{dj} = P_{d,j-1}$. A problem arises when either of these two equations are true. The problem is due to tasks jumping other tasks within the same job type. This would happen, for example, in a four task job with the set of priorities (1, 2, 2, 1). The problem tasks are the second and third task of this job. A second task has the opportunity to get ahead of third tasks, without a third task having the chance to jump second tasks. If the tagged task is the second task, it may jump third tasks that have arrived early (since it will find both second and third tasks in the queue). However,

if the tagged task is the third task, it cannot jump second tasks that have arrived early. This is because this tagged task will find no second tasks (that have arrived early) in the queue. All early second tasks must be served before the tagged second task. It is this imbalance that the equations are unable to deal with. The restriction is a minor one, since there is little benefit to having consecutive tasks of equal priority. Jobs are broken up into tasks, for the most part, to be treated differently.

Remaining Tasks: Expected Value Equation

Forming the expectation of both sides of Equation 2.7 and introducing probability terms results in the following equation.

$$
\begin{aligned}
E[X_{ci;dj}] \;=\;& E[X_{c,i-1;dj} \mid_{P_{dj}>P_{ci}}] Pr_{P_{dj}>P_{ci}} \\
&+ E[X_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci=dj}] Pr_{P_{dj}=P_{ci},\ ci=dj} \\
&+ E[X_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}<P_{c,i-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}<P_{c,i-1}} \\
&+ E[X_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}<P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}<P_{d,j-1}} \\
&+ E[X_{c,i-1;dj}(\text{jumping}) \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}} \\
&+ E[X_{c,i-1;d,j-1}(\text{no jumping}) \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}} \\
&+ E[X_{ci;d,j-1} \mid_{P_{dj}<P_{ci}}] Pr_{P_{dj}<P_{ci}} \qquad (2.8)
\end{aligned}
$$

where $Pr_{P_{dj}>P_{ci}}$ = the probability of the priority of dj being larger than the priority of ci.

Some of the probability terms arise from $P_{dj} > P_{ci}$, $P_{dj} = P_{ci}$, and $P_{dj} < P_{ci}$. The other terms arise from $ci = dj$ and $ci \neq dj$, $P_{ci} < P_{c,i-1}$ and $P_{ci} > P_{c,i-1}$, and $P_{dj} < P_{d,j-1}$ and $P_{dj} > P_{d,j-1}$. These five inequalities and one equation, create eight possible combinations for tasks ci and dj when $P_{dj} = P_{ci}$. In seven of the eight combinations, jumping will not occur. The term $Pr_{P_{dj}=P_{ci},\ ci=dj}$ accounts for four of these, $Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}<P_{c,i-1}}$ accounts for two more, and $Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}<P_{d,j-1}}$ accounts for the last. Jumping is possible with the remaining combination; when $ci \neq dj$, $P_{ci} > P_{c,i-1}$, and $P_{dj} > P_{d,j-1}$. For any ci and dj, all but one of the above probability terms will be equal to zero, with the remaining term equal to one.

It is easier to work with the equation in its decomposed form for computational purposes, than to calculate the probabilities. The result is the following equation:

$$E_{ci;dj} = \begin{cases} E_{c,i-1;dj} & P_{dj} > P_{ci} \\ E_{c,i-1;d,j-1} & P_{dj} = P_{ci} \quad ci = dj \\ & \quad \text{or} \quad P_{ci} < P_{c,i-1} \\ & \quad \text{or} \quad P_{dj} < P_{d,j-1} \\ \Theta_{ci;dj} E_{c,i-1;dj} \mid \text{jumping occurs} & P_{dj} = P_{ci} \quad ci \neq dj \\ & \quad \text{and} \quad P_{ci} > P_{c,i-1} \\ & \quad \text{and} \quad P_{dj} > P_{d,j-1} \\ (1 - \Theta_{ci;dj}) E_{c,i-1;d,j-1} \mid \text{jumping does not occur} & \\ & P_{dj} = P_{ci} \quad ci \neq dj \\ & \quad \text{and} \quad P_{ci} > P_{c,i-1} \\ & \quad \text{and} \quad P_{dj} > P_{d,j-1} \\ E_{ci;d,j-1} & P_{dj} < P_{ci}. \end{cases} \tag{2.9}$$

where $E_{ci;dj}$ = the expected number of early tasks dj that are left in the system at time t_{ci}.

$\Theta_{ci;dj}$ = the probability of tagged task ci jumping tasks dj.

The first initial condition is $E_{ci;d0} = 0$ for $1 \leq c \leq C$, $0 \leq i \leq T_c$, and $1 \leq d \leq C$. The second initial condition is $E_{c0;dj} = \lambda_d S_{dj}$ for $1 \leq c \leq C$, $1 \leq d \leq C$, and $1 \leq j \leq T_d$.

Jumping Probability

An approximate value for $\Theta_{ci;dj}$ can be found by relating the traffic intensities and arrival rates of the groups in a priority queue. A task group is the expected number of tasks of that type, and is equal to the task's arrival

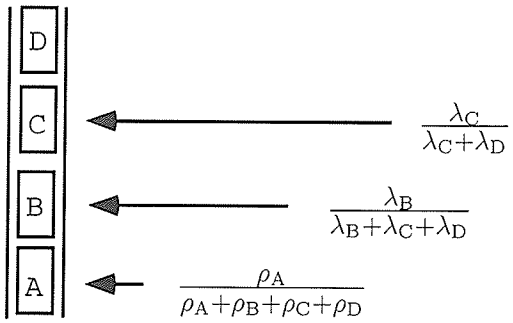

Figure 2.6: Configuration probabilities.

rate multiplied by its individual sojourn time (this is the time spent in the ready-for-service queue plus the service time). A value for $\Theta_{ci;dj}$ is found by approximating the probability that an arbitrary arrival will find group *ci* in front of group *dj*. This is done by determining the probability of every possible configuration (order) of groups in a priority queue, and summing up those where tasks *ci* are ahead of tasks *dj*. To find the probability of a given configuration, one has to determine the probability of finding the first group first, and the second group second, and so forth (see Figure 2.6). The probability of finding a group at the front of the queue is set equal to the ratio of the traffic intensity of that group over the total traffic intensity of all the groups in the queue. The probability of finding the second group second, given the first group, is set equal to the ratio of the arrival rate of that group over the sum of arrival rates of all the groups in the queue which could be in the second position (this excludes the group in the first position). The

probability of finding the third group in the third place, given the first and second group, is set equal to the ratio of the arrival rate of that group over the sum of arrival rates of all the groups in the queue which could be in the third position (this excludes the groups in the first and second position). This continues on until probabilities for all positions have been found. The probability of the configuration is the product of all positional probabilities.

A value for $\Theta_{ci;dj}$ can be calculated with Equation 2.10. The configurations are grouped together, in order to describe them algebraically. The task types ci and dj are represented as $t(1)$ and $t(2)$ respectively, so as to avoid large numbers of variable names and to make it possible to represent N task types (N being the number of task types in the priority queue). The value for N is equal to the number of elements in the set $\widetilde{U}$. It cannot include those tasks which have a preceding task of lower priority, since these tasks will always enter empty queues. The first group of configurations to be considered is the group where the tasks of type $t(1)$ form the first group of the queue. The probability of each and every one of these configurations must be included in the value for $\Theta_{t(1);t(2)}$. The sum of these probabilities is equal to the traffic intensity of task type one ($\rho_{t(1)}$) divided by the sum of the traffic intensities of all the task types in the queue ($\sum_{q \in \widetilde{U}} \rho_{t(q)}$). This ratio forms the first term in Equation 2.10.

The second (and only other) group of configurations involves tasks of type other than $t(1)$ that form the first group of the queue. Whatever subgroup

of configurations are considered, the probability of this other group being at the front of the queue must be taken into account. The second term of Equation 2.10 does this ($\frac{\rho_{t(n)}}{\sum_{q\in\widetilde{U}}\rho_{t(q)}}$) and it does it for every other task type in the queue ($\sum_{n=3}^{N}$). The next term in the expression is $\frac{\lambda_{t(1)}}{\sum_{\substack{m=1\\m\neq n}}^{N}\lambda_{t(m)}}$. This, when multiplied by the second term, gives the probability of $t(1)$ forming the second group in the queue (given that $t(n)$ forms the first group). When n is varied from 3 to N ($\sum_{n=3}^{N}$), the sum is the total probability of finding group $t(1)$ second (and in front of group $t(2)$).

The next, and last, term in Equation 2.10 represents the probabilities that are associated with $t(1)$ being the third, or fourth, or fifth, etc. group in the queue (but still ahead of $t(2)$). The number of these probabilities is dependent on the number of task types in the queue. For example, if N=4 the probability that $t(1)$ forms the third group and $t(2)$ forms the fourth group, must be added to $\Theta_{t(1);t(2)}$. In order to calculate this probability, you must multiply the term $\frac{\rho_{t(n)}}{\sum_{q\in\widetilde{U}}\rho_{t(q)}}$ by the probability of the second group being second ($\frac{\lambda_{t(P_s)}}{\sum_{\substack{m=1\\m\neq n}}^{N}\lambda_{t(m)}}$) and by the probability of the third group being third ($\frac{\lambda_{t(1)}}{\sum_{u\in\widetilde{Y}}\lambda_{t(u)}}$). No matter where in the queue group $t(1)$ is, in calculating

the probability the denominator of the first λ term is $\sum_{\substack{m=1 \\ m \neq n}}^{N} \lambda_{t(m)}$ and the numerator of the last λ term is $\lambda_{t(1)}$. Since these two are common, they can be factored out. The summation $\sum_{r=1}^{N-3}$ insures that the $t(1)$ group is considered in every position in the queue. The summation $\sum_{P_s \in \widetilde{W}}$ insures that every possible order of task types is considered. While the product sum $\prod_{s=1}^{r}$ insures that all the λ terms are multiplied together.

$$\Theta_{t(1);t(2)} = \frac{1}{\sum_{q \in \widetilde{U}} \rho_{t(q)}} [\rho_{t(1)} + \sum_{n=3}^{N} \rho_{t(n)} \frac{\lambda_{t(1)}}{\sum_{\substack{m=1 \\ m \neq n}}^{N} \lambda_{t(m)}} [1 + \sum_{r=1}^{N-3} \sum_{P_s \in \widetilde{W}} \prod_{s=1}^{r} \frac{\lambda_{t(P_s)}}{\sum_{u \in \widetilde{Y}} \lambda_{t(u)}}]] \tag{2.10}$$

where $\widetilde{U} = \{gk |\ P_{t(1)} = P_{t(2)} = P_{gk},\ P_{gk} > P_{g,k-1} \text{ or } k = 1,\ 1 \leq g \leq C\}$

$\widetilde{W} = \{2 < P_s \leq N,\ P_s \neq n,\ P_s \neq P_{w-1}\ \ \forall w :\ 2 \leq w \leq s\}$

$\widetilde{Y} = \{1 \leq u \leq N,\ u \neq n,\ u \neq P_w\ \ \forall w :\ 1 \leq w \leq s\}$

Tasks Served: Random Variable Equation

Another equation to change, is that of $L_{ci;dj}$, which also will be derived from a random variable equation. Two random variables must be defined.

$Y_{ci;d0}$ = the number of late tasks dj that arrive before time t_{ci}.

$Y_{ci;dj}$ = the number of late tasks dj that are serviced by time t_{ci}.

Proposition 2

The equation relating the two variables, is the following:

$$Y_{ci;dj} = \begin{cases} Y_{c,i-1;dj} & P_{dj} > P_{ci} \\ Y_{c,i-1;d,j-1} & P_{dj} = P_{ci} \quad ci = dj \\ & \text{or} \quad P_{ci} < P_{c,i-1} \\ & \text{or} \quad P_{dj} < P_{d,j-1} \\ Y_{c,i-1;d,j-1} \mid \text{jumping does not occur} & P_{dj} = P_{ci} \quad ci \neq dj \\ & \text{and} \quad P_{ci} > P_{c,i-1} \\ & \text{and} \quad P_{dj} > P_{d,j-1} \\ Y_{ci;d,j-1} \mid \text{jumping occurs} & P_{dj} = P_{ci} \quad ci \neq dj \\ & \text{and} \quad P_{ci} > P_{c,i-1} \\ & \text{and} \quad P_{dj} > P_{d,j-1} \\ Y_{ci;d,j-1} & P_{dj} < P_{ci}. \end{cases} \tag{2.11}$$

The initial condition is $Y_{c0;dj} = 0$ for $1 \leq c \leq C$, $1 \leq d \leq C$, and $0 \leq j \leq T_d$.

Proof

The verification of Equation 2.11 is similar to that done for Equation 2.7. When $P_{dj} > P_{ci}$ no tasks dj will be serviced during the time interval $(t_{c,i-1},\ t_{ci}]$, since the priority of task ci is higher than that of tasks dj. This means that the number of tasks dj from the late arrivals that have been serviced by time $t_{c,i-1}$ is equal to the number that have been serviced by time t_{ci}. As a result, $Y_{ci;dj} = Y_{c,i-1;dj}$. When $P_{dj} = P_{ci}$, there are two possibilities: either tasks dj will jump over tagged task ci, or they will not. If they do not jump, the number of tasks dj that run before the tagged task ci is equal to the number of tasks $[d, j-1]$ that run before the tagged task $[c, i-1]$. The result is $Y_{ci;dj} = Y_{c,i-1;d,j-1}$. If $ci = dj$, the two tasks are similar and jumping will not occur. Two other conditions that will prevent jumping from occurring

are $P_{ci} < P_{c,i-1}$ and $P_{dj} < P_{d,j-1}$. In both cases, these tasks (ci and dj respectively) enter empty queues. In order for jumping to be possible, a task dj must find the tagged task ci waiting (not preempted), when it enters the queue. Either of these two conditions will prevent that situation from occurring. More than one of the three conditions above can be true for a particular ci and dj. If $ci \neq dj$, $P_{ci} > P_{c,i-1}$, and $P_{dj} > P_{d,j-1}$, there is a chance that an arriving task dj will jump over tagged task ci. When jumping occurs, the queueing system behaves as if $P_{dj} < P_{ci}$. When $P_{dj} < P_{ci}$, a task dj will receive service before time t_{ci} if its predecessor task $[d, j-1]$ receives service before t_{ci}. Therefore, the number of tasks dj that will receive service before time t_{ci} ($Y_{ci;dj}$) is equal to the number of predecessor tasks $[d, j-1]$ that receive service before t_{ci} ($Y_{ci;d,j-1}$). The resulting equation is $Y_{ci;dj} = Y_{ci;d,j-1}$. This portion of the equation will increase the sojourn time of the tagged task, to account for some late tasks dj receiving service prior to the tagged task, when ordinarily they would not have. This only occurs with certain configurations of the groups of task types in the queue. That is, the arriving tasks dj must find one or more tasks of their type in front of the tagged task, when they arrive. If this does not happen, jumping will not occur and $Y_{ci;dj} = Y_{c,i-1;d,j-1}$ (the reasoning for which has already been discussed). □

Like Equation 2.7, the only possibilities not covered by Equation 2.11 are when $P_{ci} = P_{c,i-1}$ and/or when $P_{dj} = P_{d,j-1}$. A problem arises when either

of these two equations are true. Again, the problem is due to tasks jumping other tasks from the same job type. Consider the example of a four task job with the set of priorities (1, 2, 2, 1). The problem tasks are the second and third task of this job. A second task has the opportunity to get ahead of third tasks, without a third task having the chance to jump second tasks. Late arriving second tasks may jump the tagged task, if the tagged task is the third task (and it is waiting behind other second tasks). However, late arriving third tasks cannot jump the tagged task, if the tagged task is the second task, since the second task of these late jobs must stay behind the tagged task. In other words, there are no late arriving third tasks when the tagged task is the second task. All of the second tasks of these late jobs must be served after the tagged task, preventing their third tasks from obtaining ready-for-service status and joining the queue. This is another imbalance that is caused by consecutive tasks of equal priority.

Tasks Served: Expected Value Equation

Forming the expectation of both sides of Equation 2.11 and introducing probability terms results in the following equation.

$$
\begin{aligned}
E[Y_{ci;dj}] \;=\;& E[Y_{c,i-1;dj} \mid_{P_{dj}>P_{ci}}] Pr_{P_{dj}>P_{ci}} \\
&+ E[Y_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci=dj}] Pr_{P_{dj}=P_{ci},\ ci=dj} \\
&+ E[Y_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}<P_{c,i-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}<P_{c,i-1}} \\
&+ E[Y_{c,i-1;d,j-1} \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}<P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}<P_{d,j-1}} \\
&+ E[Y_{ci;d,j-1}(\text{jumping}) \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}} \\
&+ E[Y_{c,i-1;d,j-1}(\text{no jumping}) \mid_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}}] \\
&\quad * Pr_{P_{dj}=P_{ci},\ ci\neq dj,\ P_{ci}>P_{c,i-1},\ P_{dj}>P_{d,j-1}} \\
&+ E[Y_{ci;d,j-1} \mid_{P_{dj}<P_{ci}}] Pr_{P_{dj}<P_{ci}} \qquad (2.12)
\end{aligned}
$$

These probability terms are the same as those found in Equation 2.8. Once again, it is easier to work with the equation in its decomposed form for computational purposes, than to calculate the probabilities. The result is the following equation:

$$
L_{ci;dj} = \begin{cases}
L_{c,i-1;dj} & P_{dj} > P_{ci} & \\
L_{c,i-1;d,j-1} & P_{dj} = P_{ci} & ci = dj \\
 & \text{or} & P_{ci} < P_{c,i-1} \\
 & \text{or} & P_{dj} < P_{d,j-1} \\
\Theta_{dj;ci} L_{ci;d,j-1} \mid \text{jumping occurs} & P_{dj} = P_{ci} & ci \neq dj \\
 & \text{and} & P_{ci} > P_{c,i-1} \\
 & \text{and} & P_{dj} > P_{d,j-1} \\
(1 - \Theta_{dj;ci}) L_{c,i-1;d,j-1} \mid \text{jumping does not occur} & & \\
 & P_{dj} = P_{ci} & ci \neq dj \\
 & \text{and} & P_{ci} > P_{c,i-1} \\
 & \text{and} & P_{dj} > P_{d,j-1} \\
L_{ci;d,j-1} & P_{dj} < P_{ci}. &
\end{cases} \qquad (2.13)
$$

where $L_{ci;dj}$ = the expected number of tasks dj from the late arrivals that are serviced by time t_{ci}.

The first initial condition is $L_{c0;dj} = 0$ for $1 \leq c \leq C$, $1 \leq d \leq C$, and $0 \leq j \leq T_d$. The second is $L_{ci;d0} = \lambda_d S_{ci}$ for $1 \leq c \leq C$, $1 \leq i \leq T_c$, and $1 \leq d \leq C$.

All other equations are identical to those of the previous section. Including the final equation $S_{ci} = S_{E_{ci}} + S_{A_{ci}} + S_{L_{ci}}$, which gives a linear system of equations of the form $S = aS + b$.

