OPTIMIZED SCHEDULING FOR REPETITIVE CONSTRUCTION PROJECTS

Khaled A. El-Rayes

A Thesis

in

The School for Building

Faculty of Engineering and Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

1997

0-612-40315-7

ABSTRACT

OPTIMIZED SCHEDULING FOR REPETITIVE CONSTRUCTION PROJECTS

Khaled A. El-Rayes, Ph.D.
Concordia University, 1997

An object-oriented model is presented for optimized scheduling of repetitive construction projects such as: high-rise buildings, housing projects, highways, pipeline networks, bridges and tunnels. The model provides a number of practical features, and incorporates newly developed algorithms for scheduling of repetitive construction projects including: 1) a resource-driven scheduling algorithm for repetitive activities; 2) an interruption algorithm; and 3) an optimization procedure. The scheduling algorithm identifies the scheduled start and finish times as well as the assigned crew for each unit of a repetitive activity. The algorithm provides a schedule that complies with precedence relationships, crew availability and crew work continuity constraints, and considers the impact of a number of practical factors commonly encountered during scheduling. The interruption algorithm generates feasible interruption vectors for each repetitive activity in the project and provides added advantage over available formulations that consider arbitrary user-specified interruption vectors. The optimization procedure is based on a dynamic programming formulation. Unlike available dynamic programming formulations, the present formulation is capable of incorporating cost in the optimization process, thus offering valuable support to project team members in minimizing the overall cost of the project.

For each repetitive activity in the project, the present model assists the planner in selecting the optimum crew formation and interruption vector from a set of possible alternatives. As such, the model can be used to evaluate the impact of different project acceleration strategies (i.e. multiple crews, increased crew size, overtime policies, or additional shifts) on the overall cost. The present model is implemented as a prototype software system. The system is developed as a 32-bit windows application that supports user-friendly interface including menus, dialog boxes, and windows. A number of application examples are analyzed to illustrate the use of the model and demonstrate its capabilities. The model can be used as a decision support system for generating optimized schedules for repetitive construction projects. This should contribute to cost-effective delivery of constructed projects.

ACKNOWLEDGEMENTS

I wish to express my deepest gratitude and sincere appreciation to my supervisor Dr. O. Moselhi, for his solid support, valuable advice, and constructive guidance throughout all stages of this study. His effort and suggestions to improve the contents of this thesis are greatly appreciated.

My thanks to the faculty, staff, and colleagues at the Centre for Building Studies who helped in any way to carry out this thesis. I would like especially to thank my colleagues : Diana Abdo, Rola Majed, Ines Sequira, Dr. Tarek Hegazy, Dr. Adel Abdou, Dr. Ismail Budawi, Ahmed Mokhtar, Moataz El-Karmalawy, Hassan Zibara, Bassem AbuShakra, and Mohamed Al-Hussein for their constructive criticism and helpful advice.

Finally, very special thanks to my family, my father, mother, and sister, for their unlimited encouragement and support during the course of this study.

The financial support for this work provided by the Fonds pour la Formation de Chercheurs et l'Aide a la Recherche under the FCAR-Bourse, and the Natural Sciences and Engineering Research Council of Canada under grant no. A4430 is gratefully acknowledged.

TABLE OF CONTENTS

Nomenclature ix
List of Figures xiii
List of Tables xvii

CHAPTER 1

INTRODUCTION 1

1.1 Repetitive Construction Projects 1
1.2 Challenges in Scheduling Repetitive Construction 3
1.3 Research Objectives 6
1.4 Related Applications 7
1.5 Thesis Organization 9

CHAPTER 2

LITERATURE REVIEW 11

2.1 Introduction 11
2.2 Traditional Scheduling Techniques 12
2.3 Scheduling Techniques for Repetitive Construction 15
    2.3.1 Line of Balance (LOB) 16
    2.3.2 Linear Scheduling Method (LSM) 20
    2.3.3 Simulation Models 21
2.4 Considerations in Scheduling Repetitive Construction 25
    2.4.1 Integration of Repetitive and Non-repetitive Activities 25
    2.4.2 Optimized Scheduling 28
    2.4.3 Learning Curve 30
    2.4.4 Weather Impact 36

CHAPTER 3

PROPOSED SCHEDULING MODEL 45

3.1 Introduction 45

3.2 Object-Oriented Modelling 45
3.3 Analysis 47
3.3.1 Field Study 48
3.3.2 Repetitive Unit 50
3.3.3 Object Model 55
3.3.4 Dynamic Model 61
3.4 Design 62
3.4.1 Project 63
3.4.2 Date 65
3.4.3 Project-Data 66
3.4.4 Regular-Relation 66
3.4.5 Repetitive-Relation 68
3.4.6 Hetero-Relation 68
3.4.7 Activity 69
3.4.8 Non-Repetitive Activity 70
3.4.9 Repetitive Activity 71
3.4.10 Crew-Formation 73
3.4.11 Scheduling Calculation 75
3.4.12 Impact of Weather on Construction Productivity 83
3.5 Summary 86

CHAPTER 4

AN ALGORITHM FOR RESOURCE-DRIVEN SCHEDULING OF REPETITIVE ACTIVITIES 88

4.1 Introduction 88
4.2 Resource-Driven Scheduling 89
4.3 Proposed Scheduling Algorithm 95
4.3.1 Stage 1 97
4.3.2 Stage 2 105
4.4 Numerical Example 107
4.5 Summary 120

CHAPTER 5

SCHEDULING OPTIMIZATION 121

5.1 Introduction 121
5.2 Optimized Scheduling 121

5.3 Interruption of Crew Work Continuity 124
5.4 An Algorithm for Generating Feasible Interruption Vectors 130
    5.4.1 Limiting the Number of Interruption Vectors 130
    5.4.2 Proposed Interruption Algorithm 145
5.5 Optimization Procedure 150
    5.5.1 Forward Path 153
    5.5.2 Backward Path 158
5.6 Numerical Example 160
    5.6.1 Least Cost Solution 163
    5.6.2 Minimum Duration Solution 175
5.7 Summary 178

CHAPTER 6

IMPLEMENTATION OF THE SCHEDULING MODEL: LSCHEDULER 180

6.1 Introduction 180
6.2 Scheduling Module 181
6.3 User Interface Module 183
    6.3.1 Menus, Tool Bar, and Status Bar 183
    6.3.2 Dialog Boxes 196
    6.3.3 MDI Windows 203
6.4 Input and Output 204
6.5 Assumptions and Limitations 207
6.6 Application Example 208
6.7 Summary 214

CHAPTER 7

CONCLUSIONS 215

7.1 Conclusions 215
7.2 Research Contributions 218
7.3 Recommendation for Future Research 219

REFERENCES 221

APPENDIX I: Header Files 230

NOMENCLATURE

A:	Project cost in $.
AS_m:	Binary array indicating the availability status of crew n on site, where AS[n]=1 indicates that crew n is available and AS[n] = 0 indicates otherwise.
B:	Money value of project duration in $.
BEC^i:	Base equipment cost rate before overtime premiums (in $ per day) for activity i.
BLC^i:	Base labor cost rate before overtime premiums (in $ per day) for activity i.
BP^i:	Base output rate before overtime (in units of measurement per day) for activity i.
C^i_n:	Crew formation n of activity i.
C^{i-1}_{n*}:	Local optimum predecessor crew formation that yields the minimum overall cost (COC^i_n) of crew formation C^i_n.
COC^i_n:	Total project overall cost (in $) up to crew formation n of activity i.
$Crew_k$:	Identifies crew number assigned to unit k.
D:	Project duration in days.
D_1:	Duration to construct the first repetitive unit or section.
D_j:	Duration to construct the jth repetitive unit or section.
D_{2j}:	Duration to construct the 2jth repetitive unit or section.
D^i_{nj}:	Duration (in days) at section or unit j for crew formation n of activity i.
DC^i_n:	Total Direct cost (in $) for crew formation n of activity i.
DRUC:	Daily road-user cost in $/day.
early start:	The earliest possible start time of unit k due to crew availability.
EC^i_n:	Equipment cost rate ($ per day) for crew formation n of activity i.

F_k:	Scheduled finish time of unit k.
F^i_{nj}:	Planned finish date at section or unit j for crew formation n of activity i.
H:	Relative humidity as a percent.
i:	Repetitive activity number (from i = 1 to i = I).
$Inter^j_{nj}$:	Interruption time before the start of unit j using crew formation C^i_n.
IC^i_n:	Indirect cost of project (in \$) up to crew formation n of activity i.
ICR:	Project indirect cost rate (in \$ per day).
$Idle_k$:	Idle time imposed on the assigned crew to unit k due to compliance with activity precedence relationship constraint.
j:	Repetitive unit number (from j = 1 to j = J).
J:	Total number of repetitive units.
k:	Unit number that satisfies the user specified order of construction (Order[j]).
LC^i_n:	Labor cost rate (in \$ per day) for crew formation n of activity i.
L^i_{i-1}:	Lag (in days) between the succeeding activities i-1 and i.
m:	Crew number (from m = 1 to m = M).
M:	Total number of crews that can be assigned to construct the units of a repetitive activity, simultaneously.
$MaxAv_m$:	Latest available date of crew n on site.
MC^i_n:	Material cost rate (in \$ per unit of measurement) for crew formation n of activity i.
$MinAv_m$:	Earliest available date of crew n on site.
n:	Crew formation number (from n = 1 to n = N).
n*:	Local optimum crew formation.
NS_m:	Next possible start for crew n.

$Order_j$:	User-specified order of execution throughout all repetitive units.
OPF^i_{nj}:	Overall productivity factor due to the combined effect of weather and learning curve effect affecting crew formation C^i_n assigned to section or unit j.
OT^i_n:	Overtime (in hours) for crew formation n of activity i.
P^i_n:	Output rate (in units of measurement per day) for crew formation n of activity i.
$P_{assigned\ crew}$:	Daily output rate of the assigned crew.
PDC^i_n:	Cumulative direct cost of predecessors (in \$) for crew formation n of activity i.
PES_k:	Possible early start time of unit k due to logical precedence relationship.
PES^i_{nj}:	Possible early start date at section or unit j for crew formation n of activity i.
PF_c:	Productivity factor of general construction trade for cold weather.
PF_h:	Productivity factor of general construction trade for hot weather.
PFL^i_{nj}:	Productivity factor due to the learning curve effect for crew formation C^i_n assigned to section or unit j.
PFW^i_{nj}:	Productivity factor due to weather affecting crew formation C^i_n assigned to section or unit j.
Q_k:	Quantity of work in unit k;
Q^i_j:	Quantity of work (in units of measurement) at section or unit j of activity i.
R:	Learning rate representing the learning curve effect.
RC^i_n:	Interruption cost in \$ for crew formation C^i_n.
S^i_{nj}:	Planned start date at section or unit j for crew formation n of activity i.
S_k:	Scheduled start time of unit k.

$shift_j$:	Required shift of the start and finish times of unit k to comply with crew work continuity.
Slope:	Negative slope of a line depicting the learning curve effect on a log-log scale.
T:	Temperature in degree Fahrenheit.
TCB:	Total combined bid in $.
V_m:	Movement time of crew m to the next unit, if any.
W:	Total crew working hours per week.

LIST OF FIGURES

CHAPTER 1

Figure 1.1 Repetitive Activity Types 2

Figure 1.2 Minimizing the Duration of Repetitive Projects 5

CHAPTER 2

Figure 2.1 Graphical Representation of LOB and LSM 22

Figure 2.2 Hypothetical Learning Curve (Thomas et al 1986) 32

Figure 2.3 Learning Curve Models (Thomas et al 1986) 32

Figure 2.4 Learning Curve for Century Series Aircraft (Jelen & Black 1983) 34

Figure 2.5 Learning Curve for Century Series Aircaft on a Log-Log Scale 35

Figure 2.6 Weather Impact on Masons Productivity (Grimm & Wagner 1974) .. 43

Figure 2.7 Weather Impact on Construction Productivity (U.S. Army 1986) 43

CHAPTER 3

Figure 3.1 Montreal Housing Project 49

Figure 3.2 Winter Conditions on Site 50

Figure 3.3 Impact of Repetitive Unit Size on Scheduling 53

Figure 3.4 Class Representation 56

Figure 3.5 Activities and Relationships of a Repetitive Project 58

Figure 3.6 Proposed Object Model 60

Figure 3.7 State Diagram 62

Figure 3.8 State Diagram of *Project* 76

Figure 3.9 State Diagram of *Non-Repetitive* Activity 79

Figure 3.10 State Diagram of *Repetitive* Activity 80

Figure 3.11 State Diagram of *Regular-Relation* 81

Figure 3.12 State Diagram of *Repetitive-Relation* 81

Figure 3.13 State Diagram of *Hetero-Relation* 82

Figure 3.14 State Diagram of *Date* Class 84

CHAPTER 4

Figure 4.1 State Diagram of Crew Formation 89

Figure 4.2 Scheduling of a Typical Repetitive Activity 93

Figure 4.3 Scheduling of a Non-Typical Repetitive Activity 94

Figure 4.4 Stages Used in the Algorithm 96

Figure 4.5 Scheduling Algorithm 99

Figure 4.6 Project Schedule Without Interruptions 114

Figure 4.7 Project Schedule With Interruptions 115

CHAPTER 5

Figure 5.1 Minimizing Project Duration Using Activity Interruption 126

Figure 5.2 Possible Interruption Vectors for a Repetitive Activity 129

Figure 5.3 Identification of Upper Limit for Interruption 133

Figure 5.4 Comparing Interruption Vectors 3 and 17 for Crew 2 137

Figure 5.5 Comparing General Interruption Vectors A and B 138

Figure 5.6 Impact of Activity Interruptions on Project Duration 144

Figure 5.7 Dynamic Programming Formulation 152

Figure 5.8 Forward Path of the Optimization Procedure 159

Figure 5.9 Bridge Example (Selinger 1980) 160

Figure 5.10 Least Cost Solution 174

Figure 5.11 Least Duration Solution 177

CHAPTER 6

Figure 6.1 *LSCHEDULER* Modules 181

Figure 6.2 Developed Windows Application: *LSCHEDULER* 184

Figure 6.3 File Menu 185

Figure 6.4 Project Menu 186

Figure 6.5 Activity Menu 187

Figure 6.6 Relationship Menu 188

Figure 6.7 Edit Menu 189

Figure 6.8 Search Menu 190

Figure 6.9 Display Input Data Menu 191

Figure 6.10 Display Schedule Menu 192

Figure 6.11 Display Cost Estimates Menu 193

Figure 6.12 Display Optimization Output Menu 194

Figure 6.13 Window Menu 195

Figure 6.14 Open Dialog Box 196

Figure 6.15 Save As Dialog Box 197

Figure 6.16 Print Dialog Box 197

Figure 6.17 Print Setup Dialog Box 198

Figure 6.18 Open Project Dialog Box 198

Figure 6.19 Project Start Date Dialog Box 199

Figure 6.20 Weather and Learning Curve Dialog Box 199

Figure 6.21 Scheduling Optimization Dialog Box 199

Figure 6.22 Non-Repetitive Activity Dialog Box 200

Figure 6.23 Repetitive Activity Dialog Box 200

Figure 6.24 Non-Typical Repetitive Activity Dialog Box 201

Figure 6.25 Typical Repetitive Activity Dialog Box 201

Figure 6.26 Crew Formation Dialog Box 202

Figure 6.27 Crew Dialog Box 202

Figure 6.28 Relationship Dialog Box 203

Figure 6.29 *LSCHEDULER* Input and Output 205

LIST OF TABLES

CHAPTER 2

Table 2.1 Learning Rate for Construction Activities (Effect 1965) 34

Table 2.2 Weather Impact on Productivity of Electricians (Effect 1974) 42

Table 2.3 Effective Temperature (Effect 1974) 42

Table 2.4 Weather Impact on Piping (Department of Navy 1969) 44

Table 2.5 Impact of Rain on Work Stoppage (Cantwell 1987) 44

CHAPTER 3

Table 3.1 Data Members of *Project* Class 64

Table 3.2 Main Member Functions of *Project* Class 65

Table 3.3 Data Members of *Date* Class 65

Table 3.4 Main Member Functions of *Date* Class 65

Table 3.5 Data Members of *Project-Data* Class 66

Table 3.6 Main Member Functions of *Project-Data* Class 66

Table 3.7 Data Members of *Regular-Relation* Class 67

Table 3.8 Main Member Functions of *Regular-Relation* Class 67

Table 3.9 Data Members of Repetitive-Relation Class 68

Table 3.10 Data Members of Hetero-Relation Class 69

Table 3.11 Data Members of *Activity* Class 70

Table 3.12 Data Members of *Non-Repetitive* Class 70

Table 3.13 Main Member Functions of *Non-Repetitive* Class 71

Table 3.14 Data Members of *Repetitive* Class 72

Table 3.15 Main Member Functions of *Repetitive* Class 73

Table 3.16 Data Members of *Crew-Formation* Class 74

Table 3.17 Main Member Functions of *Crew-Formation* Class 75

CHAPTER 4

Table 4.1(A) Activity Input Data 110

Table 4.1(B) Activity Input Data (Continued) 111

Table 4.2 Crews' Input Data 113

Table 4.3(A) Project Schedule Without Interruption 116

Table 4.3(B) Project Schedule Without Interruption (Continued) 117

Table 4.4(A) Project Schedule With Interruption 118

Table 4.4(B) Project Schedule With Interruption (Continued) 119

CHAPTER 5

Table 5.1 Interruption Vectors (Russell and Caselton, 1988) 125

Table 5.2 Interruption Vectors for Crew 2 136

Table 5.3 Interruption Vectors Based on Rule 2 139

Table 5.4 Interruption Vectors for Crew 1 Based on Rule 2 141

Table 5.5 Interruption Vectors for Crew 2 Using a Step of 2 days 142

Table 5.6 Interruption Vectors for Crew 1 Using a Step of 2 days 142

Table 5.7(A) Possible Interruption Vectors 148

Table 5.7(B) Possible Interruption Vectors 149

Table 5.8 Basic Input Data 166

Table 5.9 Possible Crew Formations' Data 167

Table 5.10 Calculation of the Local Optimization Problem 168

Table 5.11 Calculation of the Optimum Solution Using the Proposed Model ... 169

Table 5.12 Impact of Project Indirect Cost 171

Table 5.13 The Optimum Solution of Selinger (1980) 172

Table 5.14 The Optimum Solution of the Two-State-Variable Formulation 173

CHAPTER 6

Table 6.1 File Menu Functions 185

Table 6.2 Project Menu Functions 186

Table 6.3 Activity Menu Functions 187

Table 6.4 Relationship Menu Functions 188

Table 6.5 Edit Menu Functions 189

Table 6.6 Search Menu Functions 190

Table 6.7 Display Input Data Menu Functions 191

Table 6.8 Display Schedule Menu Functions 192

Table 6.9 Display Cost Estimates Menu Functions 193

Table 6.10 Display Optimization Output Menu Functions 194

Table 6.11 Window Menu Functions 195

Table 6.12 Quantities of Work 209

Table 6.13 Crew Formations 211

Table 6.14 *LSCHEDULER* Optimization Results 213

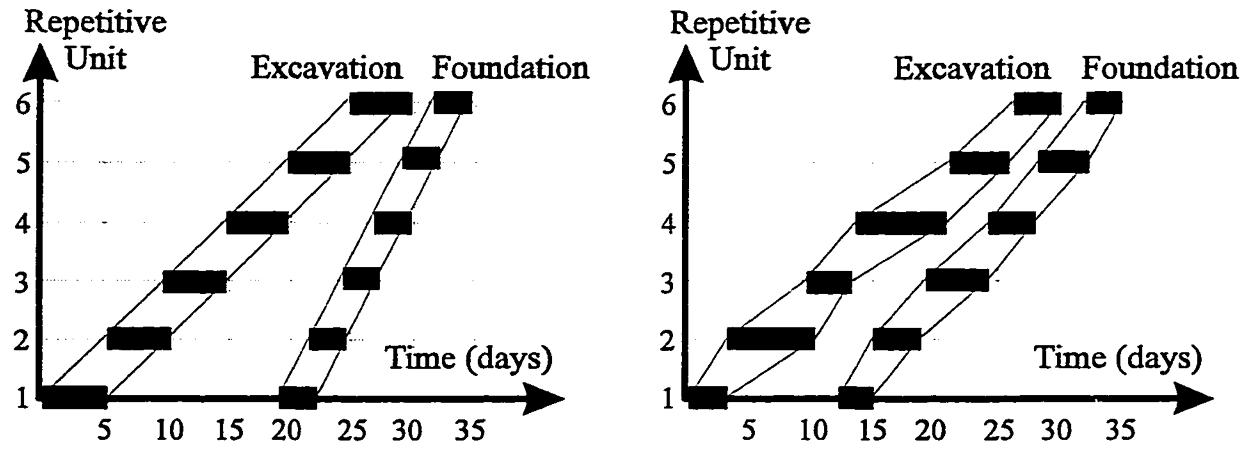
CHAPTER 1

INTRODUCTION

1.1 Repetitive Construction Projects

Repetitive construction projects consist of a number of similar or identical units. A unit could simply be a typical floor in a high-rise building, a model house in a housing project or a typical section of a pipeline network. Other examples of repetitive projects include the construction of: highways, airport runways, railways, bridges, tunnels, sewer mains and mass transit systems. In the literature, the term *linear projects* is also used to refer to this class of projects (Arditi and Albulak, 1979; Dressler, 1974; Mawdesley et al. 1989; and Selinger, 1980). In a repetitive project, construction activities can be either *repetitive* or *non-repetitive*. In a high-rise building, for example, the concrete activity that is repeated in each typical floor can be considered *repetitive*, while the excavation activity that is performed only once can be considered *non-repetitive*.

Vorster and Bafna (1992) suggested that repetitive projects be divided into two categories. In the first, repetitive activities have identical durations in all units, and can be represented graphically by two straight inclined lines as shown in Figure 1.1(a). An example of this category is a housing project, where the same set of activities performed in constructing a typical house is repeated in all housing units within the project. In the second category, repetitive activities do not have identical durations, and can be represented as shown in Figure 1.1(b). An example of this category is highway construction, where the time required for excavation may vary from one section to another. In this thesis, these two categories of projects will be referred to as *typical*, and *non-typical* repetitive projects respectively, and the general term *repetitive projects* will be used to refer to both categories.



(a) Typical **(b) Non-Typical**

Figure 1.1 Repetitive Activity Types

The majority of real world repetitive projects in construction can be classified as non-typical. The variation of an activity duration from one repetitive unit to another can generally be attributed to variations in the quantities of work encountered and/or crew productivity attained in performing the work in these units. For example, in earthmoving activities the quantity of excavation in each unit can be different because of the site topography, and productivity may vary from one unit to another due to the type of soil being excavated, learning curve effect, and/or weather impact (Moselhi and El-Rayes 1993(a)).

1.2 Challenges in Scheduling Repetitive Construction

In a repetitive project, a construction crew that is assigned to construct an activity in a number of repetitive units is often required to move from one unit to another. Scheduling the construction operations of this crew should be done in a such way to allow for maintaining crew work continuity in order to avoid unnecessary crew idle time (Ashley 1980, Birrell 1981, El-Ryaes and Moselhi 1997, Kavanagh 1985, and Reda 1990). As such, scheduling repetitive activities should be resource-driven so as to maximize the efficiency of resource utilization. A challenging task in scheduling repetitive activities is to develop and apply a resource-driven scheduling procedure that embraces flexibility and accounts for practical factors commonly encountered during construction.

As stated earlier, a repetitive construction project often includes repetitive as well as non-repetitive activities. Each of these two types of activities requires a unique scheduling technique. Non-repetitive activities can be scheduled using a traditional network-based tehnique. Repetitive activities, however, require a technique that is capable of providing resource-driven scheduling. The integration of the two scheduling techniques in an efficient operating scheduling model is a second challenging task (Chrzanowski and Johnston 1986, Moselhi and El-Rayes 1993, O'Brien 1985, Russell and Wong 1993).

A third challenging task in scheduling this class of projects is to select an optimum crew utilization option for each activity in the project so as to minimize the project duration or total cost. Minimizing the project duration is a more complicated process for repetitive projects than that for non-repetitive ones. For non-repetitive projects, the acceleration of critical activities results in a shorter overall duration for the project. For repetitive projects, however, this is not always true, due to the compliance with the crew work continuity constraint. In order to illustrate this fact, Figure 1.2 presents a project with five repetitive activities that are sequentially constructed. Figure 1.2(a) represents the schedule when normal productivity rates are achieved for all activities. Accelerating the critical activity of Beams, for example, produces a longer rather than a shorter overall duration for the project (Figure 1.2(b)), and decelerating or relaxing the Foundation activity results in a shorter duration for the project (Figure 1.2(c)).

Figure 1.2 illustrates that minimizing the duration of a repetitive project can be achieved by selecting an optimum crew utilization option for each repetitive activity. In this thesis, the word *crew formation* is used to describe a crew utilization option that may involve one or more construction crews with or without overtime policy. When faced with a project involving many repetitive activities, most of which have their own sets of possible *crew formations*, the challenging question confronting the project scheduler is: "Which is the optimum *crew formation* for each repetitive activity in the project ?".

