9-1-2001

A Decomposition Approach for the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources

Michael L. Fredley

A DECOMPOSITION APPROACH FOR THE MULTI-MODAL, RESOURCE-CONSTRAINED, MULTI-PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE AND EXPEDITING RESOURCES

DISSERTATION

Michael L. Fredley, Major, USAF

AFIT/DS/ENS/01-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Report Documentation Page

Report Date	Report Type	Dates Covered (from... to)
13 Dec 2001	Final	June 1997 - Sept 2001

Title and Subtitle A Decomposition Approach for the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources	**Contract Number**
	Grant Number
	Program Element Number
Author(s) Major Michael L. Fredley, USAF	**Project Number**
	Task Number
	Work Unit Number
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 Wright-Patterson AFB, OH 45433-7765	**Performing Organization Report Number** AFIT/DS/ENS/01-02
Sponsoring/Monitoring Agency Name(s) and Address(es) Air Force Office of Scientific Research ATTN: Major Juan R. Vasquez 801 N. Randolph St., Room 933 Arlington, VA 22203-1977	**Sponsor/Monitor's Acronym(s)**
	Sponsor/Monitor's Report Number(s)
Distribution/Availability Statement Approved for public release, distribution unlimited	
Supplementary Notes	

Abstract The field of project scheduling has received a great deal of study for many years with a steady evolution of problem complexity and solution methodologies. As solution methodologies and technologies improve, increasingly complex, real-world problems are addressed, presenting researchers a continuing challenge to find ever more effective means for approaching project scheduling. This dissertation introduces a project scheduling problem which is applicable across a broad spectrum of real-world situations. The problem is based on the well-known Resource-Constrained Project Scheduling Problem, extended to include multiple modes, generalized precedence, and expediting resources. The problem is further extended to include multiple projects which have generalized precedence, renewable and nonrenewable resources, and expediting resources at the program level. The problem presented is one not previously addressed in the literature nor is it one to which the existing specialized project scheduling methodologies can be directly applied. This dissertation presents a decomposition approach for solving the problem, including algorithms for solving the decomposed subproblems and the master problem. This dissertation also describes a methodology for generating instances of the new problem, extending the way existing problem generators describe and construct network structures and this class of problem. The methodologies presented are demonstrated through extensive empirical testing.	
Subject Terms CPM, Critical Path Method, Decomposition, Network, Network Generator, Program, Program Management, Project, Project Generator, Project Management, Project Scheduling, Scheduling, Sweeney-Murphy Decomposition	
Report Classification unclassified	**Classification of this page** unclassified
Classification of Abstract unclassified	**Limitation of Abstract** UU
Number of Pages 270	

AFIT/DS/ENS/01-02

A DECOMPOSITION APPROACH FOR THE MULTI-MODAL, RESOURCE-CONSTRAINED, MULTI-PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE AND EXPEDITING RESOURCES

DISSERTATION

Presented to the Faculty

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

Michael L. Fredley, B.S., M.S.

Major, USAF

September 2001

AFIT/DS/ENS/01-02

A DECOMPOSITION APPROACH FOR THE MULTI-MODAL, RESOURCE-CONSTRAINED, MULTI-PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE AND EXPEDITING RESOURCES

Michael L. Fredley, B.S., M.S.
Major, USAF

Approved:

	Date
Richard F. Deckro (Chairman)	
Aihua W. Wood (Dean's Representative)	
James W. Chrissis (Member)	
James T. Moore (Member)	
E. Price Smith (Member)	

Accepted:

Robert A. Calico, Jr. Date
Dean, Graduate School of Engineering and Management

Acknowledgments

I would like to express my sincere appreciation to my advisor, Dr. Richard F. Deckro, for his guidance and support throughout the course of this dissertation effort. I am also grateful to those who have served on my committee, Dr. James W. Chrissis, Dr. James T. Moore, Lt Col E. Price Smith, Col Jack A. Jackson, and Maj Edward A. Pohl, for the expertise they have shared and for their patience through this long process. I express my gratitude to my Dean’s representative, Dr. Aihua W. Wood, for her assistance in finalizing this dissertation and for her contributions during my defense. Without the mentoring of these outstanding individuals, this dissertation would not have been possible.

Special thanks to my leadership at the Air Force Studies and Analyses Agency, Col Donald P. Higgins, Jr., Col Rowayne A. Schatz, and Maj Jeffrey Marcotte, for providing me the resources I needed to complete this research. Thanks are due, also, to Mr. Steven Sovine who taught me so much about *processes* and how to handle large amounts of data, without which my analysis would have been extremely painful. Finally, I am indebted to my family which has stood by me through this effort, and every other endeavor I’ve undertaken, and given me the encouragement I needed.

Michael L. Fredley

Table of Contents

Page

Acknowledgments....iv

List of Figures....ix

List of Tables....xi

Abstract....xii

I. Introduction....1-1

Overview....1-1
Background....1-1
Research Issues....1-6
Research Objectives....1-6
Approach....1-8
Summary....1-9

II. Literature Review....2-1

Introduction....2-1
Problem Hierarchy....2-1
Single-Project Scheduling....2-3
The Project-Scheduling Problem....2-3
Resource Constraints....2-4
Mathematical Programming....2-7
Graph-Based Approaches....2-8
Implicit Enumeration....2-9
Other Approaches....2-10
Activity Crashing....2-12
Minimal Cost Project Network Problem....2-12
Project Time/Cost Tradeoff Problem....2-12
Activity Duration Crashing Problem....2-12
Multi-Modal, Resource-Constrained Project Scheduling Problem....2-13
Resource-Constrained Project Scheduling
Problem with Multiple Crashable Modes....2-17
Mode-Identity, Resource-Constrained Project Scheduling Problem....2-17
Expediting Resources....2-17
Generalized Precedence....2-19
Multi-Project Scheduling....2-20
Multi-Project Scheduling Problem....2-20
Resource-Constrained, Multi-Project Scheduling Problem....2-22
Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem....2-23

Page

Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem 2-24
Summary 2-24

III. Methodology 3-1

Introduction.... 3-1
Mathematical Formulation.... 3-1
Assumptions 3-5
Notation.... 3-5
Numbering of Activity Modes 3-8
Activity Start Time Windows 3-8
Constraints....3-12
Objective Function 3-18
Complete Model....3-19
Problem Size and Complexity....3-21
Decomposition of the MRCMPSP-GPR/EXP 3-24
Solution Methodologies 3-28
Problem Generation....3-28
Single Project/Subproblem Solution....3-29
Decomposition/Master Problem Solution 3-30
Summary 3-30

IV. Problem Generation.... 4-1

Overview.... 4-1
PAGER: Problem Generator.... 4-4
Step 1 – Specification File Input.... 4-4
Step 2 – Basic Data Generation.... 4-6
Step 3 – Network Generation.... 4-8
Step 4 – Resource Data Generation 4-40
Step 5 – Cost Data Generation 4-42
Step 6 – Problem Output....4-44
PAGER Implementation 4-44
Summary and Conclusions....4-47

V. Single Project Scheduling 5-1

Overview.... 5-1
Approaches from the Literature 5-2
Implicit Enumeration by Branch-and-Bound 5-2
Zero-One Programming.... 5-6
Implicit Enumeration by Activity Sequence.... 5-7
Basic Algorithm 5-7
Assumptions 5-9
Initialization Phase....5-10

Search Phase 5-10

Page

Bounding Rules 5-19
Bounding Rule ZDS (Zero-Duration Activity Start) 5-20
Feasibility Rule NRF (Nonrenewable Resource Feasibility) 5-20
Bounding Rule NEC (Nonrenewable Expediting Resource Cost) 5-21
Feasibility Rule EST (Early Start Time) 5-21
Feasibility Rule MD (Mode Duration) 5-22
Bounding Rule MC (Mode Cost) 5-23
Bounding Rule REC (Renewable Expediting Resource Cost) 5-24
Feasibility Rule MOD (Infeasible Modes) 5-25
Testing 5-25
Test Problem Parameters Held Constant 5-26
Test Problem Parameters Which Are Varied 5-27
Computational Contribution of Bounding Rules 5-28
Comparison to Integer Programming 5-32
Solution Results vs. Key Parameters 5-34
Solution Time 5-45
Time to Optimal Solution 5-48
Completion Time vs. *k* 5-52
Summary and Conclusions 5-55

VI. Program Decomposition Algorithm 6-1

Overview 6-1
Decomposition Approaches in the Literature 6-2
Sweeney-Murphy Decomposition 6-4
Problem Decomposition 6-4
Solving the Subproblems 6-10
Solving the Master Problem 6-11
Assumptions 6-13
Notation 6-13
Decomposition Algorithm 6-15
Correction to Sweeney-Murphy Approach 6-22
Choice of *k* 6-26
Choice of Multipliers 6-29
Acceleration Schemes 6-35
Subproblem Solution Bounding 6-36
Series Approach 6-36
Incremental Enumeration 6-38
Test Problem Design 6-38
Program Designs 6-38
Project Level Difficulty 6-40
Program Level Difficulty 6-41
Problem Generation 6-43
Testing Results 6-43
Methods of Determining Multipliers 6-43
Acceleration Schemes 6-45

Choice of *k*....6-47
Comparison to Single-Project Scheduler....6-49

Page

Non-Convergence....6-50
Summary and Conclusions....6-51

VII. Contributions and Recommendations....7-1

Contributions....7-1
Recommendations....7-5
Summary....7-6

Appendix A. Notation....A-1

Overview....A-1
Problem Types....A-1
Abbreviations and Acronyms....A-1
Mathematical Notation....A-2
Notation....A-3

Appendix B. Sample PAGER Input....B-1

Problem Generator Input....B-1

Appendix C. Sample PAGER Output....C-1

Problem File....C-1

Appendix D. Sample Scheduler Output....D-1

Appendix E. Sample Decomposition Algorithm Output....E-1

Appendix F. Best Solutions to (MP) Versus (P)....F-1

Overview....F-1
Problem Generation Input....F-1
Problem File....F-3
Key Solutions (1, 2, 99, 100, 1000)....F-5

Bibliography....BIB-1

Vita....VITA-1

List of Figures

Figure Page

2-1. Problem Hierarchy 2-2
3-1. Activity-on-Node Representation of Example Problem 1 3-2
3-2. Example Problem 2 3-3
3-3. Example Activity Start Time 3-9
3-4. Block-Angular Structure........ 3-27
4-1. Overall Flow of PAGER........ 4-5
4-2. Standard and Generalized Precedence Arcs 4-9
4-3. Project Lags........ 4-13
4-4. CNC versus RT........ 4-18
4-5. Generation of a Project Network........ 4-26
4-6. Generating a Project Network........ 4-27
4-7. Generating a Parallel Project Network 4-28
4-8. Example of Multi-Project Program 4-29
4-9. Initializing $\overline{\mathbf{R}}_0$ with Ones........ 4-30
4-10. Initializing $\overline{\mathbf{R}}_0$ with Mixed Project Nodes 4-30
4-11. Precedence-Feasible Early Start Times of Zero-Duration Activities 4-39
4-12. Distribution of Generation Times (5 to 50 Activities) 4-45
4-13. Distribution of Generation Times (5 to 42 Activities........ 4-46
4-14. Distribution of Generation Times by Number of Job 4-46
5-1. Rule vs. Average Solution Time (seconds) for 5 Activities........ 5-29
5-2. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities 5-30
5-3. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities and Varying Modes........ 5-31
5-4. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities and Varying RT........ 5-32
5-5. Scheduler vs. OSL Improvement by Restrictiveness for 10 Activities........ 5-34
5-6. Solution Results as Occurrences 5-36
5-7. Solution Results as Percentages 5-36
5-8. Infeasible Problems vs. Resource Strength 5-37
5-9. Infeasible Problems vs. RS and Mode 5-38
5-10. Infeasibilities vs. RS and Percent of Activities with GPR........ 5-39
5-11. Solution Time Exceeding 20 Seconds vs. Restrictiveness (Occurrences) 5-40
5-12. Solution Time Exceeding 20 Seconds vs. Restrictiveness (Percentages) 5-40
5-13. Solution Time Exceeding 20 Seconds vs. RS........ 5-41
5-14. Problems Solved by 2-Level Factors 5-42
5-15. Problems Solved Versus Restrictiveness........ 5-43
5-16. Problems Solved Versus Resource Strength 5-44
5-17. Cumulative Problems Solved by Time Bin and Jobs 5-46
5-18. Cumulative Problems Solved by Time Bin and RT 5-47
5-19. Cumulative Problems Solved by Time Bin and RS 5-47
5-20. Average Time to Optimal Versus Completion Time Bin 5-49
5-21. Optimal Time Bin Versus Completion Time Bin 5-49
5-22. Problems Solved Versus Completion Time Bin........ 5-50

5-23. Problems Solved Versus Completion Time Bin by RT..5-50
Figure Page

5-24. Problems Solved Versus Completion Time Bin by RS..5-51
5-25. Problem Solution Results for k=1, 10, 100, 1000..5-53
5-26. Solution Time Statistics for k=1, 10, 100, 1000..5-53
5-27. Average Completion Time vs. *k*...5-54
5-28. Average Time Per Solution..5-55
5-29. Overall Average Solution Times Versus *k*...5-55
6-1. Block-Angular Structure... 6-5
6-2. Sweeney-Murphy Subproblem.. 6-6
6-3. Sweeney-Murphy Master Problem... 6-7
6-4. Revised Sweeney-Murphy Master Problem..6-12
6-5. Sweeney-Murphy Optimality Theorem Counterexample Diagram..................................6-24
6-6. Sweeney-Murphy Optimality Theorem Counterexample Chart 6-25
6-7. Average Time Per Solution..6-28
6-8. Overall Average Solution Times Versus k...6-28
6-9. Lagrangian Dual of Original Problem (P)..6-29
6-10. Program Designs...6-40
6-11. Solution Results vs. Multiplier Type / Scheduler...6-44
6-12. Solution Results vs. Program Design 6-45
6-13. Solution Results vs. *k* 6-47
6-14. Iterations Required vs. *k* 6-48
6-15. Log Distribution of Solution Times vs. *k*...6-50
7-1. Problem Hierarchy 7-2

List of Tables

Table Page

2-1. Key Features of Project Scheduling Problems 2-3
3-1. Example Problem 2 3-3
3-2. Activity Data for Example Problem 3-22
3-3. Number of Variables 3-23
3-4. Number of Constraints 3-24
4-1. Input Parameters for Basic Data 4-6
4-2. Basic Data Variables 4-7
4-3. Input Parameters for Project Network Generation 4-25
4-4. Input Parameters for Inter-Project Network Generation 4-31
4-5. Input Parameters for Resource Data Generation 4-41
4-6. Input Parameters for Cost Data Generation 4-44
4-7. Key Features of Problem Generators 4-48
5-1. Problem Generation Parameters Held Constant 5-27
5-2. Parameters Which Are Varied 5-28
5-3. *Reduced* Test Design 5-28
5-4. Rule vs. Average Solution Time (seconds) for 5 Activities 5-29
5-5. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities 5-30
5-6. Rule vs. Problems Solved to Optimality (Within 300 sec.)
for 10 Activities and Varying Modes 5-31
5-7. Rule vs. Problems Solved to Optimality (Within 300 sec.)
for 10 Activities and Varying RT 5-31
5-8. Scheduler vs. OSL Solution Time (seconds) for 5 Activities 5-33
5-9. Scheduler vs. OSL Solution Time (seconds) for 10 Activities 5-33
5-10. Scheduler vs. OSL Improvement by Restrictiveness for 10 Activities 5-33
5-11. *Full* Test Design 5-34
5-12. *Minimal* Test Design 5-35
5-13. Solution Results 5-35
5-14. Infeasible Problems 5-37
5-15. Infeasibilities by RS and Mode 5-38
5-16. Infeasibilities by RS and Percent of Activities with Generalized Precedence (GPR) 5-38
5-17. Solution Time for 10- and 50-Activity Projects 5-48
5-18. Time to Optimal (10-Activity Projects) 5-52
5-19. Problem Solution Results for k=1, 10, 100, 1000 5-52
6-1. Sweeney-Murphy Optimality Theorem Counterexample Data 6-24
6-2. Project-Level Generation Parameters Which Vary 6-41
6-3. Project Level Generation Parameters Held Constant 6-41
6-4. Program-Level Generation Parameters Which Vary 6-42
6-5. Problem Design 6-42
6-6. Solution Time vs. Multiplier Type / Scheduler 6-44
6-7. Solution Time vs. Problem Difficulty 6-46
6-8. Value of Acceleration Schemes 6-47
6-9. Solution Times vs. *k* 6-48
6-10. Solution Time vs. *k* 6-49
7-1. Summary of Key Contributions 7-4

AFIT/DS/ENS/01-02

Abstract

The field of project scheduling has received a great deal of study for many years with a steady evolution of problem complexity and solution methodologies. As solution methodologies and technologies improve, increasingly complex, real-world problems are addressed, presenting researchers a continuing challenge to find ever more effective means for approaching project scheduling. This dissertation introduces a project scheduling problem which is applicable across a broad spectrum of real-world situations. The problem is based on the well-known Resource-Constrained Project Scheduling Problem, extended in this dissertation to include generalized precedence with minimal and maximal time lags and expediting resources. The problem is further extended to include multiple projects which have generalized precedence, renewable and nonrenewable resources, and expediting resources at the program level.

The problem presented in this dissertation is one not previously addressed in the literature nor is it one to which the existing specialized project scheduling methodologies can be directly applied. This dissertation presents a decomposition approach for solving the problem, including algorithms for solving the resulting decomposed subproblems and the master problem. This dissertation also describes a methodology for generating instances of the new problem, extending the way existing problem generators describe and construct network structures and this class of problem. The applicability of the methodologies presented is demonstrated through extensive empirical testing.

A DECOMPOSITION APPROACH FOR THE MULTI-MODAL, RESOURCE-CONSTRAINED, MULTI-PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE AND EXPEDITING RESOURCES

I. Introduction

Overview

The field of project scheduling has received a great deal of study for manyyears with a steady evolution of problem complexity and solution methodologies. As solution methodologies and technologies improve, increasingly complex, real-world problems are addressed, presenting researchers a continuing challenge to find ever more effective means for approaching project scheduling. This dissertation addresses a project scheduling problem which is applicable across a broad spectrum of real-world situations. The total problem is one not previously addressed in the literature nor is it one to which the existing specialized project scheduling methodologies can be directly applied. This dissertation presents a decomposition approach for solving the problem, including algorithms for solving the resulting decomposed subproblems and the master problem. This dissertation also describes a methodology for generating instances of the new problem, extending the way existing problem generators describe and construct network structures and this class of problem.

Background

The scheduling problem introduced by this dissertation is the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP). In most general terms, the goal of the MRCMPSP-GPR/EXP is to identify a start time for each activity in a set of related activities in order to accomplish some objective, where various classes of resources exist and their quantity can be varied. The way in which activities are related and the objective to be accomplished are what differentiate the MRCMPSP-GPR/EXP from other scheduling problems in the literature.

A set of related activities is referred to as a *project*. Projects can take on many forms, ranging from conducting cancer research or building a highway to running a political campaign or conducting a military operation. A project may be as complex as designing and building a stealth aircraft or as simple as planning a company picnic. Whatever the nature of the project, its component activities are related in two ways. First, activities may be precedence related. If one activity cannot start until another activity has finished, the two activities are said to have a *standard precedence relationship*. If, on the other hand, the start times of two activities are related, the activities are said to have a *generalized precedence relationship*. More specifically, if Activity B cannot start until some time after the start of Activity A, then Activity A is a *generalized predecessor* of Activity B with a *minimal time lag*. If Activity B must start *before* some time after the start of Activity A, then Activity A is a *generalized predecessor* of Activity B with a *maximal time lag*.

As an example of precedence relationships, consider a few of the activities required to successfully launch two fighter aircraft. Each fighter must be fueled and each must be loaded with bombs. Typically, fueling the aircraft must be completed before the bomb loaders can begin their activity (fuel and bombs do not mix well). Therefore, fueling and bomb loading have a standard precedence relationship where fueling precedes loading. When it comes time for the fighters to take off, they can use the same runway, taking off one after the other, or they can use different runways and take off at the same time. The key consideration, though, may not be that one takes off before the other, but that both take off at relatively close times so that they can rendezvous in the air and continue the mission without one having to wait a long time for the other. In this case, their takeoff times exhibit a generalized precedence relationship. If either Fighter A or Fighter B can take off first, but both must take off within a two-minute interval of each other, then one might say that Fighter A is a generalized predecessor of Fighter B with a minimal time lag of –2 minutes and a maximal time lag of +2 minutes. In this way, Fighter B could actually take off before Fighter A, but in any case, they will both take off within the desired two-minute time interval.

The second way activities can be related is by having a requirement for common resources. Both fighter aircraft require JP-4 fuel and a crew to do the fueling. If the total amount of fuel available during an air campaign is fixed, then fuel is a *nonrenewable resource*; the fuel is gone once used. Fueling crews, by contrast, are *renewable resources*, since they can be used repeatedly, but their availability at any given time is limited; there may be only two fueling crews on base. Other resources are doubly-constrained, being both renewable and nonrenewable. Bombs would be doubly-constrained if their total availability during the air campaign were limited (making them nonrenewable) *and* if the number of bombs available at any given time were limited (making them renewable). This would be the case if a base could store only up to a specific number of bombs in its bomb dump. Bomb loaders could not load more bombs at any given time than there are bombs currently in the bomb dump, but the bomb dump can be restocked up until thetime that the total number of bombs available for the campaign are exhausted.

Finding a start time for each activity in a project such that the precedence relationships are maintained and total usage of resources is within the limits of their availability is the act of *scheduling*. To further complicate the scheduling process is the potential for multiple activity execution modes. Activity execution modes are alternate ways to accomplish an activity and define the duration and resource requirements of theactivity. Suppose that the bomb dump in the above example needed to be replenished. There are a number of ways this could be done. The bombs could be loaded on two C-5 aircraft and flown straight to the base. This might take a single day. The bombs could also be loaded on a supply ship, ferried to the nearest port, and then loaded on flatbed trucks for the rest of the journey to the base. This might take two weeks. Either option for restocking the bomb dump is a legitimate execution mode, and which mode is chosen depends on how much time and how many C-5, ships, and flatbed trucks are available.

The choice of which mode is used to restock the bomb dump will likely affect other activities which depend on having bombs in the dump. The choice of mode for restocking is, at the same time, affected by other activities and their execution modes. Suppose the fighters will carry either four 2000-pound bombs or eight 500-pound bombs (two possible modes for striking targets). If the bomb dump is out of 2000-pound bombs and the ship-flatbed mode is used to replenish them, either fighters will have to use 500-pound bombs for two weeks or strike missions will have to be delayed. Consequently, the C-5 mode might be preferred. Unfortunately, if C-5 aircraft are used for other activities and are unavailable during this time, the ship-flatbed mode may be the only mode possible. (This dependency of activities on other activities is, in fact, a key motivator for careful *a priori* scheduling.)

The careful selection of an execution mode for each activity is an important part of resolving resource conflicts and is an integral part of scheduling in the presence of multiple modes. Which modes are selected will determine how long it takes to complete a project and will determine which resources are critical and which are not. Resource limitations may force a scheduler to choose non-preferred modes or to delay activities. In many situations, fortunately, resource limitations may be eased through *expediting resources*. The concept of expediting resources is simply to increase the availability of a critical resource to provide hopefully better scheduling options. If additional C-5 aircraft could be obtained, then the bomb dump might be replenished sooner and better weaponeering modes made possible for strike missions. In this situation, obtaining those C-5s seems a logical decision. However, there is a tradeoff. While regularly available resources are assumed to be available at no cost (they are *company-owned*, so to speak), expediting resources are available only at a cost. Expediting resources might be purchased, rented, or leased. To a construction company, they might be temporary workers. For the C-5s, they might be aircraft that need to be refurbished, they might be borrowed from another theater (in this case, the cost may not be dollars but opportunity cost to the lending theater), or they may be civilian aircraft with similar carriage capacity leased from a commercial air freight company.

While modes and expediting resources both give schedulers greater flexibility, they are fundamentally different. Modes typically trade greater resource requirements for shorter durations, while expediting resources affect the availability of resources (*i.e.*, demand vs. supply). Thus, modes enable shorter activity durations, while expediting resources enable a more *compact* schedule. In other words, a scheduler can always select the modes which give the shortest activity durations possible. This selection, however, may be resource-feasible only if some of the activities are delayed. Expediting resources raise the limits on resource availability and can reduce the number of activities that need to be delayed (hence, a more compact schedule).

To this point, the fundamentals of precedence relationships, resources, and activity execution modes have been explained. These are the characteristics of the MRCMPSP-GPR/EXP that constrain which choices of execution modes, start times, and expediting resource use form feasible schedules. Which of these feasible schedules is best, though, depends on the objective of the scheduler. For the MRCMPSP-GPR/EXP, a variety of objectives are available.

The most general objective of the MRCMPSP-GPR/EXP is to minimize the schedule cost. Costs come in three forms. As previously mentioned, using expediting resources incurs a cost. The mode and start time selected for an activity may also incur a cost. In a construction activity, the decision to hire skilled labor or unskilled labor is a mode choice which impacts the labor cost associated with the activity. The activity may also require a cash outlay which increases over time so that a delay in the start of the activity results in an increase in the cash outlay. The third type of cost is the project completion cost. Many projects are either rewarded for finishing earlier than planned or penalized for finishing later. The bonus / penalty is a direct cost to the project (note that a bonus is just a negative cost).

Other scheduling objectives are special cases of the cost minimization objective. Some of these are described in Chapter III.

The final characteristic of the MRCMPSP-GPR/EXP is its multi-project nature. The importance of identifying a problem as representing a single project or as having multiple projects is in the decomposability of the problem. In essence, a single-project problem and a multi-project problem are fundamentally the same except that the multi-project problem has distinct sets of activities in which the activities are in some way more strongly related. A set of activities, for example, may use some types of resources not used by any other set. Additionally, the activities in a set may have many precedence relationships with other activities in the set, but very few with activities in other sets. When a problem can be subdivided into such distinct sets, the sets are tagged as *projects* and the set of projects is called a multi-project *program*. By their nature, the multi-project program demonstrates a block-angular structure and can be decomposed using procedures such as that proposed by Sweeney and Murphy (1979). The Sweeney-Murphy approach is used in this dissertation to facilitate the solution of decomposable problems.

Though the MRCMPSP-GPR/EXP has not been addressed in the open literature, the literature is full of methodologies for solving related project scheduling problems. Generally, attempts to solve project scheduling problems with more traditional techniques, such as general integer programming (IP) approaches, have been unsuccessful (Demeulemeester and Herroelen, 1992: 1803). Researchers have, therefore, turned towards the development of specialized algorithms for solving project scheduling problems. This dissertation develops such an approach for the MRCMPSP-GPR/EXP, including algorithms for solving single- and multiple-project instances.

There has also been an effort in the literature to develop problem generators to provide consistent test cases for the multitude of solution methodologies. Unfortunately, most use measures of network complexity which provides inconsistent and confusing results (see Chapter IV). By contrast, there is a measure of network complexity which is recognized to be far superior, but only one generator attempts to use this measure. Even then, this generator constructs networks using the obsolete measure and then calculates the corresponding value of the superior measure. If the network has the desired value, it is kept; otherwise, it is discarded and another network is constructed and tested.

This dissertation develops a methodology which constructs networks using the superior measure directly. The network methodology is then built upon to develop a problem generator which is capable of generating all of the characteristics of the MRCMPSP-GPR/EXP. No other generator is currently known to provide standard and generalized precedence, expediting resources, and multiple projects.

Significant progress has been made since the 1950s in the field of project scheduling. Even so, major gaps still exist. As computational efficiency and power increase, new problems can be proposed to consider these gaps. This dissertation considers such a problem area when considering multi-modal problems with expediting resources.

Research Issues

The problem of scheduling multi-project programs with multiple modes, generalized precedence, and expediting resources has not been addressed in the project scheduling literature. No specialized solution methodologies have been developed to solve the problem and standard integer programming approaches are currently inadequate for solving problems of this type in an operationally reasonable amount of time. In addition, no existing problem generator is capable of constructing problems with the characteristics of the MRCMPSP-GPR/EXP. Furthermore, the problem generators that *are* presented in the literature generally use measures of network complexity that poorly reflect the true nature of project networks.

Research Objectives

The research presented in this dissertation fills a number of voids in the expanding field of project scheduling. Specifically, the research accomplishes the following objectives:

1. It introduces the MRCMPSP-GPR/EXP to the project scheduling literature, including a mathematical formulation of the problem. The problem includes:
   (a) Multiple activity execution modes.
   (b) Renewable, nonrenewable, and doubly-constrained resources.
   (c) Standard and generalized precedence between activities. Generalized precedence includes both minimal and maximal time lags.
   (d) Expediting resource availability which can be used by any activity requiring that resource.

   (e) An objective to minimize project / program costs, including mode costs, project / program completion costs, and expediting resource costs.
   (f) Multiple projects exhibiting characteristics (a) – (e) at both the project level and program level.
2. It presents a problem generator capable of constructing instances of the MRCMPSP-GPR/EXP.
   (a) The generator produces problem instances with all of the characteristics of the MRCMPSP-GPR/EXP.
   (b) The generator constructs project networks in a way which directly exploits a measure of network complexity which reflects the nature of networks more accurately than the measures more commonly used.
3. It develops a specialized algorithm for solving single-project instances of the MRCMPSP-GPR/EXP.
   (a) The algorithm is based on an approach for resource-constrained project scheduling from the literature, extended for multiple modes, generalized precedence with minimal and maximal time lags, expediting resource availability, and a cost-minimizing objective function.
   (b) The algorithm is designed to generate a set of *k*-best solutions to the problem rather than a single optimal solution.
4. It uses the Sweeney-Murphy Decomposition principle to decompose multi-project instances of the MRCMPSP-GPR/EXP for more efficient scheduling.
   (a) Alternate methods for finding multipliers used to relax the coupling constraints in the original problem are developed.
   (b) Once the original problem is decomposed into subproblems, the specialized algorithm developed for single-project instances of the MRCMPSP-GPR/EXP is used to solve the subproblems.
   (c) An algorithm for solving the master problem is developed.
   (d) Techniques for both speeding solution of the master problem and for accelerating the iterative solution process are developed.
   (e) An error in the approach as originally presented by Sweeney and Murphy (1979) is explained and the impact of that error is described.

5. The problem generator designed in Objective 2 is used to generate test instances which are solved to test the methodologies developed in Objectives 3 and 4.

Approach

The project scheduling literature has been reviewed to identify project scheduling problems, and their mathematical formulations, which have characteristics in common with the MRCMPSP-GPR/EXP. A number of such problems have been found. Where possible, formulations of relevant objective functions and constraints have been borrowed from the literature and modified, as necessary, to reflect characteristics unique to the MRCMPSP-GPR/EXP (*e.g.*, extending constraints for multiple projects and expediting resources). A complete mathematical formulation of the MRCMPSP-GPR/EXP is presented in Chapter III.

Chapter III also introduces a decomposition of the problem, using classical Lagrangian relaxation. Specifically, the multi-project nature of the MRCMPSP-GPR/EXP demonstrates a block-angular structure which can be exploited to decompose the problem into a number of semi-independent subproblems and a master problem. The subproblems represent the component projects, each with its own set of precedence and resource constraints. The master problem enforces the program-level precedence and resource constraints. Sweeney and Murphy (1979) present an approach for solving the decomposed problem by, first, generating a set of *k*-best solutions to each subproblem. The subproblem solutions are then combined to form a master problem (a restriction of the original problem) which is solved to find a combination of subproblem solutions (one solution from each subproblem) which is feasible to the program-level constraints and which is optimal among all such combinations. Sweeney and Murphy provide a condition under which the optimal solution to the master problem is also optimal to the original problem.

The subproblems are solved using a specialized algorithm developed in Chapter V. The algorithm is an implicit enumeration scheme based on the algorithm by Talbot (1982). The algorithm has been extended to incorporate the characteristics of the MRCMPSP-GPR/EXP. The algorithm has also been modified to generate a set of *k*-best solutions, rather than a single optimal. The resulting algorithm is further extended with a set of bounding and feasibility rules designed to speed the solution process. Though designed specifically to solve the subproblems of a decomposed multi-project problem, the specialized algorithm of Chapter V is equally applicable as a *stand-alone* scheduler for single-project instances. Extensive testing of the algorithm is reported, including a comparison of results to those obtained by solving the test problems using a standard commercial IP solver.

Chapter VI presents a procedure for relaxing / decomposing a multi-project problem and then for iteratively solving the subproblems and the master problem. The basic procedure is based on the approach proposed by Sweeney and Murphy (1979). Sweeney and Murphy, however, do not prescribe a methodology for solving either the subproblems or the master problem. In their paper, they use a standard IP approach for solving both the subproblems and the master problem. The procedure proposed in Chapter VI uses the algorithm developed in Chapter V for solving the subproblems. Chapter VI, then, develops an implicit enumeration algorithm for solving the master problem.

Chapter VI also proposes alternative approaches for generating the multipliers used to relax the original problem. These approaches are based on (1) an approach by Nauss (1979) for estimating the marginal benefit of resources in an IP and (2) the concept of Average Utilization Factor described by Kurtulus and Davis (1982) and Kurtulus and Narula (1985). Finally, Chapter VI provides additional schemes for accelerating solution of the master problem. Testing of the decomposition approach, using alternative multipliers and acceleration schemes, is reported. Results are compared to solving the problems in *whole* (using the algorithm of Chapter V) versus through decomposition.

Summary

This chapter introduced the subject scheduling problem, provided an overview of the research issues and objectives, and summarized the research approach. Chapter II presents a review of the pertinent literature on project scheduling and problem decomposition. Chapter III provides a mathematical formulation of the scheduling problem and shows how the problem may be decomposed. Chapter IV details a generator for constructing test problems, including an algorithm for generating network structures using an improved measure of network complexity. Chapters V and VI, respectively, develop algorithms for solving single-project and multi-project instances of the problem. Finally, a summary of the research, its contributions, and suggestions for future research are outlined in Chapter VII.

II. Literature Review

Introduction

The literature is replete with models representing a wide variety of project scheduling problems. This chapter reviews the models which provide a foundation for the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP). The chapter also describes the myriad of approaches developed to solve project scheduling problems, including the use of problem decomposition methods. The approaches are further evaluated in Chapters V and VI for their applicability to the MRCMPSP-GPR/EXP.

Mathematical formulations are provided for the more important problems discussed in this chapter. Note that the equations used in each of the model formulations are sequentially numbered. Once an equation has been numbered, any reuse of the equation will bear the original number. This consistency in numbering will provide insight into how one model builds upon another. Note also that the abbreviations used to denote the different scheduling problems are summarized in Appendix A for easy reference. However, the notation used in the problem formulations may not, in all cases, be consistent with the notation included in Appendix A. The formulations below retain the variable definitions given by the original authors and may, therefore, change from one formulation to another. Consequently, each variable used in a formulation is defined for that formulation only. In those cases that a variable in this chapter is inconsistent with the variables listed in Appendix A, the inconsistent variable is not used in subsequent chapters.

Problem Hierarchy

The next section provides a review of project scheduling problems from the literature, most of which are special cases of the MRCMPSP-GPR/EXP. To set the stage for this review, Figure 2-1 diagrams the hierarchical relationship of the more important problems and the MRCMPSP-GPR/EXP. Each problem is numbered so it can be easily referred to in the subsequent sections. Note that Problem 1, at the bottom of the diagram, is the resource-unconstrained Project Scheduling Problem. At the top of the diagram, Problem 12, is the MRCMPSP-GPR/EXP. Intermediate problems are *constructed* by adding characteristics to problems at a lower level or by relaxing characteristics of problems at a higher level.

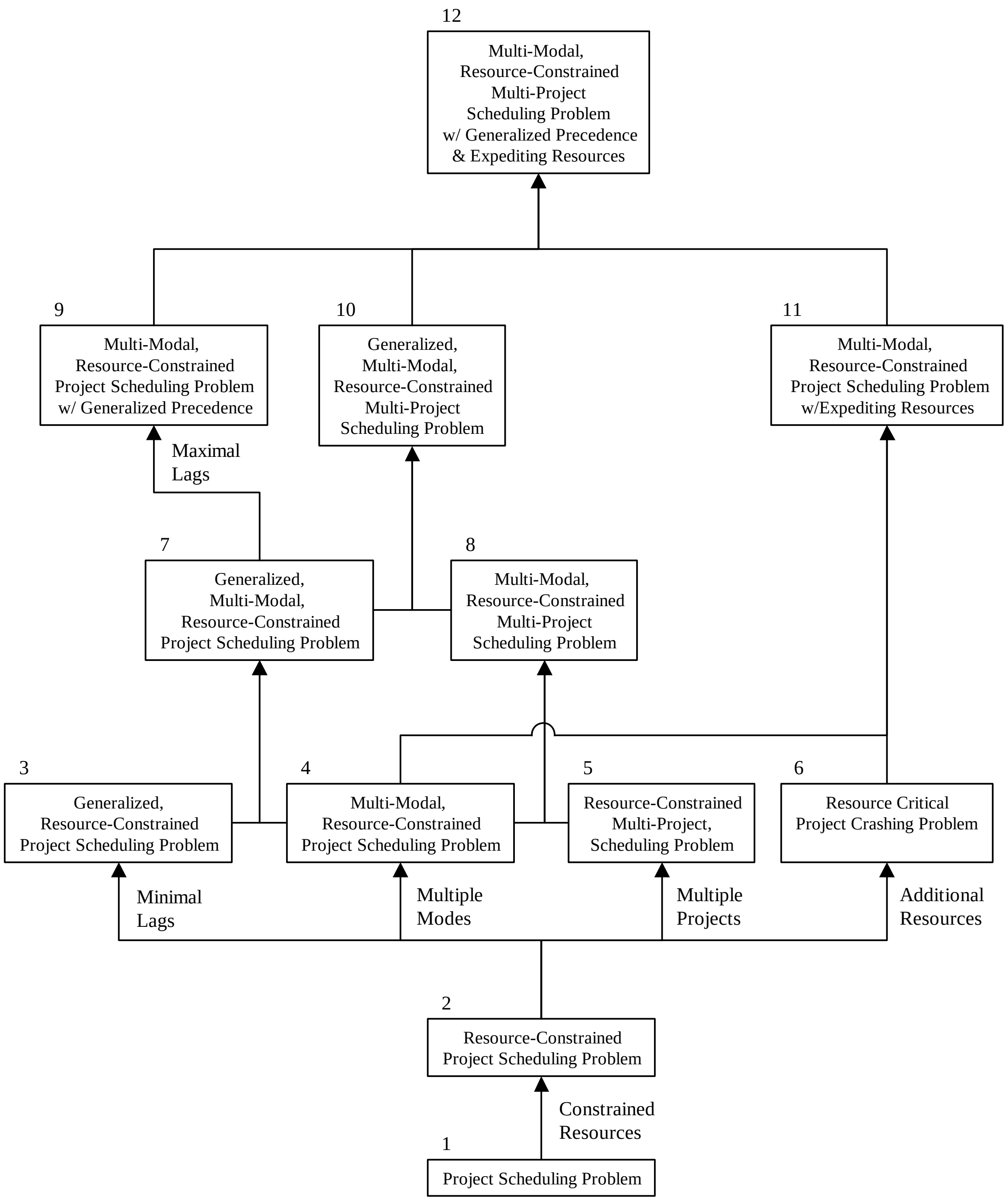

12
Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem w/ Generalized Precedence & Expediting Resources
9
Multi-Modal, Resource-Constrained Project Scheduling Problem w/ Generalized Precedence
10
Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem
11
Multi-Modal, Resource-Constrained Project Scheduling Problem w/Expediting Resources
Maximal Lags
7
Generalized, Multi-Modal, Resource-Constrained Project Scheduling Problem
8
Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem
3
Generalized, Resource-Constrained Project Scheduling Problem
4
Multi-Modal, Resource-Constrained Project Scheduling Problem
5
Resource-Constrained Multi-Project, Scheduling Problem
6
Resource Critical Project Crashing Problem
Minimal Lags
Multiple Modes
Multiple Projects
Additional Resources
2
Resource-Constrained Project Scheduling Problem
Constrained Resources
1
Project Scheduling Problem

Figure 2-1. Problem Hierarchy

Table 2-1 is also provided as a tabular summary of the most important characteristics of the problems included in Figure 2-1.

Table 2-1. Key Features of Project Scheduling Problems

	1. PSP	2. RCPSP	3. GRCPSP	4. MRCPSP	5. RCMPSP	6. RCPCP	7. GMRCPSP	8. MRCMPSP	9. MRCPSP-GPR	10. GMRCMPSP	11. MRCPSP-EXP	12. MRCMPSP-GPR/EXP
Standard Precedence	x	x	x	x	x	x	x	x	x	x	x	x
Generalized Prec (Min Lags)			x				x		x	x		x
Generalized Prec (Max Lags)									x			x
Multiple Modes				x			x	x	x	x	x	x
Expediting Resources						x					x	x
Multi-Project Problems										x		x
w/Program Nonrenew Res										x		x
w/Program Renew Res												x
w/Time-Related Projects												x
Regular Measure of Perf	x	x	x	x	x	x	x	x	x	x	x	x
Non-Regular Measure of Perf						x					x	x

Single-Project Scheduling

This section reviews single-project scheduling problems and the approaches used to solve them. The section begins with the resource-unconstrained Project Scheduling Problem as the basis for the subsequent problems. Resource constraints, activity crashing, expediting resources, and generalized precedence are then discussed and related problems are introduced.

<u>The Project Scheduling Problem</u>. The Project Scheduling Problem (PSP), Problem 1 in Figure 2-1, dates to the late 1950s (see Kelley, 1961) when the Critical Path Method (CPM) was developed (Icmeli, 1993). The PSP is the problem of scheduling a set of activities in a project to minimize the makespan of the project. Activities have fixed and known durations. Any given pair of activities (graphically represented by nodes in the project network) may be related by simple

finish-start precedence relationships (represented by network arcs) where one activity must finish before another may start. The mathematical formulation of the problem is given by:

$$\text{Minimize} \quad s_J \tag{1}$$

$$\text{subject to} \qquad s_j \ ?\ s_i \ ?\ d_i\,, \qquad ?\ i\ ?\ O_j \tag{2}$$

$$s_j \ ?\ 0 \text{ and integer}\,, \qquad ?\ j \tag{3}$$

where

s_j	=	start time of activity j
d_j	=	duration of activity j
J	=	terminal node or activity
O_j	=	set of predecessors of activity j

The PSP may be solved using the CPM. The CPM allows the activities of a project to be scheduled in a way which maintains the precedence relationships between the activities and which minimizes the duration of the project. This is done by starting each activity as soon as all of its predecessors are complete (see Shtub *et al.*, 1994: 338-341). These start times are specifically referred to as the *early start times* of the activities. The completion time of the last activity completed is the *minimum completion time* of the project.

A backwards recursion may also be made on the network where all activities are scheduled to start as late as possible while still completing the project at its minimum completion time and maintaining the precedence relationships. These are the *late start times* of the activities. Those activities whose early and late start times are identical are called *critical activities*. Each network path consisting only of critical activities is called a *critical path* (from which the name *Critical Path Method* comes). Kelley (1961) provides the mathematical basis for the CPM.

While the applicability of the CPM is limited because it deals only with the time aspect of the project without consideration for resource restrictions (see Icmeli, 1993), it remains a useful tool. It is used in many enumeration schemes to provide activity start time bounds which reduce the solution space which needs to be enumerated.

Resource Constraints. A significant limitation of the PSP is that resources are assumed to be available at ample enough levels such that they do not constrain the schedule. In reality, project resources are often limited to the point where the start times of some activities have to be delayed because insufficient resources are available. The consideration of limited resources has given rise to a myriad of resource-constrained problems, the most basic of which is the Resource-Constrained Project Scheduling Problem (RCPSP), Problem 2 in Figure 2-1.

The RCPSP has the same finish-start precedence structure and makespan minimization objective function as the PSP. However, each activity now requires a certain amount of some limited resources. The demand for a resource by an activity is assumed constant for the duration of the activity and resource availability per period is constant. Generally, there are insufficient resources in one or more periods to schedule all of the critical path activities at their earliest start time. (Consequently, the CPM alone is insufficient for developing a feasible schedule.)

One of the earliest formulations of the RCPSP was proposed by Bowman (1959) for job shop scheduling. In Bowman's formulation, shown below, 0-1 variables describe whether or not an activity is in progress in any given time period. Constraints (5) assure that each activity is in progress during the same number of time periods as the activity has units of duration. Constraints (6) prohibit activity preemption (*i.e.,* an activity being interrupted once started). Constraints (7) enforce precedence relationships by assuring that, if activity *i* precedes activity *j*, activity *j* can be in process at time *t** only if the number of periods that activity *i* is in process before time *t** is greater than or equal to the duration of activity *i*. Resource use is limited by Constraints (8). Finally, the objective function is to minimize the project duration.

$$\text{Minimize} \quad \sum_{t=t_0}^{t_1} 4^{t-t_0} \sum_{j=1}^{J} x_{jt} \tag{4}$$

$$\text{subject to} \quad \sum_{t=e_j}^{l_j+d_j-1} x_{jt} = d_j, \quad \forall j \tag{5}$$

$$d_j x_{jt} - d_j x_{j(t+1)} + \sum_{t=t^*+2}^{l_j+d_j-1} x_{jt} \le d_j, \quad t^* \in [e_j, l_j + 2], \forall j \tag{6}$$

$$d_i x_{jt^*} \le \sum_{t=1}^{t^*-1} x_{it}, \quad \forall i \in O_j, j, t^* \tag{7}$$

$$\sum_{j=1}^{J} r_{jq} x_{jt} \le R_{qt}, \quad \forall q, t \tag{8}$$

$$x_{jt} \in \{0,1\}, \quad \forall j, t \tag{9}$$

where

x_{jt} = 1 if activity j is in process at time t ; 0, otherwise

J = terminal node or activity

d_j = duration of activity j

e_j = early start time of activity j

l_j = late start time of activity j

t_0 = early project completion time

t_1 = late project completion time

r_{jq} = requirement for resource q by activity j

R_{qt} = availability of resource q in time t

O_j = set of predecessors of activity j

Since, in Bowman’s formulation, an activity requires a variable for each time period from the activity’s earliest possible start time to its latest possible finish time, this formulation generally requires many more 0-1 variables than later formulations (described next). Bowman gives an illustrative job shop example with three products and four machines. He shows that even this small problem would require 300 to 600 variables, depending on the number of time frames chosen, and even more constraints (Bowman, 1959: 624). The Bowman formulation, however, has still found utility in later research efforts (*e.g.*, Deckro and Hebert, 1989).

Pritsker *et al*. (1969) developed a 0-1 formulation of the RCPSP which provides considerable economy over the Bowman formulation. Their formulation, the Pritsker-Watters-Wolfe (PWW) model, is based on 0-1 variables which indicate the time periods in which an activity may be completed. Since the set of possible completion times of an activity can be a small subset of all of the times an activity may be in progress, typically far fewer variables are required.

In the PWW model, shown below, Constraints (11) assure that each activity completes only once. Precedence relationships are enforced by Constraints (12) while resource limits are enforced by Constraints (8). The objective shown is to minimize the project makespan.

$$\text{Minimize} \quad \overset{T}{\underset{t?1}{?}} tx_{Jt} \tag{10}$$

$$\text{subject to} \quad \overset{l_j}{\underset{t?e_j}{?}} x_{jt} ? 1, \quad ? j \tag{11}$$

$$\overset{l_i}{\underset{t ? e_i}{?}} t x_{it} ? \overset{l_j}{\underset{t ? e_j}{?}} t x_{jt} ? d_i, \quad ? i ? O_j, j \tag{12}$$

$$\overset{J}{\underset{j ? 1}{?}} r_{jq} x_{jt} ? R_{qt}, \quad ? q, t \tag{8}$$

$$x_{jt} ? ?0,1?, \quad ? j, t \tag{9}$$

where

x_{jt}	=	1 if activity *j* completes at time *t* ; 0, otherwise
J	=	terminal node or activity
d_j	=	duration of activity *j*
e_j	=	early start time of activity *j*
l_j	=	late start time of activity *j*
T	=	late project completion time
r_{jq}	=	requirement for resource *q* by activity *j*
R_{qt}	=	availability of resource *q* in time *t*
O_j	=	set of predecessors of activity *j*

In an example, Pritsker *et al.* present a three-project, eight-activity (total), three-resource problem. Their formulation requires 33 variables and 37 constraints (Pritsker *et al.*, 1969: 107). This is an improvement over the 72 variables and 125 constraints (50 variables and 94 constraints with careful size reduction) required by the Bowman formulation of the problem (Pritsker *et al.*, 1969: 107). The PWW model has been used extensively by other authors and is the model upon which the mathematical formulation in Chapter III for the MRCMPSP-GPR/EXP is based.

Blazewicz *et al.* (1983) show that the RCPSP is a generalization of the job shop scheduling problem and, as such, belongs to the NP-complete complexity class. Consequently, the breadth of approaches reported for solving the RCPSP has met with mixed success. The remainder of this subsection discusses the breadth of solution approaches for the RCPSP.

Mathematical Programming. Pritsker *et al.* solve their example problem using a general integer programming (IP) code developed by Geoffrion (Pritsker et al., 1969: 106). Other authors have also used general IP approaches to solve the RCPSP (*e.g.*, Bowman, 1959; Patterson and Huber, 1974; Patterson and Roth, 1976; Deckro and Hebert, 1989; Icmeli and Rom, 1996). One of the characteristics of the RCPSP which has been exploited by some authors to improve the efficiency of general IP approaches to the RCPSP is the existence of special ordered sets (SOS) of variables.

Beale and Tomlin (1969) introduced the concept of SOS variables. A special ordered set of variables of type 1 (SOS1) is a set of variables (continuous or integer) within which exactly one variable must be non-zero. A special ordered set of variables of type 2 (SOS2) is a set of variables within which at most two can be non-zero. In the case of SOS2 variables, the two non-zero variables must be adjacent in the ordering given to the set (Williams, 1985: 173). Constraints (11) are SOS1 variables since only one x_{jt} will be non-zero for each activity j.

The restriction that a set of variables belongs to SOS1 or SOS2 is easily modeled using binary variables and constraints, as in Constraints (11). The great computational advantage to be gained, however, comes from treating these sets algorithmically (Williams, 1985: 173). Bean (1984) points out that a general n-variable binary problem has an enumeration tree with 2^n branches. If the variables are separated into m SOS1 sets, where the ith set contains n_i variables and $n = \sum_{i=1}^{m} n_i$, then only $\prod_{i=1}^{m} n_i$ of the 2^n branches mentioned above are feasible in the multiple choice constraints defined by this partitioning. Bean presents a branch-and-bound algorithm which exploits SOS1 variables. The algorithm is successfully applied to a number of problems with up to 400 binary variables. Tripathy (1984) uses a branch-and-bound algorithm with SOS1 variables as part of a solution methodology for the school timetabling problem. He solves a problem with 3384 variables.

Despite the general usefulness of SOS variables, Patterson (1984) reports that in his comparison of exact approaches for solving the RCPSP, one approach that was considered for the comparison was solving the problem using a general purpose 0-1 program solver using Tomlin's integrated SOS procedure. The approach was eliminated because it could solve only the smallest of problem instances in the time imposed. Demeulemeester and Herroelen (1992: 1803) also report that, while the RCPSP is typically formulated as a straightforward integer program, standard IP approaches have generally proven unsuccessful. Researchers have thus turned to specialized algorithms for finding exact, or optimal, solutions (Demeulemeester and Herroelen, 1992: 1803).

Graph-Based Approaches. Balas (1970) represents the RCPSP as a disjunctive graph with the goal of eliminating the need to consider individual time periods over the project horizon.

Solutions are obtained by finding a minimum-arc disjunctive graph subject to stability conditions. Stability is represented by a generalized coefficient of internal stability – a check for feasibility with respect to available resources. Gorenstein (1972) shows how the generalized coefficient of internal stability can be calculated using a maximum-flow computation on a bipartite graph. While the network representation of the problem eliminates the dependence of the number of variables on the time horizon, Christofides *et al*. (1987) suggest that the procedure proposed for guaranteeing the feasibility of the solution requires a large computational effort that limits the use of the algorithm.

Davis and Heidorn (1971) present an algorithm where activities are broken into unit-length tasks. An *A-network* is formed where nodes represent subsets of tasks and arcs connect subsets which could be completed on adjacent days. The minimization of the project duration, then, is a matter of finding a path from start to finish in the A-network which contains a minimal number of arcs. The advantage of this procedure is that the subdivision of activities into tasks of unit length easily allows for job splitting (without any additional computational effort) and activity resource requirements can vary over the duration of the activity. The drawback to the procedure is that the number of subsets grows rapidly with problem size and only very small problems can be handled (Davis and Heidorn, 1971: B-815; Christofides *et al*., 1987: 263). Davis and Heidorn test their algorithm on 65 problems, each containing 50 to 95 unit-duration tasks (30 original activities) and involving 3 resource types. Optimal solutions were found for 48 of these problems (Davis and Heidorn, 1971: B-815).

Implicit Enumeration. Most implicit enumeration methods use partial schedules which are associated with the nodes of an enumeration tree. Branching from nodes equates to extending partial schedules. Dominance rules and lower bounds serve to reduce the number of alternatives for extending partial schedules. Methods differ in the way they branch and prune.

Talbot and Patterson (1978), rather than using a 0-1 formulation of the problem, represent the problem in structured, compact integer arrays which are directly employed by the solution procedure. This representation results in considerable memory savings. The solution procedure uses implicit enumeration of all feasible schedules, relying on network cuts to fathom partial schedules which cannot lead to an improved solution. Talbot and Patterson conduct a comparative study of their algorithm using 50 test problems. They show their algorithm to be more efficient than other enumeration procedures and competitive with the best available branch-and-bound procedure, while requiring considerably less computer storage. They claim, however, that the likelihood of obtaining an optimal solution for projects containing more than 50 activities within a reasonable amount of computation time is low. In fact, they encountered a few projects containing as few as 35 activities that could not be solved in a reasonable amount of time with their approach in 1978 (Talbot and Patterson, 1978: 1172).

Stinson *et al*. (1978) present a branch-and-bound procedure where nodes in the solution tree correspond to precedence and resource feasible assignments for a subset of the activities of the project. In a comparison of exact approaches for solving the RCPSP, Patterson (1984) determines Stinson's Procedure to be the fastest of the procedures tested at that time. Patterson's 110 test problems include up to 50 activities and 3 resource types.

Christofides *et al*. (1987) and Demeulemeester and Herroelen (1992) use the concept of *delay alternatives*. Christofides *et al*. (1987) present a delay alternative branch-and-bound algorithm based on the idea of using disjunctive arcs for resolving conflicts that are created whenever sets of activities have to be scheduled whose total resource requirements exceed the resource availabilities in some periods. For fathoming branches, the authors examine four lower bounds and computational results appear promising. Demeulemeester *et al*. (1994), however, present a counterexample to show that the procedure proposed by Christofides *et al*. does not guarantee an optimal solution. Demeulemeester *et al*. suggest a modification to this procedure which does guarantee optimality. Their modification expands the set of source nodes considered for delay arcs to ensure that partial schedules which may lead to an optimal schedule are not fathomed prematurely. Demeulemeester *et al*. test their modified approach using Patterson's 110 test problems and find that their approach optimally solves all of these test problems.

Demeulemeester and Herroelen (1992) present another delay alternative branch-and-bounding procedure where the nodes represent partial schedules in which finish times have been temporarily assigned to a subset of activities of the project. Activities are scheduled as soon as precedence and resource constraints allow, but they may be delayed based on decisions made in later stages of the search process. The algorithm shows promising results, being an average of almost 12 times faster than the best-first procedure by Stinson *et al*. previously reported by Patterson (1984) to be the most effective and efficient on the problem set considered. The algorithm is tested on projects with at most 51 activities and 3 resources.

Other Approaches. Patterson and Huber (1974) present a minimum bounding algorithm and a maximum bounding algorithm to solve the 0-1 formulation of the problem. The minimum bounding algorithm begins by fixing the project horizon at the CPM shortest possible project duration and then solving the 0-1 problem to determine feasibility. If feasible, the schedule based on the CPM duration is optimal. If infeasible, the project horizon is extended by one time unit and the 0-1 problem solved again to determine feasibility. The algorithm continues until a feasible (and consequently, optimal) schedule is found. The maximum bounding algorithm is similar except that a feasible schedule is first determined using an appropriate heuristic. The project horizon is then set at one time unit less than the project duration found with the heuristic and the 0-1 problem solved for feasibility. If infeasible, the duration from the heuristic solution is optimal. If feasible, the time horizon is again shortened and the process continued until there are no feasible schedules for a given project horizon. An optimal schedule is the last feasible schedule. Patterson and Huber demonstrate this approach to be more effective than 0-1 programming without bounding. On a set of 11 test problems, they show that less time was involved in examining a series of 0-1 problems for feasibility than was involved in solving one 0-1 problem optimally (Patterson and Huber, 1974: 997).

Zamani (under review) presents an algorithm which finds an optimal solution or a heuristic solution within a certain range of the optimal solution. His algorithm uses heuristic estimates which are continuously updated during the search process. At each level of the search tree, the heuristic estimates of partial solutions are updated by comparing them with those of their neighbors. The initial heuristic value of every partial schedule is a lower-bound on the completion of the project. Zamani reports that solution times compare favorably to other optimal algorithms, and the algorithm provides guaranteed performance bounds unlike other heuristics.

One variant of the RCPSP is the Resource Availability Cost Problem (RACP), proposed by Demeulemeester (1995). The RACP is the problem of minimizing renewable resource availability costs subject to a project due date. (Renewable resources are those which are limited on a per period basis.) More precisely, the per period availability of a resource is to be the same for all periods, but the objective of the problem is to determine what this resource level should be in order to complete the project by a fixed due date at minimal cost for resources. Demeulemeester uses a minimum bounding strategy to solve this problem. The strategy starts with minimal resource availabilities and solves a resource-constrained project scheduling decision problem to determine if a feasible schedule exists at the current levels of resource. If so, the schedule is optimal. Ifnot, resource availabilities are incrementally increased and the decision problem solved until a feasible, and optimal, schedule is found. The technique was successfully applied to a modification of Patterson's 110 test problems.

Activity Crashing. *Activity crashing* is the process of shortening the duration of an activity. The following discussion outlines a number of scheduling problems which differ in the way they crash an activity's duration.

Minimal Cost Project Network Problem. Wu and Li (1994) and Kamburowski (1995) discuss the Minimal Cost Project Network Problem (MCPNP). The concept of the problem is to crash a project network, without resource constraints, to minimize project costs. The direct costs of crashing activities are offset by indirect costs based on the duration of the project. Wu and Li (1994) outline a method for solving the problem using a minimum cut set algorithm. The key steps of the algorithm identify normal project durations, minimum cut sets, and the capacities of those cut sets. Kamburowski (1995), however, shows that the method of Wu and Li does not guarantee an optimal solution. He outlines his own optimal method which is also based on a minimum cut set and demonstrates the approach on an example with four activities.

Project Time/Cost Tradeoff Problem. The Project Time/Cost Tradeoff Problem (PTCTP) allows a project to be shortened by crashing the duration of one or more if its activities. That is, each activity has a normal duration and a crashed duration and, at a cost, the duration of an activity can be reduced from its normal duration to as short as its minimum crashed duration. The objective is to determine the start time and duration of each activity in order to complete the project by a fixed due date while minimizing the cost of crashing. The methods demonstrated for solving the problem include a minimum cut set algorithm (Phillips and Dessouky, 1977), a labeling algorithm (Elmaghraby, 1977: 58-118; Ford and Fulkerson, 1962: 151-162), a CPM time-cost tradeoff procedure (Moder *et al*., 1983: 237-251), and a Benders' Decomposition (Kuyumcu and Garcia-Diaz, 1994). While the crashing cost function is generally linear, Elmaghraby (1977) extends the model to include strictly convex cost functions, concave cost-duration functions, and discrete non-increasing functions.

Activity Duration Crashing Problem. The Activity Duration Crashing Problem, proposed by Deckro and Hebert (1989), is a discrete extension of the Project Time/Cost Tradeoff Problem, incorporating resource restrictions as well. The standard objective is to determine the start time and duration of each activity which minimizes the project duration subject to a budget for crashing and subject to resource availabilities. Deckro and Hebert base their model on Bowman's (1959) 0-1 formulation of the Resource-Constrained Project Scheduling Problem. They provide a five-activity, one-resource example which was solved using a commercial integer program solver (using branch-and-bound).

Multi-Modal, Resource-Constrained Project Scheduling Problem. The previous two problems seek to shorten the duration of activities from a *normal* duration to a crashed duration, typically at a per-period cost for crashing. The Multi-Modal, Resource-Constrained Project Scheduling Problem (MRCPSP), Problem 4 in Figure 2-1, is similar in that it allows tradeoffs between activity duration and cost. However, the *cost* incurred for changing the duration of an activity is not necessarily a monetary fee charge for each period the activity duration is shortened. Instead, the *cost* for changing the duration of an activity is incurring a different mix of required resources. More precisely, each activity can be performed in one of multiple execution modes. The mode of execution determines the activity's duration and resource requirements. For example, the U.S. Air Force's Air Mobility Command (AMC) may have the task of airlifting troops and supplies from the U.S. to a forward operating base (FOB) overseas. This task may have several possible execution modes. One mode may involve airlifting the troops and supplies, via C-5 aircraft, from the U.S. to a main operating base (MOB) overseas and then to the FOB using C-130 aircraft. An alternative mode may involve airlifting the troops and supplies directly from the U.S. to the FOB using C-17 aircraft. Obviously, the time and resources required to accomplish the task depends on the execution mode chosen.

The standard objective of the problem is to minimize the duration of the project, subject to resource constraints, by determining the start time and execution mode for each activity. The problem belongs to the NP-Hard complexity class (Kolisch, 1995: 26).

Following is the mathematical formulation of the MRCPSP. The model is almost identical to that of the RCPSP, the main difference being an additional index, *m*, is added to the decision variable to indicate which mode is selected for an activity.

$$\text{Minimize} \quad \overset{T}{\underset{t?1}{?}} tx_{J1t} \tag{13}$$

$$\text{subject to} \quad \overset{l_j}{\underset{t?e_j}{?}} \overset{M_j}{\underset{m?1}{?}} x_{jmt} ? 1, \quad ? j \tag{14}$$

$$\overset{l_i}{\underset{t?e_i}{?}} \overset{M_i}{\underset{m?1}{?}} tx_{imt} ? \overset{l_j}{\underset{t?e_j}{?}} \overset{M_j}{\underset{m?1}{?}} tx_{jmt} ? d_{im}, \quad ? i ? O_j, j \tag{15}$$

$$\overset{J}{\underset{j?1}{?}} \overset{M_j}{\underset{m?1}{?}} r_{jq} x_{jmt} ? R_{qt}, \quad ? q, t \tag{16}$$

$$x_{jmt} \ ? \ ?0,1?, \quad ? \, j, m, t \tag{17}$$

where

x_{jmt}	=	1 if activity j is executed in mode m and completes at time t ; 0, otherwise
J	=	terminal node or activity
d_{jm}	=	duration of activity j when executed in mode m
e_j	=	early start time of activity j
l_j	=	late start time of activity j
T	=	late project completion time
r_{jmq}	=	requirement for resource q by activity j executed in mode m
R_{qt}	=	availability of resource q in time t
O_j	=	set of predecessors of activity j

To solve the MRCPSP, Talbot (1982) presents a two-stage solution methodology which builds upon ideas presented earlier for the RCPSP (see Talbot and Patterson, 1978). In the first stage, the network is relabeled using a heuristic scheduling rule. This labeling process defines the order in which activities are considered for scheduling during the second stage of the procedure. The precedence and resource constraints are also stored in memory as compact arrays that are interrogated during enumeration to ensure solution feasibility. Stage 2 is an implicit enumeration algorithm which builds always-feasible partial schedules into complete schedules by considering jobs for assignment in increasing numerical order. When a complete schedule is built, if the schedule is an improved solution, bounds are tightened and the assignment procedure begins again with job 1. Ultimately, optimality is verified either by enumerating (explicitly or implicitly) all possible schedules or by achieving some theoretical bound such as the critical path. Talbot demonstrates the procedure on problems of up to 30 activities. Not all problems were solved in the 16-second time limit permitted (Talbot, 1982: 1209).

Patterson *et al*. (1989, 1990) refined Talbot's solution approach by introducing a *precedence tree* which allows a systematic enumeration of mode assignments and start times. At each level of the tree, the activities which are eligible for scheduling (vis-à-vis the precedence and resource constraints) are considered for addition to the partial schedule. In the case of minimizing the project makespan, activities are scheduled at their earliest precedence and resource feasible time. Patterson *et al*. also discuss the application of the precedence tree to the Resource-Constrained Net

Present Value Problem (RCNPVP) which is the RCPSP where the minimization of cash flows is the objective (see Icmeli and Erenguc, 1996; and Doersch and Patterson, 1977). Negative activity cash flows in the objective function of the RCNPVP would drive the start time of those activities to their late start time. Because of the increased computational times required to enumerate over all possible start times of these activities, Patterson *et al.* suggest use of their algorithm as a heuristic, where they allocate some fixed percent of the solution time to *right-shifting* the activities with negative cash flows.

Sprecher (1994) improves the procedure by Patterson *et al.* for the RCPSP by introducing the notion of an *i-partial* schedule which uniquely describes a node *i* of the enumeration tree and the associated partial schedule. Sprecher also applies four dominance criteria and one feasibility bounding rule. Sprecher performed a computational evaluation of his procedure on a set of test problems and found that his procedure revealed an acceleration factor of approximately one hundred in comparison to the original algorithm of Patterson *et al.* (1990).

Kolisch and Frase (1996) produce an additional acceleration of the procedure by Sprecher (1994) by adding three bounding rules (to shorten the time windows of feasible activity start times), two lower bounding rules, and one feasibility rule. They compare the modified procedure with the basic enumeration scheme using 250 benchmark problems and find improvement on the order of 1000 times. Sprecher and Drexl (1996a, 1996b) provide further refinements to the procedure. They present a branch-and-bound algorithm with special bounding rules which has substantially improved the computational tractability of the MRCPSP and which has nearly doubled the size of projects that can be solved to optimality (Sprecher and Drexl, 1996b: 24). Even so, in a test of 10 randomly generated problems, one problem with 16 activities, 5 modes per activity, and 4 resource types required 3 hours 56 minutes to solve. Of greater concern, Sprecher and Drexl report that the computation time seems to increase exponentially with the number of activities and the number of modes per activity (Sprecher and Drexl, 1996b: 18).

Finally, Sprecher and Drexl (1998) improve the precedence tree approach further by introducing search tree reduction schemes which exclude partial schedules from further continuation. Search tree reduction is provided by a number of bounding rules, which, they report, nearly doubles the tractability of the problem (*i.e.*, the size of problems that can be solved) (Sprecher and Drexl, 1998: 448).

Sprecher *et al*. (1997) present another approach which builds upon the delay alternative concept of Christofides *et al*. (1987) and Demeulemeester and Herroelen (1992) for the single mode case. Using the notion of *mode alternatives*, each level of the branch-and-bound tree is associated with a fixed time (decision point) at which activities can be started. Decision points occur when all activities currently in process finish. The set of eligible activities is based on the set of activities that are finished at or before the decision point. All eligible activities are temporarily scheduled at the decision point. An eligible activity (now scheduled to start at the decision point) was either previously assigned a mode or has not been assigned a mode. The eligible activities not previously assigned a mode are assigned a mode and that set of activities, with their newly assigned modes, form a *mode alternative*. Scheduling all eligible activities to start at the decision point may have caused some resource conflicts. Thus, the set of minimal delay alternatives is computed, where a *delay alternative* is a subset of the activities started at the decision point whose postponement makes the remaining scheduled activities renewable-resource feasible. A *minimal delay alternative* is one where no proper subset of the delay alternative is itself a delay alternative. A minimal delay alternative is selected and those activities making up the alternative are removed from the partial schedule at the decision point. A new decision point is calculated and the process continues until a complete schedule is found. The algorithm, then, backtracks to previously untried delay alternatives and, if there are no more delay alternatives, to untried mode alternatives.

Hartmann and Drexl (1998) present an approach based on the approach used by Stinson *et al*. (1978) for the single-mode case and almost identical to the mode and delay alternative approach of Sprecher *et al*. (1997). The difference between the single- and multi-mode approaches lies in the way partial schedules are expanded. The *mode and extension alternative* approach defines decision points and mode alternatives in the same way that the mode and delay alternative approach does. However, rather than attempting to start all eligible activities at a design point and then delaying some subset of these activities to achieve resource feasibility, the mode and extension alternative approach identifies subsets of resource feasible activities and begins one of these subsets at the decision point. Backtracking tests all untried extension alternatives before testing untried mode alternatives.

Resource-Constrained Project Scheduling Problem with Multiple Crashable Modes. Ahn and Erenguc (1998) combine the MRCPSP and the Time/Cost Tradeoff Problem to form a new problem, the Resource-Constrained Project Scheduling Problem with Multiple Crashable

Modes (RCPSPMCM). In the RCPSPMCM, the duration of each activity is not only a function of resource requirements (mode selection) but also of the amount of crashing (duration reduction by increasing direct costs). For example, mode selection for an activity might be a matter of choosing a skilled worker (charging a fixed hourly rate) or an unskilled worker (charging a different hourly rate). The skilled worker would likely require less time to accomplish the activity. Duration crashing, on the other hand, might be accomplished by paying either worker overtime, thereby shortening the duration of the activity (whichever mode was selected) without changing the mode. Because of the combinatorial nature of the problem and the success of heuristics, Ahn and Erenguc propose a heuristic approach for this problem.

Mode-Identity, Resource-Constrained Project Scheduling Problem. Salewski *et al*. (1996a, 1996b, 1997) introduce the RCPSP with Mode Identity (MIRCPSP). In many situations, such as audit-staff scheduling, time-tabling, and course scheduling, the resources correspond to individuals. This leads to an assignment-type of problem where each activity must be performed by one or more of several individuals. Mode identity refers to the generalization of the RCPSP where the set of all activities must be subdivided into subsets where all activities forming a subset must be performed in the same mode. The RCPSP, then, might be viewed as a mode identity problem where each activity is its own subset.

Expediting Resources. The concept of expediting resources was introduced by Deckro and Hebert (1989) in their Resource Critical Project Crashing Problem (RCPCP) (Problem 6 in Figure 2-1). The RCPCP is identical to the RCPSP except that it allows a project to be *crashed* by increasing critical resources in one or more periods. The objective is to determine the start time of the activities and the critical resources to increase in order to meet the project due date at minimal cost for additional resources. The problem can be extended to allow for penalty and bonus payments based on a target due date.

Deckro and Hebert base their model (shown below) on the PWW formulation of the RCPSP. The constraints limiting resources, RCPSP Constraints (8), are modified to incorporate a new integer variable, h_{qt}, representing the units of expediting resources q used in time period t. The new resource constraints are Constraints (19). New constraints are added to limit the availability of expediting resources, Constraints (20). The objective function minimizes the cost of the expediting resources, c_q. Objective (18) also includes a bonus, b_t, for early completion of the project (by time G) and a penalty, p_t, for late completion of the project (after time G).

Minimize $$\overset{T}{\underset{t?1}{?}} c_q h_{qt} ? \overset{G}{\underset{t?1}{?}} b_t x_{Jt} ? \overset{T}{\underset{t?G?1}{?}} p_t x_{Jt} \tag{18}$$

subject to $$\overset{l_j}{\underset{t?e_j}{?}} x_{jt} ? 1, \quad ? j \tag{11}$$

$$\overset{l_i}{\underset{t?e_i}{?}} tx_{it} ? \overset{l_j}{\underset{t?e_j}{?}} tx_{jt} ? d_i, \quad ? i ? O_j, j \tag{12}$$

$$\overset{J}{\underset{j?1}{?}} r_{jq} x_{jt} ? h_{qt} ? R_{qt}, \quad ? q, t \tag{19}$$

$$h_{qt} ? H_{qt}, \quad ? q, t \tag{20}$$

$$x_{jt} ? ?0,1?, \quad ? j, t \tag{9}$$

$$H_{qt} ? 0 \text{ and integer}, \quad ? j, t \tag{21}$$

where

x_{jt}	=	1 if activity j completes at time t; 0, otherwise
h_{qt}	=	units of resource q used at time t
J	=	terminal node or activity
d_j	=	duration of activity j
e_j	=	early start time of activity j
l_j	=	late start time of activity j
b_j	=	bonus for early completion of project (at time t)
c_q	=	cost of a unit of resource q
p_j	=	penalty for late completion of project (at time t)
G	=	desired project completion time
T	=	late project completion time
r_{jq}	=	requirement for resource q by activity j
R_{qt}	=	availability of resource q in time t

H_{qt} = expediting availability of resource q in time t

O_j = set of predecessors of activity j

Deckro and Hebert provide an example of the RCPCP solved using a commercial integer program solver.

Kolisch and Frase (1996) extend the concept of expediting resources to include not only renewable resources, but also nonrenewable resources. The problem they introduce, the Multi-Modal, Resource-Constrained Project Scheduling Problem with Expediting Resources (MRCPSP-EXP), also considers multiple activity execution modes (Problem 11 in Figure 2-1). They solve the problem using a modification to the implicit enumeration scheme by Sprecher (1994).

Generalized Precedence. Generalized precedence constraints extend the types of temporal relationships between activities beyond the standard finish-start precedence. Generalized precedence can be used to model finish-start, finish-finish, start-start, and start-finish precedence types. De Reyck and Herroelen (1998a, 1999) show that all four types of precedence can, in fact, be represented by the start-start precedence type with minimal time lags. The resulting problem is the Generalized Resource-Constrained Project Scheduling Problem (GRCPSP) (Problem 3 in Figure 2-1). When the GRCPSP is extended for multiple activity execution modes, the resulting problem is the Generalized, Multi-Modal, Resource-Constrained Project Scheduling Problem (GMRCPSP) (Problem 7 in Figure 2-1). The GMRCPSP has been addressed specifically by Demeulemeester and Herroelen (1997). If maximal time lags are then included in the precedence relationships, the resulting problem is the Multi-Modal, Resource-Constrained Project Scheduling Problem with Generalized Precedence (MRCPSP-GPR) (Problem 9 in Figure 2-1). This problem has been addresses by Herroelen *et al*. (1998) and De Reyck and Herroelen (1998a, 1998b, 1999). In their survey of project scheduling, Kolisch and Padman (1998) point out that with the presence of minimal and maximal time lags, a problem becomes much more complicated and standard RCPSP algorithms generally fail to obtain solutions (Kolisch and Padman, 1998: 16). Solution procedures that have been used are typically extensions of other procedures already discussed. For example, De Reyck and Herroelen (1998a) extend a procedure used for the Discrete Time/Cost Problem, and De Reyck and Herroelen (1999) use the concept of delay alternatives.

Multi-Project Scheduling

Pritsker *et al.* (1969) are perhaps the first to explicitly address problems with multiple projects. They mention the applicability of their model to multiple projects and formulate an example with multiple projects, but they do not suggest any multi-project solution methodologies other than lumping the projects together as one larger project. Since then, the following multi-project problems have been addressed with solution methodologies designed to take advantage of the multi-project nature of the problem.