2.2.3 Example Problem

To illustrate how the linear system of equations are arrived at, the equations for an example problem are developed. Consider a problem with two job classes that have the priority vectors $P_1 = (1,\ 2,\ 3)$ and $P_2 = (1,\ 2)$. Using Equation 2.9, the matrix E can be determined (see Equation 2.15). Like E, the matrix L can be determined (see Equation 2.16) from initial conditions using Equation 2.13. The elements of the L matrix can be used in Equation 2.5 to find the elements of the matrix S_L. The resulting matrix can be put into the form $S_L = aS$ (see Equation 2.17 for the values of a). The next array to be calculated, is that of S_A. Using Equation 2.4, the following results

$$\begin{bmatrix} S_{A_{11}} \\ S_{A_{12}} \\ S_{A_{13}} \\ S_{A_{21}} \\ S_{A_{22}} \end{bmatrix} = \begin{bmatrix} \bar{x}_{11} \\ \bar{x}_{11} + \bar{x}_{12} \\ \bar{x}_{11} + \bar{x}_{12} + \bar{x}_{13} \\ \bar{x}_{21} \\ \bar{x}_{21} + \bar{x}_{22} \end{bmatrix} \tag{2.14}$$

Priority	1	2	1	1	3	1
$\bar{x}$	1	1	1	1	1	1
$E[x^2]$	1	1	1	1	1	1
λ	0.1			0.1		
Sojourn Time No Jumping	1.2500	3.646	4.646	1.2500	7.083	8.083
Sojourn Time Jumping	1.2500	3.646	4.646	1.2500	7.083	8.083
Sojourn Time Simulation	1.2501	3.642	4.642	1.2497	7.080	8.080
95% Confidence Interval	0.0018	0.015	0.015	0.0015	0.060	0.060

Table 2.1: Example 1

Then using Equation 2.1, values for the matrix S_E can be found. The matrix S_E can be put into the form $S_E = bS + c$. See Equation 2.18 for the values of b and Equation 2.19 for the values of c, where $\Omega_{xy} = \delta_{xy}(r_{xy} - \bar{x}_{xy})$. Finally, summing up S_E, S_A, and S_L, gives an equation for S that is of the form $S = dS + e$. See Equation 2.20 for the values of d ($\sum \rho = \rho_{11} + \rho_{12} + \rho_{13} + \rho_{21} + \rho_{22}$) and Equation 2.21 for the values of e.

2.2.4 Numerical Examples

Four numerical examples were solved, the results of which are shown in Tables 2.1-2.4. The task sojourn times obtained by the model, have been compared to those obtained by a strict FCFS model, in order to learn what effect jumping had on waiting times. Both models require the following input data: the number of classes, the number of tasks in each class, the priority of each task, the average service time for each task, the second moment of

$$E = \begin{bmatrix} 0 & \lambda_1 S_{12} & \lambda_1 S_{13} & \Theta_{11;21}\lambda_2 S_{11} & \lambda_2 S_{12} \\ 0 & 0 & \lambda_1 S_{13} & 0 & \Theta_{12;22}\lambda_2 S_{22} + (1-\Theta_{12;22})\Theta_{11;21}\lambda_2 S_{21} \\ 0 & 0 & 0 & 0 & 0 \\ \Theta_{21;11}\lambda_1 S_{11} & \lambda_1 S_{12} & \lambda_1 S_{13} & 0 & \lambda_2 S_{22} \\ 0 & \Theta_{22;12}\lambda_1 S_{12} + (1-\Theta_{22;12})\Theta_{21;11}\lambda_1 S_{11} & \lambda_1 S_{13} & 0 & 0 \end{bmatrix} \tag{2.15}$$

$$L = \begin{bmatrix} 0 & 0 & 0 & \Theta_{21;11}\lambda_2 S_{11} & 0 \\ \lambda_1 S_{12} & 0 & 0 & \lambda_2 S_{12} & \Theta_{22;12}\lambda_2 S_{12} + (1-\Theta_{22;12})\Theta_{21;11}\lambda_2 S_{11} \\ \lambda_1 S_{13} & \lambda_1 S_{13} & 0 & \lambda_2 S_{13} & \lambda_2 S_{13} \\ \Theta_{11;21}\lambda_1 S_{21} & 0 & 0 & 0 & 0 \\ \lambda_1 S_{12} & \Theta_{12;22}\lambda_1 S_{22} + (1-\Theta_{12;22})\Theta_{11;21}\lambda_1 S_{21} & 0 & \lambda_2 S_{22} & 0 \end{bmatrix} \tag{2.16}$$

$$a = \begin{bmatrix} \Theta_{21;11}\rho_{21} & 0 & 0 & 0 & 0 \\ (1-\Theta_{22;12})\Theta_{21;11}\rho_{22} & \rho_{11}+\rho_{21}+\Theta_{22;12}\rho_{22} & 0 & 0 & 0 \\ 0 & 0 & \rho_{11}+\rho_{12}+\rho_{21}+\rho_{22} & 0 & 0 \\ 0 & 0 & 0 & \Theta_{11;21}\rho_{11} & 0 \\ 0 & \rho_{11} & 0 & (1-\Theta_{12;22})\Theta_{11;21}\rho_{12} & \Theta_{12;22}\rho_{12}+\rho_{21} \end{bmatrix} \tag{2.17}$$

$$b = \begin{bmatrix} \rho_{11} & 0 & 0 & (1-\Theta_{11;21})\rho_{21} & 0 \\ \rho_{11} & \rho_{12} & 0 & \rho_{21}-(1-\Theta_{12;22})\Theta_{11;21}\rho_{22} & (1-\Theta_{12;22})\rho_{22} \\ \rho_{11} & \rho_{12} & \rho_{13} & \rho_{21} & \rho_{22} \\ (1-\Theta_{21;11})\rho_{11} & 0 & 0 & \rho_{21} & 0 \\ \rho_{11}-(1-\Theta_{22;12})\Theta_{21;11}\rho_{12} & (1-\Theta_{22;12})\rho_{12} & 0 & \rho_{21} & \rho_{22} \end{bmatrix} \quad (2.18)$$

$$c = \begin{bmatrix} \Omega_{11}+\Omega_{21} \\ \Omega_{11}+\Omega_{12}+\Omega_{21}+\Omega_{22} \\ \Omega_{11}+\Omega_{12}+\Omega_{13}+\Omega_{21}+\Omega_{22} \\ \Omega_{11}+\Omega_{21} \\ \Omega_{11}+\Omega_{12}+\Omega_{21}+\Omega_{22} \end{bmatrix} \quad (2.19)$$

$$d = \begin{bmatrix} \rho_{11}+\Theta_{21;11}\rho_{21} & 0 & 0 & (1-\Theta_{11;21})\rho_{21} & 0 \\ \rho_{11}+(1-\Theta_{22;12})\Theta_{21;11}\rho_{22} & \rho_{11}+\rho_{12}+\rho_{21}+\Theta_{22;12}\rho_{22} & 0 & \rho_{21}-(1-\Theta_{12;22})\Theta_{11;21}\rho_{22} & (1-\Theta_{12;22})\rho_{22} \\ \rho_{11} & \rho_{12} & \sum\rho & \rho_{21} & \rho_{22} \\ (1-\Theta_{21;11})\rho_{11} & 0 & 0 & \Theta_{11;21}\rho_{11}+\rho_{21} & 0 \\ \rho_{11}-(1-\Theta_{22;12})\Theta_{21;11}\rho_{12} & \rho_{11}+(1-\Theta_{22;12})\rho_{12} & 0 & (1-\Theta_{12;22})\Theta_{11;21}\rho_{12}+\rho_{21} & \Theta_{12;22}\rho_{12}+\rho_{21}+\rho_{22} \end{bmatrix} \quad (2.20)$$

$$e = c + \begin{bmatrix} \bar{x}_{11} \\ \bar{x}_{11}+\bar{x}_{12} \\ \bar{x}_{11}+\bar{x}_{12}+\bar{x}_{13} \\ \bar{x}_{21} \\ \bar{x}_{21}+\bar{x}_{22} \end{bmatrix} \quad (2.21)$$

Priority	1	2	1	1	3	1
$\bar{x}$	2	2	2	1	1	1
$E[x^2]$	4	4	4	1	1	1
λ	0.1			0.1		
Sojourn Time No Jumping	2.7143	14.76	16.76	1.7143	81.7	82.7
Sojourn Time Jumping	2.6857	14.76	16.76	1.7714	81.7	82.7
Sojourn Time Simulation	2.6829	14.78	16.78	1.7728	82.0	83.0
95% Confidence Interval	0.0043	0.16	0.16	0.0048	3.2	3.2

Table 2.2: Example 2

the service time for each task, and the traffic intensity for each class. The numbers used in the example problems were chosen so as to demonstrate the changes in sojourn time with changes in priority, service time, and arrival rate. Data that has been obtained from industry, is used in problems in Chapter 4. The first example involves two job types with three tasks each. Table 2.1 shows that jumping will have no effect on sojourn times if the service times and arrival rates of the various tasks in the queue are the same. When the service times of the tasks differ (Table 2.2), there is a difference in the sojourn times. In this case the changes are clearly seen: the first task of the job with the higher service time experienced a decrease in its sojourn time, while the first task of the job with the lower service time experienced an increase in its sojourn time. Although it is not apparent from Equation 2.10, increasing the service time of a task, changes the values of Θ that involve this task. In Table 2.1, the values of Θ between the first tasks of the two jobs

are equal ($\Theta_{t(1);t(4)} = \Theta_{t(4);t(1)} = 0.5$). However, the increase in the service time for the first task in the first job in Table 2.2 has resulted in these values changing ($\Theta_{t(1);t(4)} = 0.66667$ and $\Theta_{t(4);t(1)} = 0.33333$). Increasing a task's service time, increases the probability that it will jump, resulting in a lower sojourn time at the expense of others. This is similar to the results of the model in the previous section. Again we see that there is conservation of work. The first job's tasks gain one half (2.7143-2.6857=0.0286) of what the second job's tasks lose (1.7143-1.7714=-0.0571). The two to one ratio here, is due to the two to one ratio of the thetas. These gains and losses are not restricted to first tasks or to the priority one queue. Other tasks and other priority queues are involved when problems involving jobs with different priority vectors are used. Similar to differences in service time, differences in arrival rates also affect Θ and subsequently sojourn time. Table 2.3 shows these differences for the simplest of examples; two jobs having one task each. The first job has twice the arrival rate of the second (0.6 compared to 0.3), which results in it having a much lower sojourn time when jumping exists (4.4 compared to the other job's 7.7). When jumping does not exist, these jobs have the same sojourn time (5.5). Simulation of the modified FCFS service discipline was also done for these example problems. The simulation software used was PROPHET version 1.2.3 (Bell-Northern Research in-house software). The program PROPHET is an umbrella program that includes the task oriented simulation language (TOSIL) and the scheduler language

Priority	1	1
$\bar{x}$	1	1
$E[x^2]$	1	1
λ	0.6	0.3
Sojourn Time No Jumping	5.500	5.50
Sojourn Time Jumping	4.400	7.70
Sojourn Time Simulation	4.489	7.38
95% Confidence Interval	0.056	0.11

Table 2.3: Example 3

Priority	1	1
$\bar{x}$	2	1
$E[x^2]$	4	1
λ	0.3	0.3
Sojourn Time No Jumping	9.50	8.50
Sojourn Time Jumping	8.00	11.50
Sojourn Time Simulation	7.67	11.87
95% Confidence Interval	0.13	0.26

Table 2.4: Example 4

(SHEL) [2]. The runs were done on a Sun SparcStation 2 (with Sun operating system). Table 2.2 shows that the simulation values compare closely with those obtained via the model. However, the same comparison in Table 2.3 shows that the model slightly overestimates the effects of jumping. Making the same comparison in Table 2.4 shows the model to slightly underestimate the effects of jumping. In Table 2.4 the differences are caused by a difference in service time. The arrival rate and the service time are the only variables which can affect Θ (see Equation 2.10), and consequently are the only variables which can affect the sojourn times of tasks in this manner. Many more simulation runs were done, with problems involving a variety of task priorities, service times, arrival rates, and job class numbers (see Tables 5.15–5.36 in the Appendix). Specifically, 44 more simulation runs were done; 11 problems run 4 times each, varying the number of batches and the batch length with each run. The number of batches was either 25 or 50, and the

batch length was either 100,000 or 200,000. Four runs were also done for example problems 1–4. The values shown in Tables 2.1–2.4 are the averages of the four different runs. The individual results for each of the runs can be found in the Appendix (see Tables 5.1–5.8). A warmup of 200,000 was used in all the runs. The execution time (real time) of the simulation runs varied from 15 minutes to over an hour. Most of the runs took about 30 minutes to execute. The analytical model, however, took only a few seconds to run (2 or 3). When the sojourn times of the tasks obtained by simulation was compared to those obtained by the model, the results were similar to example problems 1–4. That is to say, that some were slightly under, some were slightly over, and some were right on (with one poor result being the exception). In general, the approximation of Θ is sufficiently accurate for the model to be used in telephone switch design.

This model's modified FCFS queueing discipline also benefits frequently arriving messages that have a contention protocol. Acknowledgement signals that indicate that the incoming data has transferred from hardware into memory, are less likely to time out for frequently arriving messages. However, not all tasks are affected by the jumping mechanism. Any task which has a higher priority than its preceding task (it therefore cannot be a job's first task), experiences neither an increase nor a decrease in its sojourn time.

Chapter 3

Multiple Jumping and Forking

In addition to jumping, tasks in telephone switches also fork into multiples of other tasks types when they have completed service. The model presented in the last section of the previous chapter, has been further developed to include task forking. In order to do so, substantial changes were needed to be made to the equations and variables. For example, a third index to identify tasks was needed. Due to these changes, a full discussion of the equations is given.

3.1 Sojourn Times

Consider a system with C classes of jobs, where every job in class c $(1 \leq c \leq C)$ has T_c levels of tasks. Figure 3.1 shows a job with three task levels. The priority of the tasks in a class c job, will be denoted P_{cik}. The second subscript i denotes the task level, while the third subscript k identifies a particular task at this level. In Figure 3.1, the two tasks at level two are

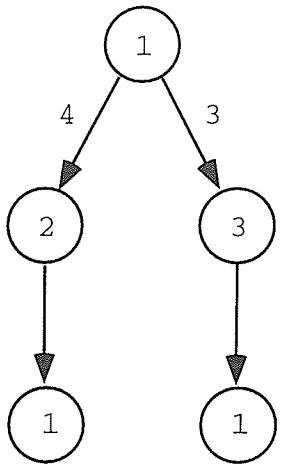

Figure 3.1: Forking example.

due to forking at level one. The number of tasks created by the forking are shown along the arrows joining the circles. The first task in this job, will be followed by four tasks of priority two and three tasks of priority three. When the first task has completed service, four tasks of priority two and three tasks of priority three enter the priority queues. Each of the tasks created by forking will spawn another task of priority one when it has completed service. Once separate paths have been created by forking, they remain that way (no merging).

Like in previous models, equations are developed to describe the influence tasks djm have on the tagged task cik. The indices for task djm are defined similar to those for task cik. The first subscript d denotes the class of the job, the second subscript j denotes the task level, and the third subscript m

identifies the task at the level. These equations will be used to determine what effect all possible tasks djm have on any tagged task cik. The equations are derived, by tagging the tasks of an arbitrary arriving class c job, and computing each task's average sojourn time.

Let the term t_{c0} denote the arrival time of the tagged class c job, and the term t_{cik} denote the time at which service of the tagged task cik is completed. The average sojourn time for task cik, S_{cik}, is defined to be the expected value of the quantity $t_{cik} - t_{c0}$. The average sojourn time for a task, is found by summing up delays due to other tasks arriving prior to t_{c0}, at t_{c0}, and after t_{c0}.

3.1.1 Early Tasks

The expected delay to task cik due to early arrivals (tasks arriving prior to t_{c0}) is denoted $S_{E_{cik}}$, and is equal to the time required to service those early tasks that received service during the time interval $(t_{c0},\ t_{cik}]$.

Remaining Tasks: Random Variable Equation

In order to develop the equation for $S_{E_{cik}}$, two random variables must be defined.

$X_{c0;djm}$ = the number of early tasks djm found in the system by the arriving tagged class c job.
$X_{cik;djm}$ = the number of early tasks djm that are left in the system at time t_{cik}.

Proposition 1

The equation relating the two variables, is the following:

$$X_{cik;djm} = \begin{cases} X_{c,i-1,x;djm} & P_{djm} > P_{cik} \\ \Psi_{djm} X_{c,i-1,x;d,j-1,y} & P_{djm} = P_{cik} \quad cik = djm \\ & \quad \text{or} \quad P_{cik} < P_{c,i-1,x} \\ & \quad \text{or} \quad P_{djm} < P_{d,j-1,y} \\ \Psi_{djm} X_{c,i-1,x;d,j-1,y} \mid \text{jumping does not occur} & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ X_{c,i-1,x;djm} \mid \text{jumping occurs} & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ \Psi_{djm} X_{cik;d,j-1,y} & P_{djm} < P_{cik} \end{cases} \tag{3.1}$$

where $c, i-1, x =$ the indices of the parent task of the tagged task cik.

$d, j-1, y =$ the indices of the parent task of the task djm.

$\Psi_{djm} =$ the number of djm tasks created by a single parent task.

The initial condition $X_{cik;d0} = 0$ for $1 \leq c \leq C$, $0 \leq i \leq T_c$, all k, and $1 \leq d \leq C$.

Proof

The verification of Equation 3.1 is as follows. When $P_{djm} > P_{cik}$ no tasks djm will be serviced during the time interval $(t_{c,i-1,x},\ t_{cik}]$. That is to say that the number of early tasks djm that are present at time $t_{c,i-1,x}$, are also present at time t_{cik}. As a result, $X_{cik;djm} = X_{c,i-1,x;djm}$. When $P_{djm} = P_{cik}$, there are two possibilities: either the task cik will jump over tasks djm, or it will not. If it does not jump, any task djm with a priority equal to that of task cik and which has not yet joined the ready-for-service queue by time $t_{c,i-1,x}$,

will still be in the system at time t_{cik}. These tasks will have predecessor tasks $[d, j-1, y]$ that are still in the system at time $t_{c,i-1,x}$. The number of predecessor tasks in the system at this time, will equal $1/\Psi_{djm}$ times the number of tasks djm that are still in the system at time t_{cik}. As a result, $X_{cik;djm} = \Psi_{djm} X_{c,i-1,x;d,j-1,y}$. If $cik = djm$, the two tasks are similar and jumping will not occur. Two other conditions will ensure that jumping does not occur are $P_{cik} < P_{c,i-1,x}$ and $P_{djm} < P_{d,j-1,y}$. In both cases, these tasks (cik and djm respectively) enter empty queues. In order for jumping to be possible, the tagged task cik must find a task (or group of tasks) djm waiting, when it enters the queue (as well as a group of its own type to jump to). If any one of these three conditions are true, jumping will not occur (more than one can be true for a particular cik and djm). If $cik \neq djm$, $P_{cik} > P_{c,i-1,x}$, and $P_{djm} > P_{d,j-1,y}$, there is a chance that tagged task cik will jump over tasks djm. When this happens, no tasks djm will be serviced during the time interval $(t_{c,i-1,x},\ t_{cik}]$. This situation is similar to what happens when $P_{djm} > P_{cik}$. It has been shown that in this case, $X_{cik;djm} = X_{c,i-1,x;djm}$. This portion of the equation will produce a decrease in the sojourn time of the tagged task, to account for the fact that the tagged task need not always wait for earlier tasks djm to be serviced before receiving service itself. This only occurs with certain configurations of the groups of task types in the queue. That is, the arriving task cik must find one or more tasks of its type in front of earlier tasks djm, when it arrives. If this does not happen, jumping will not

occur and $X_{cik;djm} = \Psi_{djm} X_{c,i-1,x;d,j-1,y}$ (the reasoning for which has already been discussed). When $P_{djm} < P_{cik}$, only those tasks djm whose predecessor tasks $[d, j-1, y]$ have not been serviced by time t_{cik} will still be in the system at t_{cik}. The number of tasks $d, j-1, y$ which have not been serviced by time t_{cik} is equal to $X_{cik;d,j-1,y}$. The result is $X_{cik;djm} = \Psi_{djm} X_{cik;d,j-1,y}$. □

The only possibilities not covered by Equation 3.1 are when $P_{cik} = P_{c,i-1,x}$ and/or when $P_{djm} = P_{d,j-1,y}$. A problem arises when either of these two equations are true. The problem is due to tasks jumping other tasks within the same job type. This would happen, for example, in the job shown in Figure 3.1 if the priority of task [1,3,1] would be 2 instead of 1. The problem tasks would then be [1,2,1] and [1,3,1] of this job. A second level task has the opportunity to get ahead of third level tasks, without a third level task having the chance to jump second level tasks. A tagged second level task has the opportunity to jump third level tasks that are early arrivals, since it can find these tasks in the queue upon its arrival. However, a tagged third level task will find no second level tasks that are early arrivals upon its arrival, and therefore cannot reciprocate the action. All early arriving second level tasks must be served before the tagged second level task.