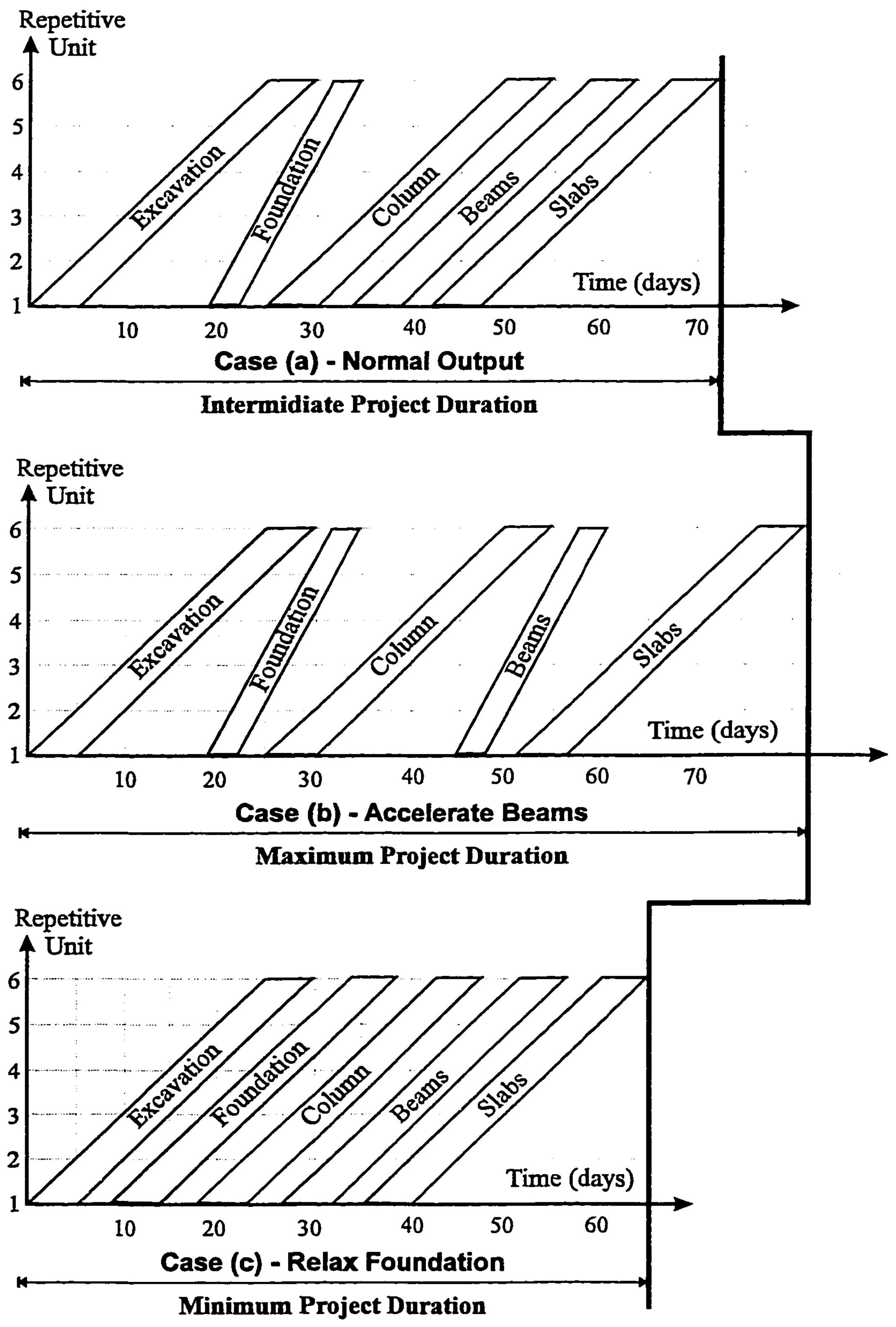


Figure 1.2 Minimizing the Duration of Repetitive Projects

Scheduling of repetitive construction projects is also affected by weather impact and learning curve effect. Weather impact has been reported to be one of the main factors causing delay and cost overruns on construction projects (Baldwin et al. 1971, Koekn and Meilhede 1981, and Laufer and Cohenca 1990). Learning curve effect causes the time required to perform a given construction activity to fall progressively as the same activity is repeated in a number of repetitive units. This is due to the fact that skill and productivity in performing tasks improves with experience and practice (Thomas et al. 1986). As such, learning curve affects the duration and scheduling of repetitive construction projects (Arditi and Albulak 1986, Diekmann et al. 1982, Drewin, 1982, and Hijazi et al 1992). Accordingly, it is of practical value to consider the impact of weather and/or learning curve effect on scheduling of repetitive construction projects. Accounting for weather impact and/or learning curve represents a fourth challenging task.

1.3 Research Objectives

The objective of this study is to develop a model for scheduling of repetitive construction projects that addresses the above mentioned challenges. In order to fulfill this objective, the following research sub-objectives are identified:

1) To provide a procedure for the integration of repetitive and non-repetitive scheduling techniques in an efficient operating scheduling model.
2) To develop an algorithm for resource-driven scheduling of repetitive activities that complies with precedence relationships, crew availability and crew work

continuity constraints, and enables the consideration of a number of practical factors commonly encountered during construction.

3) To provide an optimization procedure for generating least cost or minimum duration schedules that can be applied to all types of repetitive construction projects.
4) To automate the generation of feasible interruption options to facilitate the identification of an optimum crew formation and its associated optimum interruption option for each activity in the project.
5) To provide a procedure that enables the consideration of weather impact and/or learning curve effect on scheduling of repetitive construction projects.
6) To develop a prototype software system for scheduling of repetitive construction projects that a) incorporates the above algorithms and procedures; b) enables regular as well as optimized scheduling of repetitive construction projects; and c) provides user-friendly interface to facilitate input and output of scheduling data.

1.4 Related Applications

The proposed scheduling model can be applied to optimize the schedule of all types of repetitive construction projects, however one interesting application is highway construction and restoration projects. These projects often cause road closure and traffic congestion, resulting in lost time and money for the travelling public and local economy. In order to minimize such adverse effects, transportation agencies are currently exerting pressure on highway contractors to minimize construction time. Over the last few years, many of the state highway agencies across North America have started to apply a number of contracting and bidding methods in an attempt to reduce the construction duration of this class of projects. Bidding on cost and time is one of the most popular methods currently being used and its use has steadily increased after it was recommended by the Federal Highway Agency in the United States in 1991 (Herbsman 1993, and Herbsman et al 1995).

Bidding on cost/time is also known as (A + B) method, where "A" represents the project construction cost and "B" represents its duration. In this method, contractors are asked to bid on both the project cost and duration, and the major criterion for winning is the lowest total combined bid. The total combined bid combines the project cost and the money value of its duration as follows:

$$TCB = A + B = A + (D.DRUC) \tag{1.1}$$

where, TCB = total combined bid in \$, A = project cost in \$, B = money value of project duration in \$, D = project duration in days and DRUC = daily road-user cost in \$/day. DRUC is estimated by transportation agencies to represent the economic benefits of the road to the public and local economy. It often includes the public cost arising from the absence of the road such as those associated with additional travel time, travel distance and fuel expenses. The DRUC values were reported to vary from \$1000/day to a maximum of \$200,000/day in 101 highway projects analyzed by Herbsman (1995).

Highway contractors bidding on cost and time contracts are under a growing pressure to optimize their resource utilization so as to minimize both time and cost of these projects. A contractor attempting to reduce project time often utilizes additional resources which leads to additional project costs, and therefore the objective of minimizing project time often conflicts with that of minimizing its cost. The proposed model for optimized scheduling can be applied to highway construction projects in order to establish an optimum balance between project time reduction and its associated additional costs. This assists a contractor in formulating an optimum resource utilization strategy for delivering the project, ensuring maximized profit.

1.5 Thesis Organization

Chapter 2 presents a literature review of scheduling techniques used for repetitive construction projects. In addition, a number of practical factors that affect the scheduling process are discussed.

Chapter 3 introduces the analysis and initial design stages of a proposed object-oriented model for scheduling of repetitive construction projects. The analysis stage presents a field study and proposed object and dynamic models. The initial design stage describes classes, scheduling calculation and consideration of weather impact in the proposed model.

The detailed design of the proposed model is described in Chapters 4 and 5. In Chapter 4, practical factors that affect resource-driven scheduling of repetitive construction activities are first discussed, and a proposed flexible algorithm for resource-driven scheduling is then presented. A numerical example is analyzed to illustrate the use of the algorithm and demonstrate its capabilities.

Chapter 5 presents the design stage of the scheduling optimization functions incorporated in the present model. It describes two newly developed algorithms for generating feasible interruption options and performing scheduling optimization. A numerical example from the literature is analyzed to validate the algorithms and demonstrate their capabilities.

Chapter 6 presents the implementation stage of the present object-oriented model. The chapter describes the development of a prototype software system, and outlines its main modules, limitations, and input and output.

In chapter 7, the results of this research are summarized, the main contributions are stated, and recommendations for future research are presented.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The development of a project schedule serves several purposes before, during, and after the construction stage. Before construction, the schedule provides planned start and finish dates for individual activities as well as the project. This information is often demanded during the bidding stage according to the stipulation of many contracts. Moreover, the schedule is used to identify the resource requirements for the project in order to plan in advance the timely allocation of needed materials, labor, equipment, and money.

During construction, the schedule works as a guideline for executing the project. In order to control the project, the actual performance is measured and compared to the schedule to evaluate the progress of work. This enables early detection of deviations from the planned performance and accordingly initiating corrective actions, if needed. After construction, the as-planned schedule can be compared to the as-built schedule to facilitate analysis of claims and disputes that may arise. In addition, analyzing the as built schedule provides valuable historical data that can be used in future projects. For construction projects, in general, a number of traditional scheduling techniques are available in addition to those specially developed for repetitive construction.

2.2 Traditional Scheduling Techniques

During world war I, Henry L. Gantt developed a graphical display for relating the progress of work to a time schedule (Antill and Woodhead 1990). This was in the form of bar charts that represent work activities. Depending on the time scale of the chart, the length of bars represents the duration assigned for each activity. In spite of the advent of network planning methods, the bar chart schedule is still widely used in construction work because of its graphic and easily understood format (Nunnally, 1987). The bar chart, however, does not illustrate the logical interrelationships among project activities, and cannot identify the critical activities affecting project duration (Chrzanowski and Johnston 1986, and Stradal and Cacha 1982).

In order to overcome the limitations of bar charts, the network techniques were developed for scheduling construction projects during the period 1956-1958 (Antill and Woodhead 1990). Network techniques enable the identification of the critical activities that affect the project duration. Currently, there are two common types of networks for representing the project activities and the interrelationships among them. The first is the arrow diagram method (ADM), where activities are represented by arrows and the nodes connecting these arrows are considered events or milestones. The second type is activity on node diagram and often called precedence diagram method (PDM). In PDM, activities are represented by nodes and the arrows connecting them depict the interdependencies among these activities. ADM considers only one type of precedence relationship (finish to start), while PDM enables the consideration of different types of relationships (finish to start, start to start, finish to finish or start to finish). More detailed information about network scheduling techniques can be found in (Antill and Woodhead 1990; Harris 1978; O'brien 1965; and O'brien 1969).

Network scheduling techniques have been criticized in the literature because they focus on project duration and give little consideration to maximizing the efficiency of resource utilization (Birrell, 1980; Davies, 1974; Canadian Construction Association Business and Contractor Relation Committee 1974; and Kavanagh 1985). Birrell (1980) criticized the use of CPM and PERT in construction and argued that they were originally developed for military and industrial environment, where the United States national security emphasized the completion of the project and gave little consideration to the efficient use of resources. In construction, however, he suggested that contractors are much more interested in efficient use of resources more than early project completion. Kavanagh (1985) also criticized CPM for the same reason, and described it as a poor model of the construction process because it does not include the priorities of the controller (i.e. the site superintendent), who is most concerned about resource utilization not critical paths.

The application of traditional scheduling techniques to repetitive construction projects has been widely criticized in the literature because: 1) they produce complex and redundant schedules for repetitive projects (Johnston et al. 1986, Reda 1990, Suhail and Neale 1994, and Stradal and Cacha 1982); and 2) they are incapable of maintaining work continuity for crews that are assigned to repetitive activities (Selinger 1980, Reda 1990, Russell 1990, and Russell and Wong 1993). The complexity and redundancy of the produced schedules can be illustrated using a housing project example as the one presented in (Carr and Meyer, 1974). The example involved the construction of 200 simple housing units. The construction work of a single housing unit was broken down into 24 activities which can be represented by a simple network. A project network for the construction of 200 similar units requires 200 repetitions of the simple network of the 24 activities. The resulting network of 4,800 activities for the whole project is highly complex and its representation of the project includes significant redundancies. This complexity and redundancy problem increases directly with the increase in repetitions.

Traditional scheduling techniques are also incapable of maintaining crew work continuity, even if resource allocation is utilized. Crew work continuity provides for an effective resource utilization strategy, particularly for repetitive activities. Crews working on such activities are often involved in movement from one repetitive unit to the next, and should be scheduled in such a way to allow for a crew to finish work on one repetitive unit, then be able to move promptly to the next without delay. The application of crew work continuity during scheduling leads to maximizing the use of learning curve and minimizing idle time of each crew (Ashley 1980 and Birrell 1981).

2.3 Scheduling Techniques for Repetitive Construction

As a result of the above mentioned limitations of traditional scheduling techniques, a number of alternative techniques have been proposed in the literature for scheduling of repetitive construction projects. These techniques attempted to maintain crew work continuity constraint and can be grouped into two main categories. The first includes techniques designed only for *typical* repetitive projects (Al Sarraj, 1990; Arditi and Albulak, 1986; Carr and Meyer, 1974; NBA, 1966; and Lumsden, 1968), and the second includes techniques that are suited for both; *typical* and *non-typical* repetitive projects (Selinger, 1980; Johnston, 1981; Chrzanowski and Johnston, 1986; and Russell and Caselton, 1988).

Most of the techniques that were developed for scheduling of repetitive projects fall in the first category. Such techniques are often called the Line of Balance (LOB) (Al Sarraj, 1990; Arditi and Albulak, 1986; Carr and Meyer, 1974; NBA, 1966; and Lumsden, 1968). As stated earlier, many repetitive construction projects are of the *non-typical* type, thus rendering the LOB type techniques inadequate. The second category of scheduling techniques are often called the Linear Scheduling Method (LSM) (Selinger, 1980; Johnston, 1981; Chrzanowski and Johnston, 1986; and Russell and Caselton, 1988). LSM techniques can be used for scheduling both *typical* and *non-typical* repetitive projects, thus have apparent advantage over the LOB type techniques.

Other techniques for scheduling of repetitive projects have also been proposed in the literature using other names including: Vertical Production Method (VPM) (O'Brien, 1975; O'Brien, 1984; and O'Brien et al. 1985), Chain Bar-Charts (Mawdesley et al. 1989; and Perera 1981), Fenced Bar-charts (Melin and Whiteaker, 1981; and Melin, 1984), Time-Space Scheduling (Oldrich and Cacha, 1982), Repetitive Project Modelling (RPM) (Reda, 1990), Simulation of Repetitive Networks (SIREN) (Kavanagh, 1985), Construction Management System (Pedersen, 1972), Velocity Diagrams (Roech, 1972), and Cascade Networks (Rist, 1972). All these techniques share the same concept of maintaining crew work continuity during scheduling of repetitive activities.

2.3.1 Line of Balance (LOB)

The line of balance (LOB) technique was developed in 1942 by the U.S. Navy for planning and control of repetitive projects (Lumsden 1968). LOB was originally developed for industrial manufacturing applications, with the objective of determining the resources and speed for each stage of manufacturing in order to achieve the required output rate. Scheduling of repetitive projects is similar to that of a factory production in maintaining the continuity of work for operating crews. In 1966, the LOB technique was modified from its original manufacturing industry purposes to enable its application to housing projects in the construction industry in a published report by the National Building Agency (Programming 1966). The developed LOB schedule was presented in the form of a diagram, where the number of housing units was plotted versus time and the repetitive activities were represented by inclined bars. Ever since that report, the literature has revealed several scheduling techniques for repetitive projects which were also named LOB and shared the same concepts (Arditi and Albulak, 1979; Arditi and Albulak, 1986; Al Sarraj, 1990; Carr and Meyer, 1974; Harbert 1976; and Lumsden, 1968).

Arditi and Albulak (1979 and 1986) suggested that the large majority of planners use CPM only in non-repetitive projects because it is not an effective tool for repetitive construction. They tested this hypothesis by using both network analysis and LOB to schedule the same repetitive project of highway surface dressing to enable a comparison of the two techniques. They concluded that the LOB schedule brings better insight to the project situation, requires less time and effort to produce and provides a smoother flow of working crews than that of the CPM. They cautioned, however, that LOB is sensitive to errors in duration estimates and that the degree of detail of LOB must be carefully evaluated, as the graphical representation of too many activities can be hard to understand and that of too few activities of little use. In addition, they pointed out that a major difficulty in preparing LOB diagram lies in plotting overlapping activities having the same rate of output, and suggested using different colors for representing such activities.

Another solution for that latter problem of graphical representation has been proposed by Mansur (1989) and Hegazy et al (1993) in the form of an enhanced schedule representation, which splits the LOB diagram to two parts (i.e. top and bottom) and presents critical activities on the top part of LOB diagram and the non critical ones on the bottom part, thus avoiding overlapping of concurrent activities.

Stradal and Cacha (1982) proposed a graphical method for scheduling of repetitive projects which is similar to LOB, however it was named "Time Space Scheduling Method". They used this method to schedule project examples of pump foundation, apartment complex, multistory concrete building, a road section and a railway bridge. They concluded that this scheduling method provides smooth use of resources and simple graphical presentation.

In an attempt to provide an alternative to the graphical method of LOB, Al Sarraj (1990) provided a mathematical formulation for LOB to facilitate finding the start and finish times for each activity in every unit, the available buffer time for each activity in every unit, and information about the intersection place between succeeding activities. In another attempt, Hegazy et al. (1993) provided a computerized formulation of LOB in the form of a prototype PC-based computer program (BAL) for scheduling and control of repetitive projects. The scheduling calculations of BAL are based on LOB formulation provided by Harris and McCaffer (1989). Another variation of LOB was proposed by Birrell (1980), who introduced a construction planning and control model. The objective of the model was to achieve efficient resource utilization by maintaining crew work continuity constraint. The model is similar to LOB, however it is in a matrix format that includes two axes representing time phases and unit locations, where each crew flows through the matrix along its designated diagonal path.

Suhail and Neale (1994) proposed to combine the use of CPM and LOB techniques to schedule repetitive construction projects. They proposed using a LOB diagram to identify the required number of crews to complete each repetitive activity in the project. They also suggested using a CPM to create a number of typical networks equal to the number of typical houses. The start-events of these typical networks are linked to a start milestone activity such as project-start, and the finish-events to project-completion. As such, all the typical networks become subnetworks and can float between the project-start and project-completion activities. The number of the required crews for each typical activity calculated by LOB, is then used to perform resource allocation in CPM network. This approach utilizes CPM to schedule repetitive activities, and as such it is incapable of maintaining crew work continuity.

Thabet and Beliveau (1994) proposed to incorporate work-space constraint in scheduling repetitive work in a multistory building. They introduced a scheduling model which attempted to define and quantify work-space as a function of two parameters: work-space demand and work-space availability. The model proposed a procedure to compare work-space demand to work-space availability, which results in prompt scheduling of an activity without delay if demand is less than or equal availability. Otherwise, one of three scheduling actions is selected to account for the lack of work-space. These actions are: 1) decrease production rate; 2) interrupt flow of activity; or 3) delay activity start.

For scheduling of repetitive projects, LOB scheduling technique has apparent advantages over traditional ones. The LOB technique is demonstrated to have significant advantages including: maintained work continuity, resource-driven schedule development and informative schedule representation (Hegazy et al 1993, and Reda 1990). Although of the above mentioned advantages of LOB, its use as a scheduling technique for repetitive projects has been also criticized in the literature. Kavanagh (1985) suggested that LOB was designed to model simple repetitive production processes, and therefore it does not transplant readily into the complex construction environment. The inability of LOB to model the learning curve effect which is an important characteristic of repetitive projects has also been criticized by Ashley (1980). Arditi and Albulak (1986) discussed visual problems associated with the LOB graphical presentation in the case of overlapping activities having the same output rate as mentioned earlier. The LOB also assumes that the duration of all repetitive units of a given activity are identical, which limits the application of LOB to typical repetitive projects only. As stated earlier, many repetitive construction projects are of the non-typical type, thus rendering the LOB type techniques inadequate (Moselhi and El-Rayes 1993(b), and Hegazy et al 1993).

2.3.2 Linear Scheduling Method (LSM)

Linear Scheduling Method (LSM) is similar to LOB technique in maintaining crew work continuity, however it is capable of scheduling non-typical repetitive projects (Selinger, 1980; Johnston, 1981; Chrzanowski and Johnston, 1986; and Russell and Caselton, 1988). Another difference between LOB and LSM is the graphical presentation. Instead of the two parallel lines in LOB, each activity in LSM is represented by only one line of constant or sometimes changing slope which represents the activity rate of production. Figure 2.1 depicts the two different graphical representations of LOB and LSM for the same repetitive activity (excavation).

Johnston (1981) described the basic format of presentation for the LSM method as having two axes. One axis plots time while the perpendicular axis plots repetitive unit along the length of the project, and repetitive activities are represented by diagonal lines. He suggested that the most significant advantage of LSM is the simplicity with which it can convey a detailed work schedule. In an attempt to evaluate the capabilities of LSM, Chrzanowski and Johnston (1986) applied both LSM and CPM to schedule a roadway project in order to identify the advantages and disadvantages of each. They concluded that LSM has several attributes, the most obvious of which is its simplicity, and suggested that the user of LSM receives fairly detailed information (e.g. job progress, resource allocation and work flow through the project), without being confronted with the numerical data found in network methods. Chrzanowski and Johnston (1986), however, noted some of the limitations of LSM namely: 1) the difficulty of integrating non-repetitive activities in LSM diagrams; and 2) the fact that LSM is essentially graphical, and cannot be adopted to numerical computerization as readily as network methods.

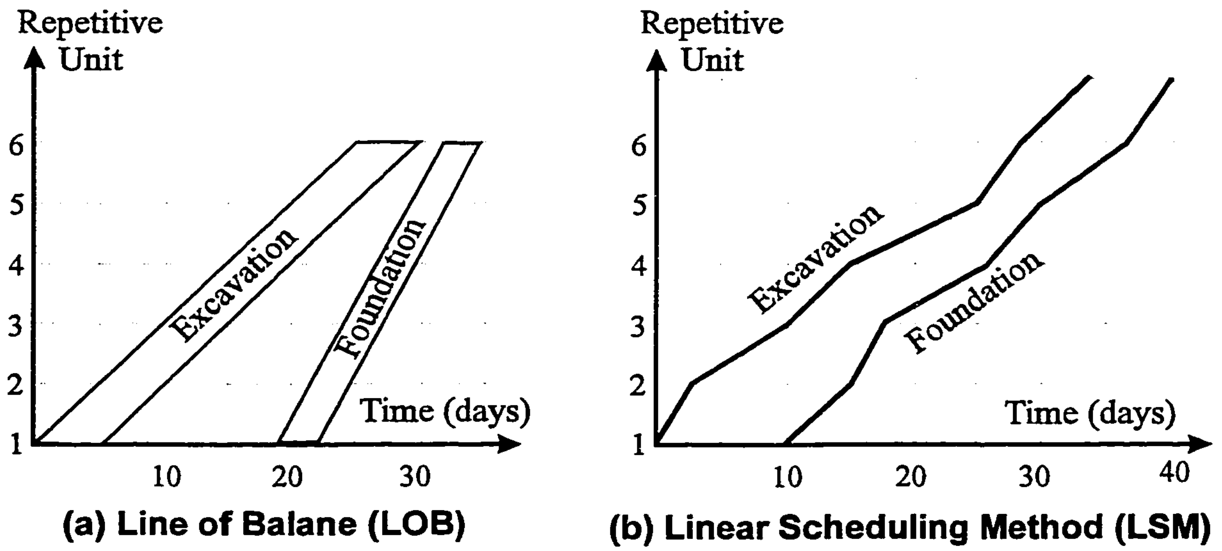


Figure 2.1 Graphical Representation of LOB and LSM

2.3.3 Simulation Models

In other attempts, simulation models have been proposed for scheduling of repetitive projects (Harris and Evans 1977, Ashley 1980, Kavanagh 1985, and

Abourizk and Halpin 1990). Harris and Evans (1977) presented a simulation game for a repetitive project of road constructions. The game involved 120 individual players, who were asked to make weekly decisions about resource utilization for the activities of the project. Based on the decisions of each player, the simulation game is run to report the project duration and cost to help the players understand the consequences of their decisions. Harris and Evans (1974) mentioned that this simulation game can be used for teaching purposes to train students and construction managers.

Ashley (1980) proposed a simulation model for scheduling of repetitive projects that attempted to resolve the crew availability problem by adopting a queuing model. The model is implemented using GPSS simulation language, and is based on the concept that repetitive units are organized in a queue to be served by the assigned crew. In 1985, Kavanagh introduced an extended simulation model that is based on the same queuing concept and implemented it using the same language. The model, however, was extended to enable the inclusion of non-repetitive activities and a simplistic consideration of learning curve effect and weather impact. Although these two simulation models attempted to resolve the crew availability problem, they do not maintain crew work continuity constraint which result in idle crew time. In his recommendations for future enhancements, Kavanagh (1985) recommended that the ability of enforcing crew work continuity be provided, and weather impact be modeled more accurately. Abourizk and

Halpin (1990) reviewed basic techniques used in simulation and presented an example of an earth-moving operation to demonstrate the use of such techniques. For simulation of construction operations, they presented a review of: 1) modeling of input data (i.e. identifying an appropriate distribution, estimating parameters of a given distribution, and testing for goodness of fit); 2) analyzing output data (i.e. checking for normality and estimating mean, variance and probabilities); and 3) validating results.

In other attempts to consider uncertainty in scheduling repetitive construction, Dressler (1974) proposed a stochastic model for scheduling linear construction sites. The model utilized a stochastic linear programming formulation to consider the variability in crew productivity on linear construction sites. The formulation, however, is limited to single repetitive construction activities and cannot consider logic precedence relationships among succeeding repetitive activities. As such, the formulation cannot provide a schedule for a repetitive construction project comprising of a number of succeeding repetitive activities.