Multi-Project Scheduling Problem. The Multi-Project Program Scheduling Problem (MPSP) is presented by Wiley (1996) and Wiley *et al.* (1998). The objective of the problem is to minimize the cost or duration of a multi-project program by crashing some of its activities. Unlike the Activity Duration Crashing Problem, however, activity crashing is tied to specific limited resources. That is, for every time period an activity is crashed, an amount of each resource is consumed. Since these resources are limited, the amount of crashing possible is limited.

Wiley's formulation is broadly based on the formulation by Deckro *et al.* (1992) for scheduling work packages. Deckro *et al.* solve the work package problem using a standard linear programming code. They also note the decomposability of the problem using algorithms such as Benders' partitioning or Dantzig-Wolfe decomposition. Berczi (1986) also models the scheduling of work packages but uses goal programming to allow for multiple objectives.

Since Wiley's multi-project model displays the familiar block-angular structure and since variables are assumed to be continuous, Wiley decomposes the problem using the Dantzig-Wolfe decomposition approach. Dantzig and Wolfe (1960) developed their decomposition principle to exploit the block-angular structure of many large linear programs by decomposing the problem based on resources. The decomposition is characterized by a subproblem for each distinct block and one master problem. During the algorithm, the subproblems are iteratively solved, and during each iteration, a solution proposal is passed to the master problem. The master problem records and keeps track of all of these proposals. The master problem, then, seeks to identify a convex combination of all the proposals submitted by Subproblem 1, and a convex combination of all the proposals submitted by Subproblem 2, and so on, which, collectively, satisfy the coupling constraints and which is optimal to the original problem.

Advantages to the decomposition approach are various. By decomposing a large problem, it is often possible to solve linear programs of a size which would otherwise be unsolvable. The ability to solve large problems becomes even more attractive when the subproblems, themselves, have a structure which can be exploited. Furthermore, the subproblems are independent and can be solved on separate processors, leading to parallelization.

Another advantage to the decomposition approach is its economic interpretation (see Baumol and Fabian, 1964; Lasdon, 1970: 160-163; Deckro *et al.*, 1998). The decomposition can be viewed as a decentralized decision process. In the context of a firm, the master problem represents the problem of the corporation which seeks to optimize the overall good of the firm. The subproblems can be viewed as subdivisions of the firm whose focus is on their respective subdivision and not on the firm as a whole. Each subdivision has a set of unique constraints to which it must adhere. There is also a set of constraints which couple the subdivisions. These may be constraints on resources for which all subdivisions compete. The solution process begins with each subdivision submitting a proposal to the firm based on a unit profit figure provided by the firm. Unfortunately, a proposal which is good for one subdivision may not be good for another subdivision or, more importantly, to the overall corporation. The firm receives these proposals from the subdivisions and determines the impact each proposal has on the corporation and, ultimately, the other subdivisions. The firm, then, revises its unit profit figures and hands those down to the subdivision. The subdivisions submit new proposals based on the revised figures. The process continues iteratively until an optimal set of decisions can be found. While the economic interpretation of the decomposition method is frequently viewed in terms of the firm, it can likewise be extended to any organization or process which can be decentralized. Clearly, balancing resources amongst the units of a joint command would be a similar process, with the overall commander acting as the *corporation* and the various units acting as *subdivisions*.

The master problem and subproblems of the decomposed MPSP can be easily solved using commercial linear program solvers. An example is provided by Wiley (1996) and Wiley *et al.* (1998) with 3 projects, a total of 39 activities, and 3 resource types (including a budget). Because of the continuity of the variables, insightful sensitivity analysis is also made available.

<u>Resource-Constrained, Multi-Project Scheduling Problem</u>. The Resource-Constrained, Multi-Project Scheduling Problem (RCMPSP) (Problem 5 in Figure 2-1), is presented by Deckro *et al.* (1991). The RCMPSP is identical to the RCPSP except that it considers multiple projects. While these multiple projects can be modeled as a single *super-project* and solved by the approaches

described for single-project problems, Deckro *et al.* (1991) propose a promising approach used to decompose the problem.

Solution of the problem is aided by recognition of the block-angular structure of the problem where the individual projects make up the blocks. This structure has been exploited by Sweeney and Murphy (1979) for the solution of large decomposable integer programs, including the multi-item scheduling problem (Sweeney and Murphy, 1981). Sweeney and Murphy (1979) develop their decomposition principle which is very similar to Dantzig-Wolfe decomposition in that it exploits the block-angular structure of large problems to decompose them into a set of smaller, easier-to-solve problems. The main difference is that the Sweeney-Murphy decomposition algorithm is designed for integer problems, while Dantzig-Wolfe has focused primarily on continuous, linear programs. The subproblems are solved to calculate a set of best solutions for each subproblem. These sets of best solutions are fed to the master problem which attempts to identify one solution from each subproblem which is both feasible and optimal to the original problem. If a combination of solutions cannot be identified, additional solutions are generated by the subproblems and fed to the master problem. This process continues iteratively until an optimal solution is found.

The Sweeney-Murphy decomposition approach has been applied by Deckro *et al.* (1991) to the RCMPSP. The master problem includes the resource constraints imposed on the overall program. The subproblems include project-specific constraints. They provide an illustrative example with eight projects. The original problem, before decomposition, would have 880 0-1 variables and 374 constraints – a prohibitively time consuming problem (Deckro *et al.*, 1991: 114). After decomposition, the largest subproblem had only 160 variables and the largest master problem had only 110 variables. Deckro *et al.* (1991) also point out that the subproblems include project-specific constraints which can be further subdivided into two sets: job completion constraints and activity precedence constraints (Deckro et al., 1991: 114) The nature of both of these sets of subproblem constraints lend themselves to further exploitation.

<u>Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem</u>. The Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem (MRCMPSP) (Problem 8 in Figure 2-1), is a further generalization of the RCPSP. It allows for multiple activity modes as well as multiple projects. Vercellis (1994) presents this problem with an objective function to maximize the Net

Present Value (NPV) of a multi-project program by determining the mode under which to perform each activity.

Vercellis solves this problem using a decomposition approach based on Lagrangian relaxation. Lagrangian relaxation is a method for simplifying, or *relaxing*, the constraint set of a problem. Suppose that the constraint set consists of a set of *complicating* constraints and a set of more tractable constraints, in the sense that, in the absence of the complicating constraints, the problem would be solved relatively easily. It is possible to *relax* the problem by incorporating the complicating constraints into the objective function using appropriate *multipliers*. If the multipliers are fixed, the relaxed problem can then be solved for the original problem variables. The solution approach then hinges on finding appropriate values for the multipliers.

Geoffrion (1974) applies the relaxation approach to integer programming problems where it can be used to fathom solutions in branch-and-bound procedures and to derive cutting planes. Chalmet and Gelders (1976) use the approach for solving a warehousing model formulated in 0-1 variables. Fisher (1981) discusses a number of important issues revolving around the use of relaxation for integer programming problems including the selection of multipliers, the choice of competing relaxations, and the incorporation of the lower and upper bounding capabilities of the Lagrangian problem into branch-and-bounding procedures.

Vercellis (1994) uses Lagrangian relaxation where he takes project precedence constraints and resource-partitioning constraints (these are the only two sets of constraints that tie projects to each other) and moves them to the objective function with Lagrangian multipliers. The approach then decomposes this Lagrangian relaxation into subproblems, one for each project, which are easier to solve than the original integer program. The subproblems are solved using the branch-and-bound algorithm presented in Speranza and Vercellis (1993). The approach was tested on a number of problems with up to 10 projects, up to 20 activities per project, 2 or 3 modes per activity, and as many as 6 renewable resources. Problem solution times were all on the order of minutes (Vercellis, 1994: 274).

<u>Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem</u>. Van Hove (1998) presents the Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem (GMRCMPSP) (Problem 10 in Figure 2-1). This is a RCPSP with multiple modes, start-start precedence relationships with minimal lags, and multiple projects. Van Hove decomposes the problem using Sweeney-Murphy decomposition and then solves the subproblems

using a modification of the enumeration scheme by Sprecher (1994). Van Hove solves a problem with 4 projects, 25 activities per project, 2 or 4 modes per activity, and 4 resources per project. The projects are coupled together by a constraint on the use of nonrenewable resources.

Summary

A wide variety of problem types and solution methodologies provide a foundation upon which to formulate and solve the MRCMPSP-GPR/EXP (Problem 12 in Figure 2-1). A formulation of the MRCMPSP-GPR/EXP, based on the PWW problem, is presented in the next chapter. Subsequent chapters present greater detail on the applicability of the above approaches to the MRCMPSP-GPR/EXP.

III. Methodology

Introduction

This chapter presents a mathematical formulation of the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP). It also discusses the complexity of the MRCMPSP-GPR/EXP and describes an approach for decomposing the problem. The decomposition of the problem serves as the foundation for the solution approach developed in this dissertation. While the specific methodology for solving the decomposed problem is detailed in subsequent chapters, this methodology is outlined in this chapter to provide an overview of the research that follows.

Mathematical Formulation

The MRCMPSP-GPR/EXP consists of a multi-project program where precedence relationships (both standard finish-start and generalized) may exist between activities within a single project or between activities of different projects. Figure 3-1 shows the activity-on-node network representation of an example problem with three projects. Within the network, nodes represent activities while precedence relationships are represented by the directed arcs between the nodes. Single-headed arcs denote standard precedence while double-headed arcs denote generalized precedence. Only one generalized precedence is shown in the example problem – between activities B6 and C4.

Each activity has one or more alternative execution modes which determine the duration and resource requirements of the activity. By selecting alternative execution modes for an activity, it may be possible to either *crash* or *extend* the duration of the activity. In practice, crashing an activity generally comes at the cost of greater resource utilization while extending an activity may release resources for use elsewhere.

Resources may be renewable, nonrenewable, or doubly constrained. Renewable resources are those which are reusable from period to period (such as manpower, machinery, and space) but are limited on a per-period basis. Nonrenewable resources are those which are expended once used (such as fuel and construction materials) and are limited at the project or program level. Doubly-constrained resources are those whose availability is limited at the project or program level, as well as on a per-period basis. Budget is a good example of a resource which may be doubly constrained. While the total program budget may be $1 million dollars, spending may be capped at $10,000 per period.

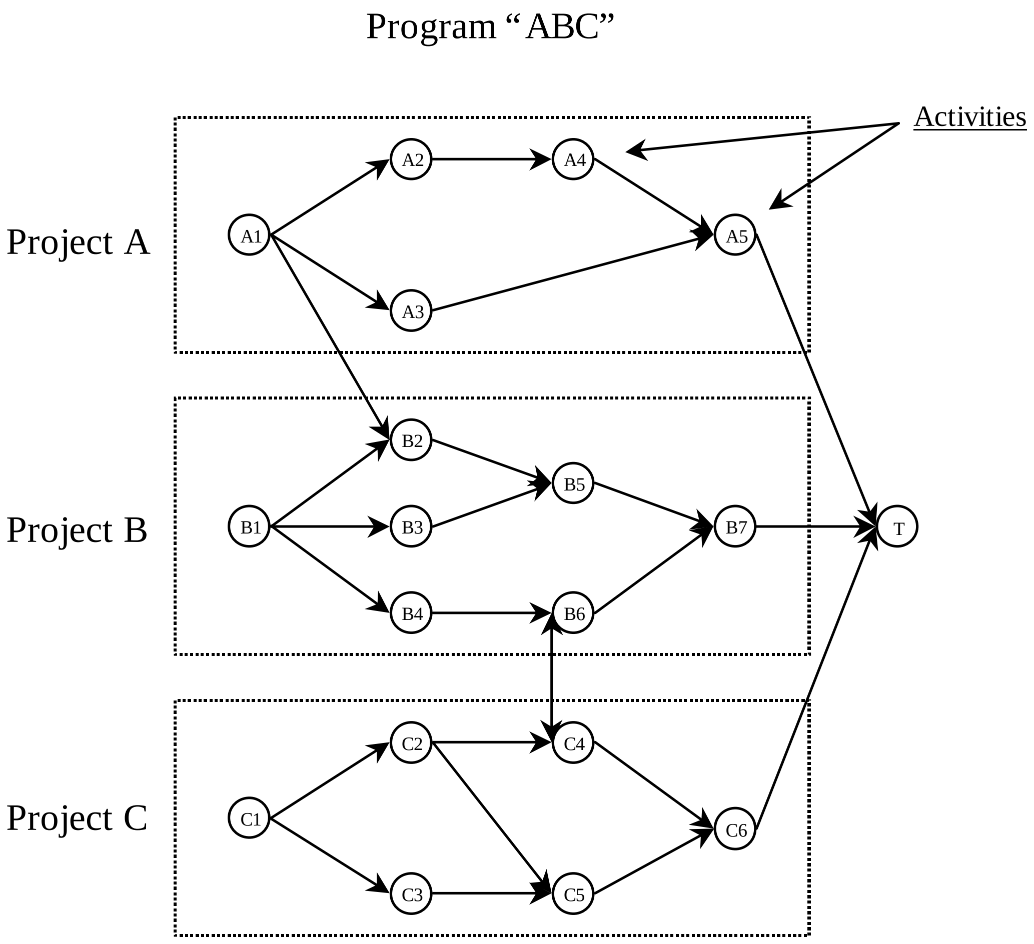


Figure 3-1. Activity-on-Node Representation of Example Problem 1

The development of a feasible program schedule is constrained by limitations on resources, as well as the program planning horizon. Because of the tradeoff between the duration and resource utilization of each activity, prudent selection of activity execution modes becomes crucial. In practice, it is generally necessary to crash some activities and extend others. For example, consider the three-activity project depicted in Figure 3-2 with activity data in Table 3-1. (Note that activity T is a dummy activity with zero duration and resource utilization.) If the shortest-duration modes are selected for each activity, then Activity 1 will require four units of some renewable resource and Activity 2 will require five units of the same resource. If sufficient resources were available over the entire project planning horizon (*i.e.*, at least nine units per period), Activities 1 and 2 could start simultaneously and project completion would occur at the end of Period 4. If, however, only eight units of resource are available in each time period, then the activities would have to be performed in series and project completion would occur at the end of Period 7. Though the shortest-duration mode was selected for each activity, there is no guarantee that this selection yields the earliest project completion time. Furthermore, this selection of modes need not even yield a feasible schedule. If, for instance, the project must be completed no later than Time Period 6, this schedule is not feasible.

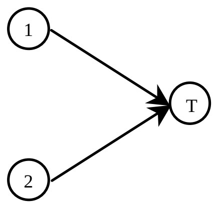


Figure 3-2. Example Problem 2

Table 3-1. Example Problem 2

Activity	Mode	Duration	Resource Utilization
1	1	3	4
	2	5	2
2	1	4	5
	2	7	2
T	1	0	0

In Example Problem 2, it is also possible to select the longest-duration modes for each activity. The activities can be started simultaneously since their combined resource utilization of five units does not exceed the per-period availability of eight units. The earliest project completion time, though, is still at the end of Period 7 (*i.e.*, the maximum duration of the two activities). Again, this selection of modes is infeasible for a planning horizon of six time periods.

The only feasible schedule for the project exists when the longest-duration mode is selected for Activity 1 and the shortest-duration mode is selected for Activity 2. The activities can be started simultaneously since they require a total of only seven units of resource and the project can be completed by the end of Period 5 which is within the project's planning horizon of six periods.

While the crashing and extending of activities (through mode selection) is vital in the program scheduling process, additional scheduling options may be made possible through the availability of additional resources. These additional, or *expediting*, resources may be obtained at some fixed price, subject to availability. For example, during a military airlift operation the availability of transport aircraft is limited on a per-period basis. It is well established that the careful assignment of specific types of transport aircraft to specific routes is essential to a successful operation (this assignment of aircraft to routes constitutes mode selection). It may be possible, however, to purchase, reassign, or lease additional aircraft to supplement the aircraft which are *regularly* available. This is, after all, the basis for the Civilian Reserve Aircraft Fleet (CRAF) program. Just as the availability of *regular* aircraft is limited so is the availability of the expediting aircraft. Furthermore, though the acquisition costs of the regular aircraft may be viewed as a sunk cost (at least in regards to the specific operation), the acquisition cost for the expediting aircraft are explicitly considered since they are incurred specifically for the given operation.

Expediting resources are so named because they give greater flexibility to the selection of activity modes and start times. Consequently, the set of feasible schedules becomes larger and it may be possible to find a feasible schedule with an earlier completion time. In Example Problem 2, if at least one unit of expediting resource is available in each time period, then selecting the shortest-duration mode for each activity will lead to a feasible schedule which can be completed by the end of Period 4 (*i.e.*, project completion has been expedited). The only question that remains is whether or not the benefit of finishing the project one time period earlier than otherwise possible outweighs the cost of the additional resource. The answer to this question depends, of course, on the cost of the expediting resource and the benefit gained by expediting project completion.

The scheduling objective may take a variety of forms. For example, the project's duration may be minimized subject to a budget restriction on expediting resources. The cost of expediting resources may, on the other hand, be minimized subject to completing the project by a fixed *due date*. Even more general objectives may include bonuses and penalties for early or late completion of the project relative to the due date or they may include costs based on the activity modes selected. Regardless of the program objective considered, any solution to the MRCMPSP-GPR/EXP will include the start time and execution mode of each activity, as well as the types and number of expediting resources to acquire.

An important note to consider is that, while more scheduling options (*i.e.*, activity execution modes and expediting resources) provide a larger set of feasible schedules from which to choose and give a planner greater flexibility, the problem also becomes larger and it becomes more difficult to find an optimal schedule.

Assumptions. The mathematical formulation of the MRCMPSP-GPR/EXP begins with the following assumptions:

1. A program consists of a fixed set of interrelated projects. The interrelationships between activities in one project and activities in another project are fixed and known.

2. A project consists of a fixed set of interrelated activities. The interrelationships between activities within a project are fixed and known.

3. An activity is performed in one of multiple alternative execution modes. Each mode has a fixed duration and per period requirement for renewable, nonrenewable, and doubly-constrained resources. The demand for a given resource remains constant from period to period.

4. Activities are not allowed to be split; once an activity begins, it will continue until complete.

5. The program has a fixed and known planning horizon. The program may also have a due date. Each individual project has an early start time no earlier than time period zero and a fixed due date no later than the program due date.

6. Activity durations and resource utilizations, as well as resource availabilities, are integer valued.

Notation. This section presents the notation used in the mathematical formulation of the MRCMPSP-GPR/EXP. The notation is explained further when introduced in the discussion that follows.

General Sets:

P	=	the number of projects
I_p	=	the set of activities in project p
M_{pi}	=	the set of execution modes for activity i of project p

Activity Interrelationship Sets:

O_0	=	the set of program-level (inter-project) standard precedence relations
O_p	=	the set of standard precedence relations within project *p*
N_0	=	the set of program-level (inter-project) generalized precedence relationships
N_p	=	the set of generalized precedence relationships in project *p*

Resource Sets:

Q^R	=	the set of all renewable resources
Q_0^R	=	the set of program-level renewable resources
Q_p^R	=	the set of renewable resources unique to project *p*
Q^N	=	the set of all nonrenewable resources
Q_0^N	=	the set of program-level nonrenewable resources
Q_p^N	=	the set of nonrenewable resources unique to project *p*

Note that doubly-constrained resources are not explicitly considered but they belong to the set of renewable resources and to the set of nonrenewable resources. Doubly-constrained resources will be identified solely by their membership in the other two sets of resources.

Special Indices:

$p(i)$	=	activity *i* of project *p*
T	=	the dummy terminal activity of the program
T_p	=	the dummy terminal activity of project *p*

Time-Related Parameters:

F	=	the early program completion time
G	=	the program completion due date
D	=	the program planning horizon ($F \leq G \leq D$)

E_p	=	the early start time of project p
F_p	=	the early completion time of project p
G_p	=	the completion due date of project p
D_p	=	the planning horizon of project p ($F_p \leq G_p \leq D_p$)

e_{pi}	=	the early start time of activity $p(i)$
l_{pi}	=	the late start time of activity $p(i)$
w_{pi}	=	$[e_{pi}, l_{pi}]$, the start time window of activity $p(i)$

d_{pim}	=	the duration of activity $p(i)$ in mode m

Resource-Related Parameters:

r^R_{pimq}	=	the units of renewable resource q required by activity $p(i)$ in mode m
R^R_{qt}	=	the units of renewable resource q available at time t
H^R_{qt}	=	the units of expediting, renewable resource q available at time t

r^N_{pimq}	=	the units of nonrenewable resource q required by activity $p(i)$ in mode m
R^N_q	=	units of nonrenewable resource q available
H^N_q	=	the units of expediting, nonrenewable resource q available

Cost Parameters:

c_{pimt}	=	cost of beginning activity $p(i)$, executed in mode m, at time t
c^R_{qt}	=	cost of an expediting unit of renewable resource q at time t
c^N_q	=	cost of an expediting unit of nonrenewable resource q
b_t	=	benefit for completing the program at time t (early completion)
a_t	=	cost for completing the program at time t (late completion)

Binary Variables:

x_{pimt} = 1, if activity $p(i)$ is executed in mode m and starts at time t

= 0, otherwise

$x_{T_p t}$ = 1, if terminal activity T_p of project p starts at time t

= 0, otherwise

x_{Tt} = 1 if program terminal activity T starts at the beginning of period t

= 0, otherwise

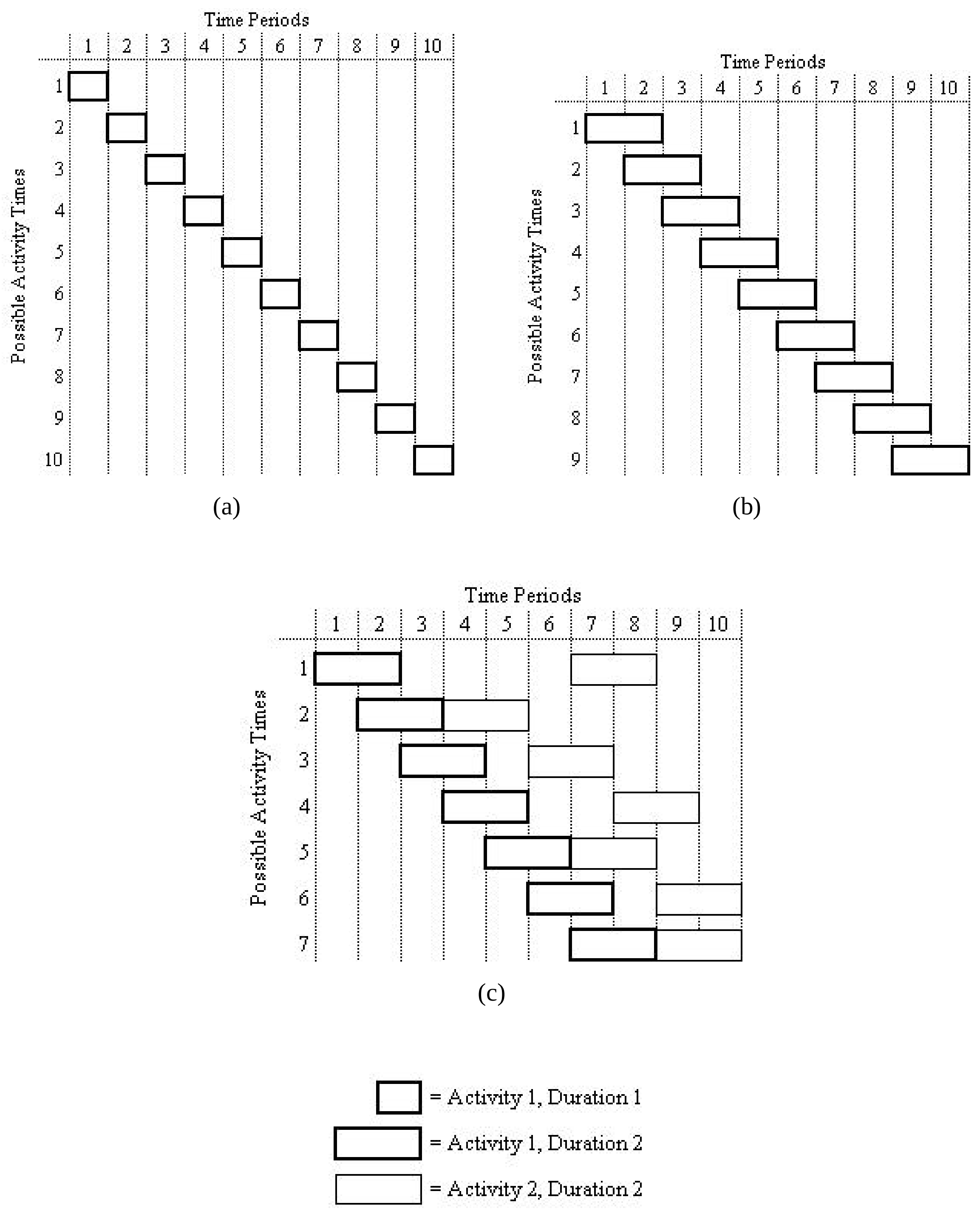
Expediting Resource Variables:

h_{qt}^{R} = the units of expediting, renewable resource q used at time t

h_{q}^{N} = the units of expediting, nonrenewable resource q obtained

Numbering of Activity Modes. The execution modes of each activity in the program are numbered in order of increasing duration. In mathematical terms, let M_{pi} be the set of execution modes for activity $p(i)$, let $\mathrm{M} = \left\| M_{pi} \right\|$ be the number of modes of activity $p(i)$, and let d_{pim} be the duration of mode m of activity $p(i)$. Then for each activity, its execution modes will be numbered such that d_{pi1} ? d_{pi2} ? ? d_{piM} .

Activity Start Time Windows. One of the advantages of the Pritsker, Watters, Wolfe (1969) model of the Resource-Constrained Project Scheduling Problem (RCPSP) over the Bowman (1959) model is its variable definition. For any given activity i, Bowman defines a 0-1 variable for every time period in which activity i could be in progress. Pritsker, Watters, and Wolfe, by contrast, define a 0-1 variable for only those time periods in which activity i can finish. This alternate variable definition serves to reduce the number of variables in the model. Because of its variable-reduction property, the Pritsker, Watters, Wolfe variable definition is used in this study, with one modification – instead of reflecting activity finish times, the variables in the subject model reflect activity start times. This modification has no technical impact on the formulation and is used simply out of this author's preference.



Note: In Part (c), Activity 2 is "randomly" placed.

Figure 3-3. Example Activity Start Time

With the modified Pritsker, Watters, Wolfe variable definition, each activity has a set of 0-1 variables which reflect the possible start times for the activity. It is possible to define a start time variable for every period in the program planning horizon, but this typically results in more variables than necessary. For example, assume a program has one activity with unit duration and that the program planning horizon is ten time periods (see Figure 3-3). The single activity would then have ten associated start time variables since it could start in any of the ten time periods in the program planning horizon (Figure 3-3a). On the other hand, if the duration of the activity were two, then the activity will have only nine associated variables since it cannot possibly start in time period ten and finish by the end of the planning horizon (Figure 3-3b).

Now consider the addition of a second activity of duration two (Figure 3-3c). If the original activity must precede the second, then the number of variables associated with the original activity reduces to seven. If the original activity begins in time eight or after, it cannot finish in time for the second activity to be complete by the end of the program planning horizon. Following this inductive reasoning, it is clear that the more constrained an activity becomes in terms of duration and activity interrelationships, the fewer variables are required to reflect all of the possible start times of the activity. This is the power of the Pritsker, Watters, and Wolfe variable definition.

Since there is considerable benefit in reducing the number of variables as much as possible (to improve computational efficiency), it is useful to determine the minimal interval of possible start times for each activity. This minimal interval is referred to as the activity's *start time window*. To determine the start time window for all of the activities in the program, the Generalized Critical Path Method (GCPM) is used.

The GCPM is a generalization of the Critical Path Method (CPM). While the CPM (see Schtub, Bard, and Globerson, 1994: 339) finds the early and late start times of activities subject to standard precedence relationships, the GCPM extends this approach to generalized precedence relationships. The GCPM algorithm is as follows:

<u>Generalized Critical Path Method (GCPM)</u>

1. Set the early start time of each activity equal to the release date of the project of which it is a member.

2. For each activity *i*, in numerical order, change its early start time to the greatest of the following:

   a. its current early start time,
   b. the early start time plus duration of each of its standard predecessors,
   c. the early start time of activity *j* plus minimum time lag between activity *j* and activity *i,* for each activity *j* which is a generalized predecessor of activity *i*.

3. If the early start time of any activity changed at Step 2, repeat Step 2.

4. For each activity *i,* in numerical order, check each activity for which activity *i* is a generalized predecessor. If the early start time of any generalized *successor* of activity *i* is greater than the early start time of activity *i* plus the maximum time lag, change the early start time of activity *i* to the greatest of the early start time minus maximum time lag of each generalized successor of activity *i.*

5. If the early start time of any activity changed at Step 4, repeat Step 2. If not, the early start time of each activity has been found.

6. Set the late start time of each activity equal to the program horizon minus its duration.

7. For each activity *i,* in reverse numerical order, change its late start time to the least of the following:

   a. its current late start time,
   b. the late start time of each of its standard successors minus the duration of activity *i,*
   c. the late start time of each activity generalized successor of activity *i* minus its minimum time lag from activity *i.*

8. If the late start time of any activity changed at Step 7, repeat Step 7.

9. For each activity *i,* in reverse numerical order, check each activity which is a generalized predecessor of activity *i*. If the late start time of activity *i* is greater than the late start time of any generalized predecessor plus its maximum lag time, change the late start time of activity *i* to the least of the late start times plus maximum time lag of each generalized predecessor of activity *i*.

10. If the late start time of any activity changed at Step 9, repeat Step 7. If not, the late start time of each activity has been found.

With the GCPM defined, activity start time windows are determined using the following four-step procedure:

<u>Determining Start Time Windows</u>

Step 1. Relax the problem to an unconstrained network problem, eliminating all of the resource considerations of the original problems. Consider only the activity precedence relationships and project early start times, E_p.

Step 2. Using Mode 1 of each activity (*i.e.*, the modes of shortest duration), use the GCPM to determine the early start time, $e_{pi,}$ of each activity (this includes the early completion time, F, of the program).

Step 3. Using Mode 1 of each activity and letting the program planning horizon, D, be the expected completion dates of the program and projects, use the GCPM again to determine the late start time, l_{pi}, of each activity.

Step 4. The start time window of activity $p(i)$ is the closed interval $[e_{pi}, l_{pi}]$.

Note that the use of the shortest-duration mode of each activity is important at Step 2. To see why, assume all activities are set to Mode 1 except for activity $p(i)$ (a non-terminal activity) which is set to Mode m^*. (We assume that the terminal activity is a dummy activity of zero duration.) Assume that the duration of Mode m^* is k units longer than that of Mode 1. When the GCPM is used to determine the early start times of the activities, either activity $p(i)$ is on a critical path or it is not. If it is, then there is at least one activity, call it activity $q(j)$ (q may equal p), which is on the same critical path and which immediately follows activity $p(i)$. If e_{pi} is the early start time of activity $p(i)$, then the early start time of activity $q(j)$ is $e_{qj} = e_{pi} + d_{pim^*} = e_{pi} + d_{pi1} + k$. If activity $q(j)$ has no other predecessors which are critical, then e_{qj} is at least one time period later than if activity $p(i)$ were executed in Mode 1 and, consequently, e_{qj} is not the earliest possible start time of activity $q(j)$. If activity $q(j)$ has other predecessors which are also critical activities or if activity $p(i)$ is not on a critical path, then the above argument is not valid. However, since we do not know *a priori* the relation of activities $p(i)$ and $q(j)$ to the critical paths, it is necessary to use Mode 1, the shortest-duration mode, for each activity to prevent such problems.

The argument for using Mode 1 for every activity in Step 3 is similar to the argument for using them in Step 2. If activity $p(i)$ is scheduled so it finishes as late as possible, but it is executed in a mode with a longer duration than that of Mode 1, the resultant start time will be earlier than if activity $p(i)$ were executed in Mode 1. Consequently, the calculated late start time of activity $p(i)$ will be too early.

Constraints. As constraints are developed below, they are consecutively numbered. The number assigned to a constraint will remain unchanged throughout the discussion. The constraints are:

Execution Mode and Activity Start Time

An activity can be executed in only one mode and is started only once. Define variable x_{pimt} to be unity if activity $p(i)$ is executed in mode m and starts at the beginning of time period t; x_{pimt} equals zero otherwise. The constraint

$$\underset{m ? M_{pi}}{?} \; \underset{t ? w_{pi}}{?} x_{pimt} \; ? \; 1, \qquad ? \, i \, ? \, I_p, ? \, p \, ? \, P \tag{1}$$

assures a unique execution mode and start time for each activity. Note that I_p is the set of activities in project p, where P is the set of projects in the program, M_{pi} is the set of modes for activity $p(i)$, and the time index t is summed over the start time window w_{pi} of activity $p(i)$.

Activity Precedence

Activity precedence may occur at the program level (*i.e.*, inter-project) or at the project level (*i.e.*, intra-project). Precedence constraints may indicate that one activity precedes another (standard precedence) or that the start times of two activities are related (generalized precedence). Standard precedence is common to scheduling networks. Generalized precedence is less common but is included for its applicability to many problems of interest, including Joint Campaign Planning (where the start times of two or more operations may need to be coordinated) and program management (where concurrent engineering approaches are being used).

Recall that the duration of activity $p(i)$ is dependent on its execution mode. Let d_{pim} be a known parameter which denotes the duration of activity $p(i)$ when executed in mode m. In addition, recall that variable x_{pimt} is unity for exactly one mode/start time combination. The duration of activity $p(i)$ is, then, represented by the term,

$$\underset{m ? M_{pi}}{?} d_{pim} \underset{t ? w_{pi}}{?} x_{pimt} \; .$$

For example, if activity $p(i)$ is executed in mode m^* and begins at time t^*, then $x_{pim^*t^*} = 1$ and the duration of activity $p(i)$ is d_{pim^*}.

The start time of activity i is given by the term:

$$\underset{m ? M_i}{?} \; \underset{t ? w_i}{?} t x_{pimt} \; .$$

Given the above expressions for the duration and start time of activity $p(i)$, standard precedence constraints can be defined. If activity $p(i)$ directly precedes activity $p(j)$, then the start time of activity *p*(*j*) must be no earlier than the start time of activity *p*(*i*) plus the duration of activity *p*(*i*). Thus,

$$\sum_{m\in M_{pj}}\sum_{t\in w_{pj}} tx_{pjmt} \ge \sum_{m\in M_{pi}}\sum_{t\in w_{pi}} tx_{pimt} + \sum_{m\in M_{pi}} d_{pim}\sum_{t\in w_{pi}} x_{pimt}, \quad \forall (i,j)\in O_p, \forall p\in P$$

or

$$\sum_{m\in M_{pi}}\sum_{t\in w_{pi}} (t+d_{pim})x_{pimt} - \sum_{m\in M_{pj}}\sum_{t\in w_{pj}} tx_{pjmt} \le 0, \qquad \forall (i,j)\in O_p, \forall p\in P \tag{2}$$

where O_p is the set of standard precedence relations in project *p* and $(i,j)\in O_p$ denotes that activity *p*(*i*) precedes activity *p*(*j*).

At the program level, assume activity *i* of project *p* directly precedes activity *j* of project $\bar{p}$. The start time of activity $\bar{p}(j)$ must, then, be no earlier than the start time of activity *p*(*i*) plus the duration of activity *p*(*i*). Thus,

$$\sum_{m\in M_{\bar{p}j}}\sum_{t\in w_{\bar{p}j}} tx_{\bar{p}jmt} \ge \sum_{m\in M_{pi}}\sum_{t\in w_{pi}} tx_{pimt} + \sum_{m\in M_{pi}} d_{pim}\sum_{t\in w_{pi}} x_{pimt}, \qquad \forall (pi,\bar{p}j)\in O_0$$

or

$$\sum_{m\in M_{pi}}\sum_{t\in w_{pi}} (t+d_{pim})x_{pimt} - \sum_{m\in M_{\bar{p}j}}\sum_{t\in w_{\bar{p}j}} tx_{\bar{p}jmt} \le 0, \qquad \forall (pi,\bar{p}j)\in O_0 \tag{3}$$

where O_0 is the set of program-level standard precedence relations and $\forall (pi,\bar{p}j)\in O_0$ denotes that activity *p*(*i*) precedes activity $\bar{p}(j)$.

Generalized precedence constraints at the project level are given by:

$$\sum_{m\in M_{pj}}\sum_{t\in w_{pj}} tx_{pjmt} - \sum_{m\in M_{pi}}\sum_{t\in w_{pi}} tx_{pimt} \ge \delta_{ij}^{\min}, \qquad \forall (i,j)\in N_p, \forall p\in P$$

$$\sum_{m\in M_{pj}}\sum_{t\in w_{pj}} tx_{pjmt} - \sum_{m\in M_{pi}}\sum_{t\in w_{pi}} tx_{pimt} \le \delta_{ij}^{\max}, \qquad \forall (i,j)\in N_p, \forall p\in P$$

or

$$\underset{m? M_{pi}}{?}\ \underset{t? w_{pi}}{?}\ tx_{pimt} \ ? \ ?_{ij}^{\min} \ ? \ \underset{m? M_{pj}}{?}\ \underset{t? w_{pj}}{?}\ tx_{pjmt} \ ? \ 0\,, \qquad ? \ (i, j) ? \ N_p, ? \, p ? \ P \tag{4}$$

$$? \ \underset{m? M_{pi}}{?}\ \underset{t? w_{pi}}{?}\ tx_{pimt} \ ? \ ?_{ij}^{\max} \ ? \ \underset{m? M_{pj}}{?}\ \underset{t? w_{pj}}{?}\ tx_{pjmt} \ ? \ 0\,, \qquad ? \ (i, j) ? \ N_p, ? \, p ? \ P \tag{5}$$

where N_p is the set of generalized precedence relationships in project p, $(i, j) ? \ N_p$ denotes that activity $p(i)$ is a generalized predecessor of activity $p(j)$, and $?_{ij}^{\min}$ and $?_{ij}^{\max}$ are the minimal and maximal start time lags between activities i and j.

At the program level, generalized precedence constraints are given by:

$$\underset{m? M_{pi}}{?}\ \underset{t? w_{pi}}{?}\ tx_{pimt} \ ? \ ?_{ij}^{\min} \ ? \ \underset{m? M_{\bar{p}j}}{?}\ \underset{t? w_{\bar{p}j}}{?}\ tx_{\bar{p}jmt} \ ? \ 0\,, \qquad ? \ (pi, \bar{p}j) ? \ N_0 \tag{6}$$

$$? \ \underset{m? M_{pi}}{?}\ \underset{t? w_{pi}}{?}\ tx_{pimt} \ ? \ ?_{ij}^{\max} \ ? \ \underset{m? M_{\bar{p}j}}{?}\ \underset{t? w_{\bar{p}j}}{?}\ tx_{\bar{p}jmt} \ ? \ 0\,, \qquad ? \ (pi, \bar{p}j) ? \ N_0 \tag{7}$$

where N_0 is the set of program-level generalized precedence relationships and $? \ (pi, \bar{p}j) ? \ N_0$ denotes that activity $p(i)$ is a generalized predecessor of activity $\bar{p}$ (j).

<u>Program/Project Completion</u>

The program has a fixed planning horizon by which the program must be completed and individual projects may have individual planning horizons as well. (If a project does not have a distinct horizon, its horizon is assumed to be the same as the program's.) Planning horizons must be chosen such that the projects / program can be feasibly completed within the planning horizons. The convention used in the Program Attributes Generator with Expediting Resources (PAGER), which is introduced in the next chapter, is to calculate the planning horizon simply by adding the duration of the longest-duration mode of each activity. The program horizon represents the minimum amount of time required to complete the program if *regularly-available* resources are constrained to a point where only one activity can be scheduled at a time and in its longest-duration mode.

The planning horizons are used in determining activity start time windows. Though planning horizons are considered when the start time windows are calculated, additional constraints are required to enforce the planning horizons. Otherwise, it is possible for an activity to start within its start time window and end beyond a project or program planning horizon. For instance, suppose project p has dummy terminal activity T_p and that T_p has a single predecessor, activity $p(i)^*$. Since activity T_p has zero duration, the late start time of activity $p(i)^*$ is calculated as the horizon of project p less the duration of activity $p(i)^*$ executed in its shortest mode. If activity $p(i)^*$ begins at its late start time but, however, in a mode of longer duration, its completion time will be beyond the project planning horizon. It becomes necessary, then, to include completion time constraints.

To simplify the completion time constraints, dummy terminal activity, T, is added for the program and dummy terminal activity, T_p, is added for each project p. Terminal activities have zero duration. Then,

$$\underset{t ? w_T}{?} t x_{Tt} ? D \tag{8}$$

$$\underset{t ? w_{T_p}}{?} t x_{T_p t} ? D_p, \qquad ? p ? P \tag{9}$$

<u>Resources</u>

There are three types of resources: renewable resources, nonrenewable resources, and doubly-constrained resources. Doubly-constrained resources are handled by extending the sets of renewable and nonrenewable resources and are not modeled explicitly. Thus, if resource q^* is a doubly-constrained resource, renewable resource constraints are added to represent the per-period restriction on resource q^* (one constraint for each time period) and a single nonrenewable resource constraint is added to represent the overall restriction on resource q^*.

Activity $p(i)$ has a per-period requirement of r^R_{pimq} units of renewable resource q if executed in mode m and there are R^R_{qt} units of renewable resource q available in time period t. Expediting units of renewable resource q are available in time period t at a per unit cost of c^R_{qt} and up to the limit of H^R_{qt} additional units. Activity $p(i)$ also has a requirement of r^N_{pimq} units of nonrenewable resource q if executed in mode m. There are R^N_q units of nonrenewable resource q available for the entire program or project. Expediting units of nonrenewable resource q are available at a per unit cost of c^N_q and up to the limit of H^N_q additional units.

Resources may be specific to a given project (project-level) or may be in demand by more than one project (program-level). Let Q^R_0 be the set of program-level renewable resources and let Q^R_p

be the set of renewable resources unique to project *p*. Similarly, let Q_0^N be the set of program-level nonrenewable resources and let Q_p^N be the set of nonrenewable resources unique to project *p*.

If variable h_{qt}^R is the number of expediting units of renewable resource *q* used in time period *t* and variable h_q^N is the number of expediting units of nonrenewable resource *q* used, then for project-level renewable resources:

$$\underset{i ? I_p}{?}\ \underset{m ? M_{pi}}{?}\ r^R_{pimq} \overset{t^*}{\underset{t ? t^* ? d_{pim} ? 1}{?}} x_{pimt} ? R^R_{qt} ? h^R_{qt}, \qquad ? q ? Q^R_p, t^* ? [E_p, D_p], p ? P$$

or

$$\underset{i ? I_p}{?}\ \underset{m ? M_{pi}}{?}\ r^R_{pimq} \overset{t^*}{\underset{t ? t^* ? d_{pim} ? 1}{?}} x_{pimt} ? h^R_{qt} ? R^R_{qt}, \qquad ? q ? Q^R_p, t^* ? [E_p, D_p], p ? P \tag{10}$$

and

$$h^R_{qt} ? H^R_{qt}, \qquad ? q ? Q^R_p, t ? [E_p, D_p], p ? P \tag{11}$$

For program-level renewable resources in demand by more than one project:

$$\underset{p ? P}{?}\ \underset{i ? I_p}{?}\ \underset{m ? M_{pi}}{?}\ r^R_{pimq} \overset{t^*}{\underset{t ? t^* ? d_{pim} ? 1}{?}} x_{pimt} ? h^R_{qt} ? R^R_{qt}, \qquad ? q ? Q^R_0, t^* ? [0, D] \tag{12}$$

and

$$h^R_{qt} ? H^R_{qt}, \qquad ? q ? Q^R_0, t ? [0, D] \tag{13}$$

For project-level nonrenewable resources:

$$\underset{i ? I_p}{?}\ \underset{m ? M_{pi}}{?}\ r^N_{pimq} \underset{t ? w_{pi}}{?} x_{pimt} ? h^N_q ? R^N_q, \qquad ? q ? Q^N_p, p ? P \tag{14}$$

and

$$h^N_q ? H^N_q, \qquad ? q ? Q^N_p, p ? P \tag{15}$$

For program-level nonrenewable resources in demand by more than one project:

$$\underset{p ? P}{?}\ \underset{i ? I_p}{?}\ \underset{m ? M_{pi}}{?}\ r^N_{pimq} \underset{t ? w_{pi}}{?} x_{pimt} ? h^N_q ? R^N_q, \qquad ? q ? Q^N_0 \tag{16}$$

and

$$h_q^N \; ? \; H_q^N, \qquad ? \, q \, ? \, Q_0^N \tag{17}$$

Objective Function. A wide variety of objective functions may be used for the MRCMPSP-GPR/EXP. The objective may be stated generally as the minimization of program costs subject to a completion due date. The generalityof the objective function is dependent on what program costs are included. Three objective functions are specifically addressed here, each successive function being a generalization of the previous. The algorithm developed for the MRCMPSP-GPR/EXP solves for the most general of the objective functions and, consequently, for the more specific.

1. Minimize Program Duration

One common objective is to minimize program duration. In this case, the objective function is:

$$\text{Minimize} \qquad z \; ? \mathop{?}\limits_{t?F}^{D} t x_{Tt} \tag{18}$$

where F is the earliest possible completion time of the program and D is the program due date.

This objective function could be accompanied by a budget constraint to restrict program cost, such as the cost of expediting resources.

2. Minimize Program Expediting and Completion Costs

A more general objective is to minimize program costs, including the cost of expediting resources and the penalty for late program completion. In this case, the objective function becomes:

$$\text{Minimize} \qquad z \; ? \mathop{?}\limits_{q?Q^R} \mathop{?}\limits_{t?0}^{D} c_{qt}^R h_{qt}^R \; ? \mathop{?}\limits_{q?Q^N} c_q^N h_q^N \; ? \mathop{?}\limits_{t?F}^{D} a_t x_{Tt} \tag{19}$$

where c_{qt}^R and c_q^N are the costs for expediting resources and a_t is the cost for completion of the program at time t. Note that program completion *costs* could be bonuses for early completion (completion by some target completion date or due date, G) and/or penalties for late completion (vis-à-vis G). Since this is a minimization function, a bonus (a negative cost) would be negative-valued. If both bonuses and penalties were considered, the final term in the objective function

$$\mathop{?}\limits_{t?F}^{D} a_t x_{Tt}$$

would be divided between two terms, one for bonuses on the interval [F, G] and the other for penalties on the interval [G+1, D] as in

$$\sum_{t=F}^{D} a_t x_{Tt} = \sum_{t=F}^{G} a_t^{\text{Bonus}} x_{Tt} + \sum_{t=G+1}^{D} a_t^{\text{Penalty}} x_{Tt} .$$

Objective Function (19) is a generalization of Objective Function (18), to minimize program duration. If the costs of expediting resources are moved to the constraint set as part of a budget constraint, or if these costs are considered negligible, then costs c_{qt}^R and c_q^N become zero. In addition, if the completion costs, a_t, are set to t, then Objective Function (19) is precisely the objective of minimizing program duration.

3. Minimize Program Mode/Time, Expediting, and Completion Costs

A further generalization is to minimize program costs, including costs assessed for executing an activity in a given mode and starting at a given time, the cost of expediting resources, and the completion costs. In this case, the objective function becomes:

Minimize

$$z = \sum_{p \in P} \sum_{i \in I_p} \sum_{m \in M_{pi}} \sum_{t=0}^{D} c_{pimt} x_{pimt} + \sum_{q \in Q^R} \sum_{t=0}^{D} c_{qt}^R h_{qt}^R + \sum_{q \in Q^N} c_q^N h_q^N + \sum_{t=F}^{G} a_t^{\text{Bonus}} x_{Tt} + \sum_{t=G+1}^{D} a_t^{\text{Penalty}} x_{Tt} \qquad (20)$$

where c_{pimt} is the cost for executing activity $p(i)$ in mode m and starting at time t.

If costs c_{pimt} are set to zero, Objective Function (20) reduces to Objective Function (19).

Complete Model. The complete formulation of the MRCMPSP-GPR/EXP, with Objective Function (20), is given by:

Minimize

$$z = \sum_{p \in P} \sum_{i \in I_p} \sum_{m \in M_{pi}} \sum_{t=0}^{D} c_{pimt} x_{pimt} + \sum_{q \in Q^R} \sum_{t=0}^{D} c_{qt}^R h_{qt}^R + \sum_{q \in Q^N} c_q^N h_q^N + \sum_{t=F}^{G} a_t^{\text{Bonus}} x_{Tt} + \sum_{t=G+1}^{D} a_t^{\text{Penalty}} x_{Tt} \qquad (20)$$

Subject To

$$\sum_{m \in M_{pi}} \sum_{t \in w_{pi}} x_{pimt} = 1, \qquad \forall i \in I_p, \forall p \in P \qquad (1)$$

$$\sum_{m \in M_{pi}} \sum_{t \in w_{pi}} (t + d_{pim}) x_{pimt} - \sum_{m \in M_{pj}} \sum_{t \in w_{pj}} t x_{pjmt} \le 0, \qquad \forall (i,j) \in O_p, \forall p \in P \qquad (2)$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?}(t ? d_{pim})x_{pimt} ? \underset{m?M_{\overline{p}j}}{?}\ \underset{t?w_{\overline{p}j}}{?} tx_{\overline{p}jmt} ? 0, \qquad ?(pi,\overline{p}j)? O_0 \tag{3}$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\min} ? \underset{m?M_{pj}}{?}\ \underset{t?w_{pj}}{?} tx_{pjmt} ? 0, \qquad ?(i,j)? N_p, ?p? P \tag{4}$$

$$? \underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\max} ? \underset{m?M_{pj}}{?}\ \underset{t?w_{pj}}{?} tx_{pjmt} ? 0, \qquad ?(i,j)? N_p, ?p? P \tag{5}$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\min} ? \underset{m?M_{\overline{p}j}}{?}\ \underset{t?w_{\overline{p}j}}{?} tx_{\overline{p}jmt} ? 0, \qquad ?(pi,\overline{p}j)? N_0 \tag{6}$$

$$? \underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\max} ? \underset{m?M_{\overline{p}j}}{?}\ \underset{t?w_{\overline{p}j}}{?} tx_{\overline{p}jmt} ? 0, \qquad ?(pi,\overline{p}j)? N_0 \tag{7}$$

$$\underset{t?w_T}{?} tx_{Tt} ? D \tag{8}$$

$$\underset{t?w_{T_p}}{?} tx_{T_pt} ? D_p, \qquad ?p ? P \tag{9}$$

$$\underset{i?I_p}{?}\ \underset{m?M_{pi}}{?} r_{pimq}^R \overset{t^*}{\underset{t?t^*?d_{pim}?1}{?}} x_{pimt} ? h_{qt}^R ? R_{qt}^R, \qquad ?q? Q_p^R, t^*? [E_p, D_p], p? P \tag{10}$$

$$h_{qt}^R ? H_{qt}^R, \qquad ?q? Q_p^R, t? [E_p, D_p], p? P \tag{11}$$

$$\underset{p?P}{?}\ \underset{i?I_p}{?}\ \underset{m?M_{pi}}{?} r_{pimq}^R \overset{t^*}{\underset{t?t^*?d_{pim}?1}{?}} x_{pimt} ? h_{qt}^R ? R_{qt}^R, \qquad ?q? Q_0^R, t^*? [0, D] \tag{12}$$

$$h_{qt}^R ? H_{qt}^R, \qquad ?q? Q_0^R, t? [0, D] \tag{13}$$

$$\underset{i?I_p}{?}\ \underset{m?M_{pi}}{?} r_{pimq}^N \underset{t?w_{pi}}{?} x_{pimt} ? h_q^N ? R_q^N, \qquad ?q? Q_p^N, p? P \tag{14}$$

$$h_q^N ? H_q^N, \qquad ?q? Q_p^N, p? P \tag{15}$$

$$\underset{p ? P}{?} \underset{i ? I_p}{?} \underset{m ? M_{pi}}{?} r^N_{pimq} \underset{t ? w_{pi}}{?} x_{pimt} ? h^N_q ? R^N_q, \qquad ? q ? Q^N_0 \tag{16}$$

$$h^N_q ? H^N_q, \qquad ? q ? Q^N_0 \tag{17}$$

$$x_{pimt} ? ?0,1?, \qquad ? p,i,m,t \tag{21}$$

$$h^R_{qt} ? 0 \text{ and integer,} \qquad ? q,t \tag{22}$$

$$h^N_q ? 0 \text{ and integer,} \qquad ? q \tag{23}$$

Problem Size and Complexity

The mathematical formulation of the MRCMPSP-GPR/EXP is, as expected, large and complicated. The number of variables and constraints for even a relatively small problem can be daunting. Returning to the example illustrated in Figure 3-1, consider the activity data listed in Table 3-2. The example problem has 3 projects, a total of 19 activities (including the dummy program terminal activity), and up to three modes per activity. In addition, there are two renewable resources, two nonrenewable resources, and one doubly-constrained resource, all considered to be program-level resources. A program planning horizon of 30 time periods has been assumed.

The number of binary variables, x_{pimt}, may be as many as

$$\overset{\|P\|}{\underset{p?1}{?}} \overset{\|I_p\|}{\underset{i?1}{?}} \overset{\|M_{pi}\|}{\underset{m?1}{?}} \overset{l_{pi}}{\underset{t?e_{pi}}{?}} 1 ? \overset{\|P\|}{\underset{p?1}{?}} \overset{l_{T_p}}{\underset{t?e_{T_p}}{?}} 1 ? \overset{l_T}{\underset{t?e_T}{?}} 1,$$

where the first term represents the non-dummy activities, the second represents the project dummy terminal activities, and the last term represents the program dummy terminal activity. This number can be approximated as $\|P\| ?\bar{I}_p ?\overline{M} ?\overline{W}$, where $\bar{I}_p$ is the average number of activities per project (including dummy terminal activities), $\overline{M}$ is the average number of modes per activity, and $\overline{W}$ is the average length of the activity start time windows. In the example problem, there are on the order of 350 binary variables, assuming 10 to be the average length of the activity start time windows. (Clearly, the *tightness* of the start time windows will affect the problem size.)

The number of renewable resource variables, h^R_{qt}, is

$$\left\|Q^R\right\| ?D,$$

where $\left\|Q^R\right\|$ is the total number of renewable and doubly-constrained resources (project-level and program-level) and D is the program planning horizon. In the example problem, there are 90 renewable resource variables.

Table 3-2. Activity Data for Example Problem

Act	Mode	Dur	R1	R2	N1	N2	D1
A1	1	2	0	4	4	4	9
	2	5	10	0	3	4	6
A2	1	4	0	3	6	8	7
A3	1	2	3	0	9	1	9
	2	9	0	6	9	1	5
A4	1	4	0	9	9	8	4
	2	6	0	9	7	7	3
	3	9	6	0	6	7	3
A5	1	3	0	10	10	2	5
B1	1	2	3	0	9	7	5
B2	1	6	7	0	5	6	9
	2	8	0	5	5	2	8
B3	1	8	8	0	2	10	4
B4	1	10	1	0	6	8	8
B5	1	4	0	2	10	5	7
	2	8	0	2	10	5	5
	3	10	6	0	10	5	2
B6	1	5	0	8	5	3	4
	2	6	0	6	5	2	3
B7	1	2	8	0	3	5	6
	2	4	6	0	3	5	4
	3	10	4	0	3	5	1
C1	1	1	0	7	8	7	4
	2	6	4	0	6	5	3
	3	10	0	7	4	5	2
C2	1	10	4	0	2	4	5
C3	1	2	3	0	5	8	9
C4	1	5	0	5	4	10	5
	2	8	0	5	3	7	4
	3	10	6	0	2	7	4
C5	1	9	0	3	8	1	3
C6	1	3	8	0	8	10	6
	2	6	7	0	6	9	5
	3	10	7	0	3	7	4
T	1	0	0	0	0	0	0

The number of nonrenewable resource variables, h_q^N, is

$$\left\|Q^N\right\|.$$

That is, one variable for each nonrenewable resource (project- and program-level) and one variable for each doubly-constrained resource (project- and program-level). In the example problem, there are three nonrenewable resource variables.

In all, the example problem has 443 variables. Table 3-3 summarizes these results.

Table 3-3. Number of Variables

Variable Type	**Number of Variables**	**Variables in Example Problem**
x_{pimt}	$\underset{p?1}{\overset{\lVert P\rVert}{?}}\ \underset{i?1}{\overset{\lVert I_p\rVert}{?}}\ \underset{m?1}{\overset{\lVert M_{pi}\rVert}{?}}\ \underset{t?e_{pi}}{\overset{l_{pi}}{?}}\ 1 ? \underset{p?1}{\overset{\lVert P\rVert}{?}}\ \underset{t?e_{T_p}}{\overset{l_{T_p}}{?}}\ 1 ? \underset{t?e_T}{\overset{l_T}{?}}\ 1$	350
h_{qt}^R	$\lVert Q^R\rVert ?D$	90
h_q^N	$\lVert Q^N\rVert$	3
Total		443

Next, consider the number of constraints. Table 3-4 outlines the number of each type of constraint, where $\overline{Q}_p^R$ is the average number of project-level renewable resources, $\overline{Q}_p^N$ is the average number of project-level nonrenewable resources, $\overline{O}_p$ and $\overline{N}_p$ are the average number of standard and generalized precedence relationships, respectively, per project, and $\lVert O_0\rVert$ and $\lVert N_0\rVert$ are the number of program-level standard and generalized precedence relationships, respectively. There are a total of 232 constraints in the example problem.

Finally, consider the complexity of the MRCMPSP-GPR/EXP. This problem is a generalization of the Resource-Constrained, Multi-Modal, Project Scheduling Problem (RCMMPSP) which is known to be NP-Hard (Kolisch, 1995: 26). In addition, Sprecher and Drexl state that if a RCMMPSP has more than one activity, just finding a *feasible* solution is an NP-hard problem (Sprecher and Drexl, 1996a: 3).

In an effort to mitigate the computational difficulty of the MRCMPSP-GPR/EXP, this dissertation develops a methodology for decomposing the original problem into smaller, more tractable problems. One of the research issues addressed is the tradeoff between the time saved in solving a number of smaller subproblems and the computation overhead associated with the iterative process of the approach. Each of the smaller problems is still NP-Hard, but it is precisely this fact which makes decomposition an appealing approach. Since the time to solve NP-Hard problems grows very quickly as the size of the problem grows, reducing the size of a problem should significantly reduce the time to solve it. If the time saved solving each of the smaller problems is greater than the computational overhead, then the decomposition approach should be applied when possible. Chapter VI addresses this research issue and provides results of testing of the decomposition approach.

Table 3-4. Number of Constraints

Constraint Type	**Number of Constraints**	**Constraints in Example Problem**
(1)	$\|P\| ?\bar{I}_p$	19
(2) and (3)	$\overline{O}_p ?\|P\| ? \|O_o\|$	24
(4) to (7)	$2\overline{N}_p ?\|P\| ? 2\|N_o\|$	2
(8)	1	1
(9)	$\leq \|P\|$	0
(10)	$\|P\| ?\overline{Q}_p^R ?\overline{D}_p$	0
(11)	$\|P\| ?\overline{Q}_p^R ?\overline{D}_p$	0
(12)	$\|Q_0^R\| ?D$	90
(13)	$\|Q_0^R\| ?D$	90
(14)	$\|P\| ?\overline{Q}_p^N$	0
(15)	$\|P\| ?\overline{Q}_p^N$	0
(16)	$\|Q_0^N\|$	3
(17)	$\|Q_0^N\|$	3
Total		232

Decomposition of the MRCMPSP-GPR/EXP

The first step in decomposing the MRCMPSP-GPR/EXP is partitioning the constraints between those which apply to individual projects and those which apply to the entire program. The program-level constraints are (3), (6), (7), (8), (12), (13), (16), and (17), while the project-level constraints are (1), (2), (4), (5), (9), (10), (11), (14), and (15). The problem with partitioned constraints becomes:

Minimize

$$z ? \underset{p?P}{?} \underset{i?I_p}{?} \underset{m?M_{pi}}{?} \overset{D}{\underset{t?0}{?}} c_{pimt} x_{pimt} ? \underset{q?Q^R}{?} \overset{D}{\underset{t?0}{?}} c_{qt}^R h_{qt}^R ? \underset{q?Q^N}{?} c_q^N h_q^N ? \overset{G}{\underset{t?F}{?}} a_t^{\text{Bonus}} x_{Tt} ? \overset{D}{\underset{t?G?1}{?}} a_t^{\text{Penalty}} x_{Tt} \quad (20)$$

Subject To

Program-Level Constraints

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} (t ? d_{pim}) x_{pimt} ? \underset{m?M_{\bar{p}j}}{?}\ \underset{t?w_{\bar{p}j}}{?} tx_{\bar{p}jmt} ? 0, \qquad ? (pi, \bar{p}j) ? O_0 \tag{3}$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\min} ? \underset{m?M_{\bar{p}j}}{?}\ \underset{t?w_{\bar{p}j}}{?} tx_{\bar{p}jmt} ? 0, \qquad ? (pi, \bar{p}j) ? N_0 \tag{6}$$

$$? \underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?} tx_{pimt} ? ?_{ij}^{\max} ? \underset{m?M_{\bar{p}j}}{?}\ \underset{t?w_{\bar{p}j}}{?} tx_{\bar{p}jmt} ? 0, \qquad ? (pi, \bar{p}j) ? N_0 \tag{7}$$

$$\underset{t?w_T}{?} tx_{Tt} ? D \tag{8}$$

$$\underset{p?P}{?}\ \underset{i?I_p}{?}\ \underset{m?M_{pi}}{?} r_{pimq}^{R} \overset{t^*}{\underset{t?t^*?d_{pim}?1}{?}} x_{pimt} ? h_{qt}^{R} ? R_{qt}^{R}, \qquad ? q ? Q_0^R, t^* ? [0, D] \tag{12}$$

$$h_{qt}^{R} ? H_{qt}^{R}, \qquad ? q ? Q_0^R, t ? [0, D] \tag{13}$$

$$\underset{p?P}{?}\ \underset{i?I_p}{?}\ \underset{m?M_{pi}}{?} r_{pimq}^{N} \underset{t?w_{pi}}{?} x_{pimt} ? h_q^N ? R_q^N, \qquad ? q ? Q_0^N \tag{16}$$

$$h_q^N ? H_q^N, \qquad ? q ? Q_0^N \tag{17}$$

Project-Level Constraints, p = 1, 2, 3,…, P

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?}\ x_{pimt} \ ?\ 1, \qquad ?\, i\, ?\, I_p, ?\, p\, ?\, P \qquad (1)$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?}\ (t\ ?\ d_{pim})x_{pimt}\ ?\ \underset{m?M_{pj}}{?}\ \underset{t?w_{pj}}{?}\ tx_{pjmt}\ ?\ 0, \qquad ?\,(i,j)\,?\,O_p, ?\,p\,?\,P \qquad (2)$$

$$\underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?}\ tx_{pimt}\ ?\ ?_{ij}^{\min}\ ?\ \underset{m?M_{pj}}{?}\ \underset{t?w_{pj}}{?}\ tx_{pjmt}\ ?\ 0, \qquad ?\,(i,j)\,?\,N_p, ?\,p\,?\,P \qquad (4)$$

$$?\ \underset{m?M_{pi}}{?}\ \underset{t?w_{pi}}{?}\ tx_{pimt}\ ?\ ?_{ij}^{\max}\ ?\ \underset{m?M_{pj}}{?}\ \underset{t?w_{pj}}{?}\ tx_{pjmt}\ ?\ 0, \qquad ?\,(i,j)\,?\,N_p, ?\,p\,?\,P \qquad (5)$$

$$\underset{t?w_{T_p}}{?}\ tx_{T_pt}\ ?\ D_p, \qquad ?\,p\,?\,P \qquad (9)$$

$$\underset{i?I_p}{?}\ \underset{m?M_{pi}}{?}\ r_{pimq}^{R}\ \overset{t^*}{\underset{t?t^*?d_{pim}?1}{?}}\ x_{pimt}\ ?\ h_{qt}^{R}\ ?\ R_{qt}^{R}, \qquad ?\,q\,?\,Q_p^R, t^*?\,[E_p, D_p], p\,?\,P \qquad (10)$$

$$h_{qt}^{R}\ ?\ H_{qt}^{R}, \qquad ?\,q\,?\,Q_p^R, t\,?\,[E_p, D_p], p\,?\,P \qquad (11)$$

$$\underset{i?I_p}{?}\ \underset{m?M_{pi}}{?}\ r_{pimq}^{N}\ \underset{t?w_{pi}}{?}\ x_{pimt}\ ?\ h_{q}^{N}\ ?\ R_{q}^{N}, \qquad ?\,q\,?\,Q_p^N, p\,?\,P \qquad (14)$$

$$h_{q}^{N}\ ?\ H_{q}^{N}, \qquad ?\,q\,?\,Q_p^N, p\,?\,P \qquad (15)$$

Variable Bounds

$$x_{pimt}\ ?\ ?0,1?, \qquad ?\,p,i,m,t \qquad (21)$$

$$h_{qt}^{R}\ ?\ 0 \text{ and integer}, \qquad ?\,q,t \qquad (22)$$

$$h_{q}^{N}\ ?\ 0 \text{ and integer}, \qquad ?\,q \qquad (23)$$

With this partitioning of constraints, the block-angular structure generally associated with Dantzig-Wolfe decomposition (Dantzig and Wolfe, 1960) becomes apparent. Figure 3-4 depicts this block-angular structure, where matrices $\mathbf{A}_p$, p = 1, 2, ..., P+1, consist of the program-level coefficients associated with project p, matrices $\mathbf{B}_p$, p = 1, 2, ..., P+1, consist of the project-level coefficients associated with project p, variables $\mathbf{x}_p$, p = 1, 2, ..., P+1, are the activity start time and expediting resource variables, and constants $\mathbf{c}_p$, p = 1, 2, ..., P+1, are the objective function coefficients associated with variable $\mathbf{x}_p$. Note that $\mathbf{x}_p$ is a *mixed* vector of variables where some of its elements are {0,1} and others are non-negative integers.

<u>Original Problem</u>

Problem (P): Minimize $$z ? \overset{P?1}{\underset{p?1}{?}} \mathbf{c}_p \mathbf{x}_p \quad (24)$$

Subject To

$$\mathbf{A}_1\mathbf{x}_1 ? \mathbf{A}_2\mathbf{x}_2 \; ?\ldots ? \mathbf{A}_P\mathbf{x}_P ? \mathbf{A}_{P?1}\mathbf{x}_{P?1} ? \mathbf{b}_0 \quad (25)$$

$$\mathbf{B}_1\mathbf{x}_1 \qquad ? \mathbf{b}_1 \quad (26)$$

$$\mathbf{B}_2\mathbf{x}_2 \qquad ? \mathbf{b}_2$$

$$\mathbf{B}_P\mathbf{x}_P \qquad ? \mathbf{b}_P$$

$$\mathbf{B}_{P?1}\mathbf{x}_{P?1} \quad ? \mathbf{b}_{P?1}$$

$$\mathbf{x}_p \;?\; \begin{matrix} ??0,1? \text{ for start time variables} \\ ?? \; 0 \text{ and integer for expediting resource variables} \end{matrix} \;, 1 ? \; p ? P ? 1 \quad (27)$$

where

$\mathbf{A}_p$ represents the program-level constraint coefficients associated with project p,

$\mathbf{B}_p$ represents the project-level constraint coefficients of project p,

p, $1 ? \; p ? \; P$, are indices representing the P projects / subproblems, and

P + 1 is the index representing the program.

Figure 3-4. Block-Angular Structure

Dantzig-Wolfe decomposition has proven valuable for linear programs with continuous variables. Dantzig-Wolfe decomposition, however, is inappropriate for integer programming problems, such as the MRCMPSP-GPR/EXP, because the master problem develops a solution by finding an optimal affine combination of candidate solutions from the subproblems (extreme points of the feasible region). In general, an affine combination of the candidate solutions from integer subproblems is not guaranteed to be integer-valued.

Sweeney and Murphy (1979), however, developed a decomposition approach for integer programs with such a block-angular structure. With Sweeney-Murphy decomposition, the individual projects can be separated into blocks, or subproblems. Each subproblem is solved to find a set of *k*-best solutions (in terms of objective function value) for that subproblem. These sets of best subproblem solutions are iteratively passed up to the master problem until the master problem identifies one solution from each subproblem (rather than an affine combination) which, collectively, are feasible to the original problem and provide the best overall solution to the original problem.

Solution Methodologies

The solution methodologies presented in this dissertation are motivated by the Sweeney-Murphy decomposition approach. Methodologies are needed for solving both the decomposed subproblems and the master problem. Additionally, a methodology for generating instances of the MRCMPSP-GPR/EXP is required for testing the solution methodologies. This section presents an overview of these methodologies.

<u>Problem Generation</u>. During a review of the literature, it was determined that no existing problem generator is capable of generating all of the characteristics of the MRCMPSP-GPR/EXP. Worse, most of the existing generators use the coefficient of network complexity (CNC) as their measure of network complexity. The CNC, the ratio of arcs to nodes in the network, is easily implemented in a problem generator, but has considerable shortcomings (detailed in Chapter IV). Thesen (1977) developed an alternate measure of network complexity, the Thesen Restrictiveness (RT), which is recognized as a much better measure. Still, only two generators (Schwindt, 1995, 1996 and Drexl *et al*., 1997)) attempt to use the RT. Unfortunately, both generators actually use the CNC to add arcs to a project network and then simply calculate the RT of the resulting

network. If the desired RT is met or exceeded, the generators stop. Unfortunately, the resulting RT may be well beyond what the user intended.

The research presented in Chapter IV demonstrates a methodology for generating project networks using the RT directly. In that way, researchers have control over the complexity of the networks which underlie their experiments. With an RT-based project network as its core, the generator developed in Chapter IV adds the additional characteristics required by the MRCMPSP-GPR/EXP. Some of the methods presented by Kolisch *et al*. (1992, 1995) are used directly or extended as necessary. Other features are added, such as an approach for converting standard precedence relationships (those produced by the network generator) into generalized precedence relationships.

Single Project / Subproblem Solution. The MRCMPSP-GPR/EXP can be solved directly as a large, single-project problem or decomposed into a set of smaller, semi-independent subproblems (themselves, single-project problems). Whichever approach is used, a methodology for solving single-project instances of the MRCMPSP-GPR/EXP is required.

Chapter V develops an implicit enumeration algorithm for solving single-project instances of the MRCMPSP-GPR/EXP. The algorithm is based on a scheme by Talbot (1982) for the Multi-Modal, Resource-Constrained Project Scheduling Problem (RCPSP). The scheme constructs project schedules by adding one activity to the schedule at a time. First, a mode is selected for the activity and, then, a start time. Feasibility tests are conducted at each step to fathom mode / start time combinations which are infeasible. Bounding tests are also conducted to see if the growing schedule is dominated by the best incumbent schedule. If the current mode / start time combination is either infeasible or dominated, that branch of the search tree is *fathomed* and a new mode / start time combination tried.

While Talbots' scheme provides a solid method for enumerating over the possible project schedules, it is extended for generalized precedence and expediting resources. It is also expanded to generate a set of *k*-best solutions rather than a single optimal. Generation of a set of solutions is explicitly required by the decomposition approach. Even if problem decomposition is not the goal, however, these alternate solutions offer a decision-maker multiple options which can be evaluated using non-objective function criteria (*e.g.*, managerial preference for certain mode choices).

Generation of the *k*-best solutions is made possible by fathoming branches of the search tree using the current *k*-th best solution rather than the current optimal solution. If a solution is at least as good as the *k*-th best, the solution is added to the set and the *k*-th best solution is dropped.

<u>Decomposition / Master Problem Solution</u>. During the solution methodology, each subproblem is solved to find the *k*-best solutions for that subproblem. These sets of *k*-best solutions are passed to a master problem (detailed in Chapter VI) which evaluates them, seeking to find one candidate solution from each subproblem which, collectively, are both feasible and optimal to the original problem. If these criteria are not met, additional solutions are generated by the subproblems and passed to the master problem. This iterative process continues until a feasible and optimal solution is found.

An algorithm for solving the master problem is developed in Chapter VI. The algorithm is an implicit enumeration algorithm, similar to that used for solving the subproblems, except that a subproblem solution is added at each level of the search tree rather than a single activity. Again, feasibility and bounding tests are conducted in an attempt to fathom unproductive branches of the tree as early as possible.

Also detailed in Chapter VI is a methodology for producing multipliers which form part of the subproblem objective functions. These multipliers are developed to encourage the subproblems to comply with program-level constraints. Theoretically, a good choice of multipliers will reduce the number of solutions required from the subproblems. This assertion is tested in Chapter VI.

Summary

This chapter presented a mathematical formulation of the MRCMPSP-GPR/EXP, discussed the size and complexity of the MRCMPSP-GPR/EXP, and showed how the MRCMPSP-GPR/EXP might be decomposed. Additionally, three methodologies were overviewed: (1) problem generation; (2) single project / subproblems solution; and, (3) decomposition / master problem solution. The next three chapters discuss in much greater detail these methodologies.

IV. Problem Generation

Overview

This dissertation addresses the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP). This problem and the solution methodologies presented in later chapters are part of a growing wealth of problem formulations and solution methodologies which comprise the field of resource-constrained project scheduling. Ferreira *et al*. (1998) point out that the need to validate, scope, and score an ever-increasing number of competing algorithms and heuristics for scheduling projects under resource constraints implies the extensive use of simulation to test them against large and significant network testing sets. In view of this need, three widely used test sets have been proposed over the years: those of Patterson (1984); of Alvarez-Valdez and Tamarit (1989); and of Kolisch *et al*. (1995) and Kolisch and Sprecher (1996).

While standard test sets have proven their worth for many researchers, two conditions mitigate their value over time:

1. They are inadequate for new problem types with characteristics not represented in the test set.

2. Researchers have little or no control over the parameters used to develop the fixed test sets. These parameters (such as the complexity of the network) can greatly influence the difficulty of problem instances.

To expand on this point, consider Patterson's (1984) test set of 110 problem instances to compare four exact procedures for makespan minimization of the single-mode resource-constrained project scheduling problem (RCPSP). Though this test set served as *the* benchmark for years, Kolisch *et al*. (1992) suggest that these test problems were not generated using a controlled design of specified parameters, consider only the single-mode and makespan-minimization cases, and have been shown to be among the easier instances of such problems, even among single-mode problems. For these reasons, Kolisch *et al*. argue that these problems should no longer be considered benchmark instances.

To overcome the shortcomings of standard test sets, the development of methods to generate project networks is a key condition in the scientific assessment of so many different procedures on so many different problem types. Unfortunately, the number of published generators available in the literature for resource-constrained project scheduling is limited (Ferreira *et al*., 1998:58). A review of the literature confirms this conclusion. Five recent publicly-accessible generators (Demeulemeester *et al*., 1993; Kolisch *et al*., 1992, 1995; Schwindt, 1995, 1996; Drexl *et al*., 1997; Agrawal *et al*., 1996) are reviewed below for this study. Their key features are summarized in Table 4-7 at the end of this chapter.