Remaining Tasks: Expected Value Equation

Forming the expectation of both sides of Equation 3.1 and introducing probability terms results in the following equation.

$$
\begin{aligned}
E[X_{cik;djm}] \;=\; & E[X_{c,i-1,x;djm} \mid_{P_{djm}>P_{cik}}] Pr_{P_{djm}>P_{cik}} \\
& + E[\Psi_{djm} X_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik=djm}] Pr_{P_{djm}=P_{cik},\ cik=djm} \\
& + E[\Psi_{djm} X_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}<P_{c,i-1,x}}] \\
& \quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}<P_{c,i-1,x}} \\
& + E[\Psi_{djm} X_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}<P_{d,j-1,y}}] \\
& \quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}<P_{d,j-1,y}} \\
& + E[X_{c,i-1,x;djm}(\text{jumping}) \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}}] \\
& \quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}} \\
& + E[\Psi_{djm} X_{c,i-1,x;d,j-1,y}(\text{no jumping}) \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}}] \\
& \quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}} \\
& + E[\Psi_{djm} X_{cik;d,j-1,y} \mid_{P_{djm}<P_{cik}}] Pr_{P_{djm}<P_{cik}} \qquad (3.2)
\end{aligned}
$$

where $Pr_{P_{djm}>P_{cik}}$ = the probability of the priority of djm being larger than the priority of cik.

Some of the probability terms arise from $P_{djm} > P_{cik}$, $P_{djm} = P_{cik}$, and $P_{djm} < P_{cik}$. The other terms arise from $cik = djm$ and $cik \neq djm$, $P_{cik} < P_{c,i-1,x}$ and $P_{cik} > P_{c,i-1,x}$, and $P_{djm} < P_{d,j-1,y}$ and $P_{djm} > P_{d,j-1,y}$. These five inequalities and one equation, create eight possible combinations for tasks cik and djm when $P_{djm} = P_{cik}$. In seven of the eight combinations, jumping will not occur. The term $Pr_{P_{djm}=P_{cik},\ cik=djm}$ accounts for four of these, $Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}<P_{c,i-1,x}}$ accounts for two more, and $Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}<P_{d,j-1,y}}$ accounts for the last. Jumping is possible with the remaining combination; when $cik \neq djm$, $P_{cik} > P_{c,i-1,x}$,

and $P_{djm} > P_{d,j-1,y}$. For any cik and djm, all but one of the above probability terms will be equal to zero, with the remaining term equal to one.

It is easier to work with the equation in its decomposed form for computational purposes, than to calculate the probabilities. The result is the following equation:

$$E_{cik;djm} = \begin{cases} E_{c,i-1,x;djm} & P_{djm} > P_{cik} \\ \Psi_{djm} E_{c,i-1,x;d,j-1,y} & P_{djm} = P_{cik} \quad cik = djm \\ & \quad \text{or} \quad P_{cik} < P_{c,i-1,x} \\ & \quad \text{or} \quad P_{djm} < P_{d,j-1,y} \\ \Theta_{cik;djm} E_{c,i-1,x;djm} \mid \text{jumping occurs} & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ (1 - \Theta_{cik;djm}) \Psi_{djm} E_{c,i-1,x;d,j-1,y} \mid \text{jumping does not occur} & \\ & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ \Psi_{djm} E_{cik;d,j-1,y} & P_{djm} < P_{cik}. \end{cases} \tag{3.3}$$

where $E_{cik;djm}$ = the expected number of early tasks djm that are left in the system at time t_{cik}.

$\Theta_{cik;djm}$ = the probability of tagged task cik jumping tasks djm.

The first initial condition $E_{cik;d0}$ is equal to zero for $1 \leq c \leq C$, $0 \leq i \leq T_c$, all k, and $1 \leq d \leq C$. The second initial condition is $E_{c0;djm} = \beta_{djm} \lambda_d S_{djm}$ for $1 \leq c \leq C$, $1 \leq d \leq C$, $1 \leq j \leq T_d$, and all m. The term β_{djm} reflects the multiples of λ_d which have been brought about by forking. For example, $\beta_{1,2,1}$ for task [1,2,1] in Figure 3.1 is equal to 4. It should be noted here, that the value for this term accumulates all the forking which has affected the

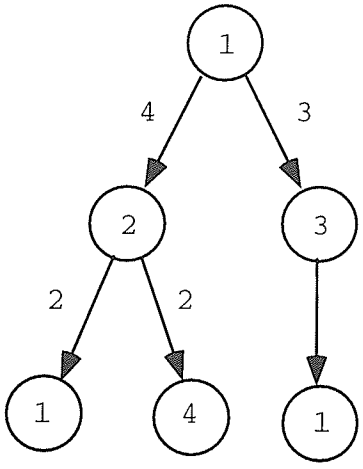

Figure 3.2: Double forking.

arrival rate for task djm. For example, $\beta_{1,3,1}$ for task [1,3,1] in Figure 3.2 is equal to 8. This is determined by multiplying the forking factors (4x2=8).

Contribution To Sojourn Time

It is now possible to calculate the expected number of early tasks djm that have been served during the time period $(t_{c0}, t_{cik}]$. This number is equal to $E_{c0;djm} - E_{cik;djm}$. It is the number of tasks that are found in the system at t_{c0}, subtracted by the number of tasks that are left in the system at t_{cik}. Since $S_{E_{cik}}$ is the delay to task cik, this difference must be multiplied by the expected service time for task djm. In addition, a term must be included to take into account the possibility that the arriving task cik will find one of these tasks either in service or in a preempted state (the term subtracts the

service that this task has already received). Summing the effect of all of the task types, results in the following equation:

$$S_{E_{cik}} = \sum_{djm; E_{c0;djm} \neq E_{cik;djm}} (E_{c0;djm} - E_{cik;djm})\bar{x}_{djm} - \delta_{djm}(\bar{x}_{djm} - r_{djm}). \quad (3.4)$$

where $\bar{x}_{djm}$ = the expected service time of task djm.

δ_{djm} = the probability that an arbitrary arriving job finds a task djm either in service or in a preempted state.

r_{djm} = the mean residual life for task djm.

A value for δ_{djm} can be determined through the use of the expression

$$\delta_{djm} = \frac{\rho_{djm}}{1 - \sum_{c=1}^{C} \sum_{i=1}^{T_c} \sum_{\substack{k \\ P_{cik} < P_{djm} \\ P_{cxy} < P_{djm}}} \rho_{cik}}. \quad (3.5)$$

where ρ = the traffic intensity $(\beta\lambda\bar{x})$.

cxy = the indices of any task which needs to be serviced before task cik.

This equation is the result of Little's result and the equation for the average completion time (from [44] pp. 64-65). The average completion time referred to here, is the amount of time that task djm is either in service or is in a preempted state. For the single class problem without forking, that time is equal to $\frac{\bar{x}_m}{1-\lambda\sum_k \bar{x}_k}$ ($P_k < P_m$ and $P_y < P_m$ where y is any task preceding k). Since this work deals with multiple classes, tasks from the other classes must be taken into account. The result, is an average completion time which

is equal to $\frac{\bar{x}_{dm}}{1-\sum_{c=1}^{C}\sum_{k}\lambda_c\bar{x}_{ck}}$ ($P_{ck} < P_{dm}$ and $P_{cy} < P_{dm}$ where cy is any task preceding ck). Finally, forking must be taken into account. The average completion time now becomes $\frac{\bar{x}_{djm}}{1-\sum_{c=1}^{C}\sum_{i=1}^{T_c}\sum_{k}\beta_{cik}\lambda_c\bar{x}_{cik}}$ ($P_{cik} < P_{djm}$ and $P_{cxy} < P_{djm}$ where cxy is any task preceding cik). The value for δ_{djm} is also the average number of tasks djm which are either in service or in a preempted state at an arbitrary point in time. This is due to the fact that there can only be zero or one of these tasks in such a state, at any given point in time. Thus, Little's result can be used to relate δ_{djm} to the average completion time, resulting in Equation 3.5. It should be noted here, that the arrival rate of task djm is equal to $\beta_{djm}\lambda_d$.

A value for r_{djm} can be found by using the equation $r_{djm} = E[x_{djm}^2]/2\bar{x}_{djm}$, where the first and second moments are known from the input data. The equation for r_{djm} is widely known and can be found in [30] (page 173).

All that remains in determining $S_{E_{cik}}$ is to find a value for $\Theta_{cik;djm}$. Derivation of the equation for $\Theta_{cik;djm}$ is similar to the derivation of the equation for $\Theta_{ci;dj}$.

$$\Theta_{t(1);t(2)} = \frac{1}{\sum_{q\in\widetilde{U}}\rho_{t(q)}}[\rho_{t(1)} + \sum_{n=3}^{N}\rho_{t(n)}\frac{\lambda_{t(1)}}{\sum_{\substack{m=1\\m\neq n}}^{N}\lambda_{t(m)}}[1 + \sum_{r=1}^{N-3}\sum_{P_s\in\widetilde{W}}\prod_{s=1}^{r}\frac{\lambda_{t(P_s)}}{\sum_{u\in\widetilde{Y}}\lambda_{t(u)}}]] \tag{3.6}$$

where $\widetilde{U} = \{fgh|\ P_{t(1)} = P_{t(2)} = P_{fgh},\ P_{fgh} > P_{f,g-1,z(\text{Parent task})}$ or $g = 1,\ 1 \le f \le C,\ 1 \le g \le T_f,$ and all $h\}$

$\widetilde{W} = \{2 < P_s \le N,\ P_s \neq n,\ P_s \neq P_{w-1}\ \ \forall w:\ 2 \le w \le s\}$

$$\widetilde{Y} = \{1 \leq u \leq N,\ u \neq n,\ u \neq P_w\ \ \forall w:\ 1 \leq w \leq s\}$$

3.1.2 Own Tasks

The delay to task cik due to arrivals at t_{c0} is denoted $Z_{A_{cik}}$, and can be calculated with the equation

$$Z_{A_{cik}} = \sum_{g=1}^{i-1} \widetilde{\omega}_{cgx} x_{cgx}^{t} + \frac{1}{\beta_{cik}} \sum_{u=1}^{\beta_{cik}} \sum_{t=1}^{u} x_{cik}^{t} + \frac{1}{\beta_{cik}} \sum_{\substack{g=i+1 \\ P_{cgy}<P_{cik} \\ P_{cvw}<P_{cik}}}^{T_c} \sum_{u=1}^{\beta_{cik}-1} \sum_{t=1}^{u} x_{cgy}^{t} + \sum_{g=2}^{T_c} \sum_{\substack{h \\ P_{cgh}\leq P_L \\ P_{cyz}\leq P_L \\ cgh\neq cgx \\ cgh\neq cik \\ cgh\neq cgy}} \sum_{t=1}^{\beta_{cgh}} x_{cgh}^{t}$$

$$+ \sum_{m=2}^{T_c} \sum_{\substack{n \\ P_{cmn}\leq P_L \\ P_{crs}\leq P_L \\ cmn\neq cgx \\ cmn\neq cik \\ cmn\neq cgy}} \sum_{t=1}^{\beta_{cmn}/2} x_{cmn}^{t}. \tag{3.7}$$

where $\widetilde{\omega}_{cgx} = \begin{cases} \sum_{t=1}^{\beta_{cgx}} & \text{if } P_{cgx} < P_{c,g+1,x} \\ \frac{1}{\beta_{cgx}} \sum_{u=1}^{\beta_{cgx}} \sum_{t=1}^{u} & \text{if } P_{cgx} > P_{c,g+1,x} \end{cases}$

x = the index of the task at level g which must be serviced before task cik and is in its path. There is one exception to this: x is equal to k when $g+1$ is equal to i (see conditions for $\widetilde{\omega}_{cgx}$).

cgy = the indices of any task that has the tagged task in its path and is executed after a cik task (it is further down the branch).

cvw = the indices of any task which lies between task cik and task cgy in the tree structure.

cyz = the indices of any task which needs to be serviced before task cgh and is in its path (excluding the first task of the job).

P_L = the lowest priority from the set of tasks including the tagged task and those that are in its path (the set being $cik + \sum_{g=1}^{i-1} cgx$). If there are several task types for which this is true, then the task type that is associated with P_L is the one that is the furthest down the path.

crs = the indices of any task which needs to be serviced before task cmn and is in its path (excluding the first task of the job).

Equation 3.7 deals with the delay to the tagged task that is caused by tasks which are in the same job as the tagged task. The terms in this equation sum the effect of those tasks, from the different parts of that job. The first term sums the delay caused by those tasks which are in the path of the tagged task, but are serviced before cik tasks (they are higher up in the branch). Task cgx is said to be in task cik's path, if task cik has task cgx in its lineage. This would be true if task cgx were the parent of task cik, or if it were the parent of the parent of task cik. The relationship could be a much more distant one. The second term sums the delay caused by those tasks which are the same as the tagged task. Since forking creates multiple tasks of the same type, the tagged task becomes one of many when it is that type.

The third term sums the delay caused by those tasks which have cik tasks in their path, but are serviced after these (cik) tasks (they are further down in the tree structure). Some of these tasks will have the opportunity for service before the tagged task, since some cik tasks will receive service before the tagged task. If the child task (or children if forking is involved) of

a *cik* task has a priority higher than that of its parent (*cik*), it too will delay the tagged task. This could move into deeper levels if the priorities permit, and with forking could involve a lot of tasks. The sojourn time of the tagged task must reflect this possibility, and the third term insures that it does.

The fourth and fifth terms sum the delay caused by those tasks which are neither in the path of the tagged task, nor do they have the tagged task in their path. If the tagged task is anything but the first task of the job, the tasks in these other branches will have the opportunity to receive service before the tagged task. The priorities of the tasks in these branches will determine whether or not they take advantage of that opportunity. Just like the third term, the algorithm considers all levels in the tree structure and any degree of forking for tasks in these branches as well.

The fourth term sums those tasks where all the tasks on the branch have a priority higher than or equal to P_L. For those tasks which have a priority equal to P_L, additional work is needed to determine whether or not these tasks joined the queue before the task associated with P_L. If these tasks joined the queue before the task associated with P_L, they are added to the sum (otherwise they are not). This is determined by examining the priorities of the tasks immediately prior to the two tasks in question. If the priority of the task prior to the task associated with P_L is lower than the other preceding task, then the service time of task (or group of tasks) *cgh* will be added to the sum. An example of a problem where this situation occurs is shown in

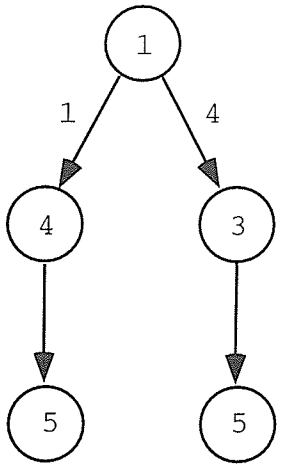

Figure 3.3: An example where branches contain equal priority tasks

Figure 3.3. In this problem, the tagged task would have to be [1,3,1] and the task *cgh* would have to be [1,3,2] for this situation to occur. Should there be more than one task type with the priority P_L in the tagged task's path, the task type associated with P_L must be the furthest task down the branch.

The fifth term is similar to the fourth term, with one exception. It deals with branches where a task has forked into two or more task types that each have the same priority. An example of this is shown in Figure 3.4. The problem occurs when the five tasks of priority two enter the queue. Their order not only determines how long tasks [1,2,1] and [1,2,2] will wait for service, but affects their succeeding tasks as well. If the tagged task is either [1,2,1], [1,3,1], [1,2,2], or [1,3,2], one or more of the tasks in the other branch could receive service before the tagged task (it depends on how the tasks

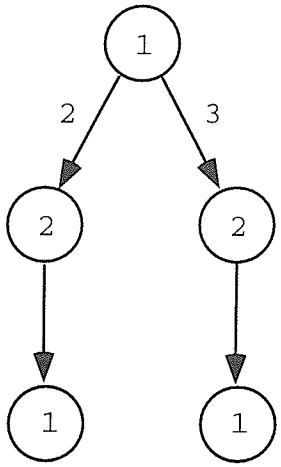

Figure 3.4: An example where a fork creates equal priority tasks

enter the queue when forking takes place). The tasks from the other branch, that enter the priority queue with the tagged task, will either be in front of or behind the tagged task. Assuming all the tasks created by a fork enter the queue in random order, the tasks from the other branch can expect to be in front of the tagged task half of the time. Branches that result from this type of fork, must involve the branch containing the tagged task. If the tagged task is not involved, the branch will be taken care of by the fourth term.