The above scheduling techniques have been applied to schedule a wide variety of repetitive construction projects. Such applications included housing projects (Carr and Meyer, 1974; and Ashley, 1980), high rise buildings (O'brien, 1975; Mangin, 1979; and Larame, 1983), highway and road construction (Harris and Evans, 1977; Johnston, 1981; and Chrzanowski and Johnston, 1986), pipelines (Dressler, 1974; and Levine et al. 1976), and pavement construction (Arditi and Albulak, 1986). It should be noted, however, that none of the above discussed techniques attempted to optimize the scheduling of repetitive construction projects.

2.4 Considerations in Scheduling Repetitive Construction

The literature indicates that scheduling repetitive construction projects is affected by: 1) integration of repetitive and non-repetitive activities; 2) optimized scheduling; 3) learning curve effect; and 4) weather impact. These factors and their impact on scheduling repetitive construction projects are discussed in the following sections.

2.4.1 Integration of Repetitive and Non-repetitive Activities

Repetitive projects are characterized by the presence of large number of repetitive activities, along with a number of non-repetitive activities. An example illustrating this fact is a high-rise building project. At early stages, activities including: move in, excavation and foundation are considered non-repetitive activities because they are carried out only once. After the initial stage of construction, the high-rise project reaches a certain point where the type of work changes to the repetitive mode. This usually happens as the construction of the first typical floor begins when all tasks that are repeated in the above floors are considered repetitive activities. Concrete superstructure, masonry, glazing, and drywall are examples of these activities (O'Brien 1985). Crews assigned to repetitive activities move continuously from one floor to the upper as the work progresses.

Chrzanowski and Johnston (1986) criticized Linear Scheduling Method (LSM) because its use is limited to repetitive activities and does not cover non-repetitive activities. The problem of scheduling repetitive projects arises from the fact that repetitive activities require different scheduling technique from that used for non-repetitive ones. The non-repetitive group of activities in the project can be scheduled using traditional network-based techniques such as the critical path method. Repetitive activities, however, require techniques that are capable of providing a resource-driven schedule that satisfies job logic, crew availability and crew work continuity constraints. Integration of the two scheduling techniques, in an efficient operating scheduling model is, therefore, of practical value.

For scheduling purposes, O'Brien (1975 and 1985) suggested that a high-rise building can be considered as a hybrid project that can be separated to two distinct modes of construction: 1) non-repetitive mode (e.g. excavation) and 2) repetitive mode that starts as the construction of the typical floor begins. He recommended that the non-repetitive portion of the project be scheduled using network techniques, and the repetitive portion using "Vertical Production Method" (VPM), where VPM is a graphical method that is similar to LOB. He also recommended that the two distinct schedule formats (i.e. network and VPM) be carried out separately and combined in a narrative report.

In 1983, Laramee studied the construction process of two high-rise buildings in order to formulate specifications for a planning and scheduling model for high-rise buildings. Although that model proposed a merge algorithm between the network technique and techniques for repetitive activities, it did not account for practical factors affecting the scheduling of repetitive activities such as: 1) type of activity (i.e. typical or non-typical); 2) various types of precedence relationships; 3) crew availability period on site; and 4) activity interruption.

In 1993, Moselhi and El-Rayes proposed an object-oriented model for integrating the scheduling of repetitive and non-repetitive activities. In their model, two types of activities were considered: *Repetitive* and *Non_Repetitive*. In order to account for different possibilities of generalized precedence relationships connecting these two types of activities, three different types of precedence relationships were introduced in the model: *Regular_Relation*, *Repetitive_Relation* and *Hetero_Relation*. The model utilized objects to represent different types of activities and relationships. The design of these objects incorporates and integrates the two scheduling techniques for repetitive and non-repetitive activities.

Russell and Wong (1993) presented a construction management system (REPCON) for scheduling of repetitive projects. REPCON attempts to combine the non-repetitive and repetitive activities within the same model. The development of the system is based on a family of five planning structures: continuous, ordered, shadow, cyclic and non-repetitive activity. The first four structures represent different types of a repetitive activity (e.g. a continuous activity maintains crew work continuity while an ordered activity allows work continuity to be interrupted). The precedence relationships among these five types of activities can be defined as: typical and non-typical. The typical relationship is used to link repetitive planning structures and the non-typical relationship is used to link non-repetitive activity to a repetitive activity. REPCON provides the ability to combine repetitive and non-repetitive activities within the same model, however it is not capable of providing optimized schedules for repetitive projects. Russell and Wong (1993) stated that their current implementation does not provide for formal optimization in terms of minimizing duration, direct cost or total cost. In addition REPCON does not consider the impact of weather nor learning curve effect in the scheduling process.

2.4.2 Optimized Scheduling

Minimizing the project duration is a more complicated process for repetitive projects. than that for non-repetitive ones due to the compliance with crew work continuity. The problem of optimizing the schedule for a repetitive project can be illustrated through the use of a simple project example as the one discussed later in chapter 5. The project example consists of four similar sections or units, each includes five repetitive activities: excavation, foundations, columns, beams, and slabs. For each of these activities the following number of possible crew formations can be: 1, 2, 3, 3, 4 and 2, respectively. A combinatorial approach can be used to solve such a problem by generating all possible combinations and selecting the one that provides the minimum project duration or overall cost. This enumeration approach becomes infeasible when the size of the problem increases (Cooper and Cooper, 1981). For this simple example, the enumeration approach requires calculating the project duration for 72 possible combinations of options.

For a general repetitive project consisting of I activities and N possible crew formations for each activity, the number of possible combinations is N^I. For example, a repetitive project consisting of 20 activities associated with 5 possible crew formations for each activity can be solved using the enumeration approach by calculating the project duration for 5^{20} possible combinations (approximately 100 trillion) . This clearly illustrates that the enumeration approach becomes more and more infeasible as the size of the problem increases.

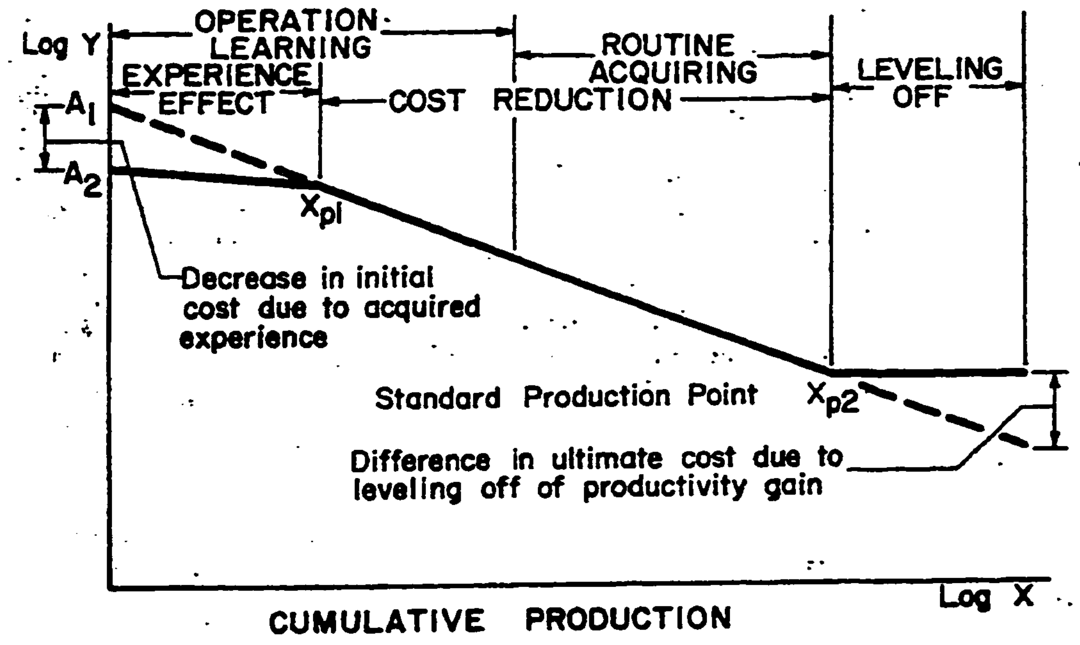
Another approach for optimizing the schedule of repetitive projects is by utilizing optimization techniques such as: linear programming (Perera 1982 and 1983, and Reda 1990), and dynamic programming (Selinger 1980, and Russell and Caselton 1988). Existing linear programming models assume that repetitive activities must have identical durations, and thus cannot provide solutions to non-typical repetitive projects. Available dynamic programming provide a flexible methodology that overcomes the limitations of linear programming models, however their optimization criterion is limited to the minimization of the overall duration of the project. While this may lead to the minimization of the indirect cost of the project, it does not guarantee its overall minimum cost.

2.4.3 Learning Curve

In repetitive construction projects, the time required to perform a given construction activity tends to fall progressively as the same activity is repeated for a sufficient number of successions (Hijazi et al 1992). This phenomenon is known as the learning curve and it is based on the fact that skill and productivity in performing tasks improves with experience and practice (Paulson, 1975). The improvements are due to several factors including: 1) greater familiarity with the task; 2) better equipment and crew coordination; 3) improved job organization; 4) enhanced engineering support; 5) better day-to-day management and supervision 6) more effective use of tools and methods; and 7) increased efficiency of material supply systems (Thomas et al. 1986).

The learning curve theory states that whenever the production quantity of a product doubles, the unit time will decline by a certain percentage of the previous unit time which is called the learning rate. The hypothetical learning curve can be divided into two phases: an operation-learning phase and a routine-acquiring phase as shown in Figure 2.2. In the operation-learning phase, labor productivity increases rapidly as workers gain knowledge and learning of the operations to be performed. In the routine-acquiring phase, more gradual improvement in labor productivity occurs due to becoming more familiar with the operation routine. Beyond the end of the routine-acquiring phase (i.e. the standard production point), no further productivity improvements occur due to the levelling off process as shown in Figure 2.2 (Thomas et al 1986). There are a number of available mathematical models that have been developed to model the learning curve phenomenon based on historical data. Thomas et al (1986) suggested that such models can be grouped under five basic types: 1) the straight line model; 2) the Stanford "B" model; 3) the cubic power model; 4) the piecewise (stepwise) model; and the 5) exponential model as shown in Figure 2.3.

The straight line model was originally developed by Wright (1936) for modeling the production of airplanes, and is based on the assumption that the learning rate remains constant. The Stanford "B" model is a modification of the straight line model to account for acquired experience in the first few cycles. The cubic model assumes that the learning rate may vary over time to account for acquired experience in the first few cycles and the levelling off in improvement as the project nears completion (Carlson 1973). The piecewise model is an approximation of the cubic model in the form a linearized model. The exponential model was developed by the Norwegian Building Research Institute and was described in a United Nations report (Effect 1965).



Log Y
OPERATION LEARNING
ROUTINE ACQUIRING
LEVELING OFF
EXPERIENCE EFFECT
COST REDUCTION
A1
A2
Xpl
Decrease in initial cost due to acquired experience
Standard Production Point
Xp2
Difference in ultimate cost due to leveling off of productivity gain
Log X
CUMULATIVE PRODUCTION

Figure 2.2 Hypothetical Learning Curve (Thomas et al 1986)

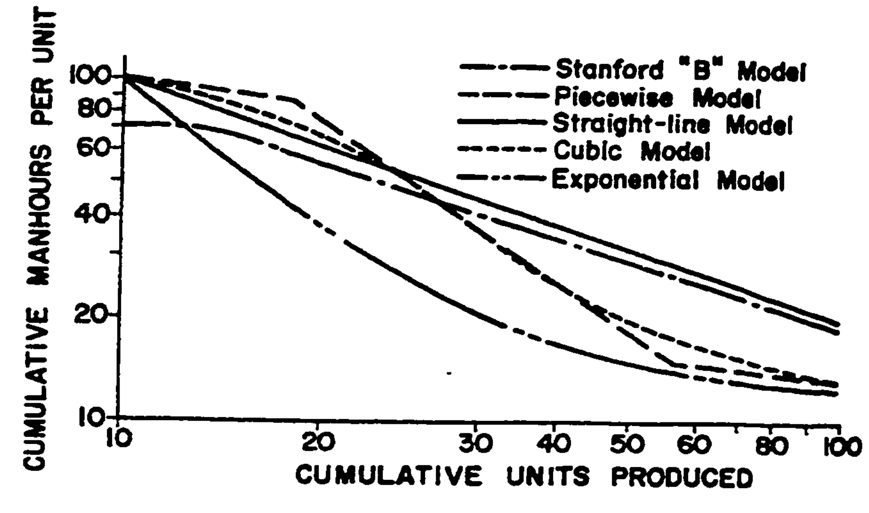


Figure 2.3 Learning Curve Models (Thomas et al 1986)

The most commonly used model for construction industry is the straight-line model due to the difficulty associated with estimating the parameters describing other models (Thomas et al 1986, Hijazi et al 1992, Gates et al. 1976, Diekmann et al. 1982, and Drewin, 1982). The name "straight-line" is commonly used because the model can be represented by a straight-line on a log-log scale.

Figure 2.4 shows a straight-line learning model for the manufacturing of the century series aircraft data. The data relates the direct-work hours per pound and cumulative number of planes (Jelen and Black, 1983). The curve shows a constant rate of time reduction (learning rate) as the aircraft number doubles. This learning rate (R) was found to be the same every time the production was doubled as follows:

$$R = D_{2j} / D_j \tag{2.1}$$

where,

D_j: time required to construct the j^{th} repetitive unit or section;

and D_{2j}: time required to construct the repetitive unit or section number 2j.

The learning rate (R) reflects the improvement in productivity rate because of learning curve effect. This rate mainly depends on the specific type of work involved (Hijazi et al 1992, and Tanner 1985), which can be determined based on historical data obtained from field observations during previous work. In a study by the Economic Commission For Europe (Effect 1965), the learning rate (R) was reported for different construction activities as shown in Table 2.1. The rate varied from 95% for entire structures to 80% for formwork panels. The R values were derived from data of previous report published by the Economic Commission For Europe (Cost 1963).

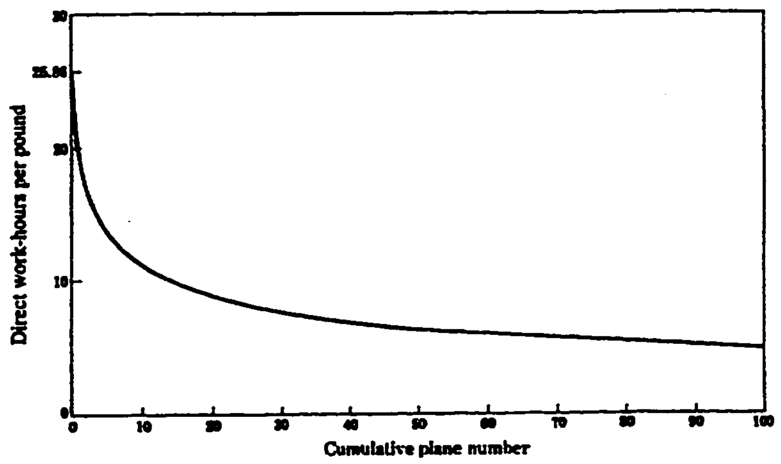


Figure 2.4 Learning Curve for Century Series Aircraft (Jelen & Black 1983)

Table 2.1 Learning Rate for Construction Activities (Effect 1965)

Construction activity (1)	Learning rate (R) x 100% (2)
Entire structure (e.g. high-rise buildings).	95%
Activities involving many operations (e.g. carpentry, electrical work, plumbing, and concreting).	90%
Activities involving few operations (e.g. masonry, painting, and floor and ceiling tile).	85%
Activities involving few operations (e.g. formwork panels, bar bending, field fabrication of trusses).	80%
Plant manufacture of building components (e.g. doors, windows, and prefabricated concrete panels)	90% - 95%

Figure 2.5 shows the same data of the Century series aircraft plotted on a log-log scale. In this graph, the relation between the two variables is represented by a straight line with a negative slope (Slope). The slope can be calculated from the original learning rate (R) as follows:

$$Slope = \log R / \log 2 \tag{2.2}$$

The negative slope of the line indicates that the time necessary to construct a repetitive unit is a fraction of the duration of the first unit. Given the slope of the line (Slope) and the duration of the first repetitive unit (D_1), the required duration to construct the j^th^ repetitive unit (D_j) can be calculated as follows:

$$D_j = D_1 . (j)^{Slope} \tag{2.3}$$

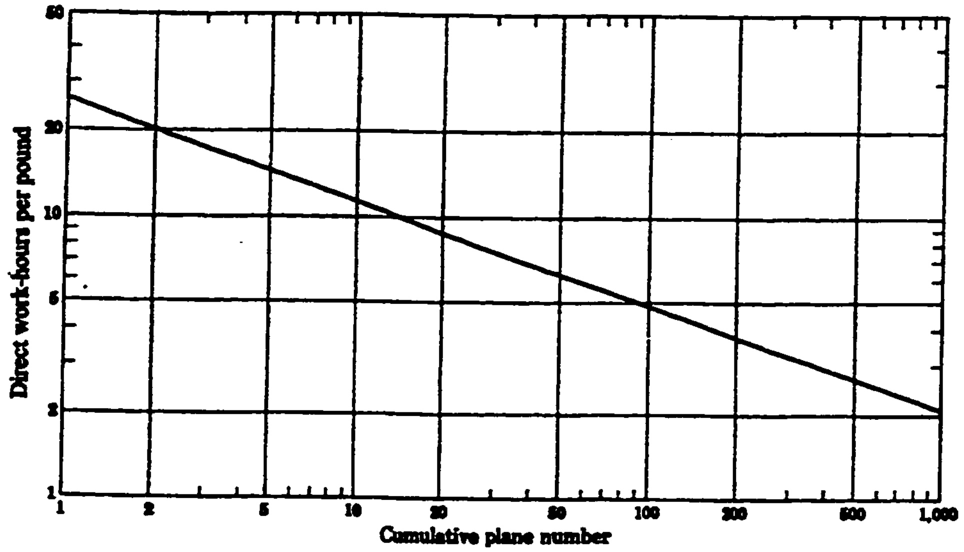


Figure 2.5 Learning Curve for Century Series Aircaft on a Log-Log Scale (Jelen and Black 1983)

Arditi and Albulak (1986) recommended that the possibility of incorporating learning curve into planning and scheduling of repetitive projects requires further research. Hijazi et al (1992) suggested that the essence of repetition and continuity inherent in repetitive construction projects allows for such a phenomenon to occur. They concluded that a repetitive project duration is often overestimated when the effect of learning curve is not considered. In another study, Gates and Scarpa (1978) suggested that the learning curve phenomenon affects the selection of the optimum number of crews to construct a repetitive activity. An increase in the number of crews leads to a decrease of the number of repetitive units assigned to each crew, and accordingly fails in fully benefiting from the learning curve phenomenon. It is, therefore, of practical value to consider such a phenomenon during scheduling of repetitive construction projects.

2.4.4 Weather Impact

Repetitive construction projects (e.g. pipelines, bridges, and highway construction) are often constructed in an outdoor environment. Weather variables on site are hard to control, and construction often has to proceed in cold or hot temperatures under different humidity conditions, exposed to wind and possible precipitation. Weather impact has been reported to be one of the main factors causing delay and cost overruns on construction projects (Baldwin et al. 1971, Koehn and Meilhede 1981, and Laufer and Cohenca 1990). In a survey conducted on Ohio contractors (Koehn and Meilhede, 1981), it was reported that winter construction causes an increase in project cost up to 60% which illustrates the significance of weather impact on construction productivity and cost. This was the reason that Johnston (1981) recommended that the variations in productivity rate due to weather should be incorporated into the schedule calculations of repetitive activities.

The impact of weather on construction activities can be in the form of reduced labor productivity and/or work stoppage. Reduced labor productivity effect is generally attributed to reduced human performance due to heat or cold stresses resulting from the combined effect of temperature, humidity and wind velocity. Weather related work stoppage is attributed to the inability of construction crews to work under severe weather conditions of heavy rain, snow and/or gusting wind. In attempts to identify the impact of weather in the form of reduced labor productivity, a number of studies have been conducted to establish the relation between labor productivity and weather conditions for a number of construction trades including: electrical work (National Electrical Contractors Association 1974), masonry (Grimm and Wagner 1974, and Sanders and Thomas 1991), equipment and manual tasks (U.S. Army Cold Regions Research and Engineering Laboratory 1986), and general construction (Koehn and Brown 1985).

The National Electrical Contractors Association conducted a study in 1974 to measure the effect of temperature on productivity (Effect 1974). The study was performed in an environmental chamber where temperature, humidity, and air velocity were monitored and controlled. The performance of simple tasks was studied under various weather conditions. The study established a wide range of productivity factors for electricians, considering the impact of effective temperature and humidity as shown in Table 2.2. Effective temperature is used to account for the cooling effect of wind, and is calculated based on actual temperature reading in °F and wind speed in mph as shown in Table 2.3.

In 1974, Grimm and Wagner studied weather impact on mason productivity over a period of nine months. During the construction of 283 test wall panels, the productivity of 51 masons were measured on site and correlated with temperature and humidity. The study provided a chart depicting the impact of temperature and humidity on mason productivity as shown in Figure 2.6. In a research conducted in Hanover, N.H. by the U.S. Army Cold Regions Research and Engineering Laboratory (1986), the effect of temperature on construction productivity of both manual and equipment construction tasks was reported. An upper and lower productivity levels were identified as shown in Figure 2.7. In another report provided by the Department of the Navy (1969), a sample of data relating temperature, relative humidity and wind speed to productivity factor of underground pipeline construction was presented (see Table 2.4).

In another study, to quantify climatic effects on construction, by Koehn and Brown

(1985), historic data of different construction crafts were utilized to derive two nonlinear relationships relating construction productivity in general to temperature and humidity. The two formulas were derived after performing regression analysis for a diverse mixture of 172 historical data points which were originally obtained for different construction trades (e.g. excavation, masonry, electrical, carpentry, and labor). Each of the two formulas is applied for a certain temperature range, the first is for cold weather (from -20 to 50 F) and the second for hot weather (from 70 to 120 °F). The productivity factor is considered unity for the temperature range from 50 to 70 °F. The two equations representing the productivity factor of general construction for both cold and hot weather are:

$$PF_c = 0.0144(T) - 0.00313(H) - 0.000107(T^2) - 0.000029(H^2) - 0.0000357(T)(H) + 0.674 \tag{2.4}$$

$$PF_h = 0.0517(T) - 0.0173(H) - 0.00032(T^2) - 0.0000985(H^2) - 0.0000911(T)(H) - 1.459 \tag{2.5}$$

where,

PF_c:	productivity factor of general construction for cold weather;
PF_h:	productivity factor of general construction for hot weather;
T:	temperature in °F;
and H:	relative humidity in percentage.

It must be noted that before applying the above two equations, they must be normalized as a function of their respective maximum values to represent productivity factor as a percent of maximum productivity.

In other studies aimed at considering the impact of weather on work stoppage, Cantwell (1987) established the daily rainfall thresholds that causes the work to stop for the whole day. Thresholds were provided for seven activities based on two surveys of contractors and visual observations as shown in Table 2.5. In the same study, it was assumed that if the daily rainfall exceeds the threshold, the activity is stopped for the whole day. In another study by Smith and Hancher (1989), the threshold was assumed to be daily accumulation of 0.1 inch or more.

In other studies aimed at considering the impact of weather on scheduling construction projects, Laramee (1983) and Hegazy (1993) have proposed monthly productivity factors representing the impact of weather on construction productivity in general. For example, a monthly productivity factor of 0.7 is used for the month of January to indicate that the modified productivity rate due to weather is considered to be 70% of that achieved in ideal weather conditions. This provides a simplified approach to account for the impact of weather on scheduling construction projects.

Moselhi and Nicholas (1990) proposed a hybrid expert system for construction planning and scheduling that considers the impact of reduced labor productivity due to weather. The system schedules the project and determines the early start and finish dates of each activity. Based on these dates, predicted temperature and humidity are extracted from the database and applied to Koehn formulas (Equations 2.4 and 2.5) to calculate the productivity factor of weather sensitive activities. Based on these productivity factors, the system generates automatically a revised or "as-possible" schedule. This hybrid system provides a practical approach to quantify the impact of weather on construction scheduling, however it assumes that all construction activities are equally affected by the same weather conditions. It also does not consider the impact of weather related work stoppage of construction activities. In 1996, Wales and AbouRizk proposed a simulation model for construction projects that considers the impact of precipitation and temperature on the duration of construction activities. The model generates the occurrence of weather variables and modifies activity durations accordingly.