Demeulemeester *et al*. (1993) stress the random generation of project networks. They argue that many project scheduling procedures work well for certain network structures and poorly for others. To properly evaluate competing scheduling procedures, networks should be generated from among all feasible networks and not be limited to networks of some particular structure. In the Demeulemeester *et al*. generator, referred to here as *DDH*, the number of nodes and arcs in the network can be specified by the user or randomly drawn from a number of probability distributions. Arcs are then added or deleted until the desired number of arcs is achieved. Activity durations, the number of renewable resources (limited to three), resource requirements and availabilities, and the events marking the project milestones are also randomly drawn from precoded distributions.

While the philosophy used in the development of the DDH generator is a valid general approach to generator design, it ignores the issue that a key aspect of a class of algorithmic designs may be the exploitation of problem structure. To evaluate an algorithm designed specifically for a target class of problems, a researcher requires a way to control the structure of the test problems. This sentiment is echoed by Kolisch *et al*. (1992, 1995) who have developed a problem generator which allows the user to set several project parameters. Some of these parameters were proposed in the literature, while others were developed by Kolisch *et al*. Entitled *ProGen*, their project generator includes single- and multi-mode activities, different categories of resources, and single- and multi-project scheduling problems. ProGen has been used by a number of researchers in recent studies, including De Reyck and Herroelen (1996), Icmeli and Erenguc (1996), Ahn and Erenguc (1998), and Van Hove (1998). In some cases, however, these researchers have added problem features of their own. Van Hove (1998), for instance, replaced the standard precedence constraints generated by ProGen with generalized precedence constraints.

Working with ProGen as a base, two additional problem generators have been developed. Schwindt (1996) extended ProGen (called *ProGen/max*) to incorporate minimal and maximal time lags, as well as some additional problem parameters, such as the estimator of network restrictiveness suggested by Thesen (1977). Drexl *et al*. (1997) extended ProGen to ProGen/? *x* in order to incorporate new modeling extensions such as partially renewable resources, changeover times, and mode and set of mode identity

One final problem generator, DAGEN, was developed by Agrawal *et al*. (1996) to employ the network complexity index introduced by Bein *et al*. (1992). The complexity index measures how far a given network is from a series-parallel network. (The definition and significance of such a network is described below in the subsection entitled *Network Complexity*.) Costs, duration, and resource requirements associated with the activities are generated randomly from uniform distributions.

While problem instances for a great number of problem classes can be generated using the generators described above, none are capable of producing all of the characteristics of the MRCMPSP-GPR/EXP. None of the generators are designed to generate (1) expediting resource availabilities and costs, (2) mode costs which can be constant, increasing, or decreasing with time, or (3) truly multi-project programs. While ProGen suggests the capability of generating multi-project problems, these problems have no precedence relationships between activities of one project and those of another (other than supersource and supersink dummy nodes which tie them together). Furthermore, the resources generated by ProGen are program-level resources only, with no consideration for project-specific resources.

This research develops a problem generator entitled PAGER (**P**rogram **A**ttributes **G**enerator with **E**xpediting **R**esources). It was designed to incorporate all of the characteristics of the MRCMPSP-GPR/EXP. As a generalization of many other problem types, PAGER's usefulness is not limited to this sole class of problem. Problem instances for the traditional PERT/CPM problem, the net present value problem, the job shop scheduling problem, the single- and multi-modal, resource-constrained project scheduling problems, and others can be generated using PAGER. The primary advantages of PAGER are fourfold:

1. It can generate not only single-project problem instances, but truly multi-project problem instances, with interrelationships between projects, program-level (shared) resources, and project-specific resources.

2. Parameters within a multi-project program can be set to independently structure each project, as well as their interrelationships. This allows each project to have unique characteristics. In ProGen, for instance, the number of activities in each project is drawn from the same distribution, whereas in PAGER, the user may specify a distinct distribution for each project.

3. It allows certain parameters (fixed by the user in other generators) to be drawn from user-defined uniform distributions. For example, in ProGen, resource strength (a measure of resource scarceness) is, by construction, the same for each resource. In realistic problems, it is reasonable to expect some resources to be scarcer than others. PAGER allows the value of resource strength to differ (randomly) for each resource.

4. Some researchers may prefer certain problem-defining measures to others. This is particularly true of measures of network complexity (*e.g.*, Thesen, 1977; Kolisch *et al.*, 1992, 1995; De Reyck and Herroelen, 1996). PAGER allows users to select from among two common measures of network complexity found in the literature, even allowing the user to use measures of network complexity simultaneously. Generated problem instances, then, reflect the user's preferences. This flexibility also allows researchers to use a single generator to produce problem sets to compare these competing measures.

The remaining sections of this chapter proceed as follows:

1. *PAGER: Problem Generator* develops the procedure for generating problems step by step.

2. *PAGER Implementation* outlines the implementation of the PAGER program.

3. *Summary and Conclusions* gives some final remarks.

PAGER: Problem Generator

The generation of problems using PAGER can be subdivided into six steps. First, PAGER reads a problem specification file, which contains all of the problem parameters. For each desired problem instance, basic problem data is then generated, a problem network is developed, resource demands and availabilities are determined, and problem cost data is generated. Finally, the problem is output into any of three formats: ProGen format (depending on problem features), PAGER format, or MPS format. Sample input and output files are found in Appendices B and C, respectively. The six steps of problem generation are discussed in the following subsections. Figure 4-1 depicts the overall flow of the problem generation algorithm.

<ins>Step 1 - Specification File Input</ins>. The specification file (illustrated in Appendix B) is a simple text file through which the user may specify problem design preferences. These preferences are the parameters used to generate basic problem data, the problem's network structure, the resource data, and the cost data.

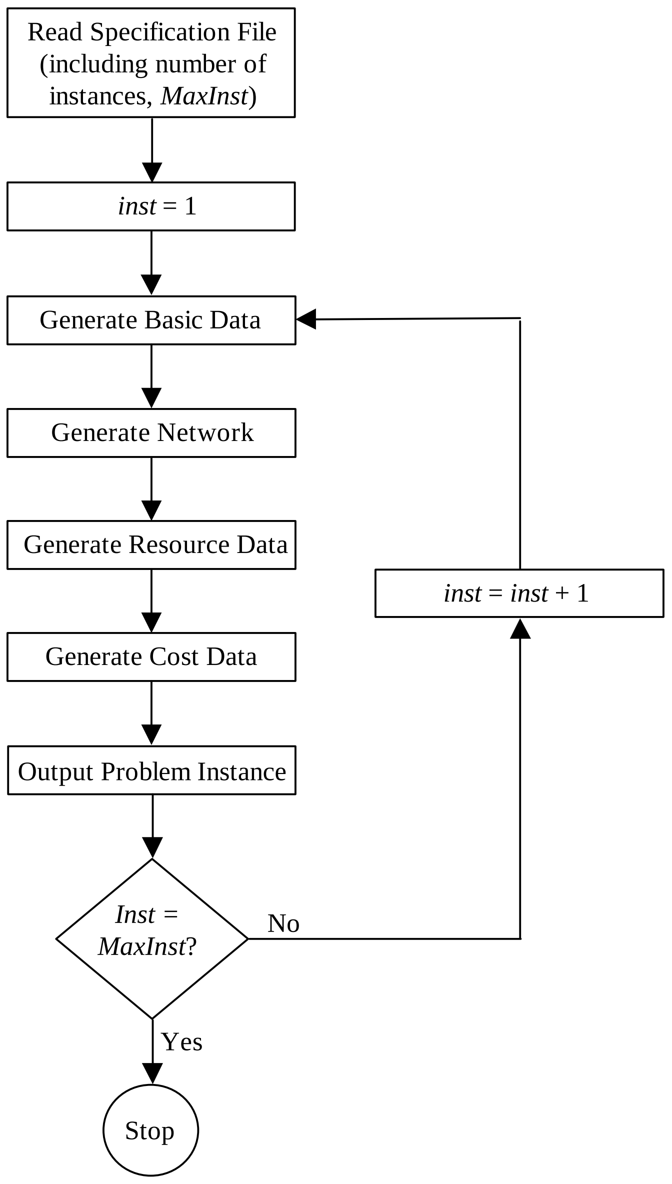


Figure 4-1. Overall Flow of PAGER

Once the specification file is read by PAGER, problem generation continues by sequentially performing Steps 2 through 6 for each problem instance desired. In the discussion which follows, the notation used in ProGen is retained wherever possible. In addition, three commonly used functions are defined as follows:

round(*x*) : rounds the value of *x* to the nearest integer

int(*x*) : truncates the value of *x* to the greatest integer ? *x* *rnd*[a, b] : a uniformly-distributed pseudorandom number from the interval [a, b]

Pseudorandom numbers are constructed by transforming [0, 1] uniformly-distributed pseudorandom numbers generated using Marsaglia's Multiply-with-Carry (MWC) generator (Wheeler, 1994). Marsaglia's MWC generator is easy to implement and produces pseudorandom number streams with an extremely long period -- about 2^{125} (Wheeler, 1994). ProGen uses the random number generator developed by Schrage (1979) which has a period of 2^{31}.

Step 2 - Basic Data Generation. Basic problem data includes the number of activities in each project, the number of modes for each activity, the program and project release dates, the program and project due date factors, and the duration of each activity mode. This data is randomly generated based on the parameters specified by the user through the problem specification file. Parameters used in generating basic data are summarized in Table 4-1.

Table 4-1. Input Parameters for Basic Data

Parameter	Definition	Bounds
P	number of projects (the program is treated as project p = 0)	[1, 10]
$J_p^{\min} / J_p^{\max}$	min/max number of activities in project p	[1, 99]
$M_p^{\min} / M_p^{\max}$	min/max number of modes per activity in project p	[1, 10]
$d_p^{\min} / d_p^{\max}$	min/max duration of activities in project p	[0, 999]
$?_p^{\min} / ?_p^{\max}$	min/max release date of project p (including p = 0)	[0, 999]
$?_p^{\min} / ?_p^{\max}$	min/max due date factor of project p (including p = 0)	[0.0, 1.0]

The procedures used to generate basic problem data (summarized in Table 4-2) are materially the same as those used in ProGen. The difference is that ProGen uses the same lower and upper bounds to generate this data for each project, while PAGER allows the user to specify different bounds for each project. The advantage is that the user has greater flexibility in designing programs. If, for example, the researcher is interested in investigating the impact of project *homogeneity* on scheduling, he/she may generate and compare problems where the projects have similar numbers of activities and activity durations versus problems where some projects have many short activities and other projects have fewer, but longer, activities. On the other hand, the researcher may want to design a program where some projects have controllably early release dates and other projects have controllably late release dates.

Table 4-2. Basic Data Variables

Parameter	Definition
P	number of projects (the program is treated as project p = 0)
J_p	the number of activities in project p
M_{pi}	the number of modes of project p, activity i
d_{pim}	the duration of project p, activity i, mode m
$?_p$	the release date of project p (including p = 0)
$?_p$	the due date factor of project p (including p = 0)

Using the parameters in Table 4-2, the following data is generated, where the program is considered to be project p = 0.

a. Number of activities in project p, including the project source and sink.

$$J_p \; ? \; round\left(rnd\left(J_p^{\min}, J_p^{\max}\right)\right) ? \; 2, \quad p \; ? \; 1,2,3,\ldots,P$$

b. Number of activities in the program, including the program supersource and supersink.

$$J_0 \; ? \; \overset{P}{\underset{i?1}{?}} J_p \; ? \; 2$$

c. Number of modes of project p, activity i.

$$M_{pi} \; ? \; round\left(rnd\left(M_p^{\min}, M_p^{\max}\right)\right), \quad p \; ? \; 1,2,3,\ldots,P$$

d. Program/Project Release Dates.

$$?_p \; ? \; round\left(rnd\left(?_p^{\min}, ?_p^{\max}\right)\right), \quad p \; ? \; 0,1,2,\ldots,P$$

e. Program/Project Due Date Factors. These are used later, in Step 3, to determine actual program and project due dates.

$$?_p \; ? \; round\left(rnd\left(?_p^{\min}, ?_p^{\max}\right)\right), \quad p \; ? \; 0,1,2,\ldots,P$$

f. Activity/mode durations. These are generated using the following algorithm, where D is a set of random integers and D_k is the kth element of D.

<u>Activity Duration Algorithm</u>

```
1.  for $p$ = 1, 2, 3, ..., $P$
2.  for $i$ = 1, 2, 3, ..., $J_p$
3.  $k$ := 1
4.  while $k \; ? \; M_{pi}$ do
          begin
5.        $D_k := round\left(rnd\left(d_p^{\min}, d_p^{\max}\right)\right)$
6.        $k := k + 1$
          end
7.  $m$ := 1
8.  while $m \; ? \; M_{pi}$ do
          begin
9.        $d^* := \min(D)$
10.       $k^* := k$ such that $D_{k^*} = d^*$
11.       $d_{pim} := d^*$
12.       $D := D \setminus \{D_{k^*}\}$
13.       $m := m + 1$
          end
```

Step 3 - Network Generation The objective of this step is to construct a connected, acyclic, non-redundant network with the user-specified complexity measure(s). Before proceeding with a description of how the network is constructed, each characteristic of the network is explained.

Generalized Precedence Relationships

In the traditional activity-on-node representation of project scheduling problems, network nodes represent activities and directed network arcs (from the end of one activity to the beginning of another) represent finish-start precedence relationships. When precedence relationships are generated such that activity *i* precedes activity *j* only if $i < j$, the project network is acyclic. Generalized precedence relationships (minimum and maximum time lags between the start times of two activities), however, have typically been treated in the literature using backward arcs (*e.g.*, from activity *j* to activity *i*), resulting in cyclic project networks (Schwindt, 1995, 1996; Drexl *et al.*, 1997; Salewski *et al.*, 1997). Using this treatment, generalized precedence relationships are created by generating cyclic project networks.

Since cyclic project networks can be intuitively confusing, PAGER uses a simplified approach to create generalized precedence relationships. Graphically, a generalized precedence is represented by a forward arc from the beginning of one activity to the beginning of another as in Figure 4-2. Generalized precedences are created by *converting* traditional finish-start precedence relationships. Recall that traditional finish-start precedence relationships are a special case of generalized precedences where the minimum time lag equals the duration of the *predecessor* activity and the maximum time lag is infinite. Generalized precedences are, therefore, created by re-specifying the minimum and maximum time lags for a subset of the existing precedence relations.

PAGER allows the user to specify a lower and upper bound for the minimum time lag and a lower and upper bound for the maximum time lag. Minimum and maximum time lags may be negative. For instance, if the minimum and maximum time lags from activity *i* to activity *j* are -3 and 5, respectively, and activity *i* starts at time *t*, then activity *j* may start anywhere in the interval [t-3, t+5]. If the minimum and maximum time lags are both negative, the relationship is still valid provided that the minimum time lag is less than or equal to the maximum time lag.

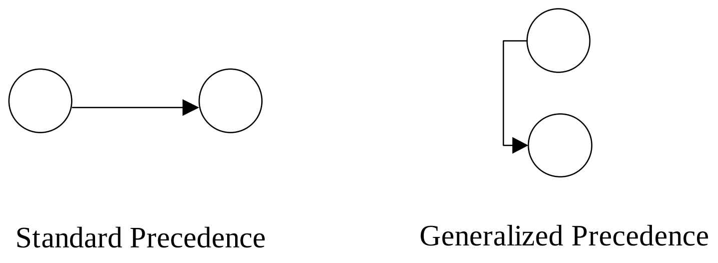


Figure 4-2. Standard and Generalized Precedence Arcs

In the definitions which follow, arc set *A* is assumed to consist of standard precedences only. This assumption can be made without loss of generality since, as stated above, generalized precedences are treated as standard precedences with respecified time lags.

<u>Connected Network</u>

The problem network must be connected.

Definition 4-1. Network Connectivity (Kolisch *et al.*, 1992: 5)

Let *G* = (*N*, *A*) be a directed graph with node set *N* and arc set *A*. *G* is *connected* if, for every node $j \in N$, there is a directed path in *G* from the single source node to *j* and a directed path in *G* from *j* to the single sink node.

Definition 4-2. Reachability (Schwindt, 1996: 7)

A node $j \in N$ is called *reachable* from node *i* if :

(i) *j* = *i*, or

(ii) there is a directed path W_{ij} with origin *i* and terminus *j*.

Definition 4-3. Reachability Matrix of a Digraph (Schwindt, 1996: 7)

The *reachability matrix* **R** of digraph *G* = (*N*, *A*) is the $n \times n$ matrix $(r_{ij})_{i,j \in N}$ with

$$r_{ij} = \begin{cases} 1, \text{ if } j \text{ is reachable from } i \\ 0, \text{ otherwise} \end{cases}.$$

Definition 4-4. Network Connectivity (Alternate Definition)

Let $G = (N, A)$ be a directed graph with reachability matrix **R**, source node s, and sink node t. G is *connected* if, for every $j ? N$, $r_{sj} = 1$ and $r_{jt} = 1$.

Acyclic Network

The network must be acyclic.

Definition 4-5. Strongly/Weakly Connected (Schwindt, 1996: 8)

Let $G = (N, A)$ be a directed graph with reachability matrix **R**. Nodes $i, j ? N$, $i ? j$, are *strongly connected* if j is reachable from i ($r_{ij} = 1$) and i is reachable from j ($r_{ji} = 1$). Nodes i and j are *weakly connected* if they are not strongly connected but are connected in the corresponding undirected graph ($r_{ij} + r_{ji} = 1$)

Definition 4-6. Cyclic/Acyclic Network

Let $G = (N, A)$ be a directed graph. G is *cyclic* if there exists any two nodes $i, j ? N$, $i ? j$, such that i and j are strongly connected. Otherwise, G is *acyclic*.

Non-Redundant Network

The network will be non-redundant. Though redundancy within the network does not invalidate the network, it does adversely affect the coefficient of network complexity (CNC). If the CNC (the average number of arcs per node) is used in determining the structure of the problem network, then redundant arcs cause an overstatement of the CNC and a network structure with fewer real temporal relationships than believed. Practically speaking, redundant arcs may also increase the number of temporal relationships which must be considered when scheduling the resultant problem.

Definition 4-7. Redundant Arc (Schwindt, 1996: 9)

Let $G = (N, A)$ be an acyclic digraph. An arc (i, j) is *redundant* if there exists a directed path W_{ij} in $G – (i,j)$ with more than one arc.

Remark 1. (Schwindt, 1996: 9)

Arc (*i*, *j*) is redundant if and only if $\bar{r}_{ij}$? 1 for G^* ? $(N, A \setminus \{(i, j)\})$ with reachability matrix $\mathbf{R}^*$.

Definition 4-8. Non-Redundant Network

Let *G* = (*N*, *A*) be an acyclic digraph. *G* is a *non-redundant network* if there are no redundant arcs in *G*.

Relationships Between Projects

Just as activities within or between projects may be temporally related, projects themselves may also be temporally related. Specifically, the source node of one project is, to some degree, related to the (numerically) preceding project. The degree of this relationship is referred to as the *project lag*.

The project lag is determined using the *project lag coefficient*, L_p, which can take any continuous value in the range of [0, 1]. Consider three cases, represented in Figure 4-3.

a. $L_p = 0$. The start of project *p*+1 is not dependent on project *p*. Hence, the source node of project *p*+1 is a successor of the program super-source node only. In Figure 4-3, the project lag coefficient between projects 1 and 2 is zero.

b. $L_p = 1$. Project *p*+1 succeeds project *p*. This is the case where one project, *p*, must be entirely completed before another project, *p*+1, begins. This might happen if the projects are sequential stages in a process where the end of one project and beginning of the next represents a milestone in the process. The source node of project *p*+1 is a successor of the sink node of project *p*. Projects 3 and 4 in Figure 4-3 have a project lag coefficient of one.

c. L_p ? $(0,1)$. The start of project *p*+1 is dependent on the successful completion of some phase of project *p*, say activity *j* of project *p*. Activity *j* of project *p* may, for instance, be the final approval of some critical technology needed for project *p*+1. In Figure 4-3, projects 2 and 3 demonstrate a project lag coefficient greater than zero but less than one. The source node of project *p*+1 is a successor of activity *j* of project *p*.

The project lag coefficient is randomly generated using the following equation:

$$L_p \; ? \; rnd\left[L_p^{\min}, L_p^{\max}\right],$$

where $L_p^{\min}$ and $L_p^{\max}$ are the user-specified minimum and maximum values of L_p, respectively. Once the project lag coefficient for project *p* has been generated, the activity in project *p* which will precede the source node of project *p*+1 is determined by:

$$j \; ? \; round\, ?1 \; ? \; L_p(J_p \; ? \; 1)?$$

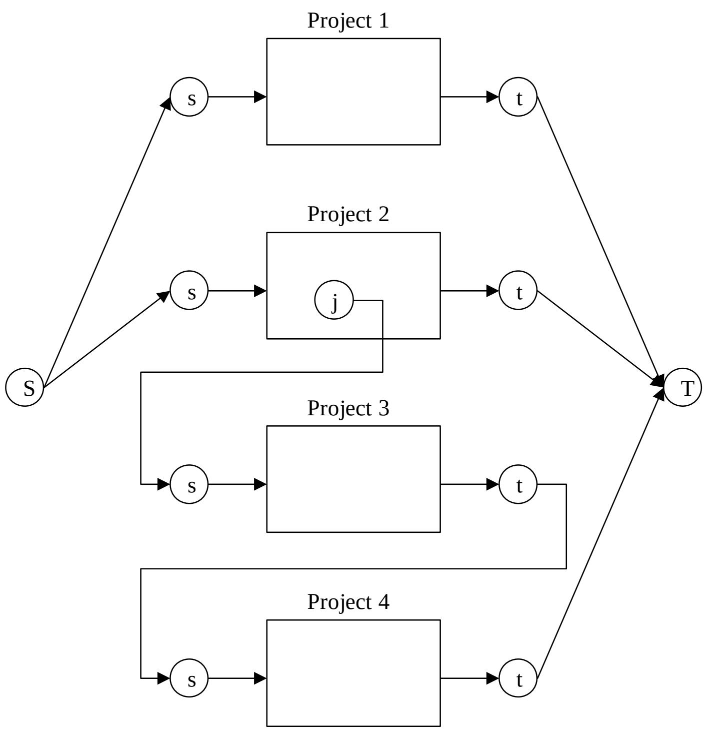


Figure 4-3. Project Lags

Network Complexity

Elmaghraby and Herroelen (1980) state that some measure of complexity in the project network is required to (1) serve as a predictor of the processing time requirements for a particular software package on a particular hardware platform and (2) enable proper comparisons between competing algorithms. Three different measures of network complexity are used in the problem generators discussed above: the coefficient of network complexity, the complexity index, and Thesen's restrictiveness measure.

Demeulemeester *et al*. (1993) and Kolisch *et al*. (1992, 1995) use the coefficient of network complexity (CNC) as their measure of network complexity. CNC, the ratio of arcs to nodes in the network, is easily implemented in a problem generator, but it has shortcomings. The measure is not normalized to the interval [0, 1] and so does not reflect network complexity relative to the number of network nodes. A CNC of 3, for example, has different implications for a network with 100 nodes than it does for a network with only 10 nodes. In the 100-node network, each node immediately precedes only 3% of the network nodes, on average. In the 10-node network, however, each node immediately precedes 30% of the network nodes, on average, which is far more constrained than the 100-node network. To alleviate this problem, some authors have used the order strength (*e.g.*, Cooper, 1976) which is calculated by dividing the number of arcs by the maximum number of possible arcs, $n(n-1)/2$. As Kolisch *et al*. (1992, 1995) suggest, though, the maximum number of possible arcs includes redundant arcs and is far greater than a realistic number of precedence relationships in a project network.

De Reyck and Herroelen (1996) study the impact of CNC on problem solution time compared to a second measure, the complexity index (CI). CI, a measure of how near a network is to being series-parallel, is defined as the number of *node reductions* (in concert with *series* and *parallel reductions*) required to reduce a project network to a *two-terminal network* (see Valdes *et al*., 1982; Bein *et al*., 1992). De Reyck and Herroelen conclude that CNC is a poor indicator of problem difficulty and propose that CI is a superior measure. Agrawal *et al*. (1996) make the same conclusion and use CI in their problem generator, DAGEN. One drawback of using CI is that it requires the use of an activity-on-arc representation of the project network as opposed to the activity-on-node representation.

A third measure of network complexity is Thesen's measure of restrictiveness (RT) (Thesen, 1977). RT measures the degree to which the number of possible activity sequences has been restricted by the imposition of precedence constraints. Schwindt (1995, 1996), who uses RT in his ProGen/max generator, perceives RT as a more intuitive measure than CI and conjectures that it will play an even more important role than CI in predicting computational effort for resource-constrained project scheduling problems. De Reyck (1995) conducted an extensive computational study and confirmed Schwindt's conjecture. Drexl *et al*. (1997) also use RT as the measure of complexity in their ProGen/? *x* generator.

While problem generators have typically relied on a single measure of network complexity, PAGER provides two measures, CNC and RT. These can be used separately or simultaneously. RT is the primary complexity measure used by PAGER because of its increasing acceptance as the best available measure, as well as its intuitive appeal. CNC is also provided as an option for three reasons. It provides the user the ability to generate problem sets comparable to other problem sets cited in the literature. It provides means to investigate the power of RT and CNC, used together, to explain the solution time of problem instances. It may, perhaps, open the door for future research in using simultaneous measures of network complexity.

Definition 4-9. Restrictiveness of a Graph (Thesen, 1977: 197)

Let G = (N, A) be an acyclic digraph with unique source node 1, unique sink node n = |N|, and reachability matrix **R**. Let ? denote the number of possible permutations of the sequence $?i_1, i_2, \ldots, i_{n?2}?$ of $N' ? ?1, \ldots, n ? 2? ? N$ such that if $j ? k ? \;\; r_{kj} ? 0$. Then, the restrictiveness of G is defined as $\mathrm{P} ? 1 ? \dfrac{\log ?}{\log ?_{\max}}$, where $?_{\max} ? (n ? 2)!$.

Remark 2. (Thesen, 1977: 197)

$\mathrm{P} ? [0,1]$, P = 0 for parallel digraphs, and P = 1 for series digraphs.

While P may be an appropriate measure of network complexity, finding ? is a difficult combinatorial problem (Schwindt, 1995). Consequently, Thesen developed and tested over 40 different indirect estimators of P and found RT to yield the lowest mean relative error with respect to P.

Definition 4-10. Disjunctive Arc (Schwindt, 1996: 18)

Let G = (N, A) be an acyclic digraph with reachability matrix **R**. *Disjunctive arcs* are imaginary, undirected arcs between pairs of nodes $i, j ? N$ such that $r_{ij} = r_{ji} = 0$.

Definition 4-11. Restrictiveness Estimator RT (Schwindt, 1996: 18)

Let $G = (N, A)$ be an acyclic digraph with unique source node 1, unique sink node $n = |N|$, and reachability matrix **R**. Let n_d denote the number of disjunctive arcs in G and let $n_d^{\max} ? \frac{(n ? 2)(n ? 3)}{2}$ be the maximum number of possible disjunctive arcs in a weakly connected digraph with node set N. Then, the restrictiveness estimator RT is defined as

$$\mathrm{RT} ? 1 ? \frac{n_d}{n_d^{\max}} ? 1 ? \frac{n(n ? 1) ? 2 \underset{i,j ? N}{?} r_{ij}}{(n ? 2)(n ? 3)} ? \frac{2 \underset{i,j ? N}{?} r_{ij} ? 6(n ? 1)}{(n ? 2)(n ? 3)}.$$

Schwindt (1996) provides Theorem 4-1 (stated here without proof) describing the behavior of RT.

Theorem 4-1. (Schwindt, 1996: 19)

(i) $\mathrm{RT} ? [0,1]$.

(ii) RT = 0 for parallel digraphs.

(iii) RT = 1 for series digraphs.

(iv) The insertion of a non-redundant arc increases RT.

(v) The insertion of a redundant arc does not affect RT.

One of the unique features of PAGER is the role the reachability matrix **R** plays in generating the problem network. While the exact procedure is reserved for the next subsection, the underlying theory is developed here.

Recall that problem generators ProGen/*max* and ProGen/? *x* use RT as the measure of network complexity. Both of these generators use the same procedure as ProGen for creating an acyclic, connected network. Generally, they

1. determine the number of start and end nodes, connecting these to the source and sink nodes, respectively,
2. determine a direct predecessor for each node which does not already have one,
3. determine a direct successor for each node which does not already have one, then

4. add additional non-redundant arcs until the desired complexity is achieved.

This ProGen-based procedure is simple to implement and is very useful when CNC is the measure of network complexity. CNC is easily controlled through this procedure since arcs are added one by one, increasing CNC by exactly 1/*n*, and the procedure is simply terminated when the desired CNC is reached.

When RT is incorporated into the above procedure, arcs are added until RT has been met or exceeded. Unfortunately, the procedure lacks direct control over RT. Unlike CNC, the addition of an arc does not, in general, increase RT by any predetermined amount. The effectof an arc addition on RT must be determined after the fact. Consider the following example.

Figure 4-4(a) depicts a network with CNC = 12/10 = 1.2 and RT = 13/28 ? 0.464. If an arc is added from node 2 to node 5, Figure 4-4(b), the CNC increases by 1/10 to 1.3 and the RT increases by 1/28 to 0.5. If, on the other hand, with the addition of an arc from node 3 to node 4, Figure 4-4(c), the CNC still increases by 1/10 to 1.3, but the RT jumps by 5/28 to 0.643. If the desired RT is somewhere in the open interval between 0.5 and 0.643, it is unclear how the RT will be achieved without a trial-and-error process of adding and removing arcs. Schwindt (1995, 1996) adds arcs until the desired RT is met or exceeded and then stops. As demonstrated above, the resulting RT may be materially beyond what the user intended.

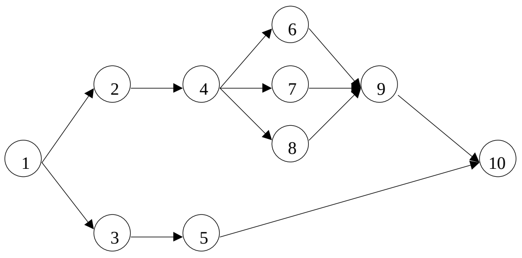
PAGER's approach to generating problem networks is to work in the domain of the reachability matrix **R**. If **R** can be manipulated directly and a corresponding network produced, then RT can be precisely controlled. The development of a project network using reachability matrix **R** requires Definitions 4-12 through 4-15 and Theorems 4-2 and 4-3. Unless otherwise noted, these definitions and theorems are original to this research.

Definition 4-12. Restricted Reachability Matrix

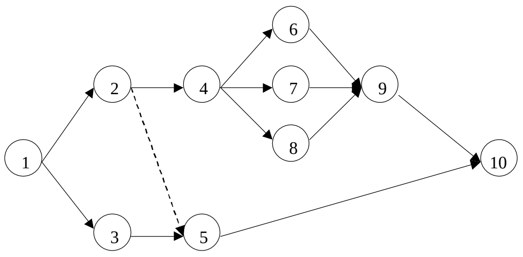
Let *G* = (*N*, *A*) be an acyclic digraph with reachability matrix **R**. Then, the *restricted reachability matrix* $\overline{\mathbf{R}}$ is the $n ? n$ matrix $?\bar{r}_{ij}?_{i,j ? N}$ with

$$\bar{r}_{ij} ? \begin{cases} 1, \text{ if } j \text{ is reachable from } i, i ? j \\ 0, \text{ otherwise} \end{cases}.$$

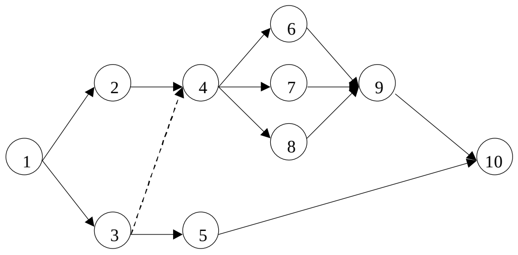
That is, $\overline{\mathbf{R}} ? \mathbf{R} ? \mathbf{I}$, where *i* is the $n ? n$ identity matrix.



(a)



(b)



(c)

Figure 4-4. CNC versus RT

Definition 4-13. Adjacency Matrix (Schwindt, 1996: 6)

Let G = (N, A) be an acyclic digraph. The *adjacency matrix* **A** of G is the $n \times n$ matrix

$$\left(a_{ij}\right)_{i,j \in N} \text{ with } a_{ij} = \begin{cases} 1, \text{ if } i \text{ precedes } j, (i, j) \in A \\ 0, \text{ otherwise} \end{cases}.$$

Theorem 4-2. $\overline{\mathbf{R}}$ Uniquely Determines **A**

Theorem: Let G = (N, A) be an acyclic, non-redundant digraph with adjacency matrix **A** and restricted reachability matrix $\overline{\mathbf{R}}$. $\overline{\mathbf{R}}$ uniquely determines the adjacency matrix **A** as follows: $\mathbf{A} ? \overline{\mathbf{R}} ? ? ?\overline{\mathbf{R}}^2 ?$, where ? operates on the elements of a matrix **X** such that $? (x_{ij}) ? \begin{cases} ?1, \text{if } x_{ij} ? 1 \\ ? \\ ?0, \text{otherwise} \end{cases}$.

Proof: Let G = (N, A) be an acyclic, non-redundant digraph with adjacency matrix **A** and restricted reachability matrix $\overline{\mathbf{R}}$. Assume, without loss of generality, that nodes are labeled such that if $\overline{r}_{ij} ? 1$, then $i < j$. Show that $a_{ij} ? \overline{r}_{ij} ? ? ?\overline{r}_{ij}^2 ?$ for each node pair $i, j ? N$.

Three cases exist for any node pair $i, j ? N$: (1) $i ? j$, (2) $i ? j$ and a_{ij} = 1, or (3) $i ? j$ and a_{ij} = 0. Consider each case separately.

<u>Case 1</u>. $i ? j$

$i ? j ? \overline{r}_{ij} ? 0$ by assumption and $\overline{r}_{ij} ? 0 ? a_{ij} ? 0$. Then,

$$\begin{aligned} \overline{r}_{ij}^2 &? \sum_{k ? N} \overline{r}_{ik}\overline{r}_{kj} \\ &? \sum_{k ? j ? i} \overline{r}_{ik}\overline{r}_{kj} ? \sum_{k ? i ? j} \overline{r}_{ik}\overline{r}_{kj} \\ &? \sum_{k ? j ? i} 0 ?\overline{r}_{kj} ? \sum_{k ? i ? j} \overline{r}_{ik} ?0 \\ &? 0 \end{aligned}$$

and so $? ?\overline{r}_{ij}^2 ?? 0$. Therefore, $0 ? a_{ij} ? \overline{r}_{ij} ? ? ?\overline{r}_{ij}^2 ?? 0 ? 0 ? 0$.

<u>Case 2</u>. $i ? j$ and a_{ij} = 1

$a_{ij} ? 1 ? r_{ij} ? 1$. That is, $(i, j) ? A ? j$ is reachable from i. Then,

$$\begin{aligned}\bar{r}_{ij}^2 &? \underset{k ? N}{?} \bar{r}_{ik}\bar{r}_{kj}\\ &? \underset{k ? i}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{k ? j}{?} \bar{r}_{ik}\bar{r}_{kj}\\ &? \underset{k ? i}{?} 0 ?\bar{r}_{kj} ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{k ? j}{?} \bar{r}_{ik} ?0\\ &? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj}\end{aligned}$$

Now, $\bar{r}_{ij}^2 ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? 1$ if and only if there exists some k, $i < k < j$, such that $\bar{r}_{ik}\bar{r}_{kj} ? 1$. Assume such a k exists. Then, there exists a directed path, W_{ij}, from i to j in G with more than one arc. This is a contradiction, however, since G is non-redundant and $a_{ij} ? 1$. Thus, $\bar{r}_{ij}^2 ? 0$ and $? ?\bar{r}_{ij}^2 ? ? 0$.

Therefore, $1 ? a_{ij} ? \bar{r}_{ij} ? ? ?\bar{r}_{ij}^2 ? ? 1 ? 0 ? 1$.

<u>Case 3</u>. $i ? j$ and $a_{ij} = 0$

Since $a_{ij} ? 0$, two possibilities exist: (a) i does not reach j ($r_{ij} = 0$) and (b) i reaches j ($r_{ij} = 1$). Consider both possibilities.

(a) i does not reach j ($r_{ij} = 0$). Then,

$$\begin{aligned}\bar{r}_{ij}^2 &? \underset{k ? N}{?} \bar{r}_{ik}\bar{r}_{kj}\\ &? \underset{k ? i}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{k ? j}{?} \bar{r}_{ik}\bar{r}_{kj}\\ &? \underset{k ? i}{?} 0 ?\bar{r}_{kj} ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? \underset{k ? j}{?} \bar{r}_{ik} ?0\\ &? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj}\end{aligned}$$

Now, $\bar{r}_{ij}^2 ? \underset{i ? k ? j}{?} \bar{r}_{ik}\bar{r}_{kj} ? 1$ if and only if there exists some k, $i < k < j$, such that $\bar{r}_{ik}\bar{r}_{kj} ? 1$. Assume such a k exists. Then, there exists a directed path, W_{ij}, from i to j in G with more than one arc. This is a contradiction, however, since i does not reach j. Thus, $\bar{r}_{ij}^2 = 0$ and $?\left(\bar{r}_{ij}^2\right) = 0$. Therefore,

$0 = a_{ij} = \bar{r}_{ij} - ?\left(\bar{r}_{ij}^2\right) = 0 - 0 = 0$.

(b) i reaches j ($r_{ij} = 1$). Then,

$$
\begin{aligned}
\bar{r}_{ij}^2 &= \sum_{k \in N} \bar{r}_{ik}\bar{r}_{kj} \\
&= \sum_{k \le i} \bar{r}_{ik}\bar{r}_{kj} + \sum_{i < k < j} \bar{r}_{ik}\bar{r}_{kj} + \sum_{k \ge j} \bar{r}_{ik}\bar{r}_{kj} \\
&= \sum_{k \le i} 0 \cdot \bar{r}_{kj} + \sum_{i < k < j} \bar{r}_{ik}\bar{r}_{kj} + \sum_{k \ge j} \bar{r}_{ik} \cdot 0 \\
&= \sum_{i < k < j} \bar{r}_{ik}\bar{r}_{kj}
\end{aligned}
$$

Now, $\bar{r}_{ij}^2 = \sum_{i < k < j} \bar{r}_{ik}\bar{r}_{kj} = 1$ if and only if there exists some k, $i < k < j$, such that $\bar{r}_{ik}\bar{r}_{kj} = 1$, or in other words, there exists a directed path, W_{ij}, from i to j in G with more than one arc. Since i does reach j, $\bar{r}_{ij}^2 = 1$ and $?\left(\bar{r}_{ij}^2\right) = 1$.

Therefore, $0 = a_{ij} = \bar{r}_{ij} - ?\left(\bar{r}_{ij}^2\right) = 1 - 1 = 0$.

QED

Remark 3.

$$
a_{ij} = \bar{r}_{ij} - ?\left(\bar{r}_{ij}^2\right) = \begin{cases} -1, \text{ if } \bar{r}_{ij} = 0 \text{ and } ?\left(\bar{r}_{ij}^2\right) = 1 \\ 0, \text{ if } \bar{r}_{ij} = 0 \text{ and } ?\left(\bar{r}_{ij}^2\right) = 0 \\ 0, \text{ if } \bar{r}_{ij} = 1 \text{ and } ?\left(\bar{r}_{ij}^2\right) = 1 \\ 1, \text{ if } \bar{r}_{ij} = 1 \text{ and } ?\left(\bar{r}_{ij}^2\right) = 0 \end{cases}, \text{ for } i, j \in N.
$$

Definition 4-14. r-Deletion

Let $\overline{\mathbf{R}}$ be a restricted reachability matrix. The *r-deletion* of node pair (l, m), $l, m \in N$, is the change from $\bar{r}_{lm} = 1$ to $\bar{r}_{lm} = 0$ in $\overline{\mathbf{R}}$.

Definition 4-15. Feasible r-Deletion

Let $\overline{\mathbf{R}}_1$ be a restricted reachability matrix. Let $\overline{\mathbf{R}}_2$ be $\overline{\mathbf{R}}_1$ after the r-deletion of some node pair (*l*, *m*), $l, m ? N$. The *r-deletion* of node pair (*l*, *m*) is *feasible* if $\mathbf{A}_2 ? \overline{\mathbf{R}}_2 ? ? (\overline{\mathbf{R}}_2^2)$ remains a proper adjacency matrix (that is, $(\mathbf{A}_2)_{ij}$ = {0, 1} for all $i, j ? N$).

Theorem 4-3. r-deletion Feasibility

Theorem: Let $\overline{\mathbf{R}}_1$ be a restricted reachability matrix with corresponding adjacency matrix **A**$_1$. Let $\overline{\mathbf{R}}_2$ be $\overline{\mathbf{R}}_1$ after the r-deletion of some node pair (*l*, *m*), $l, m ? N$. The r-deletion of node pair (*l*, *m*) is feasible if and only if $(\mathbf{A}_1)_{lm} ? 1$.

Proof: Let $\overline{\mathbf{R}}_1$ be a restricted reachability matrix with corresponding adjacency matrix **A**$_1$. Let (*l*, *m*) be some node pair, $l, m ? N$, such that $(\overline{\mathbf{R}}_1)_{lm}$ = 1. Let $\overline{\mathbf{R}}_2$ be the resulting matrix when $(\overline{\mathbf{R}}_1)_{lm}$ is changed from 1 to 0. The proof demonstrates (i) the r-deletion of (*l*, *m*) is feasible if $(\mathbf{A}_1)_{lm} ? 1$ and (ii) the r-deletion of (*l*, *m*) is NOT feasible if $(\mathbf{A}_1)_{lm} ? 0$.

(i) Show that $(\mathbf{A}_1)_{lm} ? 1 ?$ the r-deletion of (*l*, *m*) is feasible.

Assume $(\mathbf{A}_1)_{lm} ? 1$. The r-deletion of (l, m) ? $(\overline{\mathbf{R}}_2)_{lm} ? 0$.

When $i ? l$ and $j ? m$, $(\overline{\mathbf{R}}_2)_{ij} ? (\overline{\mathbf{R}}_1)_{ij}$,

$(\overline{\mathbf{R}}_2^2)_{ij} ? \underset{k ? N}{?} (\overline{\mathbf{R}}_2^2)_{ik} (\overline{\mathbf{R}}_2^2)_{kj} ? \underset{k ? N}{?} (\overline{\mathbf{R}}_1^2)_{ik} (\overline{\mathbf{R}}_1^2)_{kj} ? (\overline{\mathbf{R}}_1^2)_{ij}$, and

$\left(\mathbf{A}_2\right)_{ij} ? \left(\overline{\mathbf{R}}_2\right)_{ij} ? ? \left(\left(\overline{\mathbf{R}}_2^2\right)_{ij}\right) ? \left(\overline{\mathbf{R}}_1\right)_{ij} ? ? \left(\left(\overline{\mathbf{R}}_1^2\right)_{ij}\right) ? \left(\mathbf{A}_1\right)_{ij}$. This implies that $\left(\mathbf{A}_2\right)_{ij} ? \left(\mathbf{A}_1\right)_{ij}$ remains feasible for $i ? l$ and $j ? m$.

Now consider when $i = l$ and $j ? m$.

$$\left(\overline{\mathbf{R}}_2\right)_{lj} ? \left(\overline{\mathbf{R}}_1\right)_{lj} \text{ and}$$

$$\begin{aligned}
\left(\overline{\mathbf{R}}_2^2\right)_{lj} &? \underset{k?N}{?} \left(\overline{\mathbf{R}}_2^2\right)_{lk}\left(\overline{\mathbf{R}}_2^2\right)_{kj} \\
&? \underset{k?m}{?} \left(\overline{\mathbf{R}}_2^2\right)_{lk}\left(\overline{\mathbf{R}}_2^2\right)_{kj} ? \left(\overline{\mathbf{R}}_2^2\right)_{lm}\left(\overline{\mathbf{R}}_2^2\right)_{mj} ? \underset{k?m}{?} \left(\overline{\mathbf{R}}_2^2\right)_{lk}\left(\overline{\mathbf{R}}_2^2\right)_{kj} \\
&? \underset{k?m}{?} \left(\overline{\mathbf{R}}_2^2\right)_{lk}\left(\overline{\mathbf{R}}_2^2\right)_{kj} ? 0 ? \underset{k?m}{?} \left(\overline{\mathbf{R}}_2^2\right)_{lk}\left(\overline{\mathbf{R}}_2^2\right)_{kj} \\
&? \underset{k?m}{?} \left(\overline{\mathbf{R}}_1^2\right)_{lk}\left(\overline{\mathbf{R}}_1^2\right)_{kj} ? \left(\overline{\mathbf{R}}_1^2\right)_{lm}\left(\overline{\mathbf{R}}_1^2\right)_{mj} ? \underset{k?m}{?} \left(\overline{\mathbf{R}}_1^2\right)_{lk}\left(\overline{\mathbf{R}}_1^2\right)_{kj} \\
&? \underset{k?N}{?} \left(\overline{\mathbf{R}}_1^2\right)_{lk}\left(\overline{\mathbf{R}}_1^2\right)_{kj} \\
&? \left(\overline{\mathbf{R}}_1^2\right)_{lj}
\end{aligned}$$

Therefore, $\left(\mathbf{A}_2\right)_{lj} ? \left(\overline{\mathbf{R}}_2\right)_{lj} ? ? \left(\left(\overline{\mathbf{R}}_2^2\right)_{lj}\right) ? \left(\overline{\mathbf{R}}_1\right)_{lj} ? ? \left(\left(\overline{\mathbf{R}}_1^2\right)_{ij}\right) ? \left(\mathbf{A}_1\right)_{ij}$. If $\left(\mathbf{A}_1\right)_{ij} ? 0$, then $\left(\mathbf{A}_2\right)_{ij} ? \{0,1\}$ (see Remark 3) and if $\left(\mathbf{A}_1\right)_{ij} ? 1$, then $\left(\mathbf{A}_2\right)_{ij} ? 1$. This implies that $\left(\mathbf{A}_2\right)_{ij} ? \left(\mathbf{A}_1\right)_{ij}$ remains feasible for $i ? l$ and $j ? m$.

Consider next when $i ? l$ and $j ? m$. The same argument used for the case when $i ? l$ and $j ? m$ holds here.

Finally, consider when $i ? l$ and $j ? m$.

$$\left(\overline{\mathbf{R}}_1\right)_{lm} ? 1,\quad \left(\overline{\mathbf{R}}_2\right)_{lm} ? 0 \text{ and}$$

$$\begin{aligned}
(\overline{\mathbf{R}}_2^2)_{lm} &? \underset{k?N}{?} (\overline{\mathbf{R}}_2^2)_{lk}(\overline{\mathbf{R}}_2^2)_{km} \\
&? \underset{k?l,m}{?} (\overline{\mathbf{R}}_2^2)_{lk}(\overline{\mathbf{R}}_2^2)_{km} ? (\overline{\mathbf{R}}_2^2)_{ll}(\overline{\mathbf{R}}_2^2)_{lm} ? (\overline{\mathbf{R}}_2^2)_{lm}(\overline{\mathbf{R}}_2^2)_{mm} \\
&? \underset{k?l,m}{?} (\overline{\mathbf{R}}_2^2)_{lk}(\overline{\mathbf{R}}_2^2)_{kj} ? 0?0 ? 0?0 \\
&? \underset{k?l,m}{?} (\overline{\mathbf{R}}_1^2)_{lk}(\overline{\mathbf{R}}_1^2)_{kj} ? 0?1 ? 0?1 \\
&? \underset{k?l,m}{?} (\overline{\mathbf{R}}_1^2)_{lk}(\overline{\mathbf{R}}_1^2)_{km} ? (\overline{\mathbf{R}}_1^2)_{ll}(\overline{\mathbf{R}}_1^2)_{lm} ? (\overline{\mathbf{R}}_1^2)_{lm}(\overline{\mathbf{R}}_1^2)_{mm} \\
&? \underset{k?N}{?} (\overline{\mathbf{R}}_1^2)_{lk}(\overline{\mathbf{R}}_1^2)_{km} \\
&? (\overline{\mathbf{R}}_1^2)_{lm}
\end{aligned}$$

Since $(\mathbf{A}_1)_{lm} ? (\overline{\mathbf{R}}_1)_{lm} ? ? ((\overline{\mathbf{R}}_1^2)_{lm}) ? 1$ and $(\overline{\mathbf{R}}_1)_{lm} ? 1$,

$? ((\overline{\mathbf{R}}_1^2)_{lm}) ? 0 \;?\; ? ((\overline{\mathbf{R}}_2^2)_{lm}) ? 0$. Therefore,

$(\mathbf{A}_2)_{lm} ? (\overline{\mathbf{R}}_2)_{lm} ? ? ((\overline{\mathbf{R}}_2^2)_{lm}) ? 0 ? 0 ? 0$ which, of course, is feasible and expected.

(ii) Show that $(\mathbf{A}_1)_{lm} ? 0 \;?$ the r-deletion of (l, m) is NOT feasible.

Assume $(\mathbf{A}_1)_{lm} ? 0$. The r-deletion of (l, m) $?$ $(\overline{\mathbf{R}}_2)_{lm} ? 0$.

Consider $(\mathbf{A}_2)_{lm} ? (\overline{\mathbf{R}}_2)_{lm} ? ? ((\overline{\mathbf{R}}_2^2)_{lm}) ? 0 ? ? ((\overline{\mathbf{R}}_2^2)_{lm}) ? ?? ((\overline{\mathbf{R}}_2^2)_{lm})$.

As shown above, $? ((\overline{\mathbf{R}}_2^2)_{lm}) ? ? ((\overline{\mathbf{R}}_1^2)_{lm}) ? (\overline{\mathbf{R}}_1)_{lm} ? (\mathbf{A}_1)_{lm} ? 1 ? 0 ? 1$ which implies that $(\mathbf{A}_2)_{lm} ? ?? ((\overline{\mathbf{R}}_2^2)_{lm}) ? ?1$ which is not feasible.

QED

Network Generation Procedure

With the above definitions and theorems, it is now possible to construct a project network with precise control over RT. This is done by starting with a restricted reachability matrix $\overline{\mathbf{R}}$ where

$\bar{r}_{ij}$? 1 for all i, j ? N, $i < j$, calculating $\mathbf{A}$? $\overline{\mathbf{R}}$? ? $\overline{\mathbf{R}}^2$?, randomly choosing a node pair (l, m), 1 ? l ? m ? n, such that a_{lm} ? 1, and r-deleting node pair (l, m). This procedure, depicted in Figure 4-5, is repeated until the desired RT and CNC are obtained. Note that r-deletion is limited to (l, m) where 1 ? l, m ? n. This is done because network connectivity is maintained, by definition, when $\bar{r}_{1j}$? 1 for all j = 2,..., n and $\bar{r}_{in}$? 1 for all i = 1,..., n-1.

Table 4-3 lists the input parameters required for generation of a project network. The use of these parameters is detailed below.

Table 4-3. Input Parameters for Project Network Generation

Parameter	Definition	Bounds
$S_{p1}^{\min} / S_{p1}^{\max}$	minimum/maximum number of start activities in project p	[1, 99]
$P_{pJ}^{\min} / P_{pJ}^{\max}$	minimum/maximum number of finish activities in project p	[1, 99]
$S_p^{\max}$	max number of successors per activity for project p	[1, 99]
$P_p^{\max}$	max number of predecessors per activity for project p	[1, 99]
LF_p	fraction of arcs in project p which denote generalized prec	[0.0, 1.0]
$LL_p^{\min} / LL_p^{\max}$	lower/upper bounds on the minimum lag times for project p	[-99, 99]
$LU_p^{\min} / LU_p^{\max}$	lower/upper bounds on the maximum lag times for project p	[-99, 99]
CNC_p	coeff. of network complexity (arcs per node) for project p	[0, 999]
tol_{CNC}	tolerance on coefficient of network complexity	[0.0, 1.0]
TH_p	Thesen Restrictiveness measure for project p	[0.0, 1.0]
tol_{TH}	tolerance on Thesen Restrictiveness	[0.0, 1.0]

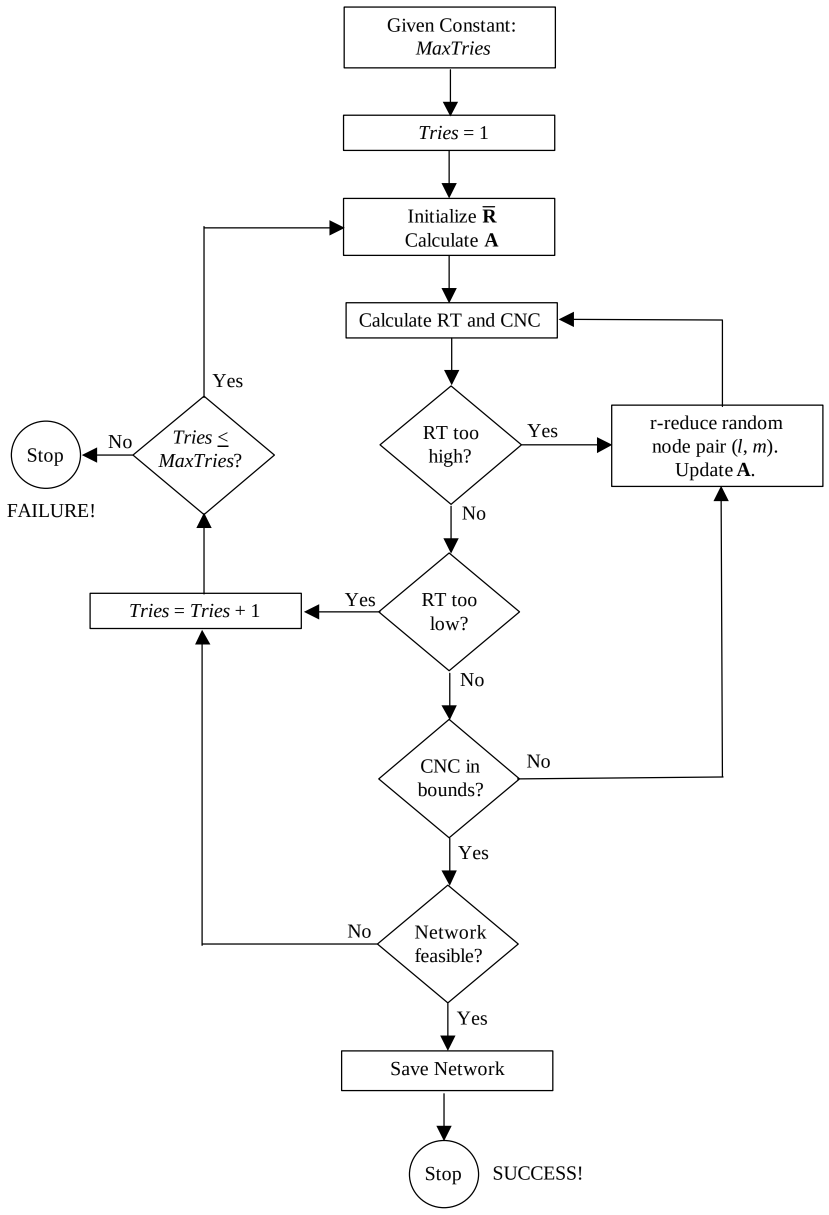


Figure 4-5. Generation of a Project Network

Figure 4-6 illustrates the project network generation procedure. The lighter gray cells of matrix $\overline{\mathbf{R}}$ correspond to entries of $\overline{\mathbf{R}}$ which cannot be changed. The darker gray cells are those which can be changed to generate a project network of some desired RT. If the desired RT is 0.39, the illustrated example yields an acyclic, non-redundant, connected digraph (network) with RT = 0.39.

$\overline{\mathbf{R}}$ $\overline{\mathbf{R}}^2$ $\mathbf{A}$

<u>Initialization</u>
RT = 1
Series Graph

	1	2	3	4	5	6	7	8	9	10
1		1	1	1	1	1	1	1	1	1
2			1	1	1	1	1	1	1	1
3				1	1	1	1	1	1	1
4					1	1	1	1	1	1
5						1	1	1	1	1
6							1	1	1	1
7								1	1	1
8									1	1
9										1
10										

	1	2	3	4	5	6	7	8	9	10
1			1	2	3	4	5	6	7	8
2				1	2	3	4	5	6	7
3					1	2	3	4	5	6
4						1	2	3	4	5
5							1	2	3	4
6								1	2	3
7									1	2
8										1
9										
10										

	1	2	3	4	5	6	7	8	9	10
1		1								
2			1							
3				1						
4					1					
5						1				
6							1			
7								1		
8									1	
9										1
10										

<u>Iteration 1</u>
r-delete (5, 6)
RT = 0.96

	1	2	3	4	5	6	7	8	9	10
1		1	1	1	1	1	1	1	1	1
2			1	1	1	1	1	1	1	1
3				1	1	1	1	1	1	1
4					1	1	1	1	1	1
5							1	1	1	1
6							1	1	1	1
7								1	1	1
8									1	1
9										1
10										

	1	2	3	4	5	6	7	8	9	10
1			1	2	3	3	5	6	7	8
2				1	2	2	4	5	6	7
3					1	1	3	4	5	6
4							2	3	4	5
5								1	2	3
6								1	2	3
7									1	2
8										1
9										
10										

	1	2	3	4	5	6	7	8	9	10
1		1								
2			1							
3				1						
4					1	1				
5							1			
6							1			
7								1		
8									1	
9										1
10										

<u>Iteration 2</u>
r-delete (7, 8)
RT = 0.93

	1	2	3	4	5	6	7	8	9	10
1		1	1	1	1	1	1	1	1	1
2			1	1	1	1	1	1	1	1
3				1	1	1	1	1	1	1
4					1	1	1	1	1	1
5							1	1	1	1
6							1	1	1	1
7									1	1
8									1	1
9										1
10										

	1	2	3	4	5	6	7	8	9	10
1			1	2	3	3	5	5	7	8
2				1	2	2	4	4	6	7
3					1	1	3	3	5	6
4							2	2	4	5
5									2	3
6									2	3
7										1
8										1
9										
10										

	1	2	3	4	5	6	7	8	9	10
1		1								
2			1							
3				1						
4					1	1				
5							1	1		
6							1	1		
7									1	
8									1	
9										1
10										

and finally,

<u>Iteration 17</u>
r-delete (2, 9)
RT = 0.39

	1	2	3	4	5	6	7	8	9	10
1		1	1	1	1	1	1	1	1	1
2					1			1		1
3						1		1		1
4						1	1	1	1	1
5								1		1
6								1		1
7									1	1
8										1
9										1
10										

	1	2	3	4	5	6	7	8	9	10
1					1	2	1	5	2	8
2								1		2
3								1		2
4								1	1	4
5										1
6										1
7										1
8										
9										
10										

	1	2	3	4	5	6	7	8	9	10
1		1	1	1						
2					1					
3						1				
4						1	1			
5								1		
6								1		
7									1	
8										1
9										1
10										

This yields the following project network:

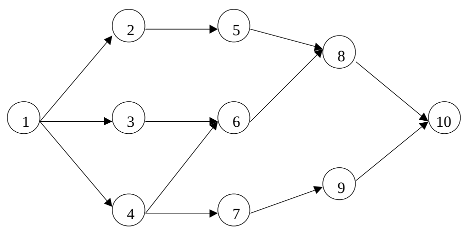


Figure 4-6. Generating a Project Network

Note that at the initialization phase of the network generation procedure when RT = 1, the network reflected in adjacency matrix **A** is precisely a series network. If the procedure continues until RT = 0, the resulting matrices (Figure 4-7) produce a parallel graph.

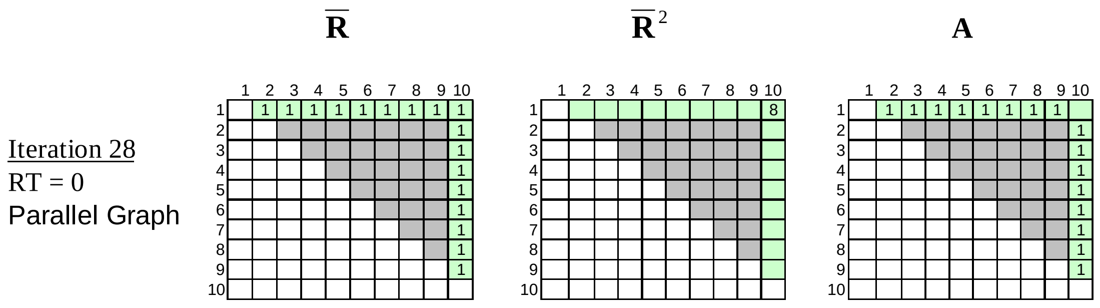


Figure 4-7. Generating a Parallel Project Network

During the generation of a project network, there are a few other feasibility rules which must be observed. The user has previously set lower and upper bounds for the number of start nodes and the number of end nodes, as well as upper bounds for the number of predecessors and successors a node may have (see Table 4-3). Additionally, to preserve the ordering of nodes, the start nodes must begin with node 2 and be consecutively numbered. The end nodes must, also, be consecutively numbered and end with node *n*-1. Therefore, once an arc is r-deleted, a check is made to assure that the above feasibility conditions are satisfied. If they are not, the arc is re-inserted and a new arc chosen for r-deletion. If a feasible network cannot be found within a limited number of trials, the program is halted with an error. The inability to generate a network within the limited number of trials is likely attributable to inconsistent user-defined parameters. For instance, there may be no feasible network for which the specified RT and CNC values can be simultaneously met. The user may then reset the specifications and start over.

To construct a problem with multiple projects, a separate network is generated for each project and then inter-project relationships are introduced. The first inter-project relationships to be introduced are the previously described inter-project lags. The addition of inter-project lags yields a multi-project program with a unique network for each project plus arcs to tie the projects to each other and to the super-source and super-sink nodes. The next step is to add additional arcs to achieve the user-specified values for the program-level RT and CNC.

The procedure for adding program-level arcs is similar to that used for generating project networks with two exceptions. First, inter-project arcs may not only pass from some project p_1 to another project p_2, but may also pass from project p_2 to project p_1. To allow arcs to originate in any project and terminate in any other project, it is possible to initialize the program-level reachability matrix, $\overline{\mathbf{R}}_0$, by arranging the reachability matrices, $\overline{\mathbf{R}}_p$, from each project *p* in block angular form. The *intersections* of these blocks correspond to the reachability of nodes in one project from nodes in another project. Figure 4-8 illustrates what $\overline{\mathbf{R}}_0$ might look like for a program with three projects and a program-level RT of 0.5. The white blocks correspond to the project reachability matrices while the gray blocks are their intersections. Unfortunately, there is no easy way to control the program-level RT with this arrangement. The intersections between project blocks could be initialized with zero entries and then ones added until the desired RT is achieved. This equates, however, to the problem experienced with single projects where feasible additions to the reachability matrix must correspond to arcs in the adjacency matrix. There is little control over RT this way.

$\overline{\mathbf{R}}_0$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2			1	1	1	1	1				1	1	1			1	1		1	1
3				1	1	1	1				1	1	1			1	1		1	1
4						1	1										1		1	1
5						1	1										1		1	1
6							1													1
7																				1
8					1	1	1		1	1	1	1	1			1	1		1	1
9					1	1	1				1	1	1			1	1		1	1
10						1	1				1	1	1				1		1	1
11						1	1						1				1		1	1
12													1							1
13																				1
14				1	1	1	1		1		1	1	1		1	1	1	1	1	1
15				1	1	1	1		1		1	1	1			1	1	1	1	1
16						1	1										1		1	1
17																			1	1
18																			1	1
19																				1
20																				

$\mathbf{A}_0$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1		1						1						1						
2			1																	
3				1	1						1	1				1				
4						1											1			
5						1											1			
6							1													
7																				1
8									1	1										
9					1						1	1				1				
10											1	1								
11						1							1				1			
12													1							
13																				1
14															1					
15				1					1									1		
16						1											1			
17																			1	
18																			1	
19																				1
20																				

Figure 4-8. Example of Multi-Project Program

An alternate way is to initialize the intersections between project blocks with ones and then r-delete node pairs corresponding to arcs in the adjacency matrix. The downside to this method, though, is that the initial reachability matrix (Figure 4-9) is overspecified (*i.e.*, it has an RT = 2.0) and the adjacency matrix is infeasible (*i.e.*, it has non-zero/one entries). Randomly selected node pairs would then need to be r-deleted until $\mathbf{A}_0$ is feasible and then additional node pairs r-deleted until the desired RT is obtained. The problem, again, is that if RT is large (close to one), it may be difficult to bring $\mathbf{A}_0$ into feasibility before the actual RT descends below the desired RT.

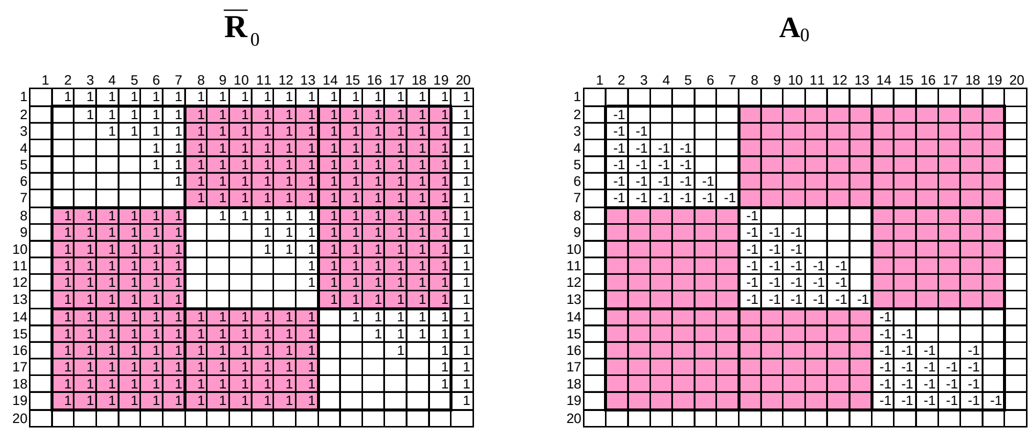


Figure 4-9. Initializing $\overline{\mathbf{R}}_0$ with Ones

To overcome the problem of generating the program-level network structure, recall that a multiple-project program can be viewed as a single super-project. Like any project, the nodes of a super-project can be numbered in a way such that if node *i* precedes node *j*, then *i* < *j*. With this in mind, the procedure used by PAGER is to randomly intermix and renumber the nodes of the projects such that nodes from the same project retain their relative ordering within the super-project and such that the predetermined inter-project lags retain their relative ordering within the super-project. A reachability matrix identical to the ones used for the projects can then be constructed (Figure 4-10) and node pairs r-deleted until the desired program-level RT is obtained.

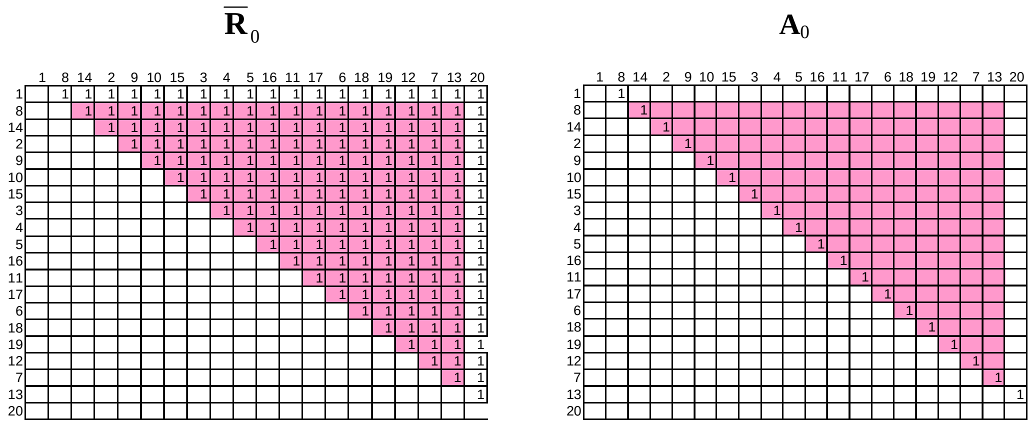


Figure 4-10. Initializing $\overline{\mathbf{R}}_0$ with Mixed Project Nodes

The second exception which makes program-level network generation different from project-level network generation is that the current project networks and inter-project lags must be maintained. Maintenance of the current structures is assured through the *MASK* matrix. The *MASK* matrix is, in essence, the reachability matrix corresponding to the current network structures when viewed as a super-project. The *MASK* matrix is obtained by finding the shortest path from every super-project node to every other super-project node. If the shortest path from one node to another is finite, then the latter node is reachable from the former and the *MASK* matrix receives a unit entry corresponding to that node pair. When randomly selecting a node pair to r-delete, not only must the node pair correspond to an arc in the adjacency matrix, but the node pair must also have a zero entry in the *MASK* matrix.

As with the project networks, node pairs are r-deleted until the program-level RT and CNC are obtained. It is possible for there to exist inconsistencies in the user-defined specifications (Table 4-4). In particular, the inter-project lags and RT may be inconsistent. If the user, for example, has specified a lag of one between each project, the resultant program-level RT will be one. If the user has specified an RT less than one, the r-deletion of node pairs will terminate before the desired RT is obtained. In this case, the program will save the current network with a warning that the specified RT could not be satisfied. (That is, the algorithm is programmed so inter-projects lags take priority over the RT.)

Table 4-4. Input Parameters for Inter-Project Network Generation

Parameter	Definition	Bounds
$L_p^{\min} / L_p^{\max}$	min/max lag between projects *p* and *p*+1, *p* = 1, 2, 3, ..., *P*-1	[0.0, 1.0]
$S_0^{\max}$	maximum inter-project successors per activity	[1, 99]
$P_0^{\max}$	maximum inter-project predecessors per activity	[1, 99]
LF_0	fraction of inter-project arcs which denote generalized prec	[0.0, 1.0]
$LL_0^{\min} / LL_0^{\max}$	lower/upper bounds on the minimum inter-project lag times	[-99, 99]
$LU_0^{\min} / LU_0^{\max}$	lower/upper bounds on the maximum inter-project lag times	[-99, 99]
CNC_0	program-level coeff. of network complexity (arcs per node)	[0, 999]
TH_0	program-level Thesen Restrictiveness measure	[0.0, 1.0]

Once the project networks and the program-level network have been generated, a fraction of the project-level and program-level arcs are randomly chosen to be *converted* into generalized precedence relationships. The user has already specified (through the specification file) the fraction of project-level and program-level arcs which will be converted. The randomly chosen subset of arcs are added to the *LAG* matrix and are given minimum and maximum lag times. The minimum and maximum lag time, *LL* and *LU*, respectively, for each lag relationship are determined using the following equations:

$$LL = round\left(rnd\left(LL_p^{\min}, LL_p^{\max}\right)\right)$$

$$LU = round\left(rnd\left(LU_p^{\min}, LU_p^{\max}\right)\right).$$

After the conversion of a subset of arcs to lag relationships is complete, the remaining arcs are added to the matrices of activity successors (*SUCC*) and predecessors (*PRED*). If the lag conversion leaves an activity without a successor, an arc is added from the activity to the sink node of the respective project. This assures that the activity must be completed before the project is completed. Similarly, if an activity is left without a predecessor, an arc is added from the source node to the activity to assure that the activity does not start before the project does.

Next, the program horizon is calculated. This is done simply by adding the duration of the longest-duration mode of each activity. The program horizon represents the minimum amount of time required to complete the program if resources are constrained to a point where only one activity can be scheduled at a time and in its longest-duration mode.

Early and late start times of each activity are also calculated in the network portion of PAGER. Early and late start times are calculated using the shortest-duration modes of each activity as explained in Chapter III. The Generalized Critical Path Method (GCPM), introduced in Chapter III, is used to determine the early and late start times with modifications to account for generalized precedence. The GCPM algorithm, as outlined in Chapter III, is repeated below.

<u>Generalized Critical Path Method (GCPM)</u>

1. Set the early start time of each activity equal to the release date of the project of which it is a member.

2. For each activity *i*, in numerical order, change its early start time to the greatest of the following:

    a. its current early start time,
    b. the early start time plus duration of each of its standard predecessors, and

   c. the early start time of activity *j* plus minimum time lag between activity *j* and activity *i,* for each activity *j* which is a generalized predecessor of activity *i.*

3. If the early start time of any activity changed at Step 2, repeat Step 2.

4. For each activity *i,* in numerical order, check each activity for which activity *i* is a generalized predecessor. If the early start time of any generalized *successor* of activity *i* is greater than the early start time of activity *i* plus the maximum time lag, change the early start time of activity *i* to the greatest of the early start time minus maximum time lag of each generalized successor of activity *i.*

5. If the early start time of any activity changed at Step 4, repeat Step 2. If not, the early start time of each activity has been found.

6. Set the late start time of each activity equal to the program horizon minus its duration.

7. For each activity *i,* in reverse numerical order, change its late start time to the least of the following:

   a. its current late start time,
   b. the late start time of each of its standard successors minus the duration of activity *i,*
   c. the late start time of each activity generalized successor of activity *i* minus its minimum time lag from activity *i.*

8. If the late start time of any activity changed at Step 7, repeat Step 7.

9. For each activity *i,* in reverse numerical order, check each activity which is a generalized predecessor of activity *i.* If the late start time of activity *i* is greater than the late start time of any generalized predecessor plus its maximum lag time, change the late start time of activity *i* to the least of the late start times plus maximum time lag of each generalized predecessor of activity *i.*

10. If the late start time of any activity changed at Step 9, repeat Step 7. If not, the late start time of each activity has been found.

While the GCPM is, in principle, fairly straightforward, implementation of the algorithm is considerably more complex. To understand the implementation, consider, first, Definitions 4-16 and 4-17.

As an example, suppose that Activities 2 and 3 (in some project) have a generalized precedence relationship and that Activities 3 and 5 (in the same project) also have a generalized precedence relationship. Using the PAGER convention, the lower numbered activity is said to be the generalized predecessor and the higher numbered activity is said to be the generalized successor. Thus, Activity 2 has one generalized successor, Activity 3. Activity 3 also has one generalized successor, Activity 5. If the set, N_i, contains the explicit generalized successors of activity i, then the problem statement would specify the following sets: N_2 = {3} and N_3 = {5}.

Definition 4-16. Explicit Generalized Precedence

Explicit Generalized Precedence is used to describe a generalized precedence relationship *explicitly specified* in the problem statement. The set, N_i, contains the *explicit generalized successors* of activity *i*.

Definition 4-17. Implied Generalized Precedence

Implied generalized precedence is used to describe a generalized precedence relationship which is either explicitly or *not* explicitly specified in the problem statement. The set, N_i^*, contains the *implicit generalized successors* of activity *i*.

Continuing the example above, the problem statement explicitly specified N_2 = {3} and N_3 = {5}. The PAGER convention specifying the higher numbered activity as the generalized successor of the lower numbered activity is used solely to avoid defining generalized precedence relationships twice in the problem statement. It should be recognized, though, that if Activity 3 is a generalized successor of Activity 2, then Activity 2 is also a generalized successor of Activity 3. Of course, the minimal and maximal time lags are different, but they, too, are related. The following relationships hold for any two activities (*i* and *j*) with a generalized precedence relationship:

$$j ? N_i \quad ? \quad i ? N_j^*$$

$$?_{ji}^{\min} ? ??_{ij}^{\max}$$

$$?_{ji}^{\max} ? ??_{ij}^{\min}$$

In the case of Activities 2 and 3, suppose $?_{23}^{\min} ? 2$ and $?_{23}^{\max} ? 5$. That is, if Activity 2 starts at time *t*, then Activity 3 must start in the interval [*t*+2, *t*+5]. This is equivalent, though, to saying that if Activity 3 starts at time *s*, then Activity 2 must start in the interval [*s*-5, *s*-2]. In other words, $?_{32}^{\min} ? ?5$ and $?_{32}^{\max} ? ?2$. Therefore, while the problem statement (using the PAGER convention) would explicitly specify that N_2 = {3}, it is implied that $2 ? N_3^*$.