An equation involving expected values can be developed by forming the expectation of both sides of Equation 3.7.

$$S_{A_{cik}} = \sum_{g=1}^{i-1} \omega_{cgx}\bar{x}_{cgx} + \gamma_{cik}\bar{x}_{cik} + \sum_{\substack{g=i+1 \\ P_{cgy}<P_{cik} \\ P_{cvw}<P_{cik} \\ \gamma_{cik}>1}}^{T_c} (\gamma_{cik}-1)\bar{x}_{cgy} + \sum_{g=2}^{T_c} \sum_{\substack{h \\ P_{cgh}\leq P_L \\ P_{cyz}\leq P_L \\ cgh\neq cgx \\ cgh\neq cik \\ cgh\neq cgy}} \beta_{cgh}\bar{x}_{cgh}$$

$$+ \frac{1}{2}\sum_{m=2}^{T_c} \sum_{\substack{n \\ P_{cmn}\leq P_L \\ P_{crs}\leq P_L \\ cmn\neq cgx \\ cmn\neq cik \\ cmn\neq cgy}} \beta_{cmn}\bar{x}_{cmn}. \qquad (3.8)$$

where $S_{A_{cik}}$ = the expected delay to task cik due to arrivals at t_{c0}.

$$\omega_{cgx} = \begin{cases} \beta_{cgx} & \text{if } P_{cgx} < P_{c,g+1,x} \\ \gamma_{cgx} & \text{if } P_{cgx} > P_{c,g+1,x} \end{cases}$$

The term γ_{cik} (in Equation 3.8) is needed to account for the possibility of cik being one of several tasks created by a fork. The value of this variable is found by determining all the possible arrangements involving the tagged task and the other tasks having the same index (cik). Consider the example shown in Figure 3.5. When $i = 2$ and $k = 1$, the tagged task is one of two tasks. Whether it is the first or the second of these tasks to receive service depends on chance. Since each of these arrangements are equally likely, the tagged task can expect the server to service one and a half of these tasks before t_{cik}

$$\frac{\bar{x}_{cik}}{2} + \frac{2\bar{x}_{cik}}{2} = \frac{3\bar{x}_{cik}}{2} \qquad (3.9)$$

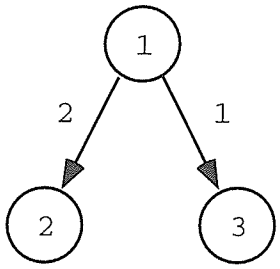

Figure 3.5: Fork with a branch of 2

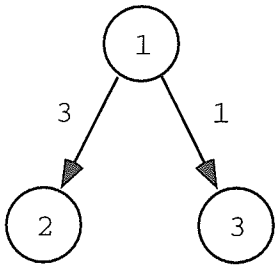

Figure 3.6: Fork with a branch of 3

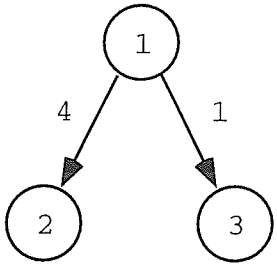

Figure 3.7: Fork with a branch of 4

The number of possible arrangements changes with a change in the number of tasks created by a fork. Consider the example shown in Figure 3.6. When $i = 2$ and $k = 1$, the tagged task is one of three tasks. Determining $\gamma_{cik}\bar{x}_{cik}$ for three tasks is similar to finding it for two tasks

$$\frac{\bar{x}_{cik}}{3} + \frac{2\bar{x}_{cik}}{3} + \frac{3\bar{x}_{cik}}{3} = \frac{6\bar{x}_{cik}}{3} \tag{3.10}$$

To confirm the pattern, a third example is considered in Figure 3.7. When $i = 2$ and $k = 1$, the tagged task is one of four tasks. Summing up the

probabilities for $\gamma_{cik}\bar{x}_{cik}$ gives the following:

$$\frac{\bar{x}_{cik}}{4} + \frac{2\bar{x}_{cik}}{4} + \frac{3\bar{x}_{cik}}{4} + \frac{4\bar{x}_{cik}}{4} = \frac{10\bar{x}_{cik}}{4}. \tag{3.11}$$

With the pattern clearly established, a general equation is possible using β_{cik}

$$\gamma_{cik} = \frac{1 + 2 + 3 + \cdots + \beta_{cik}}{\beta_{cik}} \tag{3.12}$$

In developing this equation, the examples contained only one fork. An example containing multiple forks should also be examined. Such an example has already been shown in Figure 3.2. Due to the double forking, the number of tasks with $i = 3$ and $k = 1$ is eight. Just how many of these tasks receive service before the tagged task is dependent on what happened to the previous tagged task ($i = 2$ and $k = 1$). If, for example, the tagged task at the second level received service first, the tagged task at the third level could receive service first, or second. If the tagged task at the second level received service second, the tagged task at the third level could receive service third, or fourth. Summing up all the possibilities and taking into account that the probability of each is the result of two events, gives the following equation

$$\begin{aligned}&\frac{1}{4}\left(\frac{\bar{x}_{cik}}{2} + \frac{2\bar{x}_{cik}}{2}\right) + \frac{1}{4}\left(\frac{3\bar{x}_{cik}}{2} + \frac{4\bar{x}_{cik}}{2}\right) + \frac{1}{4}\left(\frac{5\bar{x}_{cik}}{2} + \frac{6\bar{x}_{cik}}{2}\right) + \frac{1}{4}\left(\frac{7\bar{x}_{cik}}{2} + \frac{8\bar{x}_{cik}}{2}\right) \\ &\quad = \frac{1+2+3+4+5+6+7+8}{8}\bar{x}_{cik}\end{aligned} \tag{3.13}$$

This shows, that the equation for γ (Equation 3.12) can handle both single and multiple forking. One additional comment concerning the last example:

γ can also be arrived at by seeing the tagged task as one of the eight tasks, its position in the queue being anywhere from first to last, with equal probability.

A value for ω_{cgx} is found by comparing the priority of task cgx with that of task $[c, g+1, x]$. There are only two possibilities, either $P_{cgx} < P_{c,g+1,x}$ or $P_{cgx} > P_{c,g+1,x}$. In Figure 3.2 both of these possibilities exist. If i were to be 3 and k were to be 2, $\omega_{c,2,1}$ would need to be determined for the first summation in Equation 3.8. Since $P_{c,2,1} < P_{c,3,2}$, all of the $[c, 2, 1]$ tasks will be serviced before $t_{c,3,2}$. It doesn't matter whether the tagged task $[c, 2, 1]$ was served first or whether it was served last (fourth), the next tagged task $[c, 3, 2]$ will wait for all of the other $[c, 2, 1]$ tasks to be served before the server begins to serve the $[c, 3, 2]$ tasks. That is why, in this situation, $\omega_{cgx} = \beta_{cgx}$. If i were to be 3 and k were to be 1, $\omega_{c,2,1}$ would again need to be determined for the first summation in Equation 3.8. This situation is different, because $P_{c,2,1} > P_{c,3,1}$. All of the $[c, 2, 1]$ tasks do not have to be serviced before $t_{c,3,1}$. The only $[c, 2, 1]$ tasks that do need to be serviced are the ones in front of the previous tagged task ($i = 2$ and $k = 1$) and the previous tagged task itself. This number of tasks is equal to γ.

3.1.3 Late Tasks

The expected delay to task cik due to late arrivals (tasks arriving after t_{c0}) is denoted $S_{L_{cik}}$, and is equal to the time required to service late arriving tasks that received service during the time interval $(t_{c0},\ t_{cik}]$.

Tasks Served: Random Variable Equation

In order to develop the equation for $S_{L_{cik}}$, two random variables must be defined.

$Y_{cik;d0}$ = the number of late tasks djm that arrive before time t_{cik}.

$Y_{cik;djm}$ = the number of late tasks djm that are serviced by time t_{cik}.

Proposition 2

The equation relating the two variables, is the following:

$$Y_{cik;djm} = \begin{cases} Y_{c,i-1,x;djm} & P_{djm} > P_{cik} \\ \Psi_{djm} Y_{c,i-1,x;d,j-1,y} & P_{djm} = P_{cik} \quad cik = djm \\ & \quad \text{or} \quad P_{cik} < P_{c,i-1,x} \\ & \quad \text{or} \quad P_{djm} < P_{d,j-1,y} \\ \Psi_{djm} Y_{c,i-1,x;d,j-1,y} \mid \text{jumping does not occur} & \\ & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ \Psi_{djm} Y_{cik;d,j-1,y} \mid \text{jumping occurs} & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ \Psi_{djm} Y_{cik;d,j-1,y} & P_{djm} < P_{cik}. \end{cases} \tag{3.14}$$

The initial condition is $Y_{c0;djm} = 0$ for $1 \leq c \leq C$, $1 \leq d \leq C$, $0 \leq j \leq T_d$, and all m.

Proof

The verification of Equation 3.14 is similar to that done for Equation 3.1. When $P_{djm} > P_{cik}$ no tasks djm will be serviced during the time interval $(t_{c,i-1,x},\ t_{cik}]$, since the priority of task cik is higher than that of tasks djm.

This means that the number of tasks djm from the late arrivals that have been serviced by time $t_{c,i-1,x}$ is equal to the number that have been serviced by time t_{cik}. As a result, $Y_{cik;djm} = Y_{c,i-1,x;djm}$. When $P_{djm} = P_{cik}$, there are two possibilities: either tasks djm will jump over tagged task cik, or they will not. If they do not jump, the number of tasks djm that run before the tagged task cik is equal to Ψ_{djm} times the number of tasks $[d, j-1, y]$ that run before the tagged task $[c, i-1, x]$. The result is $Y_{cik;djm} = \Psi_{djm} Y_{c,i-1,x;d,j-1,y}$. If $cik = djm$, the two tasks are similar and jumping will not occur. Two other conditions that will prevent jumping from occurring are $P_{cik} < P_{c,i-1,x}$ and $P_{djm} < P_{d,j-1,y}$. In both cases, these tasks (cik and djm respectively) enter empty queues. In order for jumping to be possible, a task djm must find the tagged task cik waiting (not preempted), when it enters the queue. Either of these two conditions will prevent that situation from occurring. More than one of the three conditions above can be true for a particular cik and djm. If $cik \neq djm$, $P_{cik} > P_{c,i-1,x}$, and $P_{djm} > P_{d,j-1,y}$, there is a chance that an arriving task djm will jump over tagged task cik. When jumping occurs, the queueing system behaves as if $P_{djm} < P_{cik}$. When $P_{djm} < P_{cik}$, a task djm will receive service before time t_{cik} if its predecessor task $[d, j-1, y]$ receives service before t_{cik}. Therefore, the number of tasks djm that will receive service before time t_{cik} ($Y_{cik;djm}$) is equal to Ψ_{djm} times the number of predecessor tasks $[d, j-1, y]$ that receive service before t_{cik} ($\Psi_{djm} Y_{cik;d,j-1,y}$). The resulting equation is $Y_{cik;djm} = \Psi_{djm} Y_{cik;d,j-1,y}$. This portion of the

equation will increase the sojourn time of the tagged task, to account for some late tasks djm receiving service prior to the tagged task, when ordinarily they would not have. This only occurs with certain configurations of the groups of task types in the queue. That is, the arriving tasks djm must find one or more tasks of their type in front of the tagged task, when they arrive. If this does not happen, jumping will not occur and $Y_{cik;djm} = \Psi_{djm} Y_{c,i-1,x;d,j-1,y}$ (the reasoning for which has already been discussed). □

Like Equation 3.1, the only possibilities not covered by Equation 3.14 are when $P_{cik} = P_{c,i-1,x}$ and/or when $P_{djm} = P_{d,j-1,y}$. A problem arises when either of these two equations are true. Again, the problem is due to tasks jumping other tasks from the same job type. Consider the problem that was mentioned earlier, that would occur if the priority of task [1,3,1] was changed to 2 for the job in Figure 3.1. The problem tasks would then be [1,2,1] and [1,3,1] of this job. A second level task has the opportunity to get ahead of third level tasks, without third level tasks having the chance to jump second level tasks. Late arriving second level tasks may jump a tagged third level task, if it is waiting behind other second level tasks. However, late arriving third level tasks cannot jump a tagged second level task, since the second level task of these late jobs must stay behind the tagged task. In other words, there are no late arriving third level tasks when the tagged task is a second level task. All of the second level tasks of these late jobs must be served after the tagged second level task, preventing their third tasks from

obtaining ready-for-service status and joining the queue in time to jump the tagged task.

Tasks Served: Expected Value Equation

Forming the expectation of both sides of Equation 3.14 and introducing probability terms results in the following equation.

$$
\begin{aligned}
E[Y_{cik;djm}] \;=\;& E[Y_{c,i-1,x;djm} \mid_{P_{djm}>P_{cik}}] Pr_{P_{djm}>P_{cik}} \\
&+ E[\Psi_{djm} Y_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik=djm}] Pr_{P_{djm}=P_{cik},\ cik=djm} \\
&+ E[\Psi_{djm} Y_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}<P_{c,i-1,x}}] \\
&\quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}<P_{c,i-1,x}} \\
&+ E[\Psi_{djm} Y_{c,i-1,x;d,j-1,y} \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}<P_{d,j-1,y}}] \\
&\quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}<P_{d,j-1,y}} \\
&+ E[\Psi_{djm} Y_{cik;d,j-1,y}(\text{jumping}) \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}}] \\
&\quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}} \\
&+ E[\Psi_{djm} Y_{c,i-1,x;d,j-1,y}(\text{no jumping}) \mid_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}}] \\
&\quad * Pr_{P_{djm}=P_{cik},\ cik\neq djm,\ P_{cik}>P_{c,i-1,x},\ P_{djm}>P_{d,j-1,y}} \\
&+ E[\Psi_{djm} Y_{cik;d,j-1,y} \mid_{P_{djm}<P_{cik}}] Pr_{P_{djm}<P_{cik}} \qquad (3.15)
\end{aligned}
$$

These probability terms are the same as those found in Equation 3.2. Once again, it is easier to work with the equation in its decomposed form for computational purposes, than to calculate the probabilities. The result is

the following equation:

$$L_{cik;djm} = \begin{cases} L_{c,i-1,x;djm} & P_{djm} > P_{cik} \\ \Psi_{djm} L_{c,i-1,x;d,j-1,y} & P_{djm} = P_{cik} \quad cik = djm \\ & \quad \text{or} \quad P_{cik} < P_{c,i-1,x} \\ & \quad \text{or} \quad P_{djm} < P_{d,j-1,y} \\ \Theta_{djm;cik} \Psi_{djm} L_{cik;d,j-1,y} \mid \text{jumping occurs} & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ (1 - \Theta_{djm;cik}) \Psi_{djm} L_{c,i-1,x;d,j-1,y} \mid \text{jumping does not occur} & \\ & P_{djm} = P_{cik} \quad cik \neq djm \\ & \quad \text{and} \quad P_{cik} > P_{c,i-1,x} \\ & \quad \text{and} \quad P_{djm} > P_{d,j-1,y} \\ \Psi_{djm} L_{cik;d,j-1,y} & P_{djm} < P_{cik}. \end{cases} \tag{3.16}$$

where $L_{cik;djm}$ = the expected number of tasks djm from the late arrivals that are serviced by time t_{cik}.

The first initial condition is $L_{c0;djm} = 0$ for $1 \leq c \leq C$, $1 \leq d \leq C$, $0 \leq j \leq T_d$, and all m. The second is $L_{cik;d0} = \lambda_d S_{cik}$ for $1 \leq c \leq C$, $1 \leq i \leq T_c$, all k, and $1 \leq d \leq C$.

Contribution To Sojourn Time

It is now possible to calculate the expected delay to task cik due to late arrivals ($S_{L_{cik}}$). It is the expected number of tasks djm from the late arrivals that are serviced by time t_{cik} multiplied by the expected service time for tasks djm. Summing the effect of all of the task types, results in the following equation.

$$S_{L_{cik}} = \sum_{d=1}^{C} \sum_{j=1}^{T_d} \sum_{m} L_{cik;djm} \bar{x}_{djm} \tag{3.17}$$

The expected sojourn time for task cik is given in the equation $S_{cik} = S_{E_{cik}} + S_{A_{cik}} + S_{L_{cik}}$. Applying this equation to each task in every job ($1 \leq c \leq C$, $1 \leq i \leq T_c$, and all k) gives a linear system of equations of the form $S = aS + b$. Once again the Jacobi iteration method was used to solve for the unknowns, and no computational difficulty was encountered. The matrix S is a column vector ($S_{1,1}$, $S_{1,2,1}$, $S_{1,2,2}$, $\cdots$, $S_{2,1}$, $\cdots$, $S_{C,1}$, $\cdots$), while a and b are matrices (b is also a column vector) of known quantities.

3.2 Numerical Examples

For demonstration purposes, three numerical examples were solved by both the model and by simulation. The examples are shown in Figures 3.8–3.10. The input data (arbitrary) and the obtained sojourn times are given in Tables 3.1–3.3. The information given in the tables, goes from top to bottom and from left to right in the tree structure. For the simulation, PROPHET was once again used on a Sun SparcStation 2 (Sun operating system). The values shown in Tables 3.1–3.3, are once again the averages of four different runs. Both the number of batches and the batch length were varied for the different runs. The number of batches was either 25 or 50, and the batch length was either 100,000 or 200,000 (see Tables 5.9–5.14 in the Appendix for results). A warmup of 200,000 was used in all cases. The execution time (real time) of the simulation runs varied from 10 to 35 minutes. Most of the times were between 15 to 20 minutes. The analytical model, however,

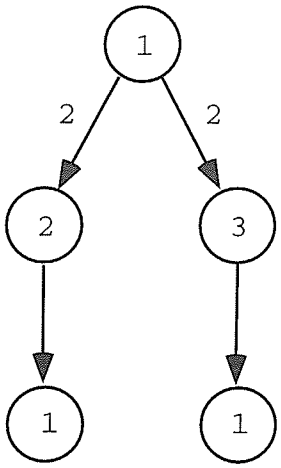

Figure 3.8: Example 5

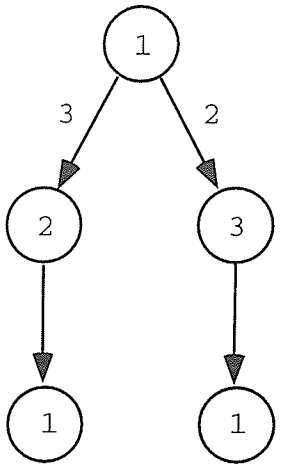

Figure 3.9: Example 6

took only a few seconds to run (2 or 3). All of the sojourn times obtained via the model matched those obtained by simulation. The values obtained by the model were all within the 95% confidence interval for the simulation values. There are a couple of other observations about the sojourn times. The sojourn time for the first task, increases from example five to example six and from example six to example seven. This is what is expected, since in both cases there is an increase in the priority one tasks and the likelihood of an arriving first task finding one in the queue is higher. The difference between the sojourn times of the second level tasks and their successors is equal to the mean service time of the successors. This was also expected, since both successors have a priority of one and both tasks enter an empty queue.