Table 2.2 Weather Impact on Productivity of Electricians (Effect 1974)

Relative humidity (%)	Effective temperature (deg. F)												
	-10	0	10	20	30	40	50	60	70	80	90	100	110
90	56	71	82	89	93	96	98	98	96	93	84	57	0
80	57	73	84	91	95	98	100	100	98	95	87	68	15
70	59	75	86	93	97	99	100	100	99	97	90	76	50
60	60	76	87	94	98	100	100	100	100	98	93	80	57
50	61	77	88	94	98	100	100	100	100	99	94	82	60
40	62	78	88	94	98	100	100	100	100	99	94	84	63
30	62	78	88	94	98	100	100	100	100	99	93	83	62
20	62	78	88	94	98	100	100	100	100	99	93	82	61

Table 2.3 Effective Temperature (Effect 1974)

Wind speed (mph)	Actual thermometer reading (deg. F)									
	50	40	30	20	10	0	-10	-20	-30	-40
Calm	50	40	30	20	10	0	-10	-20	-30	-40
5	48	37	27	16	6	-5	-15	-26	-36	-47
10	40	28	16	4	-9	-21	-33	-46	-58	-70
15	36	22	9	-5	-18	-36	-45	-58	-72	-85
20	32	18	4	-10	-25	-39	-53	-67	-82	-96
25	50	16	0	-15	-29	-44	-58	-74	-88	-104
30	28	13	-2	-18	-33	-48	-63	-79	-94	-109
35	27	11	-4	-20	-35	-49	-67	-82	-98	-113

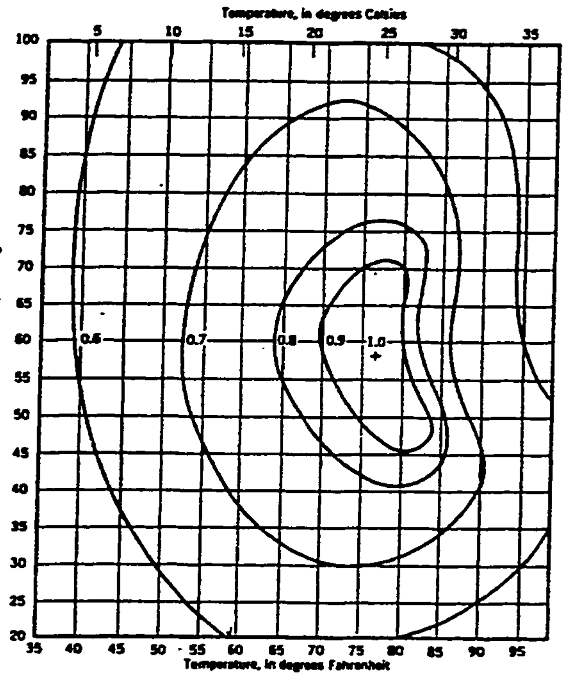


Figure 2.6 Weather Impact on Masons Productivity (Grimm & Wagner 1974)

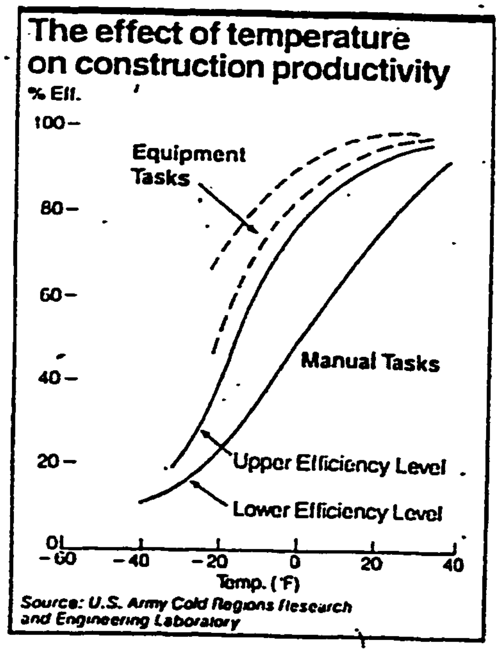


Figure 2.7 Weather Impact on Construction Productivity (U.S. Army 1986)

Table 2.4 Weather Impact on Piping (Department of Navy 1969)

Climatic Zone	Ambient Temperature (degree F)	Relative Humidity (%)	Wind Speed (mph)	Labor Efficiency (%)
Temperate	0	-	10	40-50
Temperate	30	20	20	60
Temperate	70	40	5	100
Temperate	90	50	9	60-70
Temperate	105	30	5	30
Tropical	75	60	5	90
Tropical	85	70	20	70
Tropical	92	78	5	50
Tropical	98	90	calm	30-40
Desert	87	20	calm	90
Desert	95	15	15	80
Desert	120	5	40	0-5
Desert	125	10	10	10
Desert	110	40	30	40
Arctic	40	30	10	90
Arctic	0	-	2	50-60
Arctic	-10	-	20	30
Arctic	-20	-	calm	30
Arctic	-40	-	40	0

Table 2.5 Impact of Rain on Work Stoppage (Cantwell 1987)

Activities	Threshold Value			
	Local Survey	Non-Local Survey	Visual Survey	Overall
Demolition	-	0.313"	-	0.30"
Foundations	0.80"	0.278"	1.06"	0.30"
U.G. services	-	0.278"	0.51"	0.30"
External walls	0.41"	0.164"	0.14"	0.15"
Floor slab	-	0.081"	-	0.10"
Roof steel	0.41"	0.147"	-	0.15"
Roof finish	-	0.082"	-	0.10"

CHAPTER 3

PROPOSED SCHEDULING MODEL

3.1 Introduction

This chapter presents the development of a computer model for optimized scheduling of repetitive construction projects. The model is designed embracing flexibility and practicality to ensure a wide range of possible applications to real-life projects. In an attempt to investigate the nature of the scheduling problem and identify various factors affecting it, a field study of a real-life repetitive construction project was conducted. The findings of both this study and a comprehensive literature review are carefully considered in the development of the scheduling model. The development of the model is based on object-oriented modelling which consists of three stages: analysis, design and implementation (Rumbaugh et al 1991). This chapter presents the analysis and design stages of the proposed scheduling model. The design stage is also covered in Chapters 4 and 5. The implementation stage of the proposed scheduling model is explained in Chapter 6.

3.2 Object-Oriented Modelling

In Object-Oriented Modelling, large and complex problems are decomposed and modelled as a set of objects. An object, in a model, often represents a real-world object such as: a physical object, concept or abstraction with crisp boundaries and meaning for the problem at hand (Rumbaugh et al 1991). An object encapsulates both data and functions. For example an object representing a construction activity encapsulates the required scheduling data (e.g. early start and finish dates) and scheduling functions (e.g. scheduling algorithm). A group of *objects* with similar properties (data), common behavior (functions) and common relationships to other objects can be grouped into a *class*. For example, all objects representing project activities such as excavation, foundation, concrete, etc. can be grouped into one class.

Object-Oriented Modelling utilizes several major concepts: abstraction, encapsulation, inheritance, and polymorphism (Rumbaugh et al 1991, and Weiner and Pinson 1988). Abstraction identifies the essential characteristics of an object that distinguish it from all other types of objects and thus provide crisply defined conceptual boundaries (Booch 1994). Encapsulation and data hiding are the centerpieces of object-oriented modelling, they bind data and associated functions tightly together. Both data and functions form a new entity: an object which is a variable declared to be of a specific class. Inheritance allows classes to be organized in a hierarchy, in which each class has an immediate parent or super-class, and each super-class can have several immediate children or sub-classes. If a number of sub-classes have some identical functions, it is not necessary to duplicate these functions for each sub-class. Instead, they can be defined for the super-class, and will automatically be inherited by all of the sub-classes (Powell et al, 1989).

In the literature, there are varying notations for graphical representation of the components of an object-oriented model (Booch 1994, Rumbaugh et al 1991, and Taylor 1992). For consistency, a notation system similar to that proposed by Rumbaugh et al (1991) is utilized in this thesis. As stated earlier, the development of an object-oriented model consists of three stages: analysis, design and implementation (Rumbaugh et al 1991). The analysis and design stages of the proposed scheduling model are described in the following sections.

3.3 Analysis

Analysis is the first stage in developing an object-oriented model. In this stage, a model of the real-life problem is built. The model should provide a precise abstraction of what is required to be done and how it can be done. The analysis stage outlines the model objects and their relationships and the sequence of operations that occur in the model. It produces: 1) an object model to outline the static structure of the model; and 2) a dynamic model to explain the sequence of operations (Rumbaugh et al 1991). The first step in the analysis stage is to provide a comprehensive understanding of the nature of the real-life problem. In an attempt to investigate the nature of repetitive scheduling and identify various factors affecting it, a field study of a real-life repetitive construction project was conducted.

3.3.1 Field Study

The project considered in this field study involved the construction of 35 housing units in East Montreal as shown in Figure 3.1. The objective of the study was to examine and monitor the day to day construction operations of a repetitive project and collect data relevant to construction scheduling. Site observations and relevant data were recorded using written notes, photos and videotapes. Over a period of two months during November and December 1992, discussions were conducted with various subcontractors and construction crews and the day to day construction activities of the housing project were observed including: excavation, foundations, form-work, concreting, wood framing, installation of prefabricated wooden roof trusses, installation of doors and windows, masonry, fiber-glass insulation, electrical rough-in and finishing, plumbing rough-in and finishing, gyproc wall, painting, carpeting and project main water and sewer connections.

The site observations that were found to be relevant to the development of a scheduling model for repetitive projects can be summarized as follows:

1) The housing project included repetitive as well as non-repetitive construction activities. For example, the wood framing activity was repeated in each house and can be considered a repetitive activity, while the main water and sewer connections were done only once for the entire project and can be considered a non-repetitive activity.
2) Repetitive construction activities can be either typical (i.e. activity durations are

considered identical in all houses) or non-typical (i.e. activity durations may vary from one housing unit to another). The variation of an activity duration from one housing unit to another can be attributed to variations in the quantities of work encountered in constructing different types of housing units as shown in Figure 3.1.

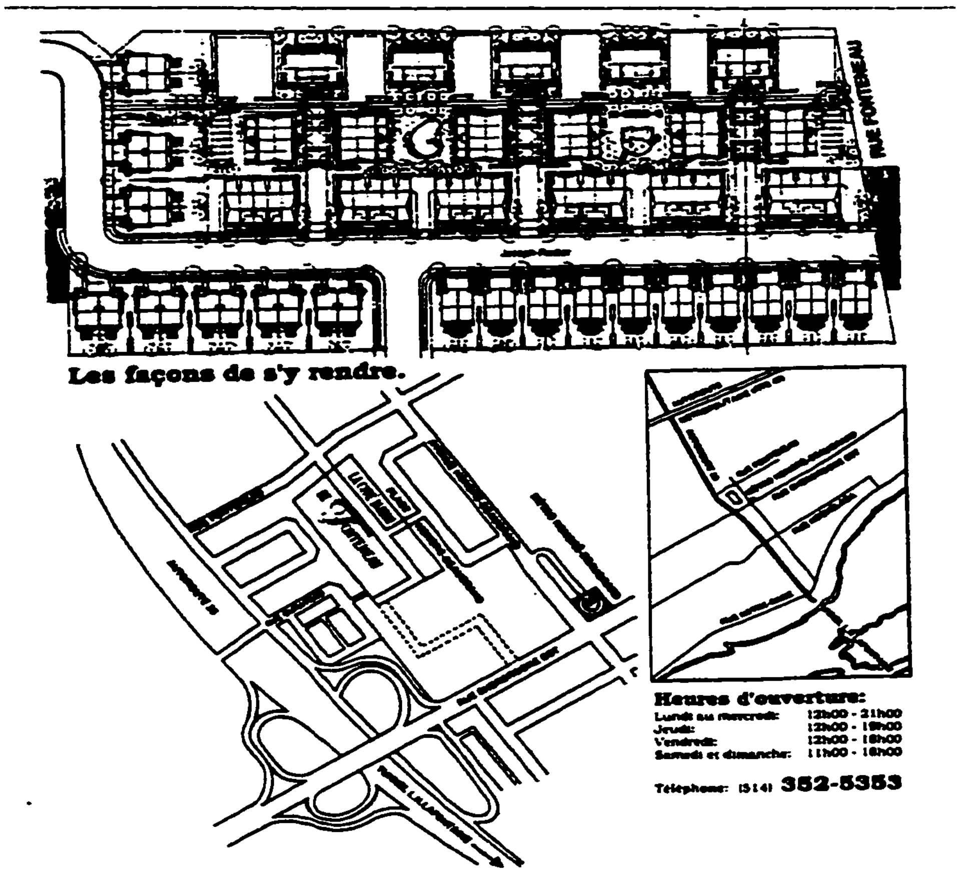

RUE FONTENEAU
Les façons de s'y rendre.
Heures d'ouverture:
Lundi au mercredi: 12h00 - 21h00
Jeudi: 12h00 - 19h00
Vendredi: 12h00 - 18h00
Samedi et dimanche: 11h00 - 18h00
Téléphone: (514) 352-5353

Figure 3.1 Montreal Housing Project

3) For a given repetitive construction activity (e.g. masonry), multiple crews are often utilized to work simultaneously on a number of houses.

4) Some construction crews cannot stay on site for the entire project duration. Due to out of site commitments, such crews can be assigned to work only

during a limited availability period on site.

5) The order of constructing the project houses (i.e. repetitive units) may vary from one repetitive activity to another.

6) Construction crews emphasized the significance of maintaining their work continuity once they move on to site. They prefer to finish work in one house and then be able to move promptly to the next without having to wait for another trade to complete its work.

7) During the month of December, the impact of cold weather on reducing crew productivity and increasing construction costs was evident, especially for construction crews that had to work outdoors such as masonry as shown in Figure 3.2.

8) In this housing project, construction crews repeated the same activity in a number of houses, moving from one to another. As such, a repetitive unit in this housing project can be considered a single house.

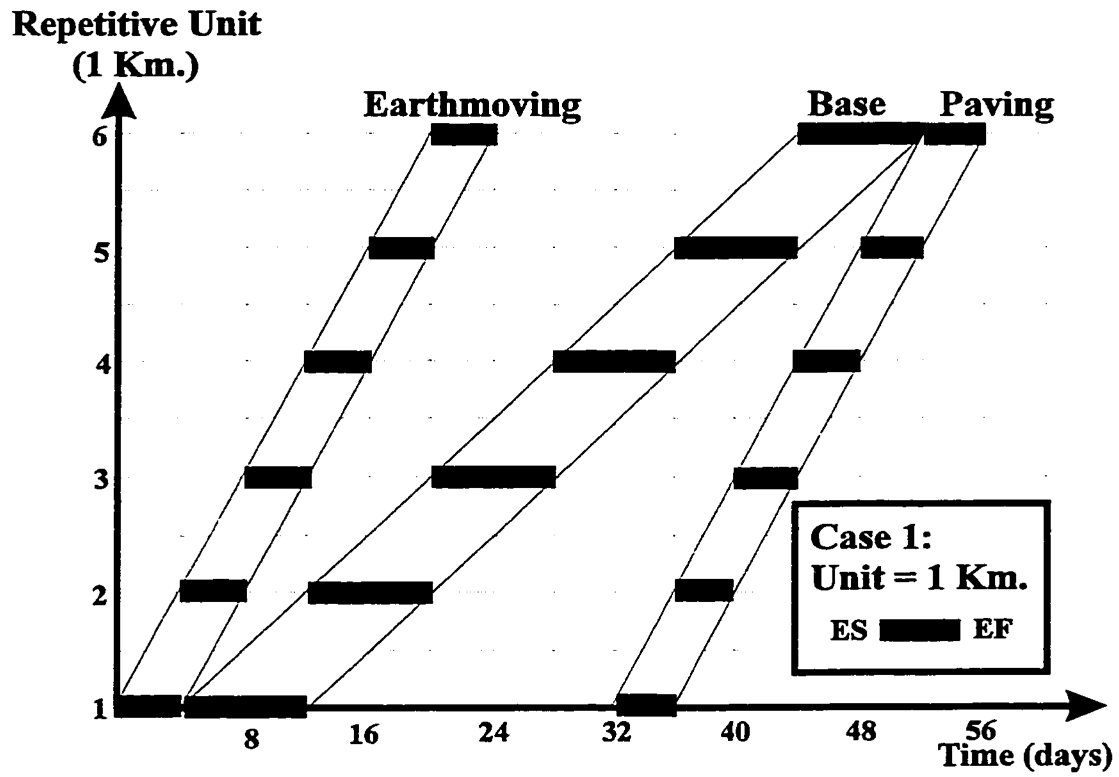
Figure 3.2 Winter Conditions on Site

3.3.2 Repetitive Unit

Repetitive construction projects consist of a number of repetitive units. A repetitive unit could simply be a house in a housing project, a specific length of a highway project or a typical floor in a high-rise building. As stated earlier in the field study, a repetitive unit in the observed housing project was considered to be a single house. Unlike housing projects, some repetitive projects do not have a unique definition of what constitutes a repetitive unit. In highway construction, for example, crews repeat the same activity and continuously move in a line along the entire length of the project. As such, there is no physical limits or space that clearly defines the size and scope of a repetitive unit which can be any 1 km. or any other length.

A primary consideration in identifying the scope or size of a repetitive unit is to prevent work interference among succeeding construction activities. In a highway construction project, for example, a crew assigned to the *base* activity may need at least 500 meters (i.e. ½ km.) of space separating it from the *earthmoving* crew of the preceding activity in order to avoid interference between the operations of the two working crews. In such a case, the scope of a repetitive unit can be identified to be a length of ½ km. of the highway project as shown in case 2 of Figure 3.3. This ensures that the *base* crew is not scheduled to start working in the first ½ km. of the project until after the *earthmoving* crew has completed all its work in the same ½ km, preventing interference between the two crews. In other repetitive construction projects, the scope of a repetitive unit can be identified similarly.

The impact of varying the scope of a repetitive unit on scheduling of repetitive construction can be illustrated using a simple example of a highway construction project. The example project involves the construction of a 6 km. stretch of a highway and can be broken down to three main repetitive activities: *earthmoving*, *base* and *paving*. The precedence relationship among the three activities is finish to start with no lag time. For each repetitive activity, a single crew is assigned to construct the entire project length, moving in a continuous line along the 6km. stretch of the highway. Figure 3.3 shows two different cases of repetitive unit scope and their impact on project scheduling. Case 1 of Figure 3.3 assumes that the 6km. stretch of the highway is divided into 6 repetitive units, each of a 1 km. length. Case 2 of the same figure, however, assumes that the same highway stretch is divided into 12 repetitive units, each of a ½ km. length. Although the number of repetitive units in case 1 is half that of case 2, the duration needed to construct any activity in a single repetitive unit in case 1 is double that of case 2. As a result, the overall activity duration required to construct the entire 6 km. for any of the three activities is unchanged in both cases as shown in Figure 3.3. This holds true for other unit sizes and other repetitive projects as well, indicating that the size of a repetitive unit does not affect the total duration of individual repetitive activities.



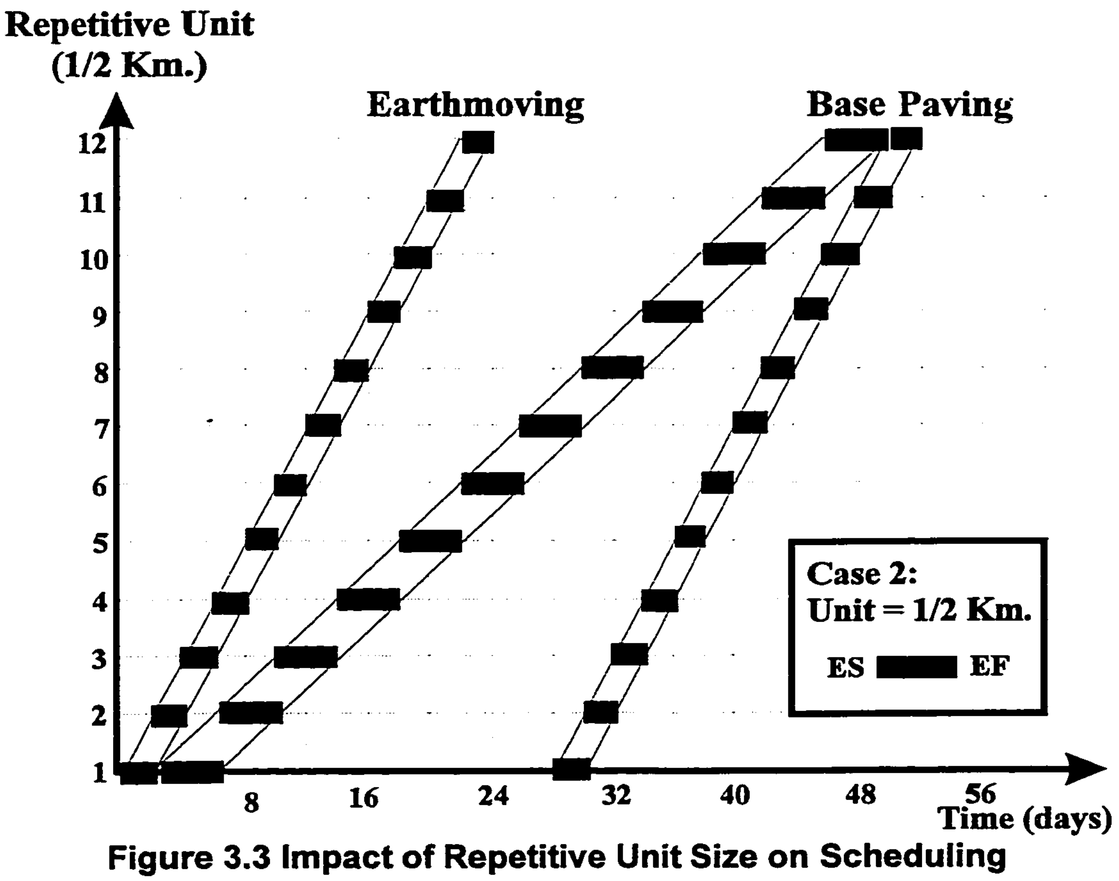


Figure 3.3 Impact of Repetitive Unit Size on Scheduling

As shown in Figure 3.3, the overall project duration in case 2 is slightly shorter than that of case 1, although the overall activity duration of each of the three activities is unchanged. The reason is that splitting the repetitive unit from 1 km. (case 1) to ½ km. (case 2) allowed the first unit of the second activity (*base*) to start earlier (i.e. after the completion of the *earthmoving* of ½ km. in case 2 rather than 1 km. in case 1). As a result, the base activity in the entire project started and finished earlier in case 2 compared to case 1 by a difference equal to the duration of earthmoving activity in ½ km. Similar situations apply in succeeding activities which result in a slightly shorter project overall duration.

As shown in Figure 3.3, the scope of a repetitive unit has no effect on the overall duration of individual repetitive activities, however it has a slight effect on the overall project duration. Reducing the size or scope of a repetitive unit results in a shorter overall project duration, however it increases the risk of work interference among succeeding activities. Therefore, it is recommended that the size of a repetitive unit be reduced as much as possible without causing interference among succeeding construction activities. It should be noted that identifying the scope of a repetitive unit is achieved among other things (e.g. work break-down structure) in the planning stage of a repetitive project. All commercially available scheduling systems (e.g. Primavera, Open Plan, etc.), albeit not for repetitive construction, are limited to project scheduling stage and do not address the planning stage. Similarly, the scope of the present model is limited to the scheduling stage of a repetitive project, however the above analysis was deemed necessary to provide the planner with a better understanding of what constitutes a repetitive unit.

3.3.3 Object Model

The purpose of an object model is to represent the structure of the model objects in terms of their data, functions and relationships to other objects. An object model should capture concepts from the real-world that are important to the application. For example, an object model for the scheduling of repetitive construction should identify objects that are essential to the scheduling process such as construction activities and relationships. The object model is represented graphically with a diagram to show the model classes, their hierarchy and association with other classes. As shown in Figure 3.4, the notation for a class is a box which may contain up to three sections. The top, middle and bottom sections of the box represent: class name, data and functions, respectively.

Inheritance is one of the main concepts in object-oriented modelling, and it allows for creating a hierarchy of classes. Such a hierarchy consists of a single overall super-class at the root, with sub-classes branching out into sub-classes. The highest levels of the hierarchy are the most general, while each lower level is more specialized than the one before it. Once a characteristic (i.e. data and/or function) is defined for a certain super-class, all the sub-classes beneath that definition inherit that characteristic. Each class in the hierarchy represents a certain level of specialization, and can inherit characteristics from more general parent classes. For example, both repetitive and non-repetitive construction activities can be modelled as two specialized sub-classes of a more general super-class representing construction activity. As such, the shared features of repetitive and non-repetitive activities can be abstracted in their super-class (i.e. construction activity) and inherited by the two sub-classes (i.e. repetitive and non-repetitive activities). As shown in Figure 3.4, the notation for inheritance is a triangle connecting a super-class at a higher level of the hierarchy to its sub-classes at a lower level.

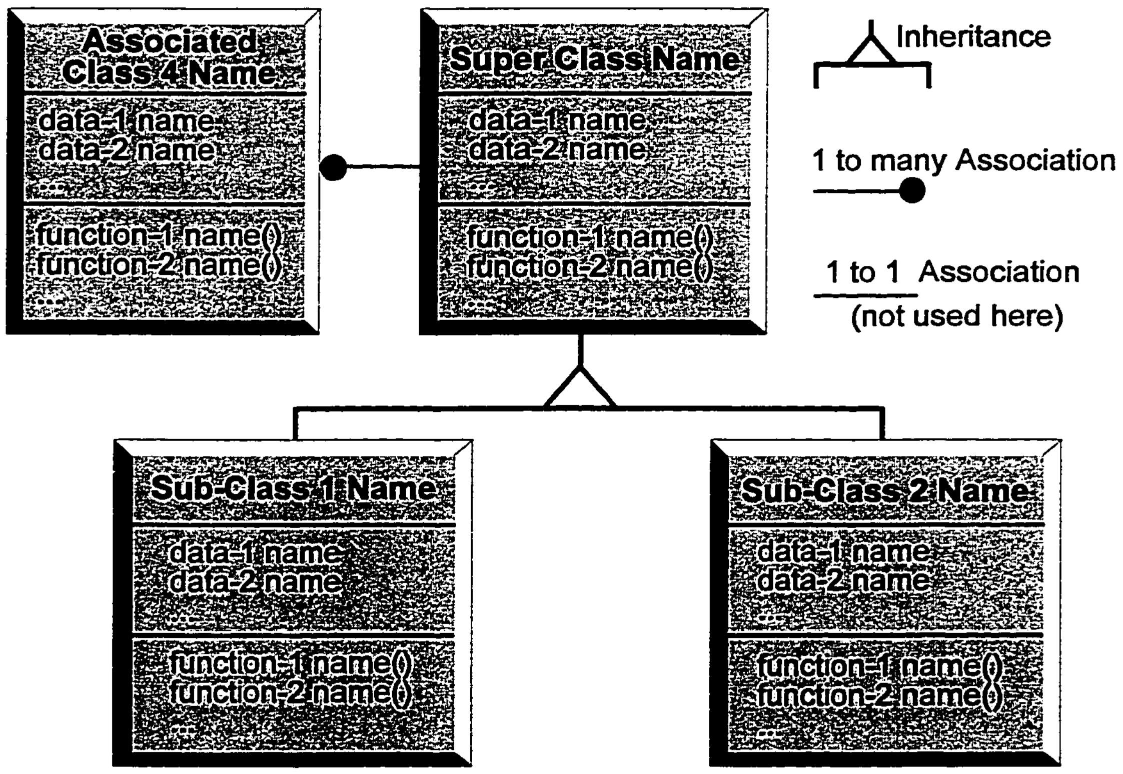


Figure 3.4 Class Representation

In the present scheduling model, an object model is developed to outline the classes of the model. The classes included in this object model are identified to facilitate the modelling of repetitive scheduling. The findings of both the field study and the literature review indicate that a repetitive construction project includes two types of activities: *Repetitive* and *Non-Repetitive* as shown in Figure 3.5. In order to account for different possibilities of generalized precedence relationships between these two types of activities, three different types of precedence relationships are designed in the present model. The first type is *Regular-Relation*, where the predecessor and successor activities are of the non-repetitive type as shown in Figure 3.5 (Relation a). The second type is *Repetitive-Relation* which defines the logical precedence relationship among two repetitive activities (Relation b). The third type is *Hetero-Relation* which can depict the relationship between either two specific units of two separate repetitive activities (Relation c), or two activities of different types (Relation d) (i.e. the predecessor is a *Non-Repetitive* activity and the successor is a specific unit of a *Repetitive* activity, or vice versa).