Though the discussion of explicit and implicit generalized precedences may appear academic, it is important to the GCPM to identify all generalized precedence relationships. To account for the unspecified relationships, set N_i^* is defined as the union of N_i (the *explicit generalized precedence relationships* of activity *i*) and the *implicit generalized precedence relationships* of activity *i*. It has been discussed that if $N_2 = \{3\}$, then $2 ? N_3^*$. It also holds that if $N_3 = \{5\}$, then $3 ? N_5^*$. Somewhat less obvious is that there is also a generalized precedence relationship between activities 2 and 5, where $2 ? N_5^*$ and $5 ? N_2^*$. Once all of the relationships have been identified, the following sets are defined: $N_2^* ? ?3, 5?$, $N_3^* ? ?2, 5?$, and $N_5^* ? ?2, 3?$. The first task of the GCPM is to define all of the implicit generalized precedence sets.

<u>Notation for Generalized Critical Path Method</u>

Activity Sets:

A^E	=	the set of activities which are eligible for labeling and have no generalized precedence relationship
A^L	=	the set of activities which are eligible for labeling and have a generalized precedence relationship
A^S	=	the set of activities which have been labeled
A^1	=	a set of activities where each activity is a generalized predecessor of every other activity in the set
O_i	=	the set of activities which precede activity *i*
S_i	=	the set of activities which succeed activity *i*
N_i	=	the set of explicit generalized successors of activity *i*
N_i^*	=	the set of implicit generalized successors of activity *i*

Time-Related Parameters:

?	=	the program release date
D	=	the program planning horizon
e_i	=	the early start time of activity $p(i)$
l_i	=	the late start time of activity $p(i)$
d_{im}	=	the duration of activity *i* in mode *m*

$?_{ij}^{\min}$ = the minimal start-start lag time between activities *i* and *j*

$?_{ij}^{\max}$ = the maximal start-start lag time between activities *i* and *j*

<u>Generalized Critical Path Method</u>

Step 1 Set *i* = 0.

Step 2 Set *i* = *i* + 1. If $i ? J$, go to Step 6. [For each activity *i* = 1 to *J*, do the following...]

Step 3 If $N_i ? ?$, go to Step 2. [If activity *i* has no explicit generalized successors, proceed to the next activity.]

Step 4 Let $N_i^* ? N_i$. Set *j* = 1. [Activity *i* has at least one explicit generalized successor.]

Step 5 Let $N_i^* ? N_i^* \quad N_{?N_i^*?_j} \setminus ?i?$, where $?N_i^*?_j$ is the *j*-th element of set N_i^*. Set $?_{ik}^{\min} ? ?_{i?N_i^*?_j}^{\min} ? ?_{?N_i^*?_j k}^{\min}$ and $?_{ik}^{\max} ? ?_{i?N_i^*?_j}^{\min} ? ?_{?N_i^*?_j k}^{\max}$ for each $k ? ?N_i^*?_j$. Set *j* = *j* + 1. If $?N_i^*?_j ? ?$, go to Step 2. Otherwise, repeat Step 5.

Step 6 Set the early start time, e_i, of each activity equal to the project release date, $?$. That is, $e_i ? \begin{array}{l} ?? \quad \text{if } d_{i1} ? 0 \\ ? \\ ?? ?1 \quad \text{if } d_{i1} ? 0 \end{array}$, for *i* = 1, 2, 3, …, *J*.

Step 7 Let $A^S = ?$, $A^E = \{1\}$, and $A^L = ?$.

Step 8 Select the lowest indexed activity, say activity *i*, where
$i ? A^E \quad ?A^L \big| j ? A^L, ?j ? N_i^* ?$.

Step 9 For each activity $j ? ?i? \quad N_i^*$, set the early start time of activity j to the greater of its current early start time and the early start time plus duration of each of its predecessors. That is, $e_j ? \begin{cases} \max ?e_j, \max\limits_{\substack{k?O_j \\ d_{k1}?0}} ?e_k ? d_{k1}?, \max\limits_{\substack{k?O_j \\ d_{k1}?0}} ?e_k ? 1?? & \text{if } d_{j1} ? 0 \\ \max ?e_j, \max\limits_{\substack{k?O_j \\ d_{k1}?0}} ?e_k ? d_{k1} ? 1?, \max\limits_{\substack{k?O_j \\ d_{k1}?0}} ?e_k ?? & \text{if } d_{j1} ? 0 \end{cases}$.

Step 10 If $i ? A^E$, let $A^S ? A^S \quad ?i?$, $A^E ? ?A^E \setminus ?i?? \quad ?j \,|\, j ? S_i, O_j ? A^S, N_j^* ? ??$, and $A^L ? A^L \quad ?j \,|\, j ? S_i, O_j ? A^S, N_j ? ??$. If $?A^E \quad A^L? ? ?$, go to Step 8.

Step 11 Given $i ? ?A^L \,|\, j ? A^L, \; ?j ? N_i^*?$, set $A^1 ? ?i? \quad N_i^*$.

Step 12 For each $i ? A^1$, in turn, set

$$e_i ? \begin{cases} \max ?e_i, \max\limits_{\substack{j?N_i^* \\ d_{j1}?0}} ?e_j ? ?_{ij}^{\max}?, \max\limits_{\substack{j?N_i^* \\ d_{j1}?0}} ?e_j ? ?_{ij}^{\max} ? 1?? & \text{if } d_{i1} ? 0 \\ \max ?e_i, \max\limits_{\substack{j?N_i^* \\ d_{j1}?0}} ?e_j ? ?_{ij}^{\max} ? 1?, \max\limits_{\substack{j?N_i^* \\ d_{j1}?0}} ?e_j ? ?_{ij}^{\max}?? & \text{if } d_{i1} ? 0 \end{cases} \quad \text{and}$$

$$e_j ? \begin{cases} \max ?e_j, \max\limits_{d_{i1}?0} ?e_i ? ?_{ij}^{\min}?, \max\limits_{d_{i1}?0} ?e_i ? ?_{ij}^{\min} ? 1?? & \text{if } d_{j1} ? 0 \\ \max ?e_j, \max\limits_{d_{i1}?0} ?e_i ? ?_{ij}^{\min} ? 1?, \max\limits_{d_{i1}?0} ?e_i ? ?_{ij}^{\min}?? & \text{if } d_{j1} ? 0 \end{cases}, \quad ?j ? N_i^*.$$

Step 13 Let $A^S ? A^S \quad A^1$, $A^E ? A^E \quad ?j \,|\, j ? S_i \text{ for } i ? A^1, O_j ? A^S, N_j^* ? ??$, and $A^L ? ?A^L \setminus A^1? \quad ?j \,|\, j ? S_i \text{ for } i ? A^1, O_j ? A^S, N_j ? ??$. If $?A^E \quad A^L? ? ?$, go to Step 8.

Step 14 Renumber activities from earliest start time to latest start time (break ties by index).

Step 15 Set the late start time, l_i, of each activity equal to one time unit more that the project horizon, D, less its duration. That is, $l_i = \begin{cases} D - d_{i1} + 1 & \text{if } d_{i1} > 0 \\ D & \text{if } d_{i1} = 0 \end{cases}$, for i = 1, 2, 3, …, J.

Step 16 Let $A^S = \emptyset$, $A^E = \{J\}$, and $A^L = \emptyset$.

Step 17 Select the highest indexed activity, say activity i, where $i \in A^E \cup \{A^L \mid j \in A^L, \forall j \in N_i^*\}$.

Step 18 For each activity $j \in \{i\} \cup N_i^*$, set the late start time of activity j to the lesser of its current late start time and the late start time of each of its successors less the duration of activity j. That is,

$$l_j = \begin{cases} \min\left\{ l_j, \min\limits_{\substack{k \in S_j \\ d_{k1} > 0}} \{l_k - d_{j1}\}, \min\limits_{\substack{k \in S_j \\ d_{k1} = 0}} \{l_k - d_{j1} + 1\} \right\} & \text{if } d_{j1} > 0 \\ \min\left\{ l_j, \min\limits_{\substack{k \in S_j \\ d_{k1} > 0}} \{l_k - 1\}, \min\limits_{\substack{k \in S_j \\ d_{k1} = 0}} \{l_k\} \right\} & \text{if } d_{j1} = 0 \end{cases}.$$

Step 19 If $i \in A^E$, let $A^S = A^S \cup \{i\}$, $A^E = \{A^E \setminus \{i\}\} \cup \{j \mid j \in O_i, S_j \subseteq A^S, N_j^* \neq \emptyset\}$, and $A^L = A^L \cup \{j \mid j \in O_i, S_j \subseteq A^S, N_j = \emptyset\}$. If $\{A^E \cup A^L\} \neq \emptyset$, go to Step 17.

Step 20 Given $i \in \{A^L \mid j \in A^L, \forall j \in N_i^*\}$, set $A^1 = \{i\} \cup N_i^*$.

Step 21 For each $i \in A^1$, in turn, set

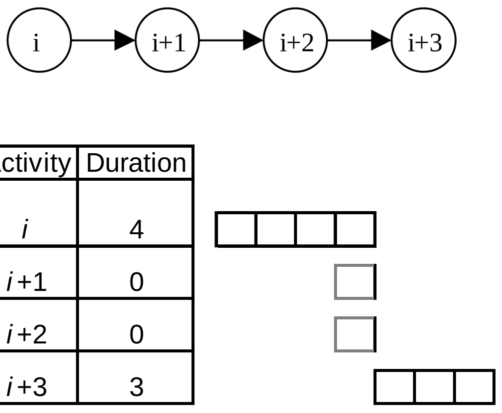
$$l_i = \begin{cases} \min\left\{ l_i, \min\limits_{\substack{j \in N_i^* \\ d_{j1} > 0}} \{l_j - \delta_{ij}^{\min}\}, \min\limits_{\substack{j \in N_i^* \\ d_{j1} = 0}} \{l_j - \delta_{ij}^{\min} - 1\} \right\} & \text{if } d_{i1} > 0 \\ \min\left\{ l_i, \min\limits_{\substack{j \in N_i^* \\ d_{j1} > 0}} \{l_j - \delta_{ij}^{\min} + 1\}, \min\limits_{\substack{j \in N_i^* \\ d_{j1} = 0}} \{l_j - \delta_{ij}^{\min}\} \right\} & \text{if } d_{i1} = 0 \end{cases}$$ and

$$l_j = \begin{cases} \min\left\{ l_j, \min\limits_{d_{i1} > 0} \{l_i + \delta_{ij}^{\max}\}, \min\limits_{d_{i1} = 0} \{l_i + \delta_{ij}^{\max} + 1\} \right\} & \text{if } d_{j1} > 0 \\ \min\left\{ l_j, \min\limits_{d_{i1} > 0} \{l_i + \delta_{ij}^{\max} - 1\}, \min\limits_{d_{i1} = 0} \{l_i + \delta_{ij}^{\max}\} \right\} & \text{if } d_{j1} = 0 \end{cases}, \quad \forall j \in N_i^*.$$

Step 22 Let $A^S ? A^S \quad A^1,\ A^E ? A^E \quad ?j \big| j ? O_i$ for $i ? A^1, S_j ? A^S, N_j^* ? ? ?$, and $A^L ? ?A^L \setminus A^1? \quad ?j \big| j ? O_i$ for $i ? A^1, S_j ? A^S, N_j ? ? ?$. If $?A^E \quad A^L? ? ?$, go to Step 17.

Steps 1 through 5 determine all of the implicit generalized precedence relationships of each activity. As the algorithm *labels* activities, it does so by labeling activities without any generalized precedence relationships one at a time (as in traditional CPM), and by labeling activities with generalized precedence relationships as a set. The algorithm, therefore, *collects* the activities with generalized precedence relationships and holds them until all such related activities are eligible for labeling based on having all of their predecessors labeled. When these sets of activities are labeled, all of the interrelationships must be known; hence, Steps 1 to 5.

The rules for scheduling zero-duration activities differ slightly from those with positive duration. If the predecessor of a zero-duration activity finishes at time t^0, then the earliest the zero-duration activity may start is t^0 rather than $t^0 + 1$. The relationship of these activities may be seen in Figure 4-11, where the precedence relationships of activities i through i+3 are shown and where activities i+1 and i+2 have zero duration.



Activity	Duration
i	4
i +1	0
i +2	0
i +3	3

Figure 4-11. Precedence-Feasible Early Start Times of Zero-Duration Activities

Steps 6 through 13 perform the early start time labeling. Each activity is first labeled to start at the project release date (Step 6), then adjusted to start as soon as all of its predecessors have finished (Step 9). For activities with no generalized precedence relationships, Step 10 updates the activity sets and labeling recommences with the next eligible activity in Step 8. For activities with generalized precedence relationships, Step 11 forms the set of all activities which are, in essence, being labeled simultaneously. In Step 12, the early start time of each activity *i* is delayed (if necessary) to assure that none of the maximum lags associated with that activity will be violated, and then all other activities in the set are delayed (if necessary) to assure that none of the minimum lags associated with activity *i* are violated. Step 13, then, updates the activity sets and labeling recommences at Step 8.

Once early start times have been determined, Step 14 renumbers the activities from earliest start time to latest start time.

Steps 15 through 22 determine the late start times of each activity by essentially reversing the early start time labeling process. Late start times are bound by the project horizon, in the same way early start times are bound by the project release date. With both early and late start times determined, the feasible start time windows of each activity are known.

Finally, program and project due dates are calculated using the due date factors found during basic data generation and the following equation:

$$DD_p \; ? \; round\left(ES_{pJ} \; ? \; ?_p * \left(LS_{pJ} \; ? \; ES_{pJ}\right)\right), \; p = 0, 1, 2, ..., P$$

where DD_0 is the program due date.

Step 4 - Resource Data Generation. The generation of data for regular renewable and nonrenewable resources is nearly identical to that detailed by Kolisch *et al.* (1992, 1995) for ProGen. The primary difference is that in multiple-project problems, ProGen generates only program-level resources (*i.e.*, project-specific resources are not considered). The procedure employed by PAGER generates project-level and program-level resources using the input data listed in Table 4-5.

The generation of resource data begins by identifying the number of renewable and nonrenewable resources for each project and for the program using the following equation:

$$|?|_p \; ? \; round\left(rnd\left(|?|_p^{\min}, |?|_p^{\max}\right)\right), \; p = 0, 1, 2, ..., P\text{-}1,$$

where $p = 0$ refers to the program-level data.

The demand for resources is generated next by identifying which resources are demanded by each activity-node combination and how much of those resources is demanded. This procedure, which uses parameters $Q^{\min}_{p?} / Q^{\max}_{p?}$, $RF_{p?}$, $r^{\min}_{p?} / r^{\max}_{p?}$, $\mathrm{P}_{p?}(F ? 1)$, $\mathrm{P}_{p?}(F ? 2)$, and $?_{RF}$, is identical to ProGen's (see Kolisch *et al.*, 1992 & 1995).

Resource availability is the last resource data to be generated. For regularly available resources, the procedure used in ProGen is employed where a minimal demand, $K^{\min}_{pq}$, and a maximal demand, $K^{\max}_{pq}$, are calculated for each resource q in project p. The availability of resource q in project p is a convex combination of the minimal and maximal demands with the resource strength as a scaling parameter. The resource strength is drawn from the uniform distribution $\left[RS^{\min}_{p?}, RS^{\max}_{p?} \right]$ and the resultant resource availability is:

$$K_{pq} ? round\left(K^{\min}_{pq} ? RS_{p?} \left(K^{\max}_{pq} ? K^{\min}_{pq} \right) \right)$$

Table 4-5. Input Parameters for Resource Data Generation

Parameter	Definition	Bounds
$\lvert ? \rvert^{\min}_{p} / \lvert ? \rvert^{\max}_{p}$	min/max number of resources of type ? for project *p*	[0, 10]
$Q^{\min}_{p?} / Q^{\max}_{p?}$	min/max number of resources of type ? requested per job in project *p*	[0, 99]
$RF_{p?}$	resource factor of resource type ? for project *p*	[0.0, 1.0]
$r^{\min}_{p?} / r^{\max}_{p?}$	min/max resource demand for resource type ? for project *p*	[0, 99]
$RS^{\min}_{p?} / RS^{\max}_{p?}$	min/max resource strength for resource type ? for project *p*	[0.0, 1.0]
$ERS^{\min}_{p?} / ERS^{\max}_{p?}$	min/max expediting resource strength for resource type ? for project *p*	[0.0, 1.0]
$\mathrm{P}_{p?}(F ? 1)$	prob. of duration-constant demands for resource type ? for project *p*	[0.0, 1.0]
$\mathrm{P}_{p?}(F ? 2)$	prob. of duration-nonincreasing demands for resource type ? for project *p*	[0.0, 1.0]
$?_{RF}$	resource factor tolerance	[0.0, 1.0]

For program-level resources, $p = 0$
For project-level resources, $p = 1, 2, ..., P-1$

Evaluation of the minimal and maximal demands is as follows, noting that r^{R}_{pimq} and r^{N}_{pimq} are the respective renewable and nonrenewable resource requirements for resource q when activity i of project p is executed in mode m. For nonrenewable resources, then

$$K^{\min}_{pq} = \sum_{i=2}^{J_p - 1} \min_{m=1}^{M_{pi}} \left\{ r^{N}_{pimq} \right\}$$

and

$$K^{\max}_{pq} = \sum_{i=2}^{J_p - 1} \max_{m=1}^{M_{pi}} \left\{ r^{N}_{pimq} \right\}.$$

For renewable resources,

$$K^{\min}_{pq} = \max_{i=2}^{J_p - 1} \left\{ \min_{m=1}^{M_{pi}} \left\{ r^{R}_{pimq} \right\} \right\}$$

and the maximal demand is the peak demand of the precedence and lag preserving the earliest start schedule. With each activity i executed in its lowest indexed mode employing maximal per-period demand, that is, where $r^{*}_{piq} = \max_{m=1}^{M_{pi}} \left\{ r^{R}_{pimq} \right\}$ and $m^{*}_{piq} = \min_{m=1}^{M_{pi}} \left\{ m_{pi} \middle| r^{R}_{pimq} = r^{*}_{piq} \right\}$, the resource-dependent early start schedule is calculated with corresponding earliest start times, ES^{q}_{pi}, and completion times, CT^{q}_{pi}. The peak per-period demand is then:

$$K^{\max}_{pq} = \max_{t=1}^{horizon} \left\{ \sum_{t = ES^{q}_{pi}+1}^{CT^{q}_{pi}} \sum_{i=2}^{J_p - 1} r^{R}_{pim^{*}_{pik}q} \right\}.$$

Note that when the resource strength is zero, there is just enough resource to complete the program in the program horizon. When the resource strength is one, there is enough resource to complete the program in its unconstrained CPM time.

To calculate the availability of expediting resources, the same minimal and maximal demands are used. However, the sum of resource strengths for regular and expediting resources should not exceed one, so that expediting resource availability is calculated as:

$$EK_{pq} = round\left(ERS_{p} \cdot \left(K^{\max}_{pq} - K^{\min}_{pq} \right) \right).$$

Total resource availability is then

$$K_{pq} ? EK_{pq} ? round\left(K_{pq}^{\min} ? RS_{p?}\left(K_{pq}^{\max} ? K_{pq}^{\min}\right) ? round\left(ERS_{p?}\left(K_{pq}^{\max} ? K_{pq}^{\min}\right)\right)\right)$$
$$? K_{pq}^{\min} ? \left(round\left(RS_{p?}\left(K_{pq}^{\max} ? K_{pq}^{\min}\right)\right) ? round\left(ERS_{p?}\left(K_{pq}^{\max} ? K_{pq}^{\min}\right)\right)\right)$$

The costs associated with expediting resources are generated in the next step.

Step 5 - Cost Data Generation. There are three types of cost data generated depending on the desired objective function: program/project completion penalties, mode costs, and expediting resource costs. The input data required to generate this data is listed in Table 4-6.

If the objective function of the program scheduling problem includes the minimization of program and project completion costs, program and project completion penalties are assessed starting at one period past their respective due dates. That is, there is no penalty if the program/project ends on or before its due date. If the program/project is one period late, the program/project penalty base value is assessed. For each additional period that the program/project is late, the penalty is increased by the penalty increment.

The program-level base penalty, PEN_{00}, and penalty increment, PEN_{01}, are specified by the user in the specification file. All other costs are related to the program penalty base value and increment. For instance, the completion penalty base value of a project *p*, PEN_{p0}, is a fraction of the program penalty base value and is generated using the following equation:

$$PEN_{p0} ? round\left(PEN_{00} * rnd\left[PEN_{p0}^{\min}, PEN_{p0}^{\min}\right]\right),$$

while the penalty increment, PEN_{p1}, is a fraction of the program penalty increment and is generated using the following equation:

$$PEN_{p1} ? round\left(PEN_{01} * rnd\left[PEN_{p1}^{\min}, PEN_{p1}^{\min}\right]\right).$$

The importance of completing project *p* is, therefore, tied directly to the importance of completing the program. If the project's penalty increment is half of the program's penalty increment, then a one period delay in the completion of the project is half as costly as a one day delay in the completion of the program.

Similarly, the costs of activity modes and start times and of expediting resources are tied to the program completion penalty base value and increment. This provides the user a way to easily reflect the relative cost of scheduling decisions to the cost of other decisions.

A final note is that the cost of activities can be time-increasing, time-constant, or time-decreasing. Permitting activity costs to change over time allows the user to design problems with positive and negative cash flows. Again, activity costs have a base value related to the program penalty base value and a per-period increment (positive or negative) related to the program penalty increment.

Table 4-6. Input Parameters for Cost Data Generation

Parameter	Definition	Bounds
PEN_{00}	program base penalty	[0, 9999]
PEN_{01}	program penalty increment	[0, 9999]
$PEN_{p0}^{\min} / PEN_{p0}^{\max}$	min/max project base penalty *, $p = 1, 2, \ldots, P$	[0.0, 99.0]
$PEN_{p1}^{\min} / PEN_{p1}^{\max}$	min/max project penalty increment **, $p = 1, 2, \ldots, P$	[0.0, 99.0]
$MC_{p0}^{\min} / MC_{p0}^{\max}$	min/max base mode cost *, $p = 1, 2, \ldots, P$	[0.0, 99.0]
$MC_{p1}^{\min} / MC_{p1}^{\max}$	min/max mode cost increment **, $p = 1, 2, \ldots, P$	[0.0, 99.0]
P_p ?G ? 1?	probability of time-increasing activity costs, $p = 1, 2, \ldots, P$	[0.0, 1.0]
P_p ?G ? 2?	probability of time-decreasing activity costs, $p = 1, 2, \ldots, P$	[0.0, 1.0]
$ERC_{p}^{\min} / ERC_{p}^{\max}$	min/max expediting renewable resource base cost *, $p = 0, 1, \ldots, P$	[0.0, 99.0]
$ENC_{p}^{\min} / ENC_{p}^{\max}$	min/max expediting nonrenewable resource base cost*, $p = 0, \ldots, P$	[0.0, 99.0]

* denotes that these values are fractions of the program base penalty cost
** denotes that these values are fractions of the program base penalty increment

<u>Step 6 - Problem Output</u>. Once a problem instance has been generated, it may be output in PAGER, ProGen, or MPS formats. The PAGER and MPS formats can reflect all of the features that PAGER is designed to produce. The ProGen format, on the other hand, is not designed to reflect generalized precedence relationships, expediting resources, or mode costs, and so ProGen

format is unavailable if any of these features are invoked. A sample PAGER output file is included as Appendix C.

PAGER Implementation

PAGER is programmed in FORTRAN 77 with a number of FORTRAN 90 extensions. It has been implemented on an IBM-compatible computer with a Pentium 750 MHz processor and 256 MB of RAM, running Windows NT. This machine was used to generate the test problems used in Chapters V and VI. Figures 4-12 through 4-13 report the distribution of times required to generate a total of 10,521 test problems. Problems ranged in size from single projects with 5 activities to four-project programs with 50 total activities. Overall, PAGER required an average of 0.95 seconds to generate a problem, with a minimum generation time under 0.01 seconds, a maximum time of 155.67 seconds, and a variance of 19.90 seconds.

Figure 4-12 is a Box and Whiskers plot of problem generation times. The whiskers show the minimum and maximum generation times, while the box shows the mean plus / minus two standard deviations. Generation times are shown according to the number of activities in the problem. Since the maximum generation time for problems with 50 activities is large compared to other problem sizes, the box and whiskers for smaller problem sizes are difficult to see. Therefore, the Box and Whiskers plot is repeated in Figure 4-13 with the 50-activity problems removed and the *y*-axis time scale decreased.

Problems with 10 activities and with 50 activities both have an outlier. One problem with 10 activities required 8.43 seconds, compared to the next longest generation time of 1.11 seconds. One problem with 50 activities took 155.67 seconds, compared to the next longest generation time of 68.51 seconds.

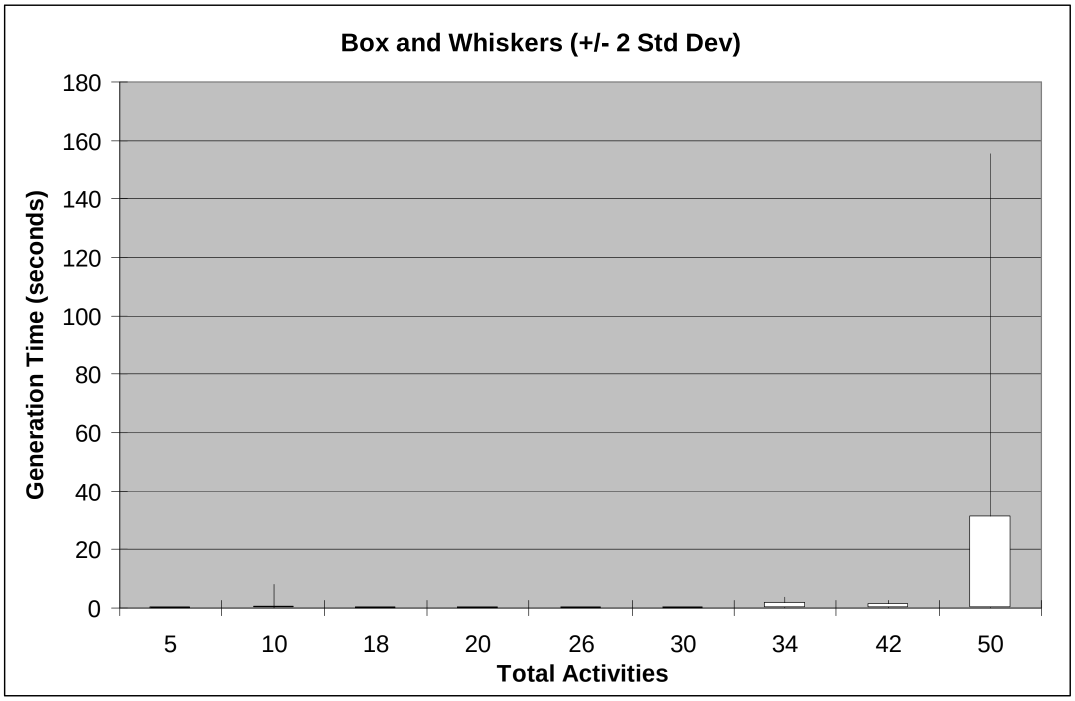


Figure 4-12. Distribution of Generation Times (5 to 50 Activities)

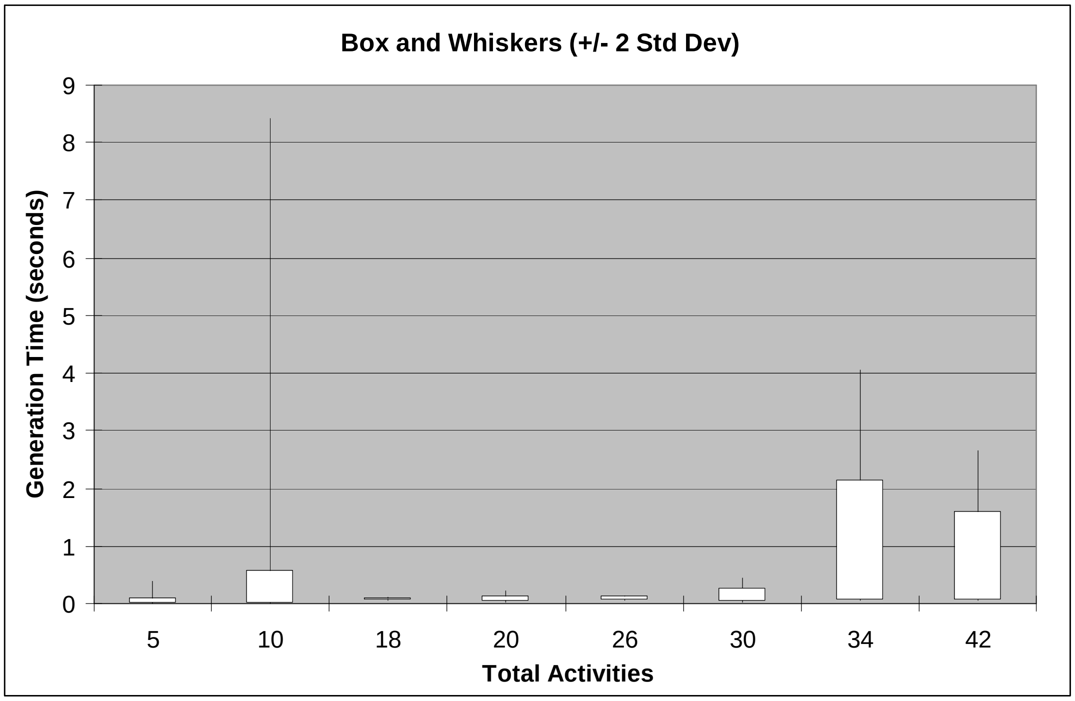


Figure 4-13. Distribution of Generation Times (5 to 42 Activities)

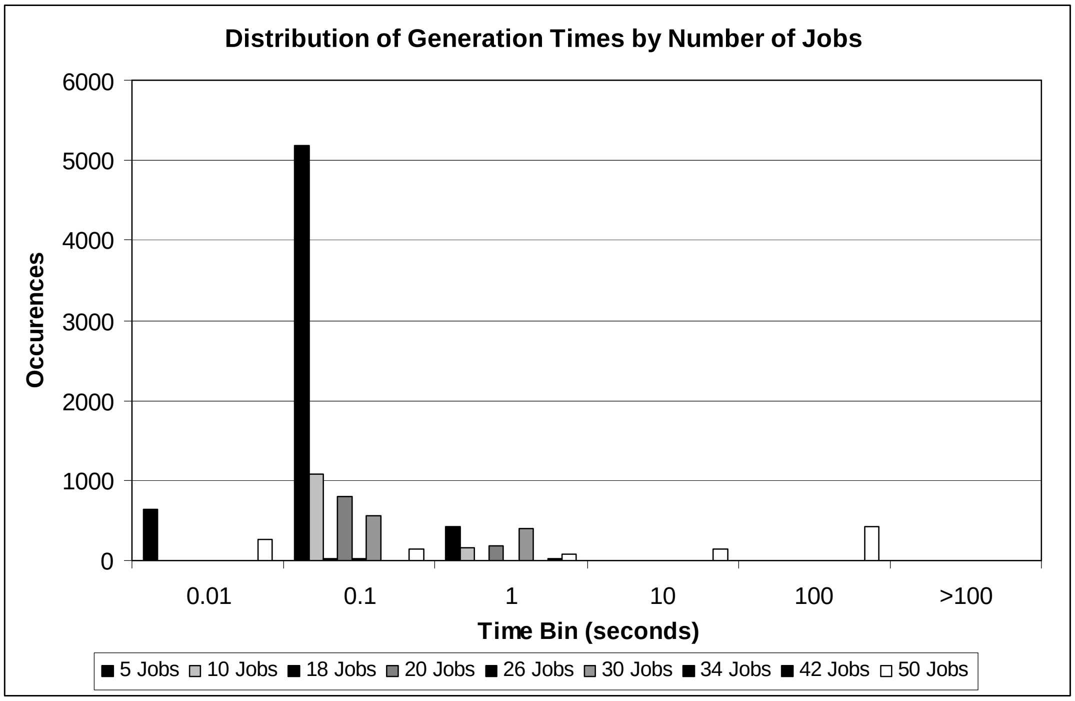


Figure 4-14. Distribution of Generation Times by Number of Jobs

Figure 4-14 presents the generation time data by time bin. The bars on the chart represent the number of problems generated within the bounds of the respective time bin. Different shaded bars are used to differentiate problems of different sizes (*i.e.*, number of activities). The vast majority of problems (95%) required no more than one second of generation time.

Summary and Conclusions

Table 4-7 summarizes the key features of PAGER and the other generators discussed above. All of the generators can generate single-project, single-mode problems with renewable resources. Differences between the generators include their multi-project capabilities, the types of resources generated, the measures of network complexity used, and the measures of resource availability used. All of the generators output problems in their own specific format. In addition, some generators are capable of output in formats used in other test sets (Patterson and ProGen) or, in the case of PAGER, in MPS format.

PAGER is the first problem generator to directly exploit the reachability matrix of a network to generate problem networks with precisely controlled values of RT. It is also the first to simultaneously use two measures of network complexity, RT and CNC, in the network generation process. PAGER fills the need to generate multi-project problems with project-specific networks and resources, interrelationships between projects, and program-level network structure and resources.

Table 4-7. Key Features of Problem Generators

	DDH	ProGen	..../max	..../? *x*	DAGEN	PAGER
Multi-Project Problems		x		x		x
w/Program-Level Resources		x		x		x
w/Project-Specific Resources						x
w/Time-Related Projects						x
w/Unique Project Networks						x
Multiple Modes		x	x	x		x
Minimum Time Lags			x	x		x
Maximum Time Lags			x			x
Changeover Times				x		
Mode and Set of Mode Identity				x		
Forbidden Periods				x		
Mode Costs (Time-Constant)				x		x
Time-Increasing/-Decreasing						x
Expediting Resource Costs						x
Network Complexity Measures:						
Coefficient of Network Complexity		x				x
Thesen's Restrictiveness			x	x		x
Direct Use of Thesen Restrictiveness						x
Complexity Index					x	
Choice of Measures						x
Simultaneous Measures						x
Renewable Resources	x	x	x	x	x	x
Nonrenewable Resources		x	x	x	x	x
Partially-Renewable Resources				x		
Expediting Resources						x
Resource Availability Measures:						
Resource Factor		x	x	x		x
Resource Strength		x	x	x		x
Parameter(s) Randomly Drawn			x			x
Output:						
Patterson Format			x			
ProGen Format		x	x			x
PAGER Format						x
MPS Format						x

V. Single Project Scheduling

Overview

This chapter presents an algorithm for solving the Multi-Modal, Resource-Constrained Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCPSP-GPR/EXP). The MRCPSP-GPR/EXP is the single-project, special case of the Multi-Modal, Resource-Constrained, *Multi*-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRC*M*PSP-GPR/EXP). While Chapter III highlighted the nature of the multi-project MRC*M*PSP-GPR/EXP, the ability to solve either single-project or multi-project problems is entirely dependent on the availability of an algorithm for solving single-project problems. More specifically, every instance of the MRC*M*PSP-GPR/EXP falls into one of the following three categories:

1. The problem is a *true* single-project MRCPSP-GPR/EXP. That is, an instance which represents a real-world problem which is a single project.
2. The problem is a multi-project MRC*M*PSP-GPR/EXP which is treated and scheduled as a single *super project*.
3. The problem is a *subproblem* of a larger multi-project instance of the MRC*M*PSP-GPR/EXP. While a multi-project program may be scheduled as a *super project* as in Category 2 above, it may also be scheduled using the decomposition approach presented in the next chapter. When scheduled using the decomposition approach, the decomposed subproblems must still be solved - as single-project instances.

In any of the preceding categories, solution of the problem starts with the scheduling of a single project.

With the goal of developing an appropriate solver for the single-project MRCPSP-GPR/EXP, this chapter begins with a discussion of solution approaches from the literature and how they relate to this particular problem. Specifically, approaches used for related problems are reviewed for their applicability to the MRCPSP-GPR/EXP, keeping in mind the unique characteristics of the MRCPSP-GPR/EXP, as well as its intended uses.

An approach for solving the MRCPSP-GPR/EXP is then presented, beginning with the development of a basic algorithm and then adding on additional bounding rules designed to increase the speed of the basic algorithm and, consequently, the size of problems which can be solved. Testing of the algorithm is reported, followed by a chapter summary. To aid the reader in following the notation used in this chapter and other chapters, refer to Appendix A for a complete listing of symbols, variables, and parameters.

Approaches from the Literature

The single-project MRCPSP-GPR/EXP shares many characteristics with other scheduling problems discussed in the literature. Most obvious are the finish-start precedence relationships between activities and the dependence of activity completion time and resource use on the activity completion mode. Other characteristics are less common – some of which preclude use of some of the proven solution techniques in the literature. The desire to use the solution algorithm in the decomposition methodology presented in the next chapter also puts constraints on the approach used to solve the MRCPSP-GPR/EXP. In light of these characteristics and constraints, solution approaches from the literature have been evaluated for their applicability to the MRCPSP-GPR/EXP. The intent is to identify proven approaches which may form the basis of an approach for the MRCPSP-GPR/EXP.

Implicit Enumeration by Branch-and-Bound. Among the approaches found in the literature for solving project-scheduling problems, the most efficient are branch-and-bound enumeration algorithms. These algorithms reduce the enumeration tree by searching among *active* schedules only. These algorithms have been presented by Stinson *et al.* (1978), Christofides (1987), and Demeulemeester and Herroelen (1992) for the Resource-Constrained Project Scheduling Problem (RCPSP); by Patterson *et al.* (1989, 1990), Sprecher (1994), Sprecher and Drexl (1996a, 1998), Sprecher *et al.* (1997), and Hartmann and Drexl (1998) for the Multi-Modal, Resource-Constrained Project Scheduling Problem (MRCPSP); by Demeulemeester and Herroelen (1997) for the Generalized, Multi-Modal, Resource-Constrained Project Scheduling Problem (GMRCPSP); by De Reyck and Herroelen (1998a, 1998b) and Herroelen *et al.* (1998) for the Multi-Modal, Resource-Constrained Project Scheduling Problem with Generalized Precedence (MRCPSP-GPR); and by Van Hove (1998) for the Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem (GMRCMPSP). Understanding the efficiency of these algorithms requires some discussion of active scheduling.

Consider, first, Definitions 5-1 through 5-8, provided by Sprecher *et al.* (1995).

Definition 5-1. Schedule (Sprecher *et al.*, 1995: 97)

Consider a project with J activities. Let s_j and m_j be the respective start time and execution mode of activity j. A *schedule,* $\mathbf{S} = (\mathbf{s}, \mathbf{m})$, is a combination of J-tuples, $\mathbf{s} = (s_1,\ldots, s_J)$ and $\mathbf{m} = (m_1, \ldots, m_J)$, which provide the start time and execution mode of each activity j, $j = 1, \ldots, J$.

Definition 5-2. Feasible Schedule (Sprecher *et al.*, 1995: 97)

A *schedule* **S** is called *feasible* if the precedence relations are maintained and the resource constraints are met.

Definition 5-3. Left Shift (Sprecher *et al.*, 1995: 97)

A *left shift* of an activity j, $j = 1, \ldots, J$, is an operation on a feasible schedule **S**, which derives a feasible schedule $\mathbf{S}'$, such that $s'_j ? s_j$ and $s'_i ? s_i$, for i, $i = 1, \ldots, J$, $i ? j$.

In words, left shifting an activity consists of moving the start time of an activity to an earlier time without moving the start time of any other activity and while maintaining feasibility.

Definition 5-4. One-Period Left Shift (Sprecher *et al.*, 1995: 97)

A left shift of an activity j, $j = 1, \ldots, J$, is called a *one-period left shift* if $s_j ? s'_j ? 1$.

Definition 5-5. Local Left Shift (Sprecher *et al.*, 1995: 97)

A *local left shift* of an activity j, $j = 1, \ldots, J$, is a left shift of activity j which is obtainable by one or more successively applied one-period left shifts of activity j.

Sprecher (1994: 97) notes that within a local left shift, each intermediate derived schedule has to be feasible, by definition.

Definition 5-6. Global Left Shift (Sprecher *et al.*, 1995: 97)

A *global left shift* of an activity j, $j = 1, \ldots, J$, is a left shift of activity j which is not obtainable by a local left shift.

Definition 5-7. Semi-Active Schedule (Sprecher *et al.*, 1995: 97)

A *semi-active schedule* is a feasible schedule where none of the activities *j*, *j* = 1, …, *J*, can be locally left shifted.

Definition 5-8. Active Schedule (Sprecher *et al.*, 1995: 98)

An *active schedule* is a feasible schedule where none of the activities *j*, *j* = 1, …, *J*, can be locally or globally left shifted.

The efficiency of algorithms which search over active schedules only hinges on the concept that, under appropriate conditions, a project must have an active schedule which is optimal (Sprecher, 1994). The necessary conditions for this to be true are (1) the project schedule is feasible and (2) the project schedule's objective function is a regular measure of performance. The feasibility of the project schedule is an obvious condition. However, the regular measure of performance condition requires further explanation.

Definition 5-9. Regular Measure of Performance

Consider a scheduling problem with the objective to minimize some measure of schedule fitness, ? . Let **S** = (**s**, **m**) be a feasible schedule for the problem and let ? (**s**, **m**) represent the fitness of schedule **S**. ? is a *regular measure of performance* if ? (**s**, **m**) < ? (**s'**, **m**) implies that s_j ? s_j' for at least one *j*, *j* = 1, …, *J*.

Definition 5-10. Non-Regular Measure of Performance

Consider a scheduling problem with the objective to minimize some measure of schedule fitness, ? . ? ?is a *non-regular measure of performance* if it is not a regular measure of performance.

In simple terms, a regular measure of performance is one where a decrease in the objective function value implies that at least one activity starts earlier in the improved schedule than it does in the competing schedule. The objective of minimizing the project makespan (see Chapter II, Equation (10)) is an example of a regular measure of performance (Sprecher, 1994). Kolisch and Padman (1998: 3) explain that a regular measure of performance is one where "we can compare two schedules for a given problem which differ only in the finish time of one activity and we can state that the schedule which has the smaller finish time for this activity is at least as good as the other schedule, *i.e.*, the former dominates the latter." Consequently, left-shifting the start time of any activity never results in a worse objective value function.

The preceding discussion of active schedules and regular measures of performance is important because of their positive impact on the execution time of so many approaches in the literature. Unfortunately, the MRCPSP-GPR/EXP cannot exploit the concept of active schedules for two key reasons.

First, the objective function of the MRCPSP-GPR/EXP (Chapter III, Equation (20)) is a non-regular measure of performance (Kolisch and Frase, 1996: 139). This is a consequence of the availability of expediting resources and the corresponding objective of minimizing project costs, including the cost of expediting resources. Consider a project, **P**, with no expediting resources and an objective function to minimize makespan. Since **P** has an objective function which is a regular measure of performance, it follows that **P** has an active schedule, **S**, which is optimal. By definition, it is impossible to left shift any activity in **S**, while maintaining precedence and resource feasibility. Suppose, however, that there is some activity *j* which could be left shifted while maintaining precedence feasibility, but which would result in a resource conflict. If expediting resources (at no cost) were now made available which would permit the left shifting of activity *j*, then the current schedule, **S**, is no longer an active schedule. Assuming that left shifting activity *j* alone would result in an active schedule (*i.e.*, addition of the expediting resources did not impact the ability of other activities, including the terminal activity, to be left shifted), then the new schedule, **S′**, would have the same value as the previous schedule, **S**. However, if there is a positive cost associated with the expediting resources which made **S′** feasible and the objective function is expanded to include the cost of expediting resources, then **S′** is, in fact, dominated by **S**. Furthermore, since it was the addition of expediting resources alone that enabled activity *j* to shift left to form active schedule **S′**, schedule **S**, a non-active schedule, remains optimal even for the expanded objective function. Hence, the inclusion of expediting resources makes an objective function a non-regular measure of performance and it is no longer sufficient to search only the active schedules to find an optimal. This result is confirmed by Kolisch and Frase (1996: 139).

Second, the decomposition approach described in Chapter III, for solving the multi-project MRCMPSP-GRP/EXP, requires the generation of the *k*-best solutions of single-projects. Even in the absence of expediting resources (which would make the objective function a regular measure of performance), it is still necessary to enumerate over the non-active schedules. The reason lies in the interdependence of the projects at the program level where projects are temporally related and must also compete for common renewable resources. Suppose that a project **P** has two schedules of *equal value*: $\mathbf{S_1}$, an active schedule, and $\mathbf{S_2}$, a non-active schedule. This is entirely possible whether the objective function is a regular or non-regular measure of performance. If project **P** must now adjudicate the start times of its activities and its resource requirements with other projects at the program-level, project **P** is indifferent to the two schedules, $\mathbf{S_1}$ and $\mathbf{S_2}$, provided there is no program-level cost associated with the two schedules. However, it is possible that schedule $\mathbf{S_2}$, the non-active schedule, is feasible as to the program-level temporal relationships and resource availabilities, while $\mathbf{S_1}$, the active schedule, is not. Therefore, in the development of the set of *k*-best solutions to the single-project problem, all schedules, active and non-active, must be evaluated. This evaluation leads not only to the enumeration of possibly all of the optimal solutions, but also to the enumeration of equal-valued, suboptimal solutions.

It should be noted that Van Hove (1998) develops a similar decomposition approach for the multi-project GMRCMPSP, but one where only active schedules are considered. Because the GMRCMPSP does not include expediting resources, its objective function is a regular measure of performance. From the previous discussion, however, one might suspect that Van Hove's subproblem solver would still need to enumerate non-active schedules. Enumerating non-active schedules is unnecessary, though, because Van Hove assumes that projects are temporally independent. Therefore, Van Hove enumerates the *k*-best active schedules of a project, all of which are, by assumption, temporally and renewable-resource feasible at the program level. The question that remains in Van Hove's approach is which of the *k*-best active schedules adjudicates best with the other projects for nonrenewable resources. Van Hove demonstrates the utility of this approach for the development of Air Tasking Orders (ATOs) in the wartime campaign planning process. The approach, however, is inadequate for the nature of the MRCPSP-GPR/EXP.

Having eliminated the active-schedule enumeration schemes, there are still two approaches to evaluate for their applicability to the MRCPSP-GPR/EXP: zero-one programming and an implicit enumeration scheme by Talbot (1982).

Zero-One Programming. Some of the earliest attempts to solve the RCPSP were based in zero-one programming. These attempts differed not so much in the procedure used to solve the zero-one program, but in the way the zero-one program was formulated (*e.g.*, Bowman, 1959; Pritsker *et al*., 1969). Chapter III presents a complete zero-one formulation of the MRCPSP-GPR/EXP using the variable definitions proposed by Pritsker *et al*. (1969). This formulation can be solved directly, without modification, by any general zero-one program solver.

As discussed in Chapter II, zero-one programming attempts at solving the resource-constrained project scheduling problems have generally led to solution times which are orders of magnitude greater than those required by specialized algorithms. This unfortunate reality is undoubtedly true for the more general MRCPSP-GPR/EXP. Nonetheless, zero-one programming is still a valid approach and advances in zero-one programming have improved the efficiency of zero-one solvers. One of these key advances is the concept of Special Ordered Sets (SOS) of Variables. As described in Chapter II, the exploitation of the SOS variables in the project scheduling problems significantly reduces the number of leaves in a search tree (corresponding to feasible solutions that must be explicitly or implicitly evaluated) and improves solution time. Because of its applicability to the MRCPSP-GPR/EXP, zero-one programming is a candidate approach to be computationally compared to other applicable approaches.

Implicit Enumeration by Activity Sequence. Talbot (1982) presents an implicit enumeration scheme for the MRCPSP, where partial schedules in the enumeration scheme are extended based solely on a predetermined activity sequence, rather than on feasibility tests like the branch-and-bound methods. Though it lacks some of the elegance of the branch-and-bound methods, its simplicity provides a precise and straightforward way to assure that all schedules have been enumerated (implicitly or explicitly). The approach also lends itself to being extended for generalized precedence and expediting resources. Because of its flexibility to evolve for the characteristics of the MRCPSP-GPR/EXP and its ability to enumerate all schedules, Talbot's algorithm provides the best starting point for developing an approach for the MRCPSP-GPR/EXP.

Basic Algorithm

The basic algorithm for the MRCPSP-GPR/EXP, hereafter referred to simply as the *Scheduler*, is an extension of the algorithm by Talbot (1982) for the MRCPSP. For the MRCPSP-GPR/EXP, the algorithm by Talbot must be extended to account for generalized precedence constraints and expediting resources. Extension of the algorithm constitutes this section. The next section presents a number of bounding rules designed to improve the efficiency of the Scheduler.

These rules are presented separately, rather than being incorporated directly into the basic algorithm, for two reasons. (1) Treating the rules as options enables their contribution to solution time, singly and in combination, to be more readily assessed. (2) If testing reveals that solution times are negatively impacted by any bounding rule, the rule may be easily eliminated from the solution algorithm. Note that bounding rules may improve the solution time of problems with certain characteristics while increasing solution times for problems with other characteristics.

The Scheduler is a depth-first implicit enumeration scheme which descends the branches of the search tree to find feasible improving solutions. Each level, *i*, of the search tree represents a partial schedule where only *i* activities have been scheduled. One activity is added to the schedule at each level. Partial schedules are augmented until all activities are scheduled and a complete feasible solution is found. Complete solutions are stored in a *k* x (*J* + 1) x 2 array, where *k* is the number of *best* solutions desired, and *J* is the number of activities in the problem. For each activity, the solution array stores two values: (1) its execution mode and (2) its start time. The objective function value is stored in Row 0 of the array. The solution array is initialized with appropriately large values (*e.g.*, 9999999).

When the algorithm finds a feasible solution, the objective function value of the solution is compared to that of the *k*-th best solution in the solution array. If the objective function value of the new solution is less than or equal to that of the *k*-th best solution, the new solution is added to the solution array and the ranking of the new solution is determined. The solution in the *k*-th position in the solution array is dropped from the array.

While solutions are added to the array of *k*-best solutions, a counter is incremented to record the number of solutions which have become part of the array. In the event that there are fewer than *k* feasible solutions, the value of *k* is reset to the count of feasible solutions. This assures that only feasible solutions are reported.

Using a depth-first search allows the solution array to be filled with *k* solutions as quickly as possible. These *k* solutions replace the artificially large values with which the solution array is initialized. Since branches of the search tree may be fathomed if any feasible solution on that branch is dominated by the current *k*-th best solution, having the solution array filled with good solutions provides a tighter upper bound which allows earlier fathoming of branches.

When scheduling a project, a scheduler may wish to present a decision-maker with a set of feasible schedules rather than a single optimal schedule generally returned by most approaches.

While a scheduling *purist* may always look for an optimal solution, decision-makers may prefer an *alternate-optimal* solution, or even a *mathematically sub-optimal* solution, for subjective or non-quantifiable reasons. The methodology used here for finding *k*-best schedules for a project provides decision makers such options. The methodology also provides the sets of solutions required by the decomposition algorithm developed in the next chapter (see Chapter VI).

The algorithm has two phases: an initialization phase and a search phase. Before the algorithm is presented, the key assumptions are outlined.

Assumptions.

1. Activity modes are numbered in order of increasing duration.
2. Only time-constant and time-increasing mode costs (cash flows) are considered.
3. All costs are non-negative.
4. Renewable resource availability need not be constant, but availabilities beyond the project horizon, *D*, are such that schedules completing beyond the project horizon are dominated by the set of *k*-best solutions. This would be true, for example, if the availability of resources were zero for periods $D+1$, $D+2$, and so on. In this case, only schedules completed by *D* would be feasible. This assumption is stated in such a way that the usual assumption of constant resource availability can be relaxed, while at the same time assuring the problem remains bounded and optimality can be assured. The Program Attributes Generator with Expediting Resources (PAGER) described in Chapter IV, as well as most other problem generators, uses the sum of activity durations, with each activity in its longest-duration mode, as the project horizon. This is certainly a convenient upper bound on the project makespan. The project manager may, however, choose any arbitrary value as the project horizon, provided there exists at least one precedence and resource-feasible schedule that can be completed by the chosen horizon.
5. During the Initialization Phase when the early- and late-start times of each activity are calculated, the activities are scheduled using their shortest duration mode. If a mode of longer duration were used, the early start of successor activities would be greater than otherwise possible and the late start time of predecessor activities would be less than otherwise possible.
6. Activities of zero duration (including, but not limited to, the *dummy* start and end activities of a project) may be included in the project. They may represent cash flows not associated

with a specific activity, milestones within a project, or dummy project source/sink nodes within a multi-project program.

Initialization Phase. During the initialization phase, the problem (with the resource constraints relaxed) is solved using a CPM-type labeling routine. The Generalized Critical Path Method (GCPM), detailed in Chapter IV, calculates the early and late start times of the activities based on their generalized precedence relationships. With early start times determined, the order that activities are added to the schedule is fixed, early start time first (in the case of ties, low activity number first). Fixing the order in which activities are added to the schedule is a departure from algorithms which gain their efficiency by enumerating only the active schedules. These active-schedule algorithms are proven to converge because they enumerate the *active* schedules of *permutations* of activities. In scheduling the MRCPSP-GPR/EXP, though, *all* schedules must be enumerated (implicitly or explicitly), so there is no computational advantage to permutating activities. The advantage, however, of adding activities in numerical order is a more straightforward implementation of the search scheme.

Search Phase. During the enumeration of the search tree, the algorithm descends from one level to the next, adding activities to the previous partial schedule. Activities are scheduled to start only at times and in modes which are feasible to the generalized precedence and resource constraints. When a leaf of the search tree is reached, the newly found schedule is added to the set of *k*-best solutions if its objective function value is as good as the objective function value of the current *k*-th best solution and discarded otherwise. The algorithm, then, *backtracks*, first to unexplored start times of the current activity and mode assignment, then to unexplored modes of the current activity. When all modes and start times of the activity at the current level have been exhausted, the algorithm backtracks to the previous activity to continue the enumeration of its modes and start times. When the algorithm tries to backtrack from the source node, it terminates.

The algorithm and associated notation are outlined below. Without loss of generality, assume activities are numbered in the order in which they are scheduled so that activity *i* is added at level *i* of the tree.

Notation for Search Algorithm

Activity Indexes:

i	=	the activity added at level i of the search tree
m_i	=	the currently scheduled mode of activity i
s_i	=	the currently scheduled start time of activity i

Activity Sets:

O_i	=	the set of activities which precede activity i
N_i	=	the set of activities which have a direct start-start lag relationship with activity i
N_i^*	=	the set of activities which have a direct or indirect lag relationship with activity i

Resource Sets:

Q^R	=	the set of all renewable resources
Q^N	=	the set of all nonrenewable resources

Time-Related Parameters:

F	=	the early program completion time
G	=	the program completion due date
D	=	the program planning horizon, or *drop dead* date ($F \leq G \leq D$)
e_i	=	the early start time of activity i
l_i	=	the late start time of activity i
w_i	=	$[e_i, l_i]$, the start time window of activity i
d_{im_i}	=	the duration of activity i in mode m_i
$?_{ij}^{\min}$	=	the minimal start-start lag time between activities i and j
$?_{ij}^{\max}$	=	the maximal start-start lag time between activities i and j

Resource-Related Parameters:

$r_{im_iq}^R$	=	the units of renewable resource q required by activity i in mode m_i

$R_{i,qt}^{R}$ = the units of renewable resource q remaining at time t at level i

$H_{i,qt}^{R}$ = the units of expediting, renewable resource q remaining at time t at level i

$r_{im_iq}^{N}$ = the units of nonrenewable resource q required by activity i in mode m_i

$R_{i,q}^{N}$ = units of nonrenewable resource q remaining at level i

$H_{i,q}^{N}$ = the units of expediting, nonrenewable resource q remaining at level i

Cost Parameters:

$c_{im_is_i}^{M}$ = the cost incurred by scheduling activity i in mode m_i at start time s_i at level i (for terminal activity J, this is the completion penalty)

$c_{im_is_i}^{R}$ = the cost of expediting, renewable resources incurred by scheduling activity i in mode m_i at start time s_i at level i

$c_{im_i}^{N}$ = the cost of expediting, nonrenewable resources incurred by scheduling activity i in mode m_i at level i

C_i = the total partial schedule cost after level i

$C_J^{(k)}$ = the total complete schedule cost of the (current) kth-best schedule

---

Basic MRCPSP-GPR/EXP Project Scheduler

Step 0 *Initialization (Start Time Labeling).* Run Generalized Critical Path Method (GCPM) Algorithm detailed in Chapter IV to calculate activity start time windows.

Step 1 Let i = 1. Assign activity 1 (the source node) its single mode, $m_1 = 1$, and early start time, $s_1 = e_1$.

Step 2 Let $i := i + 1$ and assign the next activity in order (activity i) its first mode, $m_i = 1$.

Step 3 *Nonrenewable Resource Feasibility.* Determine if m_i is feasible to the nonrenewable resource constraints (*i.e.*, the sum of regular and expediting nonrenewable resources is sufficient for activity i's nonrenewable resource demand). That is, if

$r^{N}_{im_i q}$? $R^{N}_{i?1,q}$? $H^{N}_{i?1,q}$, ? q ? Q^{N} , then m_i is nonrenewable-resource feasible. If not feasible, go to Step 12.

Step 4 Assign activity i its early start time, $s_i = e_i$.

Step 5 *Minimum Start Time Feasibility*. Determine if activity i's current start time is feasible to the precedence and start-start minimal lag constraints. Depending on the duration, d_{im_i} , of activity i and the duration, d_{jm_j} , of its predecessor activity j, the following conditions must hold for minimum start time feasibility. If infeasible, go to Step 11.

	d_{im_i} ? 0	d_{im_i} ? 0
d_{jm_j} ? 0	s_i ? s_j ? d_{jm_j}, ? j ? O_i s_i ? s_j ? ?$^{\min}_{ji}$, ? j ? C_i, j ? i	s_i ? s_j ? d_{jm_j} ? 1, ? j ? O_i s_i ? s_j ? ?$^{\min}_{ji}$? 1, ? j ? C_i, j ? i
d_{jm_j} ? 0	s_i ? s_j ? 1, ? j ? O_i s_i ? s_j ? ?$^{\min}_{ji}$? 1, ? j ? C_i, j ? i	s_i ? s_j, ? j ? O_i s_i ? s_j ? ?$^{\min}_{ji}$, ? j ? C_i, j ? i

Step 6 *Maximum Start Time Feasibility*. Determine if activity i's current start time is feasible to the start-start maximal lag and project horizon constraints. Depending on the duration of activity i and its predecessor, the following conditions must be true for maximum start time feasible. If not feasible, go to Step 12.

	d_{im_i} ? 0	d_{im_i} ? 0
d_{jm_j} ? 0	$s_i ? s_j ? ?^{\max}_{ji}, ?j ? C_i, j ? i$ $s_i ? D ? d_{im_i} ? 1$	$s_i ? s_j ? ?^{\max}_{ji} ? 1, ?j ? C_i, j ? i$ $s_i ? D$
d_{jm_j} ? 0	$s_i ? s_j ? ?^{\max}_{ji} ? 1, ?j ? C_i, j ? i$ $s_i ? D ? d_{im_i} ? 1$	$s_i ? s_j ? ?^{\max}_{ji}, ?j ? C_i, j ? i$ $s_i ? D$

Step 7 *Renewable Resource Feasibility.* Determine if activity *i*'s current mode and start time are feasible to the renewable resource constraints (*i.e.*, the sum of regular and expediting renewable resources in each period over which activity *i* extends is sufficient for activity *i*'s renewable resource demand). That is, if $r^R_{im_iq} ? R^R_{i?1,qt} ? H^R_{i?1,qt}, ? q ? Q^R, s_i ? t ? s_i ? d_{im_i}$, then activity *i*'s current mode and start time are renewable-resource feasible. If not feasible, go to Step 11.

Step 8 *Adjust Resources and Costs.* The new partial schedule formed by adding activity *i* in mode m_i at start time s_i is feasible and may lead to an improved solution. Adjust resource availabilities and the schedule cost as follows:

$$?? \quad R^N_{iq} ? \begin{cases} R^N_{i?1,q} ? r^N_{im_iq} \text{ if } r^N_{im_iq} ? R^N_{i?1,q} \\ 0 \text{ if } r^N_{im_iq} ? R^N_{i?1,q} \end{cases}, ? q ? Q^N$$

$$?? \quad H^N_{iq} ? \begin{cases} 0 \text{ if } r^N_{im_iq} ? R^N_{i?1,q} \\ H^N_{i?1,q} ? R^N_{i?1,q} ? r^N_{im_iq} \text{ if } r^N_{im_iq} ? R^N_{i?1,q} \end{cases}, ? q ? Q^N$$

$$?? \quad R^R_{iqt} ? \begin{cases} R^R_{i?1,qt} ? r^R_{im_iqt} \text{ if } r^R_{im_iqt} ? R^R_{i?1,qt} \\ 0 \text{ if } r^R_{im_iqt} ? R^R_{i?1,qt} \end{cases}, ? q ? Q^R, s_i ? t ? s_i ? d_{im_i}$$

$$?? \quad H^R_{iqt} \; ? \begin{cases} 0 \text{ if } r^R_{im_iqt} \; ? \; R^R_{i?1,qt} \\ H^R_{i?1,qt} \; ? \; R^R_{i?1,qt} \; ? \; r^R_{im_iqt} \text{ if } r^R_{im_iqt} \; ? \; R^R_{i?1,qt} \end{cases}, ? \; q \; ? \; Q^R, s_i \; ? \; t \; ? \; s_i \; ? \; d_{im_i}$$

$$?? \quad C_i \; ? \; C_{i?1} \; ? \; c^N_{im_i} \; ? \; c^M_{im_is_i} \; ? \; c^R_{im_is_i}$$

Step 9 If activity *i* is NOT the terminal sink node, go to Step 2. Otherwise (*i.e.*, activity *i* is the terminal node), if this schedule is as good as the current *k*-th best solution, add this complete schedule to the set of *k*-best solutions.

Step 10 *Adjust Resources and Costs.* Remove activity *i* in mode m_i at start time s_i from the current complete schedule. Adjust resource availabilities and the schedule cost.

Step 11 *Backtrack by Start Time.* Assign activity *i* start time $s_i := s_i + 1$. If start time, s_i, is less than or equal to the late start time of activity *i* ($s_i \; ? \; l_i$), go to Step 5.

Step 12 *Backtrack by Mode.* Assign activity *i* mode $m_i := m_i + 1$. If mode, m_i, is less than or equal to the maximum number of modes of activity *i*, go to Step 3.

Step 13 *Backtrack by Activity.* Backtrack to activity $i := i - 1$. If $i \; ? \; 0$, go to Step 11.

Step 14 Stop. Algorithm *complete* and *k*-best solutions found.

In the mathematical formulation of precedence relationships (Chapter III, Equations (2) through (7)), it is generally assumed that both activities in a precedence relationship have non-zero duration. Note, however, that the conditions in Steps 5 and 6 contain cases where one or both activities in a precedence relationship have zero duration. As discussed in Chapter IV (see Figure 4-11 and accompanying text), the rules for activities with zero duration are somewhat different than for those with non-zero duration. Steps 5 and 6 implement these alternate rules.

Since a cost is incurred only for the use of expediting resources, regular and expediting resource availabilities must be accounted for separately. This accounting is done in Steps 8 and 10 where resource availabilities are adjusted. When adding an activity to a partial schedule in Step 8, one of two cases is true: 1) the demand by the activity for a particular resource is no greater than the current *regular* availability of that resource or 2) the demand for the resource *is* greater than the current regular availability of that resource. If the demand is no greater than the current regular availability, then the regular availability alone is decremented. If, on the other hand, the demand exceeds current regular availability, then all of the regular availability is used first and then expediting resources are used to meet the balance of the resource demand. Conversely, in Step 10 where activities are removed from the partial schedules, the freed resources are used first to *backfill* the expediting pool of resources and then the regular pool.

With the Basic MRCPSP-GPR/EXP Project Scheduler outlined, attention is now turned to the convergence of the algorithm, in Theorem 5.1.

Theorem 5-1. Optimality of the Basic MRCPSP-GPR/EXP Project Scheduler

Theorem: If **P** is a feasible MRCPSP-GPR/EXP, the best solution found by the MRCPSP-GPR/EXP Project Scheduler is an optimal solution for **P**.

Proof: Let **P** be a feasible MRCPSP-GPR/EXP with objective function ? .

Define a schedule of **P** to be a precedence- and resource-feasible assignment of a mode and start time to each activity in **P**, along with the accompanying expediting resources required to make that assignment feasible.

Let **S** be an optimal schedule for **P** with objective function value ? ?**S**?.

Must show that the Basic MRCPSP-GPR/EXP Project Scheduler finds a schedule **S**? with ? ?**S**??? ? ?**S**?.

Let **A** be an explicit enumeration of all possible schedules of **P**, where each activity *i* in **P** may be performed in any of its respective modes, m_i, and may start at any time in the interval ?1,? ?. Then, **A** contains all schedules of **P** and, consequently, ? a schedule **S**?? **A** such that ? ?**S**??? ? ?**S**?.

Now show that the Basic MRCPSP-GPR/EXP Project Scheduler eliminates all schedules, $\overline{\mathbf{S}}$ in **A**, where ? ?$\overline{\mathbf{S}}$?? ? ?**S**?, but does not eliminate **S**?.

Enumeration Control. Most of the steps of the Basic MRCPSP-GPR/EXP Project Scheduler control the incrementing and backtracking of the algorithm. A few additional steps limit the activity start time windows (Steps 5 and 6) and provide basic resource feasibility tests (Steps 3 and 7). In the absence of Steps 3, 5, 6, and 7, the algorithm would explicitly enumerate every mode assignment of each activity, as well as every possible start time for each activity from t = 1 to ? . Must show, then, that the start time limitations and feasibility tests do not eliminate solutions which would dominate all other solutions.

Reduction of Project Horizon. In its initialization phase, the Basic MRCPSP-GPR/EXP Project Scheduler eliminates all schedules which complete after the project horizon, *D*. By design, the project horizon and renewable resource availabilities are defined in such a way that ? a schedule which completes by *D* without the need for expediting resources. In PAGER, as with most other problem generators, *D* is defined as the sum of all activity durations, with each activity in its longest-duration mode. Regular renewable resource availabilities are, then, generated sufficiently high so that the above condition is always true. If resource availabilities are given and not generated, then *D* must be chosen such that: (1) *D* is no less than the sum of all activity durations, with each activity in its longest-duration mode, and (2) the above regular renewable resource condition holds. Assume, for the moment, that resource availability is constant. This assumption is relaxed later.

Let $\mathbf{S}?$ be a schedule for **P**, such that some activity, $i?$, completes after time *D*. By the way *D* is defined, scheduling activities back-to-back (regardless of order) at most spans *D*. Consequently, scheduling activity $i?$ to complete after *D* implies that schedule $\mathbf{S}?$ creates some time interval $?t_1, t_2?$, with $t_2 ? D$, when no activity is in process. Because resource availability is assumed to be constant, all activities which are scheduled to start after t_2 can be shifted $t_2 ? t_1 ? 1$ time units to the left, while maintaining their relative temporal relations and without increasing expediting resource costs. (If there are other such time intervals remaining, they may be eliminated in the same way.)

Now, every activity in the revised schedule, $\mathbf{S}_D^?$, completes by time D. Since activity costs are non-decreasing in time and since activity $i?$ starts earlier in schedule $\mathbf{S}_D^?$ than it did in schedule $\mathbf{S}?$, the cost of activity $i?$ under schedule $\mathbf{S}_D^?$ is no greater than its cost under schedule $\mathbf{S}?$. The same holds true for any other activity shifted to create schedule $\mathbf{S}_D^?$. Therefore, the cost of the revised schedule, $\mathbf{S}_D^?$, is no greater than that of schedule $\mathbf{S}?$, $? \, ?\mathbf{S}_D^? ?? \, ? \, ?\mathbf{S}??$. This implies that, for every schedule, $\mathbf{S}?$, which completes after D, ? a schedule, $\mathbf{S}_D^?$, which completes by D which has objective function value less than or equal to $\mathbf{S}?$. Therefore, if all schedules which complete after D are eliminated, there yet remains an optimal schedule for **P**.

The assumption of constant resource availabilities may, now, be relaxed with the assumption that times beyond D are either infeasible (*e.g.*, zero resource availability), dominated, or of no practical interest to the Program Manager.

Reduction of Early/Late Activity Start Times. The initialization phase also calculates early and late start time windows, with the assumption that no activity starts before time 1 or completes after time D. The proof for the GCPM, in Chapter IV, shows that no activity may start before its calculated early start time or after its late start time without violating generalized precedence constraints or pushing some other activity outside the assumed start or completion bounds of the project. Hence, eliminating activity start times outside their respective early/late start times (Steps 5 and 6) eliminates only infeasible assignments and, therefore, cannot eliminate an optimal solution.

Resource Feasibility Tests. Steps 3 and 7 perform tests to check if adding an activity (in a given mode and start time) to the current partial schedule creates a resource conflict. If so, the next mode and start time assignment are checked. Again, these steps eliminate only infeasible activity assignments and cannot, therefore, eliminate an optimal solution.

Optimality Test. Step 9 is executed only when a complete (and feasible) schedule, $\bar{\mathbf{S}}$, has been constructed. A comparison between $\bar{\mathbf{S}}$ and the currently best schedule, $\hat{\mathbf{S}}$, is made. If $?\,(\bar{\mathbf{S}})\,?\,?\,(\hat{\mathbf{S}})$, then $\bar{\mathbf{S}}$ is eliminated. Otherwise, $\bar{\mathbf{S}}$ becomes the current best solution, $\hat{\mathbf{S}} := \bar{\mathbf{S}}$. When all schedules have been enumerated and tested, let the remaining best solution be relabeled $\mathbf{S}^{?}$.

It has been shown that the Basic MRCPSP-GPR/EXP Project Scheduler explicitly enumerates all schedules of P within the bounds of the project horizon, *D*. It has also been shown that limiting the search for schedules that complete by *D* does not eliminate any schedule which dominates all other schedules. Therefore, the Basic MRCPSP-GPR/EXP Project Scheduler has been shown to eliminate all schedules, $\bar{\mathbf{S}}$ in **A**, where $?\,(\bar{\mathbf{S}})\,?\,?\,(\mathbf{S}^{?})$. Since no other schedules remain untested, $\mathbf{S}^{?}$ is an optimal solution, where $?\,(\mathbf{S}^{?})\,?\,?\,(\mathbf{S})$.

Bounding Rules

Recall that, in the Scheduler, activities are scheduled to start only at times and in modes which are feasible to the generalized precedence and resource constraints. In addition to the basic algorithm's rudimentary feasibility tests, more advanced feasibility and *goodness* tests (based on objective function value) may be applied to eliminate partial schedules which lead to infeasible complete schedules or schedules dominated by the current *k*-th best schedule. By applying better feasibility tests and by checking schedule goodness at each level of the tree rather than only at the leaves where complete schedules are found, unproductive branches of the tree may be fathomed sooner, thereby reducing the portion of the tree explicitly enumerated. The efficiency of the algorithm may be improved based on the degree to which feasibility and bounding rules prune the search tree. This section provides a number of these rules.

<u>Bounding Rule ZDS (Zero-Duration Activity Start)</u>. A zero-duration activity may have a cost as well as nonrenewable resource demands associated with it. It does not, however, have any associated renewable resource demands. With the assumption that all mode costs are time constant or time increasing, there is no benefit to delaying a zero-duration activity, either to reduce its cost or to make it renewable resource feasible. Therefore, the algorithm does not enumerate any but the earliest feasible start time of a zero-duration activity. Backtracking, consequently, proceeds to enumerating any unenumerated modes of the zero-duration activity or backtracking to a previous activity (*i.e.*, level) in the precedence tree.

Bounding Rule ZDS may replace Step 11 which becomes:

Backtrack by Start Time. If the duration of activity *i* is zero, go to Step 12. Otherwise, assign activity *i* start time $s_i = s_i + 1$. If start time, s_i, is less than or equal to the late start time of activity *i* ($s_i \le l_i$), go to Step 5.

<u>Feasibility Rule NRF (Nonrenewable Resource Feasibility)</u>. Step 3 checks if there is sufficient remaining nonrenewable resources available to schedule activity *i* in mode m_i. A stronger bound on the feasibility of the current mode selection, however, is to verify that the remaining available nonrenewable resources are at least as great as the demand, not only of the current activity in its selected mode, but also the demand of all remaining unscheduled activities in their lowest demand modes as well. That is,

$$r^N_{im_iq} + \sum_{j=i+1}^{J} \min_{m \in M_j} \left\{ r^N_{jmq} \right\} \le R^N_{i-1,q} - H^N_{i-1,q}, \ \forall q \in Q^N .$$

Sprecher and Drexl (1996a: 19) show that this bounding rule can be easily implemented as a preprocessing step. For each nonrenewable resource, *q*, the modes of each activity, *i*, are compared to find the minimum possible usage of resource *q* by activity *i*:

$$rmin^N_{iq} = \min_{m \in M_i} \left\{ r^N_{imq} \right\} \quad \text{for } i = 1, \ldots, J.$$

Then, the input data is adjusted by reducing the requirement for resource *q* by each mode of activity *i* by the minimum possible usage of resource *q* by activity *i*:

$$\bar{r}^N_{imq} = r^N_{imq} - rmin^N_{iq} \quad \text{for } i = 1, \ldots, J, \ m \in M_i, \ q \in Q^N$$

Finally, the input data is adjusted by reducing the availability of resource *q* by the sum of the activities' minimum possible requirements:

$$\overline{R}_q^N = R_q^N - \sum_{i=1}^{J} rmin_{iq}^N \quad \text{for } q \in Q^N$$

Therefore, bounding rule NRF is applied as Step 0a and is added between Steps 0 and 1. Step 3 is applied as normal.

Bounding Rule NEC (Nonrenewable Expediting Resource Cost). It is possible, at any level of the search tree, for an activity to be scheduled which is feasible as to the availability of nonrenewable resources but which cannot lead to an improved schedule. Assume activity *i* is being added to the *i*-1 partial schedule in mode m_i and that Step 3 has determined the addition of activity *i* to be nonrenewable resource feasible. It is possible, however, that nonrenewable resource feasibility can be achieved only at the cost of some quantity of expediting resources. If the cost of those expediting resources plus the *running* cost of the *i*-1 partial schedule exceeds the cost of the current *k*-th best solution, then the addition of activity *i* in mode m_i cannot lead to an improved schedule. Activity *i* in mode m_i is, therefore, rejected as an improving addition to the *i*-1 partial schedule and its corresponding branch fathomed.