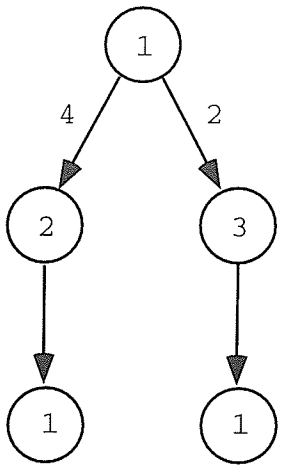

Figure 3.10: Example 7

Priority	1	2	3	1	1
$\bar{x}$	1	1	1	1	1
$E[x^2]$	1	1	1	1	1
λ	0.05				
Sojourn Time: Model	1.1316	4.105	14.24	5.105	15.24
Sojourn Time: Simulation	1.1315	4.097	14.19	5.097	15.19
95% Confidence Interval	0.0015	0.016	0.11	0.016	0.11

Table 3.1: Example 5

Priority	1	2	3	1	1
$\bar{x}$	1	1	1	1	1
$E[x^2]$	1	1	1	1	1
λ	0.05				
Sojourn Time: Model	1.1579	6.275	24.19	7.275	25.19
Sojourn Time: Simulation	1.1578	6.257	24.12	7.257	25.12
95% Confidence Interval	0.0013	0.031	0.26	0.031	0.26

Table 3.2: Example 6

Priority	1	2	3	1	1
$\bar{x}$	1	1	1	1	1
$E[x^2]$	1	1	1	1	1
λ	0.05				
Sojourn Time: Model	1.1842	9.234	41.95	10.234	42.95
Sojourn Time: Simulation	1.1838	9.202	42.05	10.202	43.05
95% Confidence Interval	0.0016	0.061	0.69	0.061	0.69

Table 3.3: Example 7

Chapter 4

Optimization by Simulated Annealing

4.1 Introduction

Earlier (in Section 1.1), the large number of possible solutions to the design problem was discussed. The design problem can be made into an optimization problem, if costs are assigned to the waiting times of the tasks in the queueing system. This would help eliminate many inferior design configurations quickly and easily. A very good method of solving combinatorial optimization problems is by using simulated annealing. One of the strengths of simulated annealing is that it requires no deep insight into the combinatorial structure of the problem. It can produce solutions as good as (or better than) tailored heuristics which exploit problem characteristics. This is good for the current problem, since it is unique and there are no known characteristics which can be exploited.

Simulated annealing has been applied successfully to a wide variety of

combinatorial optimization problems. It has been used to solve the traveling salesman problem [29] [3] [6] [21], the quadratic assignment problem [5] [11] [46], the graph coloring problem [7] [28], and the graph partitioning problem [27]. It has been used to solve the job shop scheduling problem [34] [24], the facility layout problem [22], the vehicle routing problem [23], and the football pool problem [33]. It has also been used in computer design; component placement and wire routing [29]. This is to mention but a few of the papers that have used simulated annealing with very good results. The method has enjoyed great success. Collins et al. [10] have done a bibliography of simulated annealing papers, which is now somewhat dated. However, it does provide a good survey of the earlier work done in this area. There are also some books which have been written on the subject [1] [32]. No references were found that solved the particular problem of this chapter (the telephone switch design problem as a combinatorial problem).

Kirkpatrick et al. [29] have shown that combinatorial optimization problems can be viewed as analogous to large physical systems of the kind studied in statistical mechanics. Statistical mechanics deals with analyzing aggregate properties of large numbers of atoms in liquids and solids. The behavior is characterized by random fluctuations about a most probable behavior, this being the average behavior of the system at that temperature. Material is annealed by first being melted, and then having its temperature slowly lowered with a long time spent at temperatures near its freezing point. The

period of time at each temperature must be long enough to allow a thermal equilibrium to occur. If this is not done, certain random fluctuations will be frozen into the material and the true low energy state will not be reached. The process is much like growing a crystal from a melt; if the temperature is lowered too quickly, glass or a crystal with many defects may be the result.

The annealing process can be simulated using a Monte Carlo procedure like the one developed by Metropolis et al. [37]. In this procedure, the motion of atoms in contact with a heat bath at a given temperature is simulated. The procedure works by randomly displacing the elements, one at a time, by a small amount and calculating the resulting change in the energy (ΔE). If $\Delta E < 0$, the displacement is accepted and the resulting configuration is used as the starting point for the next iteration. If $\Delta E \geq 0$, the displacement is accepted with probability $P(\Delta E) = \exp(-\Delta E/k_b T)$. The term k_b is Boltzmann's constant, and the term T is the temperature. The value for $P(\Delta E)$ is compared with a random variable drawn from a uniform distribution over the interval (0,1). This step is repeated until equilibrium is achieved. When equilibrium has been reached, the temperature is lowered and the procedure is repeated.

The analogy between the physical process and the combinatorial problem is now evident. A feasible solution to the combinatorial optimization problem can be seen as a configuration of atoms. The value of the objective function of the feasible solution, is like the energy of the configuration of

atoms. The optimal solution to the combinatorial problem is analogous to the lowest energy state of the physical system. Heuristic methods for optimization problems that rely on iterative improvement, seek solutions that strictly lower the objective function from one iteration to the next. This would be similar to rapidly quenching the material from a high temperature to a low temperature. The result of such an action is known to produce higher final energy levels than would have been obtained had the cooling process been done differently. The analogy (which has proven to be a good one), suggests that a better solution to the combinatorial optimization problem can be found by occasionally allowing increases in the objective function. Simulated annealing provides a mechanism for accepting increases in objective value in a controlled fashion.

4.2 Problem Description

The problem involves grouping all of the functions (F_c) in job class c ($1 \leq c \leq C$) into a number of groups (G_c), where $G_c \leq F_c$. Let G equal the set of numbers (G_1, G_2, $\cdots$, G_C). As an example, consider the problem given in Table 4.1. In this case, $F_1 = 15$ and $F_2 = 7$. Once the functions have been grouped into tasks, the number of tasks in the first class is G_1 and the number of tasks in the second class is G_2. The number of tasks cannot exceed the number of functions. The maximum number of tasks that can be created (where every function forms its own group), is 15 for the first class

Function #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Service Time Mean	1.5	4	3	4	3	5	9	12	1.5	12	16	3	4	1.5	4
Service Time Second Moment	2.25	16	9	16	9	25	81	144	2.25	144	256	9	16	2.25	16
Waiting Time Cost	4	1	4	1	4	2	3	1	4	3	1	4	1	4	1

Function #	1	2	3	4	5	6	7
Service Time Mean	4	6	4	5	3	4	5
Service Time Second Moment	16	36	16	25	9	16	25
Waiting Time Cost	1	3	1	2	3	1	2

Table 4.1: A 2 class, 22 function problem

and 7 for the second class. Therefore, $G_1 \leq 15$ and $G_2 \leq 7$. The minimum number of tasks that can be created is one for each job. Therefore, $G_1 \geq 1$ and $G_2 \geq 1$. For each function cj (function j in job c) there is an associated service time mean $\bar{x}_{cj}$ and service time second moment $E[x_{cj}^2]$.

Once the number of tasks in each class has been decided (the numbers in G), the formation of the groups can be determined. Say, for example, that $G_1 = 3$ and $G_2 = 2$. Five numbers will be used to form the five tasks. The numbers denote the last function in the task. In the first job, one of these numbers will be 15. The other two can be any of the numbers between 1 and 14 inclusive (providing they are not the same one). Say, for example, they are 2 and 11. The first task will include the first 2 functions, the second task will include functions 3 to 11 inclusive, and the third task will include functions 12 to 15 inclusive. With regards to the second class, one number between 1

and 6 inclusive will mark the end of the first task, and 7 will mark the end of the second. Say, for example, the number is 4. Let Π_{ci} be the last function in group i and job c, where $1 \leq i \leq G_c$ for all c and $1 \leq \Pi_{ci} \leq F_c$ with $\Pi_{ci} = F_c$ for the last task i in job c. Let Π equal the set of these numbers, where $\Pi = (\Pi_{1,1},\ \Pi_{1,2},\ \cdots,\ \Pi_{1,G_1},\ \Pi_{2,1},\ \cdots,\ \Pi_{2,G_2},\ \cdots,\ \Pi_{C,1},\ \cdots,\ \Pi_{C,G_C})$. For the example, $\Pi_1 = (2,\ 11,\ 15)$ and $\Pi_2 = (4,\ 7)$. Combined then, $\Pi = (2,\ 11,\ 15,\ 4,\ 7)$.

Once Π has been determined, values for $\bar{x}_{ci}$ (service time mean for group ci) and $E[x_{ci}^2]$ (service time second moment for group ci) can be calculated. The service time mean for group ci is found by summing the service time means of the functions in the group. The service time second moment for group ci is determined by first calculating the variances of the service times for the functions in the group. This can be done through the use of the expression $E[x^2] = \sigma^2 + \mu^2$ (where σ^2 is the variance and μ is the mean). The variances of the service times for the functions in the group are then summed, giving the variance of the service time for the task [45]. With both the mean and the variance of the service time for the task now known, the expression $E[x^2] = \sigma^2 + \mu^2$ can again be used to determine the second moment for group ci.

In addition to the number of tasks and the task structure (grouping arrangement), a priority must be assigned to each task. This too has its restrictions. In the example, the first job has been grouped into 3 tasks. The

range of priorities is then limited to 3 as well, since there is no practical reason to have any more. It is the assigning of priorities that greatly increases the number of feasible design configurations. If the number of tasks is small, the number of ways to assign priorities is also small. With a 3 task job, the first task can be assigned one of 3 priorities (1, 2, or 3), and the remaining tasks can be assigned one of 2 priorities. The result is $3(2^2) = 12$ ways to assign priorities. However, if the number of tasks is large, the number of ways to assign priorities is also large. The first job in the example can be grouped into 15 tasks. The result is $15(14^{14}) = 1.67\text{x}10^{17}$ different priority assignments. This is only the number of design configurations if the job is grouped into 15 tasks. For the total number of design configurations, configurations involving 14 tasks, 13 tasks, etc. have to be added. There are 14 ways to group 15 functions into 14 tasks. This has to be multiplied by the number of ways to assign priorities ($14(13^{13}) = 4.24\text{x}10^{15}$). The total number of design configurations involving 14 tasks is $5.94\text{x}10^{16}$. It is evident that as the number of functions in a job class becomes large, the number of design configurations involving that job becomes very large, very quickly. The number of job classes also greatly affects the problem size. If the number of design configurations possible with a 15 function job is approximately $3\text{x}10^{17}$, the number of design configurations possible with two 15 function jobs (2 classes) is $9\text{x}10^{34}$.

Based on execution times solving the example problem, and an approx-

imation of 10^{23} solutions in the solution space, it would take 10^{14} years to exhaustively enumerate all the solutions. Clearly, exhaustive enumeration is not a possible method of solution. Multiclass problems are even too big for implicit enumeration techniques such as Dynamic Programming and Branch and Bound (unless they are used to search only a portion of the solution space). This is a problem for which a heuristic is required. In the next section a discussion of alternative heuristics is given. As far as the complexity of the problem is concerned, it is not known if the problem is NP-Hard. However, it is known that the problem is NP-Complete. The problem is an integer programming problem, and these are known to be NP-Complete.

Let P_{ci} be the priority for group ci, where $P = (P_{1,1},\ P_{1,2},\ \cdots,\ P_{1,G_1},\ P_{2,1},\ \cdots,\ P_{2,G_2},\ \cdots,\ P_{C,1},\ \cdots,\ P_{C,G_C})$. Once the values for P and service time ($\bar{x}_{ci}$ and $E[x_{ci}^2]$) have been determined, the sojourn times for the tasks (S_{ci}) can be calculated using the algorithm given in Section 2 of Chapter 2. The unrestricted grouping of the functions does not allow the use of the algorithm given in Chapter 3. Upon determining the sojourn time for the tasks in a particular configuration, the objective function value is obtained by multiplying the waiting times by a cost function. Let CF_{cj} be the waiting time cost of function j in job c, where $1 \leq j \leq F_c$ for all c. Let CT_{ci} be the waiting time cost of task ci, where $1 \leq i \leq G_c$ for all c. The values of CT are calculated by summing the values of CF that are contained within each task. The objective is to minimize W, where $W = CT_{1,1}S_{1,1} + CT_{1,2}S_{1,2} + \cdots + CT_{1,G_1}S_{1,G_1} +$

$CT_{2,1}S_{2,1} + \cdots + CT_{2,G_2}S_{2,G_2} + \cdots + CT_{C,1}S_{C,1} + \cdots + CT_{C,G_C}S_{C,G_C}$.
Minimizing is done by finding the right values for G, Π and P. The goal is to find a configuration which will decrease the waiting time of important functions without seriously degrading the service provided to the less important functions.

4.3 Alternative Heuristics

Genetic algorithms are based on the notion of propagating new solutions from parent solutions. They employ mechanisms that are modeled after those believed to apply in genetics. The best offspring off the parent solutions are retained for the next generation. It is a process that encourages the survival of the fittest. There are three component processes of genetic algorithms: reproduction, crossover, and mutation. Reproduction is the random pairing of individuals (solutions) from a population to create one or more offsprings from them. In the case of the current problem, the task structure of one solution might be combined with the priority assignment of another, to form a new solution. Crossover is defined as gene exchange, or the exchange of attributes. This would be possible in the case of two very similar solutions. Two solutions that have the same number of tasks and task formation, could could create offspring by obtaining priorities for each the tasks from one of the two solutions. An offspring might have obtained the priority for its first task from the first solution's first task, the priority for its second task from

the second solution's second task, the priority for its third task from the first solution's third task, and so on. Mutation is the introduction of a random element, often used to amend the result of a gene exchange when the outcome does not meet restrictions. In this case, consecutive priorities of equal value is a situation where a mutation would be needed.

Neural networks are founded on a generalized stimulus/response (or reinforcement/decay) paradigm. The method emphasizes the importance of structural links, and rules for transmitting signals across these links (which allow the signals to be amplified or attenuated in various combinations). Neural networks vary by their topology (their interconnectedness), the states that can be realized, and the functions that govern the system's dynamics. A neural network can be organized, by appropriate choice of topology, states, and functions, to behave as an optimizing system for a combinatorial problem. The function to be optimized is replaced by an energy function, where constraints are incorporated by means of penalty functions. With this approach, the domain for the energy function is represented as a set of neural interconnections. This is generally accomplished for combinatorial problems with a zero-one integer programming formulation, where each variable is thought of as a neuron that can fire with a voltage output in the range of 0 to 1. Inhibitory connections are introduced so that the energy function can discourage neurons from values that violate the original problem constraints. The connections between the neurons are called synapses, and the magni-

tude of the signal (inhibitory or excitatory) transmitted from one neuron to another is called the synapse value. The goal of the process is to induce the state of the system to evolve over time to an equilibrium state. When there is no further change in state values, a locally optimal solution has been reached. It is important to note, that a neural network is inherently parallel. This approach cannot be evaluated independently of computer hardware. It is not designed for serial computation, and is presently only simulated on present computers.

Tabu search constitutes a meta-procedure that organizes and directs the operations of a subordinate method. The method is a constrained search procedure, where each step consists of solving a secondary optimization problem and admitting only those solutions (moves) that are not excluded by the currently reigning tabu conditions. The tabu conditions have the goal of preventing cycling and forcing the exploration of new regions. The need to avoid cycling arises, because an unconstrained choice of moves (pursuing those with high evaluations) will likely lead back to the same local optimum. The method has a non-backtracking stipulation, such that no move is allowed that is equivalent to, or dominated by, the reversal of a recently made move. Tabu search makes use of a list (called a tabu list) of attributes of past moves. These attributes are recorded in the same sequence as their corresponding solutions are generated. Broadly speaking, the list defines a set of constraints that must be satisfied by all admissible moves. The elements

of the tabu list are continually being replaced as attributes of new moves are recorded. In some variants, tabu list entries are not created for all moves, but only for moves with particular properties (such as those that cause specified variables to change in specified directions). There are instances where a move that is classed as tabu may be legitimately accepted. The aspiration level component of tabu search can override tabu status if an aspiration level can be achieved by the move. There are also facets of tabu search that are referred to as intermediate and long term memory functions. Intermediate term memory operates by recording and comparing features of a selected number of best trial solutions generated during a particular search. The long term memory function employs principles that are roughly the reverse of those for intermediate term memory. It penalizes features that are found to be prevalent in previous executions of the search process, in an effort to produce a new and better starting point.

4.4 Algorithm

A few comments should be made before the algorithm is discussed in detail. The first, is that Boltzmann's constant is not required in these computations since the temperature is merely a control parameter. The second comment, is that this control parameter which is analogous to the temperature in the work done by Metropolis et al. [37], will also be referred to here as the temperature. It is this parameter which controls the probability

of accepting an increase in the value of the objective function. The greater the temperature, the greater the probability an increase will be accepted. An overview of the algorithm will be given first, followed by a more detailed outline.

1. Obtain an initial configuration and temperature.

2. Attempt E_p exchanges with new configurations (E_p = epoch).

3. Has equilibrium been attained? If not, go to 2.

4. Lower the temperature.

5. Has the material been sufficiently cooled? If not, go to 2.

6. Stop.

There are a couple of implementation issues that remain. The first concerns equilibrium. Equilibrium is said to have been reached, if the objective function is not systematically decreasing. This involves saving the value of the objective function every time an equilibrium test is made. If the current value of the objective function is close to any of the previous values (at this temperature), then equilibrium has been reached. After performing a lengthy set of computational experiments, Golden and Skiscim [20] found that if the objective function is not systematically decreasing, a reduction beyond that is highly unlikely.

The second implementation issue is the sequence of temperatures which the algorithm will consider. Hiquebran *et al* [23] developed a simple and effective means of obtaining the starting and final temperature by limiting the maximum allowable increase in objective function (in any one iteration) at these points. It is their method of obtaining these temperatures that will be used in this work. The starting temperature is determined by limiting the maximum allowable increase in objective function to a value of 10% of the initial solution. If the assumption is made, that a value of 0.001 from the probability function is small enough to be negligible, then the value of Δ must be less than 6.908 (or approximately 7) times as large as the temperature if it has a chance of acceptance. This is a direct result of the probability function ($P = \exp(-\Delta/T)$). At the start of the algorithm, Δ is allowed to be up to 10% of the initial solution. If I is the objective function value of the initial solution, the starting temperature need be no higher than $(0.1)I/7$. The final temperature is found in a similar manner. What the maximum allowable increase at termination should be, depends upon the magnitude of the objective function and the precision of the desired results. Therefore, it is left as a user defined parameter. If δ_f is the maximum allowable increase in the objective function at termination, then the final temperature is equal to $\delta_f/7$.

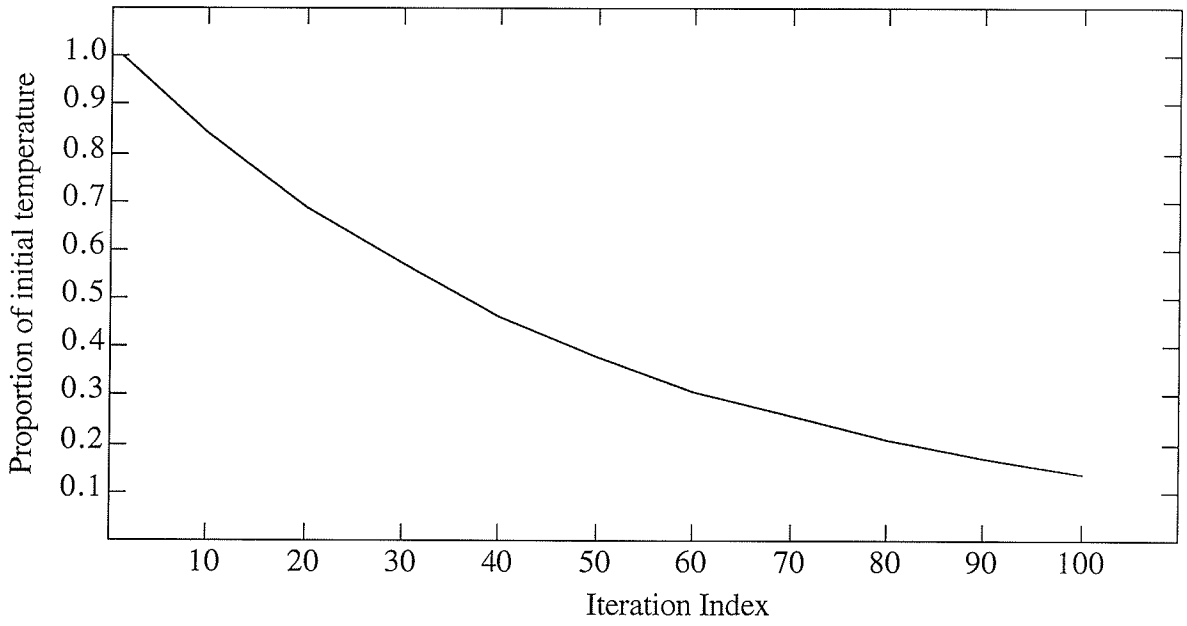

Figure 4.1: Geometric temperature profile.