Hetero-Relation provides added flexibility and practicality to the present scheduling model. It is capable of representing a precedence relationship between two repetitive activities in two different units (Relation d), a needed feature in facilitating the modelling of construction requirements and job constraints in real-life projects. For example in a high-rise building project as the one shown in Figure 3.5, it is sometimes required that the drywall activity on a specific floor should not start until after the completion of two higher floors of the preceding glazing activity to ensure a weather-tight environment for drywall construction. In the present model, a *Hetero-Relation* can be used to specify such a relationship.

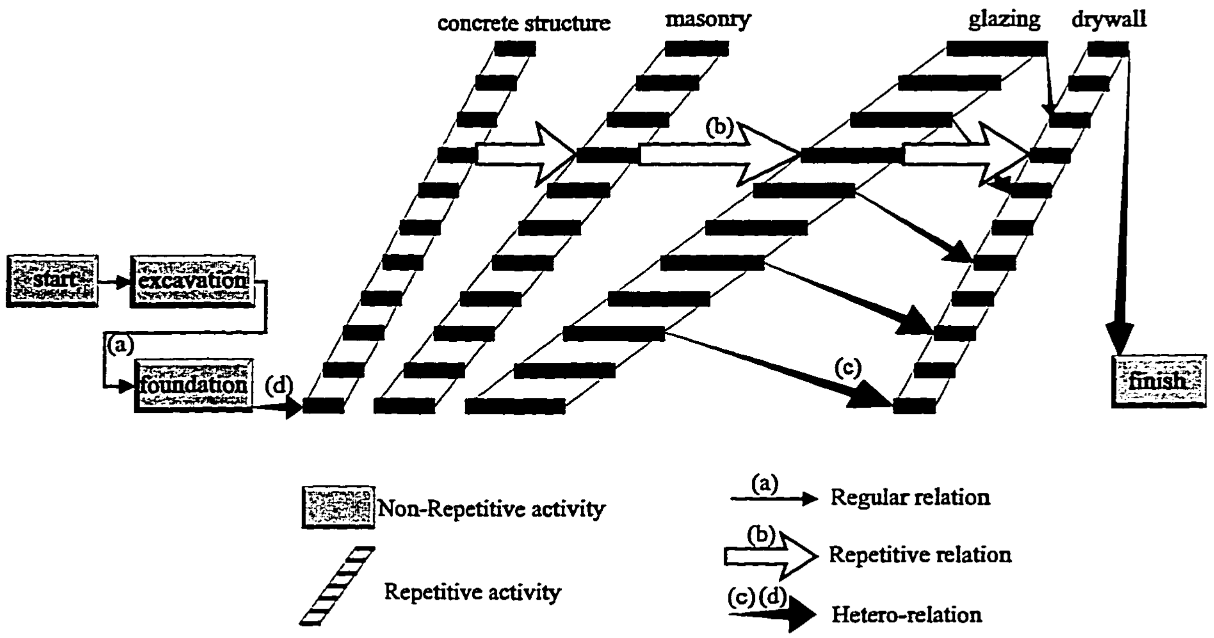
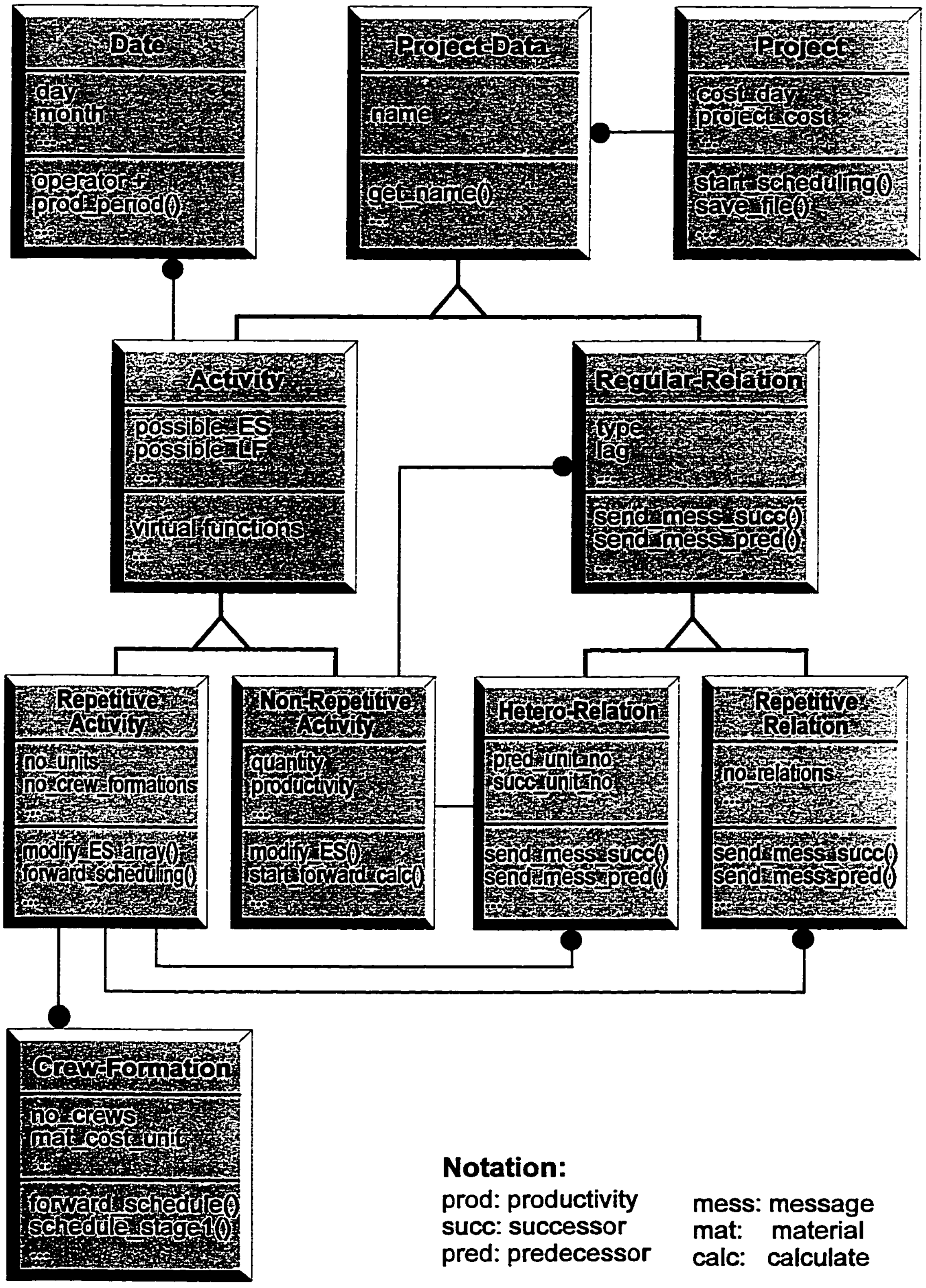


Figure 3.5 Activities and Relationships of a Repetitive Project

The present object model makes use of data encapsulation and inheritance concepts offered by object-oriented modelling in order to develop a properly derived and organized hierarchy of classes as shown in Figure 3.6. At the top of the hierarchy, there is the most generic class: *Project-Data*. This class includes only one character variable called "name" which is considered to be an identifier and a common data to all lower level classes or sub-classes in the hierarchy as shown in Figure 3.6. At the second level of the hierarchy, *Activity* and *Regular-Relation* classes are derived from the super-class *Project-Data*. At the third level of the hierarchy, *Non-repetitive* and *Repetitive* classes are derived from the super-class *Activity*, and *Repetitive-Relation* and *Hetero-Relation* classes are derived from the super-class *Regular-Relation*.

The lower level classes in the hierarchy represent an increased specialization while higher level ones represent more generalization. An *Activity* class can encapsulate the data (e.g. early start and finish dates) and their behavior (functions) in a general way as shown in Figure 3.6. Using inheritance, *Non-Repetitive* and *Repetitive* activity classes can inherit all the features of *Activity* class, and include additional specialized data and functions. As such, much of the coding of the super-class is reused in the derived sub-classes, avoiding duplication and benefiting from the shared data and functions among different classes.

In addition to the classes of the hierarchy, there are three classes that are included in the model and have associations with the hierarchy classes. The three classes are: *Project*, *Date*, and *Crew-Formation*. In addition to inheritance, there can be other types of association among classes of the same model. For example, each *Repetitive* activity object can have one or more associated *Crew-Formation* objects. As shown in Figure 3.6, the notation for such an association is a line connecting the two classes which may have a darkened circle at its ends indicating one or more associations. An association is often implemented in programming languages (e.g. C++) as a pointer from one class to another.



Notation:

prod: productivity	mess: message
succ: successor	mat: material
pred: predecessor	calc: calculate

Figure 3.6 Proposed Object Model

3.3.4 Dynamic Model

The object model, discussed in the previous section, describes the static structure of an object-oriented model. It shows the structure of objects and their relationships to one another at a single moment in time, however it does not show how the attributes of the objects change over time (Rumbaugh et al 1991). The objective of a dynamic model, on the other hand, is to describe the sequences of operations that occur to the objects of the model over time. The main concepts in a dynamic model are *states* and *messages or events*. A *state* of an object is the values of its attributes at one point in time. Over time, objects send *messages* to one another, resulting in a series of changes to their *states*. A *message* is a stimulus from one object to another, which may lead to changing the *state* of the receiving object, returning a *message* to the original sender object and/or sending a *message* to a third object. The pattern of *messages*, *states* and *state* transitions of an object can be represented by a *state diagram*. A dynamic model generally includes a number of *state diagrams*, each representing a specific class.

A state diagram relates messages and states, and can be represented by a graph whose nodes are states and whose directed arcs are messages or state transitions as shown in Figure 3.7. A state representing one object can receive a message from an external class as shown in Figure 3.7 (Rumbaugh et al 1991). In the proposed scheduling model, a dynamic model is formulated which includes a number of state diagrams. The state diagrams of the proposed model are explained later in the scheduling calculation (section 3.4.11).

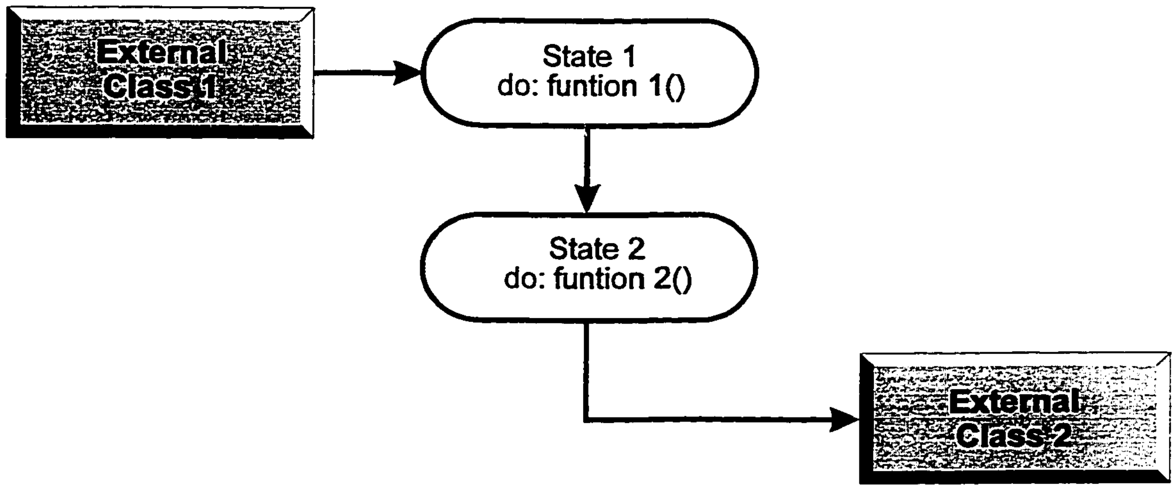


Figure 3.7 State Diagram

3.4 Design

Design is the second stage in developing an object-oriented model. It involves object design which expands and builds upon the outcome of the analysis stage to provide a detailed design for each object in the model. The design of an object entails design of data structures as well as functions or algorithms which are required to achieve the purpose of the object. As such, object design provides a detailed design of the data and functions encapsulated within each object of the model. In the proposed scheduling model, each class is designed to facilitate the modelling of repetitive scheduling. The design of each of the model classes (i.e. data members and functions) is briefly described in the following sections. For each class, this includes a brief discussion of the class, its data members, and its main member functions. A computer code outlining a more detailed list of data members and member functions for each class is included in Appendix I.

3.4.1 Project

Project class is designed to represent the characteristics of a repetitive construction project. This type of projects include different types of activities and relationships which are represented by various objects in the proposed scheduling model. In order to enable effective management of these objects, *Project* class should be able to perform the main functions of a search table (i.e. store, access, and retrieve various objects). There are different methods for implementing a search table. The simplest method is an array of objects. A second method involves constructing a linked list of objects. The third method is the most efficient and involves an advanced level of complexity, it provides a search table by creating a binary search tree (Weiner and Pinson, 1988, p. 106). This third method is utilized to design *Project* class which provides a flexible search tree that can include and manipulate various activity and relationship objects.

Project class is not limited to homogeneous objects only. In fact, it can include and manipulate all the heterogeneous objects included within in a repetitive project (i.e. *Repetitive* activity, *Non-Repetitive* activity, *Regular-Relation*, *Repetitive-Relation*, and *Hetero-Relation*). The utilization of inheritance and polymorphism in implementing a binary search tree for manipulating heterogeneous objects is discussed in more detail in (Weiner and Pinson, 1988, p. 170). A similar approach is used to design and develop *Project* class. To enable the inclusion of heterogeneous objects, *Project* class is composed of multiple nodes, each of which includes a pointer to a generic *Project-Data* object. This pointer can point to any specific object of the sub-classes of *Project-Data* class (e.g. *Repetitive-Activity* or *Regular-Relation*).

Project class is designed to perform a number of functions including: 1) determining the presence of an object in the project; 2) inserting a new object to the project; 3) sorting project activities; 4) initiating scheduling calculations; 4) displaying scheduling results; 5) saving project data to a binary file; and 6) opening a binary file and retrieving project data. The data members and main member functions of *Project* class are listed in Tables 3.1 and 3.2, respectively.

Table 3.1 Data Members of *Project* Class

Data	Data Type	Description
data	a pointer	a pointer to an object derived from Project-Data (e.g. Repetitive or Non-Repetitive activities).
cost_day	integer	project time value in $/day.
no_non_repetitives	integer	number of non-repetitive activities in the project.
no_repetitives	integer	number of repetitive activities in the project.
project_duration	float	project duration in days.
project_cost	float	project total cost in $.
project_start	a pointer	a pointer to a *Date* object including project start.
weather_sensitivity	integer	a variable indicating user-specified option for considering weather impact on scheduling.
learning_curve	integer	a variable indicating user-specified option for considering learning curve effect on scheduling.
optimization	integer	a variable indicating user-specified optimization option. Values of 0, 1 or 2 indicates no optimization, minimum duration optimization or minimum total cost optimization, respectively.

Table 3.2 Main Member Functions of *Project* Class

Function	Description
insert()	inserts a new object to the project
is_present()	determines the presence of an object in the project
sort()	sorts project activities
start_scheduling()	initiates project scheduling calculations
save_file()	saves project data to a binary file
open_file()	opens project data from a binary file

3.4.2 Date

Date class is designed to represent the calendar dates for the start and finish of different activities in the project. *Date* class is designed to perform a number of functions including: 1) determining the weekday for a calendar date; 2) adding an activity duration to a calendar date; 3) subtracting a duration from a calendar date; and 4) calculating productivity factor due to weather impact for a specific construction trade. The data members and main member functions of *Date* class are listed in Tables 3.3 and 3.4, respectively.

Table 3.3 Data Members of *Date* Class

Data	Data Type	Description
day	integer	calendar day of the month (i.e. 1 to 31)
month	integer	calendar month of the year (i.e. 1 to 12)
year	integer	calendar year (e.g. 1997)
weekday	string	calendar weekday (e.g. Monday)

Table 3.4 Main Member Functions of *Date* Class

Function	Description
operator +	adds an activity duration to a calendar date
prod_period()	estimates productivity factor due to weather for a specific construction trade over a given period (e.g. Sept. 12 to Oct. 3)

3.4.3 Project-Data

At the highest level of the model hierarchy is *Project-Data* class which is the most generic one as shown in Figure 3.6. This class serves as an abstract class at the top of the hierarchy, from which more specialized classes can be derived. It also forms an interface between all the classes of the hierarchy and the *Project* class. The data members and main member functions of *Project-Data* class are listed in Tables 3.5 and 3.6, respectively.

Table 3.5 Data Members of *Project-Data* Class

Data	Data Type	Description
name	string	an identifier and a common data member to all lower level classes in the hierarchy

Table 3.6 Main Member Functions of *Project-Data* Class

Function	Description
get_name()	returns the object name to an outside inquiring object

3.4.4 Regular-Relation

At the second level of the hierarchy, *Regular-Relation* class is derived from *Project-data* as shown in Figure 3.6. It is derived from the parent class (i.e. *Project-Data*) which entitles it to inherit all the data members of the parent class (i.e. name). In addition, *Regular-Relation* includes more specialized set of data and functions that models its own requirements and behaviour. This class is designed to represent the general precedence relationship among two non-repetitive activities. This relationship type can be either : finish to start (FS), start to start (SS), finish to finish (FF), or start to finish (SF) with or without lag time. The data members and main member functions of *Regular-Relation* class are listed in Tables 3.7 and 3.8, respectively.

Table 3.7 Data Members of *Regular-Relation* Class

Data	Data Type	Description
pred_name	string	name of predecessor activity
succ_name	string	name of successor activity
Apred	a pointer	a pointer to predecessor activity object
Asucc	a pointer	a pointer to successor activity object
type	integer	precedence relationship type (i.e. finish to start, start to start, finish to finish or start to finish)
lag	integer	lag time in working days

Table 3.8 Main Member Functions of *Regular-Relation* Class

Function	Description
send_mess_succ()	is invoked during forward pass scheduling by a message from its predecessor activity object. In response, this function sends a message to the successor activity object which includes early start and finish dates of the predecessor activity and relationship type and lag. This message also invokes the successor activity object to modify its early start date and start forward pass scheduling.
send_mess_pred()	is invoked during backward pass scheduling by a message from its successor activity object. In response, this function sends a message to the predecessor activity object which includes late start and finish dates of the successor activity and relationship type and lag. This message also invokes the predecessor activity object to modify its late finish date and start backward pass scheduling.

3.4.5 Repetitive-Relation

Repetitive-Relation represents the precedence relationship between two repetitive activities. It is derived from *Regular-Relation* at the third and the lowest level of the hierarchy. In addition to the data and function members inherited from the parent class, *Repetitive-Relation* includes the data member listed in Table 3.9. The member functions of *Repetitive-Relation* are similar to those of *Regular-Relation*, however they exchange messages with *Repetitive* activity objects.

Table 3.9 Data Members of Repetitive-Relation Class

Data	Data Type	Description
no_relations	integer	number of repetitive units linked in the two successive construction activities by this relation

3.4.6 Hetero-Relation

Hetero-Relation represents a precedence relationship between either two different activity types or two activities at different repetitive units to provide added flexibility to the scheduling model as previously mentioned in section 3.3.3. This class is derived from *Regular Relation* at the third level of the hierarchy, and it includes two additional data members as shown in Table 3.10.

Table 3.10 Data Members of Hetero-Relation Class

Data	Data Type	Description
pred_unit_no	integer	identifying number of a predecessor repetitive unit
succ_unit_no	integer	identifying number of a successor repetitive unit

It should be noted that one of these two variables is automatically initiated with zero if its activity is specified by the user to be non-repetitive. Otherwise, these variables are initiated with the appropriate repetitive unit number as specified by the user. The member functions of *Hetero-Relation* are similar to those of *Regular-Relation*, however they exchange messages with two types of activity objects (i.e. *Non-Repetitive* and *Repetitive*).

3.4.7 Activity

In the other branch of the hierarchy, *Activity* class is derived from the parent class (*Project-Data*) which entitles it to inherit all the data members of the parent class (i.e. name[10]). In addition, *Activity* includes more specialized set of data that models its own requirements and behaviour as shown in Table 3.11. It should be noted that this class does not include member functions of its own, however it includes the generic data which applies to both repetitive and non-repetitive activities. It forms an intermediate level class, from which more detailed and specialized classes can be derived (i.e. *Non-Repetitive* and *Repetitive*).

Table 3.11 Data Members of *Activity* Class

Data	Data Type	Description
Rpred	a pointer	a pointer to a predecessor relationship object
Rsucc	a pointer	a pointer to a successor relationship object
no_pred	integer	number of predecessor non-repetitive activities
no_succ	integer	number of successor non-repetitive activities
mess_pred	integer	a counter indicating number of messages received from predecessor non-repetitive activities
mess_succ	integer	a counter indicating number of messages received from successor non-repetitive activities
possible_ES	integer	early possible start date of activity
possible_LF	integer	late possible finish date of activity
actual_start	integer	actual start date of activity
actual_finish	integer	actual finish date of activity
weather_type	integer	a variable indicating activity weather sensitivity. Values of 0, 1, 2, 3, 4, 5 or 6 indicates insensitive to weather, masonry, electrical, labour tasks, manual tasks, underground piping or general construction, respectively

3.4.8 Non-Repetitive Activity

At the third and the lowest level of the hierarchy, *Non-Repetitive* Activity class is derived from *Activity*. This class is designed to represent the characteristics of non-repetitive activities of the project. The data members and main member functions of *Non-Repetitive* class are listed in Tables 3.12 and 3.13, respectively.

Table 3.12 Data Members of *Non-Repetitive* Class

Data	Data Type	Description
quantity	float	quantity of work in units of measurement
productivity	float	crew daily output in units/day
prod_factor	float	productivity factor due to weather impact
mat_cost_unit	float	material cost rate in $/unit
lab_cost_day	float	labour cost rate in $/day
equip_cost_day	float	equipment cost rate in $/day

Table 3.13 Main Member Functions of *Non-Repetitive* Class

Function	Description
modify_ES()	is invoked during forward pass scheduling by a message from its predecessor relationship object. The message includes predecessor early start and finish dates, and precedence relationship type and lag. In response to the message, this function modifies the possible_ES data member of the activity object
modify_LF()	is invoked during backward pass scheduling by a message from its successor relationship object. The message includes successor late start and finish dates, and precedence relationship type and lag. In response to the message, this function modifies the possible_LF data member of the activity object
start_forward_ calculations()	is invoked by a message from its predecessor relationship object. It calculates activity duration and sends message to successor relationship objects containing information about scheduled early start and finish dates of the activity
start_backward_ calculations()	is invoked by a message from its successor relationship object. It calculates late start date and sends message to predecessor relationship objects containing information about scheduled late start and finish dates of the activity
display_calendar_ dates()	displays activity scheduling results

3.4.9 Repetitive Activity

This class represents the special characteristics of the repetitive activities in the project. The data and functions of this class are designed to consider factors affecting scheduling of repetitive activities. At the third level of the hierarchy, this class is derived from *Activity* class. In addition to the data and function members inherited from its parent classes, *Repetitive-Activity* includes data members listed in Table 3.14. The main member functions of this class are listed in Table 3.15.

Table 3.14 Data Members of *Repetitive* Class

Data	Data Type	Description
no_units	integer	number of repetitive units
no_crew_ formations	integer	number of available crew formations
chosen_formation	integer	identification number of the selected optimum crew formation
no_rep_pred	integer	number of predecessor repetitive activities
no_rep_succ	integer	number of successor repetitive activities
mess_rep_pred	integer	a counter indicating number of messages received from predecessor repetitive activities
mess_rep_succ	integer	a counter indicating number of messages received from successor repetitive activities
quantity	a pointer	a pointer to a float vector of size [no_units] that includes quantities of work of all repetitive units
interruption_vector	a pointer	a pointer to an integer vector of size [no_units] that includes interruption to crew work continuity in all repetitive units
execution_order	a pointer	a pointer to an integer vector of size [no_units] that includes user-specified order of execution throughout all repetitive units
PES	a pointer	a pointer to an integer vector of size [no_units] that includes possible early start date of all repetitive units
Rep_pred	a pointer	a pointer to predecessor Repetitive-Relationship objects
Rep_succ	a pointer	a pointer to successor Repetitive-Relationship objects
crew_formations	a pointer	a pointer to available Crew-Formation objects

Table 3.15 Main Member Functions of *Repetitive* Class

Function	Description
modify_ES_array()	is invoked during forward pass scheduling by a message from its predecessor relationship object. The message includes predecessor early start and finish dates, and precedence relationship type and lag. In response to the message, this function modifies the PES vector.
start_forward_ calculations()	is invoked by a message from its predecessor relationship object. In response, the function sends a message to the Project object inquiring about user-specified optimization option. If the user specifies regular scheduling with no optimization, the function sends a message to Crew-Formation object invoking it to apply an algorithm for resource-driven scheduling (described later in Chapter 4).
start_forward_ optimization()	is invoked to apply an algorithm for schedule optimization algorithm (described later in Chapter 5).