This bounding rule is further strengthened when applied in conjunction with Bounding Rule NRF. If the cost of expediting resources resulting not only from the addition of activity *i* in mode m_i but also from the addition of the remaining unscheduled activities in their least demanding modes is considered, fathoming of unimproving branches can occur higher in the tree and the total search time is reduced.

Bounding Rule NEC, then, can be applied as Step 3a between Steps 3 and 4 and can be expressed as follows:

If $C_{i-1} + \sum_{k \in K^N} c_k^N \cdot \max\left\{0, r_{im_ik}^N + \sum_{j=i+1}^{J} \min_{m \in M_j}\left\{r_{jmk}^N\right\} - R_{i,k}^N\right\} < C_J^{(k)}$, add activity *i* in mode m_i.

Otherwise, go to Step 12 (*i.e.*, fathom the current branch).

Feasibility Rule EST (Early Start Time). The basic algorithm enumerates over the entire GCPM start time window for each activity, relying on Step 5 to determine the generalized precedence feasibility of each start time. In any branch of the search tree, however, the earliest feasible start time of an activity may be explicitly determined based on the completion time of generalized predecessor activities which have already been scheduled. Consequently, Steps 4 and 5 may be replaced by Bounding Rule EST (call it Step 4/5) as follows:

Assign activity i the maximum of its early start time, the latest completion time of all its predecessors, and its earliest start-start minimum lag feasible time. That is,

$$s_i ? \begin{cases} \max\left\{ e_i, \max\limits_{\substack{j ? O_i \\ d_{jm_j} ? 0}} \{ s_j ? d_{jm_j} \}, \max\limits_{\substack{j ? O_i \\ d_{jm_j} ? 0}} \{ s_j ? 1 \}, \max\limits_{\substack{j ? C_i^* \\ djm_j ? 0}} \{ s_j ? ?^{\min}_{ji} \}, \max\limits_{\substack{j ? C_i^* \\ djm_j ? 0}} \{ s_j ? ?^{\min}_{ji} ? 1 \} \right\} & \text{if } d_{im_i} ? 0 \\ \max\left\{ e_i, \max\limits_{\substack{j ? O_i \\ d_{jm_j} ? 0}} \{ s_j ? d_{jm_j} ? 1 \}, \max\limits_{\substack{j ? O_i \\ d_{jm_j} ? 0}} \{ s_j \}, \max\limits_{\substack{j ? C_i^* \\ djm_j ? 0}} \{ s_j ? ?^{\min}_{ji} ? 1 \}, \max\limits_{\substack{j ? C_i^* \\ djm_j ? 0}} \{ s_j ? ?^{\min}_{ji} \} \right\} & \text{if } d_{im_i} ? 0 \end{cases} .$$

Since all subsequently enumerated start times for any activity are minimal start time feasible, the direction given in Step 12 to go to Step 5 may be changed to redirect to Step 6.

Feasibility Rule MD (Mode Duration). Step 6 determines if the current start time of an activity is feasible to the maximal lags of its generalized predecessors. It also verifies that the current start time plus duration of the activity does not exceed the project due date. Feasibility Rule MD goes further to assure that if activity i is scheduled in mode m_i at start time s_i, the earliest possible finish time of the remaining unscheduled activities (in their shortest-duration modes) does also not exceed the project due date. If any do, the current start time and later start times are infeasible and those branches of the tree are fathomed.

Feasibility Rule MD checks if any critical path emanating from activity i results in any activity finishing after the project due date. If so, not only can the current and future start times of activity i in mode m_i be fathomed, but the late start time of activity i can be reduced to $l_i = s_i – 1$ for any modes of equal or greater duration.

To determine the critical paths emanating from activity i, the GCPM is first run to determine the early start time of each activity. Recall that GCPM is run with the shortest-duration mode of each activity. The (shortest-mode) duration of activity i is, then, artificially increased by one and temporarily fixed. The GCPM is run again. Those activities whose start times have been delayed as a result of activity i's duration being increased are on a *mode critical path* from activity i (*i.e.*, activity i has no free slack). It follows that any of these mode critical path activities are also delayed if activity i's start is delayed. To simplify the rule, however, recall that no activity can finish later than the terminal dummy activity, *J*. Therefore, it is sufficient to consider whether or not activity *J* is on a mode critical path from activity *i*.

Bounding Rule MD, then, may be incorporated as Step 6a as follows:

Mode Duration Feasibility. Let the *delay*, ? , resulting from activity *i* being scheduled in mode m_i at start time s_i be defined as the difference between s_i and activity *i*'s early start time, e_i, plus the difference between the duration of activity *i* in mode m_i and its shortest-duration mode.

$$? \; ? \; ?s_i \; ? \; e_i ?? \; ?d_{im} \; ? \; d_{i1}?$$

In other words, any activity on activity *i*'s mode critical paths is delayed both by activity *i* starting later than its early start time and by activity *i* being scheduled in a mode longer than its shortest-duration mode. Therefore, activity *i* cannot be scheduled in mode m_i at start time s_i if (1) activity *J* is on activity *i*'s mode critical path and (2) the early start time of activity *J* plus ? exceeds the project horizon *D*,

$$e_J \; ?? \; ? D.$$

<u>Bounding Rule MC (Mode Cost)</u>. If the cost of scheduling activity *i* in mode m_i at start time s_i plus the running cost of the *i*-1 partial schedule exceeds the current *k*-th best solution, then start time s_i and any later start time for activity *i* in mode m_i leads to a dominated solution and can be fathomed. If, in addition, the lowest mode costs of the remaining unscheduled activities are also added, an even stronger bound can be achieved.

Recall that an activity's mode cost is not only a function of its mode, but also of its start time. Therefore, in a fashion similar to that for Bounding Rule MD, the GCPM is run to determine the early start time of each activity. The early time of activity *i* is, then, artificially increased by one time unit and temporarily fixed. The GCPM is run again. Those activities whose start times have been delayed as a result of activity *i*'s start time being delayed are on a *start time critical path* from activity *i*. These activities, though, are not necessarily on a mode critical path from activity *i*. Therefore, Bounding Rule MC may be added as Step 6b as follows:

Define $? \, ?_i^M$ as the set of activities on a mode critical path from activity *i*. Define $? \, ?_i^S$ as the set of activities on a start time critical path from activity *i*. Then, each unscheduled activity, *j*, is in set $? \, ?_i^M$, set $? \, ?_i^S$, or neither set. If activity *j* is in neither set, ? an activity $i_j?$ such that $j \, ? \, ? \, ?_{i_j?}^M$ or $j \, ? \, ? \, ?_{i_j?}^S$, even if that activity is the dummy source activity.

Then, let $?^{M} ? ?s_i ? e_i ?? ?d_{im} ? d_{i1}?$ be the delay for activities in set $?_i^M$. Let $?^{S} ? ?s_i ? e_i ?$ be the delay for activities in set $?_i^S$. Define corresponding delays for activities in sets $?_{i_j?}^M$ and $?_{i_j?}^S$.

Now, compute the running cost of the *i*-1 partial schedule plus the cost of scheduling activity *i* in mode m_i at start time s_i plus the mode cost of the unscheduled activities in their minimum cost mode and earliest feasible start time. If that cost is no greater than the current *k*-th best solution, retain activity *i* in mode m_i at start time s_i. Mathematically, if

$$C_{i?1} ? c^M_{im_is_i} ? \underset{\substack{j?i?1 \\ j??_i^M}}{\overset{J}{?}} \min_{m?M_j} ?c^M_{jm ?e_j ?? _i^M ?}? ? \underset{\substack{j?i?1 \\ j??_i^S}}{\overset{J}{?}} \min_{m?M_j} ?c^M_{jm ?e_j ?? _i^S ?}? ?$$

$$\underset{\substack{j?i?1 \\ j??_{i_j?}^M}}{\overset{J}{?}} \min_{m?M_j} ?c^M_{jm ?e_j ?? _{i_j?}^M ?}? ? \underset{\substack{j?i?1 \\ j??_{i_j?}^S}}{\overset{J}{?}} \min_{m?M_j} ?c^M_{jm ?e_j ?? _{j i?}^S ?}? ? C_J^{(k)},$$

then retain the current start time. Otherwise, go to Step 12.

<u>Bounding Rule REC (Renewable Expediting Resource Cost)</u>. As is the case with nonrenewable resources, the addition of activity *i* in mode m_i at start time s_i may be renewable resource feasible but lead to a dominated solution. This is the case when other running costs plus the cost of renewable expediting resources required for feasibility exceeds the value of the current *k*-th best solution. Thus, a check for dominance may be added as Step 7a as follows: If

$$C_{i?1} ? \underset{k?K^R}{?} \overset{s_i ? d_{im_i} ?1}{\underset{t?s_i}{?}} \max ?0, r^R_{im_ik} ? R^R_{i,kt} ?? C_J^{(k)},$$

then the current partial schedule may lead to an improved solution. If not, go to Step 11.

If used in conjunction with Bounding Rules NEC so that the cost of nonrenewable expediting resources and renewable expediting resources are considered, fathoming occurs even earlier. The resulting equivalent fathoming condition would be:

$$C_{i+1} = \sum_{k \in K^N} c_k^N \max\left\{0, r^N_{im_ik} + \sum_{j=i+1}^{J} \min_{m \in M_j}\left(r^N_{jmk}\right) - R^N_{i,k}\right\} + \sum_{k \in K^R} \sum_{t=s_i}^{s_i+d_{im_i}-1} \max\left(0, r^R_{im_ik} - R^R_{i,kt}\right) + C_J^{(k)}$$

Feasibility Rule MOD (Infeasible Modes). Feasibility Rule MOD tests each mode of each activity to determine if it is feasible as to the renewable and nonrenewable resource constraints. A mode, m_i, of activity i is infeasible vis-à-vis a nonrenewable resource, q^N, if the usage of q^N by m_i plus the minimal usage of q^N by all other activities exceeds the availability (regular plus expediting) of q^N. A mode, m_i, of activity i is infeasible vis-à-vis a renewable resource, q^R, if the usage of q^R by m_i exceeds the availability (regular plus expediting) of q^R in the time period where the availability of q^R is greatest. Note that when comparing renewable resource usage against availability, the time period when that resource is most available must be determined. This is required because of the assumption of nonconstant resource availability.

Feasibility Rule MOD is performed at the beginning of the search phase to eliminate infeasible modes as soon as possible. The rule is inserted into the algorithm as Step 0b as follows: For each mode, m_i, of each activity, i, if

$$r^N_{im_iq} + \sum_{j \neq i} \min_{m \in M_j}\left(r^N_{jmq}\right) > R^N_q + H^N_q$$

for any nonrenewable resource, q^N, or if

$$r^R_{im_iq} > \max_{t \in [1,D]}\left(R^R_{tq} + H^R_{tq}\right)$$

for any renewable resource, q^R, then mode, m_i, of activity i is infeasible and eliminated from further consideration.

Testing

Extensive testing was conducted to address a number of issues about the MRCPSP-GPR/EXP Scheduler. These include, but are not limited to, an investigation into the computational contribution of the optional bounding rules and a comparison of the algorithm versus a general integer programming solution approach. Each issue is addressed separately below.

For all testing, problem instances were generated using PAGER and solved using a 750 MHz, Pentium III processor with 256 MB of Random Access Memory (RAM). A total of 4992 problems were generated, most of which were solved in a variety of ways (*i.e.*, using different combinations of bounding rules and / or alternate values of *k*). The total number of tests conducted is 52,521.

Note that in some of the charts below, the term *job* is used in place of *activity*. The two terms are intended to be equivalent and *job* is used simply to conserve space.

Test Problem Parameters Held Constant. A review of Chapter IV reveals an extensive list of parameters that can be set in PAGER to generate tailor-made problem instances. Some parameters, such as the minimum and maximum number of start or end nodes, can be altered to *shape* the underlying project network. In this particular case, the parameters are set to give the most flexibility to PAGER, with the minimums being set to one and the maximums set to the number of activities in the project. This is required to assure that the network Restrictiveness parameter controls the network structure. For example, a Restrictiveness of one leads to an end-to-end string of activities, requiring the minimum number of start and end nodes to be one. On the other hand, a Restrictiveness of zero produces a network where there are not temporal relationships at all between the activities. This requires that the maximum number of start and end nodes be at least as great as the number of activities.

Other parameters are held constant to manage the size of the experimental design. Varying some of these parameters might produce interesting excursions to this study. The parameters held constant throughout testing include:

?? Lower and Upper Bounds on Activity Lags: When a generalized precedence exists between two activities, say activities *i* and *j*, then the difference between the start times of activities *i* and *j* must be no less than their minimal lag and no greater than their maximal lag. Minimal lags were randomly drawn from between –2 and +2 and maximal lags from between +4 and +8. For instance, suppose the minimal lag is randomly chosen to be –1 and the maximal lag is randomly chosen to be +6. This implies that activity *j* may start as early as one time period before the start of activity *i* or as late as 6 time periods after the start of activity *i*. The choice of intervals [–2, +2] and [+4, +8] for randomly drawing minimal and maximal lags, respectively, was arbitrary. These values were chosen simply to give some variety to the generalized precedences, while allowing for the possibility of concurrent activity start times.

?? Resource Demands: The number of units of a particular resource that an activity may require was randomly drawn from between 1 and 10.

?? Project Penalty Cost: If a project is due at time *t*, then a project completion penalty is assessed starting at time *t*+1. The penalty to be assessed at time *t*+1 was randomly drawn from between 500 and 750 units. For each period beyond time *t*+1, the penalty assessed is increased by some increment, randomly drawn from between 400 and 500 units. Again, this was a matter of preference.

?? Mode Costs: Each scheduled activity is assessed a cost which is a function of the mode and start time. When activity modes are generated, each is assigned a baseline cost randomly drawn from between 50 and 100 units. A mode's baseline cost is assessed if the activity is scheduled in that particular mode at the activity's early start time. If the activity is scheduled later than at its early start time, the mode's baseline cost plus a time-dependent incremental cost is assessed. The incremental cost associated with a mode was also randomly drawn from between 50 and 100 units.

?? Expediting Resource Costs: If an expediting resource, either renewable or nonrenewable, is used, an expediting resource cost is assessed. Each expediting resource is assigned a cost randomly drawn from between 0 and 50 units.

Table 5-1 summarizes the problem parameters held constant throughout testing.

Table 5-1. Problem Generation Parameters Held Constant

	Min	Max
Minimal Lag	-2	2
Maximal Lag	4	8
Resource Demand	1	10
Base Project Penalty	500	750
Project Penalty Increment	400	500
Base Mode Cost	50	100
Mode Cost Increment	50	100
Expediting Resource Cost	0	50

Test Problem Parameters Which Are Varied. A number of key parameters used to generate test problems were varied throughout the testing. Some of these are parameters identified by other researchers (*e.g.*, Kolisch *et al.*, 1995; Schwindt, 1996; Van Hove, 1998) as having the greatest effect on problem difficulty. Others are key features of MRCPSP-GPR/EXP that may impact problem difficulty. These parameters are outlined in Table 5-2. Table 5-2 does not list the values that these parameters might take. The parameter values are, instead, introduced when each experiment is described below.

The test designs that are introduced in this section are referred to as the *full*, *reduced*, and *minimal* designs. The adjectives describing the designs are used simply to reflect their relative scopes and to provide a convenient means of referring to them.

Table 5-2. Parameters Which Are Varied

PARAMETER
Number of Modes Per Activity
Job Duration, Maximum
Lag Fraction
Project Network Restrictiveness
Number of Renewable/Nonrenewable Resources
Renewable/Nonrenewable Resource Factor
Regular Renewable/Nonrenewable Resource Strength
Total Renewable/Nonrenewable Resource Strength

Computational Contribution of Bounding Rules. The first experiment conducted was designed to assess the contribution each of the eight optional bounding rules makes to solution time. Since each rule reduces the algorithmic search space, each should, theoretically, improve overall problem solution time. However, there is computational overhead associated with each rule. Therefore, an experiment to determine if there is a practical contribution by the rules is essential.

Table 5-3. *Reduced* Test Design

PARAMETER	LEVELS		
Number of Modes Per Activity	1	3	
Job Duration, Maximum	10	20	
Lag Fraction	0.00	0.20	
Project Network Restrictiveness	0.00	0.50	1.00
Number of Renewable/Nonrenewable Resources	1	3	
Renewable/Nonrenewable Resource Factor	0.50	1.00	
Regular Renewable/Nonrenewable Resource Strength	0.00	0.50	1.00
Total Renewable/Nonrenewable Resource Strength	1.00		
Total Combinations = 288			

The experiment was conducted by generating 1440 projects with five activities each using the reduced test design in Table 5-3. The *reduced* design contains 288 design points. Five projects were generated for each design point.

The projects were scheduled using the basic algorithm and, then, using each individual bounding rule. The results, shown in Table 5-4, list the rule(s) applied, the solution times, and the improvement in solution time offered by each rule (as a percentage of the solution time without rules). Figure 5-1 shows the results graphically. Rule MC showed the greatest single-rule improvement, solving problems (on average) in 0.002 of the time required by the basic solution.

On the other hand, Rules MD, MOD, and NRF were only slightly better than solving with no rules at all. When all the rules are combined, the solution algorithm solved the problem set in 0.001 of the basic case solution time.

Table 5-4. Rule vs. Average Solution Time (seconds) for 5 Activities

	Solution Time (seconds)				Ave Time as
Rule	Min	Average	Max	Std Dev	% of "None"
None	0	5.808	170.7	17.407	100.0%
MD	0.000	5.545	166.530	16.557	95.5%
MOD	0.000	5.493	166.110	16.452	94.6%
NRF	0.000	5.485	166.220	16.466	94.4%
EST	0.000	4.293	128.250	12.777	73.9%
ZDS	0.000	2.259	57.840	6.320	38.9%
REC	0.000	0.991	44.290	3.558	17.1%
NEC	0.000	0.497	28.050	2.169	8.6%
MC	0.000	0.009	2.330	0.087	0.2%
All	0.000	0.003	0.950	0.030	0.1%

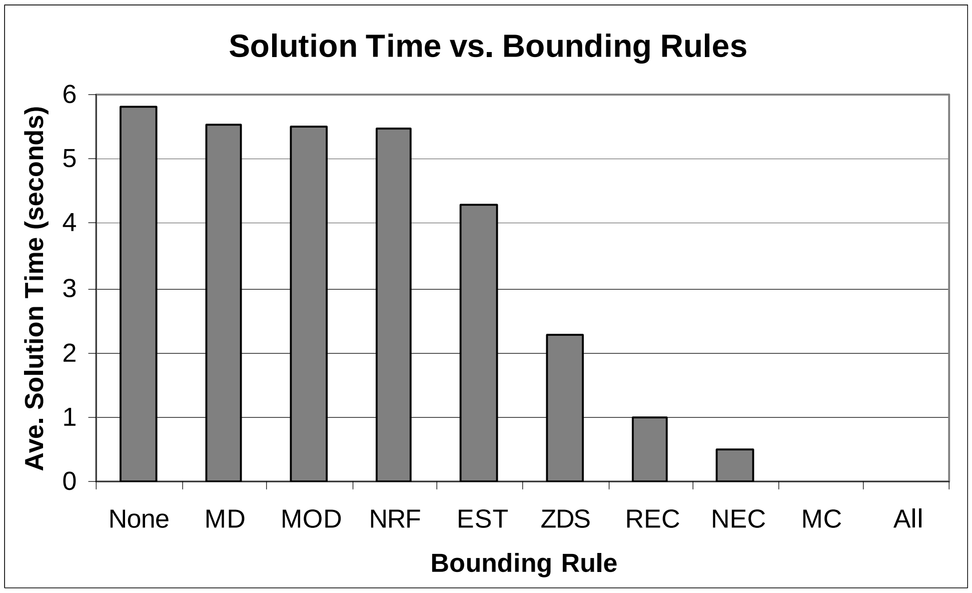


Figure 5-1. Rule vs. Average Solution Time (seconds) for 5 Activities

The next step in the investigation of the bounding rules was to generate 288 ten-activity projects, one instance for each design point. These were solved both with all rules and without rules. A time limit of 300 seconds (5 minutes) was imposed on the solution time for each problem. When none of the rules were applied, only 54 of the 288 problems solved to optimality within the time limit. By contrast, 256 problems solved to optimality within 300 seconds when all ofthe rules were applied (see Table 5-5 and Figure 5-2).

Table 5-5. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities

Rule	Number of Optimal Solutions Found	% of Total Problems Solved
None	54	18.8%
All	256	88.9%

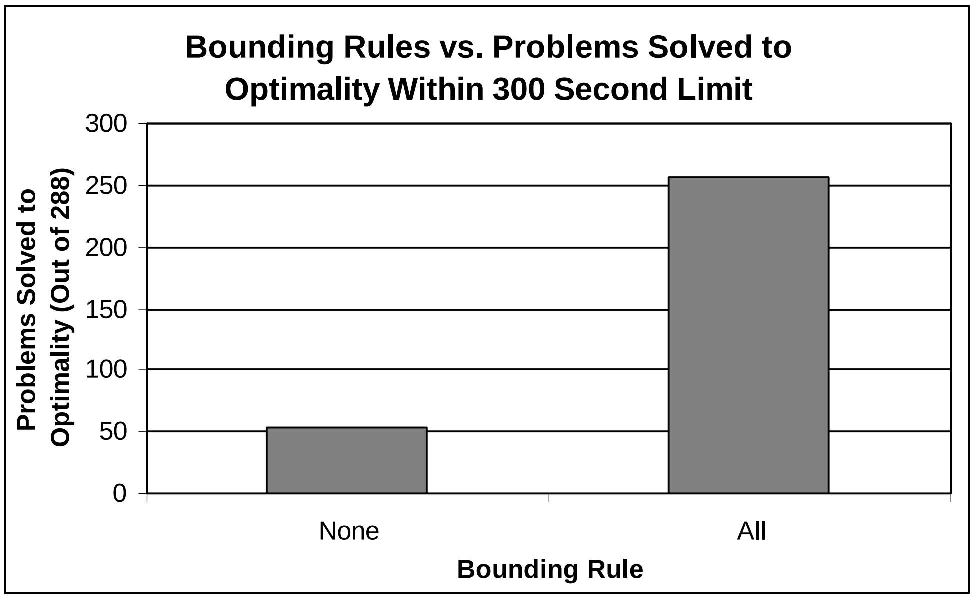


Figure 5-2. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities

When the number of problems solved to optimalty were tallied as a function of the problem characteristics, neither the fraction of generalized precedences, the resource factor (RF), or the resource strength (RS) were important factors in the number of problems solved within the time limit. However, the number of modes and the network restrictiveness (RT) were important factors. When no bounding rules were used, only problems (54 of 144) with a single mode were solved within the time limit (Table 5-6 and Figure 5-3). When all of the rules were used, however, 133 of the 144 problems with a single mode (92.4%) solved to optimality within the time limit and 123 of the 144 problems with three modes (89.6%) solved to optimality within the time limit.

Table 5-6. Rule vs. Problems Solved to Optimality (Within 300 sec.)

for 10 Activities and Varying Modes

Rule	Modes 1	Modes 3	Total
None	54	0	54
All	133	123	256

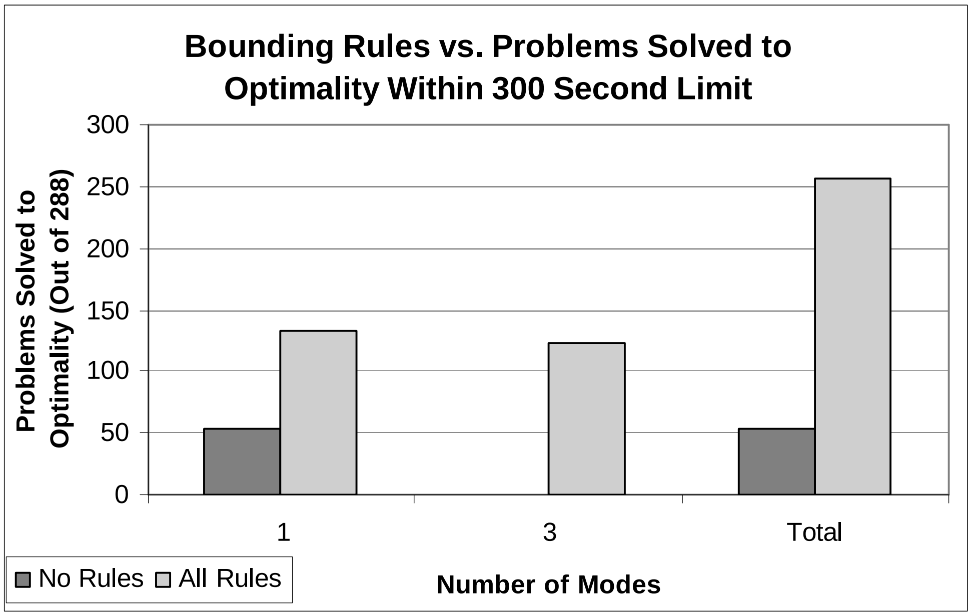


Figure 5-3. Rule vs. Problems Solved to Optimality (Within 300 sec.)

for 10 Activities and Varying Modes

Table 5-7 and Figure 5-4 show the results for varying levels of RT. When no bounding rules are used, almost all of the problems solved to optimality within the time limit have an RT of 1.0 (the easiest case). When all bounding rules are used, 71.9%, 94.8%, and 100% of the problems are solved with an RT of 0.0, 0.5, and 1.0, respectively. Based on these results, the bounding rules materially improve solution time. All further experiments use all bounding rules.

Table 5-7. Rule vs. Problems Solved to Optimality (Within 300 sec.)

for 10 Activities and Varying RT

Rule	Network Restrictiveness 0.0	Network Restrictiveness 0.5	Network Restrictiveness 1.0	Total
None	0	6	48	54
All	69	91	96	256

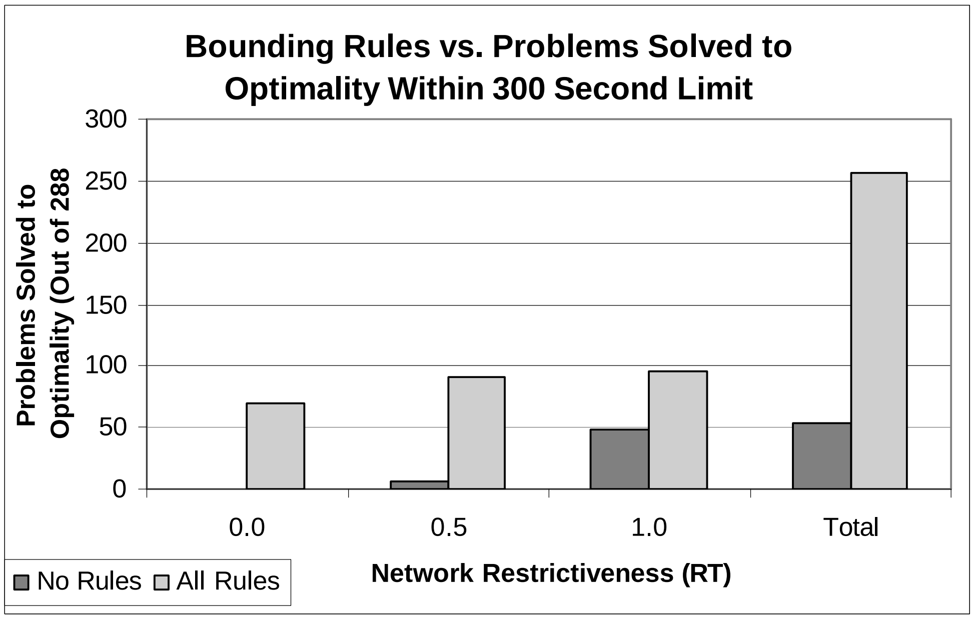


Figure 5-4. Rule vs. Problems Solved to Optimality (Within 300 sec.) for 10 Activities and Varying RT

<u>Comparison to Integer Programming</u> As previously discussed, no other specialized algorithm for solving the MRCPSP-GPR/EXP exists in the literature, leaving only general IP solvers available for project scheduling. This new algorithm was tested against a leading commercial IP solver, IBM's Optimization Solutions Library (OSL). OSL has the benefit of exploiting special ordered sets of variables (SOS variables).

The same 1440 five-activity instances and 288 ten-activity instances used for testing the bounding rules were used to compare the new algorithm against OSL. Of the 1440 five-activity instances, OSL solved 1405 to completion within a 15-minute time limit. Of the remaining 35 instances (2.4%) which exceeded the maximum allowed 15 minutes of CPU time, ten were allowed to run for 2 hours each without successfully completing. On average, the Scheduler solved the 1440 test instances in 0.002 the time it took OSL to solve the 1405 (see Table 5-8). Recall, though, that the 1440 instances that the Scheduler solved included the 35 instances which were too difficult for OSL to solve in 15 minutes.

Table 5-8. Scheduler vs. OSL Solution Time (seconds) for 5 Activities

	Solution Time (seconds)				Ave Time as
Rule	Min	Average	Max	Std Dev	% of "OSL"
OSL	0.03	1.71	63.74	5.89	100.0%
Scheduler	0.00	0.00	0.95	0.03	0.2%

When the Scheduler and OSL were compared against the 288 ten-activity problems (Figure 5-9), OSL failed to solve 31 instances (10.8%) within a 12-hour time limit. Comparing the instances OSL did solve to the Scheduler results, the Scheduler still solved the problem instances in 5.2% of the time required by OSL.

Table 5-9. Scheduler vs. OSL Solution Time (seconds) for 10 Activities

	Solution Time (seconds)				Ave Time as
Rule	Min	Average	Max	Std Dev	% of "OSL"
OSL	0.10	326.72	29460.57	1957.34	100.0%
Scheduler	0.00	17.09	872.71	88.35	5.2%

Taking a closer look at the Scheduler versus OSL for solving ten-activity projects, consider the impact of RT. RT had a particular impact on the relative solution times of the Scheduler and OSL (see Table 5-10 and Figure 5-5). The higher the Restrictiveness (the easier the underlying network), the more the Scheduler improved solution time. For totally unrestricted networks (RT = 0.0), the Scheduler was only about three times as fast as OSL. For increasingly restricted networks, the Scheduler considerably decreases solution time The results of this analysis confirm the literature that general IP solvers are not usually as efficient solving project scheduling problems as specialized algorithms.

Table 5-10. Scheduler vs. OSL Improvement by Restrictiveness for 10 Activities

RT	OSL	Sub	Improvement
0.0	131.260	44.252	0.337
0.5	188.008	13.379	0.071
1.0	698.890	0.017	0.000
Total	326.723	17.085	0.052

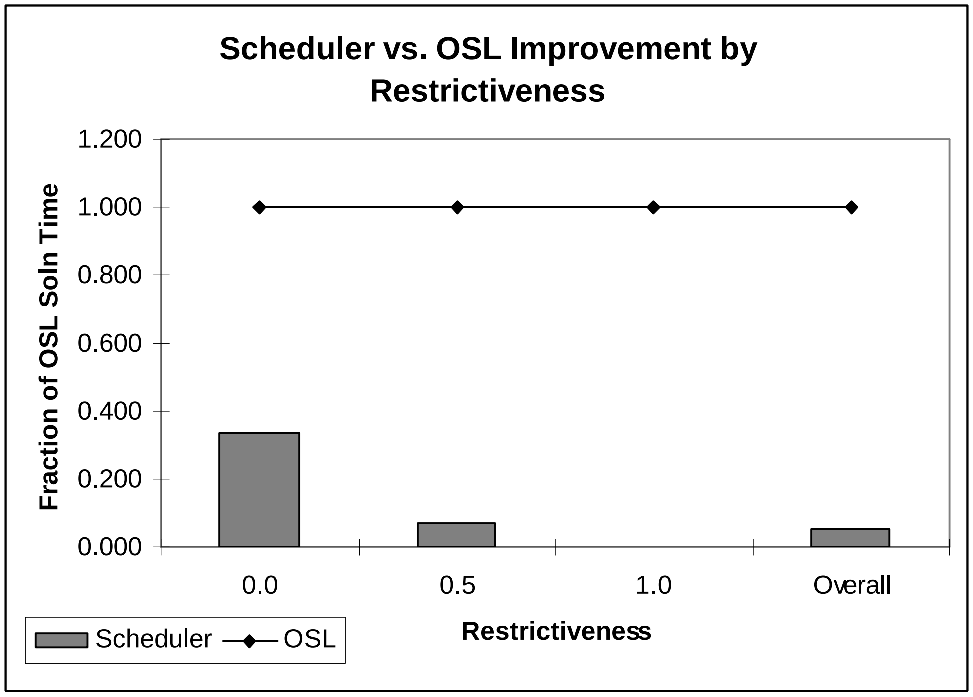


Figure 5-5. Scheduler vs. OSL Improvement by Restrictiveness for 10 Activities

<u>Solution Results vs. Key Parameters</u>. Attention now turns to the question of how key parameters affect solution results. To answer this question, the *full* test design in Table 5-11 was used.

Table 5-11. *Full* Test Design

PARAMETER	LEVELS				
Number of Modes Per Activity	1	3			
Job Duration, Maximum	10	20			
Lag Fraction	0.00	0.20			
Project Network Restrictiveness	0.00	0.25	0.50	0.75	1.00
Number of Renewable/Nonrenewable Resources	1	3			
Renewable/Nonrenewable Resource Factor	0.50	1.00			
Regular Renewable/Nonrenewable Resource Strength	0.00	0.50	1.00		
Total Renewable/Nonrenewable Resource Strength	0.00	0.50	1.00		
Total Combinations = 960					

One problem instance was generated at each of the 960 design points for projects with 10, 20, and 30 activities. Additionally, one instance at each of the 576 design points in the *minimal* test design (Figure 5-12) was generated for projects with 50 activities.

Table 5-12. *Minimal* Test Design

PARAMETER	LEVELS		
Number of Modes Per Activity	1	3	
Job Duration, Maximum	10	20	
Lag Fraction	0.00	0.20	
Project Network Restrictiveness	0.00	0.50	1.00
Number of Renewable/Nonrenewable Resources	1	3	
Renewable/Nonrenewable Resource Factor	0.50	1.00	
Regular Renewable/Nonrenewable Resource Strength	0.50	1.00	
Total Renewable/Nonrenewable Resource Strength	0.00	0.50	1.00
Total Combinations = 576			

Each of the problem instances was solved using the Scheduler with a maximum time limit of 20 seconds. Since the objective of this experiment was to take a broad view of *solvability* as a function of key parameters, the 20-second time limit was selected to control the total time required to solve the 3456 test problems. Table 5-13 shows the overall results, listing the number of problem instances which were infeasible, the number which exceeded the 20-second time limit, and the number solved to optimality. Figures 5-6 and 5-7, chart the number of occurrences and relative percentage of each result, respectively. The reason for these results was investigated further.

Table 5-13. Solution Results

	JOBS				JOBS				
RESULT	10	20	30	50	10	20	30	50	Total
Infeasible	85	134	156	97	8.9%	14.0%	16.3%	16.8%	472
Over 20s Limit	203	429	515	330	21.1%	44.7%	53.6%	57.3%	1477
Optimal	672	397	289	149	70.0%	41.4%	30.1%	25.9%	1507
Total	960	960	960	576	100.0%	100.0%	100.0%	100.0%	3456

Consider first the infeasible problems. Though an in-depth discussion of the infeasible problems has little bearing on the effectiveness of the Scheduler, it does provide worthwhile insights into the nature of the MRCPSP-GPR/EXP.

Kolisch *et al.* (1995) report that a low availability of resources can lead to infeasible problem generation. The results in Table 5-14 confirm this conclusion, where a RS (regular plus expediting) of zero accounts for 68% of infeasible problems overall and an RS of 0.50 accounts for 32%.

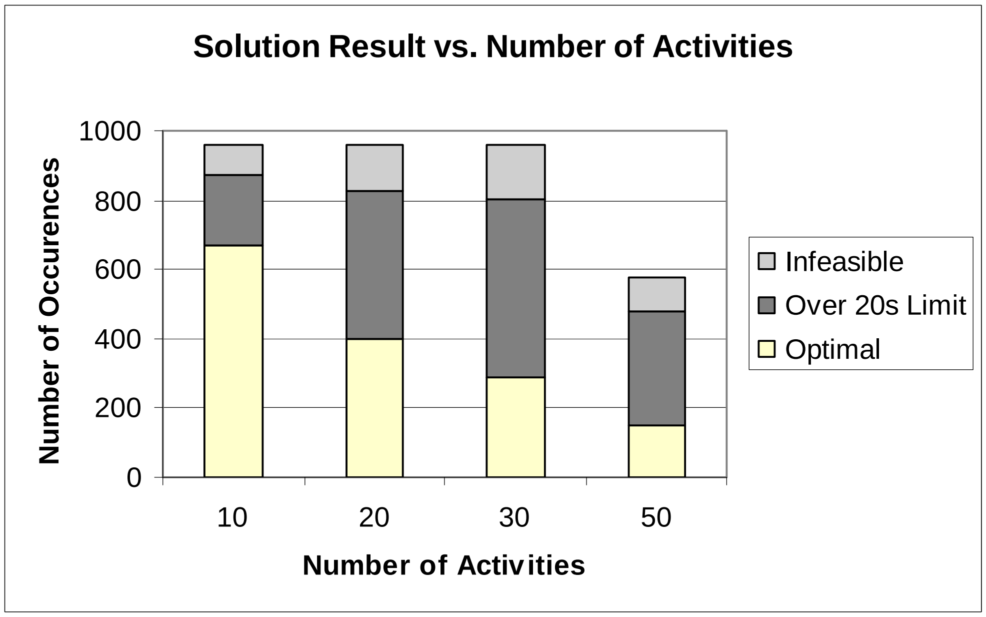


Figure 5-6. Solution Results as Occurrences

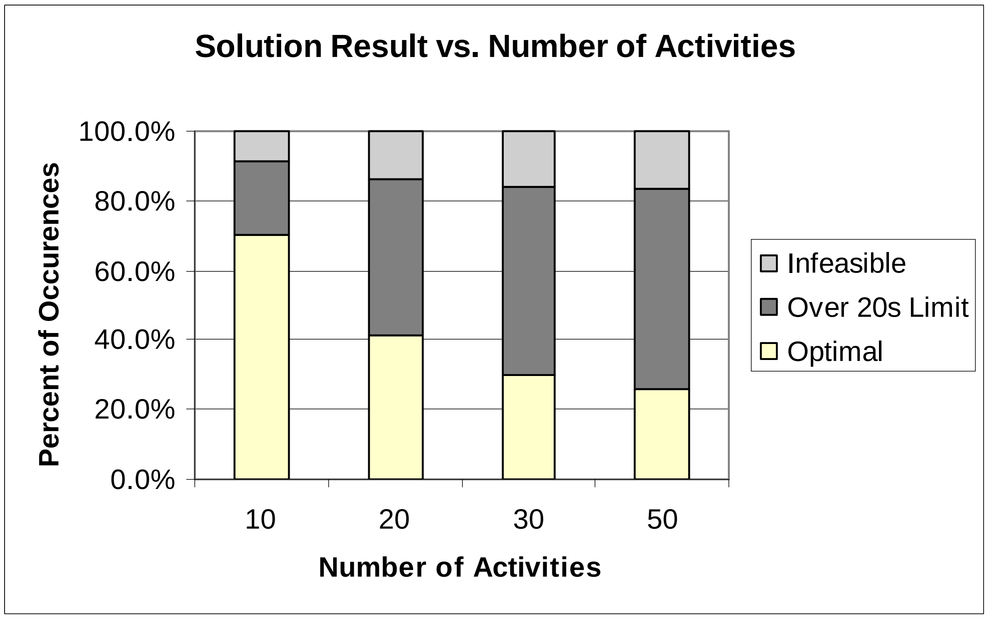


Figure 5-7. Solution Results as Percentages

Table 5-14. Infeasible Problems

	Activities					
RS	10	20	30	50	Total	Percent
0.0	70	94	99	59	322	68.22%
0.5	15	40	57	38	150	31.78%
1.0	0	0	0	0	0	0.00%
Total	85	134	156	97	472	100.00%

Figure 5-8 shows the percentage of infeasibilities accountable to each level of RS for activities of different size.

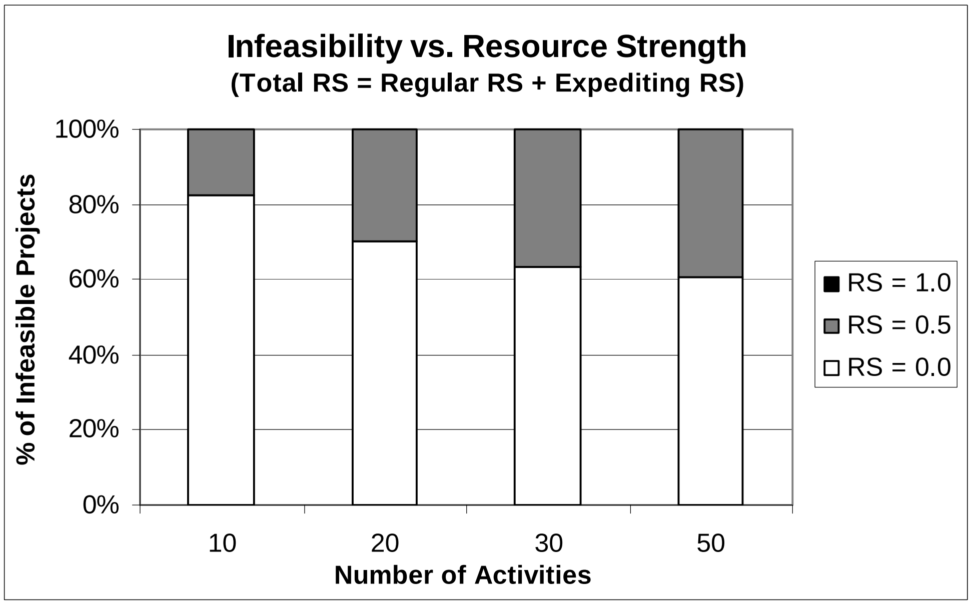


Figure 5-8. Infeasible Problems vs. Resource Strength

The increase in the number of infeasible problems as a function of RS is compounded by both the number of modes in the project and the percent of activities with generalized precedence. Table 5-15 shows that three-mode projects account for 65% of the infeasibilities, while single-mode projects account for only 35 %. When modes and RS are considered together, three-mode projects with a RS of zero account for 51% of the infeasibilities. Figure 5-9 also depicts the relationship of RS and mode in infeasibilities. Note that the chart includes all infeasible instances; therefore, the sum of the two columns adds to 100%.

Table 5-15. Infeasibilities by RS and Mode

	MODES		MODES			
RS	1	3	1	3	Total	Percent
0.00	81	241	17.2%	51.1%	322	68.22%
0.50	83	67	17.6%	14.2%	150	31.78%
1.00	0	0	0.0%	0.0%	0	0.00%
Total	164	308	34.7%	65.3%	472	100.00%

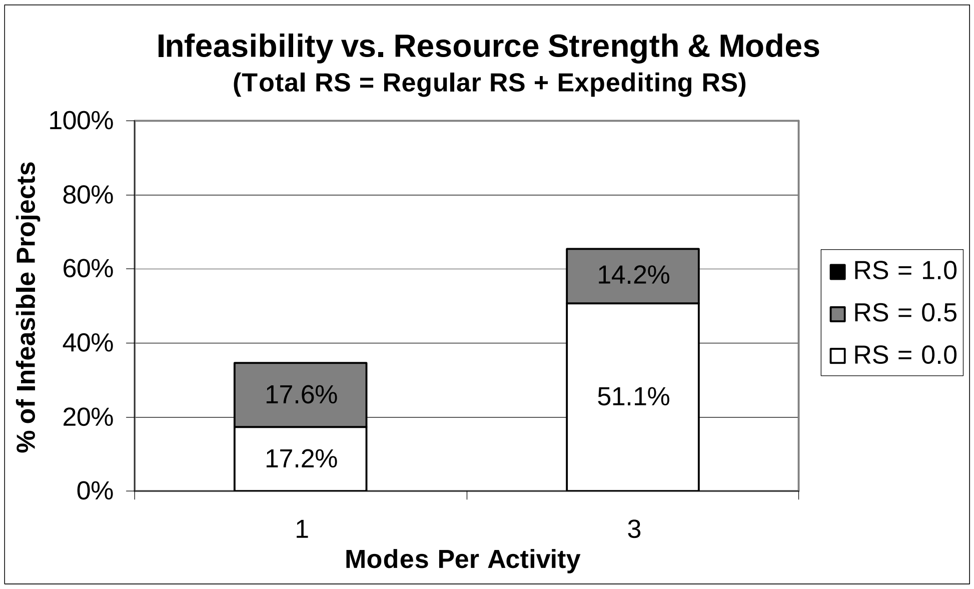


Figure 5-9. Infeasible Problems vs. RS and Mode

Table 5-16 shows that projects in which 20% of the activities have generalized precedence account for 76% of the infeasibilities, while projects with only standard finish-start precedence account for only 24%. When generalized precedence and RS are considered together, problems where 20% of activities have generalized precedence and where RS is zero account for 45% of the infeasibilities (depicted also in Figure 5-10).

Table 5-16. Infeasibilities by RS and Percent of Activities with Generalized Precedence (GPR)

	GPR %		GPR %			
RS	0%	20%	0%	20%	Total	Percent
0.00	108	214	22.9%	45.3%	322	68.22%
0.50	5	145	1.1%	30.7%	150	31.78%
1.00	0	0	0.0%	0.0%	0	0.00%
Total	113	359	23.9%	76.1%	472	100.00%

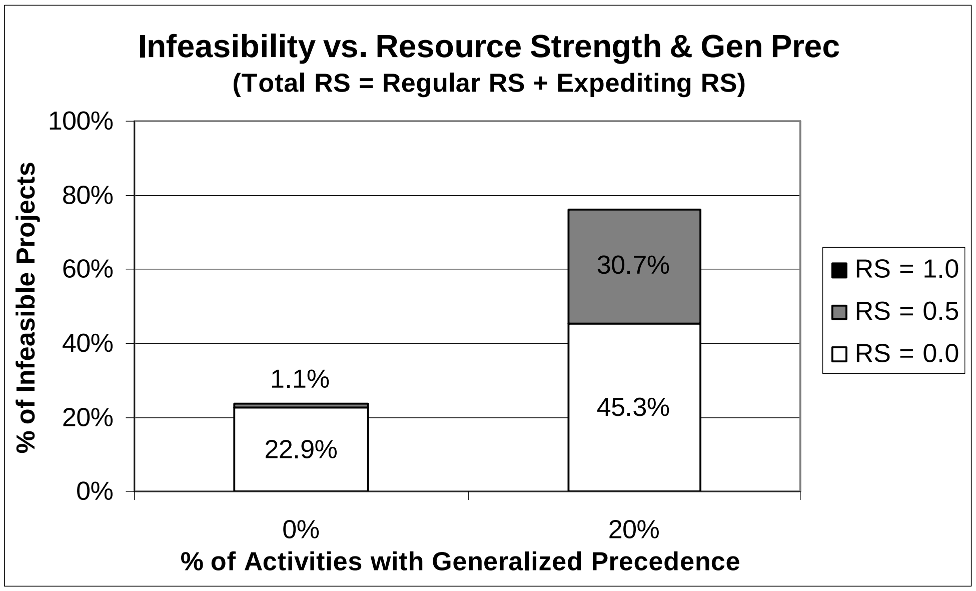


Figure 5-10. Infeasibilities vs. RS and Percent of Activities with GPR

Consider, next, the problems which are not solved within the 20-second time limit. Figure 5-11 shows the total number of problems which exceeded the 20-second time limit versus network Restrictiveness. Since the total number of feasible problems was different for each project size, the same data is presented in Figure 5-12, standardized as the percentage of problems exceeding the time limit attributable to each level of RT. Note that Restrictiveness does not appear to play as important a role as it did in the number of problems which were infeasible. As the number of activities in the problem increases, so does the percent of problems with an RT of 1.0 exceeding the time limit. However, the percentages attributable to the other levels of RT remain fairly proportional in relation to each other. For example, going from problems with 20 activities to those with 30 activities shows that of the problems which exceed the 20-second limit, the percent attributable to an RT of 1.0 increases from 6.1% to 8.5%. Although the percentage attributable to an RT of 1.0 reduces the absolute percentages attributable to the other RT levels, an RT of 0.25 still attributes between 64-69% of what an RT of 0.0 attributes, an RT of 0.5 still attributes between 63-70% of what an RT of 0.0 attributes, and an RT of 0.75 still attributes between 58-64% of what an RT of 0.0 attributes.

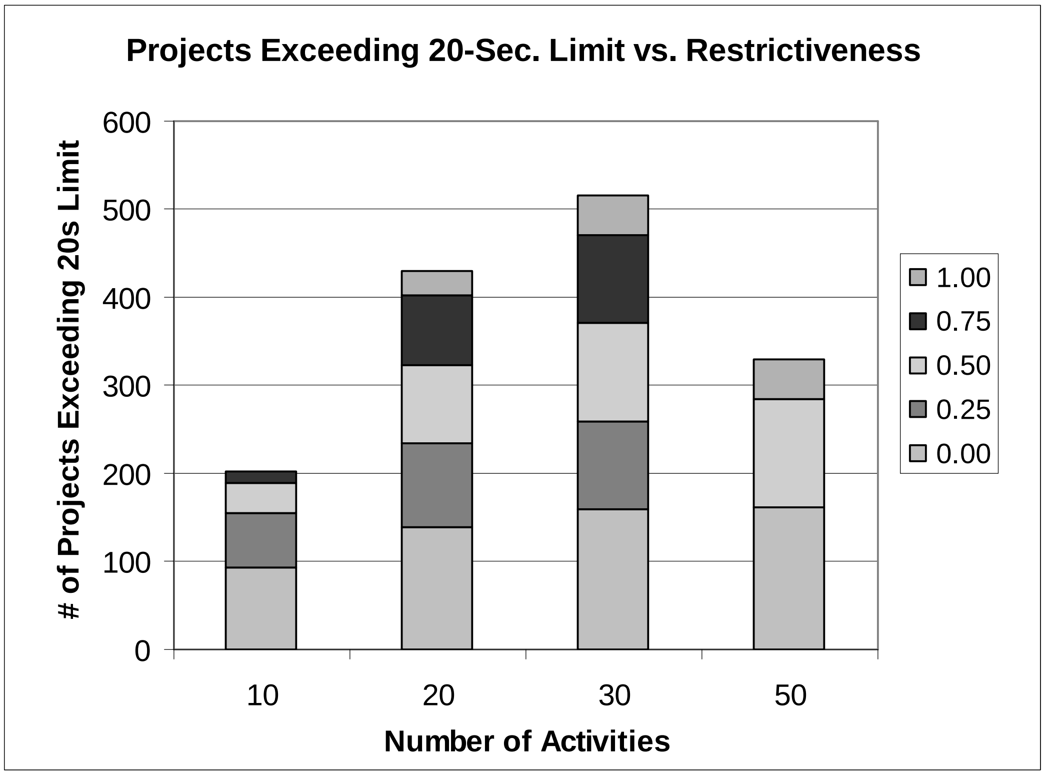


Figure 5-11. Solution Time Exceeding 20 Seconds vs. Restrictiveness (Occurrences)

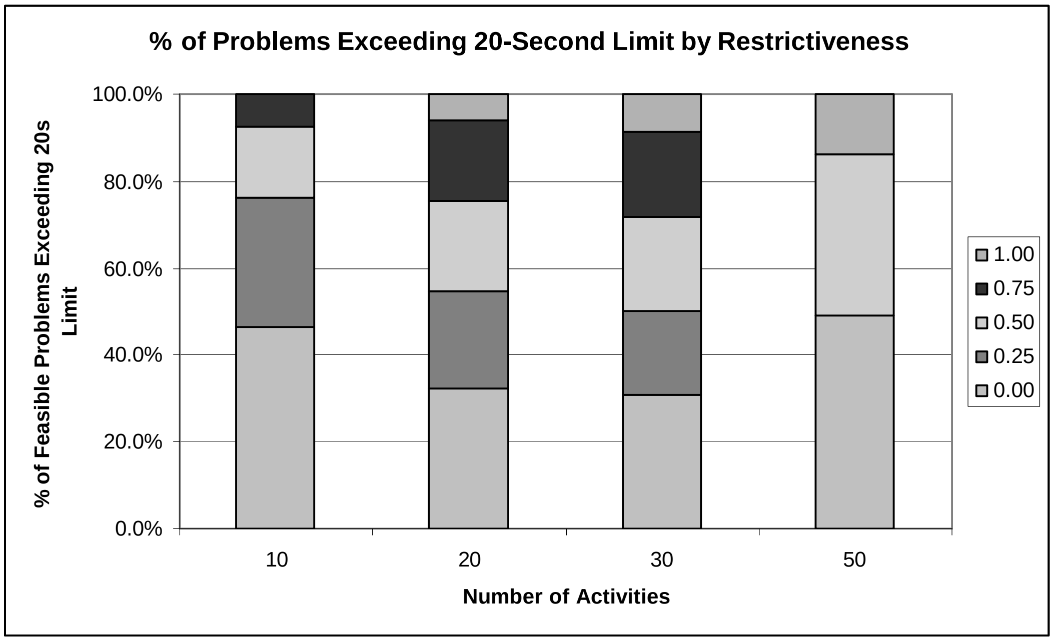


Figure 5-12. Solution Time Exceeding 20 Seconds vs. Restrictiveness (Percentages)

Though restrictiveness is only moderately important to the number of activities which do not solve in the 20-second time limit, resource strength is a significant factor. Figure 5-13 shows the percentage of feasible problems which exceed the time limit versus RS. The *x*-axis is divided by the RS of regular resources, with multi-shaded columns representing the different levels of RS of expediting resources (ERS). There are a total of six columns representing the combinations of regular and expediting resources. Problems were generated to have a total RS of at most 1.0. Therefore, there is no column corresponding to an ERS of 1.0 when the RS is 0.5 or 1.0. Nor is there a column corresponding to an ERS of 0.5 when RS equals 1.0.

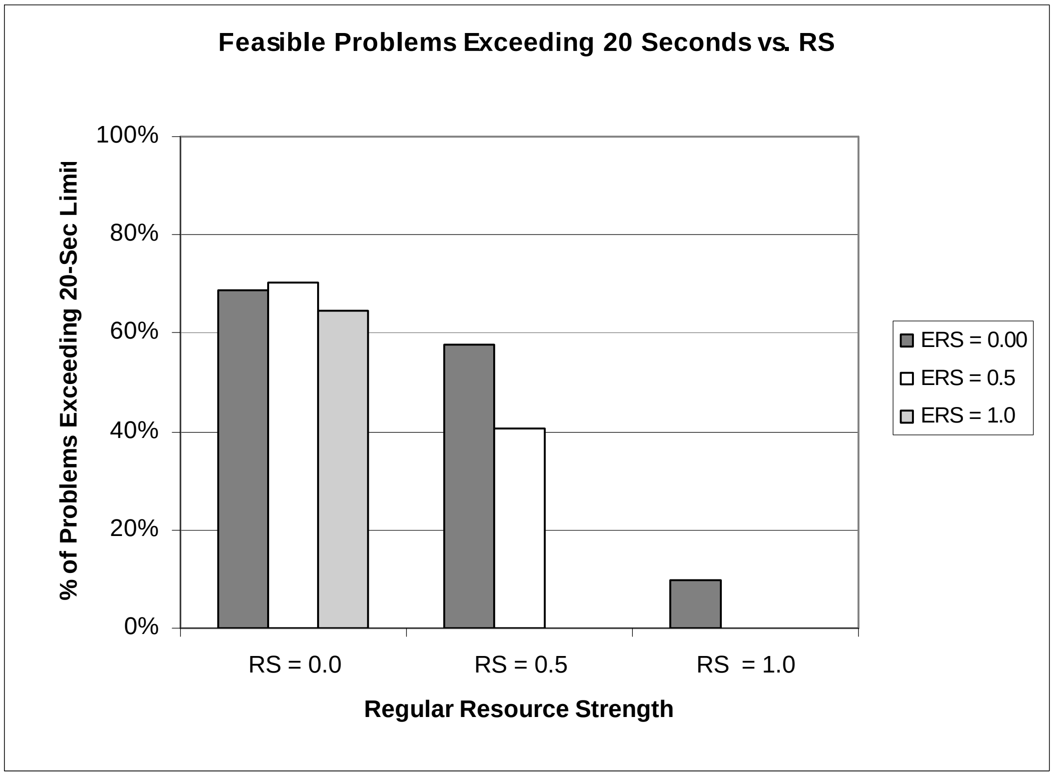


Figure 5-13. Solution Time Exceeding 20 Seconds vs. RS

Note that when the regular RS is 0.0, the impact of expediting RS is negligible, with roughly the same percentage of problems exceeding the time limit. On the other hand, when RS = 0.5, including additional resources improved to some degree the number of problems which were solved within the time limit. This effect is likely a result of the way the Scheduler enumerates solutions. Recall that the Scheduler attempts first to schedule activities at their early start times and then at progressively later times. The existence of expediting resources generally makes more time-compressed schedules feasible. Since the more time-compressed schedules are enumerated early on, if the cost savings from a shorter project outweighs the cost of the expediting resources, the schedules found early on provide tighter bounds on the optimal solution and, thus, allows quicker fathoming of unproductive partial schedules. Quicker fathoming, in turn, leads to faster completion of the algorithm. Finally, most problems with an RS = 1.0, the easiest ofthe problems, can be solved within the time limit.

Consider the feasible problems which are solved within the 20-second time limit. Figure 5-14 shows the relative *solvability* of the levels in each of the two-level factors. The *x*-axis shows the two-level factors: modes, duration (Dur), number of renewable / nonrenewable resources (#Res), resource factor (RF), and percent of activities with GPR (Lag%). Along with the factors are listed both levels: respectively termed the 1*st* level and the 2*nd* level. The *y*-axis shows the number of problems solved to optimality as a ratio of the 1*st* factor to the 2*nd* factor. The columns represent problems with 10, 20, 30, and 50 activities each.

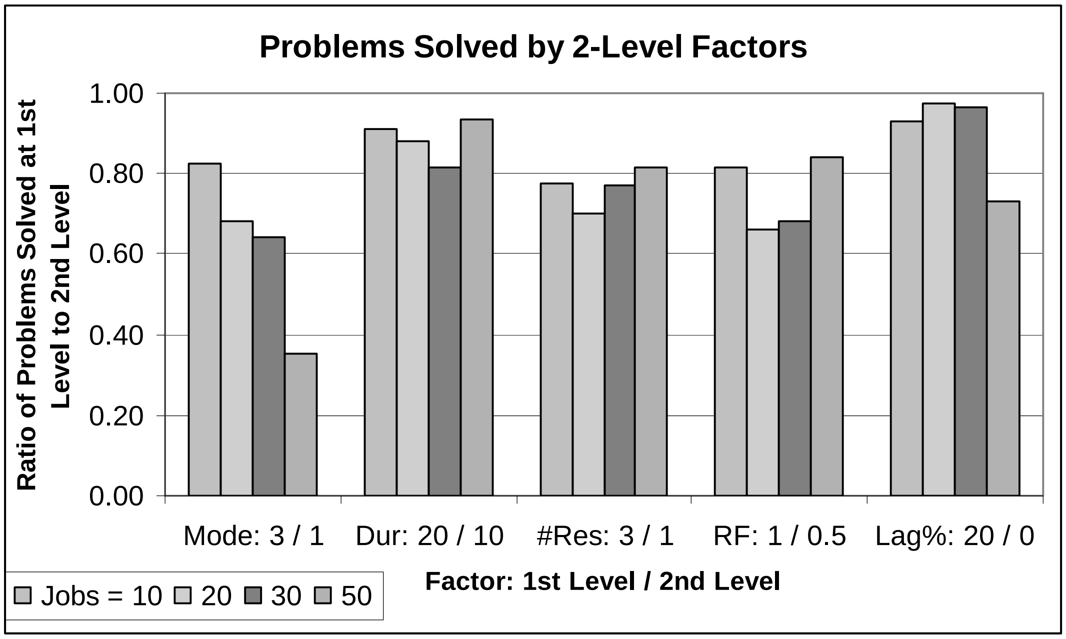


Figure 5-14. Problems Solved by 2-Level Factors

As an example, consider the number of modes per activity. An activity may have either three modes (the 1*st* level) or one mode (the 2*nd* level). Focusing just on problems with ten activities, note that the light gray column above the label *Mode: 3 / 1* indicates a value of 0.83. This value reflects that the number of ten-activity problems solved to optimality in which each activity has three modes is 83% of the number of problems solved in which activities have only one mode. Hence, problems with ten activities and three modes per activity are somewhat more difficult than similar problems with only one mode per activity. This is really no surprise since the size of the problems grows as the number of modes per activity increases.

Having reviewed the chart, note that modes per activity is the most clearly influential two-level factor on the ability to solve problems. As problem size (*i.e.*, activities) increases, the impact of modes also increases. Trends in the other factors are not quite so clear, but it is evident that all of the factor levels identified as the 1*st* level are at least more difficult than levels identified as the 2*nd* level. This is not surprising since the 1*st* and 2*nd* levels were identified so that the theoretically more difficult level was the first level; thus, maintaining ratios below 1.0.

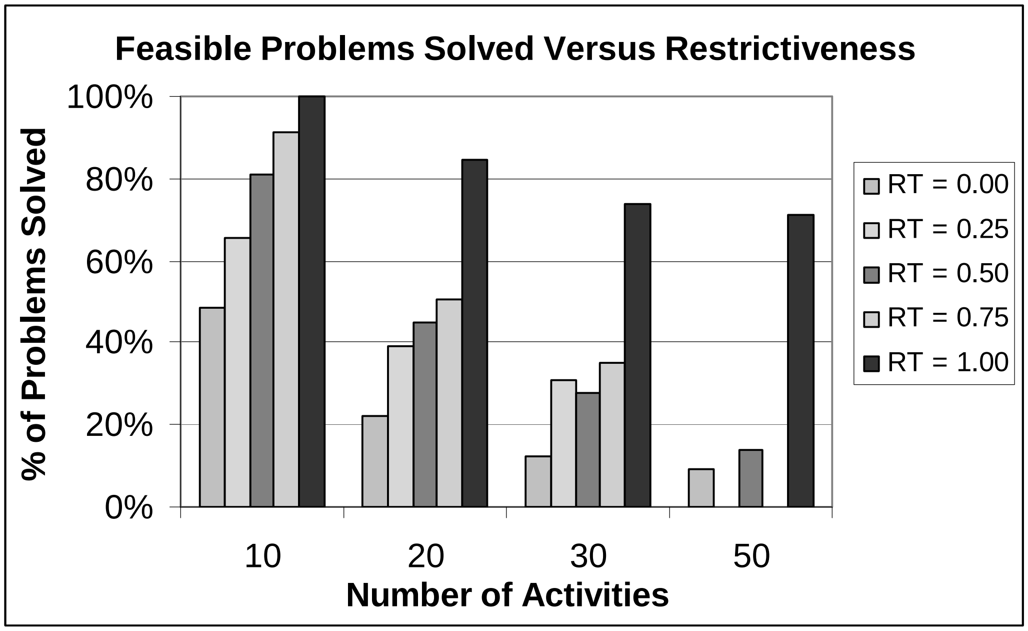


Figure 5-15. Problems Solved Versus Restrictiveness

Turning now to the factors with more than two levels, Figure 5-15 shows a very clear trend in the impact of network restrictiveness on the percentage of feasible problems solved within the 20-second time limit. As RT increases (*i.e.*, the network structure becomes more constrained), the percentage of problems solved increases as well. The Scheduler performed very well on problems with relatively high RT values, even for the problems with 50 activities. An RT of 0.0, by contrast, makes a problem much more difficult and relatively few of these problems (especially in projects with over ten activities) solved to optimality within 20 seconds.

An analysis of resource strengths also provides some interesting insights (see Figure 5-16). The easiest problems are those with a regular RS of 1.0. These are problems where enough free (*i.e.*, no cost) resources are available to schedule every activity at its early start time (the GCPM schedule). For problems with ten activities, there is a near linear increase in the percentage of problems solved as regular RS increases. For any level of RS, the percentage of problems solved also increases in near-linear fashion for increasing ERS.

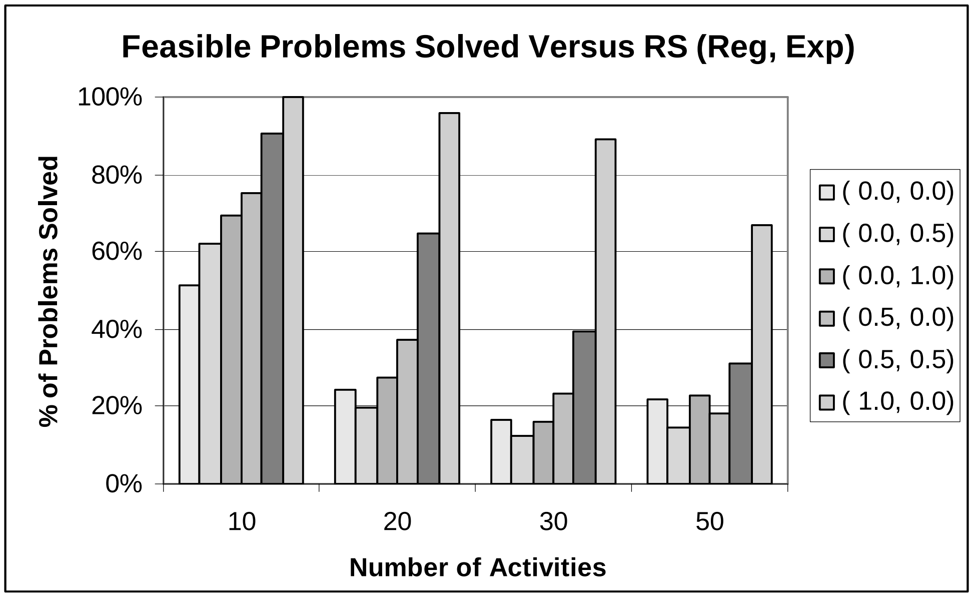


Figure 5-16. Problems Solved Versus Resource Strength

For problems with more than ten activities, a different phenomenon presents itself. While the above observations (those of increasing RS yielding an increasing percentage of problems solved) hold true for resource strengths of 0.5 and 1.0, this is not the case for RS of 0.0. When the RS is 0.0, an ERS of 0.5 provides fewer solved problems than an ERS of 0.0. This is contrary to the aforementioned trends. This apparent aberration may be explained by the tradeoff between computational overhead and upper bounding of the solution. The more expediting resources the Scheduler has to trade, the more overhead required to account for resources and their costs. Thus, given some fixed RS, the problems solved should decrease as ERS increases. On the other hand, the Scheduler searches for schedules beginning with earlier activity start times and continuing to progressively later start times. When ERS is high, more schedules with relatively early start times become feasible, allowing for a good upperbound on the objective function to be found early in the search. The upper bound allows faster fathoming of unproductive partial schedules, resulting in faster solution times. Thus, given some fixed RS, the problems solved should increase as ERS increases. Characteristics of the problem itself and of the Scheduler may be driving solution time, and consequently the number of problems solved, in opposing directions. Defining this tradeoff in greater detail may be worth further investigation.

<u>Solution Time</u>. Having reviewed the impact of key parameters on the problem results (*i.e.*, feasibility and tractability, defined as solvable in 20 seconds or less), consider now in more detail the solution times required by the Scheduler. A discussion of those problems which were solved in 20 seconds or less is provided first. Results are, then, reported on a subset of problems which were allowed to solve without time limit. The same test set used in the previous subsection is used here.

Figures 5-17, 5-18, and 5-19 show the cumulative number of feasible problems solved, broken out by time bin. The *x*-axis shows the time bins, which are 0.01, 0.1, 1, 10, and 20 seconds. If the time bins were labeled T^1 through T^5, respectively, then a problem falls into time bin, T^i, if it took longer than T^{i-1} to solve but no more than T^i. For instance, a problem which took 0.06 seconds to solve falls into time bin T^2, 0.1.

Figure 5-17 presents the cumulative number of feasible problems solved by number of project activities (or jobs), Figure 5-18 by RT, and Figure 5-19 by RS. The most noteworthy observation is that, in most cases, the number of problems solved in the first 0.01 seconds comprises at least 50% of all problems solved in 20 seconds or less. For example, note in Figure5-17 that for problems with 10 activities, 77% of problems solved within the 20-second time limit, while 48% of problems solved within 0.01 second. Therefore, 62% of problems which solved within the 20-second time limit did, in fact, solve within 0.01 second. Some insights into this result are provided in the next subsection.

Table 5-17 shows solutions times for a subset of 10- and 50-activity projects where no time limit was imposed. The subsets come from the problems which previously required more than 20 seconds to solve. A total of 146 problems with 10 activities were solved to optimality. As reported above, 203 problems exceeded the previous 20-second time limit. A few of the most difficult of these problems were not solved to optimality in this experiment due to the excessive solution time. One of these problems, a problem with three modes per activity, an RT of 0.0, three renewable and three nonrenewable resources, an RF of 1.0, an RS of 0.0, and an ERS of 1.0, was terminated without completionafter 302,800 seconds (over 84 hours). Problems of similar difficulty were, therefore, not attempted. In all cases, the problems expected to take a similarly long time to solve had an RT of 0.0 (a totally unconstrained network), an RF of 1.0 (every activity requiring every resource), and an RS of 0.0 (so few regularly available resources as to eliminate the possibility of scheduling any two activities to be in progress at the same time without incurring an expediting resource cost, provided any expediting resources were even available).

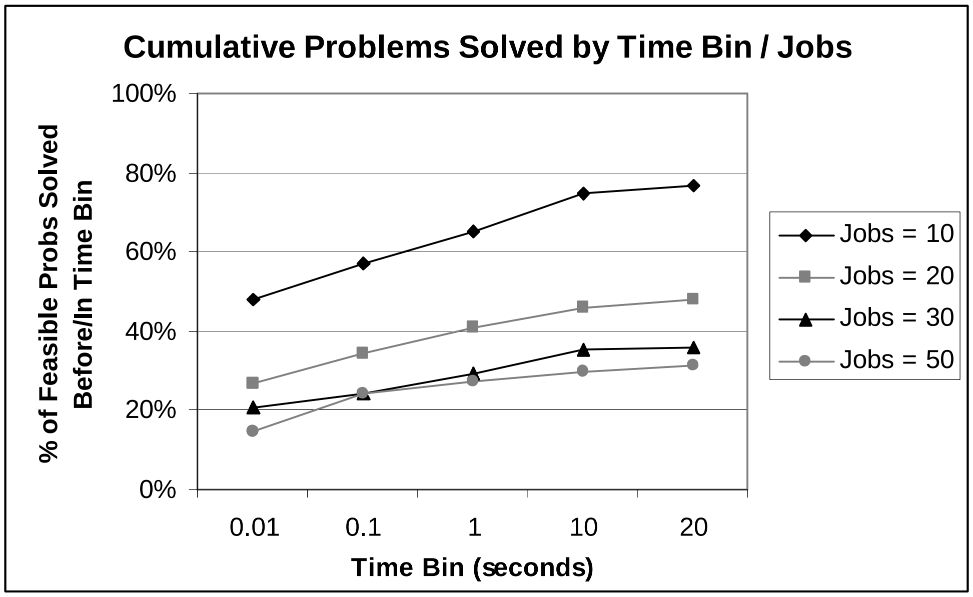


Figure 5-17. Cumulative Problems Solved by Time Bin and Jobs

Of the 10-activity problems solved to optimality, the average solution time was just over 32 minutes, with a minimum time of 20.2 seconds and a maximum time of just over 19 hours. Twenty-three of the 146 problems required longer than the average solution time.

Two projects with 50 activities were also solved to optimality. Both problems had three modes per activity, an RT of 0.5, three renewable and three nonrenewable resources, and an RF of 0.5. The problems differed only in their resource strengths. One problem had an RS of 1.0 with an ERS of 0.0 (no expediting resources, but sufficient regular resources to schedule all activities at their early start time). This problem required 88.2 seconds to solve. The other problem had an RS and an ERS both equal to 0.5. This problem required 10,994.7 seconds (a little over 3 hours) to solve to optimality.

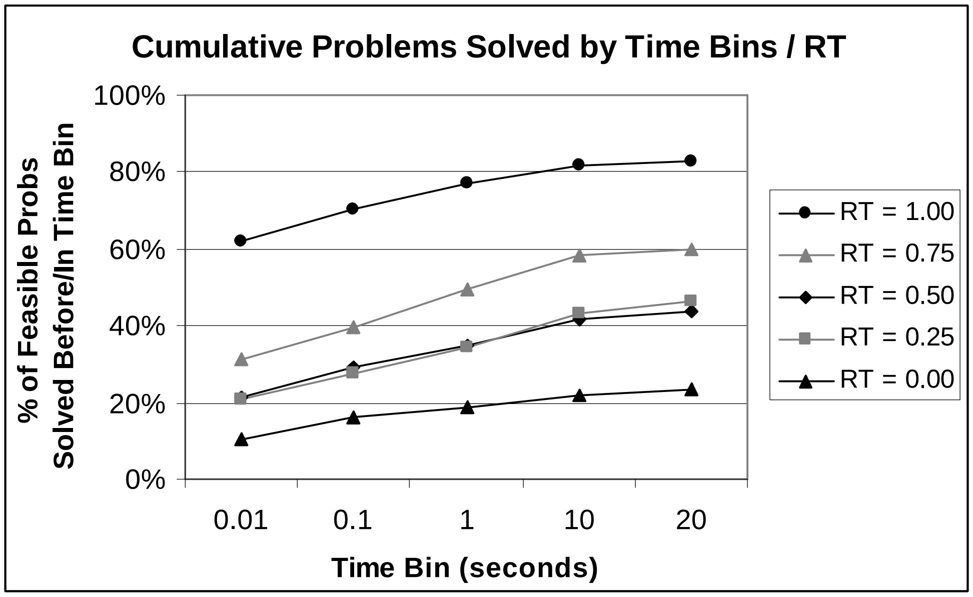


Figure 5-18. Cumulative Problems Solved by Time Bin and RT

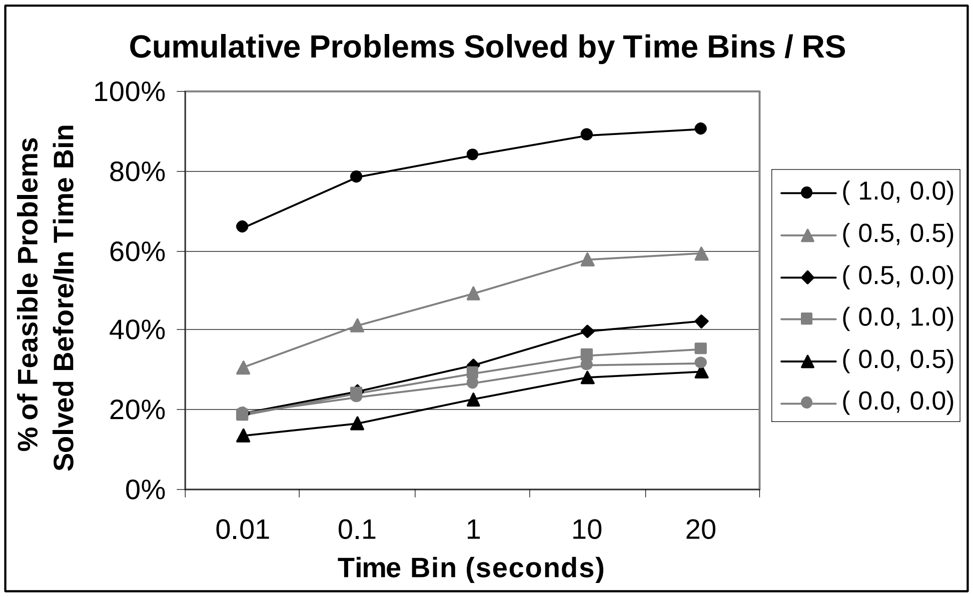


Figure 5-19. Cumulative Problems Solved by Time Bin and RS

Table 5-17. Solution Time for 10- and 50-Activity Projects

	Solution Time (seconds)				
Activities	Count	Min	Average	Max	Std Dev
10	146	20.2	1920.6	69356.2	7495.4
50	2	88.2	5541.5	10994.7	5453.2

Time to Optimal Solution. In the previous subsection, it was shown that generally more than half of all problems solved within the 20-second time limit were, in fact, solved within the first 0.01 second. This result is understood by focusing on the time it took for the Scheduler to find an optimal solution compared to the time it took to complete the solution process. As each problem was solved, any time the Scheduler found a solution better than the incumbent best solution, the time this solution was found was recorded. When the solution process was completed, then, not only was the total solution time reported, but the time required to find the optimal solution was also reported. The difference between the time for the entire solution process and the time to find the optimal, therefore, is the time required to verify that the optimal is indeed optimal. Ideally, any enumeration scheme finds a good solution, or upper bound, early in the process to enable quicker fathoming of unproductive branches. No upper bound is better, of course, than an optimal solution.