The remaining temperatures are determined by the use of the following equation:

$$T = ITr^{n-1} \tag{4.1}$$

where IT is the initial temperature, r is the cooling rate, and n is the iteration index. The equation is the geometric temperature function [10], and is a popular cooling method. Johnson *et al* [27] examined alternative methods of cooling (logarithmic, linear, etc.) and found them to be inferior (to geometric) for their particular problem (graph partitioning). The temperature profile for the geometric function is shown in Figure 4.1.

It was necessary to also consider a temperature function that had a much lower cooling rate in the early portion of the annealing process. An inverted logistic function has such a property. It is described by the following equation:

$$T = \frac{IT}{1 + e^{cx}} \tag{4.2}$$

where c is a constant and $x = [\frac{50}{\text{number of temperatures}-1}](n-1) - 25$; for $n > 1$ ($T = IT$ for $n = 1$). A value of 0.25 was chosen for the constant c, because it provided a low cooling rate at the start of the annealing, while not cooling it too fast in the middle portion of the process. A profile for this temperature function is shown in Figure 4.2. The latter half of this curve is similar to the geometric function. The logistic function (if broken up) contains a family of curves, one of which is the geometric function.

The detailed outline is an expansive version of the overview. The steps in the detailed outline follow each other in such a way, that groups of them correspond directly to the steps of the overview. Here then, is a closer look at the algorithm.

1. Randomly generate and evaluate an initial configuration.

2. Determine the initial temperature.

3. Initialize epoch counter ($i = 1$).

4. Initialize exchange counter ($j = 1$).

5. Generate and evaluate a new configuration.

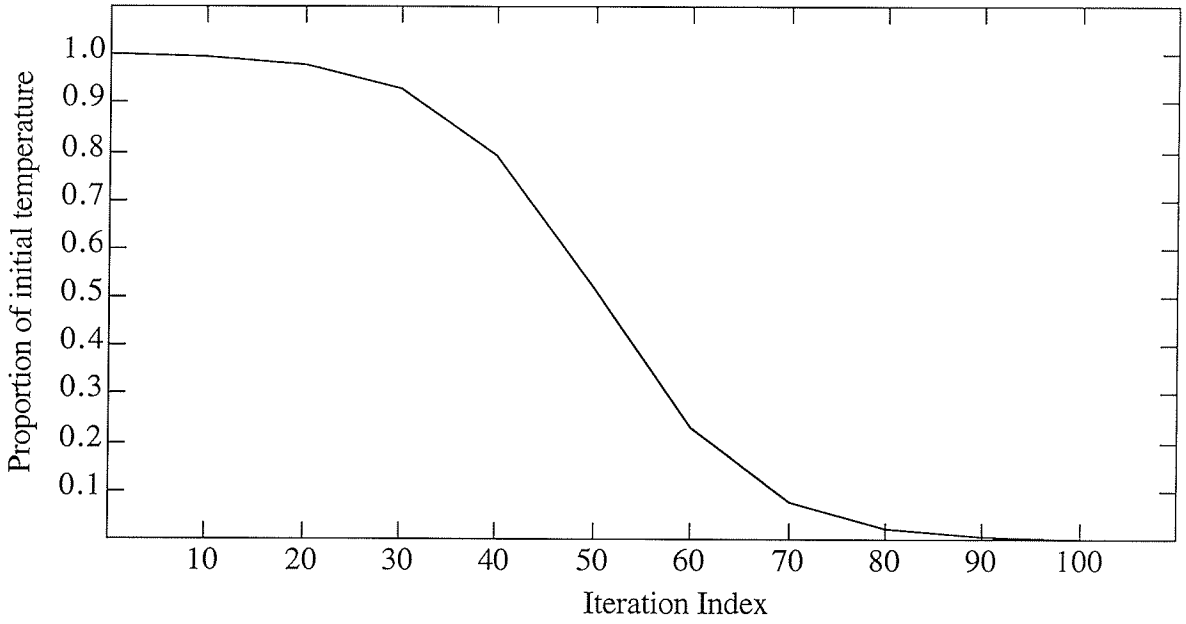

Figure 4.2: Logistic temperature profile.

6. Calculate the change in the objective function ($\Delta = New - Current$).

7. If $\Delta < 0$, accept the exchange ($Current \Leftarrow New$) and go to 10.

8. If $\Delta \geq 0$, generate random number β ($0 \leq \beta \leq 1$).

9. If $\beta < \exp(-\Delta/T)$, accept the exchange ($Current \Leftarrow New$).

10. If the # of attempted exchanges $j < E_p$, increment j and go to 5.

11. If $i = 1$ (first epoch), go to 15.

12. Initialize the equilibrium counter ($k = 1$).

13. If $\frac{|Array(k)-Current|}{Current} \leq \epsilon$ (equilibrium criterion constant), go to 17.

14. If $k < i - 1$, increment k and go to 13.

15. Add $Current$ to array ($Array(i) = Current$).

16. Increment epoch counter ($i = i + 1$) and go to 4.

17. Lower the temperature.

18. If T is not the final temperature, go to 3.

19. Stop.

A neighbourhood structure for generating configurations remains to be defined. When generating a configuration, there are three questions which must be answered. How many tasks will the job be broken up into? How will the functions be grouped into the tasks? What priority will be assigned to each task? These questions have already been implicitly given in Section 2. In the algorithm, each of these questions is answered with a feasible set of random numbers. The sets of numbers were generated in such a way, so as to search groups of configurations in a thorough manner. The algorithm starts by determining how many tasks the job will be broken up into. It then assigns each one a priority number. Finally, a set of pointers which group the functions into tasks, are generated. A new set of pointers are generated for each new configuration. After a user-defined number of groupings ($GroupNum$), a new set of priority numbers are generated. This means

that after $GroupNum$ configurations have been generated (with a common set of priority numbers), a new set of priority numbers are generated. With these new priority numbers, $GroupNum$ new configurations are generated with each one having a different set of pointers for grouping. Duplication is possible, although unlikely due to the randomization. After a user-defined number of priority assignments ($PriorityNum$), the algorithm will once again determine how many tasks the job will be broken up into. If the problem consists of several job types, this will be done for each job type. This will occur every $GroupNum$ x $PriorityNum$ configurations. This process of configuration generation is independent of temperature and epoch number. It allows the algorithm to do both a breadth search and a depth search through the solution space.

An additional feature was added to the basic simulated annealing algorithm. This feature involved storing the best solution so far. It is possible in any single run for the final solution to be worse than the best solution found during the run. It is not computationally expensive (in time or storage) to store details of the best solution found so far. The results show that this is a valuable feature, providing better solutions than those resulting from the basic algorithm in a significant number of the runs.

Function #	1	2	3	4	5	6	7	8	9	10
Service Time Mean	0.4	4.0	3.0	3.0	9.0	4.0	3.5	3.0	0.4	2.0
Service Time Second Moment	0.1621	16.21	9.120	9.120	82.08	16.21	12.41	9.120	0.1621	4.053
Waiting Time Cost	5.0	4.0	3.0	2.0	1.0	2.0	3.0	4.0	5.0	4.0

Table 4.2: Problem one.

4.5 Numerical Results

The first two problems (four in total were tested) are taken from [15], where the mean and second moment of the service time for functions in a practical communication processor are given. Also given, are two priority assignment schemes. The first (problem one) is one that is commonly found in real implementations, while the second (problem two) was found to result in lower values for message transit delay. These priorities are the only information given concerning the urgency for execution. The values for waiting time cost are directly related to these priorities, with high priorities yielding high waiting time costs and low priorities yielding low waiting time costs. Values for problem one are given in Table 4.2.

The problems were solved many times, using various values for the six parameters. The number of temperatures was either 100, 200, 300, or 400 (500 for a couple of cases involving problem four). The epoch was either 25 or 50. The number of priority configurations for a given task number was either 10 or 20 (arbitrarily chosen). The number of group configurations for a given priority configuration was either 10 or 20 (arbitrarily chosen). The magnitude

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2405.5	28.3
100	50	2401.6	56.7
200	25	2401.6	56.4
200	50	2391.2	125.3
300	25	2385.2*	137.4
300	50	2388.8*	176.3
400	25	2391.2	137.3
400	50	2387.3*	381.4
mean		2394.1	137.4
standard deviation		7.68	111.0

Table 4.3: PriorityNum = 10, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$ (urand)

of the maximum allowable increase in objective function at termination (δ_f) was either 3.0 or 1.0. Except for problem four, where the values were 6.0 and 2.0 due to larger objective function values. The equilibrium criterion constant (ε) was either 0.05, 0.1, or 0.15. Some smaller values were also tried (less than 0.05), with the results in keeping with those recorded.

For Problem 1, Table 4.3 shows that (in general) the objective function values decrease as the number of temperatures increases and as the size of the epoch increases. The table also shows that there appears to be no benefit to going beyond 300 temperatures. In the tables that follow, the other four parameters were varied. For Table 4.4, the number of priority configurations for a given task number was increased from 10 to 20. Comparing Tables 4.3 and 4.4 show that this increase resulted in a slight improvement. When the tables were compared, individual values were compared; as well as the mean

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2388.6	38.6
100	50	2402.5*	56.7
200	25	2402.5*	56.9
200	50	2388.9*	154.0
300	25	2396.3	88.9
300	50	2385.0*	207.3
400	25	2388.9*	186.6
400	50	2385.0*	292.8
mean		2392.2	135.2
standard deviation		7.24	90.1

Table 4.4: PriorityNum = 20, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$ (urand)

and standard deviation of the objective function values and the execution times. For example, a comparison of Tables 4.3 and 4.4 show that 5 of the objective function values decreased, while 3 increased. The mean of the objective function values decreased, as well as the mean of the execution time. The standard deviation of the objective function values and the standard deviation of the execution time, both decreased as well. All runs were done on a Sun Sparc 5. The algorithm was encoded in FORTRAN and compiled with the FORTRAN77 compiler.

At this point, a different random number generator was used to dispel any doubt that the results were being affected by the choice of a particular generator. The new generator (taken from [39]) is widely used and its period (for all practical purposes) is infinite. The previous generator (urand) was taken from the WatFor77 library. The runs done in Tables 4.3 and 4.4 were

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2393.5	43.5
100	50	2399.8*	60.0
200	25	2399.8*	60.2
200	50	2389.5	115.3
300	25	2398.7	146.9
300	50	2387.1	253.4
400	25	2389.5	115.6
400	50	2385.4	344.7
mean		2392.9	142.5
standard deviation		5.88	105.6

Table 4.5: PriorityNum = 10, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$ (ran1)

repeated with the new generator (see Tables 4.5 and 4.6). When comparing Table 4.5 to Table 4.3 and Table 4.6 to Table 4.4, the differences are found to be negligible. This new generator (ran1) will be used for all subsequent runs.

For Table 4.7, δ_f was lowered from 3.0 to 1.0. Comparing Tables 4.6 and 4.7 show that this did not improve the solutions. For Table 4.8, the number of group configurations for a given priority configuration was increased from 10 to 20. Comparing Tables 4.7 and 4.8 show that this produced a slight improvement. For Table 4.9, ε was increased from 0.1 to 0.15; and then was decreased to 0.05 for Table 4.10. Comparing Tables 4.8, 4.9, and 4.10 show that there is no improvement for either of these changes.

Finally, for Table 4.11 a logistic temperature function is used (instead of a geometric temperature function). Comparing Tables 4.8 and 4.11 show that

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2395.7	31.3
100	50	2395.7*	74.8
200	25	2395.7	72.4
200	50	2393.0*	109.2
300	25	2385.4	105.9
300	50	2390.4	184.0
400	25	2393.0*	109.5
400	50	2385.5*	258.8
mean		2391.8	118.2
standard deviation		4.33	71.7

Table 4.6: PriorityNum = 20, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$ (ran1)

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2395.7	31.3
100	50	2395.7*	74.6
200	25	2395.7*	72.2
200	50	2393.0	109.8
300	25	2385.4	118.2
300	50	2385.9*	306.0
400	25	2393.0	169.4
400	50	2385.5	279.2
mean		2391.2	145.1
standard deviation		4.80	99.8

Table 4.7: PriorityNum = 20, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2405.1	28.6
100	50	2384.6	80.3
200	25	2384.6	66.1
200	50	2390.6*	138.1
300	25	2385.7	94.9
300	50	2385.2	209.1
400	25	2390.6*	135.9
400	50	2395.4	328.5
mean		2390.2	135.2
standard deviation		7.14	95.3

Table 4.8: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2405.1	28.0
100	50	2384.6	59.0
200	25	2384.6	59.6
200	50	2390.6*	135.6
300	25	2385.7	94.8
300	50	2385.2	197.8
400	25	2390.6*	135.1
400	50	2395.4	350.6
mean		2390.2	132.6
standard deviation		7.1	103.4

Table 4.9: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.15$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2405.1	32.1
100	50	2384.6	76.3
200	25	2384.6	59.3
200	50	2390.6*	134.7
300	25	2385.7	95.8
300	50	2385.2	197.9
400	25	2390.6*	217.4
400	50	2395.4	238.5
mean		2390.2	131.5
standard deviation		7.1	78.1

Table 4.10: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.05$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	2399.1*	27.9
100	50	2408.9*	71.7
200	25	2408.9*	73.4
200	50	2391.5	141.1
300	25	2391.1	77.3
300	50	2388.0	205.8
400	25	2391.5	143.0
400	50	2385.7*	366.7
mean		2395.6	138.4
standard deviation		9.1	107.8

Table 4.11: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$ (logistic temperature function)

a geometric temperature function performs better with this problem. The mean for the objective function values in Table 4.11 is considerably higher than that found in Table 4.8, with 6 of the 8 runs in Table 4.11 contributing to this by having higher objective values.

Function #	1	2	3	4	5	6	7	8	9	10
Service Time Mean	0.4	4.0	3.0	3.0	9.0	4.0	3.5	3.0	0.4	2.0
Service Time Second Moment	0.1621	16.21	9.120	9.120	82.08	16.21	12.41	9.120	0.1621	4.053
Waiting Time Cost	7.0	6.0	1.0	2.0	3.0	4.0	5.0	6.0	7.0	6.0

Table 4.12: Problem Two

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3500.3	30.1
100	50	3506.5*	57.3
200	25	3506.5*	57.5
200	50	3492.9*	125.1
300	25	3506.5*	92.8
300	50	3492.9*	188.0
400	25	3492.9*	124.9
400	50	3491.5*	251.4
mean		3498.8	115.9
standard deviation		6.9	74.1

Table 4.13: PriorityNum = 10, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$

The values for problem two are given in Table 4.12. This problem was dealt with in the same way as Problem 1 was. That is, it was solved many times, using various values for the six parameters. The same values for the parameters were used. The results of the various runs are given in Tables 4.13–4.19. Most of the results are similar to those obtained for problem one. Observations concerning an increase in *PriorityNum* (see Tables 4.13 and 4.14), changes in ε (see Tables 4.15, 4.16, and 4.17), and temperature function are the same (see Tables 4.15 and 4.18). However, there are a couple

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3512.9*	30.5
100	50	3491.6	74.6
200	25	3491.6	76.8
200	50	3493.2	134.1
300	25	3491.6	89.8
300	50	3498.0*	191.9
400	25	3493.2	132.5
400	50	3491.3	353.1
mean		3495.4	135.4
standard deviation		7.4	100.6

Table 4.14: PriorityNum = 20, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3512.9*	30.0
100	50	3491.6	75.9
200	25	3491.6	74.8
200	50	3493.2	132.0
300	25	3491.6	89.0
300	50	3490.5	234.3
400	25	3493.2	133.5
400	50	3490.5	249.5
mean		3494.4	127.4
standard deviation		7.6	78.2

Table 4.15: PriorityNum = 20, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3512.9*	31.0
100	50	3491.6	73.9
200	25	3491.6	74.6
200	50	3493.2	134.7
300	25	3491.6	91.7
300	50	3490.5	298.8
400	25	3493.2	134.5
400	50	3490.5	248.9
mean		3494.4	136.0
standard deviation		7.6	92.5

Table 4.16: PriorityNum = 20, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.15$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3512.9*	30.5
100	50	3491.6	72.2
200	25	3491.6	74.4
200	50	3493.2	132.3
300	25	3491.6	90.3
300	50	3490.5	226.5
400	25	3493.2	134.5
400	50	3490.5	274.2
mean		3494.4	129.4
standard deviation		7.6	82.8

Table 4.17: PriorityNum = 20, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.05$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3496.2	39.1
100	50	3496.2*	50.9
200	25	3496.2*	59.9
200	50	3494.0	127.7
300	25	3501.3*	80.3
300	50	3497.3*	191.0
400	25	3494.0	127.9
400	50	3492.9*	309.0
mean		3496.0	123.2
standard deviation		2.6	90.5

Table 4.18: PriorityNum = 20, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.1$ (logistic temperature function)

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	3497.3	33.9
100	50	3503.3*	55.8
200	25	3503.3*	56.6
200	50	3500.1*	126.7
300	25	3494.0	97.1
300	50	3492.9*	205.2
400	25	3500.1*	127.6
400	50	3494.3*	329.1
mean		3498.2	129.0
standard deviation		4.2	97.5

Table 4.19: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$

Function #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Service Time Mean	1.5	4	3	4	3	5	9	12	1.5	12	16	3	4	1.5	4
Service Time Second Moment	2.25	16	9	16	9	25	81	144	2.25	144	256	9	16	2.25	16
Waiting Time Cost	4	1	4	1	4	2	3	1	4	3	1	4	1	4	1

Table 4.20: Problem Three

of minor differences. There was a slight improvement when δ_f was lowered from 3.0 to 1.0 (compare Tables 4.14 and 4.15). The second difference is that poorer results are obtained when $GroupNum$ is increased from 10 to 20 (compare Tables 4.15 and 4.19).