3.4.10 Crew-Formation

Often, there are more than one crew utilization option that can be used to construct a *Repetitive* activity. For example, a masonry activity in a housing project can be constructed using: 1) a single crew without overtime hours; 2) a single crew with overtime hours or; 3) two crews working simultaneously in different houses without overtime hours. Each of these crew utilization options can be identified as a unique *crew formation* to construct the activity. In this thesis, the word "*crew formation*" is used to describe a crew utilization option that may involve utilizing one or more construction crews with or without overtime policy. This class is designed to represent the characteristics of a "*crew formation*" to enable schedule optimization which is discussed later in Chapter 5. The data members and main member functions of *Crew-Formation* class are listed in Tables 3.16 and 3.17, respectively.

Table 3.16 Data Members of *Crew-Formation* Class

Data	Data Type	Description
no_crews	integer	number of crews that can work simultaneously on an activity in different repetitive units
productivity_ normal	a pointer	a pointer to a float vector of size [no_crews] that includes daily output of each crew in units/day
material_cost_ unit	a pointer	a pointer to an integer vector of size [no_crews] that includes material cost rate of each crew in $/unit
labor_cost_day	a pointer	a pointer to an integer vector of size [no_crews] that includes labor cost rate of each crew in $/day
equipment_cost _day	a pointer	a pointer to an integer vector of size [no_crews] that includes equipment cost rate of each crew in $/day
interuption_cost _day	a pointer	a pointer to an integer vector of size [no_crews] that includes interruption cost rate of each crew in $/day
early_available_ date	a pointer	a pointer to an integer vector of size [no_crews] that includes early available date of each crew on site
late_available_d ate	a pointer	a pointer to an integer vector of size [no_crews] that includes late available date of each crew on site
interruption_OK	a pointer	a pointer to an integer vector of size [no_crews] that indicates user-specification whether the work continuity of each crew can be interrupted or not
productivity_ factor	a pointer	a pointer to a float vector of size [no_units] that includes productivity factor due to weather and/or learning curve effect for each repetitive unit
assigned_crew	a pointer	a pointer to an integer vector of size [no_units] that includes identification number of the selected crew for each repetitive unit
duration	a pointer	a pointer to an integer vector of size [no_units] that includes calculated duration for each repetitive unit
ES_array	a pointer	a pointer to an integer vector of size [no_units] that includes calculated early start date of each repetitive unit
LF_array	a pointer	a pointer to an integer vector of size [no_units] that includes calculated late finish date of each repetitive unit

Table 3.17 Main Member Functions of *Crew-Formation* Class

Function	Description
forward_schedule()	is invoked during forward pass scheduling by a message from the associated repetitive activity object. The function applies a developed algorithm for resource-driven scheduling (described later in Chapter 4).
Calculate_interruption()	is invoked during scheduling optimization to automatically generate a feasible set of interruption options. These options are used to identify the optimum interruption option for the crew formation based on a developed algorithm for scheduling optimization (described later in Chapter 5).

3.4.11 Scheduling Calculation

In the proposed model, scheduling calculation is performed by exchanging messages among various objects in the model. An overview of the scheduling calculation is illustrated graphically in a number of state diagrams (Figures 3.8 to 3.13). The state diagram of a specific class illustrates the messages received from and/or sent to external classes and how they invoke various member functions of the local class as shown in Figures 3.8 to 3.13.

Scheduling calculation is invoked by the user after completing the data input for all project activities and relationships. Upon user request, the project object starts scheduling calculation by searching for an activity that has no predecessors as shown in Figure 3.8. Such an activity represents the start activity of the project and it can be repetitive or non-repetitive. The project object then sends a message to the start activity object, invoking it to modify its early start. In response, that activity initiates its possible early start variable with the start date of the project. The project object then sends a second message to the same activity object invoking it to start forward pass calculations as shown in Figure 3.8. The activity, accordingly, examines if it has received messages from all its predecessors. If this condition is not true, the activity waits until receiving other messages from the rest of its predecessors. Otherwise, the activity object starts self-scheduling based on its type as shown in the two state diagrams of *Repetitive* or *Non-Repetitive* activity classes (Figures 3.9 and 3.10).

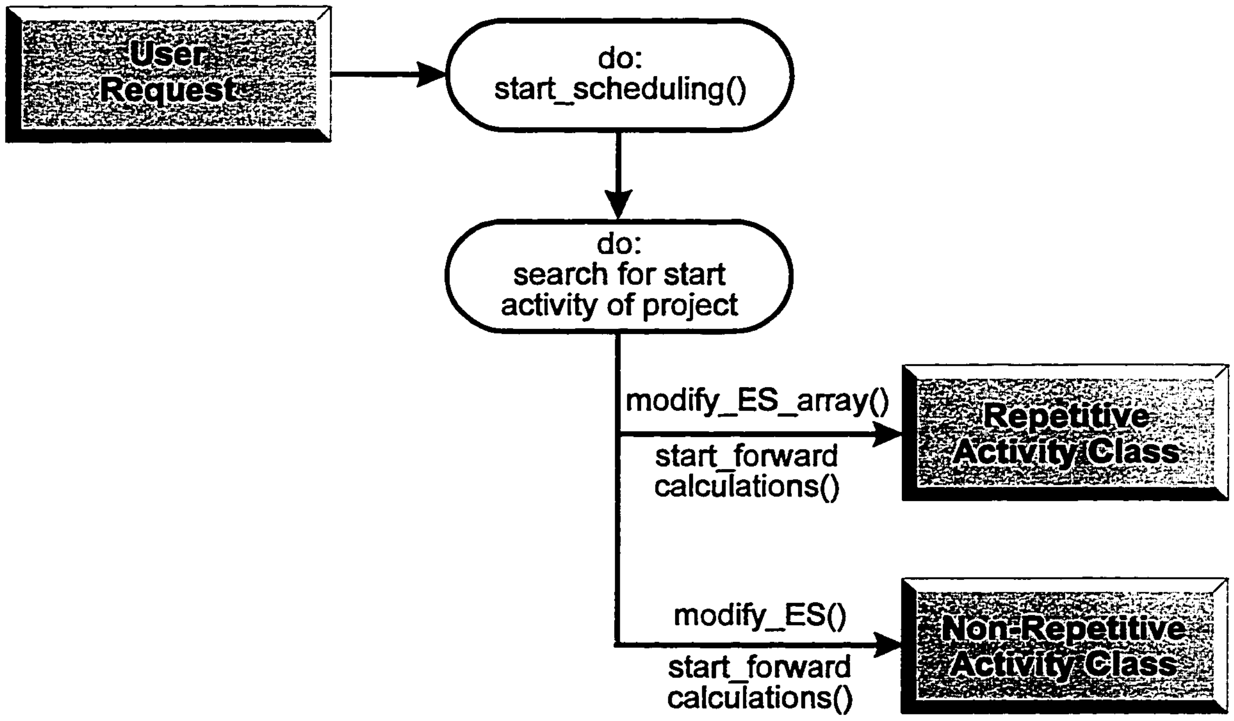

User Request
do: start_scheduling()
do: search for start activity of project
modify_ES_array()
start_forward calculations()
Repetitive Activity Class
modify_ES()
start_forward calculations()
Non-Repetitive Activity Class

Figure 3.8 State Diagram of *Project*

For a Non-Repetitive activity object, the process of self-scheduling starts by calculating the initial duration (D) without considering weather impact in working days as follows:

$$D = Q / P \tag{3.1}$$

where,

Q : quantity of work in units of measurement;

and P : crew daily output in units/day.

The initial early finish date (IEF) of the activity is then calculated by adding the initial duration obtained from equation (3.1) to the early start date (ES) of the activity as follows:

$$IEF = ES + D \tag{3.2}$$

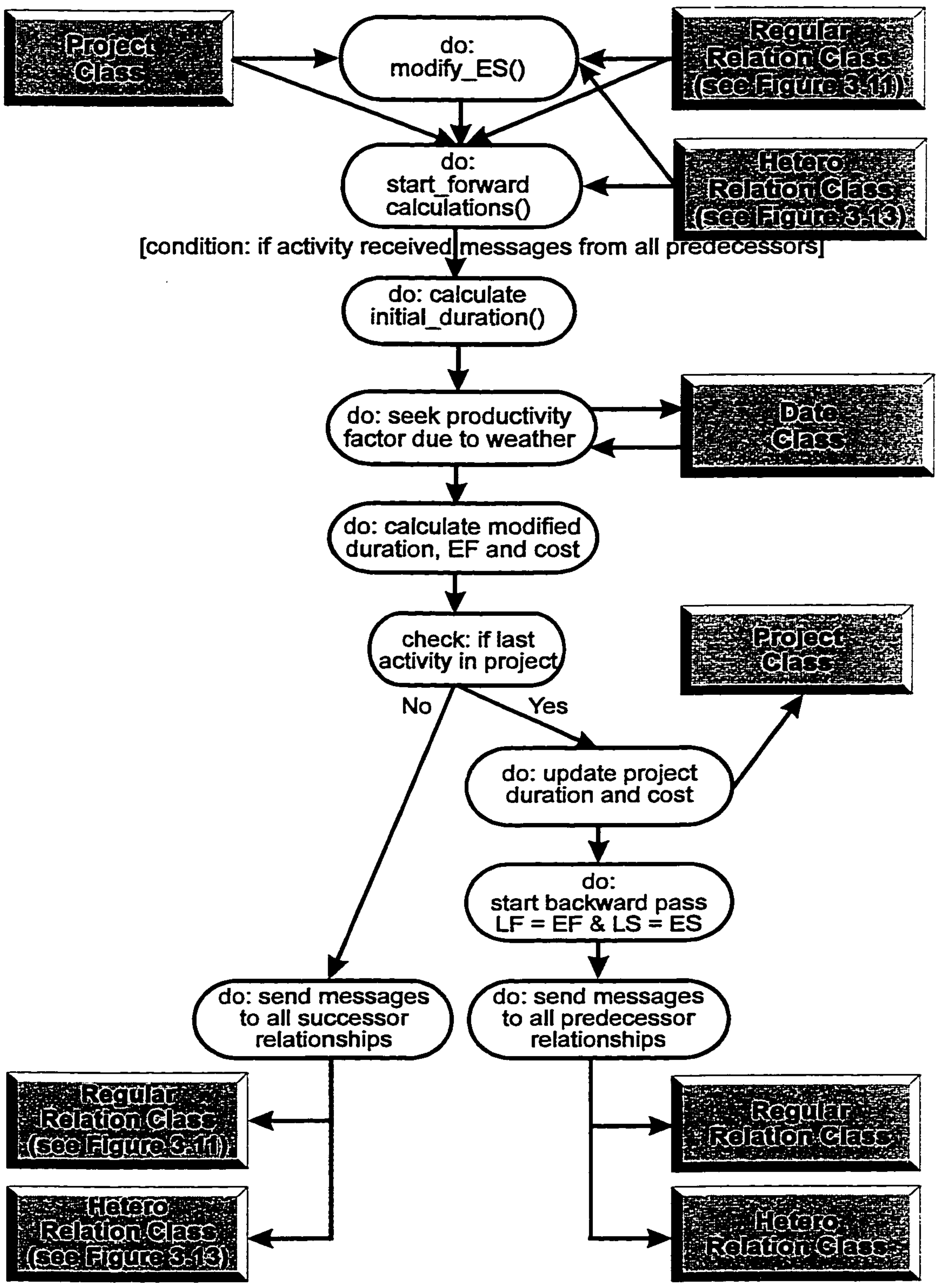
If the user has specified the activity to be weather-sensitive, the activity object then starts to modify its duration and costs due to the impact of weather. This is achieved by sending a message to a *Date* object, seeking an average productivity factor due to weather as shown in Figure 3.9. The message includes the type of activity (e.g. masonry, electrical, etc.) and the initial start and finish dates of the activity (e.g. ES = October 12 and IEF = November 20). In response to this message, the *Date* object calculates an average productivity factor due to weather for the specified construction period and returns it to the inquiring *Non-Repetitive* activity object. The calculation of this average productivity factor by a member function of the *Date* class is explained later in section 3.4.12. Upon receiving the average productivity factor from the *Date* object, the Non-Repetitive activity object proceeds to modify its duration and cost. It calculates its modified duration (MD), its modified early finish date (EF) and its direct cost in $ (DC) as follows:

$$MD = D / PFW \tag{3.3}$$
$$EF = ES + MD \tag{3.4}$$
$$DC = Q \times MC + MD \times (LC + EC) \tag{3.5}$$

where,

PFW:	average productivity factor received from *Date* class message;
MC:	material cost rate in $/unit of measurement;
LC:	labor cost rate in $/day;
and EC:	equipment cost rate in $/day.

For a *Repetitive* activity object, the process of self-scheduling starts by examining the user-specified option for schedule optimization as shown in Figure 3.10. The activity object sends a message to the *Project* object inquiring about the schedule optimization option. If the user specifies schedule optimization, the activity object applies an optimization algorithm which is described later in chapter 5. Otherwise, the activity object proceeds with forward pass scheduling to identify the early start and finish dates of the activity in each repetitive unit as shown in Figure 3.10. The forward pass scheduling is initiated by sending a message to the *Crew-Formation* object specified by the user to construct the activity as shown in Figure 3.10. In response to this message, the Crew-Formation object identifies early start and finish dates of the activity in each repetitive unit by applying the resource-driven scheduling algorithm presented in Chapter 4.



Project Class
do: modify_ES()
Regular Relation Class (see Figure 3.11)
do: start_forward calculations()
Hetero Relation Class (see Figure 3.13)
[condition: if activity received messages from all predecessors]
do: calculate initial_duration()
do: seek productivity factor due to weather
Date Class
do: calculate modified duration, EF and cost
check: if last activity in project
Project Class
No
Yes
do: update project duration and cost
do: start backward pass LF = EF & LS = ES
do: send messages to all successor relationships
do: send messages to all predecessor relationships
Regular Relation Class (see Figure 3.11)
Hetero Relation Class (see Figure 3.13)
Regular Relation Class
Hetero Relation Class

Figure 3.9 State Diagram of *Non-Repetitive* Activity

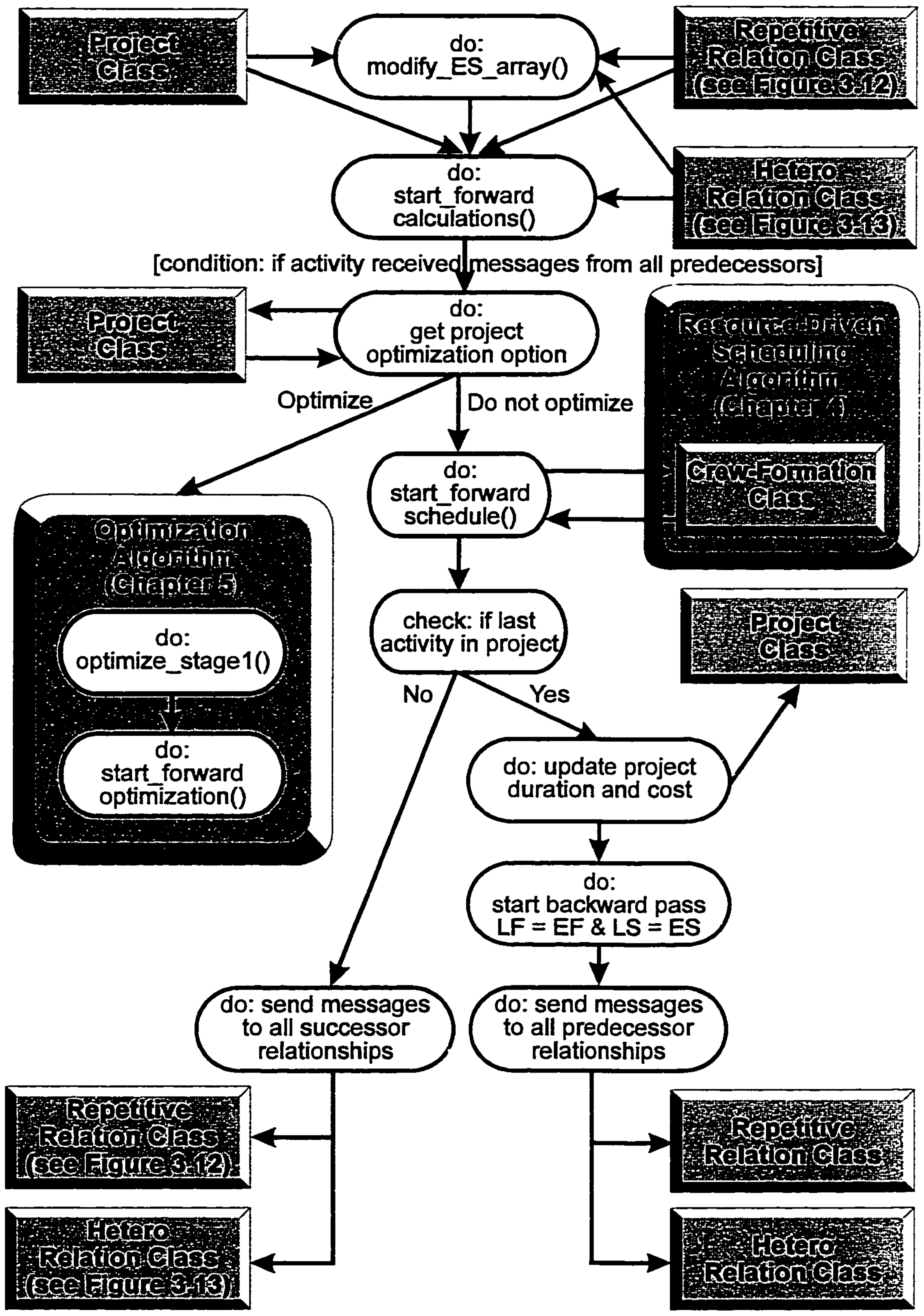

Project Class
do: modify_ES_array()
Repetitive Relation Class (see Figure 3-12)
do: start_forward calculations()
Hetero Relation Class (see Figure 3-13)
[condition: if activity received messages from all predecessors]
Project Class
do: get project optimization option
Resource-Driven Scheduling Algorithm (Chapter 4)
Optimize
Do not optimize
do: start_forward schedule()
Crew-Formation Class
Optimization Algorithm (Chapter 5)
do: optimize_stage1()
do: start_forward optimization()
check: if last activity in project
Project Class
No
Yes
do: update project duration and cost
do: start backward pass LF = EF & LS = ES
do: send messages to all successor relationships
do: send messages to all predecessor relationships
Repetitive Relation Class (see Figure 3-12)
Hetero Relation Class (see Figure 3-13)
Repetitive Relation Class
Hetero Relation Class

Figure 3.10 State Diagram of *Repetitive* Activity

Upon the completion of self-scheduling, each activity object (which can be *Repetitive* or *Non-Repetitive*) examines if it is the last activity in the project as shown in Figures 3.9 and 3.10. If this condition is true, the activity object sends a message to the Project object to update the project completion date, and then the activity starts the backward pass calculation as shown in Figures 3.9 and 3.10. Otherwise, the activity object sends messages to all the successor relationship objects informing them with its early start and finish dates. In response to these messages, each relationship object passes this information along with the relationship type and lag in working days to the successor activity object in the form of a message as shown in Figures 3.11, 3.12 and 3.13. Based on this information, the successor activity object repeats the above described process of self-scheduling.

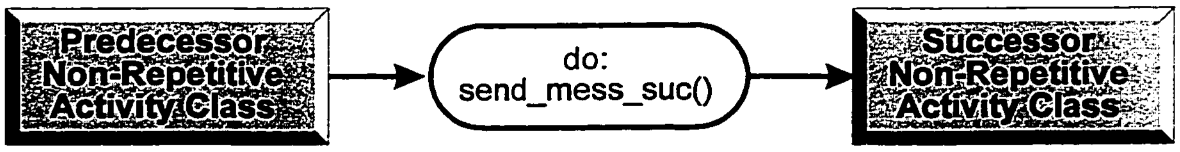

Predecessor Non-Repetitive Activity Class
do: send_mess_suc()
Successor Non-Repetitive Activity Class

Figure 3.11 State Diagram of *Regular-Relation*

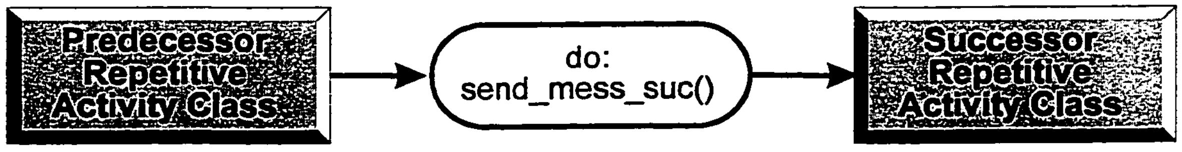

Predecessor Repetitive Activity Class
do: send_mess_suc()
Successor Repetitive Activity Class

Figure 3.12 State Diagram of *Repetitive-Relation*

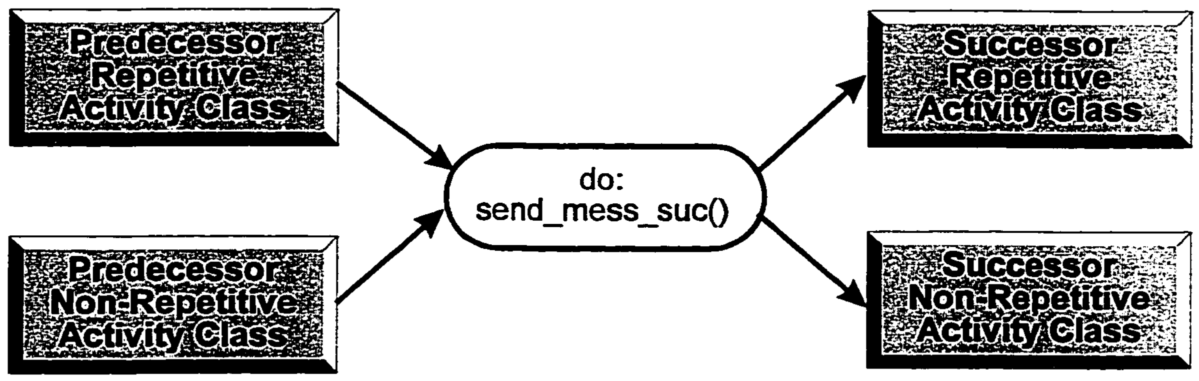

Predecessor Repetitive Activity Class
Predecessor Non-Repetitive Activity Class
do: send_mess_suc()
Successor Repetitive Activity Class
Successor Non-Repetitive Activity Class

Figure 3.13 State Diagram of *Hetero-Relation*

This process of sending and receiving messages among activities and their relationship objects propagates throughout the whole project starting with the first and ending with the last activity. As such, the forward pass calculation is carried out, and upon its completion an opposite direction process will automatically start from the last activity and propagates through to the first activity in the project. This backward pass calculation is performed to identify late start and finish dates for each activity in the project. In the scheduling model, scheduling of the non-repetitive group of activities is carried out in a process similar to network-based techniques but through an exchange of messages. Scheduling of the repetitive group of activities, however, is based on a developed resource-driven scheduling algorithm which is explained in the following chapter. As such, the model is capable of integrating scheduling techniques for non-repetitive and repetitive activities.

3.4.12 Impact of Weather on Construction Productivity

The impact of weather on construction productivity in the proposed model is identified in *Date* class. As shown in Figure 3.14, a *Date* object may receive a message from a *Non-Repetitive* activity object or from a *Crew-Formation* object inquiring about the impact of weather on construction duration. The message includes the type of construction (e.g. masonry) and the scheduled construction period (e.g. November 15 to December 6). In response to this message, the *Date* object estimates the average productivity factor due to weather (PFW) for the specified construction period and returns this information in the form of a message to the inquiring object. As mentioned earlier in Chapter 2, the impact of weather on construction activities can either be partial or complete; partial loss is generally attributed to reduced labor productivity and complete to work stoppage which interrupts those activities (Moselhi et al 1997). In the *Date* object, daily productivity factors combining the impact of reduced labor productivity and work stoppage are estimated. These daily productivity factors are then averaged over the specified construction period to estimate the average productivity factor as follows:

$$PFW = \frac{1}{C} \sum_{i=1}^{C} PF_i^{rl} \times PF_i^{ws} \qquad (3.6)$$

where,

PFW: average productivity factor considering the combined impact of reduced labor productivity and work stoppage during specified construction period;

PF_i^{rl}: productivity factor considering the reduced labor productivity due to temperature, humidity and wind speed on day i;

PF_i^{ws}: productivity factor considering the effect of work stoppage due to precipitation on day i; and

C: number of days within the specified activity duration before considering weather impact.