The *x*-axis of Figure 5-20 is divided into the completion time bins used previously (*i.e.*, the time bins used to divide the completion times, not the times to optimal). For the set of problems completing within each of the completion time bins, the times it took to find an optimal solution to each problem in the set were averaged. These averages are reflected on the *y*-axis. For instance, for problems with ten activities which took at least ten seconds to solve but no more than 20, an optimal solution was found, on average, in just under six seconds.

The results in Figure 5-20 are not the most revealing, however. If the time required to find an optimal solution are also binned and, then, compared to the completion time bins, a much clearer picture is presented. Figure 5-21 presents this picture. Note that for the vast majority of problems, an optimal solution was found in no more than 0.01 second. In some cases, though, it still took up to 20 seconds to complete the algorithm.

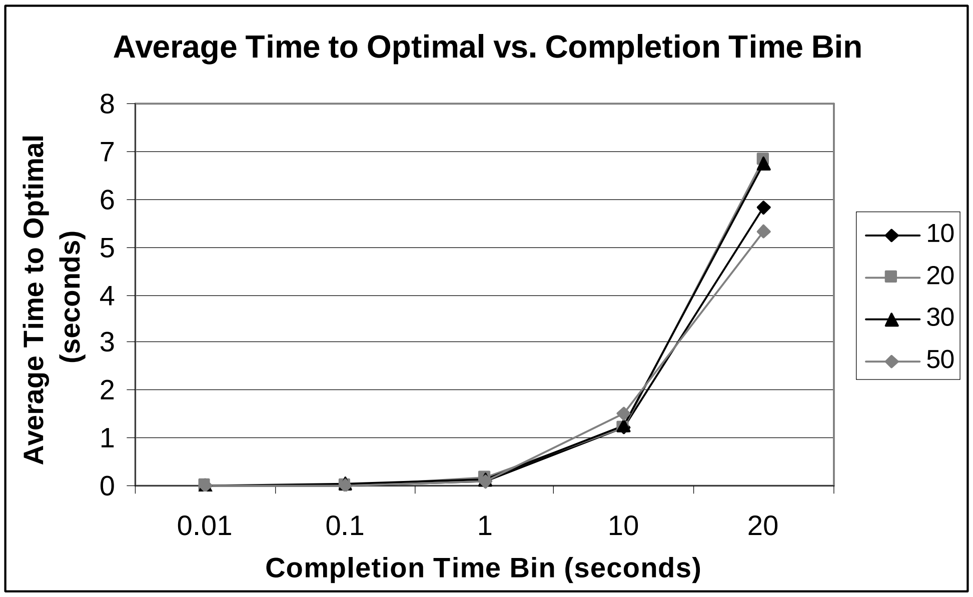


Figure 5-20. Average Time to Optimal Versus Completion Time Bin

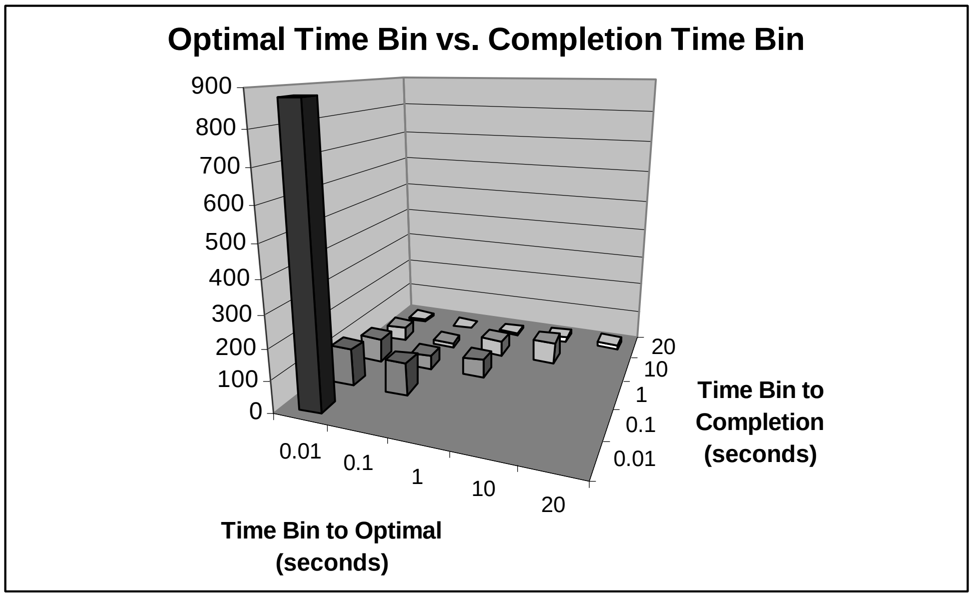


Figure 5-21. Optimal Time Bin Versus Completion Time Bin

Figure 5-22 presents the data in another way, showing the time bins to optimal for the different size problems (*i.e.*, number of activities). As expected, the smaller the problem, the sooner the Scheduler can be expected to find an optimal.

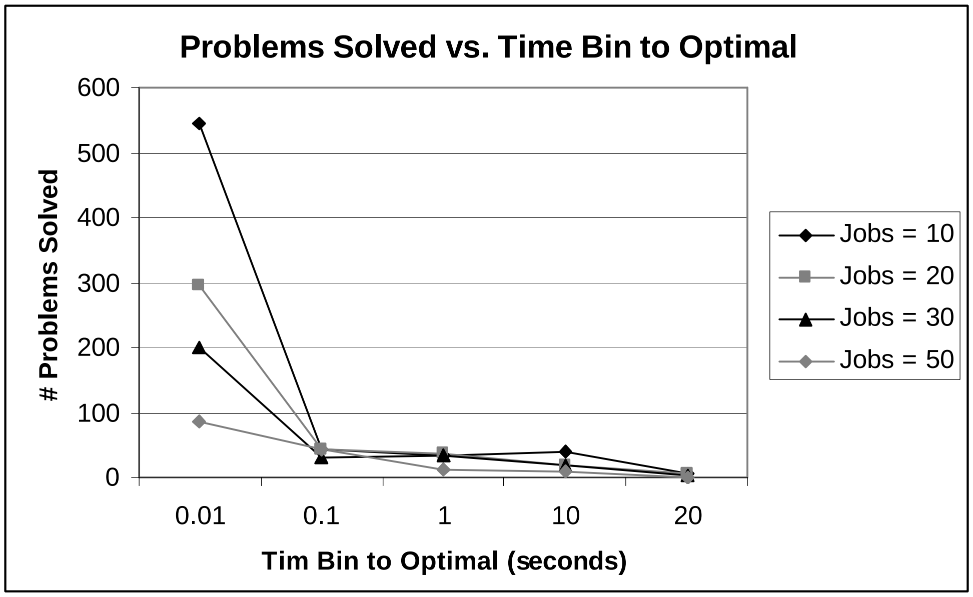


Figure 5-22. Problems Solved Versus Completion Time Bin

Taking a look at the time to find an optimal solution versus RT (Figure 5-23) shows that, as expected, the higher the RT (and, hence, the easier the problem), the sooner the Scheduler is expected to find an optimal.

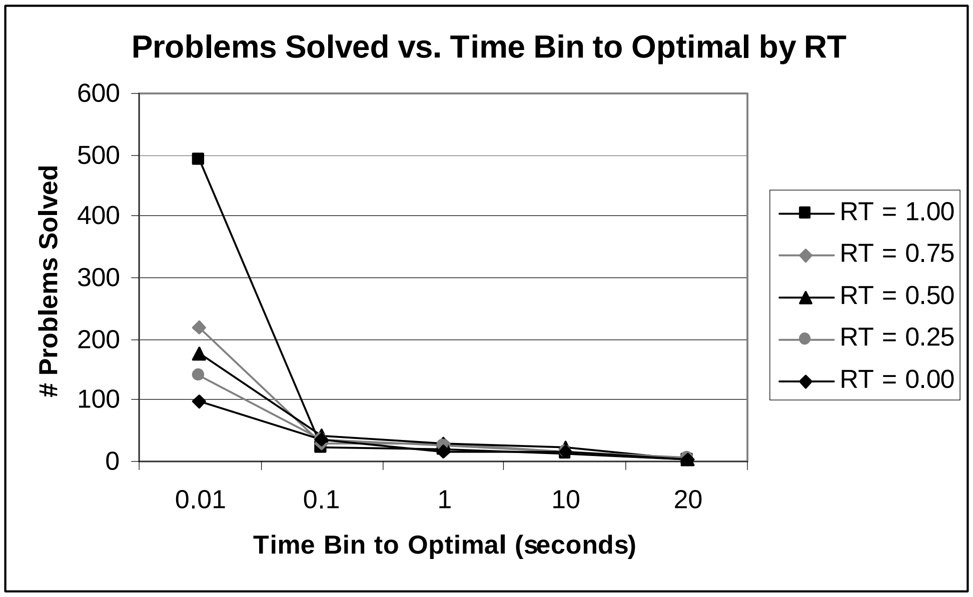


Figure 5-23. Problems Solved Versus Completion Time Bin by RT

Finally, the time to find an optimal solution can be compared to the RS (Figure 5-24). Most noteworthy, here, is that the Scheduler finds more optimal solutions in 0.01 second when (RS, ERS) equals (0.0, 1.0) than it equals (0.5, 0.0). One might expect the overhead associated with accounting for expediting resources to significantly slow down the solution process. As seen before, though, this overhead is overcome by the degree to which the expediting resources enable schedules with early start times, and their relatively good objective function values, to be feasible. It could be speculated that changing the costs of activity modes (which are start time dependent) relative to the cost of expediting resources might change this balance and lead to somewhat different results. An investigation into this hypotheses is outside the scope of this study, but may be worth future consideration.

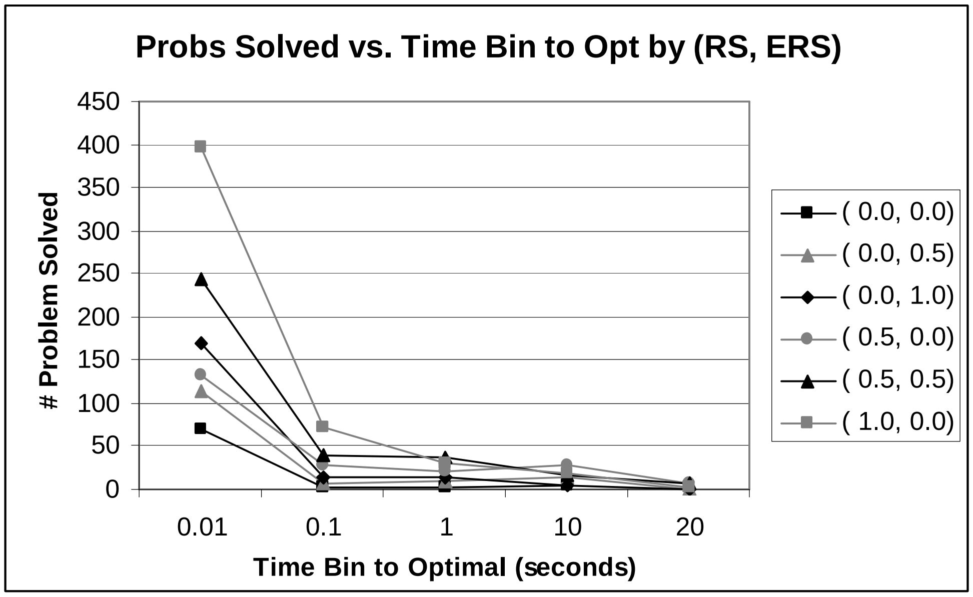


Figure 5-24. Problems Solved Versus Completion Time Bin by RS

Returning to the 10-activity problems solved to optimality without time limit reveals that, on average, the optimal was found in the first 33.3% of the solution time and that the remaining 66.7% of the time was spent verifying the optimal (see Table 5-18). Noteworthy is that for the 10-activity problem which took the longest to solve (about 19 hours), an optimal solution was actually found in the first 0.04 seconds.

Table 5-18. Time to Optimal (10-Activity Projects)

	Time to Optimal (% of Total Solution Time)			
Activities	Min	Average	Max	Std Dev
10	0.0%	33.3%	100.0%	34.4%

Turning to the 50-activity problems shows that the problem which solved in 88.2 seconds required 74.9 seconds to find an optimal (84.9% of solution time) while the problem which solved in 10,994.7 seconds (around 3 hours) required 3982.3 seconds (around an hour) to find an optimal (only 36.2% of solution time).

Completion Time vs. *k*. The scheduling algorithm developed in this chapter is used to solve the subproblems of the decomposition approach discussed in the next chapter. For the decomposition approach to work, each subproblem (or project) must be solved to find the *k*-best schedules for that project. Besides finding the *k*-best schedules for purposes of the decomposition algorithm, a scheduler may be interested in the *k*-best simply to be able to present alternatives to a decision-maker.

To assess the impact on solution time of the choice of *k*, the test set used above (with 10-, 20-, 30-, and 50-activity projects) was solved again for varying values of *k*. To do so, the problems which were solved within the 20-second time limit imposed above were resolved to find the 10, 100, and 1000 best solutions. Since it is reasonable to expect that the Scheduler should take longer to track a higher number of *best* solutions, the imposed solution time limit was increased to 60 seconds. Table 5-19 and Figure 5-25 show that for all values of *k*, most problems were solved within the 60-second time limit.

Table 5-19. Problem Solution Results for k=1, 10, 100, 1000

	k-best Solutions				
	1	10	100	1000	Total
Exceeds Limit	0	1	19	70	1567
Optimal	1507	1506	1488	1437	5938
Total	1507	1507	1507	1507	7505

Reviewing a few fundamental statistics related to the solution time reveals (in Figure 5-26) that while the maximum time required to solve the problems approaches the 60-second limit (especially for *k* = 100, 1000), the average completion times were relatively low (under seven seconds). Figure 5-26 also shows the average time to find an optimal.

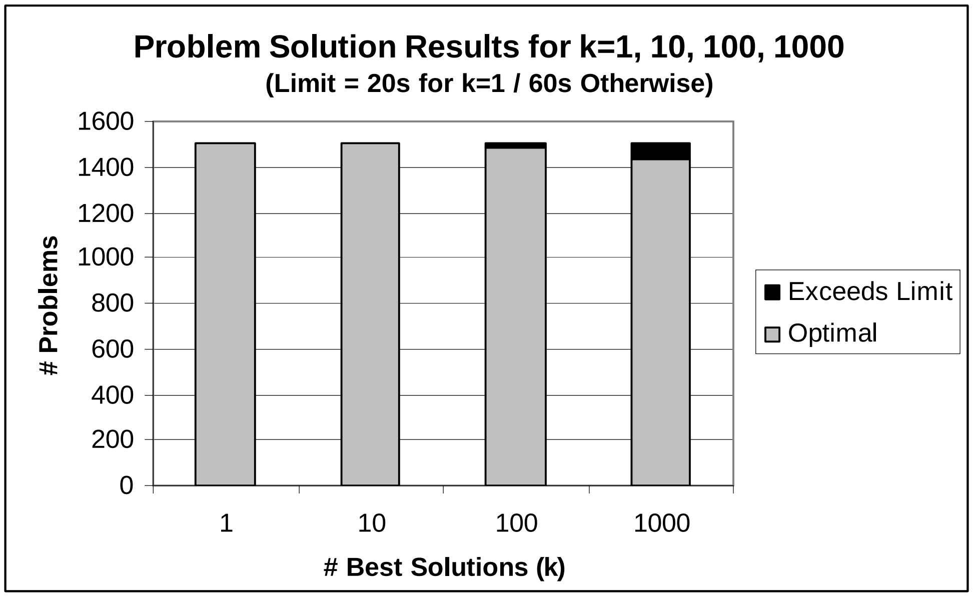


Figure 5-25. Problem Solution Results for k=1, 10, 100, 1000

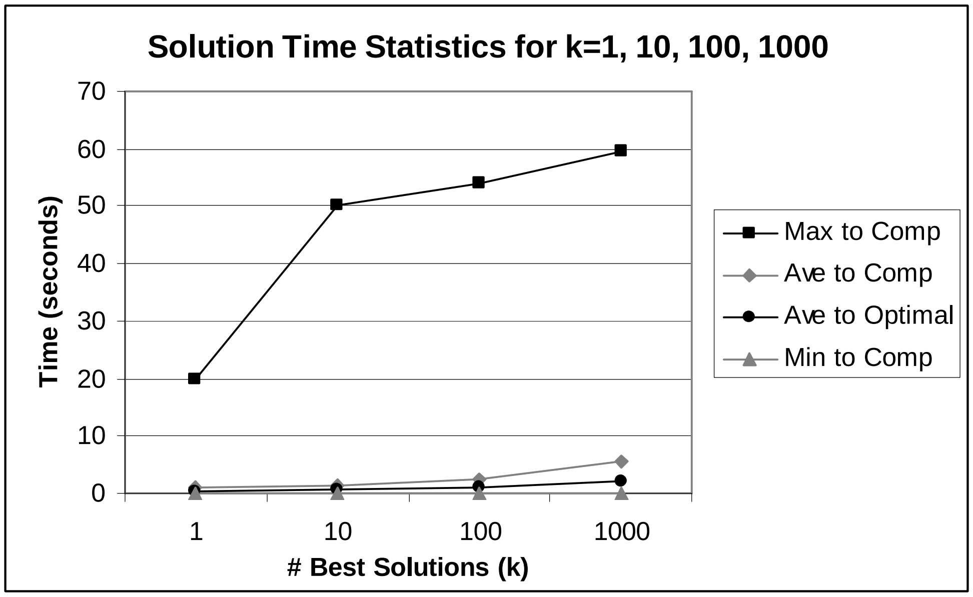


Figure 5-26. Solution Time Statistics for k=1, 10, 100, 1000

The average time to completion is shown in Figure 5-27, this time, breaking completion time out by the number of activities in the project.

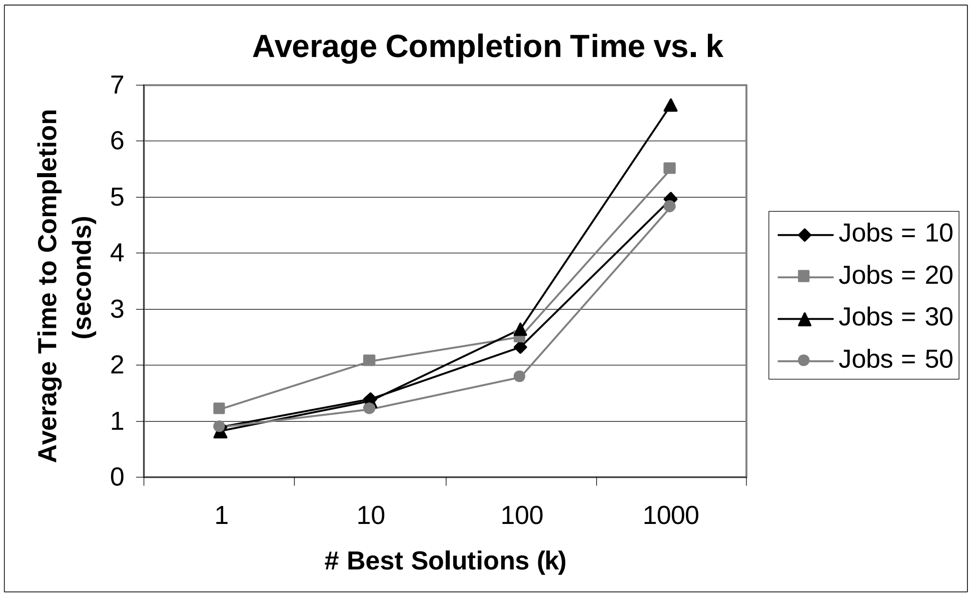


Figure 5-27. Average Completion Time vs. *k*

While the time to complete the solution process increases as *k* increases, the average time required per solution found drops off dramatically (Figure 5-28). This result suggests that the marginal cost (in terms of solution time) of increasing *k* gets very small as *k* get larger. The implication of this finding (to be addressed in greater detail in the next chapter) is that if one wishes to evaluate the set of *k*-best solutions to find a solution with certain properties (*e.g.*, feasibility in the decomposition approach), it may be better to generate a greater number of solutions initially than to risk having to resolve the problem if the intial set does not include a solution with the desired properties.

Figure 5-29 shows the overall average solution time and overall average time to find an optimal solution versus *k*. In this figure, however, the *x*-axis is scaled proportional to *k*. Consequently, the nature of the time versus *k* relationship can be better viewed and predictions can be made about the expected solution time for other values of *k*, $k < 1000$.

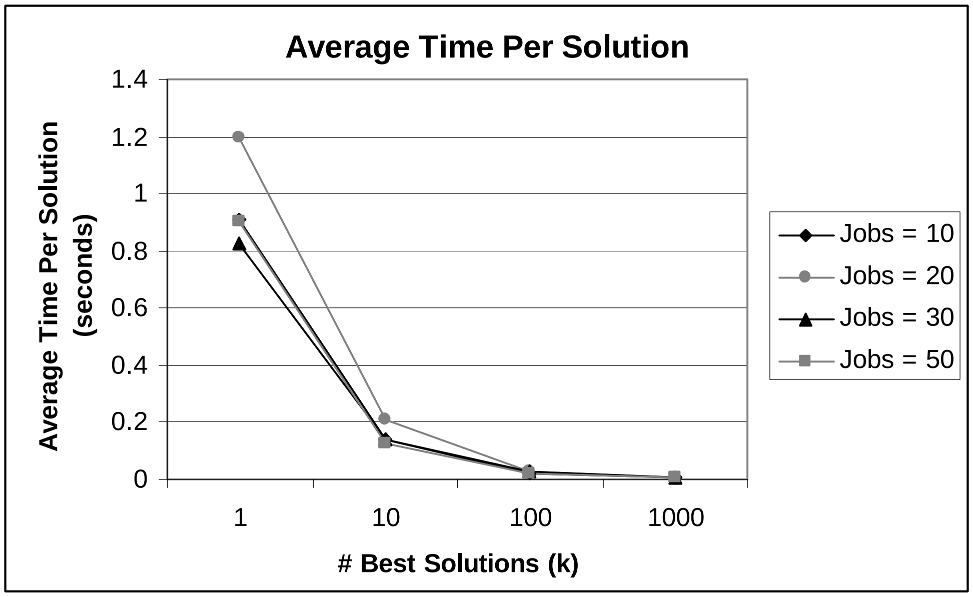


Figure 5-28. Average Time Per Solution

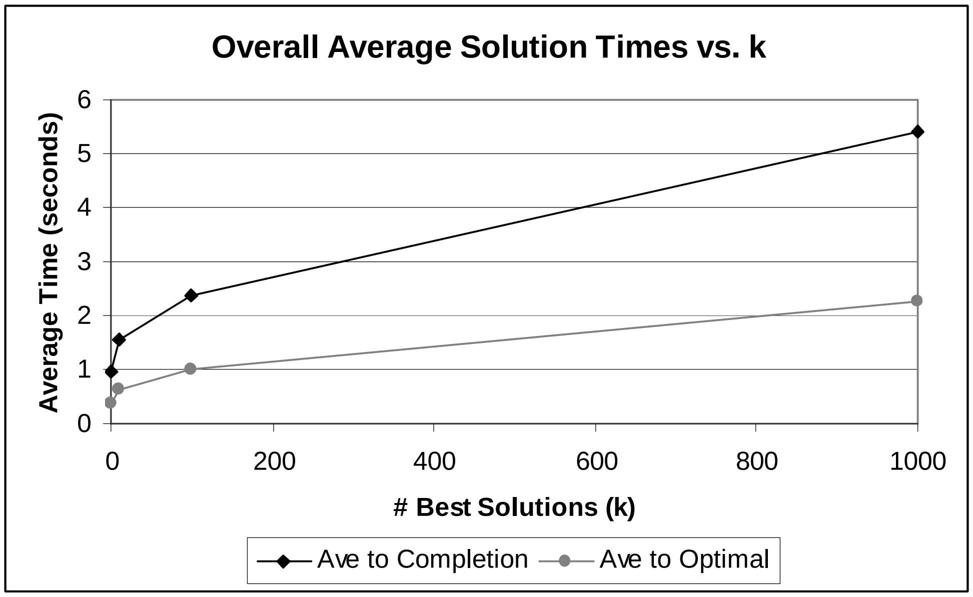


Figure 5-29. Overall Average Solution Times Versus *k*

Summary and Conclusions.

In this chapter, the literature was reviewed for solution approaches applicable to the MRCPSP-GPR/EXP. Two approaches, integer programming and an implicit enumeration scheme by Talbot (1982), were identified for their potential as either a direct approach for solving the MRCPSP-GPR/EXP or as a basis which could be extended for the MRCPSP-GPR/EXP.

The algorithm by Talbot was extended to incorporate the characteristics of generalized precedence and the availability of expediting resources. Additional bounding rules to increase the speed of the algorithm were presented.

The resulting Scheduler was tested to (1) evaluate the computational contribution of the bounding rules; (2) assess the speed of the Scheduler versus a commercially available IP solver; (3) evaluate the problem characteristics which most impact solution time; (4) investigate how early in the solution process the optimal solution is actually enumerated; and (5) assess the impact on solution time of solving a problem to find *k*-best solutions for varying values of *k*.

The results of testing were positive. The Scheduler is the first specialized algorithm capable of solving the single-project MRCPSP-GPR/EXP and its completion times were favorable compared to the commercial IP solver. The solution times required by the Scheduler for finding *k*-best solutions appear to grow slowly enough to make the Scheduler an appropriate solver for the subproblems in the decomposition approach presented in Chapter VI.

VI. Multi-Project Scheduling Through Decomposition

Overview

The Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP) concerns scheduling a hierarchically structured, multi-project *program*. At the lower level of the hierarchy are *P* projects. Each project is composed of a set of multi-modal activities which are related by generalized precedence and which compete for limited renewable and nonrenewable project-level resources. Project-level resources are wholly controlled and allocated by the project. Each activity in a project may, in addition, require some quantity of limited renewable and nonrenewable resources which are *common* to the projects (program-level resources). Project activities may also be related through generalized precedence to activities in other projects.

At the upper level of the hierarchy is the program, which controls and allocates the program-level resources (or those resources common to the projects) and which deconflicts any activity start times that violate the program-level generalized precedences. The objective of the problem is to minimize total *program costs*, which may, in some cases, result in schedules which would be suboptimal at the project-level if the projects had been treated as independent problems, free of program-level considerations.

The MRCMPSP-GPR/EXP could be modeled and solved as a single *super project* using the approach developed in Chapter V. However, as the size of the program increases, so does the difficulty in scheduling a program as a single super project. Decomposition of related multi-project problems, by contrast, has proven to be a successful approach for *dividing and conquering* such problems (*e.g.*, Deckro *et al.*, 1991; Van Hove, 1998). The decomposition of multi-project programs also lends itself to valuable economic interpretations, such as those proposed by Baumol and Fabian (1964) and Lasdon (1970).

The approach developed in this chapter decomposes the multi-project MRCMPSP-GPR/EXP into a number of smaller, independent subproblems (the individual projects) and a single master problem (the program). The master problem adjudicates the competing *demands* of the subproblems, which include the requirements for common resources and time slots in which to schedule activities.

Each subproblem is solved to find a set consisting of the *k*-best solutions (in terms of objective function value) to that problem. When all of the subproblems have been solved, the sets of best solutions are passed up to the master problem which attempts to identify one solution from each subproblem, the combination of which is feasible and optimal to the original problem. Chapter V developed a specialized algorithm for generating the *k*-best solutions for the subproblems. This chapter focuses on the mechanics of the multi-project decomposition and on solving the master problem.

As with the subproblems, the approach for solving the master problem permits the generation of *k*-best solutions to the master problem. As presented in Theorem 6-1, under certain conditions an optimal solution to the master problem is optimal to the original problem. Other than the optimal solution, however, the *k*-best master problem solutions are not necessarily the *k*-best solutions to the original problem. It is possible to set $k = 1$ to find only one optimal solution, if desired.

The basic decomposition algorithm is first presented, followed by a number of acceleration schemes. Comprehensive testing, which focuses on issues such as the benefit of the acceleration schemes and the speed of the decomposition algorithm versus solving the problem as a single project using the algorithm developed in Chapter V, is then detailed.

Decomposition Approaches in the Literature

One of the earlier decomposition approaches in the literature came from the work of Dantzig and Wolfe (1960). Dantzig and Wolfe developed a decomposition approach for linear programming (LP) which exploits the block-angular structure exhibited by many problems to subdivide the problem into smaller subproblems that can be solved independently. A *master problem* coordinates the solution process through Lagrangian multipliers which act as *prices* charged to the subproblems for the use of resources that are common to the subproblems. The master problem searches for the optimal mix (convex combination) of subproblem solutions that is optimal to the original problem.

The Dantzig and Wolfe decomposition approach has been used successfully by Wiley (1996) and Wiley *et al*. (1998) for the Multi-Project Scheduling Problem (MPSP). The objective of the MPSP is to minimize the cost or duration of a multi-project program by *crashing* or *extending* some of its activities. However, activity crashing is tied to specific limited resources. That is, for every time period an activity is crashed, an additional amount of each resource is consumed. Since these resources are limited, so is the amount of crashing possible. The MPSP addresses the multi-project problem at a high enough level of aggregation that the variables can be assumed to be continuous. (The continuity of variables is a basic assumption of Dantzig-Wolfe Decomposition.) As a result, the Dantzig-Wolfe Decomposition approach for the MPSP has limited direct applicability to the MRCMPSP-GPR/EXP where the mathematical programming formulation dictates the use of zero-one variables. Nevertheless, their use of decomposition for multi-project scheduling is one of the few in the literature and adds to the theoretical basis upon which decomposition methodologies, in general, are built.

Sweeney and Murphy (1979) present a decomposition principle which is similar to Dantzig-Wolfe decomposition in that it exploits the block-angular structure of large problems to decompose them into a set of smaller, easier-to-solve problems. The main difference is that Sweeney-Murphy Decomposition, relying on the principle of Lagrangian relaxation, is designed for problems with integer variables, while Dantzig-Wolfe Decomposition assumes continuous, linear variables. The subproblems are solved to calculate a set of best solutions for each subproblem. These sets of best solutions are passed to the master problem which attempts to identify one solution from each subproblem which, when combined, are both feasible and optimal to the original problem. If a combination of solutions cannot be identified, additional solutions are generated from the subproblems and fed to the master problem. This process continues iteratively until an optimal solution is obtained.

Deckro *et al*. (1991) use the Sweeney-Murphy Decomposition approach to solve an instance of the Resource-Constrained, Multi-Project Scheduling Problem (RCMPSP). In solving their problem, Deckro *et al*. deal with resource constraints exclusively at the master level. The resulting subproblems are simple resource-unconstrained, single-project scheduling problems (the optimal solution can be found using the standard Critical Path Method). A modified zero-one programming code was used to find the best solutions to the subproblems, as well as to solve the master problem.

Van Hove (1998) uses Sweeney-Murphy Decomposition to solve the Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem (GMRCMPSP). In his formulation of the GMRCMPSP, subproblems (individual projects) contain generalized precedences with minimal lags, renewable resources, and nonrenewable resources. The master, or program-level, problem contains *only* nonrenewable resources. Consequently, the master problem consists of selecting one solution from each subproblem which is feasible to the program-level nonrenewable resource constraints and that minimizes the program makespan. The decomposition approach proved very successful to Van Hove's problem, allowing solution of a problem with 8 subproblems, or projects, and a total of 116 activities.

Sweeney-Murphy Decomposition

Because of recent success using the Sweeney-Murphy Decomposition approach for solving the RCMPSP and the GMRCMPSP, the approach provides a solid basis for solving the multi-project MRCMPSP-GPR/EXP. This section presents the basic Sweeney-Murphy Decomposition approach, discusses the choice of how many solutions to generate from each of the subproblems, and develops alternative approaches for obtaining multipliers.

Problem Decomposition. Chapter III develops a complete zero-one formulation of the MRCMPSP-GPR/EXP. The block-angular structure of the problem takes the form in Figure 6-1.

Note that constraints (3) consist of constraints which pertain to individual projects, while constraints (2) are the coupling constraints which adjudicate the demands made on the program by the projects. Using traditional Lagrangian relaxation methods (Geoffrion, 1974), the problem is decomposed into *P* independent subproblems, SP_p?? ?, shown in Figure 6-2.

The multipliers, ? , in SP_p?? ? weight the objective functions of the subproblems in an attempt to enforce the program-level constraints. In this way, the program influences the scheduling decisions made at the project level. If the multipliers are zero, the projects are scheduled without regard for program-level constraints.

Once the subproblems have been solved to find the *k*-best solutions to each problem, the solutions are passed up to the program level where they are evaluated in a search for a combination which is feasible and optimal to the original problem. The master problem, (MP), takes the form in Figure 6-3.

Note that for any given *p* and any given *k*, $\mathbf{A}_p\mathbf{y}_p^k$ is a constant. (MP) is solved to find an optimal solution vector, $\mathbf{?}^*$, which identifies the optimal combination of subproblem solutions. If *k* is large enough so that all feasible solutions to each subproblem are included in problem (MP), then (MP) is equivalent to the original problem, (P) (Sweeney and Murphy, 1979: 1130). Solving (MP), then, provides an optimal solution to (P). If (MP) does not contain all subproblem solutions (the number of which could be intractable), then (MP) is a restriction of (P). Sweeney and Murphy prove, however, that under certain conditions, an optimal solution to (MP) is optimal to (P) (Sweeney and Murphy, 1979: 1131).

<u>Original Problem</u>

Problem (P): Minimize

$$z ? \overset{P?1}{\underset{p?1}{?}} \mathbf{c}_p \mathbf{x}_p \tag{1}$$

Subject To

$$\mathbf{A}_1\mathbf{x}_1 ? \mathbf{A}_2\mathbf{x}_2 \;?\ldots ?\; \mathbf{A}_P\mathbf{x}_P ? \mathbf{A}_{P?1}\mathbf{x}_{P?1} \quad ? \mathbf{b}_0 \tag{2}$$

$$\begin{array}{lllll} \mathbf{B}_1\mathbf{x}_1 & & & & ? \mathbf{b}_1 \\ & \mathbf{B}_2\mathbf{x}_2 & & & ? \mathbf{b}_2 \\ & & \mathbf{B}_P\mathbf{x}_P & & ? \mathbf{b}_P \\ & & & \mathbf{B}_{P?1}\mathbf{x}_{P?1} & ? \mathbf{b}_{P?1} \end{array} \tag{3}$$

$$\mathbf{A}_p ? \begin{matrix} ? \\ ? \\ ? \end{matrix}\begin{matrix} \mathbf{A}_{N_p} & \mathbf{A}_{H_p} \\ \mathbf{0} & \mathbf{A}_{H_p} \end{matrix}\begin{matrix} ? \\ ? \\ ? \end{matrix}, \; \mathbf{B}_p ? \begin{matrix} ? \\ ? \\ ? \end{matrix}\begin{matrix} \mathbf{B}_{N_p} & \mathbf{B}_{H_p} \\ \mathbf{0} & \mathbf{B}_{H_p} \end{matrix}\begin{matrix} ? \\ ? \\ ? \end{matrix}, \; \mathbf{x}_p ? \begin{matrix} ? \\ ? \\ ? \end{matrix}\begin{matrix} \mathbf{x}_{N_p} \\ \mathbf{x}_{H_p} \end{matrix}\begin{matrix} ? \\ ? \\ ? \end{matrix}, \; 1 ? \; p ? P ? 1$$

$$\mathbf{x}_{N_p} \; ? \; ?0,1?, \; 1 ? \; p ? P ? 1 \tag{4}$$

$$\mathbf{x}_{H_p} \; ? \; 0, \text{integer}, \; 1 ? \; p ? P ? 1 \tag{5}$$

where

$\mathbf{A}_{N_p}$ represents the program-level precedence constraint coefficients,

$\mathbf{A}_{H_p}$ represents the program-level expediting resource coefficients,

$\mathbf{B}_{N_p}$ represents the project-level precedence constraint coefficients of project *p*,

$\mathbf{B}_{H_p}$ represents the project-level expediting resource coefficients of project *p*,

$\mathbf{x}_{N_p}$ represents the zero-one variables associated with the activities of project *p*,

$\mathbf{x}_{H_p}$ represents the integer variables associated with the expediting resources of project *p*,

p, $1 ? p ? P$, are indices representing the *P* projects / subproblems, and

$P + 1$ is the index representing the program.

Figure 6-1. Block-Angular Structure

Subproblem for Project *p*

Problem SP$_p$? ?: Minimize $? \mathbf{c}_p ? \boldsymbol{\mu}\mathbf{A}_p ? \mathbf{x}_p$ (5)

Subject To $\mathbf{B}_p\mathbf{x}_p ? \mathbf{b}_p$ (3)

$$\mathbf{A}_p ? \begin{bmatrix} \mathbf{A}_{N_p} & \mathbf{A}_{H_p} \\ \mathbf{0} & \mathbf{A}_{H_p} \end{bmatrix}, \mathbf{B}_p ? \begin{bmatrix} \mathbf{B}_{N_p} & \mathbf{B}_{H_p} \\ \mathbf{0} & \mathbf{B}_{H_p} \end{bmatrix}, \mathbf{x}_p ? \begin{bmatrix} \mathbf{x}_{N_p} \\ \mathbf{x}_{H_p} \end{bmatrix}, 1 ? p ? P ? 1$$

$\mathbf{x}_{N_p} ? ?0,1?, 1 ? p ? P ? 1$ (4)

$\mathbf{x}_{H_p} ? 0, \text{integer}, 1 ? p ? P ? 1$ (5)

where

$\boldsymbol{\mu}$ are Lagrangian multipliers associated with the coupling constraints (2)

$\mathbf{A}_{N_p}$ represents the program-level precedence constraint coefficients,

$\mathbf{A}_{H_p}$ represents the program-level expediting resource coefficients,

$\mathbf{B}_{N_p}$ represents the project-level precedence constraint coefficients of project *p*,

$\mathbf{B}_{H_p}$ represents the project-level expediting resource coefficients of project *p*,

$\mathbf{x}_{N_p}$ represents the zero-one variables associated with the activities of project *p*,

$\mathbf{x}_{H_p}$ represents the integer variables associated with the expediting resources of project *p*,

p, $1 \leq p \leq P$, are indices representing the *P* projects / subproblems, and

P + 1 is the index representing the program.

Figure 6-2. Sweeney-Murphy Subproblem

A lower bound, $LB(\boldsymbol{\mu})$, on the optimal solution to (P) can be obtained as the combination of best solutions from each subproblem and is given by the following equation:

$$LB(\boldsymbol{\mu}) = (\mathbf{c}_1 - \boldsymbol{\mu}\mathbf{A}_1)\mathbf{y}_1^1 + (\mathbf{c}_2 - \boldsymbol{\mu}\mathbf{A}_2)\mathbf{y}_2^1 + \cdots + (\mathbf{c}_P - \boldsymbol{\mu}\mathbf{A}_P)\mathbf{y}_P^1 + (\mathbf{c}_{P+1} - \boldsymbol{\mu}\mathbf{A}_{P+1})\mathbf{y}_{P+1}^1 + \boldsymbol{\mu}\mathbf{b}_0 \quad (11)$$

Define $\mathbf{y}_p(\boldsymbol{\lambda}^*)$ to be the solution to subproblem *p* in the optimal solution to (MP). An upper bound, *UB*, on the optimal solution to (P) can be obtained by solving (MP) and letting

$$UB = \sum_{p=1}^{P+1} \mathbf{c}_p \mathbf{y}_p(\boldsymbol{\lambda}^*) \quad (12)$$

where

$$\{\mathbf{y}_1(\boldsymbol{\lambda}^*), \mathbf{y}_2(\boldsymbol{\lambda}^*), \ldots, \mathbf{y}_P(\boldsymbol{\lambda}^*), \mathbf{y}_{P+1}(\boldsymbol{\lambda}^*)\} \quad (13)$$

is the corresponding set of subproblem solutions.

Master Problem

Problem (MP):

Minimize $$\overset{K_1}{\underset{k?1}{?}} (\mathbf{c}_1\mathbf{y}_1^k)?_1^k ? \overset{K_2}{\underset{k?1}{?}} (\mathbf{c}_2\mathbf{y}_2^k)?_2^k ? \ldots ? \overset{K_P}{\underset{k?1}{?}} (\mathbf{c}_P\mathbf{y}_P^k)?_P^k ? \overset{K_{P?1}}{\underset{k?1}{?}} (\mathbf{c}_{P?1}\mathbf{y}_{P?1}^k)?_{P?1}^k \qquad (7)$$

Subject To

$$\overset{K_1}{\underset{k?1}{?}} (\mathbf{A}_1\mathbf{y}_1^k)?_1^k ? \overset{K_2}{\underset{k?1}{?}} (\mathbf{A}_2\mathbf{y}_2^k)?_2^k ? \ldots ? \overset{K_P}{\underset{k?1}{?}} (\mathbf{A}_P\mathbf{y}_P^k)?_P^k ? \overset{K_{P?1}}{\underset{k?1}{?}} (\mathbf{A}_{P?1}\mathbf{y}_{P?1}^k)?_{P?1}^k ? \mathbf{b}_? \qquad (8)$$

$$\overset{K_1}{\underset{k?1}{?}} ?_1^k \qquad ? ? \qquad (9)$$

$$\overset{K_2}{\underset{k?1}{?}} ?_2^k \qquad ? ?$$

$$\overset{K_P}{\underset{k?1}{?}} ?_P^k \qquad ? ?$$

$$\overset{K_{P?1}}{\underset{k?1}{?}} ?_{P?1}^k \qquad ? ?$$

$$?_p^k ? ?0,1? \quad ? p,k \qquad (10)$$

where

K_p ? the number of feasible solutions of Subproblem *p*,

$\mathbf{y}_p^k$? a rank-ordered feasible solution *k* of Subproblem *p*,

$?_p^k$? 1 if solution *k* of Subproblem *p* is selected; 0, otherwise.

Figure 6-3. Sweeney-Murphy Master Problem

Now, define

$$? ? UB ? LB?\boldsymbol{\mu}? \qquad (14)$$

to be the difference between the upper bound and lower bound on the solution to (P), and define

$$? _p ? \big(\mathbf{c}_p ? \mathbf{\mu A}_p\big)\mathbf{y}_p^{K_p} ? \big(\mathbf{c}_p ? \mathbf{\mu A}_p\big)\mathbf{y}_p^1 \qquad (15)$$

to be the difference between the worst solution to subproblem p and the best solution to subproblem p. Optimality conditions are given in the Sweeney-Murphy Optimality Theorem, provided as Theorem 6-1.

Theorem 6-1. Sweeney-Murphy Optimality Theorem (Sweeney and Murphy, 1979: 1131)

Theorem: If $?_p ? ?$ for $1 ? p ? P ? 1$, then $\big(\mathbf{y}_1(?^*), \mathbf{y}_2(?^*), \ldots, \mathbf{y}_P(?^*), \mathbf{y}_{P?1}(?^*)\big)$ is an optimal solution to (P).

Proof: The value of $\big(\mathbf{y}_1(?^*), \mathbf{y}_2(?^*), \ldots, \mathbf{y}_P(?^*), \mathbf{y}_{P?1}(?^*)\big)$ in Problem (P) is an upper bound. That is, $UB ? \sum_{p?1}^{P?1} \mathbf{c}_p\mathbf{y}_p(?^*)$.

Suppose $?$ a $\bar{j} ? k_{\bar{p}}$ for subproblem $\bar{p}$ such that $\mathbf{y}_{\bar{p}}^{\bar{j}}$ is part of a feasible solution to (P) yielding a value $\bar{z} ? UB$.

Show that this supposition leads to a contradiction and, hence, no subproblem solutions not already included in (MP) can be part of a better solution to (P).

The minimum value of a Lagrangian relaxation of (P) with $\mathbf{y}_{\bar{p}}$ held fixed at $\mathbf{y}_{\bar{p}}^{\bar{j}}$ is given by $\bar{z} ? \sum_{\substack{p?1 \\ p?\bar{p}}}^{P?1} \big(\mathbf{c}_p ? \mathbf{\mu A}_p\big)\mathbf{y}_p^1 ? \big(\mathbf{c}_{\bar{p}} ? \mathbf{\mu A}_{\bar{p}}\big)\mathbf{y}_{\bar{p}}^{\bar{j}} ? \mathbf{\mu b}_0$. Therefore,

$$
\begin{aligned}
\bar{z} &= \sum_{\substack{p=1\\ p\neq\bar{p}}}^{P+1} \left(\mathbf{c}_p - \boldsymbol{\mu}\mathbf{A}_p\right)\mathbf{y}_p^1 + \left(\mathbf{c}_{\bar{p}} - \boldsymbol{\mu}\mathbf{A}_{\bar{p}}\right)\mathbf{y}_{\bar{p}}^{j} + \boldsymbol{\mu}\mathbf{b}_0 \\
&= \sum_{p=1}^{P+1} \left(\mathbf{c}_p - \boldsymbol{\mu}\mathbf{A}_p\right)\mathbf{y}_p^1 - \left(\mathbf{c}_{\bar{p}} - \boldsymbol{\mu}\mathbf{A}_{\bar{p}}\right)\mathbf{y}_{\bar{p}}^1 + \left(\mathbf{c}_{\bar{p}} - \boldsymbol{\mu}\mathbf{A}_{\bar{p}}\right)\mathbf{y}_{\bar{p}}^{j} + \boldsymbol{\mu}\mathbf{b}_0 \\
&= LB(\boldsymbol{\mu}) + \left(\mathbf{c}_{\bar{p}} - \boldsymbol{\mu}\mathbf{A}_{\bar{p}}\right)\mathbf{y}_{\bar{p}}^{j} - \left(\mathbf{c}_{\bar{p}} - \boldsymbol{\mu}\mathbf{A}_{\bar{p}}\right)\mathbf{y}_{\bar{p}}^1 \\
&\geq LB(\boldsymbol{\mu}) + \Delta_{\bar{p}} \\
&\geq LB(\boldsymbol{\mu}) + \min_p \left\{\Delta_p\right\} \\
&\geq LB(\boldsymbol{\mu}) + \Delta \\
&\geq UB
\end{aligned}
$$

This is a contradiction since it was assumed that $\bar{z} < UB$. Therefore, $\left(\mathbf{y}_1(\boldsymbol{\mu}^*), \mathbf{y}_2(\boldsymbol{\mu}^*), \ldots, \mathbf{y}_P(\boldsymbol{\mu}^*), \mathbf{y}_{P+1}(\boldsymbol{\mu}^*)\right)$ is an optimal solution to (P).

The Sweeney-Murphy Optimality Theorem hinges upon the identification of the *k*-best solutions to each subproblem. The Scheduler presented in Chapter V is designed to find the *k*-best solutions to the subproblems and is used for that purpose. One important consideration, though, that is neither discussed by Sweeney and Murphy nor addressed yet in this discussion, is the possibility of multiple solutions of equal value.

The Sweeney-Murphy Optimality Theorem shows that, under specific conditions, no solution not already in the set of *k*-best solutions can contribute to a better solution to the original problem. This can be true, though, only if the solutions already in the set of *k*-best are strictly better than the solutions not in the set. If a solution not in the set has a value equal to that of the *k*-th best solution, then the *k*-th best solution cannot be used in the calculation of Δ_p which is used in the optimality test. Suppose that a set of *k*-best solutions has been generated and that a solution exists that is not in the set but which has a value equal to that of the *k*-th best solution. Since the set contains only *k* solutions, it is rather arbitrary as to whether the *k*-th best solution currently in the set or the alternate solution of equal value should be included in the set. It is entirely possible that including the alternate solution in the set would lead to a better solution than that possible with the current *k*-th best solution. Consequently, as subproblems are solved, if a solution of equal value to the *k*-th best is dropped from the set, then Δ_p is calculated using, not the value of the *k*-th best solution (*i.e.*, the worst solution in the set), but the value of the next worst solution in the set. Specifically, if Solutions k-1, k-2,…, k-n ($n < k$) all have the same objective function value as the k-th best solution and if Solution k-n-1 has a better objective function value than the k-th best solution, then the objective function of Solution k-n-1 is used to calculate $?_p$. If all solutions of equal value to the k-th best solution are in the set, then $?_p$ is calculated using the value of the k-th best solution.

<u>Solving the Subproblems</u>. The first step in the Sweeney-Murphy Decomposition process is solving the subproblems to find their respective k-best solutions. The decomposition of the multi-project MRCMPSP-GRP/EXP leads to subproblems which are instances of the single-project MRCPSP-GRP/EXP, which can be solved using the approach developed in Chapter V. This is true with one exception: Subproblem P+1.

While Subproblems 1 through P are actual single projects, Subproblem P+1 is of an entirely different nature. Subproblem P+1 consists of two types of variables: (1) the variable representing the execution time of the program-level terminal activity and (2) the variables representing the program-level expediting resources. These variables appear in the coupling constraints of both (P) and (MP) to (1) tie the program completion time to the scheduled execution times of the other activities and (2) determine the quantity of program-level expediting resources required to make a combination of solutions from Subproblems 1 through P (the real *project* subproblems) resource feasible.

Subproblem P+1 contributes to the program-level objective function by assessing a penalty against the program based on the execution time of the program-level terminal activity (the program completion time) and by charging the program for the program-level expediting resources used by the projects.

When separated from the coupling constraints, though, the independent Subproblem P+1 becomes a rather trivial problem (see Chapter III for the zero-one formulation). Its constraints consist exclusively of the upper bound, D (program horizon), on the program completion time and the upper bounds on expediting resource use. Consequently, regardless of the penalty associated with program completion, or the cost of expediting resources, or even the choice of $\boldsymbol{\mu}$, the optimal solution to Subproblem P+1 will always be zero for all variables. Of course, this optimal is infeasible to (MP), and thus (P), for any program with a non-zero duration (a basic assumption).

Like the other subproblems, a set of *k*-best solutions could be generated for Subproblem *P*+1, but this would undoubtedly require an extremely large *k* just to provide a solution that makes (MP) feasible.

Another option for dealing with Subproblem *P*+1 is to generate its best solutions *on the fly*. This approach is suggested by Sweeney and Murphy for subproblems that are *rich* in near-optimal solutions (Sweeney and Murphy, 1979: 1134). Simply stated, rather than contributing a set of *k*-best solutions from which the master problem can draw, the subproblem is incorporated directly into a revised master problem, (MP2) (shown in Figure 6-4).

<u>Solving the Master Problem</u>. The Revised Master Problem (MP2) is solved using an implicit enumeration algorithm. Implicit enumeration is used primarily for two reasons. First, the procedure used to construct master problem solutions allows for quick and efficient fathoming of large sets of infeasible or dominated subproblem solution combinations. Second, the algorithm efficiently produces a set of *k*-best solutions to the master problem.

The algorithm attempts to build feasible master problem solutions by combining solutions from the subproblems. Starting with the first solutions of each subproblem, the algorithm moves from subproblem to subproblem, adding on a new subproblem solution until either (1) one solution from each subproblem has been combined to form a complete, feasible solution to the master problem or (2) at some point in the building process, the current partial solution is found to be infeasible or dominated by the *k*-th best solution to the master. In either case, the algorithm backtracks, first to untried solutions to the current subproblem, then to previous subproblems and *their* untried solutions.

When a complete, feasible solution to the master problem is found, it is compared against the current *k*-th best solution to the master. If the new solution is at least as good, it is added to the solution array and the *k*-th best solution is removed. Complete solutions are stored in a *k* x (*n* +1) x 2 array, where *k* is the number of best solutions desired and *n* is the number of activities in the problem. For each activity, the solution array stores two values: (1) its execution mode and (2) its start time. The objective function value is stored in Row 0 of the array. The solution array is initialized with large values.

Note that while the above approach yields the *k*-best solutions to (MP), there is no guarantee that these are the *k*-best solutions to (P). The Sweeney-Murphy Optimality Conditions guarantee only that the best solution to (MP) is optimal to (P). The *k*-best solutions to (MP) provide *k*-good solutions to (P), but these are not the *k*-best solutions to (P) unless all feasible solutions to the subproblems are included in (MP); in other words, if (MP) equals (P).

Revised Master Problem

Problem (MP2):

Minimize $$\overset{K_1}{\underset{k?1}{?}} (\mathbf{c}_1\mathbf{y}_1^k)?_1^k \; ? \; \overset{K_2}{\underset{k?1}{?}} (\mathbf{c}_2\mathbf{y}_2^k)?_2^k \; ? \; \ldots ? \; \overset{K_P}{\underset{k?1}{?}} (\mathbf{c}_P\mathbf{y}_P^k)?_P^k \; ? \; \mathbf{c}_{P?1}\mathbf{x}_{P?1} \qquad (16)$$

Subject To

$$\overset{K_1}{\underset{k?1}{?}} (\mathbf{A}_1\mathbf{y}_1^k)?_1^k \; ? \; \overset{K_2}{\underset{k?1}{?}} (\mathbf{A}_2\mathbf{y}_2^k)?_2^k \; ? \; \ldots ? \; \overset{K_P}{\underset{k?1}{?}} (\mathbf{A}_P\mathbf{y}_P^k)?_P^k \; ? \; \mathbf{A}_{P?1}\mathbf{x}_{P?1} \quad ? \, \mathbf{b}_? \qquad (17)$$

$$\overset{K_1}{\underset{k?1}{?}} ?_1^k \qquad ? \; ? \qquad (9)$$

$$\overset{K_2}{\underset{k?1}{?}} ?_2^k \qquad ? \; ?$$

$$\overset{K_P}{\underset{k?1}{?}} ?_P^k \qquad ? \; ?$$

$$\mathbf{B}_{P?1}\mathbf{x}_{P?1} \quad ? \, \mathbf{b}_{P?1} \qquad (3)$$

$$?_p^k \; ? \; ?0,1? \quad ? \, p,k \qquad (10)$$

where

K_p ? the number of feasible solutions of Subproblem *p*,

$\mathbf{y}_p^k$? a rank-ordered feasible solution *k* of Subproblem *p*,

$?_p^k$? 1 if solution *k* of Subproblem *p* is selected; 0, otherwise.

Figure 6-4. Revised Sweeney-Murphy Master Problem

As an example, consider the four-project problem described in Appendix F. This problem was solved to find the 1000-best solutions using the Scheduler and then re-solved to find the 1000-best solutions using the decomposition approach. In the case of the decomposition approach, each subproblem provided (to the master) a set of their 1000-best solutions, leading to a master problem with a total of 1000^4 possible subproblem combinations. Both approaches found the same optimal (a single optimal in this case) with an objective function value of 19,680. However, the 1000-th best solution from the Scheduler had a value of 24,752 while the 1000-th best solution from the decomposition approach had a value of 32,760. In fact, the set of 1000 solutions passed up from each of the subproblems to the master problem did not contain a combination leading to a solution of 24,752 (the 1000-th best Scheduler solution). The closest solutions found by the decomposition approach were 24,704 (ranked 99) and 24,769 (ranked 100).

Consequently, there is a tradeoff between the time required to obtain a set of *k* solutions and the quality of those solutions. If finding an optimal solution quickly is the primary goal, and obtaining a set of good alternative solutions is only secondary, then the decomposition approach should be used. If, however, a set of *k*-best solutions is required, then the Scheduler from Chapter V is a more appropriate solution approach.

Assumptions. Before proceeding to the notation and description of the Decomposition Algorithm, note the following assumptions.

1. Subproblems corresponding to the projects, Problems SP_p?? ?, are solved in the order in which they are numbered.
2. Subproblem solutions are rank ordered, k ? 1 being an optimal.

Notation. The following notation is used in the Decomposition Algorithm.

Problem Types:

SP_p?? ?	=	the subproblem corresponding to Project *p*

Indices:

i	=	a project activity
m_i	=	the mode of activity *i*
s_i	=	the scheduled start time of activity *i*
k_p	=	the counter indicating the current solution of Subproblem *p*

Solution Parameters:

P	=	the total number of subproblems
K_p	=	the number of best solutions from Subproblem p
K_0	=	the desired number of best solutions to the master problem
z_{pk}	=	the objective function value of the k-th solution to Subproblem p
Z_k	=	the objective function value of the k-th solution to the master problem (MP)
$?_p$	=	the difference between the worst and best solutions to Subproblem p
$\boldsymbol{\mu}$	=	the Lagrangian multipliers associated with the coupling constraints

Activity Sets:

O_i	=	the set of activities which precede activity i
N_i	=	the set of activities which have a direct start-start lag relationship with activity i
M_{k_p}	=	the set of mode assignments associated with solution k_p
S_{k_p}	=	the set of start time assignments associated with solution k_p

Resource Sets:

Q^R	=	the set of program-level renewable resources
Q^N	=	the set of program-level nonrenewable resources

Time-Related Parameters:

d_{im_i}	=	the duration of activity i in mode m
$?_{ij}^{\min}$	=	the minimal start-start lag time between activities i and j
$?_{ij}^{\max}$	=	the maximal start-start lag time between activities i and j

Cost Parameters:

$c_{k_p}^N$	=	cost of nonrenewable expediting resources required by k_p
$c_{k_p}^R$	=	cost of renewable expediting resources required by k_p
C_p	=	the accumulated cost of the current solutions of Subproblems 1 through p

Resource-Related Parameters:

$r^R_{im_iq}$	=	the units of renewable resource q required by activity i in mode m_i
R^R_{pqt}	=	the units of renewable resource q remaining in time t after projects 1 through p have been added to the program schedule
H^R_{pqt}	=	the units of expediting, renewable resource q remaining in time t after projects 1 through p have been added to the program schedule
$r^N_{im_iq}$	=	the units of nonrenewable resource q required by activity i in mode m_i
R^N_{pq}	=	units of nonrenewable resource q remaining after projects 1 through p have been added to the programschedule
H^N_{pq}	=	the units of expediting, nonrenewable resource q remaining after projects 1 through p have been added to the program schedule
$r^N_{k_pq}$	=	the total demand by solution k_p for nonrenewable resource q
$r^R_{k_pqt}$	=	the total demand by solution k_p for renewable resource q at time t

Decomposition Algorithm. The Decomposition Algorithm is now outlined, followed by a narrative description of the scheme.

Decomposition Algorithm

Step 0 *Initialization*.

Obtain $\bar{\boldsymbol{\mu}}$, an initial value of $\boldsymbol{\mu}$.

Set Z_k ? an arbitrarily large number (999999 in this study) for $1 ? k ? K_0$.

Set K_p = the initial number of solutions to generate for Subproblems $p, 1 ? p ? P$.

Set ? ? an arbitrarily large number (*e.g.*, 999999).

Set $?_p ?$ an arbitrarily small, non-negative number (*e.g.*, 0) for $1 ? p ? P$.

Step 1 For *p* such that $?_p ? ?$, solve $SP_p ?\boldsymbol{\mu}?$ to obtain the K_p-best solutions for Subproblems *p*, $1 ? p ? P$ (the *project*-related subproblems). For each subproblem, record if a solution equal to the *k*-th best is dropped from the set of *k*-best.

Step 2 If, for subproblem *p* ($1 ? p ? P$), a solution with value equal to that of solution K_p is dropped from the set of K_p-best solutions, let $?_p ? \max_k ?z_{pk} \mid z_{pk} ? z_{pK_p} ?? z_{p1}$. Otherwise, let $?_p ? z_{pK_p} ? z_{p1}$.

Step 3 Record $LB?\boldsymbol{\mu}?? \overset{P}{\underset{p?1}{?}} z_{p1}$. Set *p* = 0 and $C_0 ? 0$.

Step 4 Let *p* = *p* + 1 and let $k_p ? 0$.

Step 5 Let $k_p ? k_p ? 1$. If $k_p ? K_p$, go to Step 15.

Step 6 *Test for Dominance*. Test whether $C_{p?1}$ plus the objective function value of subproblem solution k_p, z_{k_p}, is dominated by the value of the K_0-best master problem solution, Z_{K_0}. Thus, if $Z_{K_0} ? C_{p?1} ? z_{k_p}$, subproblem solution k_p cannot lead to an improved master solution, so go to Step 5.

Step 7 *Nonrenewable Resource Feasibility*. Determine if k_p is feasible to the nonrenewable resource constraints (*i.e.*, the sum of regular and expediting nonrenewable resources is sufficient for the nonrenewable resource demand of solution k_p). Let $r^N_{k_p} ? \overset{J_p}{\underset{\substack{i?1 \\ m_i? M_{k_p}}}{?}} r^N_{im_iq}$ be the total demand by solution k_p for nonrenewable resource *q*. If $r^N_{k_p} ? R^N_{p?1,q} ? H^N_{p?1,q}, ? q ? Q^N$, then k_p is nonrenewable-resource feasible. If not feasible, go to Step 5.

Step 8 *Test for Dominance*. Test whether $C_{p?1}$ plus the objective function value of subproblem solution k_p, z_{k_p}, plus the cost of nonrenewable expediting resources required by k_p, $c_{k_p}^N$, is dominated by the value of the K_0-best master problem solution, Z_{K_0}. Thus, if Z_{K_0} ? $C_{p?1}$? z_{k_p} ? $c_{k_p}^N$, subproblem solution k_p cannot lead to an improved master solution, so go to Step 5.

Step 9 *Generalized Precedence Feasibility*. Determine if subproblem solution k_p is feasible as to the program-level generalized precedences. For each activity *i* in *p* which has a generalized precedence relationship with an activity *j* in any of projects 1 through *p*-1, the following conditions must be true for feasibility. If not feasible, go to Step 5. (Note: these are the same generalized precedence conditions used in the Scheduler and discussed in Chapter V.)

	d_{im_i} ? 0	d_{im_i} ? 0
d_{jm_j} ? 0	s_i ? s_j ? d_{jm_j}, ? j ? O_i s_i ? s_j ? ? $_{ji}^{\min}$, ? j ? C_i s_i ? s_j ? ? $_{ji}^{\max}$, ? j ? C_i	s_i ? s_j ? d_{jm_j} ? 1, ? j ? O_i s_i ? s_j ? ? $_{ji}^{\min}$? 1, ? j ? C_i s_i ? s_j ? ? $_{ji}^{\max}$? 1, ? j ? C_i
d_{jm_j} ? 0	s_i ? s_j ? 1, ? j ? O_i s_i ? s_j ? ? $_{ji}^{\min}$? 1, ? j ? C_i s_i ? s_j ? ? $_{ji}^{\max}$? 1, ? j ? C_i	s_i ? s_j, ? j ? O_i s_i ? s_j ? ? $_{ji}^{\min}$, ? j ? C_i s_i ? s_j ? ? $_{ji}^{\max}$, ? j ? C_i

Step 10 *Renewable Resource Feasibility*. Determine if k_p is feasible to the renewable resource constraints (*i.e.*, the sum of regular and expediting renewable resources in each period

is sufficient for the renewable resource demand of solution k_p). Let

$r^R_{k_pqt} ? \mathop{?}\limits_{\substack{i?1 \\ t?\,?s_i, s_i ? d_{im_i} ?1? \\ m_i ? M_{k_p} \\ s_i ? S_{k_p}}}^{J_p} r^R_{im_iq}$ be the total demand by solution k_p for renewable resource q in time

period t. If $r^R_{k_pqt} ? R^R_{p?1,qt} ? H^R_{p?1,qt}, ? q ? Q^R, s_1 ? t ? s_{J_p}, ?s_1, s_{J_p}?? S_{k_p}$, then k_p is renewable-resource feasible. If not feasible, go to Step 5.

Step 11 *Adjust Resources and Costs.* The new partial master solution schedule formed by adding subproblem solution k_p is feasible and may lead to an improved master solution. Adjust program-level resource availabilities and the master schedule cost as follows:

$$?? \quad R^N_{pq} ? \begin{cases} R^N_{p?1,q} ? r^N_{k_pq} \text{ if } r^N_{k_pq} ? R^N_{p?1,q} \\ 0 \text{ if } r^N_{k_pq} ? R^N_{p?1,q} \end{cases}, ? q ? Q^N$$

$$?? \quad H^N_{pq} ? \begin{cases} 0 \text{ if } r^N_{k_pq} ? R^N_{p?1,q} \\ H^N_{p?1,q} ? R^N_{p?1,q} ? r^N_{k_pq} \text{ if } r^N_{k_pq} ? R^N_{p?1,q} \end{cases}, ? q ? Q^N$$

$$?? \quad R^R_{pqt} ? \begin{cases} R^R_{p?1,qt} ? r^R_{k_pqt} \text{ if } r^R_{k_pqt} ? R^R_{p?1,qt} \\ 0 \text{ if } r^R_{k_pqt} ? R^R_{p?1,qt} \end{cases}, ? q ? Q^R,\ s_1 ? t ? s_{J_p}, ?s_1, s_{J_p}?? S_{k_p}$$

$$?? \quad H^R_{pqt} ? \begin{cases} 0 \text{ if } r^R_{k_pqt} ? R^R_{p?1,qt} \\ H^R_{p?1,qt} ? R^R_{p?1,qt} ? r^R_{k_pqt} \text{ if } r^R_{k_pqt} ? R^R_{p?1,qt} \end{cases}, ? q ? Q^N,\ s_1 ? t ? s_{J_p}, ?s_1, s_{J_p}?? S_{k_p}$$

$$?? \quad C_p ? C_{p?1} ? c^N_{k_p} ? c^R_{k_p}$$

Step 12 *Test for Dominance.* Test whether C_p is dominated by the value of the K_0-best master problem solution, Z_{K_0}. Thus, if $Z_{K_0} ? C_p$, subproblem solution k_p cannot lead to an improved master solution, so remove subproblem solution k_p from the current partial solution , adjust resource availabilities and the partial solution cost, and go to Step 5.

Step 13 If subproblem *p* is NOT the last subproblem, $p ? P$, go to Step 4. Otherwise, this complete solution is as good as the current K_0 -best solution, so add this solution to the set of K_0 -best and re-rank solutions.

Step 14 *Adjust Resources and Costs.* Remove subproblem solution k_p from the current master schedule. Adjust resource availabilities and the master schedule cost. Go to Step 5.

Step 15 *Backtrack by Subproblem*. Let $p ? p ? 1$. If $p ? 1$, go to Step 5.

Step 16 *Test for Optimality.* (Note that *p* = 0.) Let $UB ? Z_1$. Calculate $? ? UB ? LB?\mathbf{\mu}?$. If $?_p ? ?$ for $1 ? p ? P$, then Z_1 is optimal. Stop. Algorithm *complete*.

Step 17 Z_1 is NOT optimal. For *p* such that $?_p ? ?$, increase K_p, the number of solutions to generate for Subproblem *p*. Go to Step 1.

Step 0 of the algorithm is an initialization step. In this step, an initial value of the Lagrangian multipliers, $\mathbf{\mu}$, are obtained using one of the methods described later in this chapter. The method used to obtain the multipliers may affect the performance of the algorithm but does not impact the flow of the algorithm itself.

Step 0 is also used to set initial values for four sets of variables. The initial values for Z_k, $1 ? k ? K_0$, are set to arbitrarily large numbers. These numbers are replaced by the objective function values of feasible solutions to the master problem as these solutions are generated. The initial values for K_p, $1 ? p ? P$, are set according to a scheme for choosing how many solutions to generate for each subproblem, also discussed later in the chapter. Finally, a large value for ? and small values for $?_p$, $1 ? p ? P$, are set. While these values are meaningless at this point in the algorithm, they are set necessarily but simply to satisfy the condition in Step 1 which invokes a first-time solution of each subproblem to generate sets of *k*-best solutions.

Step 1 comprises solution of the subproblems to generate sets of *k*-best solutions. The first pass through this step requires that all subproblems be solved (hence, the initial values for the deltas set in Step 0), while subsequent passes through the step require the solution of only those subproblems which fail the optimality test of Steps 15 and 16.

Step 2 calculates a (meaningful) $?_p$ for each subproblem based on the solution sets generated in Step 1.

Step 3 records the lower bound on the optimal solution to (P) as the sum of the optimal solutions to the subproblems. This step also sets p = 0 and C_0 ? 0 .

Step 4 increments counter p by one so that in the first pass p = 1 and the algorithm begins constructing a solution to (MP) by adding a candidate solution from Subproblem 1. (While this step identifies which subproblem is the current subproblem, the next step identifies which candidate solution from the current subproblem to add.) k_p is also set to zero in this step.

Step 5 increments k_p by one. In the first pass through this step, k_p = 1 and the algorithm begins constructing a solution to (MP) by adding Solution 1 to Subproblem 1.

As the algorithm builds a solution to (MP) by incrementally adding a solution from each subproblem, Step 6 tests whether or not the cost of the previous partial solution plus the objective function value of the candidate subproblem solution being added exceeds the objective function value of the *k*-th best solution currently recorded for (MP). If so, the partial solution being constructed cannot lead to an improved solution to (MP), and the candidate solution to the current subproblem is fathomed by returning to Step 5 where the next candidate solution to the current subproblem is nominated. Note that rejecting a candidate solution at this step requires no accounting for resources or costs, since no resources or costs have been charged yet for adding this solution. Consequently, this simple dominance test saves potentially considerable time otherwise required for charging and subsequently refunding resources and costs for an unproductive candidate solution.

If the test at Step 6 is passed, Step 7 determines if there are sufficient nonrenewable resources (regular plus expediting) available to add the candidate solution. If not, the candidate solution is rejected by returning to Step 5. Note that no resources or costs are charged at this step, either, in order to save the time otherwise required to account for resources and costs associated with a solution which may be infeasible or unproductive.

Step 8 builds upon the feasibility test in Step 7. Step 7 determined that there are sufficient regular plus expediting nonrenewable resources available to make the candidate solution feasible. However, if expediting resources must be used, the cost of those resources may make the candidate subproblem solution too costly. If so, the candidate solution is fathomed by returning to Step 5.

Steps 9 and 10 continue testing the candidate solution by checking for precedence and renewable resource feasibility, respectively. If at either step the candidate is determined to be infeasible, the solution is fathomed by returning to Step 5.

Once the algorithm has reached Step 11, it has determined that the candidate solution to the current subproblem is feasible to the precedence constraints as well as to the renewable and nonrenewable resource limitations. It has also determined that the cost of the previous partial schedule plus the objective function value of the candidate solution plus the expediting nonrenewable resource costs required to add the candidate solution do not exceed the currently recorded *k*-th best solution to (MP). Therefore, Step 11 adds the candidate solution to the previous partial schedule by charging its resource requirements against the available resources and by adding its associated costs.

Up to this point, however, no dominance tests of the candidate solution included the cost of expediting renewable resources. Step 12, therefore, performs one final dominance test on the new partial solution which includes the cost of expediting renewable resources required by the candidate solution. If the new partial solution cannot lead to an improved solution to (MP), the candidate solution is removed from the partial solution, resources and costs are adjusted, and the algorithm returns to Step 5 where the next candidate solution is considered.

If the new partial solution is feasible and is not dominated by the *k*-th best solution to (MP), then Step 13 checks to see if a solution has been added by each subproblem. If not, the algorithm returns to Step 4 where *p* is once again incremented by one and candidate solutions from the next subproblem are considered. If a solution from each subproblem has been added, then the new partial solution is, in fact, a complete solution to (MP). This solution is added to the set of solutions to (MP) and the set of solutions is re-ranked from best to worst.

Once the complete solution has been recorded, the last subproblem solution to be added is removed (in Step 14) by adjusting resources and costs and returning to Step 5.

When returning to Step 5 from any other step, the next rank-ordered solution to the current subproblem becomes the next candidate solution. If, however, all rank-ordered solutions to the current Subproblem *p** have been checked in context of partial solution *p**-1, then the algorithm proceeds to Step 15 where it backtracks by subproblem. In other words, Subproblem *p**-1 becomes the current subproblem. If *p**-1 is greater than or equal to one, the algorithm returns to Step 5 where the next rank-ordered solution to Subproblem *p**-1 becomes the new candidate solution. In this way, the algorithm implicitly enumerates every possible combination of the subproblem solutions.

If, at Step 15, *p**-1 equals zero, then all combinations of subproblem solutions have been implicitly enumerated. Therefore, an optimality test is performed to determine if an optimal solution found for (MP) is optimal to (P). If so, the algorithm terminates successfully. If not, K_p for each subproblem *p* violating the optimality condition is increased (in Step 17), and the algorithm returns to Step 1 where larger sets of best solutions are generated for each of the violating subproblems. This marks the beginning of the next iteration and the algorithm proceeds as before.

Note that if the optimal is not found on the first iteration, the array of master solutions is not reinitialized to large numbers. The solutions with which it was previously filled during the first iteration remain in the array. This provides a tighter upper bound and faster fathoming for future iterations.

<u>Correction to Sweeney-Murphy Approach</u>. The solution approach proposed by Sweeney and Murphy, and implemented in the Decomposition Algorithm above, hinges upon iteratively adding subproblem solutions to the master problem until the optimality criterion (provided in the Sweeney-Murphy Optimality Theorem) is met. The use of the Sweeney-Murphy Optimality Theorem as the one and only stopping criterion suggests reliance on the theorem as a necessary condition for optimality. Such use of the theorem, however, is inappropriate. While the theorem provides a sufficient condition for optimality, it does not, in fact, provide a necessary condition (see counter example below). In such cases, the algorithm may terminate without clearly indicating the optimality of the solution. If a given problem meets the sufficient condition in Step 12 of the Decomposition Algorithm, the algorithm stops with the result that the current optimal solution to (MP) is optimal to (P). If, on the other hand, the given problem does not meet the sufficient condition in Step 12, one cannot say if the current optimal solution to (MP) is optimal to (P) or not.