Problems 3 and 4 are derived from industry specific data for a telephone switch. In order to avoid revealing proprietary information, the service time means are only relatively correct and are not the actual values. Similarly, the waiting time costs correspond to relative priorities and not the actual values. Values for Problem 3 are given in Table 4.20. The results of the various runs are given in Tables 4.21–4.28. Comparing Tables 4.21 and 4.22 show that there is an increase in objective function value when the number of priority configurations for a given task number is increased from 10 to 20. Like Problem 2, there is a slight improvement in objective function value when δ_f is lowered from 3.0 to 1.0 (see Tables 4.21 and 4.23). Comparing Tables 4.23 and 4.24 show that there is a considerable increase in the objective function when the number of group configurations for a given priority configuration is increased from 10 to 20. However, when both the number of priority con-

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6122.2*	83.8
100	50	6122.2*	136.5
200	25	6122.2*	136.2
200	50	6105.4	370.3
300	25	6122.2*	201.6
300	50	6105.4	426.6
400	25	6105.4	269.1
400	50	6105.4*	546.3
mean		6113.8	271.3
standard deviation		9.0	163.0

Table 4.21: PriorityNum = 10, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6183.7	42.9
100	50	6132.6	111.1
200	25	6132.6	111.7
200	50	6124.7	363.8
300	25	6132.6	182.0
300	50	6111.3*	335.8
400	25	6052.6	267.7
400	50	6147.5	584.2
mean		6127.2	249.9
standard deviation		36.9	176.6

Table 4.22: PriorityNum = 20, GroupNum = 10, $\delta_f = 3.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6122.2*	66.1
100	50	6122.2*	132.6
200	25	6122.2*	132.7
200	50	6122.2*	364.7
300	25	6122.2*	210.1
300	50	6052.6	423.6
400	25	6122.2*	271.1
400	50	6106.3	598.1
mean		6111.5	274.9
standard deviation		24.4	178.5

Table 4.23: PriorityNum = 10, GroupNum = 10, $\delta_f = 1.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6163.7	70.4
100	50	6215.1	118.3
200	25	6215.1	118.3
200	50	6145.3	251.4
300	25	6215.1	185.6
300	50	6106.5	474.3
400	25	6145.3	350.6
400	50	6106.5	676.4
mean		6164.1	280.7
standard deviation		46.5	209.0

Table 4.24: PriorityNum = 10, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6100.8	62.2
100	50	6100.8*	126.8
200	25	6100.8*	127.1
200	50	6099.7	279.2
300	25	6100.8*	172.7
300	50	6135.9	354.9
400	25	6099.7	256.5
400	50	6114.5*	454.2
mean		6106.6	229.2
standard deviation		12.8	131.8

Table 4.25: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$

figurations and the number of group configurations are increased to 20, the lowest objective function values and execution times are found (compare Tables 4.23 and 4.25). Once again, there is no improvement in the quality of the solution when ε is changed from 0.1 to 0.15 or 0.05 (see Tables 4.25, 4.26 and 4.27). The logistic temperature function yielded higher values for the objective function for Problem 3 (than the geometric temperature function), and in some cases yielded considerably higher execution times (compare Tables 4.25 and 4.28). The higher execution times are most likely due to the extra processing required to accept a solution and make it the current solution. This is done much more frequently with the logistic temperature function.

The values for problem four are given in Table 4.29. The problem consists of two job types; with the first job being the same as the one used in problem three. Both jobs are part of an actual 23 function job with a single fork. The

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6100.8	60.4
100	50	6100.8*	123.9
200	25	6100.8*	123.7
200	50	6099.7	291.5
300	25	6100.8*	178.5
300	50	6135.9	526.1
400	25	6099.7	279.7
400	50	6114.5*	544.5
mean		6106.6	266.0
standard deviation		12.8	183.8

Table 4.26: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.15$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6100.8	60.7
100	50	6100.8*	124.8
200	25	6100.8*	124.9
200	50	6099.7	256.9
300	25	6100.8*	176.0
300	50	6135.9	361.2
400	25	6099.7	255.2
400	50	6114.5*	462.5
mean		6106.6	227.8
standard deviation		12.8	134.3

Table 4.27: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.05$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	6222.4*	43.6
100	50	6126.2	125.1
200	25	6126.2	126.8
200	50	6095.3*	240.7
300	25	6142.8	296.7
300	50	6147.9*	698.4
400	25	6095.3*	271.9
400	50	6120.2	730.8
mean		6134.5	316.8
standard deviation		40.3	259.9

Table 4.28: PriorityNum = 20, GroupNum = 20, $\delta_f = 1.0$, $\varepsilon = 0.1$ (logistic temperature function)

Function #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Service Time Mean	1.5	4	3	4	3	5	9	12	1.5	12	16	3	4	1.5	4
Service Time Second Moment	2.25	16	9	16	9	25	81	144	2.25	144	256	9	16	2.25	16
Waiting Time Cost	4	1	4	1	4	2	3	1	4	3	1	4	1	4	1

Function #	1	2	3	4	5	6	7
Service Time Mean	4	6	4	5	3	4	5
Service Time Second Moment	16	36	16	25	9	16	25
Waiting Time Cost	1	3	1	2	3	1	2

Table 4.29: Problem Four

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13000.8	107.5
100	50	13000.8	230.1
200	25	13000.8	234.3
200	50	12810.2	688.6
300	25	12810.2	515.6
300	50	12810.2	1363.2
400	25	12810.2	932.2
400	50	12608.4	1361.4
mean		12856.5	679.1
standard deviation		137.6	500.1

Table 4.30: PriorityNum = 10, GroupNum = 10, $\delta_f = 6.0$, $\varepsilon = 0.1$

15 functions before the fork were made into one job, and the 7 functions in one of the branches was made into the other. The algorithm is able to handle multiple job types, so an example was created to both demonstrate this and observe the results. This problem is identical to the one given in Section 2.

The results of the various runs for problem four are given in Tables 4.30–4.37. There are some similarities between these results and those obtained for the previous three problems. Once again, other values for ε did not result in better solutions. No improvement was realized when increasing it from 0.1 to 0.15 (see Tables 4.30 and 4.31), and solutions deteriorated when it was decreased from 0.1 to 0.05 (see Tables 4.30 and 4.32). Once again, the geometric temperature function was found to be the better overall temperature function (see Tables 4.30 and 4.37). This has been true for all of the problems tested, and leads to the conclusion that the difference is

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13000.8	137.1
100	50	13000.8	249.9
200	25	13000.8	231.0
200	50	12810.2	839.2
300	25	12810.2	416.4
300	50	12810.2	1212.7
400	25	12810.2	537.8
400	50	12608.4	1479.6
mean		12856.5	638.0
standard deviation		137.6	493.6

Table 4.31: PriorityNum = 10, GroupNum = 10, $\delta_f = 6.0$, $\varepsilon = 0.15$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13429.2	136.1
100	50	13108.4	273.6
200	25	13000.8	231.8
200	50	12810.2	558.4
300	25	12810.2	413.0
300	50	12810.2	997.6
400	25	12810.2	534.7
400	50	12608.4	1698.3
mean		12923.5	605.4
standard deviation		252.6	515.8

Table 4.32: PriorityNum = 10, GroupNum = 10, $\delta_f = 6.0$, $\varepsilon = 0.05$

due to the structure of this type of combinatorial problem and its associated solution space.

There are also some differences between these results and those obtained from the previous three problems. This problem is the largest of the four, and differences are most likely due to the increase in the solution space. One difference is that more temperatures are sometimes needed to obtain the best solution. The results for earlier problems showed that there was no benefit to going beyond 300 temperatures. The values for the objective function for 400 temperatures were never lower than those obtained for a lower number of temperatures. However with this problem, some values at 400 temperatures are the lowest (see Tables 4.30 and 4.36). When this happened, additional runs at 500 temperatures were done. In both cases, the objective function values did not improve for either epoch (25 or 50).

A second difference is that better solutions were obtained with a larger value for δ_f (see Tables 4.30 and 4.33). Larger values for δ_f were used (2.0 and 6.0), because of the larger values of the objective function. As far as $PriorityNum$ and $GroupNum$ are concerned, increasing either one of them (or both of them) resulted in poorer solutions. Increasing the number of priority configurations for a given task number from 10 to 20, deteriorated the solutions (see Tables 4.30 and 4.34). Increasing the number of group configurations for a given priority configuration from 10 to 20, significantly increased the values of the objective function (see Tables 4.30 and 4.35).

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13392.5	197.4
100	50	13108.4	275.3
200	25	13108.4	386.0
200	50	12810.2	677.5
300	25	12810.2	445.8
300	50	12810.2	1196.4
400	25	12810.2	657.1
400	50	12810.2	1908.2
mean		12957.5	718.0
standard deviation		221.4	573.5

Table 4.33: PriorityNum = 10, GroupNum = 10, $\delta_f = 2.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	12979.9*	149.5
100	50	12871.8	270.6
200	25	12871.8	232.8
200	50	12858.9	606.7
300	25	12867.8	396.9
300	50	12858.9	1335.6
400	25	12858.9	728.3
400	50	12865.8	1641.7
mean		12879.2	670.3
standard deviation		41.1	546.5

Table 4.34: PriorityNum = 20, GroupNum = 10, $\delta_f = 6.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13658.9	104.6
100	50	13149.6	291.8
200	25	13149.6	244.0
200	50	13075.7	539.9
300	25	13149.6	354.1
300	50	12875.4	1225.3
400	25	13075.7	488.3
400	50	12926.6	1515.5
mean		13132.6	595.4
standard deviation		236.9	503.2

Table 4.35: PriorityNum = 10, GroupNum = 20, $\delta_f = 6.0$, $\varepsilon = 0.1$

Increasing both these numbers from 10 to 20, resulted in a large increase in the objective function (see Tables 4.30 and 4.36). Low values for these two numbers makes the search much more of breadth than of depth. Intuitively, this does seem like a good search strategy for a large solution space (it allows the algorithm to cover a lot of ground).

The last difference, is that the logistic temperature function did produce some good solutions when the number of temperatures was low (see Table 4.37). These solutions are better than those produced by the geometric temperature function in this temperature range (see Table 4.30). The slow initial cooling of the logistic temperature function has compensated for the small number of temperatures. However, the geometric temperature function consistently produces more good solutions if given more time (i.e. 400 temperatures).

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13457.1	158.7
100	50	13429.0*	614.8
200	25	13429.0*	377.1
200	50	13320.3	448.0
300	25	13143.4	386.9
300	50	12914.7	1026.6
400	25	13320.3	441.8
400	50	12870.5	1648.9
mean		13235.5	637.9
standard deviation		233.9	479.4

Table 4.36: PriorityNum = 20, GroupNum = 20, $\delta_f = 6.0$, $\varepsilon = 0.1$

# of Temperatures	Epoch	Objective Function Value (*Best)	Time (sec)
100	25	13057.4	192.0
100	50	12608.4	321.1
200	25	12608.4	353.6
200	50	13116.6*	665.7
300	25	12884.7*	378.8
300	50	12935.0	1091.9
400	25	13116.6*	623.0
400	50	12855.6*	1455.2
mean		12897.8	635.2
standard deviation		204.3	434.6

Table 4.37: PriorityNum = 10, GroupNum = 10, $\delta_f = 6.0$, $\varepsilon = 0.1$ (logistic temperature function)

Chapter 5

Conclusions and Recommendations

A queueing model was developed that models the actual behaviour of call processing tasks in a telephone switch. The first implementation of the model used a queueing discipline that allowed tasks to jump to the head of the queue if a similar task was receiving service when it arrived. The results of this model showed that this modification to the first-come first-served (FCFS) discipline does affect sojourn times. The differences show that using a FCFS model to model tasks for this type of problem will result in wrong values. The results indicated that some tasks gained while others lost when jumping occurred. Differences in sojourn times were influenced by the task's arrival rate and its service time. Those tasks with relatively high arrival rates and/or relatively long service times experienced a decrease in their sojourn time. These tasks had more opportunities to jump, and in doing so lowered their expected waiting time. On the other hand, those tasks with

relatively low arrival rates and/or relatively short service times experienced an increase in their sojourn time. These tasks had less opportunities to jump, and suffered for it. There was conservation of work, as one job class gained what the other lost.

Having seen that a correct model needed to allow jumping, a model was then developed that was closer to the actual behaviour of tasks in the switch's operating system. With this model, tasks were allowed to jump and join similar tasks regardless of where they were in the queue and whether or not these tasks were being served at the time of arrival. Equations for the sojourn time were developed. Computer code was then written for them, and some problems were solved. The results were similar to those obtained for the previous model. Tasks with high arrival rates and/or long service times benefitted at the expense of those that don't. The difference with this model is that the tasks affected were not just first tasks or those with the highest priority. In addition, the results were compared to those obtained via simulation. The comparison was a favorable one. The sojourn times obtained via the model were either slightly under, slightly over, or right on the values obtained by simulation (with one poor result being the only exception). These small differences are mostly due to the approximation of the jumping probability (Θ).

The model was further developed to include task forking. Forking occurs in these switches, so the model was further extended to allow analysis of

problems involving forking. Equations for the sojourn time were developed. Computer code was then written for them, and some problems were solved. In addition, the results were compared to those obtained via simulation. The sojourn times obtained via the model matched those obtained by simulation; with all values falling within the 95% confidence interval.

In the last part of the work, the problem of system design was formulated as a combinatorial optimization problem by assigning costs to the waiting times of the tasks. The optimization problem was solved using the simulated annealing method. Due to the problem's neighbourhood structure, the algorithm has six parameters. In general, the objective function values decrease as the number of temperatures increases and as the size of the epoch increases. For the problems tested, there was no benefit to going beyond 300 temperatures and an epoch of 50 (with the exception of a couple of instances with the largest problem where 400 temperatures were needed). A small value for the maximum allowable increase in the objective function at termination (δ_f) is better for problems where the objective function ranged from 2400–6100 (a value of 1 was found to be best), while a larger value (6) produced better results for the fourth problem where the objective function ranged from 12600–13600. The three values for the equilibrium criterion constant (ε) that were examined (0.05, 0.1, and 0.15), produced very similar results. The value 0.1 produced the best results, but the other two values usually matched these results, only rarely producing worse ones. The best

values for the last two parameters, the number of priority configurations for a given task number and the number of group configurations for a given priority configuration, are found by experimentation. Each of the four problems responded differently to the two values given to each of these parameters. Sometimes the differences within a particular problem, were large. The simulated annealing algorithm is sensitive to these two parameters. Put another way, the algorithm is sensitive to how the neighbourhood structure is searched.

Lastly, the geometric temperature function was found to be best for this particular design problem, with one exception. The exception being a very large problem solved by a small number of temperatures. In this case, the slow initial cooling of the logistic function compensates for what would otherwise be too rapid cooling. In many of the runs, the storage of the best solution so far, helped in finding an objective function value lower than the final solution. In conclusion, the algorithm can help system designers by providing them with the structure of a very good design in a short period of time.

Future research on the queueing model would involve formulating the problem in another way. Perhaps the problem can be formulated as a feedback queueing system, similar to that of Paterok [38]. A different formulation could lead to simpler equations and less restrictions (such as the one that exists for consecutive tasks of equal priorities). Another area of research

would be to study the problem with non-preemptive service. Or to study the problem using a combination of preemptive and non-preemptive service with preemption-distance priorities. Future research on the combinatorial optimization problem would be to solve the problem using another heuristic. Other approaches, like genetic algorithms, neural networks, and tabu search could be tried and compared to this approach. Another possible research area would be to develop a real-time design improvement algorithm. This algorithm would continously evaluate and change the system design as the demands placed on the switch change with time. For such an algorithm to be possible, requires a flexible operating system. Functions must be able to be regrouped and reassigned priorities.

Appendix

Batch		Sojourn Time					
#	Length						
25	100000	1.25065	3.64076	4.64076	1.24979	7.08805	8.08805
25	200000	1.24943	3.63572	4.63572	1.24975	7.06484	8.06484
50	100000	1.25047	3.64699	4.64699	1.24962	7.08811	8.08811
50	200000	1.24975	3.64275	4.64275	1.24968	7.08034	8.08034
Average		1.25008	3.64156	4.64156	1.24971	7.08034	8.08034

Table 5.1: Example 1: Sojourn Time

Batch		95% Confidence Interval					
#	Length						
25	100000	0.00278	0.02082	0.02082	0.00190	0.07335	0.07335
25	200000	0.00149	0.01368	0.01368	0.00161	0.06329	0.06329
50	100000	0.00171	0.01548	0.01548	0.00149	0.05987	0.05987
50	200000	0.00109	0.01070	0.01070	0.00112	0.04358	0.04358
Average		0.00177	0.01517	0.01517	0.00153	0.06002	0.06002

Table 5.2: Example 1: 95 % Confidence Interval

Batch		Sojourn Time					
#	Length						
25	100000	2.68248	14.86485	16.86485	1.77207	83.52211	84.52211
25	200000	2.68070	14.71052	16.71052	1.77146	80.02606	81.02606
50	100000	2.68459	14.79834	16.79834	1.77426	82.30394	83.30394
50	200000	2.68365	14.74520	16.74520	1.77352	81.96124	82.96124
Average		2.68286	14.77973	16.77973	1.77283	81.95334	82.95334

Table 5.3: Example 2: Sojourn Time

Batch							
#	Length	95% Confidence Interval					
25	100000	0.00590	0.21921	0.21921	0.00602	4.24672	4.24672
25	200000	0.00408	0.14383	0.14383	0.00446	2.80742	2.80742
50	100000	0.00383	0.16541	0.16541	0.00486	3.25272	3.25272
50	200000	0.00339	0.09904	0.09904	0.00366	2.40387	2.40387
Average		0.00430	0.15687	0.15687	0.00475	3.17768	3.17768

Table 5.4: Example 2: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time	
25	100000	4.46861	7.34247
25	200000	4.46931	7.34463
50	100000	4.50450	7.41511
50	200000	4.51438	7.43106
Average Sojourn Time		4.48920	7.38332

Table 5.5: Example 3: Sojourn Time

# of Batches	Batch Length	95% Confidence Interval	
25	100000	0.07396	0.14254
25	200000	0.05831	0.11831
50	100000	0.05185	0.10195
50	200000	0.04167	0.08196
Average Confidence Interval		0.05645	0.11119

Table 5.6: Example 3: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time	
25	100000	7.60631	11.76378
25	200000	7.64462	11.83068
50	100000	7.70771	11.95690
50	200000	7.71730	11.94650
Average Sojourn Time		7.66899	11.87447

Table 5.7: Example 4: Sojourn Time

# of Batches	Batch Length	95% Confidence Interval	
25	100000	0.16442	0.34773
25	200000	0.13556	0.27341
50	100000	0.12066	0.24575
50	200000	0.08447	0.16574
Average Confidence Interval		0.12628	0.25816

Table 5.8: Example 4: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time				
25	100000	1.1313	4.092	14.17	5.092	15.17
25	200000	1.1314	4.094	14.17	5.094	15.17
50	100000	1.1319	4.099	14.20	5.099	15.20
50	200000	1.1315	4.102	14.20	5.102	15.20
Average Sojourn Time		1.1315	4.097	14.19	5.097	15.19

Table 5.9: Example 5: Sojourn Time

# of Batches	Batch Length	95% Confidence Interval				
25	100000	0.0019	0.0213	0.1361	0.0213	0.1361
25	200000	0.0016	0.0171	0.1173	0.0171	0.1173
50	100000	0.0013	0.0144	0.0982	0.0144	0.0982
50	200000	0.0010	0.0113	0.0841	0.0113	0.0841
Average Confidence Interval		0.0015	0.0160	0.1089	0.0160	0.1089

Table 5.10: Example 5: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time				
25	100000	1.1576	6.250	24.07	7.250	25.07
25	200000	1.1579	6.252	24.13	7.252	25.13
50	100000	1.1578	6.262	24.14	7.262	25.14
50	200000	1.1578	6.264	24.15	7.264	25.15
Average Sojourn Time		1.1578	6.257	24.12	7.257	25.12