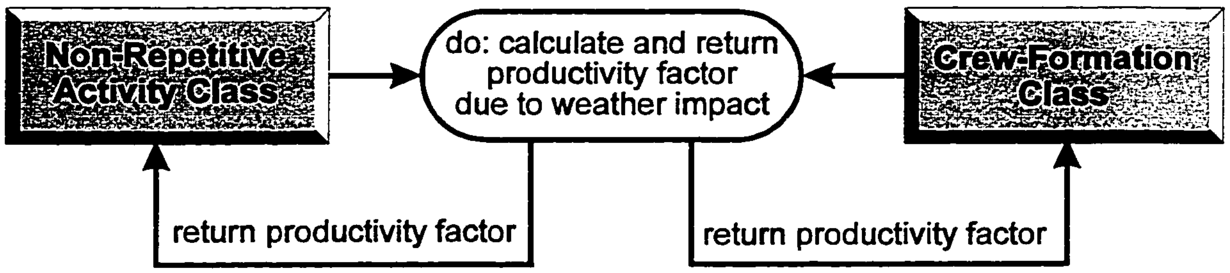

Non-Repetitive Activity Class
do: calculate and return productivity factor due to weather impact
Crew-Formation Class
return productivity factor
return productivity factor

Figure 3.14 State Diagram of *Date* Class

In order to identify the weather impact due to reduced labor productivity, the *Date* object estimates the factor PF_i^{rl} based on: 1) daily temperature, humidity and wind speed extracted from its database; and 2) formulas and tables relating these weather conditions to productivity for a number of construction trades, in a similar manner to that proposed by Moselhi et al. (1997). *Date* class includes a number of these formulas and tables obtained from the literature for the construction trades of: 1) electrical work (National Electrical Contractors Association 1974); 2) masonry (Grimm and Wagner 1974); 3) manual tasks; 4) equipment tasks (Engineering News Record March 20 1986); 5) underground pipelines (Department of the Navy

June 1969), and 6) general construction (Koehn and Brown 1985).

In order to identify the weather impact due to work stoppage caused by precipitation, the *Date* object estimates the factor PF_i^{ws} as follows:

$$PF_i^{ws} = \frac{W - R_i}{W} \tag{3.7}$$

where,

W: total daily working hours; and

R_i: lost working hours due to rain on day i which are stored in the database of *Date* class.

In order to estimate productivity factor due to weather, *Date* class includes a database representing the weather characteristics for the city of Montreal. The database consists of three arrays of daily effective temperature, humidity, and precipitation for an average year. It should be noted that the first two arrays of effective temperature and humidity are obtained from a previous thesis in the Centre for Building Studies by Nicholas (1989). These weather parameters were estimated based on historical weather for a period of ten years (1971 to 1980). For each day of the year, the hourly weather data recorded during the daily working hours (i.e. 9 a.m. to 5 p.m.) were averaged. It should be pointed out that the weather data in that thesis did not include precipitation data. In this thesis, however, the database of *Date* class expands on that latter work to include lost hours due to precipitation (R_i) (Moselhi et al 1997). R_i values are estimated based on the hourly precipitation data for the city of Montreal for the same ten year period treated by Nicholas (1989) for consistency. The original precipitation data file obtained from Environment Canada was in the form of hourly accumulation of precipitation in mm for a ten year period. The number of precipitation hours on day i (R_i) is estimated by averaging the number of precipitation hours occurring during the daily working hours of the same day (e.g. August 2nd) over the ten years period considered in this thesis (1971 to 1980).

3.5 Summary

A model for optimized scheduling of repetitive construction projects is presented. The model is based on object-oriented modelling and its development consists of three stages: analysis, design and implementation. This chapter covered the analysis stage which involved a field study and the development of an object model and a dynamic model. The field study was conducted to identify practical factors affecting scheduling of repetitive construction. This chapter also introduced the design stage for various classes, and the design of scheduling calculation and the consideration of weather impact in the proposed model. The scheduling calculation in the model is designed to enable the integration of scheduling techniques for non-repetitive and repetitive activities which are commonly encountered in repetitive construction projects. Scheduling of the non-repetitive group of activities is performed in a process similar to that of network-based techniques but through an exchange of messages. Scheduling of the repetitive group of activities, however, is based on a newly developed resource-driven scheduling algorithm (explained in Chapter 4). The consideration of weather impact in the proposed model provides practical advancements over available ones. It utilizes different formulas to estimate reduced labor productivity of various construction trades and it considers the combined impact of reduced labor productivity and work stoppage. The design stage of the model is continued in the following two chapters, namely the design of two main algorithms for resource-driven scheduling and scheduling optimization. The implementation stage of the proposed model will be explained in Chapter 6.

CHAPTER 4

AN ALGORITHM FOR RESOURCE-DRIVEN SCHEDULING OF REPETITIVE ACTIVITIES

4.1 Introduction

This chapter focuses on the detailed design of the proposed object-oriented model, utilizing the concepts described in the initial design stage (section 4 of Chapter 3). It presents a proposed algorithm for resource-driven scheduling of repetitive activities. The algorithm is included in *Crew-Formation* class. As stated earlier in Chapter 3, scheduling calculation in the proposed model is performed by exchanging messages among various objects. When a repetitive activity object receives a message invoking it to proceed with scheduling calculations, it sends another message to its associated crew formation object. The latter message includes information about: 1) number of repetitive units; 2) quantity of each repetitive unit; 3) possible early start of each unit that satisfies precedence relationships; and 4) user-specified execution order of the repetitive units. Based on this information and crew formation data, the *Crew-Formation* object identifies early start and finish dates of the activity in each repetitive unit by applying an algorithm for resource-driven scheduling as shown in Figure 4.1.

4.2 Resource-Driven Scheduling

In a repetitive activity, a construction crew often repeats the same work of that activity, moving from one repetitive unit, in the project, to another. In a highway project, for example, an earthmoving crew repeats the earthmoving operations from one unit to the next in the project. In order to maintain work continuity in this environment, repetitive units must be scheduled in such a way to enable timely movement of crews from one unit to the next, avoiding crew idle time. This is known as the "crew work continuity constraint", and its application during scheduling provides for an effective resource utilization strategy, that leads to: 1) maximization of the benefits from the learning curve effect for each crew; 2) minimization of idle time of each crew; and 3) minimization of the off-on movement of crews on a project once work has begun (Birrell 1981, Ashley 1980).

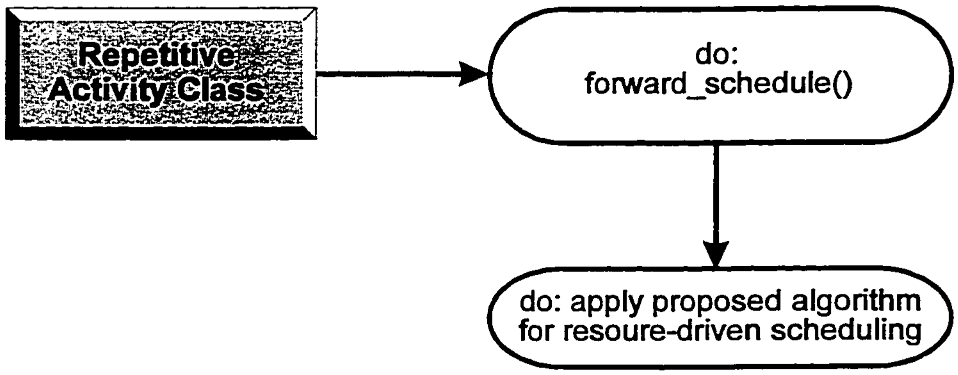

Repetitive Activity Class
do: forward_schedule()
do: apply proposed algorithm for resoure-driven scheduling

Figure 4.1 State Diagram of Crew Formation

Despite the apparent advantages of maintaining crew work continuity, its strict application may lead to a longer overall project duration in some cases such as that shown in the first solution generated in the numerical example analyzed at the end of this chapter. Selinger (1980) suggested that the violation of crew work continuity constraint, by allowing work interruptions, may reduce the overall project duration and, accordingly, the project indirect cost. However it should be noted that work interruptions result in idle crew time and accordingly may lead to increased direct cost. Russell and Wong (1993) criticized the application of work continuity for all activities, and suggested that work continuity condition should be satisfied but not strictly enforced in scheduling repetitive activities. As stated earlier in Chapter 2, the application of traditional scheduling techniques (i.e. barcharts and networks) to the scheduling of repetitive units in projects has been widely criticized in the literature for their inability to maintain crew work continuity, even if resource levelling is utilized (Selinger 1980, Reda 1990, and Russell and Wong 1993). In addition, such techniques initially assume unlimited availability of resources in the development of a project schedule, and through resource allocation revise the project schedule in order to comply with resource availability.

Unlike traditional scheduling techniques, resource-driven scheduling directly accounts for crew work continuity as well as resource availability to facilitate effective resource utilization. As such, resource-driven scheduling of repetitive activities requires the satisfaction of three constraints: 1) precedence relationships, 2) crew availability, and 3) crew work continuity. The precedence relationship constraint defines the job logic between successive construction activities, in compliance with the construction methods used. For example, pouring concrete should be preceded by formwork erection. The crew availability constraint depends on the available number of crews that can be assigned to construct an activity in all repetitive units of the project. For example, a single crew assigned to construct 9 successive units (case 1 of Figures 4.2 and 4.3) can start work on the second unit only after finishing the first, while three crews assigned to construct the same units (cases 2 through 5 of Figures 4.2 and 4.3) can work simultaneously on the first three units. The third constraint is the crew work continuity which requires that the schedule of repetitive units be organized in such a way to maintain work continuity of the assigned crews. For example, a crew assigned to construct an activity in a number of repetitive units should be able to move promptly from one unit to the next without delay in order to minimize its idle time (see Figures 4.2 and 4.3).

As stated earlier in Chapter 3, the field study indicated the need to consider a number of practical factors commonly encountered during the scheduling of repetitive activities. An examination of resource-driven scheduling of repetitive activities, including the findings of the field study and pertinent literature, indicates the need to account for: 1) type of activity (i.e. typical or non-typical); 2) number of crews assigned to work simultaneously on an activity; 3) interruption of crew work continuity; 4) crew availability period on site; 5) the order of executing repetitive units; 6) weather impact; and 7) learning curve effect. Accounting for these factors provides added flexibility and practicality in scheduling this class of projects. It should be noted that different combinations of these factors creates a number of possible cases as shown in Figures 4.2 and 4.3. The two figures illustrate the impact of the first five factors, considering typical (Figure 4.2) and non-typical repetitive activities (Figure 4.3), respectively. Unlike case 1 of Figure 4.2, case 1 of Figure 4.3 shows how activity durations vary from one repetitive unit to another. The remaining four cases (i.e. multiple crews, interruptions, crew availability period on site and construction sequence) are also depicted in Figures 4.2 and 4.3. For example, case 4 of Figures 4.2 and 4.3 shows the impact of a crew availability period starting from time zero to days 20 and 19, respectively. The two figures clearly demonstrate that the above factors have a direct impact on determining the scheduled start and finish times of each repetitive unit as well as its assigned crew.

A number of methods has been proposed in the literature for scheduling of repetitive activities. (Al Sarraj 1990, Carr and Meyer, 1974, Chrzanowski and Johnston 1986, Eldin and Senouci 1994, Johnston 1981, Moselhi and El-Rayes 1993, Reda 1990, Russell and Caselton 1988; Russell and Wong 1993; and Selinger 1980). These scheduling methods, however, do not reveal an explicit methodology or an algorithm that considers the above seven factors. This chapter presents a simple, flexible and comprehensive algorithm for resource-driven scheduling that is applicable to typical and non-typical repetitive activities and accounts for the seven factors stated earlier. The algorithm establishes the scheduled start and finish times for each activity, and identifies the crew assigned to each activity, in such a way that ensures practicality and flexibility.

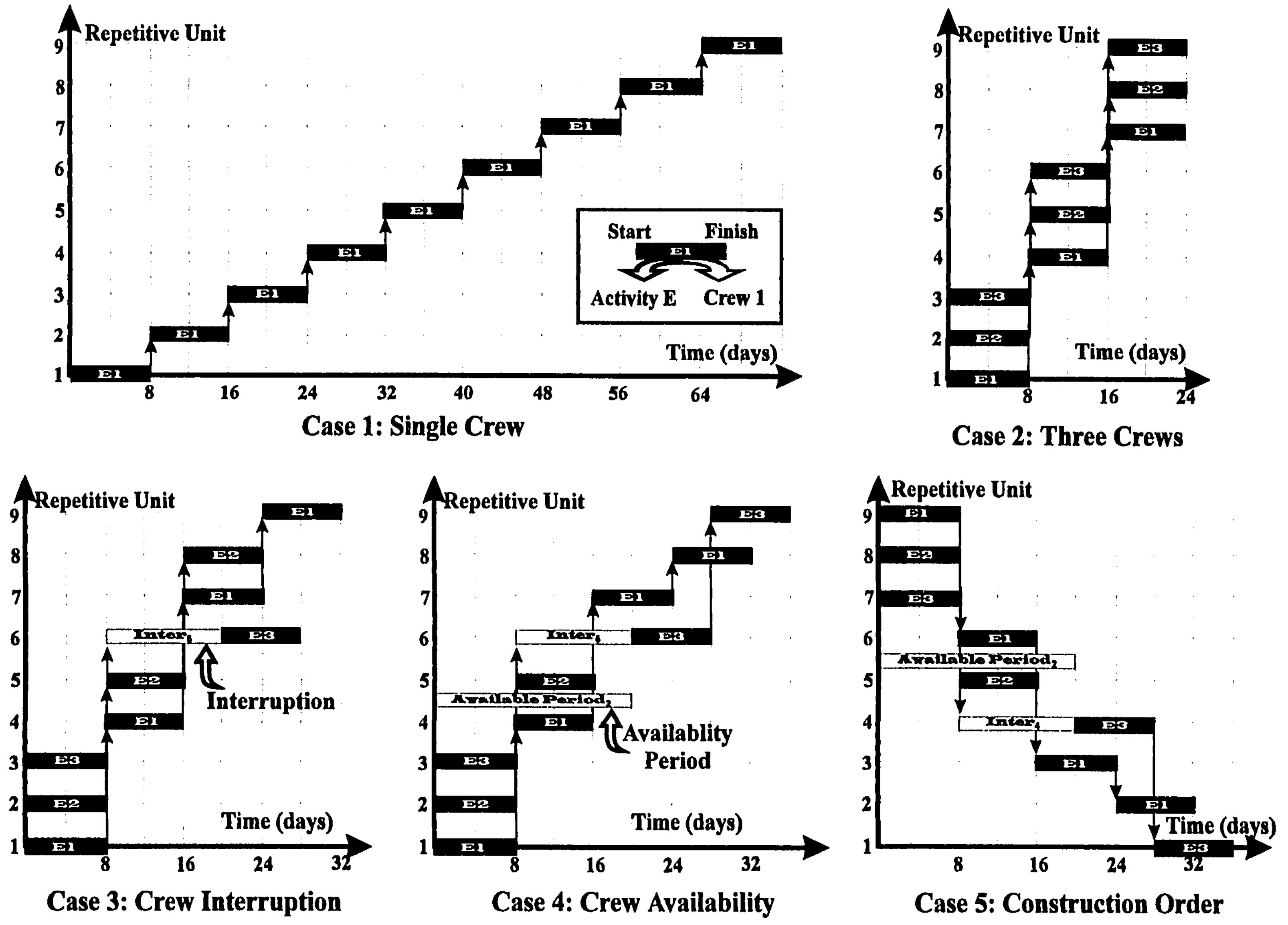


Figure 4.2 Scheduling of a Typical Repetitive Activity

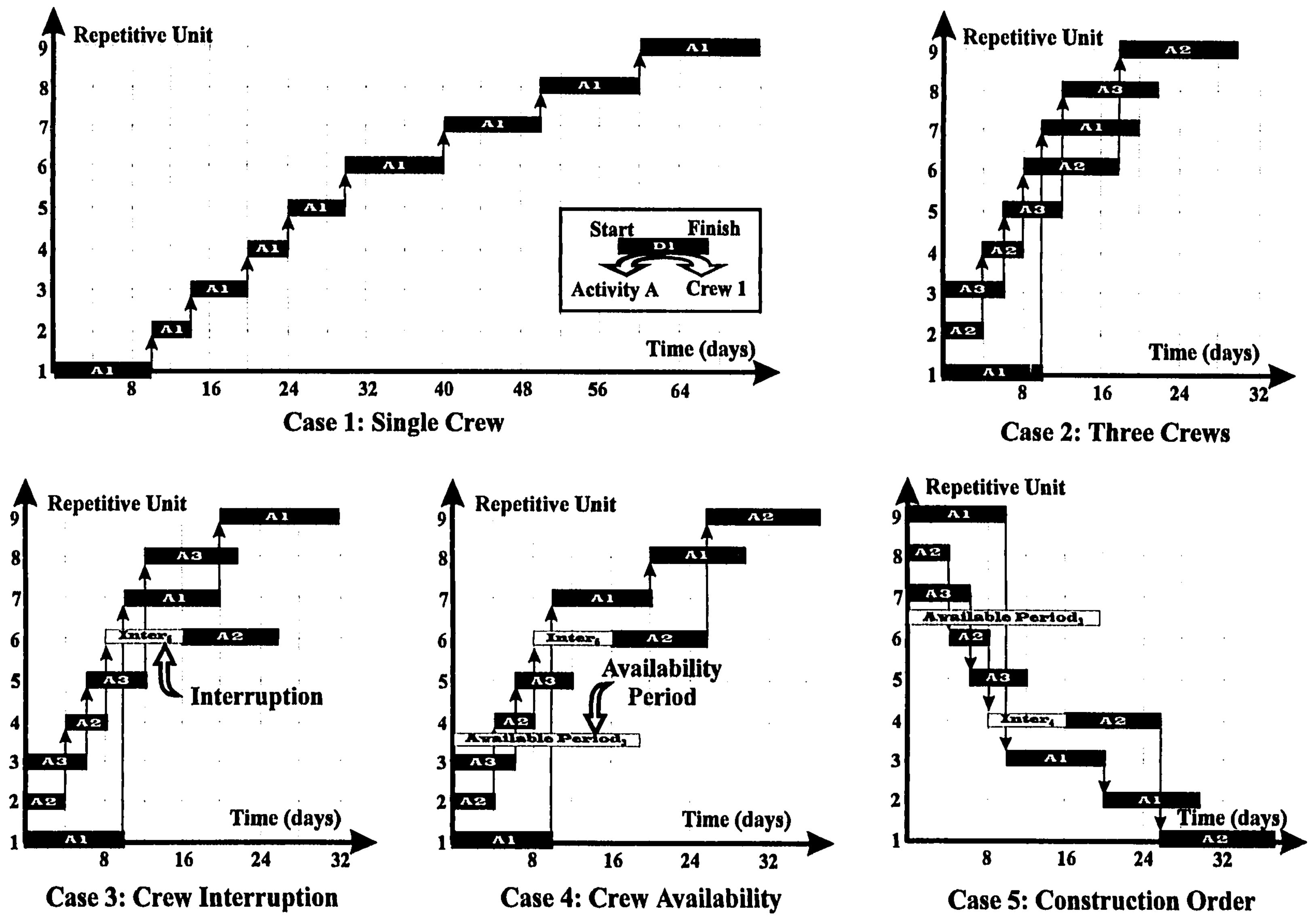


Figure 4.3 Scheduling of a Non-Typical Repetitive Activity

4.3 Proposed Scheduling Algorithm

In the development of this algorithm, three constraints have been considered: 1) logical precedence relationships; 2) crew availability; and 3) crew work continuity. The algorithm is designed embracing practicality and flexibility. It considers: typical and non-typical repetitive activities, single and multiple crews, activity interruption, crew availability period on site, sequence of construction operations, weather impact and learning curve effect. The scheduling algorithm is applied to each repetitive activity in the project to identify the scheduled start and finish times of all of its repetitive units as well as their assigned crews. As shown in Figures 4.4 and 4.5, the algorithm is performed in two stages: the first achieves compliance with logical precedence relationships and crew availability constraints, and the second achieves further compliance with crew work continuity constraint.

For each repetitive unit (j) of the activity being considered, stage 1 scans available crews and identifies the earliest one that can be assigned to unit j. The early possible start time of the selected crew is identified as the *early start* time of the unit. This *early start* time of the unit due to crew availability constraint is then compared to its possible early start time due to precedence relationship constraint (PES_j), and the latest value of the two is identified as the scheduled start time of the unit (S_j). For example in Figure 4.4, the seventh unit has *early start* = *10* and $PES_7 = 16$, and therefore its $S_7 = 16$. The duration of the unit (D_j) is then calculated and accordingly its scheduled finish time (F_j) is identified. As such, stage 1 identifies the scheduled start and finish times as well as the assigned crew of each repetitive unit in compliance with both logical precedence relationships and crew availability constraints, and considering the seven factors described earlier.

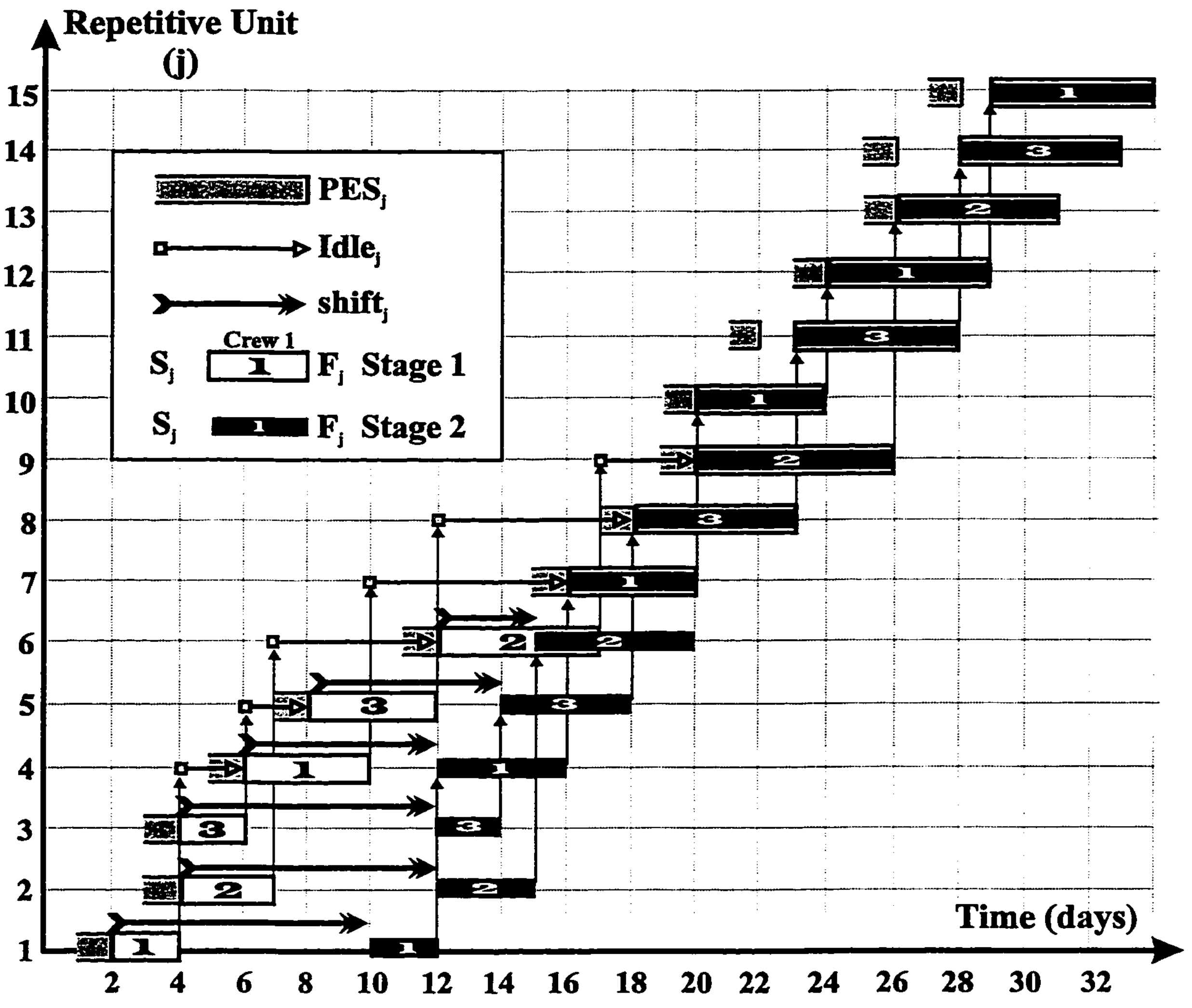


Figure 4.4 Stages Used in the Algorithm

The developed schedule of stage 1, however, does not necessarily maintain work continuity constraint for all crews. For example in stage 1 of Figure 4.4, crew number 1 that is assigned to the fourth unit has to remain idle for 6 days before being able to start working on the seventh unit (i.e. $idle_7 = 6$), thus violating crew work continuity. Stage 2 is designed to achieve further compliance with crew work continuity constraint, by shifting the developed activity schedule of stage 1, if need be. For example in Figure 4.4, the scheduled start and finish times of the fourth unit in stage 1 are shifted by 6 days in stage 2 (i.e. $shift_4 = 6$) in order to eliminate crew idle time and maintain work continuity of crew 1 between units 4 and 7.

4.3.1 Stage 1

For each repetitive activity being considered, stage 1 starts with initializing the next start array (NS_m) with the user-specified earliest available date ($MinAv_m$) of each crew m in loop number 1, starting with the first crew (m=1) through the last (m=M) as follows:

$$NS_m = MinAv_m \qquad (4.1)$$

where,

NS_m:	next possible start for crew m; and
$MinAv_m$:	earliest available date of crew m on site.

The present algorithm allows the consideration of a user-specified crew availability period on site (i.e. earliest available date ($MinAv_m$) and latest available date ($MaxAv_m$)). The algorithm, however, provides a default value of zero for all elements of $MinAv_m$, indicating that crew m can be available on site as early as needed, unless otherwise specified.