One can continue with further iterations of the algorithm in hopes that the sufficient condition will eventually be met, but one of three realities will certainly be faced:

1. The algorithm iterates to a point where the sufficient condition is met.
2. The algorithm terminates prematurely without the sufficient condition met and without knowing if the current optimal to (MP) is optimal to (P). However, the current optimal to (MP) is feasible to (P), since (MP) is a restriction of (P), so the optimal to (MP) may be treated as a heuristic solution to (P).
3. The algorithm is allowed to iterate to a point at which all feasible solutions of all subproblems have been generated, but the sufficient condition is still not met. In this case, (MP) is equivalent to (P) and one can conclude that the optimal to (MP) must be optimal to (P), but this determination is made without the Sweeney-Murphy Optimality Theorem being met.

In the third case above, optimality of (P) can be established without the sufficient condition being met, but the cost of generating all solutions to all subproblems may be high (perhaps higher than solving the problem without decomposition). One would, therefore, like to find a tighter necessary condition than the generation of all subproblem solutions. This is a subject for further investigation.

Consider, now, the following example which counters the use of the Sweeney-Murphy Optimality Theorem as a necessary condition for optimality. The example consists of a four-project program (see Figure 6-5). Each project has a dummy start activity (AS, BS, CS, and DS, respectively). The dummy start activities have no duration, no cost, and use no resources. Each project also has two (numbered) activities with durations, costs, and resource use (project- and program-level). Each project has a terminal activity (AT, BT, CT, and DT, respectively) which has no duration and uses no resources, but each terminal activity has a cost which represents the project completion cost. Finally, the projects are tied together by a dummy start activity (S) and a terminal activity (T), neither of which has a duration or uses resources. However, terminal activity (T) has a cost representing the program completion cost.

The data in Table 6-1 show that each numbered activity has a duration of one unit. Each numbered activity also requires one unit of a renewable resource specific to their respective projects. There is sufficient project-level resource availability that project-level resources do not limit the scheduling process (any precedence feasible project schedule is also resource feasible).

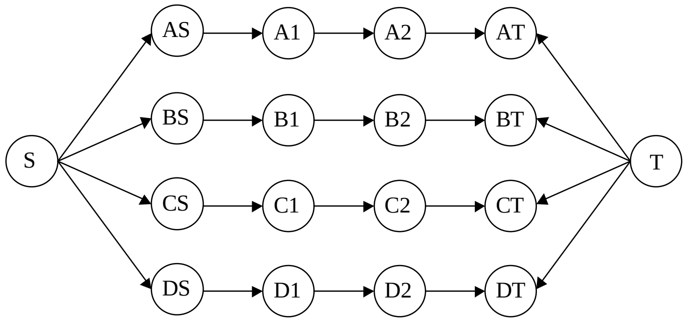


Figure 6-5. Sweeney-Murphy Optimality Theorem Counterexample Diagram

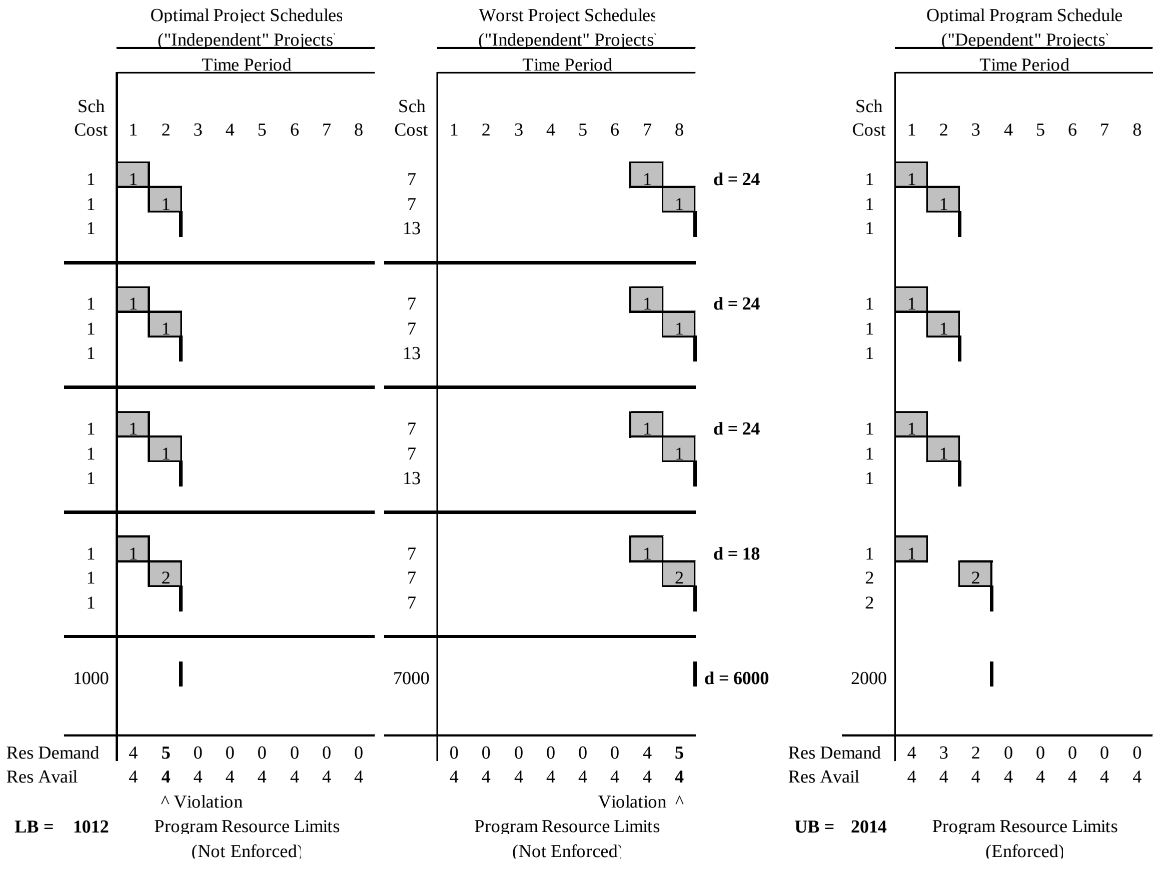
Table 6-1. Sweeney-Murphy Optimality Theorem Counterexample Data

Project	Activity	Duration	Early Start (ES)	Base Cost @ ES	Per Period Incremental Cost	Project Renew Resource	Program Renew Resource
A	A1	1	1	1	1	1	1
	A2	1	2	1	1	1	1
	AT	0	2	1	2	0	0
						Limit 2	
B	B1	1	1	1	1	1	1
	B2	1	2	1	1	1	1
	BT	0	2	1	2	0	0
						Limit 2	
C	C1	1	1	1	1	1	1
	C1	1	2	1	1	1	1
	CT	0	2	1	2	0	0
						Limit 2	
D	D1	1	1	1	1	1	1
	D2	1	2	1	1	1	**2**
	DT	0	2	1	**1**	0	0
						Limit 2	
"T"	T	0	2	1000	1000	0	0
						Limit 0	
							Limit 4

When the problem is decomposed, each project becomes a separate subproblem. The only limiting constraints within each subproblem are the precedence constraints (*e.g.*, A1 precedes A2, A2 precedes AT). The cost of a subproblem schedule is the numbered activity costs (which depend on when the activities are scheduled) and the project completion cost.

A fifth subproblem is composed of the program terminal activity. The only constraint within this subproblem (when severed from the program-level constraints) is that activity T must occur no earlier than its early start time (Time 2).

Figure 6-6 shows three alternative schedules for the program. Note that the representation of the project and program terminal activities is a straight vertical line lying between two time periods. The line actually falls at the back side of the time unit with which it is associated. For example, if the program terminal activity T occurs at its early start time of 2, the line representing the activity is on the border of time units 2 and 3. Note also that the program horizon is 8 (the sum of all non-zero duration activities).



UB - LB = 2014 - 1012 = 1002 > d for Projects A through D

Figure 6-6. Sweeney-Murphy Optimality Theorem Counterexample Chart

The first set of schedules in Figure 6-6 are the optimal subproblem solutions if the program-level constraints are relaxed. Each activity starts at its early start time (including activity T). The total cost of this set of solutions is 1012; this is the lower bound LB. The second set of schedules is the worst subproblem solutions where every activity starts at its late start time. The importance of this set of solutions is that it defines the maximum possible value for the ? of each subproblem (the worst solution minus the best solution). These values are ? = 24, 24, 24, 18, and 6000, respectively. Note that ? for Subproblem D is less than that of Subproblems A, B, and C, because the cost for delaying project completion is only 1 per time unit rather than 2 per time unit.

When program-level constraints are now considered, the set of best solutions to the subproblems do not form a feasible combination. Note that the demands for the program-level resource are marked inside the gray boxes representing the activities. The problem is that the total demand for the program-level resource in Time 2 exceeds the availability of four. To resolve this conflict, any of activities A2, B2, C2, or D2 could be shifted one unit to the right. The optimal program schedule results from shifting Activity D2, because the cost of doing so is only two (one for delaying D2 and one for delaying DT), while the cost of delaying any of the other activities is three (only one for delaying the activity itself, but *two* for delaying completion of its corresponding project).

Assuming all subproblem solutions are generated, the total cost of the optimal schedule to (MP) and to (P) is 2014. This is UB. So $? ? UB ? LB$ is 1002 which is larger than all subproblem ? except for subproblem "T." Hence, the condition of the Sweeney-Murphy Optimality Theorem is not met, but an optimal solution has been found.

<u>Choice of *k*</u>. One choice that must be made to implement Sweeney-Murphy Decomposition is the number of best solutions, *k*, to pass from each subproblem to the master problem. The smaller *k* is, the less time spent generating the set of subproblem solutions. If *k* is small, the master problem also takes less time to enumerate the combinations of subproblem solutions. Therefore, *k* should ideally be as small as possible, while assuring optimality.

On the other hand, though *k* should ideally be chosen as small as possible, the lower *k* is, the higher the possibility that the algorithm may not find an optimal to (P) in the first iteration. If the algorithm has to conduct subsequent iterations, overall solution time could increase significantly. The only leveraging provided by the previous iteration is that the previous subproblem solutions can be used to partially initialize the arrays of subproblem solutions. If k_1 subproblem solutions were generated in the first iteration and $k_2 ? k_1$ subproblem solutions are to be generated in the second iteration, then the first k_1 rank-ordered positions in the solution array are filled with the previous set of best solutions and the remaining $k_2 ? k_1$ positions are filled with an appropriately large value (*e.g.*, 999999). Since fathoming of branches in the subproblem solver is based on the value of the *k*-th best solution (an arbitrarily large value), even the leveraging from the previous solutions is likely to be minimal. Consequently, the time required to solve the subproblem in subsequent iterations is most likely at least as long as the time to solve the subproblem in the first iteration.

Another compounding factor is the possibility that only a subset of subproblems may need to be resolved at a subsequent iteration, if any. As a result, an *a priori* attempt to find just the right value of *k* is a difficult task at best. The right choice is a function of the time required to solve the subproblems for a smaller set of solutions versus a larger set, the probability that a subproblem will need to be resolved in a subsequent iteration (which also depends on the size of the set in the previous iteration), and the amount of duplicative effort to solve a subproblem again for a larger value of *k*. Of course, all of these values (the time to solve the subproblems for varying sizes of *k* and the probability that the sets of subproblem solutions of some given size will provide an optimal combination) are dependent on the characteristics (network complexity, resource strength, etc.) of each subproblem and of the master problem. The implementer of Sweeney-Murphy Decomposition, then, must choose the initial value of *k* based on experience, intuition, or empirical analysis if time allows. An empirical analysis is provided in this study.

Figures 6-7 and 6-8 (repeated from Chapter V) provide some insight into the time required to solve the subproblems for varying values of *k*. Based on values of *k* of 1, 10, 100, and 1000, these figures show, respectively, the average time required per solution and the overall solution time for each *k*. The marginal amount of time required to find one additional solution decreases significantly as *k* increases. The resultant growth in overall solution time also flattens out. If one is risk averse, preferring to solve each subproblem for a large enough *k* so that iteration is not necessary, then these results provide some assurance that if *k* is larger than minimally necessary, overall solution time does not suffer significantly.

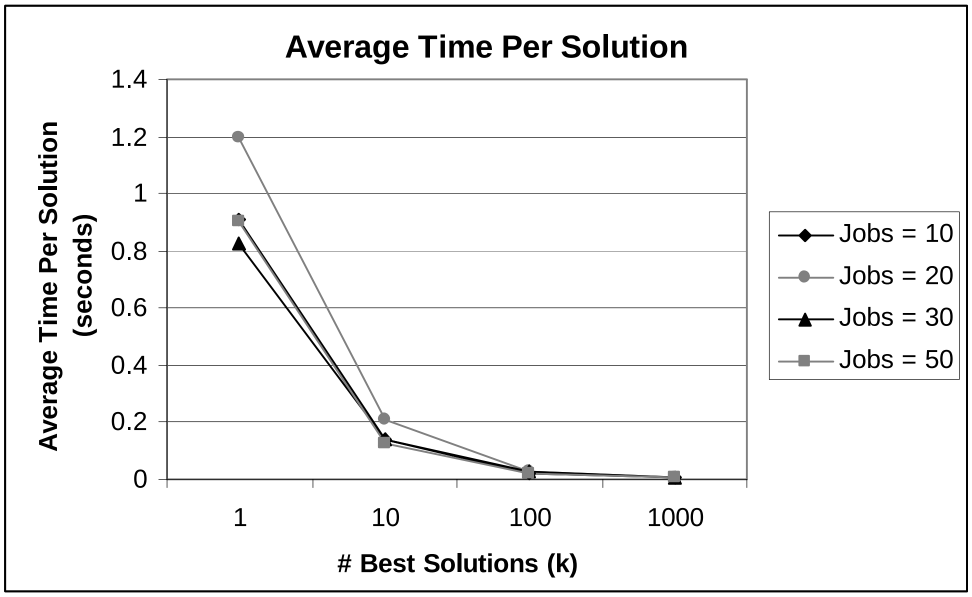


Figure 6-7. Average Time Per Solution

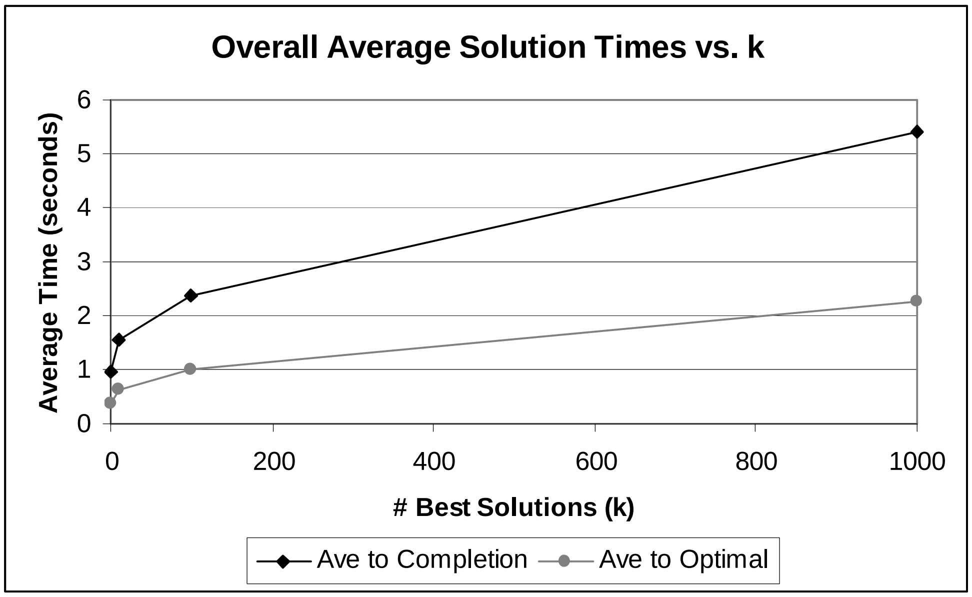


Figure 6-8. Overall Average Solution Times Versus k

For the risk prone who is willing to gamble multiple iterations to reduce initial subproblem solution time, their payoff depends on the structure of the master problem. If the master problem contains no program-level constraints, then the optimal subproblem solutions form an optimal solution to (P). In this case, solving the subproblems for a very small k would be preferable. If, however, the master problem is highly constrained, then a large number of solutions from each subproblem may be required even to find a feasible combination, not to mention an optimal one.

The section on testing, below, tests various schemes for choosing k based on the respective difficulty of subproblems and the master problem.

Choice of Multipliers. A second choice required for implementation of Sweeney-Murphy Decomposition is initial values for the Lagrangian multipliers. Multipliers in the subproblems are a counterbalance to k. The better the multipliers, the fewer the subproblem solutions required to find an optimal solution to (P). The easier it is to generate the k solutions, the less important it is to find the optimal multipliers (Sweeney and Murphy, 1979: 1133).

In the context of Lagrangian Relaxation, multipliers are used in the subproblem objective functions to weight the importance of the coupling constraints and the scarcity of resources. If the multipliers are zero, then the coupling constraints are ignored when solving the subproblems. If the multipliers are arbitrarily large, then the subproblems over-emphasize the coupling constraints, leading to a solution which may be harmful to the possibly competing interests of the subproblem. *Good* multipliers, by contrast, are most likely somewhere between zero and an arbitrarily large number. Good multipliers influence the subproblems to provide solutions to the master problem which lead to tight bounds on the optimal solution to (P). That is, $\Delta = UB - LB(\boldsymbol{\mu})$ is minimized.

The value of $\boldsymbol{\mu}$ which maximizes $LB(\boldsymbol{\mu})$ and which, therefore, provides the best choice of $\boldsymbol{\mu}$ is the solution to problem (D), the Lagrangian dual of (P) (shown in Figure 6-9).

Lagrangian Dual of (P)

Problem (D):

$$\max_{\boldsymbol{\mu}\geq 0} LB(\boldsymbol{\mu}) = \max_{\boldsymbol{\mu}\geq 0} \min_{\boldsymbol{\mu}\geq 0} \left\{ \sum_{p=1}^{P} (\mathbf{c}_p - \boldsymbol{\mu}\mathbf{A}_p)\mathbf{x}_p + \boldsymbol{\mu}\mathbf{b}_0 \middle| \mathbf{B}_p\mathbf{x}_p \leq \mathbf{b}_p, \mathbf{x}_p \geq 0 \text{ and integer } \forall p \right\} \quad (18)$$

Figure 6-9. Lagrangian Dual of Original Problem (P)

Sweeney and Murphy (1979: 1132) discuss the difficulty of solving (D) and suggest two alternative approaches for choosing multipliers. Both are LP-based methods, using Lagrangian duality theory common in the literature (*e.g.*, Geoffrion, 1974; Fisher, 1981). One approach for choosing $\boldsymbol{\mu}$ is to solve the LP relaxation of (P) and use the dual variables associated with the coupling constraints. If it is inconvenient to solve the LP relaxation of (P), Sweeney and Murphy suggest initially setting $\boldsymbol{\mu} ? \mathbf{0}$, and then, after solving (MP) the first time, set $\boldsymbol{\mu}$ to the dual variables associated with coupling constraints in the LP relaxation of (MP). This second method is used by Sweeney and Murphy in a sample problem, as well as by Deckro *et al.* (1991). Unfortunately, neither approach can be used for the MRCMPSP-GPR/EXP.

As previously mentioned, both approaches proposed by Sweeney and Murphy are LP-based. Neither the subproblems nor the master problem of the MRCMPSP-GPR/EXP are solved with LP-based methods, so multipliers based on these approaches are not readily available. Still, it is important to identify a means by which the master problem can influence the solutions provided by the subproblems. For this, a more economic interpretation of the multipliers is in order.

Lagrangian multipliers are the dual variables associated with the constraints of an LP. When an LP is solved to optimality, the dual variables, also known as *shadow prices*, reflect the increase in the objective function value possible if the right-hand side of the associated constraint is raised by one unit. That is, a shadow price is the marginal value of one additional unit of resource given the current optimal solution. Hence, for binding constraints, shadow prices may be positive, reflecting the maximum amount the decision-maker should be willing to pay for an additional unit of resource. For non-binding constraints, shadow prices are zero, because there is no value in purchasing more units of a resource which is already in excess. The challenge in this study is to estimate shadow prices without solving an LP.

Adding to the challenge of estimating shadow prices is the reality that the shadow prices must be estimated before the problem is initially solved. In fact, the shadow prices need to be estimated in order to solve the problem. This leads to the concept of *provisional dual prices*, proposed by Baumol and Fabian (1964). Provisional dual prices reflect the marginal profitability of a resource when used as prescribed in the current solution, not necessarily the optimal solution. The primary difference between these and regular duals is that these prices may be negative (Baumol and Fabian, 1964: 7).

If the Dantzig-Wolfe Decomposition approach could be used for solving the MRCMPSP-GPR/EXP, there would be an iterative process where the program sets initial prices that it charges the projects for program-level resources. These prices are *recalculated* each time the master problem is solved and these new prices are passed to the projects. With Sweeney-Murphy Decomposition, good prices need to be initially estimated, especially since a single iteration would be ideal. In this situation, the provisional dual prices are, in essence, a *forecast* of the value of resources, since they must be estimated even before an initial allocation is made.

Since the goal of multipliers is to enable the program to impact the decisions made by the projects, Baumol and Fabian suggest that:

> The means to induce subdivisions to increase activities which produce external economies and to reduce activities which produce external diseconomies is accomplished by the addition to divisional earnings of a per unit subsidy or bonus of appropriate magnitude for every external economy yielding output, and a per unit penalty on those products which involve diseconomies. Baumol and Fabian (1964: 4)

Substituting *project* for *subdivision*, projects should be rewarded for using resources available in excess and should be charged for using resources that are in high demand by other projects. Since project activities have multiple modes of execution, it is difficult to know *a priori* exactly *how much* of any given resource a project will demand. Activities can also be scheduled in multiple time periods, so it is difficult to know *when* the resources will be demanded. The remainder of this subsection is devoted to the proposition of four potential approaches for choosing multipliers, or provisional dual prices. Each approach is tested in the next section.

The first two approaches for choosing multipliers are not elegant, but they are valid approaches. The first approach is to choose zero for all multipliers and the second approach is to choose an arbitrarily large value for the multipliers. As previously discussed, neither of these approaches would be expected to perform well, but they do, in some sense, provide bounds by which other approaches can be evaluated.

The third approach for choosing multipliers is based on the work of Nauss (1979). To estimate the marginal benefit of resources in an IP, Nauss solves the IP to optimality and, then in turn, varies the right-hand side of each resource constraint by one unit and re-solves the problem. The estimated marginal benefit of a resource is then the difference between the optimal objective function value of the original problem and the optimal objective function value of the problem with the respective resource constraint varied. Obviously, if the MRCMPSP-GPR/EXP could be solved to optimality easily enough to uses Nauss' method directly, the whole purpose behind finding the multipliers in the first place would be moot. Nauss' concept, though, can be used.

In what this work refers to as a *Modified Nauss Approach* (MNA), the original problem (P) is solved to find a feasible solution. The Scheduler developed in Chapter V is applied to (P), but is stopped before completion. Termination of the Scheduler is triggered only after it finds at least one solution and has run for some user-defined length of time. Use of the Scheduler as a heuristic for these purposes is not only convenient, but is guaranteed to find a feasible solution, if one exists. Drexl and Grunewald (1993) point out that, in general, heuristics may, especially in the presence of scarce resources, not even be able to find a feasible schedule. The Scheduler is designed specifically for this problem type and its use assures that a feasible sol utions is found and that multipliers can be calculated. The analysis in Chapter V also showed that the optimal solution to a problem is generally found relatively quickly using the Scheduler (most of the computing time is used to verify the solution), so a relatively good heuristic solution should be found.

To estimate the marginal benefit of nonrenewable resources, the regular availability of each nonrenewable resource is varied, in turn, by one unit and (P) is re-solved until the Scheduler terminates according to the criteria previously selected. The difference in solutions represents the marginal benefit of an additional unit of nonrenewable resource. (A similar approach was used by Van Hove, 1998).

Renewable resources are more difficult because there is not a single availability for each resource. In fact, each resource has a separate availability for each time period. Therefore, to estimate the marginal benefit of a renewable resource, the regular availability of the resource in each period is concurrently varied by one unit and (P) resolved until the Scheduler terminates according to the criteria previously selected. The difference in solutions represents the total benefit of an additional unit of nonrenewable resource in each time period. To avoid over-estimating the value of the resource per period, the total benefit is divided by the program duration in the heuristic solution.

The fourth approach for choosing multipliers is based on the concept of *Average Utilization Factor* (AUF) described by Kurtulus and Davis (1982) and Kurtulus and Narula (1985). As defined by Kurtulus and Davis, the AUF is calculated for each time period and is the ratio of the total amount of resource required to the amount available, based on a time only analysis of the program. The value of this measure is, in principle, equivalent to the Resource Strength (RS) used in previous chapters. However, RS is not generally known, so AUF must be calculated. RS was used in the generation of problems, is a factor known *outside* the problem, and is the same for each time period, by construction. AUF, by contrast, is used *within* the problem where the value is not known *a priori*, but must be calculated. AUF may also vary between time periods.

The AUF Approach, then, consists of applying the Generalized Critical Path Method (GCPM), developed in Chapter IV, to the problem. This provides the time only schedule of the program. The next step is to calculate the total demand for each resource in each time period. Recall that the GCPM uses the activity execution modes of least duration. These shortest-duration modes, however, are not necessarily the mode choices the projects would make from a resource perspective. To assure that the AUF accounts for the possibly higher resource demands of alternative modes, an activity's demand for a resource is based on the highest possible demand from among the activity's modes. The AUFs for renewable and nonrenewable resources are, then, calculated using the following equations:

$$AUF_{qt}^{R} = \frac{\sum_{p=1}^{P} \sum_{\substack{i=1 \\ t \in [s_i, s_i + d_{i1} - 1]}}^{J_p} \max_{1 \le m \le M_i} \{ r_{imqt}^{R} \}}{R_{qt}^{R}}, \forall q, t \qquad (19)$$

$$AUF_{q}^{N} = \frac{\sum_{p=1}^{P} \sum_{i=1}^{J_p} \max_{1 \le m \le M_i} \{ r_{imq}^{N} \}}{R_{q}^{N}}, \forall q \qquad (20)$$

where

AUF_{qt}^{R}	=	AUF for renewable resource q at time t
AUF_{q}^{N}	=	AUF for nonrenewable resource q
r_{imqt}^{R}	=	requirement for renewable resource q at time t by activity i in mode m
r_{imq}^{N}	=	requirement for nonrenewable resource q by activity i in mode m
R_{qt}^{R}	=	availability of renewable resource q at time t
R_{q}^{N}	=	availability of nonrenewable resource q
s_i	=	start time of activity i
d_{i1}	=	duration of activity i in mode 1

If $AUF ? 1$ for some resource in some time period, the demand for the resource is no more than its availability. The price charged by the program for this resource in this time period should be zero.

If $AUF ? 1$ for some resource in some time period, the demand for the resource exceeds its regular availability and projects should pay a premium for using this resource. The price depends on the degree to which *AUF* exceeds one. If the total demand for the resource is less than its regular *plus* expediting availability in the stated time period, the demand for the resource may be met using expediting resources. In this case, the price charged by the program for a unit of this resource is the cost of a unit of expediting resource (a value which is given in the problem statement). If the total demand for the resource is greater than its regular *plus* expediting availability, then resource feasibility is only achievable through some combination of activity mode changes and / or activity delays (in the case of nonrenewable resources, only mode changes).

The minimal cost combination of mode changes and activity delays could be calculated to provide the estimated cost of resource feasibility in the given time period. Such a calculation would have to account for the *ripple* effect that the changes would have on subsequent activities and time periods. If an activity is delayed, other activities may also need to be delayed, adding to the cost of the initial delay and possibly creating resource infeasibilities in future time periods. Even if a simple mode change is made, the duration of the changed activity may increase, causing the same effects on future activities and time periods caused by an activity delay.

Given the reality that the resource demands (upon which the resource infeasible condition results) is only a rough estimate, the computational cost of calculating the minimal cost combination of mode changes and activity delays would not likely prove worthwhile. Instead, the cost of the time-only schedule is calculated. The cost of modes and of expediting resource usage is included in the time-only cost, but resource infeasibilities are temporarily ignored. In essence, it is the resource infeasibilities in the time only schedule that force an alternate solution.

The original problem is, then, solved heuristically as in the Modified Nauss Approach, yielding a resource feasible solution. The difference in the costs of the time-only schedule and the resource-feasible schedule is calculated. This difference is the consequence of the resource infeasibilities in the time-only schedule. Finally, the cost difference is partitioned among the resources whose $AUF ? 1$ and whose demand could not be met with expediting resources. The partitioning is based on the relative contribution to resource infeasibility of the violating resources as follows:

$$p_{qt}^{R} ? ? \;\; ? ? \frac{AUF_{qt}^{R}}{? \underset{t}{?} \underset{q}{?} AUF_{qt}^{R} ? \underset{q}{?} AUF_{q}^{N} ?} ?, ? q, t \qquad (21)$$

$$p_{q}^{N} ? ? \;\; ? ? \frac{AUF_{q}^{N}}{? \underset{t}{?} \underset{q}{?} AUF_{qt}^{R} ? \underset{q}{?} AUF_{q}^{N} ?} ?, ? q \qquad (22)$$

where

? is the total cost difference between the time only and resource feasible schedules

p_{qt}^{R} is the price charged for renewable resource q in time period t

p_{q}^{N} is the price charged for nonrenewable resource q

In this way, the program shares the cost of a resource feasible schedule by charging the projects for using resources in time periods that cause resource conflicts in the time only schedule.

All of the proposed methods for choosing multipliers are, to this point, theoretically based. It remains to be seen under what conditions each method performs well. Testing of the approaches is presented in a subsequent section.

Acceleration Schemes

A number of optional schemes may be used to increase the speed of the decomposition approach. The acceleration schemes presented here can be used together with the multipliers, but the schemes are more direct approaches for bounding and constraining the subproblem solutions. The *Incremental Enumeration* scheme also provides quicker solution of the master problem. When Sweeney and Murphy presented their decomposition approach, they solved the master problem to completion without testing solutions for optimality until the end. In fact, an optimal solution may be found and proven to be optimal early in the enumeration. The *Incremental Enumeration* scheme makes a more proactive use of the optimality conditions to find and confirm an optimal solution before completely enumerating the master problem.

Subproblem Solution Bounding. This scheme is based on finding a good solution to the original problem (P) before the subproblems are solved. With a good solution to (P), upper bounds on the subproblem solution values can be determined and used to initialize the subproblem solution arrays. The upper bounds permit faster fathoming of unproductive subproblem solutions than the arbitrarily large values with which the solution arrays would otherwise be initialized.

This option is executed by first finding a heuristic solution (*HS*) to (P). The Scheduler is used for this purpose as was previously done for finding Lagrangian multipliers. A minimal cost for each subproblem is then calculated. A minimal cost is easily obtained by using the early start times (based on the GCPM) of each activity in the subproblem. Using the minimum base and incremental mode costs possible for each activity (*i.e.*, the minimum from among the modes of the activity), the mode costs for all activities, starting at their early start times, are added. This *early-start-time* (EST) *schedule* may not be resource feasible, but it does provide a minimal cost, z_p^{EST}, for Subproblems *p*, $1 \le p \le P$.

Finally, an upper bound, UB_{p^*}, on the objective function value for subproblem *p** is obtained by subtracting the sum of the early-start-time schedule costs, z_p^{EST}, for the other subproblems, $p \ne p^*$, from the heuristic solution, *HS*, to (P). That is,

$$UB_{p^*} = HS - \sum_{p \ne p^*} z_p^{\text{EST}} .$$

As solutions are generated for Subproblem *p**, any solution greater than UB_{p^*} is fathomed since such a solution would yield a program cost greater than the heuristic solution to (P) previously found.

Series Approach. Since subproblems are solved in series (vice in parallel), information obtained from the *k*-best solutions to Subproblems 1 through *p*-1 can be used when solving Subproblem *p*. The goal is to eliminate from the set of *k*-best solutions to Subproblem *p* as many solutions which would not be feasible to (P) when used in concert with any of the solutions to Subproblems 1 through *p*-1.

Three types of information obtained from Subproblems 1 through *p*-1 can be used when solving Subproblem *p*: an upper bound on the objective function value, reduced activity start time windows, and constraints on resource use. This information is obtained and used as follows:

?? Upper Bound on Objective Function Value. As with the *Subproblem Solution Bounding* scheme above, an initial heuristic solution to (P) can be obtained. Once Subproblems 1 through *p*-1 have been solved, the best solution values to Subproblems 1 through *p*-1 can be added to the value of the early-start-time schedules of Subproblems *p*+1 through *P*. This sum can be subtracted from the heuristic solution value, *HS*, to obtain a new upper bound on the solution to *p*. When solving Subproblem *p**, the upper bound on the objective function value is given by

$$UB_{p^*} \; ? \; HS \; ? \underset{p \, ? \, p^*}{?} z_{p1} \; ? \underset{p \, ? \, p^*}{?} z_p^{\text{EST}} \; .$$

?? Reduced Activity Start Time Windows. The *k*-best solutions to each of Subproblems 1 through *p*-1 can be compared to find the earliest time (within the set of *k*-best solutions) that each activity in the subproblem starts. The latest start time of each activity can be found in like manner. Before solving Subproblem *p*, the early and late start times for activities in previously solved projects are fixed and the GCPM used to calculate new early and late start times for Subproblem *p*. Doing so reduces the number of solutions in the set of *k*-best solutions to Subproblem *p* which are precedence infeasible when used in combination with the solutions to Subproblems 1 through *p*-1.

?? Constraints on Resource Use. The limitations on program-level resources can be directly considered when solving each of the subproblems to reduce the number of subproblem solutions which are resource infeasible at the program-level. To do so, the set of program-level resources can be added to the set of project-level resources in each subproblem. The full complement of regular and expediting resources is initially made available to each subproblem, and there is no charge to the subproblem for using expediting resources. Program-level resources, therefore, do not impact the cost of a subproblem solution, but serve only to eliminate subproblem solutions which cannot possibly lead to feasible solutions to (P). Once Subproblems 1 through *p*-1 have been solved, the minimal usage of program-level resources from among the *k*-best solutions to each subproblem is used to decrement the availability of these resources to Subproblem *p*. As a result, each subproblem is increasingly more constrained by the program-level resources and its set of *k*-best solutions provides a higher percentage of solutions which are resource feasible when used in combination with the solutions to the previous subproblems.

Incremental Enumeration. Each time a feasible master problem solution is found, recalculate ? and the subproblem deltas, $?_p$, and test for optimality. In this case, however, the $?_p$ for each subproblem is calculated using the current subproblem solution rather than the *k*-th best solution to that subproblem. The advantage of this approach is that if the feasible master problem solution is an optimal solution to (P), this test might prove the solution to be optimal without having to implicitly enumerate all $k_1 ? k_2 ? k_3 ? \quad ? k_P$ possible subproblem solution combinations. Note that failing the optimality test at this point does not imply that the current feasible solution is not optimal, only that enumeration of solution combinations must continue until optimality can be established. Note, too, that this option is useful only if the primary consideration is to find an optimal solution rather than finding the *k*-best solutions to (MP). While this option leads to an optimal, it may terminate before the *k*-best solutions to (MP) have been found.

Test Problem Design

The experimental design for testing the approaches presented in this chapter can be divided into two parts: the *problem design* and the *solution design*. The solution design, or the manner in which solution approaches are applied to the problems, is discussed in subsequent sections in conjunction with the results of those approaches. This section discusses the problems generated to test the solution approaches.

Each problem used for testing can be defined in terms of its program structure, the difficulty of its component projects, and the difficulty imposed by the program-level constraints. These problem characteristics are described below, followed by a discussion of how the characteristics are combined to form a set of 54 test problems used throughout the remainder of the chapter.

Program Designs. Five basic program structures are used for testing. These program structures differ in the way projects relate to each other temporally and in the presence or absence of program-level renewable and nonrenewable resources. The five program structures are depicted in Figure 6-10 as Program A through Program E. In each depiction, temporal relationships are represented by the network structure presented, where the blocks represent distinct projects and the circles represent dummy start and terminal activities. Lines between projects (as in Program D) represent generalized precedence between activities in one project and activities in another project. The vertical bar labeled *NR* denotes the presence of program-level nonrenewable resources while the horizontal bar labeled *RR* denotes the presence of program-level renewable resources.

Program A consists of a set of nearly independent projects. There are no program-level resources and the projects are tied together merely by a dummy start activity and an end activity. The projects, however, cannot be solved in isolation because the end activity represents the completion of the program, which is dependent on the completion times of the projects. Decisions made at the project level, consequently, impact the completion cost (and overall cost) incurred by the program.

Program B consists of a set of projects which are related only by their requirements for common, program-level, nonrenewable resources. In the absence of (1) expediting resources at the project *and* program levels and (2) maximum time lags between activities, this program becomes the multi-project GMRCMPSP addressed by Van Hove (1998).

Program C builds upon Program B with the addition of program-level renewable resources.

Program D is an extension of Program C, where generalized precedences between activities in different projects are added.

Finally, Program E is, in some sense, a special case of Program D. Program E has generalized precedences between activities in different projects, but the precedences exist only between the terminal activity of one project and the start activity of the next. Consequently, projects follow one from another. Program E may also contain renewable resources controlled by the program, but since projects do not overlap in time, these program-level resources can be treated as though *passed down* to the projects. The same cannot be said of program-level nonrenewable resources where their allocation to projects constrains the execution options of the projects' activities.

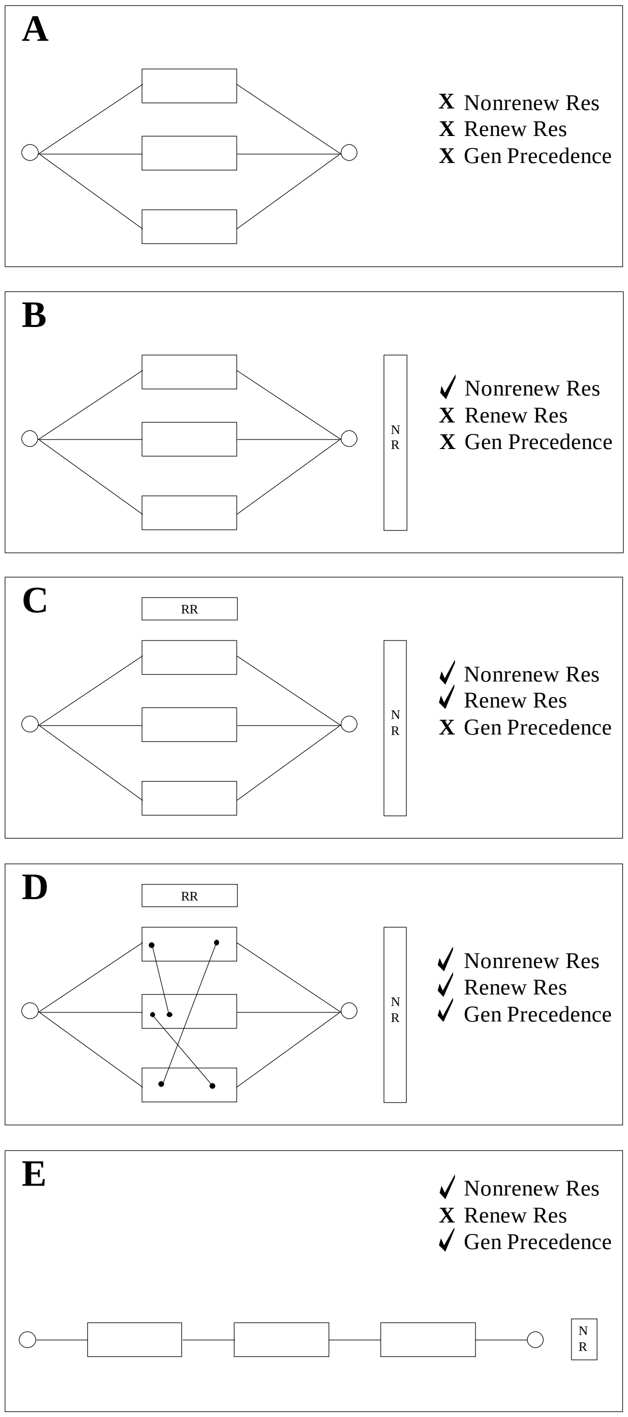


Figure 6-10. Program Designs

<u>Project Level Difficulty</u>. The projects which comprise each program differ in their degree of difficulty to schedule. Based on the results of Chapter V, six problem parameters were identified

as being significant factors in problem solvability. With the exception of *number of activities*, two levels of each factor were chosen and partitioned to form an *Easy* set of parameters and a *Hard* set of parameters. (*Number of activities* is dealt with separately and discussed later.) Table 6-2 outlines the significant factors and the levels chosen to form the *Easy* and *Hard* sets of parameters. Note that fewer activity execution modes and fewer resources make easier projects, while higher network restrictiveness and higher regular resource strength contribute to easier projects. The parameters held constant for problem generation are outlined in Table 6-3.

Table 6-2. Project-Level Generation Parameters Which Vary

PARAMETER	LEVELS	
Designator	"Easy"	"Hard"
Number of Modes Per Activity	1	3
Project Network Restrictiveness	0.75	0.25
Number of Renewable/Nonrenewable Resources	1	3
Regular Renewable/Nonrenewable Resource Strength	1.00	0.50
Total Renewable/Nonrenewable Resource Strength	0.00	0.50

Table 6-3. Project Level Generation Parameters Held Constant

PARAMETER	Min	Max
Job Duration, Maximum	10	10
Lag Fraction	0.20	0.20
Minimal Lag	-2	2
Maximal Lag	4	8
Renewable/Nonrenewable Resource Factor	1.00	1.00
Resource Demand	1	10
Base Project Penalty	500	750
Project Penalty Increment	400	500
Base Mode Cost	50	100
Mode Cost Increment	50	100
Expediting Resource Cost	0	50

Program Level Difficulty. Programs also differ in the difficulty of the program-level constraints. Depending on the program structure being addressed and its corresponding features, program-level generalized precedences and resources are generated to be either *Easy* or *Hard*. Table 6-4 shows the program-level parameters which vary and the values which define the *Easy* and *Hard* sets. Note that the factor levels used to form the *Easy* and *Hard* sets were chosen based on results of Chapter V. While fewer resources and higher regular resource strength should clearly make for easier program-level constraints, it is unclear in advance of testing whether higher program network restrictiveness really makes the problem easier or if it makes the problem harder. If the problem were solved as a single super-project, higher network restrictivenesswould certainly make the problem easier. On the other hand, the higher restrictiveness may make it more difficult in the decomposition approach to find feasible sets of subproblem solutions. Since the exact impact of program restrictiveness is not known*a priori*, the values have been chosen consistent with the results of Chapter V.

Table 6-4. Program-Level Generation Parameters Which Vary

PARAMETER	LEVELS	
Designator	"Easy"	"Hard"
Program Network Restrictiveness	0.75	0.25
Number of Renewable/Nonrenewable Resources	1	3
Regular Renewable/Nonrenewable Resource Strength	1.00	0.50
Total Renewable/Nonrenewable Resource Strength	0.00	0.50

Table 6-5. Problem Design

Program Structure	Projects	Jobs Per Project	Total Jobs	Project / Program Difficulty	Total Problems
A	4	4 8 12	18 34 50	Easy / NA Hard / NA	6
B	4	4 8 12	18 34 50	Easy / Easy Easy / Hard Hard / Easy Hard / Hard	12
C	4	4 8 12	18 34 50	Easy / Easy Easy / Hard Hard / Easy Hard / Hard	12
D	4	4 8 12	18 34 50	Easy / Easy Easy / Hard Hard / Easy Hard / Hard	12
E	4	4 8 12	18 34 50	Easy / Easy Easy / Hard Hard / Easy Hard / Hard	12
Total					54

Problem Generation. With program structures designed and the characteristics of *Easy* and *Hard* projects and programs defined, a total of 54 programs were generated using PAGER (described in Chapter IV). Each program consists of four projects, all projects being either *Easy* or *Hard*. Table 6-5 shows the design applied to problem generation.

Testing Results

Testing was conducted to:

1. Evaluate the alternate methods of determining multipliers
2. Assess the performance of the acceleration schemes
3. Evaluate alternate choices of *k*
4. Compare the decomposition approach to the single-project Scheduler from Chapter V.

All test problems were generated using the Program Attributes Generator with Expediting Resources (PAGER) presented in Chapter IV and solved using a 750 MHz, Pentium III processor with 256 MB of Random Access Memory (RAM).

Methods of Determining Multipliers. The 54 test problems outlined in the previous section were solved using each of the methods for determining multipliers. Each problem was solved to find a single optimal solution. At each iteration of the decomposition algorithm, 100 solutions from each subproblem were generated. The problems were also solved using the single-project Scheduler to find a single optimal solution. A solution time limit of 20 minutes per problem was imposed to control the total time to solve all test problems.

Figure 6-11 shows the percentage of problems which were solved to optimality within the time limit and the percentage which exceeded the time limit. The AUF method of determining multipliers was most successful at solving the set of problems within the time limit, followed by using no multiplier at all, then the MNA method, and the single-project Scheduler. Using an arbitrarily large number for the multipliers was least productive.

For the problems which solved to optimality within the imposed time limit, solutions times are reported in Table 6-6. Problem decomposition led to more problems solved and generally faster solution times than the single-project Scheduler, except when arbitrarily large multipliers were used. When comparing just the multiplier methods, though, the results in Table 6-6 are mixed, with no method clearly dominating the others. Using no multipliers at all had the best average solution time, but it found fewer solutions (the number reported in the *Count* column of Table 6-6) than the AUF method. The MNA method appears to be dominated by the AUF method and by using no mulitpliers, but further investigation is merited to determine if problem characteristics effect the performance of each method.

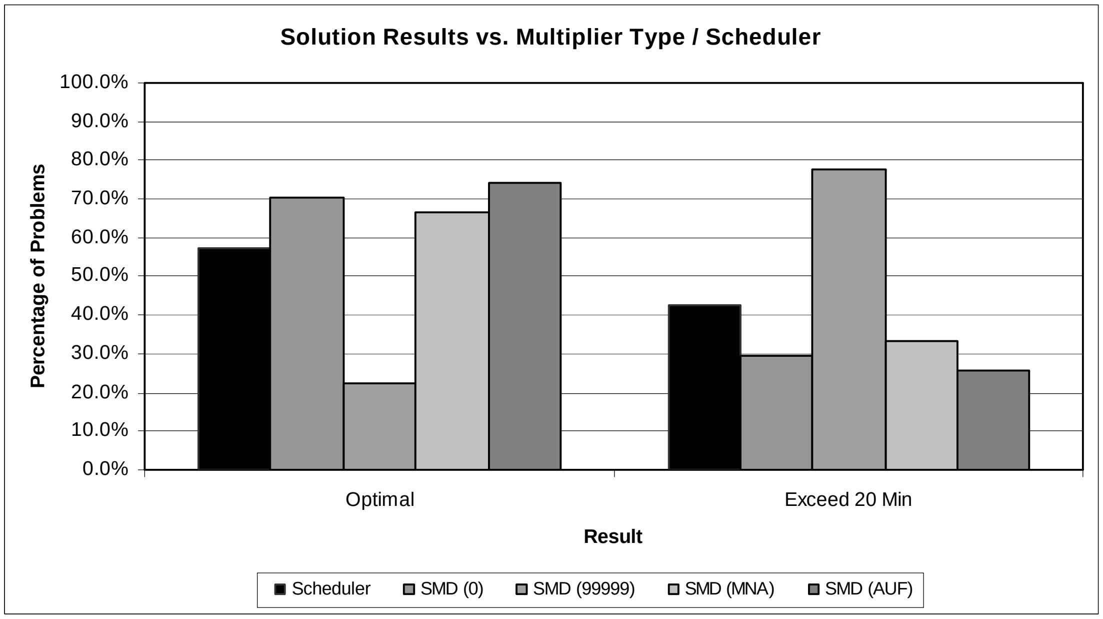


Figure 6-11. Solution Results vs. Multiplier Type / Scheduler

Table 6-6. Solution Time vs. Multiplier Type / Scheduler

		Solution Time (seconds)			
Approach	Count	Minimum	Average	Maximum	Std Dev
Scheduler	31	0.00	66.75	1030.23	216.95
SMD (0)	38	0.02	38.81	724.00	144.12
SMD (99999)	12	0.02	61.89	724.04	199.68
SMD (MNA)	36	0.02	56.75	724.03	156.85
SMD (AUF)	40	0.02	67.02	960.06	207.34

Solution results are also shown in Figure 6-12 versus the program designs. Program designs correspond to and are numbered consistent with Figure 6-10. The program design with no program-level constraints, Design A, was solved to optimality 100% of the time within the 20-minute time limit, while the design with renewable resources, nonrenewable resources, and precedence constraints at the program-level, Design D, was solved to optimality only 41.7% of the time within the 20-minute time limit.

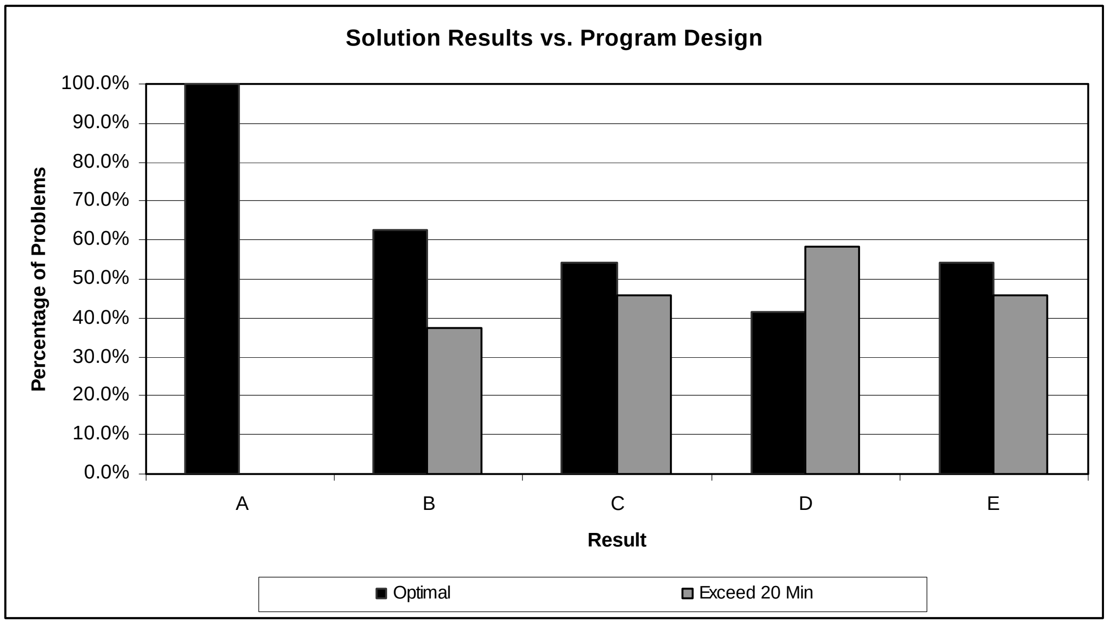


Figure 6-12. Solution Results vs. Program Design

Table 6-7 reports solution time as a function of problem characteristics. The table shows each program design, the difficulty imposed by the program and project constraints (as discussed in the previous section), and each multiplier approach. The corresponding count of and solution times of problems solved to optimality within the time limit are shown. Again, no method clearly dominates the others as one method solves more problems in some cases than other methods and solves fewer problems in other cases.

Since no method of finding multipliers is better in all cases than the others, the AUF method is used to determine multipliers for the remainder of testing since it succeeded at producing the most optimal solutions within the imposed time limit.

Acceleration Schemes. The set of 54 problems were solved with and without the acceleration schemes, this time generating 1000 solutions from each subproblem at each iteration. Table 6-8 shows the number of problems that were solved to optimality within the imposed 20-minute time limit, as well as solution times. Using the acceleration schemes not only resulted in finding more solutions, but the average solution time and standard deviation were smaller. Acceleration schemes were, therefore, used in testing and are, in fact, represented in the results presented above for multiplier methods.

Table 6-7. Solution Time vs. Problem Difficulty

Program Design	Program Difficulty	Project Difficulty	Multiplier Approach	Count	Solution Time (seconds) Minimum	Average	Maximum	Std Dev
A	NA	Easy	0	3	0.03	0.04	0.06	0.01
			99999	3	0.02	0.04	0.06	0.02
			MNA	3	0.03	0.04	0.06	0.01
			AUF	3	0.02	0.04	0.07	0.02
		Hard	0	3	0.05	245.97	724.00	338.07
			99999	3	0.05	246.01	724.04	338.07
			MNA	3	0.04	245.99	724.03	338.07
			AUF	3	0.04	246.00	724.03	338.07
B	Easy	Easy	0	3	0.03	0.04	0.06	0.01
			99999	1	4.32	4.32	4.32	0.00
			MNA	3	0.04	0.05	0.07	0.01
			AUF	3	0.04	0.05	0.07	0.01
		Hard	0	2	1.14	7.37	13.60	6.23
			99999	0	na	na	na	na
			MNA	1	1.14	1.14	1.14	0.00
			AUF	2	1.14	7.40	13.66	6.26
	Hard	Easy	0	3	0.03	0.04	0.05	0.01
			99999	0	na	na	na	na
			MNA	3	0.03	0.04	0.05	0.01
			AUF	3	0.03	0.04	0.05	0.01
		Hard	0	2	0.09	76.74	153.38	76.65
			99999	0	na	na	na	na
			MNA	3	2.38	110.84	221.75	89.57
			AUF	1	2.49	2.49	2.49	0.00
C	Easy	Easy	0	3	0.03	0.04	0.05	0.01
			99999	1	0.04	0.04	0.04	0.00
			MNA	3	0.02	0.04	0.06	0.02
			AUF	3	0.02	0.04	0.06	0.02
		Hard	0	3	0.28	185.44	546.21	255.13
			99999	0	na	na	na	na
			MNA	2	0.30	299.47	598.64	299.17
			AUF	3	0.29	203.08	598.62	279.72
	Hard	Easy	0	1	0.09	0.09	0.09	0.00
			99999	1	0.05	0.05	0.05	0.00
			MNA	1	0.05	0.05	0.05	0.00
			AUF	3	0.04	327.34	960.06	447.49
		Hard	0	1	2.10	2.10	2.10	0.00
			99999	0	na	na	na	na
			MNA	0	na	na	na	na
			AUF	1	6.94	6.94	6.94	0.00
D	Easy	Easy	0	2	0.03	0.04	0.04	0.00
			99999	1	0.03	0.03	0.03	0.00
			MNA	2	0.03	0.03	0.03	0.00
			AUF	2	0.03	0.03	0.03	0.00
		Hard	0	2	2.41	2.59	2.76	0.17
			99999	0	na	na	na	na
			MNA	1	2.27	2.27	2.27	0.00
			AUF	2	2.26	2.57	2.87	0.31
	Hard	Easy	0	1	0.03	0.03	0.03	0.00
			99999	1	0.03	0.03	0.03	0.00
			MNA	1	0.03	0.03	0.03	0.00
			AUF	2	0.03	153.15	306.27	153.12
		Hard	0	1	0.41	0.41	0.41	0.00
			99999	0	na	na	na	na
			MNA	1	48.16	48.16	48.16	0.00
			AUF	1	5.30	5.30	5.30	0.00
E	Easy	Easy	0	3	0.02	0.04	0.05	0.01
			99999	1	0.05	0.05	0.05	0.00
			MNA	3	0.02	0.03	0.05	0.01
			AUF	3	0.02	0.03	0.04	0.01
		Hard	0	1	0.53	0.53	0.53	0.00
			99999	0	na	na	na	na
			MNA	1	97.20	97.20	97.20	0.00
			AUF	1	0.49	0.49	0.49	0.00
	Hard	Easy	0	3	0.02	0.03	0.05	0.01
			99999	0	na	na	na	na
			MNA	3	0.03	0.03	0.04	0.00
			AUF	3	0.03	0.04	0.05	0.01
		Hard	0	1	3.13	3.13	3.13	0.00
			99999	0	na	na	na	na
			MNA	2	9.38	112.05	214.72	102.67
			AUF	1	9.26	9.26	9.26	0.00

Table 6-8. Value of Acceleration Schemes

		Solution Time (seconds)			
Acceleraton Schemes?	Count	Minimum	Average	Maximum	Std Dev
No	28	0.01	92.33	846.48	227.91
Yes	34	0.02	50.59	903.07	193.39

Choice of *k*. The test set of 54 problems was again solved, now for varying levels of *k*. The number of solutions, *k*, generated by each subproblem was varied from 100 to 1000 to 10,000. Figure 6-13 shows the percentage of problems solved to optimality within the 20-minute time limit for each level of *k*, while Table 6-9 lists solution time statistics for these problems. While more problems were solved with *k* = 100, a value of *k* = 100 did require, in some cases, more iterations. Figure 6-14 shows the number of iterations required to find the optimal solutions for each level of *k*.

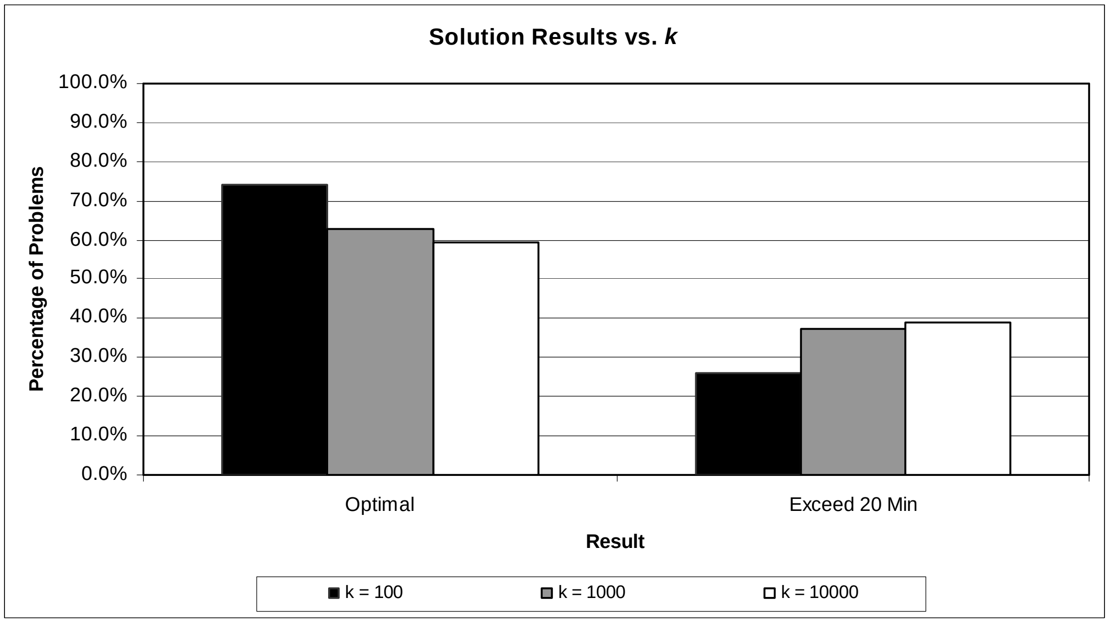


Figure 6-13. Solution Results vs. *k*

Note that in 9 cases, *k* = 100 required more than one iteration. Only one case of *k* = 1000 required more than one iteration (it required seven iterations) and *k* = 10,000 never required more than one iteration. Since there is a tradeoff between the reduced time to solve the subproblems for fewer solutions and the risk of having to iterate more than once, solution times for the varying levels of *k* need to be compared.

Table 6-9. Solution Times vs. *k*

		Solution Time (seconds)			
Approach	Count	Minimum	Average	Maximum	Std Dev
k = 100	40	0.02	75.19	783.87	212.44
k = 1000	34	0.02	50.59	903.07	193.39
k = 10000	32	0.03	64.69	960.13	219.18

Table 6-10 shows solutions times vs. *k*, arranged by program design. The easiest program design is Design A, having no program-level constraints. Since any optimal to each of the subproblems is feasible, and thus optimal, to the master problem, generating a single optimal for each subproblem would be sufficient for finding an optimal solution to the original problem. As expected, the smaller *k* is, the faster the algorithm solves a problem for Design A.

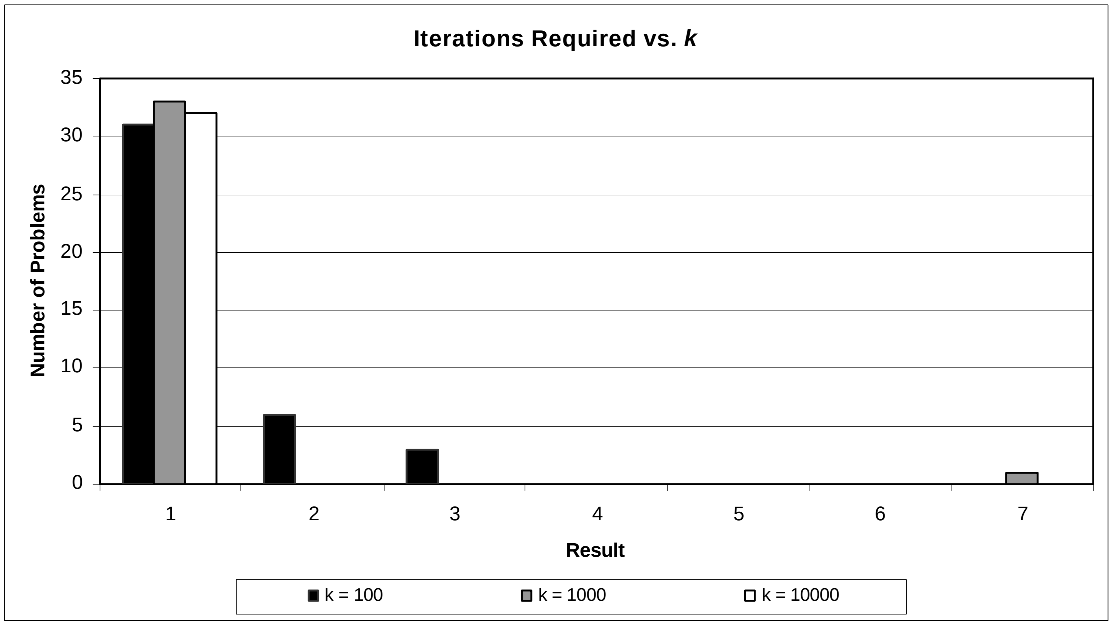


Figure 6-14. Iterations Required vs. *k*

For the other program designs, the results in Table 6-10 are not so clear since one value of *k* may solve fewer problems than another value of *k*, but the average solution time for the problems that were solved is lower. To better understand the distribution of solution times, refer to Figure 6-15.

Table 6-10. Solution Time vs. *k*

			Solution Time (seconds)			
Program Design	k	Count	Minimum	Average	Maximum	Std Dev
A	100	6	0.02	123.01	724.01	268.83
	1000	6	0.03	133.16	734.06	269.74
	10000	6	0.04	305.84	960.13	427.71
B	100	10	0.02	16.79	153.39	45.71
	1000	8	0.02	0.23	0.96	0.30
	10000	8	0.05	2.45	6.40	2.37
C	100	8	0.03	69.58	546.23	180.19
	1000	6	0.03	0.23	0.58	0.21
	10000	5	0.04	3.79	9.80	4.36
D	100	6	0.03	0.87	2.75	1.16
	1000	5	0.02	1.74	7.66	2.98
	10000	5	0.05	18.40	82.51	32.24
E	100	10	0.02	153.99	783.87	307.85
	1000	9	0.02	101.02	903.07	283.57
	10000	8	0.03	13.05	51.06	21.80

Figure 6-15 shows the distribution of solution times for each value of *k*. The vertical segments of the graph show the solution times for the corresponding value of *k*. The times are plotted on a logarithmic scale to better show the distribution of times at the bottom of the graph.

While most solution times are clustered at the bottom of the plot (relatively short solution times), there are a few for each value of *k* with relatively long solution times. Perhaps most noteworthy is the behavior of points in the middle of the plot. As *k* increases, there is a general shift of times upward, as well as a spreading out of solution times.

Given the greater number of problems solved and generally faster solution times with *k* = 100, generating 100 solutions for each subproblem at each iteration of the algorithm appears to be the most effective, even if, on occasion, more iterations must be made.

Comparison to Single-Project Scheduler. As previously seen in Table 6-6, the decomposition approach outperformed the single-project Scheduler in terms of number of problems solved within a 20-minute time limit and in terms of solution time.

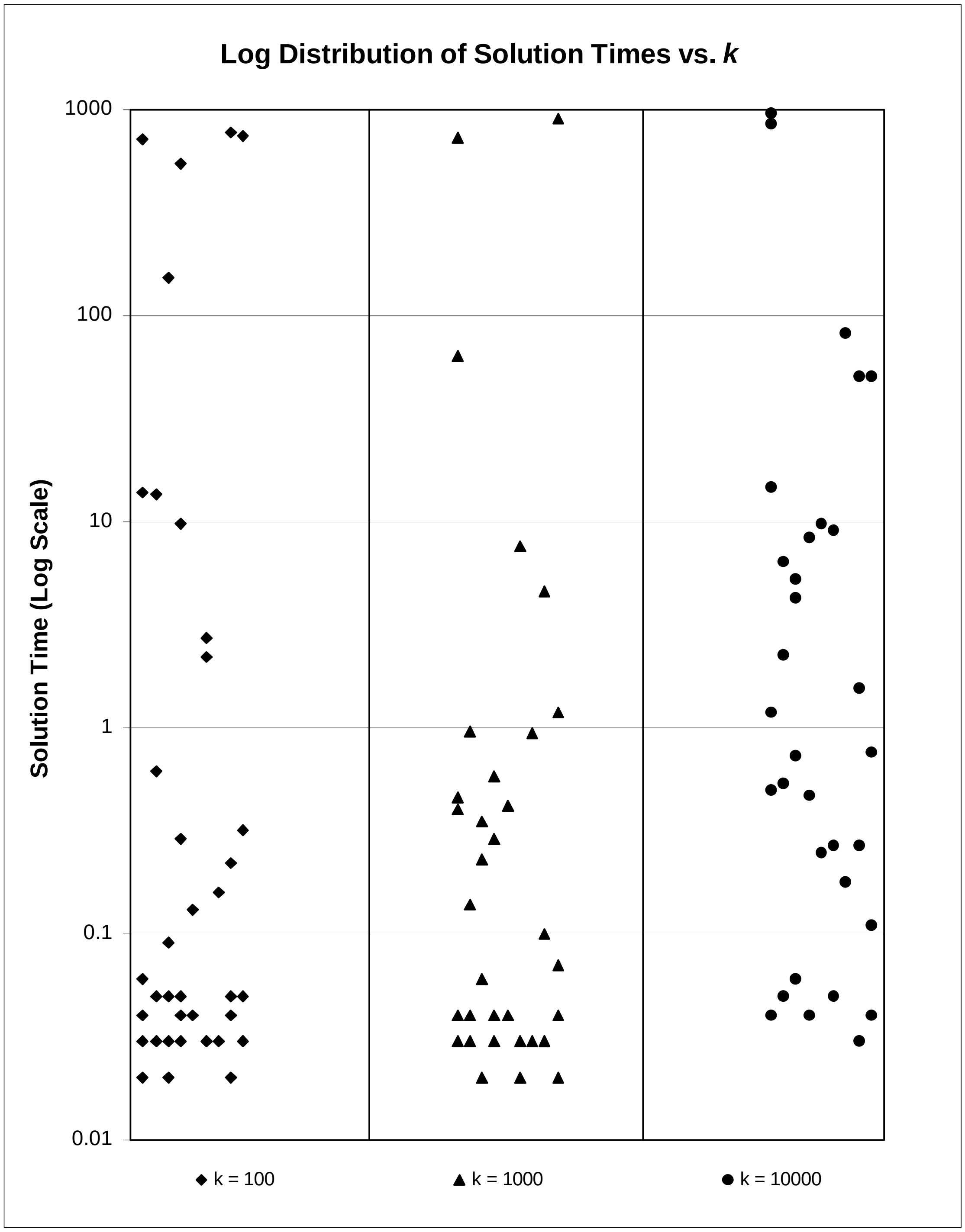


Figure 6-15. Log Distribution of Solution Times vs. *k*

Non-Convergence. While the focus of the analysis in this section has been on the problems that solved within a 20-minute time limit, the question remains how long it takes to solve the other problems. A number of the these other problems were solved with a two-hour time limit. One problem of Design B, with both difficult program- and project-level constraints, solved to optimality in just over 42 minutes, while a problem of Design C, also with difficult constraints, required just over 1.5 hours to solve to optimality.

On the other hand, a problem of type D and a problem of type E still failed to solve in the time allotted. In both cases, the objective function value found in 2 hours was no better than that found in 20 minutes. Because of the misuse of the Sweeney-Murphy Optimality Condition as being necessary and not just sufficient, it is possible that the failure to establish an optimal solution results from an inability to classify the best found solution as optimal rather than not being able to find an optimal solution. For this reason, finding a necessary condition to establish optimality is a worthwhile area for future research.

Summary and Conclusions

The decomposition approach presented in this chapter proved effective for solving the MRCMPSP-GPR/EXP, even for problems with as many as 50 activities. The decomposition approach solved more problems than the single-project Scheduler and in less time. The AUF method of determining Lagrangian multipliers appeared most useful, as did generating 100 solutions to each subproblem at each iteration. The lack of a necessary condition to establish optimality makes it difficult to determine if any given problem will converge to an optimal solution. However, even in the cases where the best-found solution cannot be established as optimal, the set of *k* solutions produced by the algorithm may still be considered good heuristic solutions.

VII. Contributions and Recommendations

This chapter presents an overview of the research in this dissertation, outlining the most significant contributions of the research (summarized in Table 7-1) and suggesting areas for further research.

Contributions

This dissertation introduced the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources (MRCMPSP-GPR/EXP) to the project scheduling literature. The MRCMPSP-GPR/EXP builds upon the classic Resource-Constrained Project Scheduling Problem (RCPSP), extending the RCPSP for multiple activity execution modes, generalized precedence with minimal and maximal time lags, and expediting resources, all within a multi-project framework. The multi-project framework for the MRCMPSP-GPR/EXP allows for generalized precedence relationships and resource constraints (both renewable and nonrenewable) at the program level, not just at the project level. A mathematical formulation of the MRCMPSP-GPR/EXP was constructed and extended from the 0-1 formulation of the RCPSP by Pritsker *et al.* (1969).

A problem generator for the MRCMPSP-GPR/EXP was developed as part of this research. The Program Attributes Generator with Expediting Resources (PAGER) gives the user extensive flexibility to define the parameters of the problem to be generated. This allows the user to craft any of the problems diagrammed in Figure 7-1 (repeated from Figure 2-1), all of which are special cases of the MRCMPSP-GPR/EXP.

The most important feature of PAGER is the method it uses to construct the underlying project network structure. PAGER uses the Restrictiveness measure proposed by Thesen (Thesen, 1977), which defines the degree to which a network is constrained by its component arcs. This measure is recognized as being far superior to other measures of network complexity, and PAGER is the first generator to directly exploit this particular measure. Use of the Thesen Restrictiveness measure gives the user of PAGER unparalleled control over the complexity of the project network. Such control is imperative in designing an experiment to evaluate any algorithm for solving a project scheduling problem since the effectiveness of an algorithm is directly impacted by the complexity of the project network (Kolisch *et al.*, 1992).

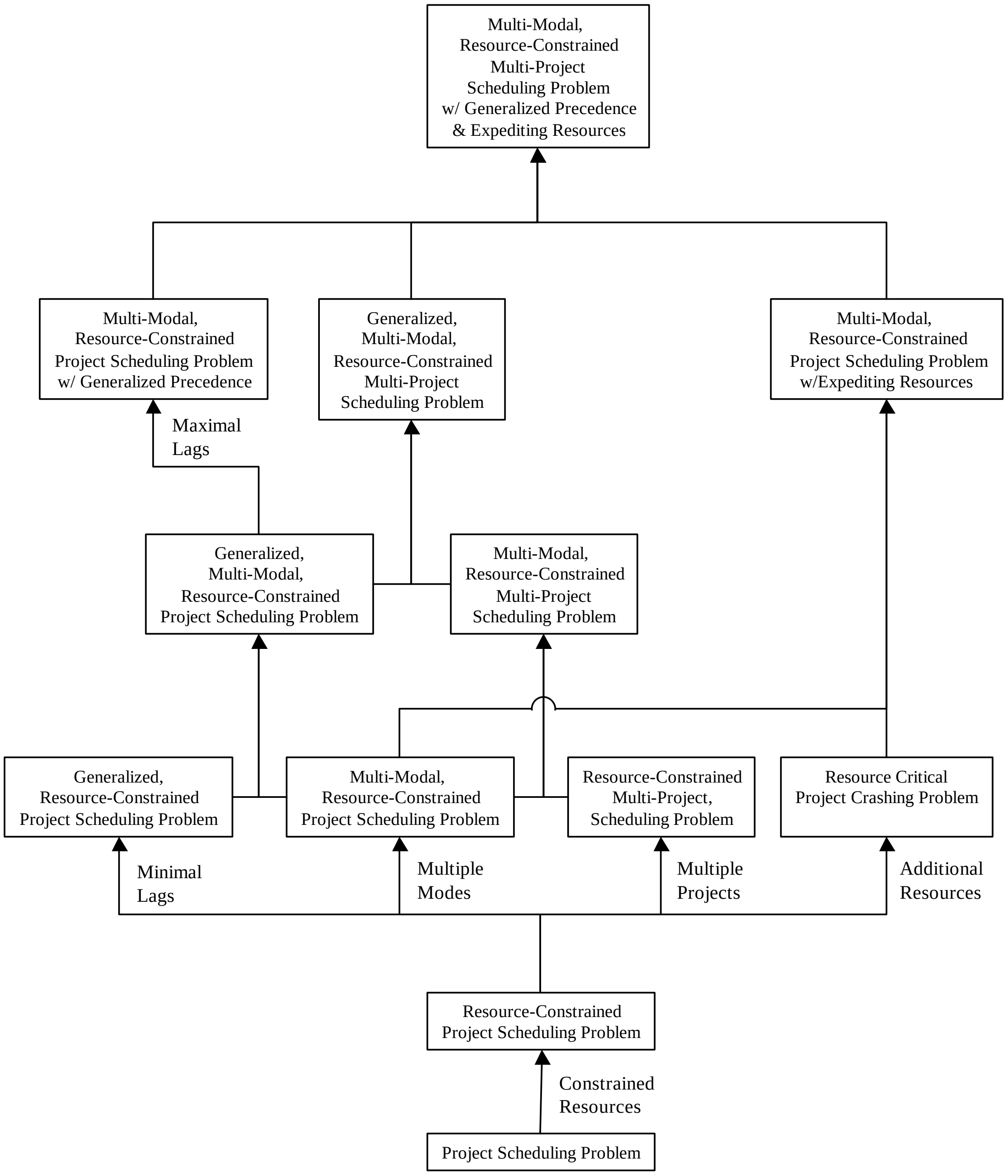

Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem w/ Generalized Precedence & Expediting Resources
Multi-Modal, Resource-Constrained Project Scheduling Problem w/ Generalized Precedence
Generalized, Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem
Multi-Modal, Resource-Constrained Project Scheduling Problem w/Expediting Resources
Maximal Lags
Generalized, Multi-Modal, Resource-Constrained Project Scheduling Problem
Multi-Modal, Resource-Constrained Multi-Project Scheduling Problem
Generalized, Resource-Constrained Project Scheduling Problem
Multi-Modal, Resource-Constrained Project Scheduling Problem
Resource-Constrained Multi-Project, Scheduling Problem
Resource Critical Project Crashing Problem
Minimal Lags
Multiple Modes
Multiple Projects
Additional Resources
Resource-Constrained Project Scheduling Problem
Constrained Resources
Project Scheduling Problem

Figure 7-1. Problem Hierarchy

The principal focus of this research was the development of two methodologies for solving the MRCMPSP-GPR/EXP, one treating any problem instance as a single project, the other exploiting the decomposability of multi-project instances of the MRCMPSP-GPR/EXP.