Table 5.11: Example 6: Sojourn Time

# of Batches	Batch Length	95% Confidence Interval				
25	100000	0.0017	0.0399	0.3519	0.0399	0.3519
25	200000	0.0013	0.0329	0.2646	0.0329	0.2646
50	100000	0.0012	0.0289	0.2407	0.0289	0.2407
50	200000	0.0009	0.0234	0.1955	0.0234	0.1955
Average Confidence Interval		0.0013	0.0313	0.2632	0.0313	0.2632

Table 5.12: Example 6: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time				
25	100000	1.1833	9.190	41.94	10.190	42.94
25	200000	1.1838	9.195	42.22	10.195	43.22
50	100000	1.1840	9.211	41.98	10.211	42.98
50	200000	1.1840	9.212	42.06	10.212	43.06
Average Sojourn Time		1.1838	9.202	42.05	10.202	43.05

Table 5.13: Example 7: Sojourn Time

# of Batches	Batch Length	95% Confidence Interval				
25	100000	0.0023	0.0769	0.9362	0.0769	0.9362
25	200000	0.0016	0.0663	0.6731	0.0663	0.6731
50	100000	0.0015	0.0555	0.6305	0.0555	0.6305
50	200000	0.0011	0.0470	0.5048	0.0470	0.5048
Average Confidence Interval		0.0016	0.0614	0.6862	0.0614	0.6862

Table 5.14: Example 7: 95 % Confidence Interval

# of Batches	Batch Length	Sojourn Time					
25	100000	1.41	7.57	8.57	1.45	51.10	52.10
25	200000	1.42	7.58	8.58	1.46	51.15	52.15
50	100000	1.41	7.60	8.60	1.45	52.06	53.06
50	200000	1.42	7.60	8.60	1.45	51.87	52.87
Average Sojourn Time		1.42	7.59	8.59	1.45	51.55	52.55

Table 5.15: Example 8: Simulation

Priority	1	2	1	1	3	1
$\bar{x}$	1	1	1	1	1	1
$E[x^2]$	1	1	1	1	1	1
λ	0.2			0.1		
Sojourn Time: Model	1.37	7.62	8.62	1.54	51.67	52.67
Sojourn Time: Simulation	1.42	7.59	8.59	1.45	51.55	52.55

Table 5.16: Example 8: Comparison with model

# of Batches	Batch Length	Sojourn Time			
25	100000	1.37	16.24	1.47	26.26
25	200000	1.38	16.35	1.47	26.39
50	100000	1.38	16.59	1.47	26.73
50	200000	1.38	16.53	1.47	26.63
Average Sojourn Time		1.38	16.43	1.47	26.50

Table 5.17: Example 9: Simulation

Priority	1	2	1	2
$\bar{x}$	1	1	1	1
$E[x^2]$	1	1	1	1
λ	0.3		0.15	
Sojourn Time: Model	1.32	16.64	1.60	26.73
Sojourn Time: Simulation	1.38	16.43	1.47	26.50

Table 5.18: Example 9: Comparison with model

# of Batches	Batch Length	Sojourn Time							
25	100000	1.17	2.96	6.90	7.90	1.18	3.03	12.02	13.02
25	200000	1.17	2.95	6.88	7.88	1.18	3.02	11.85	12.85
50	100000	1.17	2.96	6.91	7.91	1.18	3.02	12.00	13.00
50	200000	1.17	2.96	6.91	7.91	1.18	3.02	11.93	12.93
Average Sojourn Time		1.17	2.96	6.90	7.90	1.18	3.02	11.95	12.95

Table 5.19: Example 10: Simulation

Priority	1	2	3	1	1	2	4	1
$\bar{x}$	1	1	1	1	1	1	1	1
$E[x^2]$	1	1	1	1	1	1	1	1
λ	0.1				0.05			
Sojourn Time: Model	1.16	2.92	6.93	7.93	1.22	3.11	12.00	13.00
Sojourn Time: Simulation	1.17	2.96	6.90	7.90	1.18	3.02	11.95	12.95

Table 5.20: Example 10: Comparison with model

# of Batches	Batch Length	Sojourn Time		
25	100000	4.54	6.06	7.73
25	200000	4.57	6.09	7.76
50	100000	4.60	6.16	7.85
50	200000	4.62	6.17	7.86
Average Sojourn Time		4.58	6.12	7.80

Table 5.21: Example 11: Simulation

Priority	1	1	1
$\bar{x}$	1	1	1
$E[x^2]$	1	1	1
λ	0.5	0.3	0.1
Sojourn Time: Model	3.81	5.72	13.26
Sojourn Time: Simulation	4.58	6.12	7.80

Table 5.22: Example 11: Comparison with model

# of Batches	Batch Length	Sojourn Time					
25	100000	2.73	12.25	13.25	1.98	61.67	62.67
25	200000	2.73	12.14	13.14	1.97	60.31	61.31
50	100000	2.73	12.23	13.23	1.98	62.22	63.22
50	200000	2.73	12.19	13.19	1.97	61.21	62.21
Average Sojourn Time		2.73	12.20	13.20	1.98	61.35	62.35

Table 5.23: Example 12: Simulation

Priority	1	2	1	1	3	1
$\bar{x}$	2	1	1	1	1	1
$E[x^2]$	4	1	1	1	1	1
λ	0.15			0.1		
Sojourn Time: Model	2.71	12.22	13.22	2.05	61.67	62.67
Sojourn Time: Simulation	2.73	12.20	13.20	1.98	61.35	62.35

Table 5.24: Example 12: Comparison with model

# of Batches	Batch Length	Sojourn Time			
25	100000	2.60	12.41	1.83	10.89
25	200000	2.60	12.38	1.83	10.78
50	100000	2.60	12.39	1.83	10.85
50	200000	2.60	12.41	1.83	10.81
Average Sojourn Time		2.60	12.40	1.83	10.83

Table 5.25: Example 13: Simulation

Priority	1	2	1	2
$\bar{x}$	2	1	1	1
$E[x^2]$	4	1	1	1
λ	0.15		0.15	
Sojourn Time: Model	2.64	12.83	1.77	10.44
Sojourn Time: Simulation	2.60	12.40	1.83	10.83

Table 5.26: Example 13: Comparison with model

# of Batches	Batch Length	Sojourn Time			
25	100000	2.60	28.69	1.83	42.58
25	200000	2.60	28.18	1.83	41.48
50	100000	2.60	28.96	1.83	42.84
50	200000	2.60	28.63	1.83	42.22
Average Sojourn Time		2.60	28.62	1.83	42.28

Table 5.27: Example 14: Simulation

Priority	1	2	1	2
$\bar{x}$	2	2	1	1
$E[x^2]$	4	4	1	1
λ	0.15		0.15	
Sojourn Time: Model	2.64	30.10	1.77	39.81
Sojourn Time: Simulation	2.60	28.62	1.83	42.28

Table 5.28: Example 14: Comparison with model

# of Batches	Batch Length	Sojourn Time							
25	100000	2.41	8.28	26.31	27.31	1.48	6.26	109.47	110.47
25	200000	2.41	8.25	25.99	26.99	1.48	6.23	104.10	105.10
50	100000	2.42	8.28	26.14	27.14	1.48	6.26	107.03	108.03
50	200000	2.41	8.26	25.98	26.98	1.48	6.25	106.37	107.37
Average Sojourn Time		2.41	8.27	26.11	27.11	1.48	6.25	106.74	107.74

Table 5.29: Example 15: Simulation

Priority	1	2	3	1	1	2	4	1
$\bar{x}$	2	2	1	1	1	1	1	1
$E[x^2]$	4	4	1	1	1	1	1	1
λ	0.0909				0.0909			
Sojourn Time: Model	2.42	8.41	26.12	27.12	1.47	6.04	107.68	108.68
Sojourn Time: Simulation	2.41	8.27	26.11	27.11	1.48	6.25	106.74	107.74

Table 5.30: Example 15: Comparison with model

# of Batches	Batch Length	Sojourn Time		
25	100000	19.30	24.30	27.06
25	200000	18.80	23.47	26.04
50	100000	19.07	23.95	26.63
50	200000	19.02	23.85	26.57
Average Sojourn Time		19.05	23.89	26.58

Table 5.31: Example 16: Simulation

Priority	1	1	1
$\bar{x}$	5	3	1
$E[x^2]$	25	9	1
λ	0.1	0.1	0.1
Sojourn Time: Model	18.94	22.96	28.96
Sojourn Time: Simulation	19.05	23.89	26.58

Table 5.32: Example 16: Comparison with model

# of Batches	Batch Length	Sojourn Time								
25	100000	1.45	3.97	4.97	1.43	6.74	7.74	1.42	38.07	39.07
25	200000	1.45	3.97	4.97	1.43	6.74	7.74	1.41	38.28	39.28
50	100000	1.45	3.99	4.99	1.43	6.77	7.77	1.42	38.48	39.48
50	200000	1.45	3.98	4.98	1.44	6.76	7.76	1.42	38.42	39.42
Average Sojourn Time		1.45	3.98	4.98	1.43	6.75	7.75	1.42	38.31	39.31

Table 5.33: Example 17: Simulation

Priority	1	2	1	1	3	1	1	4	1
$\bar{x}$	1	1	1	1	1	1	1	1	1
$E[x^2]$	1	1	1	1	1	1	1	1	1
λ	0.05			0.1			0.15		
Sojourn Time: Model	1.62	3.99	4.99	1.45	6.77	7.77	1.35	38.75	39.75
Sojourn Time: Simulation	1.45	3.98	4.98	1.43	6.75	7.75	1.42	38.31	39.31

Table 5.34: Example 17: Comparison with model

# of Batches	Batch Length	Sojourn Time								
25	100000	2.33	7.71	8.71	3.22	20.76	21.76	4.06	90.78	91.78
25	200000	2.33	7.69	8.69	3.22	20.53	21.53	4.05	88.39	89.39
50	100000	2.33	7.71	8.71	3.23	20.61	21.61	4.06	89.72	90.72
50	200000	2.33	7.71	8.71	3.22	20.54	21.54	4.05	90.24	91.24
Average Sojourn Time		2.33	7.71	8.71	3.22	20.61	21.61	4.06	89.78	90.78

Table 5.35: Example 18: Simulation

Priority	1	2	1	1	3	1	1	4	1
$\bar{x}$	1	1	1	2	1	1	3	1	1
$E[x^2]$	1	1	1	4	1	1	9	1	1
λ	0.075			0.075			0.075		
Sojourn Time: Model	2.16	7.73	8.73	3.20	20.63	21.63	4.13	91.00	92.00
Sojourn Time: Simulation	2.33	7.71	8.71	3.22	20.61	21.61	4.06	89.78	90.78

Table 5.36: Example 18: Comparison with model

Bibliography

[1] Aarts, E., and Korst, J. (1989), *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, Chichester.

[2] Blouin, F., and Giroux, N. (1993), "TOSIL and SHEL reference manuals", Bell-Northern Research Ltd..

[3] Bonomi, E., and Lutton, J.L. (1984), "The N-City Travelling Salesman Problem: Statistical Mechanics And The Metropolis Algorithm", *SIAM Review*, Vol. 26, No. 4, pp. 551–568.

[4] Burakowski, W. (1988), "Analysis of a Single Processor with Mixed Priorities and Deterministic Feedback", *Computer Communications Systems*, A.G. Cerveira (Editor), Elsevier Science Publishers B.V., North-Holland, pp. 231–240.

[5] Burkard, R.E., and Rendl, F. (1984), "A thermodynamically motivated simulation procedure for combinatorial optimization problems", *European Journal of Operational Research*, Vol. 17, pp. 169–174.

[6] Cerny, V. (1985), "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *Journal Of Optimization Theory And Applications*, Vol. 45, No. 1, pp. 41–51.

[7] Chams, M., Hertz, A., and Werra, D. de (1987), "Some experiments with simulated annealing for coloring graphs", *European Journal of Operational Research*, Vol. 32, pp. 260–266.

[8] Cobham, A., 1954, Priority Assignments in Waiting Line Problems. *Operations Research*, Vol. 2, pp. 70–76.

[9] Cobham, A., 1955, Priority Assignment–A Correction. *Operations Research*, Vol. 3, p. 547.

[10] Collins, N.E., Eglese, R.W., and Golden, B.L. (1988), "Simulated Annealing–An Annotated Bibliography", *American Journal Of Mathematical And Management Sciences*, Vol. 8, pp. 209–307.

[11] Connolly, D.T. (1990), "An Improved Annealing Scheme for the QAP", *European Journal of Operational Research*, Vol. 46, pp. 93–100.

[12] Daigle, J., and Houstis, C. (1979) "A Preemptive Priority Queueing Analysis With Application To A Terminal Controller In A Communications Environment", *National Telecommunications Conference Records*, Washington, DC, November 27–29, pp. 13.3.1–13.3.6.

[13] Daigle, J., and Houstis, C. (1981) "Analysis of a Task Oriented Multipriority Queueing System", *IEEE Transactions on Communications*, Vol. COM-29, No. 11, November, pp. 1669–1677.

[14] Daigle, J. (1981) "Queueing Analysis of Tasking Structure in Task Oriented Multipriority Communication Processing Systems with Multiple Job Classes", *National Telecommunications Conference Records*, New Orleans, Louisiana, November 29–December 3, pp. E3.3.1–E3.3.8.

[15] Daigle, J., Liu, S., and Wong, K. (1983) "Design Alternatives for Communication Processing Systems", *Globecom '83 Conference Records*, San Diego, California, November 28–December 1, pp. 10.5.1–10.5.9.

[16] Daigle, J. (1986) "Task-Oriented Queueing: An Analysis Tool for Software Design of Communication Processing Systems", *IEEE Transactions on Communications*, Vol. COM-34,No. 3, March, pp. 250–256.

[17] Daigle, J. (1990) "Message Delays at Packet-Switching Nodes Serving Multiple Classes", *IEEE Transactions on Communications*, Vol. COM-38, No. 4, April, pp. 447–455.

[18] Eglese, R.W. (1990), "Simulated Annealing: A Tool for Operational Research", *European Journal of Operational Research*, Vol. 46, pp. 271–281.

[19] Glover, F., and Greenberg, H.J. (1989), "New approaches for heuristic search: A bilateral linkage with artificial intelligence", *European Journal of Operational Research*, Vol. 39, pp. 119–130.

[20] Golden, B.L., and Skiscim, C.C. (1986), "Using Simulated Annealing to Solve Routing and Location Problems", *Naval Research Logistics Quarterly*, Vol. 33, pp. 261–279.

[21] Goldstein, L., and Waterman, M. (1988), "Neighborhood Size In The Simulated Annealing Algorithm", *American Journal Of Mathematical And Management Sciences*, Vol. 8, pp. 409–423.

[22] Heragu, S.S., and Alfa, A.S. (1992), "Experimental analysis of simulated annealing based algorithms for the layout problem", *European Journal of Operational Research*, Vol. 57, pp. 190–202.

[23] Hiquebran, D.T., Alfa, A.S., Shapiro, J.A., and Gittoes, D.H. (1994), "A Revised Simulated Annealing and Cluster-First Route-Second Algorithm Applied To The Vehicle Routing Problem", *Engineering Optimization*, Vol. 22, pp. 77–107.

[24] Ishibuchi, H., Misaki, S., and Tanaka, H. (1995), "Modified simulated annealing algorithms for the flow shop sequencing problem", *European Journal of Operational Research*, Vol. 81, pp. 388–398.

[25] Jaiswal, N. (1968) *Priority Queues*, Academic Press, New York.

[26] Jennings, A. (1977) *Matrix Computation for Engineers and Scientists*, John Wiley & Sons, New York.

[27] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C. (1989), "Optimization By Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning", *Operations Research*, Vol. 37, No. 6, pp. 865–892.

[28] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C. (1991), "Optimization By Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring And Number Partitioning", *Operations Research*, Vol. 39, No. 3, pp. 378–406.

[29] Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1983), "Optimization by Simulated Annealing", *Science*, Vol. 220, No. 4598, pp. 671–680.

[30] Kleinrock, L. (1975), *Queueing Systems (Volume I: Theory)*, John Wiley & Sons, New York.

[31] Kleinrock, L. (1976), *Queueing Systems (Volume II: Computer Applications)*, John Wiley & Sons, New York.

[32] Laarhoven, P.J.M. van, and Aarts, E.H.L. (1987), *Simulated Annealing: Theory and Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

[33] Laarhoven, P.J.M. van, Aarts, E.H.L., Lint, J.H. van, and Wille, L.T. (1989), "New Upper Bounds for the Football Pool Problem for 6, 7, and 8 Matches", *Journal of Combinatorial Theory*, Series A, Vol. 52, pp. 304–312.

[34] Laarhoven, P.J.M. van, Aarts, E.H.L., and Lenstra, J.K. (1992), "Job Shop Scheduling By Simulated Annealing", *Operations Research*, Vol. 40, No. 1, pp. 113–125.

[35] Little, J. (1961), "A Proof of the Queueing Formula $L = \lambda W$", *Operations Research*, Vol. 9, pp. 383–387.

[36] McClelland, J.L., and Rumelhart, D.E. (1988), *Explorations In Parallel Distributed Processing*, The MIT Press, Cambridge, Massachusetts.

[37] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E. (1953), "Equation of State Calculations by Fast Computing Machines", *The Journal Of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092.

[38] Paterok, M. (1991), "Expected Sojourn Times for M/G/1—Feedback Queues with Mixed Priorities", *IBM Research Report*, RC 17083, # 75618, July 25.

[39] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T. (1992), *Numerical Recipes in FORTRAN*, 2nd ed., Cambridge University Press, Cambridge.

[40] Ruschitzka, M. (1978), "An Analytical Treatment of Policy Function Schedulers", *Operations Research*, Vol. 26, No. 5, pp. 845–863.

[41] Schrage, L. (1967), "The Queue M/G/1 With Feedback To Lower Priority Queues", *Management Science*, Vol. 13, No. 7, pp. 466–474.

[42] Simon, B. (1984), "Priority Queues with Feedback", *Journal of the Association for Computing Machinery*, Vol. 31, No. 1, January, pp. 134–149.

[43] Smit, G. (1980), "Call Processing Software", *Telesis*, No. 4.

[44] Takacs, L. (1962), *Introduction to the Theory of Queues*, Oxford University Press, New York.

[45] Walpole, R.E., and Myers, R.H. (1989), *Probability and Statistics for Engineers and Scientists*, Macmillan Publishing Company, New York.

[46] Wilhelm, M.R., and Ward, T.L. (1987), "Solving Quadratic Assignment Problems by 'Simulated Annealing'", *IIE Transactions*, March, pp. 107–119.

[47] Wolff, R.W. (1970), "Work-Conserving Priorities", *J. Appl. Prob.*, Vol. 7, pp. 327–337.

[48] Wolff, R.W. (1982), "Poisson Arrivals See Time Averages", *Operations Research*, Vol. 30, No. 2, pp. 223–231.

[49] Wolff, R.W. (1989), *Stochastic Modeling And The Theory Of Queues*, Prentice Hall, Englewood Cliffs, New Jersey.