The first methodology for solving the MRCMPSP-GPR/EXP is a specialized, implicit enumeration algorithm based on the scheme by Talbot (1982) for the Multi-Modal RCPSP (MRCPSP). Talbot's algorithm was extended for generalized precedence with minimal and maximal time lags and for expediting resources. Since the objective of the MRCMPSP-GPR/EXP is to minimize project costs, including those for expediting resources, the objective function of the MRCMPSP-GPR/EXP is a non-regular measure of performance. The non-regularity of the objective function makes the majority of the bounding rules in the literature inapplicable for the MRCMPSP-GPR/EXP. Consequently, special bounding rules were developed and incorporated into the implicit enumeration algorithm. Testing of the algorithm with and without the new bounding rules showed a significant acceleration in the speed of the algorithm with the bounding rules. The algorithm was also demonstrated to be a significant improvement over a general 0-1 programming approach with Special Ordered Sets (SOS) of variables as implemented in IBM's Optimization Solutions Library (OSL). No other approach in the literature is capable of solving the MRCMPSP-GPR/EXP, making the specialized algorithm developed in this dissertation the first of its kind.

An additional feature built into the specialized *single-project* algorithm is the ability to generate a set of *k*-best solutions, not just a single optimal. The set of *k*-best solutions may be useful to a decision-maker who might prefer one mathematically optimal solution over another, or even a mathematically *inferior* (but close to optimal) solution, for non-mathematical reasons. The set of *k*-best solutions is also required by the decomposition approach, which is the second methodology developed for solving the MRCMPSP-GPR/EXP.

The decomposition approach for solving the MRCMPSP-GPR/EXP is based on the work by Sweeney and Murphy (1979). The approach uses Lagrangian relaxation to decompose the original problem into a number of subproblems (representing the multiple projects) and a master problem (containing the program-level constraints). A number of multipliers for relaxing the original problem were developed and tested, the most efficient of which is based on the Average Utilization Factor (AUF) described by Kurtulus and Davis (1982) and Kurtulus and Narula (1985).

Since Sweeney and Murphy (1979) do not specify how to solve the subproblems or the master problem, subproblems were solved to generate a set of *k*-best solutions using the single-project algorithm previously described. An implicit enumeration algorithm for solving the master problem was also developed as part of this research. The decomposition approach was

Table 7-1. Summary of Key Contributions

Contribution	Extension	New Feature	Theoretical
Mathematical Formulation of MRCMPSP-GPR/EXP	X		
Problem Generator for MRCMPSP-GPR/EXP	X	X	X
Directly Exploited Thesen Restrictiveness as Measure of Network Complexity			X
Made Tailorable to Vast Array of Problem Types		X	
Specialized Algorithm for Single-Project Instances of the MRCMPSP-GPR/EXP	X	X	X
Addressed Generalized Precedence and Expediting Resources			X
Developed New Set of Bounding Rules			X
Incorporated Approach for Generating Set of *k*-Best Solutions		X	
Decomposition Algorithm for Multi-Project Instances of the MRCMPSP-GPR/EXP	X		X
Built Upon Specialized Algorithm for Single-Project Instances	X		
Addressed Generalized Precedence, Renewable & Nonrenewable Resources, and Expediting Resources at the Program Level			X
Developed New Approaches for Obtaining Lagrangian Multipliers			X
Developed Scheme for Choosing Number of Solutions to Generate from Each Subproblem			X
Developed Special Acceleration Schemes			X
Incorporated Approach for Generating Set of *k*-Good Solutions	X		
Discovered Error in Sweeney-Murphy (1979) Decomposition Algorithm			X

further enhanced by three acceleration schemes. Testing showed that more problems could be solved within a fixed time limit with the decomposition approach than by solving the problems as a single project. Testing also showed that the acceleration schemes further increase the number of problems which can be solved within a fixed time limit. Finally, multiple choices of *k*, the number of best solutions generated for each subproblem, were tested to determine their impact on solution time. It was shown that, in general, a choice of 100-best solutions from each subproblem led to the most problems solved within a fixed solution time.

Table 7-1 provides a summary of key research contributions. For each contribution, Table 7-1 identifies whether the contribution is an *extension* of research presented in the literature or a *new feature* which has not been addressed in the literature, and whether or not the contribution is of a theoretical (versus applied) nature.

Recommendations

The research presented in this dissertation unfolded a number of areas for further research. They include:

1. Van Hove (1998) introduced the concept of generalized precedence with time lags dependent on the mode chosen for the related activities. Although Van Hove did this for minimal lags only, PAGER could easily be expanded to include generalized precedence with minimal *and* maximal lags based on mode selection. To expand PAGER in this way would require a straightforward re-definition of the array which describes the generalized precedences to add two additional indices, specifying the mode selected for each of the related activities. While such an expansion would not be a theoretical advancement, it would allow the flexibility necessary to generate problems of the type proposed by Van Hove.
2. Both solution algorithms developed in this dissertation (the single-project *Scheduler* and the multi-project decomposition algorithm) have been used to find optimal solutions (or sets of *k*-best solutions). Both algorithms, however, could be terminated before completion to provide a heuristic solution (or set of solutions) to a problem. As discussed in Chapter V, the single-project Scheduler often finds an optimal solution to a problem very quickly, even if it requires an extensive amount of time to verify the optimality of the solution. If, each time one of the algorithms found a solution, the solution were compared to a

theoretical lower bound (*e.g.*, the linear program relaxation), it might be possible to use the algorithms on much larger problems to find a solution within a desired tolerance of the theoretical lower bound (*e.g.*, 5-10%). One advantage of both algorithms in this dissertation is that, by their nature, they will always provide feasible solutions only. This is not the case with all heuristics (Drexl and Grunewald, 1993). Therefore, both algorithms, used as heuristics, might favorably compare to other heuristics in the literature.

3. The decomposition approach developed in this dissertation should easily lend itself well to parallelization. Since each subproblem is independent of the others, they could be solved in parallel, thereby reducing (perhaps significantly) the overall time required to solve a problem. There is, of course, some computational overhead associated with solving problems in parallel, but the time saved in solving the subproblems would most likely compensate for this overhead. This should be especially true the more subproblems (or projects) there are in the problem.
4. The Optimality Theorem presented by Sweeney and Murphy (1979: 1131) provides a sufficient condition to establish the optimality of the best solution to the decomposition master problem. Chapter VI showed, however, that the Sweeney-Murphy Optimality Theorem provides no necessary conditions. Consequently, the decomposition algorithm presented by Sweeney and Murphy may fail to terminate successfully, even if an optimal solution to the original problem has been found. Development of a necessary condition would significantly advance the functionality of the Sweeney-Murphy Decomposition approach.

Summary

Compared to many disciplines, the field of project scheduling is still in its infancy. This dissertation has advanced this growing field, introducing the MRCMPSP-GPR/EXP to the literature and contributing two methodologies for solving the MRCMPSP-GPR/EXP. This dissertation has also contributed to the more general fields of networks (in particular, the generation of networks) and integer programming (especially the decomposition of large problems). Like all research, this dissertation has also fostered new questions and areas for research. The hope of this researcher is that the body of knowledge will continue to grow and that larger and more important problems can be addressed.

APPENDIX A. Notation

Overview

This appendix provides a *Rosetta Stone* of notation used throughout this dissertation. The following sections list, respectively: (1) the notation used to describe the different types of project scheduling problems, (2) an alphabetical listings of abbreviations and acronyms, and (3) mathematical notation. Except as otherwise noted, the notation presented here is used consistently throughout this dissertation.

Problem Types

GMRCMPSP:	Generalized, Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem
GMRCPSP:	Generalized, Multi-Modal, Resource-Constrained Project Scheduling Problem
MPSP:	Multi-Project Scheduling Problem
MRCMPSP-GPR:	Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence
MRCMPSP-GPR/EXP:	Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources
MRCPSP:	Multi-Modal, Resource-Constrained Project Scheduling Problem
MRCPSP-GPR:	Multi-Modal, Resource-Constrained Project Scheduling Problem with Generalized Precedence
MRCPSP-GPR/EXP:	Multi-Modal, Resource-Constrained Project Scheduling Problem with Generalized Precedence and Expediting Resources
NPVP:	Net Present Value Problem
PSP:	Project Scheduling Problem
RCMPSP:	Resource-Constrained, Multi-Project Scheduling Problem
RCPSP:	Resource-Constrained Project Scheduling Problem

Abbreviations and Acronyms

ATO:	Air Tasking Order
AUF:	Average Utilization Factor
CI:	Complexity Index
CNC:	Coefficient of Network Complexity

CPM:	Critical Path Method
(D):	Lagrangian Dual of Problem (P)
ERS:	Expediting Resource Strength
IP:	Integer program
GCPM:	Generalized Critical Path Method
LP:	Linear program
(MP):	Sweeney-Murphy Master Problem
(MP2):	Revised Sweeney-Murphy Master Problem
(P):	LP formulation of project scheduling problem
PAGER:	**P**rogram **A**ttributes **G**enerator with **E**xpediting **R**esources
RS:	Resource Strength
RT:	Network Restrictiveness (of Thesen)
SOS:	Special Ordered Set
SP$_p$?? ?:	Sweeney-Murphy Subproblem *p*

Mathematical Notation

The notation provided in this section is listed alphabetically. However, any given letter may be represented by its Roman or Greek equivalents, its lower or upper cases, or by different formats (i.e., italics and bold). The representation of a letter denotes the type of mathematical entity it symbolizes. Using the letter "x" (and its Greek equivalent "?") as an example, the following list correlates the letter representation to the mathematical entity and lays out the ordering of notation based on its representation.

x:	Scalar (e.g., index, constant)
X:	Scalar
X:	Scalar (e.g., upper bound on index, constant)
X:	Set
x:	Vector
X:	Matrix
?:	Vector
?:	Function

Notation.

0:	(Zero) Index associated with program-level sets (i.e., the program is *Project 0*)
a_{ij}:	Binary variable: 1, if activity i directly precedes activity j; 0, otherwise
a_t:	Cost for completing the program at time t
A:	Set of network arcs (Chapter IV only)
A:	Set of all schedules
A^E:	Set of activities which are eligible for labeling and have no generalized precedence relationship (used in the GCPM)
A^L:	Set of activities which are eligible for labeling and have a generalized precedence relationship (used in the GCPM)
A^S:	Set of activities which have been labeled (used in the GCPM)
A^1:	Set of activities where each activity is a generalized predecessor every other activity in the set (used in the GCPM)
$\mathbf{A}$:	Adjacency matrix
$\mathbf{A}_{N_p}$:	Matrix of program-level generalized precedence constraint coefficients associated with project p
$\mathbf{A}_{H_p}$:	Matrix of program-level expediting resource constraint coefficients associated with project p
$\mathbf{A}_p$:	Matrix of program-level constraint coefficients associated with project p
$\mathbf{b}_p$:	Vector of right-hand side coefficients of constraint set p
$\mathbf{B}_{N_p}$:	Matrix of project-level generalized precedence constraint coefficients associated with project p
$\mathbf{B}_{H_p}$:	Matrix of project-level expediting resource constraint coefficients associated with project p
$\mathbf{B}_p$:	Matrix of constraint coefficients pertaining to project p
$c^N_{k_p}$:	Cost of nonrenewable expediting resources required by k_p
c^N_q:	Cost of an expediting unit of nonrenewable resource q
$c^R_{k_p}$:	Cost of renewable expediting resources required by k_p
c^R_{qt}:	Cost of an expediting unit of renewable resource q at time t
C_p:	Accumulated cost of the current solutions of Subproblems 1 through p
CNC_p:	Coefficient of network complexity for project p

$\mathbf{c}_p$:	Vector of costs associated with project p
d_{im_i}:	Duration of activity *i* in mode *m*
$d_p^{\min}$:	Minimum duration of activities in project *p*
$d_p^{\max}$:	Maximum duration of activities in project *p*
d_{pim}:	Duration of activity *p*(*i*) in mode *m*
D:	Program planning horizon
D_p:	Planning horizon of project *p*
$?_p$:	Due date factor of project *p* (Chapter IV only)
$?_p$:	Difference between the worst and best solutions to Subproblem *p*
$?_p^{\min}$:	Minimum due date factor of project *p* (Chapter IV only)
$?_p^{\max}$:	Maximum due date factor of project *p* (Chapter IV only)
$?$:	Difference between upper bound and lower bound of (MP)
$?_{ij}^{\boldsymbol{min}}$:	Minimal start-start lag time between activities *i* and *j*
$?_{ij}^{\boldsymbol{max}}$:	Maximal start-start lag time between activities *i* and *j*
e_{pi}:	Early start time of activity *p*(*i*)
E_p:	Early start time of project *p*
$ENC_p^{\min}$:	Minimum expediting nonrenewable resource base cost for project *p*
$ENC_p^{\max}$:	Maximum expediting nonrenewable resource base cost for project *p*
$ERC_p^{\min}$:	Minimum expediting renewable resource base cost for project *p*
$ERC_p^{\max}$:	Maximum expediting renewable resource base cost for project *p*
$ERS_{p?}^{\min}$:	Minimum expediting resource strength for resource type ? for project *p*
$ERS_{p?}^{\max}$:	Maximum expediting resource strength for resource type ? for project *p*
$?_{RF}$:	Resource factor tolerance
F:	Early program completion time
F_p:	Early completion time of project *p*
$?$:	Objective function of a scheduling problem
$?(\mathbf{S})$:	Objective function value of a particular schedule **S**

G:	Graph (Chapter IV only)
G:	Program completion due date
G_p:	Completion due date of project p
h_q^N:	Units of expediting, nonrenewable resource q used
h_{qt}^R:	Units of expediting, renewable resource q used at time t
H_{pq}^N:	Units of expediting, nonrenewable resource q remaining after projects 1 through p have been added to the program schedule
H_q^N:	Units of expediting, nonrenewable resource q available
H_{pqt}^R:	Units of expediting, renewable resource q remaining in time t after projects 1 through p have been added to the program schedule
H_{qt}^R:	Units of expediting, renewable resource q available at time t
i:	Index associated with activities / jobs (see also j). Also, index associated with levels of a search tree.
I_p:	Set of activities / jobs in project p
j:	Index associated with activities / jobs (see also i)
J:	Number of activities / jobs
J_p:	Number of activities / jobs in project p
$J_p^{\min}$:	Minimum number of activities / jobs in project p
$J_p^{\max}$:	Maximum number of activities / jobs in project p
k:	Index associated with solutions to a problem. Also, used generically as in "k-best" solutions
K_p:	Number of solutions to project p
$K_{pq}^{\min}$:	Minimum total demand for resource q in project p
$K_{pq}^{\max}$:	Maximum total demand for resource q in project p
l_{pi}:	Late start time of activity $p(i)$
L_p:	Lag coefficient of project p
$L_p^{\min}$:	Minimum lag coefficient of project p
$L_p^{\max}$:	Maximum lag coefficient of project p
$LB(\boldsymbol{\mu})$:	Lower bound on solution to Problem (P)

LF_p:	Fraction of arcs in project p which denote generalized precedence
$LL_p^{\min}$:	Lower bound on the minimum lag times for project p
$LL_p^{\max}$:	Upper bound on the minimum lag times for project p
$LU_p^{\min}$:	Lower bound on the maximum lag times for project p
$LU_p^{\max}$:	Upper bound on the maximum lag times for project p
$?_p^k$:	Zero-one variable associated with the k^{th} solution to project p
? :	Vector of variables ? representing solution to Sweeney-Murphy Master Problem
m_j:	Execution mode of activity j
M_{k_p} :	Set of mode assignments associated with solution k_p
$M_p^{\min}$:	Minimum number of modes per activity in project p
$M_p^{\max}$:	Maximum number of modes per activity in project p
M_{pi} :	Set (or number) of execution modes for activity i of project p
$MC_{p0}^{\min}$:	Minimum base mode cost
$MC_{p0}^{\max}$:	Maximum base mode cost
$MC_{p1}^{\min}$:	Minimum mode cost increment
$MC_{p1}^{\max}$:	Maximum mode cost increment
$\mathbf{m}$:	J-tuple of the execution modes of each activity j, $j = 1, \ldots, J$
$\boldsymbol{\mu}$:	Lagrangian multipliers used in Sweeney-Murphy Decomposition
n_d:	Number of disjunctive arcs in a graph
N:	Set of network nodes (Chapter IV only)
N_i:	Set of activities which have an explicit generalized precedence relationship with activity i (see Definition 4-16)
N_i^* :	Set of activities which have an implicit generalized precedence relationship with activity i (see Definition 4-17)
N_p:	Set of generalized precedence relationships in project p
O_i:	Set of activities which precede activity i
O_p:	Set of standard precedence relations within project p
p:	Index associated with projects
$p(i)$:	Activity i of project p

P:	Restrictiveness of a graph
P_p ?G ? 1?:	Probability of time-increasing activity costs for project *p*
P_p ?G ? 2?:	Probability of time-decreasing activity costs for project *p*
$\mathrm{P}_{p?}\,(F\ ?\ 1)$:	Probability of duration-constant demands for resource type ? for project *p*
$\mathrm{P}_{p?}\,(F\ ?\ 2)$:	Probability of duration-nonincreasing demands for resource type ? for project *p*
P:	Number / set of projects in a multi-project program
$P_p^{\min}$:	Maximum number of predecessors per activity for project *p*
$P_{pJ}^{\min}$:	Minimum number of finish activities in project *p*
$P_{pJ}^{\max}$:	Maximum number of finish activities in project *p*
PEN_{00} :	Program base penalty
PEN_{01} :	Program penalty increment
$PEN_{p0}^{\min}$:	Minimum project base penalty
$PEN_{p0}^{\max}$:	Maximum project base penalty
$PEN_{p1}^{\min}$:	Minimum project penalty increment
$PEN_{p1}^{\max}$:	Maximum project penalty increment
P:	Denotes a scheduling problem
p_q^N :	Price charged to a project for nonrenewable resource *q*
p_{qt}^R :	Price charged to a project for renewable resource *q* in time period *t*
? :	Number of possible permutations of a number sequence (Chapter IV only)
? :	Total cost difference between a time only and a resource feasible schedules
Q^N :	Set of all nonrenewable resources
Q_0^N :	Set of program-level nonrenewable resources
Q_p^N :	Set of nonrenewable resources unique to project *p*
Q^R :	Set of all renewable resources
Q_0^R :	Set of program-level renewable resources
Q_p^R :	Set of renewable resources unique to project *p*

$Q_{p?}^{\min}$:	Minimum number of resources of type ? requested per job in project *p*
$Q_{p?}^{\max}$:	Maximum number of resources of type ? requested per job in project *p*
r_{ij}:	Binary variable: 1, if activity *j* is reachable from activity *i*; 0, otherwise
$r_{im_iq}^{N}$:	Units of nonrenewable resource *q* required by activity *i* in mode m_i
$r_{k_p}^{N}$:	Total demand by solution k_p for nonrenewable resource *q*
r_{pimq}^{N} :	Units of nonrenewable resource *q* required by activity *p*(*i*) in mode *m*
$r_{im_iq}^{R}$:	Units of renewable resource *q* required by activity *i* in mode m_i
$r_{k_pt}^{R}$:	Total demand by solution k_p for nonrenewable resource *q* at time *t*
r_{pimq}^{R} :	Units of renewable resource *q* required by activity *p*(*i*) in mode *m*
$r_{p?}^{\min}$:	Minimum resource demand for resource type ? for project *p*
$r_{p?}^{\max}$:	Maximum resource demand for resource type ? for project *p*
RT:	Restrictiveness measure of Thesen
R_{pq}^{N}:	Units of nonrenewable resource *q* remaining after projects 1 through *p* have been added to the program schedule
R_{pqt}^{R} :	Units of renewable resource *q* remaining in time *t* after projects 1 through *p* have been added to the program schedule
R_{q}^{N} :	Units of nonrenewable resource *q* available
R_{qt}^{R}:	Units of renewable resource *q* available at time *t*
$RF_{p?}$:	Resource factor of resource type ? for project *p*
$RS_{p?}^{\min}$:	Minimum resource strength for resource type ? for project *p*
$RS_{p?}^{\max}$:	Maximum resource strength for resource type ? for project *p*
R:	Reachability matrix
$?_{p}$:	Release date of project *p*
$?_{p}^{\min}$:	Minimum release date of project *p*
$?_{p}^{\max}$:	Maximum release date of project *p*
s_j:	Start time of activity *j*

S_i:	Set of activities which succeed activity i
S_{k_p} :	Set of start time assignments associated with solution k_p
$S_p^{\max}$:	Maximum number of successors per activity for project p
$S_{p1}^{\min}$:	Minimum number of start activities in project p
$S_{p1}^{\max}$:	Maximum number of start activities in project p
$\mathbf{s}$:	J-tuple of the start time of each activity j, $j = 1, \ldots, J$
$\mathbf{S}$:	Schedule of problem **P**
UB:	Upper bound on solution to Problem (P)
tol_{CNC}:	Tolerance on coefficient of network complexity
tol_{TH}:	Tolerance on Thesen Restrictiveness
T:	Dummy terminal activity of a program
T_p:	Dummy terminal activity of project p
TH_p:	Thesen Restrictiveness measure for project p
? :	Resource type
$\lvert ? \rvert_p^{\min}$:	Minimum number of resources of type ? for project p
$\lvert ? \rvert_p^{\max}$:	Maximum number of resources of type ? for project p
w_{pi}:	$[e_{pi}, l_{pi}]$, the start time window of activity $p(i)$
W_{ij}:	Directed path from activity i to activity j
x_{pimt}:	Binary variable: 1, if activity $p(i)$ is executed in mode m and starts at time t; 0, otherwise
$x_{T_p t}$:	Binary variable: 1, if terminal activity T_p of project p starts at time t; 0, otherwise
x_{Tt} :	Binary variable: 1, program terminal activity T starts at time t; 0, otherwise
$\mathbf{x}_{H_p}$:	Integer variables associated with the expediting resources of project p
$\mathbf{x}_p$:	Vector of variables associated with project p
$\mathbf{y}_p^k$:	kth solution to project p
z:	Objective function value of a mathematical programming problem

Appendix B. PAGER Input

This appendix provides an example of a Specification File used to define the parameters required by the Program Attributes Generator with Expediting Resources (PAGER).

Problem Generator Input

```
************************************************************************
SPECIFICATIONS:
************************************************************************
GENERAL INFORMATION -
Problem Name                        : Test Program
************************************************************************
PROGRAM -
Number of Projects                  : 4
Minimum Program Due Date Factor     : 0.00
Maximum Program Due Date Factor     : 0.00
************************************************************************
PROJECTS -                           <---------FOR EACH PROJECT-------->
Minimum Number of Jobs              : 4 4 4 4 4
Maximum Number of Jobs              : 4 4 4 4 4
Minimum Project Release Dates       : 1 1 1 1 1
Maximum Project Release Dates       : 1 1 1 1 1
Minimum Project Due Date Factors    : 0.00 0.00 0.00 0.00 0.00
Maximum Project Due Date Factors    : 0.00 0.00 0.00 0.00 0.00
************************************************************************
MODES -                              <---------FOR EACH PROJECT-------->
Minimum Number of Job Modes         : 1 1 1 1 1
Maximum Number of Job Modes         : 1 1 1 1 1
Minimum Job Duration                : 1 1 1 1 1
Maximum Job Duration                : 10 10 10 10 10
************************************************************************
PROJECT NETWORKS -                   <---------FOR EACH PROJECT-------->
Minimum Number of Start Jobs        : 1 1 1 1 1
Maximum Number of Start Jobs        : 1000 1000 1000 1000 1000
Minimum Number of End Jobs          : 1 1 1 1 1
Maximum Number of End Jobs          : 1000 1000 1000 1000 1000
Maximum Successors Per Job          : 1000 1000 1000 1000 1000
Maximum Predecessors Per Job        : 1000 1000 1000 1000 1000
Min Start-Start Lag Fraction        : 0.20 0.20 0.20 0.20 0.20
Max Start-Start Lag Fraction        : 0.20 0.20 0.20 0.20 0.20
Min on Lower Bound of Lag           : -0.2 -0.2 -0.2 -0.2 -0.2
Max on Lower Bound of Lag           : 0.2 0.2 0.2 0.2 0.2
Min on Upper Bound of Lag           : 0.4 0.4 0.4 0.4 0.4
Max on Upper Bound of Lag           : 0.8 0.8 0.8 0.8 0.8
Use CNC (Arcs/Nodes) (1=Yes)        : 0
Network Complexity Tolerance        : 0.00
CNC (Arcs/Nodes)                    : 0.00 0.00 0.00 0.00 0.0
Use Thesen Restrictiveness (1=Yes)  : 1
Restrictiveness Tolerance           : 0.1
Thesen Restrictiveness              : 0.75 0.75 0.75 0.75 0.75
************************************************************************
PROGRAM NETWORK -
Min Proj Lag for Each Pair          : 0.00 0.00 0.00 0.00
Max Proj Lag for Each Pair          : 0.00 0.00 0.00 0.00
Maximum Inter-Proj Successors/Job   : 1000
Maximum Inter-Proj Predecessors/Job : 1000
Min Start-Start Lag Fraction        : 0.20
Max Start-Start Lag Fraction        : 0.20
Min on Lower Bound of Lag           : -0.2
Max on Lower Bound of Lag           : 0.2
Min on Upper Bound of Lag           : 0.4
Max on Upper Bound of Lag           : 0.8
Program-Level CNC                   : 0.00
Program-Level Restrictiveness       : 0.25
```

```
************************************************************************
RENEWABLE RESOURCES -               PROGRAM   <----FOR EACH PROJECT---->
Min Number of Renewable Resources   : 3 1 1 1 1 1
Max Number of Renewable Resources   : 3 1 1 1 1 1
Min Number of Res Requested Per Job : 0 0 0 0 0 0
Max Number of Res Requested Per Job : 10 10 10 10 10 10
Renewable Resource Factor           : 1.00 1.00 1.00 1.00 1.00 1.00
Minimum Per-Period Res Demand       : 1 1 1 1 1 1
Maximum Per-Period Res Demand       : 10 10 10 10 10 10
Minimum Renew Resource Strength     : 0.50 1.00 1.00 1.00 1.00 1.00
Maximum Renew Resource Strength     : 0.50 1.00 1.00 1.00 1.00 1.00
Min Exped Renew Resource Strength   : 0.50 0.00 0.00 0.00 0.00 0.00
Max Exped Renew Resource Strength   : 0.50 0.00 0.00 0.00 0.00 0.00
Prob of Duration Constant Demand    : 0.00 0.00 0.00 0.00 0.00 0.00
************************************************************************
NONRENEWABLE RESOURCES -            PROGRAM   <----FOR EACH PROJECT---->
Min Number of Nonrenewable Resources: 3 1 1 1 1 1
Max Number of Nonrenewable Resources: 3 1 1 1 1 1
Min Number of Res Requested Per Job : 0 0 0 0 0 0
Max Number of Res Requested Per Job : 10 10 10 10 10 10
Nonrenewable Resource Factor        : 1.00 1.00 1.00 1.00 1.00 1.00
Minimum Resource Demand             : 1 1 1 1 1 1
Maximum Resource Demand             : 10 10 10 10 10 10
Minimum Nonrenew Resource Strength  : 0.50 1.00 1.00 1.00 1.00 1.00
Maximum Nonrenew Resource Strength  : 0.50 1.00 1.00 1.00 1.00 1.00
Min Exped Nonrenew Resource Strength: 0.50 0.00 0.00 0.00 0.00 0.00
Max Exped Nonrenew Resource Strength: 0.50 0.00 0.00 0.00 0.00 0.00
Prob of Duration Constant Demand    : 0 0 0 0 0 1
************************************************************************
OBJECTIVE FUNCTION -
Completion Penalty   (1 = Include)  : 1
Mode Costs           (1 = Include)  : 1
Exped Resource Costs (1 = Include)  : 1
************************************************************************
COSTS DATA - (*/** => Value is Fraction of Program Penalty Min/Increment
Program Penalty Minimum and Incr    : 1000 1000
Project Penalty Minimum Range *     : 0.50 0.75
Project Penalty Increment Range **  : 0.40 0.50
Base Mode Cost Range *              : 0.05 0.10
Mode Cost Increment Range **        : 0.05 0.10
Prob of Time-Increasing Job Costs   : 1.00
Prob of Time-Decreasing Job Costs   : 0.00
Exped Renew Resource Cost Range *   : 0.00 0.05
Exped Nonrenew Resource Cost Range *: 0.00 0.05
************************************************************************
TOLERANCES -
Resource Factor                     : 0.1
Maximum Trials                      : 200
************************************************************************
```

Appendix C. PAGER Output

This appendix provides an example of a problem file generated by the Program Attributes Generator with Expediting Resources (PAGER). The problem statement is in PAGER format.

Problem File

```
************************************************************************
Program Name                     : Test Program
Number of Projects               : 4
************************************************************************
                              GENERAL DATA:
Proj      Release Due   Proj   MPM  Renewable Nonrenewable
 No  Jobs  Date   Date Horizon Time Resources  Resources
---- ---- ------- ---- ------- ---- --------- -----------
  0   18      1     25    54     25     3          3
  1    4      1     10    10     10     1          1
  2    4      1     10    10     10     1          1
  3    4      1     19    19     19     1          1
  4    4      1     15    15     15     1          1
************************************************************************
                    PROGRAM-AS-PROJECT CONVERSION DATA
SUCCESSORS:
    Proj Job  No    No
 No  No   No Mode Success Successors
--- ---- --- ---- ------- ----------------------------------------
  1   0    1   1      4     2   6  10  14
  2   1    1   1      1     3
  3   1    2   1      1     4
  4   1    3   1      1     5
  5   1    4   1      1    18
  6   2    1   1      1     7
  7   2    2   1      1     8
  8   2    3   1      1     9
  9   2    4   1      2    18  16
 10   3    1   1      1    11
 11   3    2   1      2    12   6
 12   3    3   1      1    13
 13   3    4   1      1    18
 14   4    1   1      1    15
 15   4    2   1      1    16
 16   4    3   1      1    17
 17   4    4   1      1    18
 18   0   18   1      0
************************************************************************
START-START LAGS:
Job Lag Lag Min Max
 No  No Job Lag Lag
--- --- --- --- ---
  0   0   0   0   0
  7   1  15   0   7
************************************************************************
MODE DATA WITH RESOURCES:
Job Mode      Resource Requirements
 No  No  Dur    R 1  R 2  R 3  R 4  R 5  R 6  R 7    N 1  N 2  N 3  N 4  N 5  N 6  N 7
--- ---- ---    ---  ---  ---  ---  ---  ---  ---    ---  ---  ---  ---  ---  ---  ---
  1   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
  2   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
  3   1    3    3   10    7    3    0    0    0      4    8    3    8    0    0    0
  4   1    7    2   10    6    4    0    0    0      3    9    1    3    0    0    0
  5   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
  6   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
  7   1    2    9    5    6    0    7    0    0      4    7    8    0    3    0    0
  8   1    8    9   10    9    0    1    0    0      8    5    3    0    2    0    0
  9   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
 10   1    0    0    0    0    0    0    0    0      0    0    0    0    0    0    0
 11   1    9    3    9    3    0    0    2    0     9    4    1    0    0    3    0
 12   1   10    7    9    5    0    0    7    0     4    6    7    0    0    5    0
 13   1    0    0    0    0    0    0    0    0     0    0    0    0    0    0    0
 14   1    0    0    0    0    0    0    0    0     0    0    0    0    0    0    0
 15   1    9    7    7    6    0    0    0    4     3    8    6    0    0    0    5
 16   1    6    8    8    3    0    0    0    2     4    2    8    0    0    0    6
 17   1    0    0    0    0    0    0    0    0     0    0    0    0    0    0    0
 18   1    0    0    0    0    0    0    0    0     0    0    0    0    0    0    0
************************************************************************
REGULAR RENEWABLE RESOURCE AVAILABILITY:
Units
R 1   R 2   R 3   R 4   R 5   R 6   R 7
----  ----  ----  ----  ----  ----  ----
  17    21    16     4     7     7     4
************************************************************************
EXPEDITING RENEWABLE RESOURCE AVAILABILITY:
Units/Cost
R 1      R 2      R 3      R 4      R 5      R 6      R 7
-------- -------- -------- -------- -------- -------- --------
   8  42   11  15    7  16    0  39    0  22    0  18    0  24
************************************************************************
REGULAR NONRENEWABLE RESOURCE AVAILABILITY:
Units
N 1   N 2   N 3   N 4   N 5   N 6   N 7
----  ----  ----  ----  ----  ----  ----
  39    49    37    11     5     8    11
************************************************************************
EXPEDITING NONRENEWABLE RESOURCE AVAILABILITY:
Units/Cost
 N 1      N 2      N 3      N 4      N 5      N 6      N 7
-------- -------- -------- -------- -------- -------- --------
  20  22   25  15   19  40    0  22    0  11    0  36    0   9
************************************************************************
COMPLETION/MODE COSTS:
Job Mode  Base  Incr Start End
 No  No   Cost  Cost Time  Time
--- ---- ------ ---- ----- -----
  1   1      0    0     1    30
  2   1      0    0     1    45
  3   1     64   72     1    45
  4   1     79   70     4    48
  5   1    691  425    10    54
  6   1      0    0     1    38
  7   1     61   78     1    39
  8   1     72   80     3    41
  9   1    583  457    10    48
 10   1      0    0     1    30
 11   1     64   54     1    30
 12   1     96   54    10    45
 13   1    580  497    19    54
 14   1      0    0     1    40
 15   1     72   78     1   40
 16   1     90   89    10    49
 17   1    654  487    15    54
 18   1   1000 1000    25    54
************************************************************************
```

Appendix D. Scheduler Output

This appendix provides an example of an output file generated by the Scheduler used to solve single-project instances of the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence andExpediting Resources (MRCMPSP-GPR/EXP).

```
************************************************************************
Program Name                      : Test Program
Number of Projects                :          1
Date                              :   03/19/01
Time                              :   08:58:30
Number of Solutions               :          1
Total Solution Time (Seconds)     :        .41
************************************************************************
Solns Discarded-Project 1         :         37
************************************************************************
Solution      1:   Objective Function Value =      26391

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   2       10
   4   3       14
   5   1       17
   6   1        1
   7   2       14
   8   1        1
   9   1       18
  10   1        1
  11   3        1
  12   3        5
  13   1       12
  14   1        1
  15   2        5
  16   2        2
  17   1       13
  18   1       18

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7  R 8  R 9  R10  R11  R12  R13  R14  R15
------  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    2     0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
    3     0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
    4     0    1    0    0    0    0    0    0    0    0    0    0    0    0    0
    5     1    0    3    0    0    0    0    0    0    0    0    0    0    0    0
    6     1    0    3    0    0    0    0    0    0    0    0    0    0    0    0
    7     1    0    3    0    0    0    0    0    0    0    0    0    0    0    0
    8     1    0    3    0    0    0    0    0    0    0    0    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7  N 8  N 9  N10  N11  N12  N13  N14  N15
---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    5    7    6    0    0    0    1    1    0
************************************************************************
```

Appendix E. Sample Decomposition Algorithm Output

This appendix provides an example of an output file generated by the decomposition algorithm used to solve multi-project instances of the Multi-Modal, Resource-Constrained, Multi-Project Scheduling Problem with Generalized Precedence and Expediting Resources.

```
**************************************************
Program Name                      : Test Program
Number of Projects                :          4
Date                              :   03/24/01
Time                              :   15:03:54
Number of Solutions               :       1000
Total Solution Time (Seconds)     :       1.06
**************************************************
Solns Discarded-Project 0         :       1921
Solns Discarded-Project 1         :         18
Solns Discarded-Project 2         :          0
Solns Discarded-Project 3         :          0
Solns Discarded-Project 4         :          0
**************************************************
Solution      1:
Objective Function Value =      19680

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       10
  13   1       19
  14   1        1
  15   1       11
  16   1       20
  17   1       25
  18   1       25

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
     0    0    0    0    0    0    0    0
    10    1    3    1    0    0    0    0
    11    6    0    1    0    0    0    0
    12    6    5    4    0    0    0    0
    13    6    5    4    0    0    0    0
    14    6    5    4    0    0    0    0
    15    6    5    4    0    0    0    0
    16    6    5    4    0    0    0    0
    17    6    5    4    0    0    0    0
    18    6    5    4    0    0    0    0
    19    6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

APPENDIX F. Best Solutions to (MP) Versus (P)

Overview

This appendix provides an example showing that the *k*-best solutions to Problem (MP) are not necessarily the *k*-best solutions to Problem (P), as described in Chapter 6. The following sections contain: (1) the PAGER input used to generate the example, (2) the resulting problem statement, and (3) key solutions to the problem when solved using the Scheduler and when solved using the decomposition approach. The Scheduler solutions are the best solutions to Problem (P), while the decomposition solutions are the best solutions to Problem (MP). Note that Solutions 1 and 2 from both approaches are identical. Solution 1000 to (MP) is greater than that to (P). The 1000-th best solution to (P) is not even contained in the set of the 1000 best solutions to (MP). In fact, the objective function values of Solutions 99 and 100 from the decomposition approach straddle the value of the 1000-th best solution to (P).

Problem Generation Input

```
************************************************************************
SPECIFICATIONS:
************************************************************************
GENERAL INFORMATION -
Problem Name                        : Test Program
************************************************************************
PROGRAM -
Number of Projects                  : 4
Minimum Program Due Date Factor     : 0.00
Maximum Program Due Date Factor     : 0.00
************************************************************************
PROJECTS -                           <---------FOR EACH PROJECT-------->
Minimum Number of Jobs              : 4 4 4 4 4
Maximum Number of Jobs              : 4 4 4 4 4
Minimum Project Release Dates       : 1 1 1 1 1
Maximum Project Release Dates       : 1 1 1 1 1
Minimum Project Due Date Factors    : 0.00 0.00 0.00 0.00 0.00
Maximum Project Due Date Factors    : 0.00 0.00 0.00 0.00 0.00
************************************************************************
MODES -                              <---------FOR EACH PROJECT-------->
Minimum Number of Job Modes         : 1 1 1 1 1
Maximum Number of Job Modes         : 1 1 1 1 1
Minimum Job Duration                : 1 1 1 1 1
Maximum Job Duration                : 10 10 10 10 10
************************************************************************
PROJECT NETWORKS -                   <---------FOR EACH PROJECT-------->
Minimum Number of Start Jobs        : 1 1 1 1 1
Maximum Number of Start Jobs        : 1000 1000 1000 1000 1000
Minimum Number of End Jobs          : 1 1 1 1 1
Maximum Number of End Jobs          : 1000 1000 1000 1000 1000
Maximum Successors Per Job          : 1000 1000 1000 1000 1000
Maximum Predecessors Per Job        : 1000 1000 1000 1000 1000
Min Start-Start Lag Fraction        : 0.20 0.20 0.20 0.20 0.20
Max Start-Start Lag Fraction        : 0.20 0.20 0.20 0.20 0.20
Min on Lower Bound of Lag           : -0.2 -0.2 -0.2 -0.2 -0.2
Max on Lower Bound of Lag           : 0.2 0.2 0.2 0.2 0.2
Min on Upper Bound of Lag           : 0.4 0.4 0.4 0.4 0.4
```

```
Max on Upper Bound of Lag           : 0.8 0.8 0.8 0.8 0.8
Use CNC (Arcs/Nodes) (1=Yes)        : 0
Network Complexity Tolerance        : 0.00
CNC (Arcs/Nodes)                    : 0.00 0.00 0.00 0.00 0.0
Use Thesen Restrictiveness (1=Yes)  : 1
Restrictiveness Tolerance           : 0.1
Thesen Restrictiveness              : 0.75 0.75 0.75 0.75 0.75
************************************************************************
PROGRAM NETWORK -
Min Proj Lag for Each Pair          : 0.00 0.00 0.00 0.00
Max Proj Lag for Each Pair          : 0.00 0.00 0.00 0.00
Maximum Inter-Proj Successors/Job   : 1000
Maximum Inter-Proj Predecessors/Job : 1000
Min Start-Start Lag Fraction        : 0.20
Max Start-Start Lag Fraction        : 0.20
Min on Lower Bound of Lag           : -0.2
Max on Lower Bound of Lag           : 0.2
Min on Upper Bound of Lag           : 0.4
Max on Upper Bound of Lag           : 0.8
Program-Level CNC                   : 0.00
Program-Level Restrictiveness       : 0.25
************************************************************************
RENEWABLE RESOURCES -               PROGRAM   <----FOR EACH PROJECT---->
Min Number of Renewable Resources   : 3 1 1 1 1 1
Max Number of Renewable Resources   : 3 1 1 1 1 1
Min Number of Res Requested Per Job : 0 0 0 0 0 0
Max Number of Res Requested Per Job : 10 10 10 10 10 10
Renewable Resource Factor           : 1.00 1.00 1.00 1.00 1.00 1.00
Minimum Per-Period Res Demand       : 1 1 1 1 1 1
Maximum Per-Period Res Demand       : 10 10 10 10 10 10
Minimum Renew Resource Strength     : 0.50 1.00 1.00 1.00 1.00 1.00
Maximum Renew Resource Strength     : 0.50 1.00 1.00 1.00 1.00 1.00
Min Exped Renew Resource Strength   : 0.50 0.00 0.00 0.00 0.00 0.00
Max Exped Renew Resource Strength   : 0.50 0.00 0.00 0.00 0.00 0.00
Prob of Duration Constant Demand    : 0.00 0.00 0.00 0.00 0.00 0.00
************************************************************************
NONRENEWABLE RESOURCES -            PROGRAM   <----FOR EACH PROJECT---->
Min Number of Nonrenewable Resources: 3 1 1 1 1 1
Max Number of Nonrenewable Resources: 3 1 1 1 1 1
Min Number of Res Requested Per Job : 0 0 0 0 0 0
Max Number of Res Requested Per Job : 10 10 10 10 10 10
Nonrenewable Resource Factor        : 1.00 1.00 1.00 1.00 1.00 1.00
Minimum Resource Demand             : 1 1 1 1 1 1
Maximum Resource Demand             : 10 10 10 10 10 10
Minimum Nonrenew Resource Strength  : 0.50 1.00 1.00 1.00 1.00 1.00
Maximum Nonrenew Resource Strength  : 0.50 1.00 1.00 1.00 1.00 1.00
Min Exped Nonrenew Resource Strength: 0.50 0.00 0.00 0.00 0.00 0.00
Max Exped Nonrenew Resource Strength: 0.50 0.00 0.00 0.00 0.00 0.00
Prob of Duration Constant Demand    : 0 0 0 0 0 1
************************************************************************
OBJECTIVE FUNCTION -
Completion Penalty   (1 = Include)  : 1
Mode Costs           (1 = Include)  : 1
Exped Resource Costs (1 = Include)  : 1
************************************************************************
COSTS DATA - (*/** => Value is Fraction of Program Penalty Min/Increment
Program Penalty Minimum and Incr    : 1000 1000
Project Penalty Minimum Range *     : 0.50 0.75
Project Penalty Increment Range **  : 0.40 0.50
Base Mode Cost Range *              : 0.05 0.10
Mode Cost Increment Range **        : 0.05 0.10
Prob of Time-Increasing Job Costs   : 1.00
Prob of Time-Decreasing Job Costs   : 0.00
Exped Renew Resource Cost Range *   : 0.00 0.05
Exped Nonrenew Resource Cost Range *: 0.00 0.05
************************************************************************
TOLERANCES -
Resource Factor                     : 0.1
Maximum Trials                      : 200
************************************************************************
```

Problem File

```
************************************************************************
Program Name                     : Test Program
Number of Projects               : 4
************************************************************************
                              GENERAL DATA:
Proj      Release Due   Proj   MPM  Renewable Nonrenewable
 No  Jobs  Date   Date Horizon Time Resources  Resources
---- ---- ------- ---- ------- ---- --------- ------------
  0   18      1     25    54     25     3          3
  1    4      1     10    10     10     1          1
  2    4      1     10    10     10     1          1
  3    4      1     19    19     19     1          1
  4    4      1     15    15     15     1          1
************************************************************************
                    PROGRAM-AS-PROJECT CONVERSION DATA
SUCCESSORS:
    Proj Job  No    No
 No  No   No Mode Success Successors
--- ---- --- ---- ------- ---------------------------------------
  1   0    1   1      4     2   6  10  14
  2   1    1   1      1     3
  3   1    2   1      1     4
  4   1    3   1      1     5
  5   1    4   1      1    18
  6   2    1   1      1     7
  7   2    2   1      1     8
  8   2    3   1      1     9
  9   2    4   1      2    18  16
 10   3    1   1      1    11
 11   3    2   1      2    12   6
 12   3    3   1      1    13
 13   3    4   1      1    18
 14   4    1   1      1    15
 15   4    2   1      1    16
 16   4    3   1      1    17
 17   4    4   1      1    18
 18   0   18   1      0
************************************************************************
START-START LAGS:
Job Lag Lag Min Max
 No  No Job Lag Lag
--- --- --- --- ---
  0   0   0   0   0
  7   1  15   0   7
************************************************************************
MODE DATA WITH RESOURCES:
Job Mode      Resource Requirements
 No  No  Dur    R 1  R 2  R 3  R 4  R 5  R 6  R 7    N 1  N 2  N 3  N 4  N 5  N 6  N 7
--- ---- ---    ---  ---  ---  ---  ---  ---  ---    ---  ---  ---  ---  ---  ---  ---
  1   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
  2   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
  3   1    3     3   10    7    3    0    0    0      4    8    3    8    0    0    0
  4   1    7     2   10    6    4    0    0    0      3    9    1    3    0    0    0
  5   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
  6   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
  7   1    2     9    5    6    0    7    0    0      4    7    8    0    3    0    0
  8   1    8     9   10    9    0    1    0    0      8    5    3    0    2    0    0
  9   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
 10   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
 11   1    9     3    9    3    0    0    2    0      9    4    1    0    0    3    0
 12   1   10     7    9    5    0    0    7    0      4    6    7    0    0    5    0
 13   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
 14   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
 15   1    9     7    7    6    0    0    0    4      3    8    6    0    0    0    5
 16   1    6     8    8    3    0    0    0    2      4    2    8    0    0    0    6
 17   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
 18   1    0     0    0    0    0    0    0    0      0    0    0    0    0    0    0
```

```
************************************************************************
REGULAR RENEWABLE RESOURCE AVAILABILITY:
Units
R 1   R 2   R 3   R 4   R 5   R 6   R 7
----  ----  ----  ----  ----  ----  ----
  17    21    16     4     7     7     4
************************************************************************
EXPEDITING RENEWABLE RESOURCE AVAILABILITY:
Units/Cost
R 1      R 2      R 3      R 4      R 5      R 6      R 7
-------- -------- -------- -------- -------- -------- --------
   8  42   11  15    7  16    0  39    0  22    0  18    0  24
************************************************************************
REGULAR NONRENEWABLE RESOURCE AVAILABILITY:
Units
N 1   N 2   N 3   N 4   N 5   N 6   N 7
----  ----  ----  ----  ----  ----  ----
  39    49    37    11     5     8    11
************************************************************************
EXPEDITING NONRENEWABLE RESOURCE AVAILABILITY:
Units/Cost
 N 1      N 2      N 3      N 4      N 5      N 6      N 7
-------- -------- -------- -------- -------- -------- --------
  20  22   25  15   19  40    0  22    0  11    0  36    0   9
************************************************************************
COMPLETION/MODE COSTS:
Job Mode  Base  Incr Start End
 No  No   Cost  Cost Time  Time
--- ---- ------ ---- ----- -----
  1   1       0    0     1    30
  2   1       0    0     1    45
  3   1      64   72     1    45
  4   1      79   70     4    48
  5   1     691  425    10    54
  6   1       0    0     1    38
  7   1      61   78     1    39
  8   1      72   80     3    41
  9   1     583  457    10    48
 10   1       0    0     1    30
 11   1      64   54     1    30
 12   1      96   54    10    45
 13   1     580  497    19    54
 14   1       0    0     1    40
 15   1      72   78     1    40
 16   1      90   89    10    49
 17   1     654  487    15    54
 18   1    1000 1000    25    54
************************************************************************
```

Key Solutions (1, 2, 99, 100, 1000)

Solutions to (P) – From the Scheduler

```
*************************************************
Program Name                     : Test Program
Number of Projects               :          4
Date                             :   03/24/01
Time                             :   15:03:52
Number of Solutions             :       1000
Total Solution Time (Seconds)    :       1.10
*************************************************
Solns Discarded-Project 1        :       5328




*************************************************
Solution      1:
Objective Function Value =      19680

 Job Mode Start Time
---- ---- ----------
   1   1       1
   2   1       1
   3   1       1
   4   1       4
   5   1      10
   6   1       9
   7   1      10
   8   1      12
   9   1      19
  10   1       1
  11   1       1
  12   1      10
  13   1      19
  14   1       1
  15   1      11
  16   1      20
  17   1      25
  18   1      25

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0    0    0    0    0    0    0    0
   10    1    3    1    0    0    0    0
   11    6    0    1    0    0    0    0
   12    6    5    4    0    0    0    0
   13    6    5    4    0    0    0    0
   14    6    5    4    0    0    0    0
   15    6    5    4    0    0    0    0
   16    6    5    4    0    0    0    0
   17    6    5    4    0    0    0    0
   18    6    5    4    0    0    0    0
   19    6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

Solutions to (MP) – From Decomposition

```
*************************************************
Program Name                     : Test Program
Number of Projects               :          4
Date                             :   03/24/01
Time                             :   15:03:54
Number of Solutions             :       1000
Total Solution Time (Seconds)    :       1.06
*************************************************
Solns Discarded-Project 0        :       1921
Solns Discarded-Project 1        :         18
Solns Discarded-Project 2        :          0
Solns Discarded-Project 3        :          0
Solns Discarded-Project 4        :          0
*************************************************
Solution      1:
Objective Function Value =      19680

 Job Mode Start Time
---- ---- ----------
   1   1       1
   2   1       1
   3   1       1
   4   1       4
   5   1      10
   6   1       9
   7   1      10
   8   1      12
   9   1      19
  10   1       1
  11   1       1
  12   1      10
  13   1      19
  14   1       1
  15   1      11
  16   1      20
  17   1      25
  18   1      25

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0    0    0    0    0    0    0    0
   10    1    3    1    0    0    0    0
   11    6    0    1    0    0    0    0
   12    6    5    4    0    0    0    0
   13    6    5    4    0    0    0    0
   14    6    5    4    0    0    0    0
   15    6    5    4    0    0    0    0
   16    6    5    4    0    0    0    0
   17    6    5    4    0    0    0    0
   18    6    5    4    0    0    0    0
   19    6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
**************************************************
Solution      2:
Objective Function Value =      19706

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       10
  13   1       19
  14   1        1
  15   1       10
  16   1       20
  17   1       25
  18   1       25

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
     0    0    0    0    0    0    0    0
    10    8   10    7    0    0    0    0
    11    6    0    1    0    0    0    0
    12    6    5    4    0    0    0    0
    13    6    5    4    0    0    0    0
    14    6    5    4    0    0    0    0
    15    6    5    4    0    0    0    0
    16    6    5    4    0    0    0    0
    17    6    5    4    0    0    0    0
    18    6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
**************************************************
Solution      2:
Objective Function Value =      19706

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       10
  13   1       19
  14   1        1
  15   1       10
  16   1       20
  17   1       25
  18   1       25

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
     0    0    0    0    0    0    0    0
    10    8   10    7    0    0    0    0
    11    6    0    1    0    0    0    0
    12    6    5    4    0    0    0    0
    13    6    5    4    0    0    0    0
    14    6    5    4    0    0    0    0
    15    6    5    4    0    0    0    0
    16    6    5    4    0    0    0    0
    17    6    5    4    0    0    0    0
    18    6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
************************************************
Solution     99:
Objective Function Value =      22191

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        5
   5   1       11
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       12
  13   1       21
  14   1        1
  15   1       10
  16   1       21
  17   1       26
  18   1       26

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   10     1    1    2    0    0    0    0
   11     1    1    2    0    0    0    0
   12     6    5    4    0    0    0    0
   13     6    5    4    0    0    0    0
   14     6    5    4    0    0    0    0
   15     6    5    4    0    0    0    0
   16     6    5    4    0    0    0    0
   17     6    5    4    0    0    0    0
   18     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
************************************************
Solution     99:
Objective Function Value =      24704

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       18
  13   1       27
  14   1        1
  15   1       12
  16   1       22
  17   1       27
  18   1       27

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   18     6    5    4    0    0    0    0
   19     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
************************************************
Solution    100:
Objective Function Value =      22196

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       11
   8   1       13
   9   1       20
  10   1        1
  11   1        1
  12   1       11
  13   1       20
  14   1        1
  15   1       11
  16   1       21
  17   1       26
  18   1       26

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   11     6    0    1    0    0    0    0
   12     6    0    1    0    0    0    0
   13     6    5    4    0    0    0   0
   14     6    5    4    0    0    0    0
   15     6    5    4    0    0    0    0
   16     6    5    4    0    0    0    0
   17     6    5    4    0    0    0    0
   18     6    5    4    0    0    0    0
   19     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
************************************************
Solution    100:
Objective Function Value =      247 69

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        4
   5   1       10
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       11
  13   1       20
  14   1        1
  15   1       12
  16   1       23
  17   1       28
  18   1       28

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   12     6    5    4    0    0    0    0
   13     6    5    4    0    0    0    0
   14     6    5    4    0    0    0    0
   15     6    5    4    0    0    0    0
   16     6    5    4    0    0    0    0
   17     6    5    4    0    0    0    0
   18     6    5    4    0    0    0    0
   19     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
```

```
*************************************************
Solution   1000:
Objective Function Value =      24752

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        4
   4   1        8
   5   1       14
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       17
  13   1       26
  14   1        1
  15   1       10
  16   1       20
  17   1       25
  18   1       26

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   10     1    1    2    0    0    0    0
   11     1    1    2    0    0    0    0
   12     1    6    5    0    0    0    0
   13     1    6    5    0    0    0    0
   14     1    6    5    0    0    0    0
   17     6    5    4    0    0    0    0
   18     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
*************************************************
```

```
*************************************************
Solution   1000:
Objective Function Value =      32760

 Job Mode Start Time
---- ---- ----------
   1   1        1
   2   1        1
   3   1        1
   4   1        5
   5   1       11
   6   1        9
   7   1       10
   8   1       12
   9   1       19
  10   1        1
  11   1        1
  12   1       17
  13   1       26
  14   1        1
  15   1       13
  16   1       26
  17   1       31
  18   1       31

Expediting Renewable Resource Usage:
 Time   Units
Period  R 1  R 2  R 3  R 4  R 5  R 6  R 7
------  ---  ---  ---  ---  ---  ---  ---
    0     0    0    0    0    0    0    0
   17     6    5    4    0    0    0    0
   18     6    5    4    0    0    0    0
   19     6    5    4    0    0    0    0

Expediting Nonrenewable Resource Usage:
Units
N 1  N 2  N 3  N 4  N 5  N 6  N 7
---  ---  ---  ---  ---  ---  ---
  0    0    0    0    0    0    0
*************************************************
```

BIBLIOGRAPHY

Agrawal, M.K., S.E. Elmaghraby, and W.S. Herroelen. "DAGEN: A Generator of Testsets for Project Activity Nets," European Journal of Operational Research, 90: 376-382 (1996).

Alvares-Valdes, R., and J.M. Tamarit. "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis," in Advances in Project Scheduling. (R. Slowinski and J. Weglarz (eds.)), Elsevier Science Publishers, Amsterdam, pp.113-134, 1989.

Ahn, T. and S.S. Erenguc. "The Resource-Constrained Project Scheduling Problem with Multiple Crashable Modes: A Heuristic Procedure," European Journal of Operational Research, 107: 250-259 (1998).

Balas, Egon. "Project Scheduling with Resource Constraints," in Applications of Mathematical Programming Techniques. New York: American Elsevier Publishing Company, Inc., 1970.

Baumol, William J., and Tibor Fabian. "Decomposition, Pricing for Decentralized and External Economies," Management Science, 11: 1-32 (September 1964).

Beale, E.M.L., and J.A. Tomlin. "Special Facilities in a General Mathematical Programming System for Non-Convex Problems Using Ordered Sets of Variables," in J. Lawrence (ed.), Proceedings of the 5th International Conference on Operations Research, Tavistok, London.

Bean, James C. "A Lagrangian Algorithm for the Multiple Choice Integer Program," Operations Research, 32: 1185-1193 (September-October 1984).

Bein, W.W., J. Kamburowski, and M.F.M. Stallmann. "Optimal Reduction of Two-Terminal Directed Graphs," SIAM Journal on Computing, 221: 1112-1129 (1992).

Berczi, Andrew. "The Scheduling of Workpackage Networks Through Goal Programming," XXVII International Conference of the Institute of Management Sciences (TIMS), July 1986.

Blazewicz, J., J.K. Lenstra, and A.H.G. Rinnooy Kan. "Scheduling Projects to Resource Constraints: Classification and Complexity," Discrete Applied Mathematics, 5: 11-24 (1983).

Bowman, Edward H. "The Schedule-Sequencing Problem," Operations Research, 7: 621-624 (1959).

Chalmet, Luc G., and Ludo F. Gelders. "Lagrangean Relaxations for Solving a Warehousing Model," Paper presented at the ORSA-TIMS Joint National Meeting, November 1976.

Christofides, Nicos, R. Alvarez-Valdes, and J.M. Tamarit. "Project Scheduling with Resource Constraints: A Branch and Bound Approach," European Journal of Operational Research, 29: 262-273 (1987).

Cooper, D.F. "Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation," Management Science, 22: 1186-1194 (July 1976).

Dantzig, George B., and Philip Wolfe. “Decomposition Principle for Linear Programs,” Operations Research, 8: 101-111 (January-February 1960).

Davis, Edward W., and George E. Heidorn. “An Algorithm for Optimal Project Scheduling Under Multiple Resource Constraints,” Management Science, 17: B803-B816 (August 1971).

De Reyck, B. “On the Use of the Restrictiveness as a Measure of Complexity for Resource Constrained Project Scheduling", Onderzoeksrapport Nr. 9535, Department of Applied Economics, Katholieke Universiteit Leuven (1995).

De Reyck, B., and W. Herroelen. “On the Use of the Complexity Index as a Measure of Complexity in Activity Networks,” European Journal of Operational Research, 91: 347-366 (1996).

----. “A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations,” European Journal of Operational Research, 111: 152-174 (1998a).

----. “An Optimal Procedure for the Resource-Constrained Project Scheduling Problem with Discounted Cash Flows and Generalized Precedence Relations,” Computers and Operations Research, 25: 1-17 (1998b).

----. “The Multi-Mode Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations,” European Journal of Operational Research, 119: 538-556 (1999).

Deckro, R.F., and J.E. Hebert. “Resource Constrained Project Crashing,” Omega International Journal of Management Science, 17: 69-79 (1989).

Deckro, R.F., J.E. Hebert, and W.A. Verdini. “Project Scheduling with Work Packages,” Omega International Journal of Management Science, 20: 169-182 (1992).

Deckro, Richard F., E.P. Winkofsky, John E. Hebert, and Roger Gagnon. “A Decomposition Approach to Multi-Project Scheduling,” European Journal of Operational Research, 51: 110-118 (1991).

Deckro, Richard F., Michael L. Fredley, John C. Van Hove, and Victor D. Wiley. “Resource Planning and Coordination in Multi-Project Programs.” Address to INFORMS Conference. Montreal, Canada. September 1998.

Demeulemeester, E., B. Dodin, and W. Herroelen. “A Random Activity Network Generator,” Operations Research, 41: 972-980 (September-October 1993).

Demeulemeester, Erik, Willy Herroelen, Wendell P. Simpson, Sami Baroum, James H. Patterson, and Kum-Khiong Yang. “On a Paper by Christofides et al. For Solving the Multiple-Resource Constrained, Single Project Scheduling Problem,” European Journal of Operational Research, 76: 218-228 (1994).

Demeulemeester, Erik, and Willy Herroelen. “A Branch-and-Bounding Procedure for the Generalized Resource-Constrained Project Scheduling Problem,” Operations Research, 45: 201-212 (March-April 1997).

----. "A Branch-and-Bounding Procedure for the Multiple Resource-Constrained Project Scheduling Problem," Management Science, 38: 1803-1818 (December 1992).

Demeulemeester, Erik. "Minimizing Resource Availability Costs in Time-Limited Project Networks," Management Science, 41: 1590-1598 (October 1995).

Doersch, Robert H., and James H. Patterson. "Scheduling a Project to maximize its Present Value: A Zero-One Programming Approach," Management Science, 23: 882-889 (April 1977).

Drexl, A., and J. Grunewald. "Nonpreemptive Multi-Mode Resource-Constrained Project Scheduling," IIE Transactions, 25: 74-81 (1993).

Drexl, A., R. Nissen, J.H. Patterson, and F. Salewski. ProGen/? x -- An Instance Generator for Resource-Constrained Project Scheduling Problems with Parially Renewable Resources and Further Extensions. Technical Report, Institut fur Betriebswirtschaftslehre, Universitat Keil, 1997.

Elmaghraby, Salah E. Activity Networks: Project Planning and Control by Network Models. New York: Wiley-Interscience Publication, 1977.

Elmaghraby, Salah E., and Willy S. Herroelen. "On the Measurement of Complexity in Activity Networks," European Journal of Operations Research, 5: 223-234 (1980).

Ferreira, J. Antunes, L. Valadares Tavares, and J. Silva Coelho. "A General Generator of Project Networks in Termos of Their Morphological Features," Proceedings of the Sixth International Workshop on Project Management and Scheduling. Istanbul, Turkey, July 1998.

Fisher, Marshall L. "The Lagrangean Relaxation Method for Solving Integer Programming Problems," Management Science, 27: 1-18 (January 1981).

Ford, L.R., Jr., and D.R. Fulkerson. Flows in Networks. Princeton, NJ: Princeton University Press, 1962.

Geoffrion, A.M. "Lagrangean Relaxation for Integer Programming," Mathematical Programming Study, 2: 82-114 (1974).

Gorenstein, Samuel. "An Algorithm for Project (Job) Sequencing with Resource Constraints," Operations Research, 20: 835-850 (1972).

Hartmann, Sonke, and Andreas Drexl. "Project Scheduling with Multiple Modes: A Comparison of Exact Algorithms," Networks, 32: 283-297 (1998).

Herroelen, Willy, Bert De Reyck, and Erik Demeulemeester. "Resource-Constrained Project Scheduling: A Survey of Recent Developments," Computers & Operations Research, 25: 279-302 (1998).

Icmeli, Oya. "Project Scheduling Problems: A Survey," International Journal of Operations & Production Management, 13: 80-91 (1993).

Icmeli, O. and S.S.Erenguc. "A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Discounted Cash Flows," Management Science, 42: 1395-1408 (1996).

Icmeli, Oya, and Walter Rom. "Solving the Resource Constrained Project Scheduling Problem with Optimization Subroutine Library," Computer in Operations Research, 23: 801-817 (1996).

Kamburowski, J. "On the Minimum Cost Project Schedule," Omega International Journal of Management Science, 23: 463-465 (1995).

Kelley, James E., Jr. "Critical-Path Planning and Scheduling: Mathematical Basis," Operations Research, 9: 296-320 (1961).

Kolisch, Rainer, and Arno Sprecher. PSPLIB – A Project Scheduling Problem Library. Manuskripte aus den Instituten Fur Betriebswirtschaftslehre der Universitat Kiel Nr. 396, Christian-Albrechts-Universitat zu Kiel, March 1996.

Kolisch, Rainer, and Thomas Frase. "Minimizing Resource Costs When Meeting Tight Deadlines in a Project Environment," Abstracts of the Fifth International Workshop on Project Management and Scheduling: 139-142, Poznan, Poland (April 1996).

Kolisch, R., and R. Padman. An Integrated Survey of Project Scheduling. Manuskripte aus den Instituten Fur Betriebswirtschaftslehre der Universitat Kiel Nr. 463, Christian-Albrechts-Universitat zu Kiel, July 1998.

Kolisch, Rainer, Arno Sprecher, and Andreas Drexl. "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems," Management Science, 41: 1693-1703 (October 1995).

----. Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems. Manuskripte aus den Instituten Fur Betriebswirtschaftslehre Nr. 301, Christian-Albrechts-Universitat zu Kiel, December 1992.

Kolisch, Rainer. Project Scheduling Under Resource Constraints: Efficient Heuristics for Several Problem Classes. Heidelberg, Germany: Physica-Verlag, 1995.

Kurtulus, I., and E.W. Davis. "Multi-Project Scheduling: Catagorization of Heuristic Rules Performance," Management Science, 28: 161-172 (February 1982).

Kurtulus, Ibrahim S., and Subhash C. Narula. "Multi-Project Scheduling: Analysis of Project Performance," IIE Transactions, 17: 58-66 (1985).

Kuyumcu, Ahmet, and Alberto Garcia-Diaz. "A Decomposition Approach to Project Compression with Concave Activity Cost Functions," IIE Transactions, 26: 63-73 (November 1994).

Lasdon, Leon S. Optimization Theory for Large Systems. New York: Macmillan Publishing Co., Inc., 1970.

Moder, J. J., C.R. Phillips, and E.W. Davis. Project Management with CPM, PERT, and Precedence Diagramming (Third Edition). New York: Van Nostrand Reinhold Company, 1983.

Nauss, Robert M. Parametric Integer Programming. Columbia: University of Missouri Press, 1979.

Patterson, James H. "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem," Management Science, 30: 854-867 (July 1984).

Patterson, J.H., R. Slowinski, F.B. Talbot, and J. Weglarz. "An Algorithm for a General Class of Precedence and Resource Constrained Scheduling Problems," Advances in Project Scheduling. (R. Slowinski and J. Weglarz (eds.)), Elsevier Science Publishers, Amsterdam, pp. 3-28, 1989.

----. "Computational Experience with a Backtracking Algorithm for Solving a General Class of Precedence and Resource-Constrained -Scheduling Problems," European Journal of Operational Research, 49: 68-79 (1990).

Patterson, James H., and Walter D. Huber. "A Horizon-Varying, Zero-One Approach to Project Scheduling," Management Science, 20: 990-998 (February 1974).

Patterson, James H., and Glenn W. Roth. "Scheduling a Project Under Multiple Resource Constraints: A Zero-One Programming Approach," AIIE Transations, 8: 449-455 (1976).

Phillips, Steve, Jr., and Mohamed I. Dessouky. "Solving the Project Time/Cost Tradeoff Problem Using the Minimal Cut Concept," Management Science, 24: 393-400 (December 1977).

Pritsker, A. Alan, Lawrence J. Watters, and Philip M. Wolfe. "Multiproject Scheduling with Limited Resources: A Zero-one Programming Approach," Management Science: 16: 93-108 (September 1969).

Salewski, Frank, Andreas Schirmer, and Andreas Drexl. "Project Scheduling Under Resource and Mode Identity Constraints: Model, Complexity, Methods, and Application," European Journal of Operations Research, 102: 88-110 (1997).

----. Project Scheduling Under Resource and Mode Identity Constraints. Part I: Model, Complexity Status, and Methods. Unpublished Manuscript. Kiel, Germany, January 1996a.

----. Project Scheduling Under Resource and Mode Identity Constraints. Part II: An Application to Audit-Staff Scheduling. Unpublished Manuscript. Kiel, Germany, January 1996b.

Schwindt, C. ProGen/max: A New Problem Generator for Different Resource- Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report 449, Institut fur Wirtschaftstheorie und Operations Research, Universitat Karlsruhe, July 1995.

----. Generation of Resource-Constrained Project Scheduling Problems with Minimal and Maximal Time Lags. Technical Report 489, Institut fur Wirtschaftstheorie und Operations Research, Universitat Karlsruhe, November 1996.

Shtub, Avraham, Jonathan F. Bard, and Shlomo Globerson. Project Management: Engineering, Technology, and Implementation. Englewood Cliffs, NJ: Prentice Hall, 1994.

Speranza, M. Grazia, and Carlo Vercellis. "Hierarchical Models for Multi-Project Planning and Scheduling," European Journal of Operational Research, 64: 312-325 (1993).

Sprecher, Arno. Resource-Constrained Project Scheduling: Exact Methods for the Multi-Mode Case. Berlin: Springer-Verlag, 1994.

Sprecher, Arno, and Andreas Drexl. "Multi-Mode, Resource-Constrained Project Scheduling by a Simple, General, and Powerful Sequencing Algorithm," European Journal of Operational Research, 107: 431-450 (1998).

----. Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General, and Powerful Sequencing Algorithm. Part I: Theory. Unpublished Manuscript. Kiel, Germany, January 1996a.

----. Solving Multi-Mode Resource-Constrained Project Scheduling Problems by a Simple, General, and Powerful Sequencing Algorithm. Part II: Computation. Unpublished Manuscript. Kiel, Germany, January 1996b.

Sprecher, Arno, Rainer Kolisch, and Andreas Drexl. "Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem," European Journal of Operational Research, 80: 94-102 (1995).

Sprecher, Arno, Sonke Hartmann, and Andreas Drexl. "An Exact Algorithm for Project Scheduling with Multiple Modes," OR Spektrum, 19: 195-203 (1997).

Stinson, Joel P., Edward W. Davis, and Basheer M. Khumawala. "Multiple Resource-Constrained Scheduling Using Branch-and-Bound," AIIE Transactions, 10: 252-259 (September 1978).

Sweeney, Dennis J., and Richard A. Murphy. "A Method of Decomposition for Integer Programs," Operations Research, 27: 1128-1141 (1979).

----. "Branch and Bound Methods for Multi-Item Scheduling," Operations Research, 29: 853-864 (September-October 1981).

Talbot, F. Brian, and James H. Patterson. "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource-Constrained Scheduling Problems," Management Science, 24: 1163-1174 (July 1978).

Talbot, F. Brian. "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case," Management Science, 28: 1197-1210 (October 1982).

Thesen, A. "Measures of the Restrictiveness of Project Networks, " Networks, 7: 193-208 (1977).

Tripathy, Arabinda. "School Timetabling – A Case in Large Binary Integer Linear Programming," Management Science,30: 1473-1489 (December 1984).

Valdes, Jacobo, Robert E. Tarjan, and Eugene L. Lawler. "The Recognition of Series Parallel Digraphs," SIAM Journal on Computing, 11: 298-313 (May 1982).

Van Hove, John C. An Integer Program Decomposition Approach to Combat Planning. PhD dissertation, Air Force Institute of Technology, Wright-Patterson AFB OH 45433, July 1998.

Vercellis, Carlo. "Constrained Multi-Project Planning Problems: A Lagrangean Decomposition Approach," European Journal of Operational Research, 78: 267-275 (1994).

Wheeler, Bob. "The Mother of All Random Number Generators/Marsaglia." Electronic Mail. 19:32:08EDT, 28 Oct 1994.

Wiley, Victor D. Optimization Analysis for Design and Planning of Multi-Project Programs. MS thesis, AFIT/GOR/ENS/96M-18. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1996.

Wiley, Victor D., Richard F. Deckro, and Jack A. Jackson, Jr. "Optimization Analysis for Design and Planning of Multi-Project Programs," European Journal of Operational Research, 107: 492-506 (1998).

Williams, H.P. Model Building in Mathematical Programming, 2d Ed. Chichester: John Wiley and Sons, 1985.

Wu, Y., and C. Li. "Minimal Cost Project Networks: the Cut Set Parallel Difference Method," Omega International Journal of Management Science, 22: 401-407 (1994).

Zamani, M. Reza. "A High Performance Near-Exact Algorithm for the Resource-Constrained Project Scheduling Problem," European Journal of Operational Research (Under Review).

Vita

Major Michael L. Fredley was born in Hammond, Indiana. He graduated from Mason City High School in Mason City, Iowa, in May 1982. He pursued his undergraduate studies at the University of Utah where he graduated cum laude with a Bachelor of Science degree in Mathematics in June 1989. He was commissioned through AFROTC Detachment 850 at the University of Utah where he was recognized as a Distinguished Graduate and nominated for a Regular Commission.

Major Fredley's first Air Force assignment was to the 49th Test Squadron, Barksdale AFB, Louisiana, where he served as a Weapons Test Analyst for the B-52 and B-1B weapon systems. In August 1993, he entered the Graduate School of Engineering, Air Force Institute of Technology. He graduated in May 1995 with a Masters of Science degree in Operations Research and was designated as a Distinguished Graduate. He then entered the Ph.D. program in Operations Research where he reached candidacy before being reassigned to the Air Force Studies and Analyses Agency at the Pentagon in October 1998. Major Fredley is currently assigned to Yongsan Garrison, Seoul, Korea, where he serves as the Air Analyst to Combined Forces Command and United States Forces – Korea.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 25-09-2001	**2. REPORT TYPE** Doctoral Dissertation	**3. DATES COVERED** *(From – To)* Jun 1997 – Sep 2001

4. TITLE AND SUBTITLE A DECOMPOSITION APPROACH FOR THE MULTI-MODAL, RESOURCE-CONSTRAINED, MULTI-PROJECT SCHEDULING PROBLEM WITH GENERALIZED PRECEDENCE AND EXPEDITING RESOURCES	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Michael L. Fredley, Major, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology, Graduate School of Engineering and Management (AFIT/EN), 2950 P Street, Building 640, WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/DS/ENS/01-02
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Office of Scientific Research, Attn: Major Juan R. Vasquez, 801 N. Randolph St., Room 933, Arlington, VA 22203-1977; Comm: (703) 696-8431, e-mail: juan.vasquez@afosr.af.mil	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFOSR
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The field of project scheduling has received a great deal of study for many years with a steady evolution of problem complexity and solution methodologies. As solution methodologies and technologies improve, increasingly complex, real-world problems are addressed, presenting researchers a continuing challenge to find ever more effective means for approaching project scheduling. This dissertation introduces a project scheduling problem which is applicable across a broad spectrum of real-world situations. The problem is based on the well-known Resource-Constrained Project Scheduling Problem, extended to include multiple modes, generalized precedence, and expediting resources. The problem is further extended to include multiple projects which have generalized precedence, renewable and nonrenewable resources, and expediting resources at the program level.

The problem presented is one not previously addressed in the literature nor is it one to which the existing specialized project scheduling methodologies can be directly applied. This dissertation presents a decomposition approach for solving the problem, including algorithms for solving the decomposed subproblems and the master problem. This dissertation also describes a methodology for generating instances of the new problem, extending the way existing problem generators describe and construct network structures and this class of problem. The methodologies presented are demonstrated through extensive empirical testing.

15. SUBJECT TERMS

CPM, Critical Path Method, Decomposition, Network, Network Generator, Program, Program Management, Project, Project Generator, Project Management, Project Scheduling, Scheduling, Sweeney-Murphy Decomposition

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Richard F. Deckro, DBA (ENS)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	270	(937) 255-6565, ext 4325, e-mail: Richard.Deckro@afit.edu