Having generated NS_m, the algorithm proceeds with scheduling following loop number 2 of Figure 4.5. For each repetitive unit (j), the algorithm:

1) Identifies the repetitive unit (k) in conformity with the user-specified construction sequence ($Order_j$) as follows:

$$k = Order_j \tag{4.2}$$

$Order_j$:	user-specified order of execution throughout all repetitive units;
and k:	unit number that satisfies the specified $Order_j$;

The algorithm provides the user with three options to specify the order of executing repetitive units: a) ascending order (i.e. $Order_j$ = {1,2,3,...,J-1,J}, where J is the total number of repetitive units in the project); b) descending order (i.e. $Order_j$ = {J,J-1,..,3,2,1}); or c) specified order (e.g. $Order_j$ = {4,3,J,...,J-1,2}). The algorithm provides a default option of ascending order unless otherwise specified. For example, the order of executing the earthmoving activity, treated later in the numerical example, can be identified as user-specified order (i.e. $Order_j$={4,3,2,1,5,6,7,8,9,10,11,12,13,14,15}).

2) Initializes the availability status array (AS_m) of each crew with a binary value of 1 in nested loop number 3, starting with the first crew (m=1) through the last (m=M) as follows:

$$AS_m = 1 \tag{4.3}$$

where,

AS_m:	binary array indicating the availability status of crew m on site, where AS_m = 1 indicates that crew m is available and AS_m = 0 indicates otherwise;

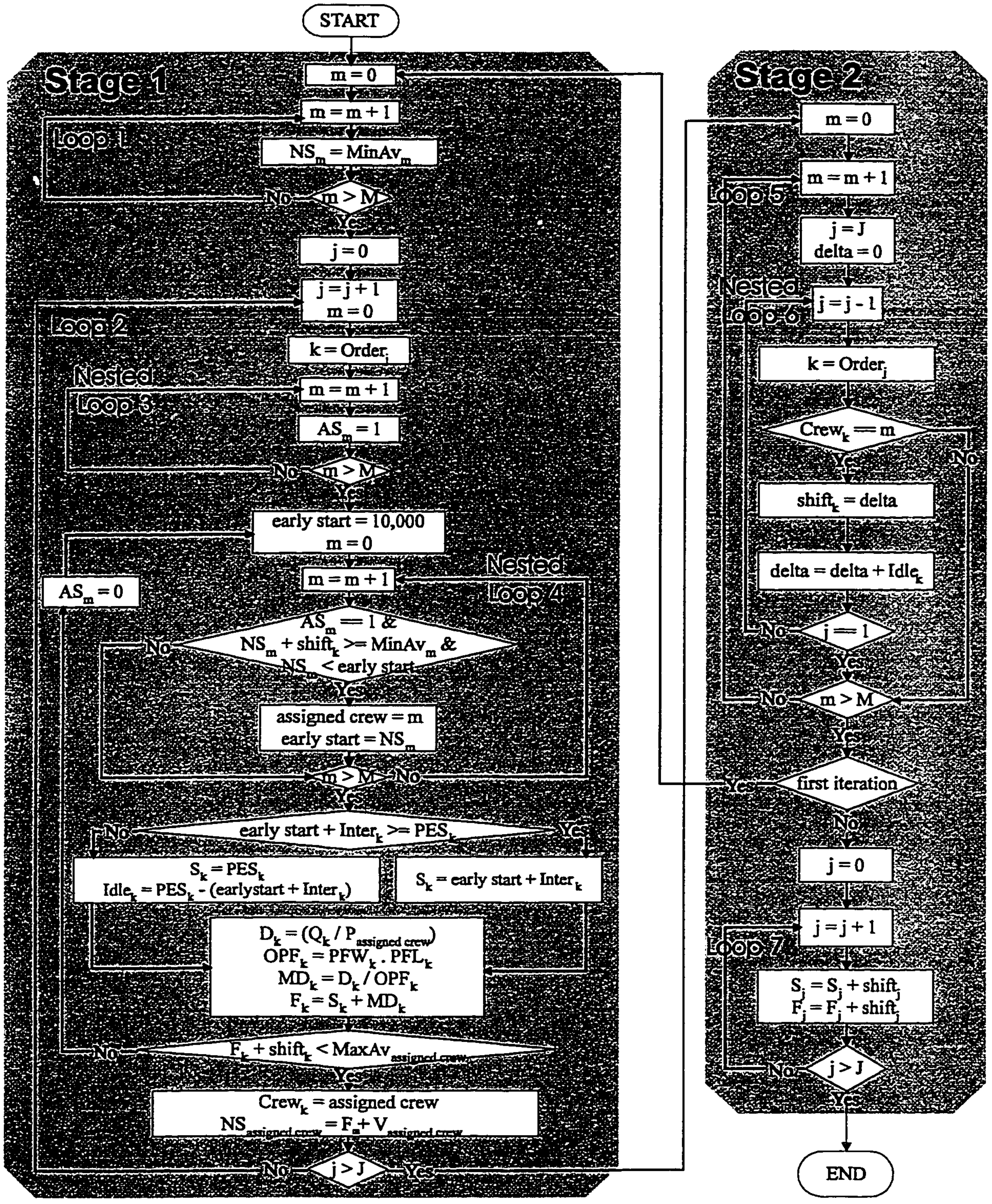


Figure 4.5 Scheduling Algorithm

3) Initializes the *early start* variable of unit k with a large number (e.g. 10,000). A large number is used to be replaced initially by NS_m of crew 1 in the first iteration of loop 4, and later by the smallest NS_m of the remaining crews in subsequent iterations of the same loop.

4) Scans all crews starting with first crew (m=1) through the last (m=M) to identify the first available crew that can be assigned to unit k and its earliest possible start time. As shown in nested loop 4 of Figure 4.5, this scanning process examines for each crew m if: a) it is available on site (i.e. $AS_m = 1$); b) its next assignment date is greater than or equal to its user-specified earliest available date on site (i.e. $NS_m + shift_k >= MinAv_m$); and c) its next possible start (NS_m) is less than the early start variable of unit k (i.e. $NS_m <$ early start). If any of these three conditions is not true, examine the next crew (m+1) and repeat step 4). Otherwise continue with crew m and move forward to the next step (i.e. step 5) after initially identifying crew m as the *assigned crew* to unit k and its next possible start as the *early start* of unit k as follows:

$$assigned\ crew = m \tag{4.4}$$
$$early\ start = NS_m \tag{4.5}$$

where,

assigned crew:	identification number of the assigned crew; and
early start:	the earliest possible start time of unit k due to crew availability.

5) Identifies the unit interruption ($Inter_k$), if any. The algorithm permits user-specified interruption to crew work continuity. For example, interrupting the grub and remove stumps activity, treated later in the second solution of the numerical example, can be achieved by specifying an interruption of 4 days before the start of the sixth and the seventh units (i.e. $Inter_6$=$Inter_7$=4). The algorithm provides a default value of zero to all elements of $Inter_k$, indicating the strict application of crew work continuity unless otherwise specified. Based on the data generated in step 4) and the user-specified activity interruption, the algorithm compares the possible early start time due to logical precedence relationship of unit k (PES_k) to its earliest start time due to crew availability (early start + $Inter_k$). If PES_k is less than or equal to (early start + $Inter_k$), the crew can start work immediately on unit k after completing work on its previously assigned unit, and the scheduled start time of unit k (S_k) can be identified as follows:

$$S_k = early\,start + Inter_k \tag{4.6}$$

where,

S_k: scheduled start time of repetitive unit k;

If PES_k is greater than (early start + $Inter_k$), the crew has to remain idle until the possible early start time due to activity precedence (PES_k) allows it to start. The scheduled start time of unit k (S_k) and the idle time imposed on its assigned crew ($Idle_k$) can be calculated as follows:

$$S_k = PES_k \tag{4.7}$$
$$Idle_k = PES_k - (early\ start + Inter_k) \tag{4.8}$$

where,

$Idle_k$:	idle time imposed on the assigned crew to unit k due to compliance with activity precedence relationship constraint.

6) Calculates the unimpacted duration of unit k before considering the impact of weather and/or learning curve as follows:

$$D_k = \frac{Q_k}{P_{\text{assigned crew}}} \tag{4.9}$$

where,

D_k:	duration of repetitive unit k;
Q_k:	quantity of work in unit k;
$P_{\text{assigned crew}}$:	daily output rate of the assigned crew.

7) Estimates the productivity factor for the assigned crew to unit k due to weather impact (PFW_k), if specified by the user. In order to achieve this, the *Crew-Formation* object sends a message to a *Date* object seeking PFW_k. The message includes the applicable type of construction trade (e.g. masonry) and the scheduled start and finish dates of the activity obtained from steps 5) and 6). In response to this message, the *Date* object calculates an average productivity factor due to weather for the specified construction period of the

activity in unit k, and returns the calculated factor in another message to the inquiring *Crew-Formation* object. The estimation of PFW_k inside the *Date* object is based on the procedure described earlier in section 3.4.12.

8) Estimates the productivity factor for the assigned crew to unit k due to learning curve effect (PFL_k), if specified by the user. PFL_k is estimated based on equations 2.2 and 2.3, assuming a linear learning model and an R value of 0.9 which represents the majority of construction work (see Table 2.1).

9) Calculates overall productivity factor for the assigned crew to unit k due weather and learning curve effect (OPF_k) as follows:

$$OPF_k = PFW_k \times PFL_k \tag{4.10}$$

10) Calculates the modified duration and finish times of unit k as follows:

$$MD_k = \frac{D_k}{OPF_k} \tag{4.11}$$

$$F_k = MD_k + S_k \tag{4.12}$$

where,

MD_k:	modified duration of activity in unit k due to weather and/or learning curve effect; and
F_k:	scheduled finish time of repetitive unit k.

11) Examines if the finish time of crew m, assigned to unit k, (F_k) is earlier than the user-specified latest available date on site ($MaxAv_m$), if any. This feature in the algorithm provides the user with a practical option to specify that a certain crew can be available to work on site for a limited period of time, after which it cannot be assigned additional work. The algorithm provides a default value of a large number to all elements in the $MaxAv_m$ array, indicating that crews can stay on site for as long as needed unless otherwise specified by the user. If the condition examined in this step is violated, repeat the scanning process of all possible crews once again (i.e. perform steps 3 through 7) after identifying crew m as unavailable (i.e. AS_m=0), and accordingly disregard crew m in step 4). Otherwise, confirm the assignment of crew m to unit k as follows:

$$Crews_k = assigned\ crew \tag{4.13}$$

$$NS_{\text{assigned crew}} = F_k + V_{\text{assigned crew}} \tag{4.14}$$

where,

$Crew_k$:	identifies crew number assigned to unit k; and
V_m:	movement time of crew m to the next unit, if any. The default value is assumed to be zero.

12) Repeat steps 1) to 11) for the next repetitive unit (j=j+1) until (j=J). As such, the start and finish times (S_j and F_j) as well as the assigned crew ($Crew_j$) are identified for each unit (j), in compliance with crew availability and activity

precedence relationship constraints.

4.3.2 Stage 2

As stated earlier, the developed schedule of stage 1 may cause crew idle time in some units. For example in Figure 4.4, crew 1 is assigned to construct units 1, 4, 7, 10, 12 and 15, respectively. The developed schedule of stage 1 does not maintain work continuity for crew 1 and results in its idle time in a number of units (i.e. $Idle_1=0$, $Idle_4=2$, $Idle_7=6$, $Idle_{10}=0$, $Idle_{12}=0$, $Idle_{15}=0$). In order to eliminate crew idle time and maintain crew work continuity constraint, stage 2 shifts the developed schedule of some units, if any, in stage 1 to a later time. The shift is calculated for all units assigned to each crew m in loop 5 and nested loop 6 of stage 2 as shown in Figure 4.5. The required shift of a unit ($shift_k$) is identified as the summation of all idle times of later scheduled units assigned to the same crew. For example in Figure 4.4,

$shift_{15} = 0;$

$shift_{12} = Idle_{15} = 0;$

$shift_{10} = Idle_{15} + Idle_{12} = 0;$

$shift_7 = Idle_{15} + Idle_{12} + Idle_{10} = 0;$

$shift_4 = Idle_{15} + Idle_{12} + Idle_{10} + Idle_7 = 6;$

$shift_1 = Idle_{15} + Idle_{12} + Idle_{10} + Idle_7 + Idle_4 = 8.$

The identified shift of each unit is used to shift the scheduled start and finish times of stage 1. This shift, however, may cause the working period of some crews to be shifted outside their availability periods that was considered in the developed schedule of stage 1. Therefore, stage 1 is repeated for a second iteration to consider the impact of such shifts on crew availability. It should be noted that all values of $shift_j$ array are assumed to be zero in the first iteration of stage 1, and are obtained as mentioned above in the second iteration of stage 1. After the second iteration of stages 1 and 2, the developed schedule of stage 1 is shifted in loop 6 of stage 2 as follows:

$$S_j = S_j + shift_j \tag{4.15}$$

$$F_j = F_j + shift_j \tag{4.16}$$

where,

$shift_j$: required shift of the start and finish times of unit j to comply with crew work continuity constraint.

It should be noted that the algorithm does not guarantee a feasible solution when one theoretically exists, employing all crews with their limited availability periods. The algorithm, however, guarantees such a feasible solution once that condition on crew availability is relaxed.

4.4 Numerical Example

A numerical example of a highway project is analyzed in order to illustrate the use of the proposed algorithm and demonstrate its capabilities. The project involves the construction of a three lane highway for a stretch of 15 kilometres, and consists of five consecutive activities: *cut and chip trees*, *grub and remove stumps*, *earthmoving*, *base* and *paving*. The project is divided, for simplicity, into 15 repetitive units, each has a length of one kilometer. While in this example each unit is assumed to have a length of one kilometer, the developed scheduling algorithm can be applied to different lengths as it is designed to accept varying scope of works in repetitive units. Each of the five activities is repeated at each of the 15 segments or units of the project. The precedence relationships among these sequential activities are finish to start with no lag time. In this example, the activities of *cut and chip trees*, *grub and remove stumps* as well as *earthmoving* do not have identical durations in all units and thus can be classified as non-typical repetitive activities. The activities of *base* and *paving*, however, have identical durations in all units and thus can be classified as typical repetitive activities. The two categories of repetitive activities are represented in this example to illustrate the flexibility of the algorithm.

The activity and crew data, used in this example, are summarized in Tables 4.1 and 4.2, respectively. Activity data in each unit include: 1) quantity of work in units specified by the user (Q_j); 2) interruption to crew work continuity, if any ($Inter_j$); and

3) order of execution ($Order_j$). The data used in Table 4.1 are reasonably assumed based on the size of the project being considered. For example, the seventh kilometer ($j = 7$) of the *earthmoving* activity has a quantity of 6500 m^3 ($Q_7 = 6500$), will be the second unit to be executed ($Order_2 = 7$), and since there is no specified interruption to its assigned crew, its $Inter_7 = 0$.

Crew data is summarized in Table 4.2. For each repetitive activity, the data includes: 1) daily output in unit/day (P_m); 2) earliest available date on site ($MinAv_m$); and 3) latest available date site ($MaxAv_m$). In an effort to use realistic data, the daily output of crews are obtained from Means Construction Cost Data (1993), and modified to SI units. Data pertaining to crew availability period on site are assumed to demonstrate some practical features in the developed algorithm. For example, crew 1 that can be assigned to the *base* activity has a daily output rate of 3200 m^2/day (P_1=3200), can move to site on day 26 ($MinAv_1$=26) and can stay on site for as long as needed ($MaxAv_1$=large number). In this example, the availability period of some crews is assumed to be limited (e.g. crews 1 and 2 of the *earthmoving* activity), while that of others is not (e.g. crews 1, 2 and 3 of the *paving* activity) as shown in Table 4.2.

Another possible and practical scenario of resource utilization occurs when a crew has to move out of site after a first working period and remain absent for an intermittent period, after which it can return and resume a second working period.

Such a scenario can occur, for example, due to out of site concurrent commitment, especially, in cases where multiple projects serviced by the same resource pool. The present algorithm can account for such a scenario by replacing the two availability periods of the same crew with two periods of two distinct crews.

For example, crew 3 of the activity *cut and chip trees* can be available on site for two separate periods of time. The first availability period starts on the move-in day and extends for 18 days, after which the crew has to move and remain out of site for another 6 days. The second availability period can start immediately after the out of site period and can extend for as long as needed. In this case, the two availability periods of crew 3 are replaced with two periods of two distinct crews: 3 and 4. The first working period of crew 3 can be represented by that of crew 3 which extends from project start day ($MinAv_3=0$) to day 18 ($MaxAv_3=18$). The second availability period of crew 3, however, can be represented by that of an artificial or alias fourth crew which can be available on site from day 24 ($MinAv_4=24$) and can stay on site for as long as needed ($MaxAv_4$=large number) as shown in Figure 4.6 and columns 5 and 6 of Table 4.2.

Table 4.1(A) Activity Input Data

Repetitive Unit (Km.) j (1)	Repetitive Activity								
	Cut and chip trees			Grub and remove stumps			Earthmoving		
	Quantities in m^2 Q_j (2)	Order $Order_j$ (3)	Interruption in days $Inter_j$ (4)	Quantities in m^2 Q_j (5)	Order $Order_j$ (6)	Interruption in days $Inter_j$ (7)	Quantities in m^3 Q_j (8)	Order $Order_j$ (9)	Interruption in days $Inter_j$ (10)
1	12000	1	0	12000	1	0	6000	4	0
2	12000	2	0	12000	2	0	6000	3	0
3	18000	3	0	18000	3	0	6000	2	0
4	12000	4	0	12000	4	0	7000	1	0
5	18000	5	0	18000	5	0	8600	5	0
6	30000	6	0	30000	6	0	7000	6	0
7	36000	7	0	36000	7	0	6500	7	0
8	30000	8	0	30000	8	0	6000	8	0
9	24000	9	0	24000	9	0	6000	9	0
10	24000	10	0	24000	10	0	6000	10	0
11	18000	11	0	18000	11	0	6000	11	0
12	12000	12	0	12000	12	0	6000	12	0
13	12000	13	0	12000	13	0	6000	13	0
14	12000	14	0	12000	14	0	6000	14	0
15	12000	15	0	12000	15	0	6000	15	0

Table 4.1(B) Activity Input Data (Continued)

Repetitive Unit (Km.) j (1)	Repetitive Activity					
	Base			Paving		
	Quant-ities in m^2 Q_j (2)	Order $Order_j$ (3)	Interruption in days $Inter_j$ (4)	Quant-ities in m^2 Q_j (5)	Order $Order_j$ (6)	Interruption in days $Inter_j$ (7)
1	32000	1	0	32000	1	0
2	32000	2	0	32000	2	0
3	32000	3	0	32000	3	0
4	32000	4	0	32000	4	0
5	32000	5	0	32000	5	0
6	32000	6	0	32000	6	0
7	32000	7	0	32000	7	0
8	32000	8	0	32000	8	0
9	32000	9	0	32000	9	0
10	32000	10	0	32000	10	0
11	32000	11	0	32000	11	0
12	32000	12	0	32000	12	0
13	32000	13	0	32000	13	0
14	32000	14	0	32000	14	0
15	32000	15	0	32000	15	0

As for the impact of the order of execution, it is demonstrated in the numerical example by assigning a user-specified execution order to the *earthmoving* activity as shown in column 9 of Table 4.1(A). In real-life, the order of executing earthmoving operations is often affected by site topography in order to minimize the cost of cut and fill. For simplicity in this example, the order of executing the *earthmoving* activity is assumed to start at the 4th km. and move in a descending order towards the first km., and then complete the remainder of the activity by starting at the 5th km. and moving in an ascending order towards the 15th km.

In order to demonstrate the flexibility of the algorithm with respect to activity interruption, the numerical example was analyzed twice, with and without activity interruption. First, scheduling calculations were performed without activity interruption. The calculations are initiated by sending a message to the first activity in the project (i.e. *cut and chip trees*) in order to start self scheduling. Upon receiving this message, the *cut and chip* activity initiates its PES_j array with the start time of the project (i.e. $PES_j=0$ for all j), and then starts self scheduling by calling the algorithm described earlier. Upon the completion of the calculations, the *cut and chip trees* activity sends a message to the *grub and remove stumps* activity via the relationship object linking the two. Based on this message, the *grub and remove stumps* activity modifies its early start array (PES_j) based on the finish times of its predecessor and the type of relationship. The remaining activities perform scheduling calculations and sending and receiving messages in a similar and sequential manner. The scheduling results are summarized in Table 4.3 and Figure 4.6, indicating an overall project duration of 87 days.

The process described above was repeated, but with interrupting the *grub and remove stumps* activity. An interruption of 4 days is assumed to occur before the start of the sixth and seventh units ($Inter_6=Inter_7=4$). Considering that activity interruption, the proposed algorithm was applied. The scheduling results are shown in Table 4.4 and Figure 4.7, indicating an overall project duration of 83 days. Although this illustrates an apparent advantage of activity interruption in reducing overall project time, it should be assessed on a project by project basis. Activity interruption violates the concept of crew work continuity and often leads to crew idle time, inefficient resource utilization and additional costs in mobilization and demobilization. These factors should be considered and carefully evaluated before implementing any activity interruption. As shown in Tables 4.3 and 4.4 and Figures 4.6 and 4.7, the present algorithm is capable of maintaining all precedence relationships, crew availability and crew work continuity constraints, and of accounting for a number of factors affecting the scheduling of repetitive activities. As such, the scheduling algorithm offers a number of practical capabilities.

Table 4.2 Crews' Input Data

Repetitive Activity (1)	Crew # m (2)	Crew ID from Means (3)	Daily Output in unit/day P_m (4)	Earliest Available Day $MinAv_m$ (5)	Latest Available Day $MaxAv_m$ (6)
Cut and chip trees	1	B-7	3000	0	*
	2	B-7	3000	0	*
	3	B-7	3000	0	18
	4	B-7	3000	24	40
Grub and remove stumps	1	B-30	4000	0	*
	2	B-30	4000	0	*
Earthmoving	1	2 crews B33-E	1200	10	70
	2	2 crews B33-F	800	10	70
Base	1	B-36	3200	26	*
	2	B-36	3200	26	*
	3	B-36	3200	26	*
	4	B-36	3200	26	*
Paving	1	B-25	4000	0	*
	2	B-25	4000	0	*
	3	B-25	4000	0	*

* is a default large number, indicating crew availability for as long as needed

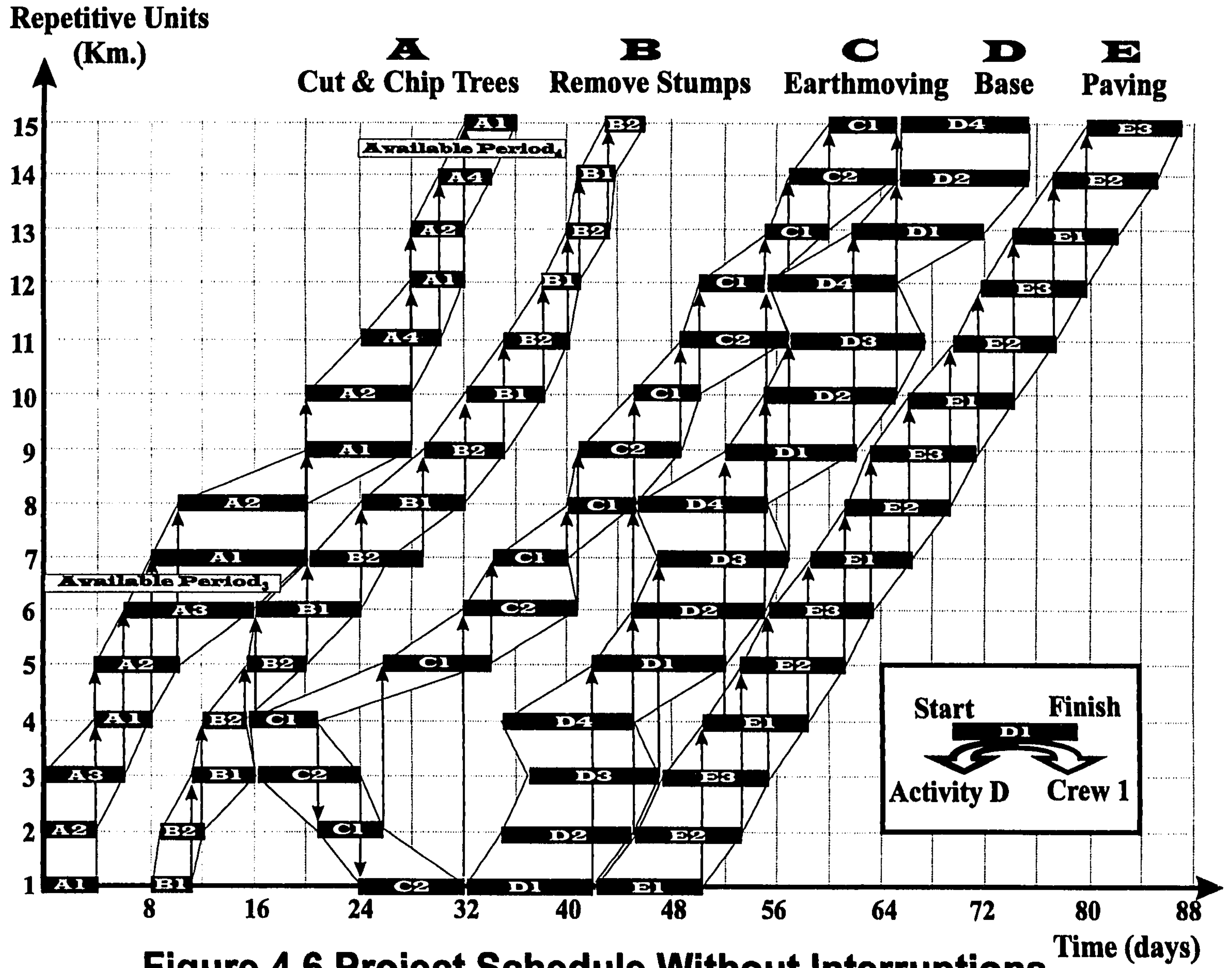


Figure 4.6 Project Schedule Without Interruptions

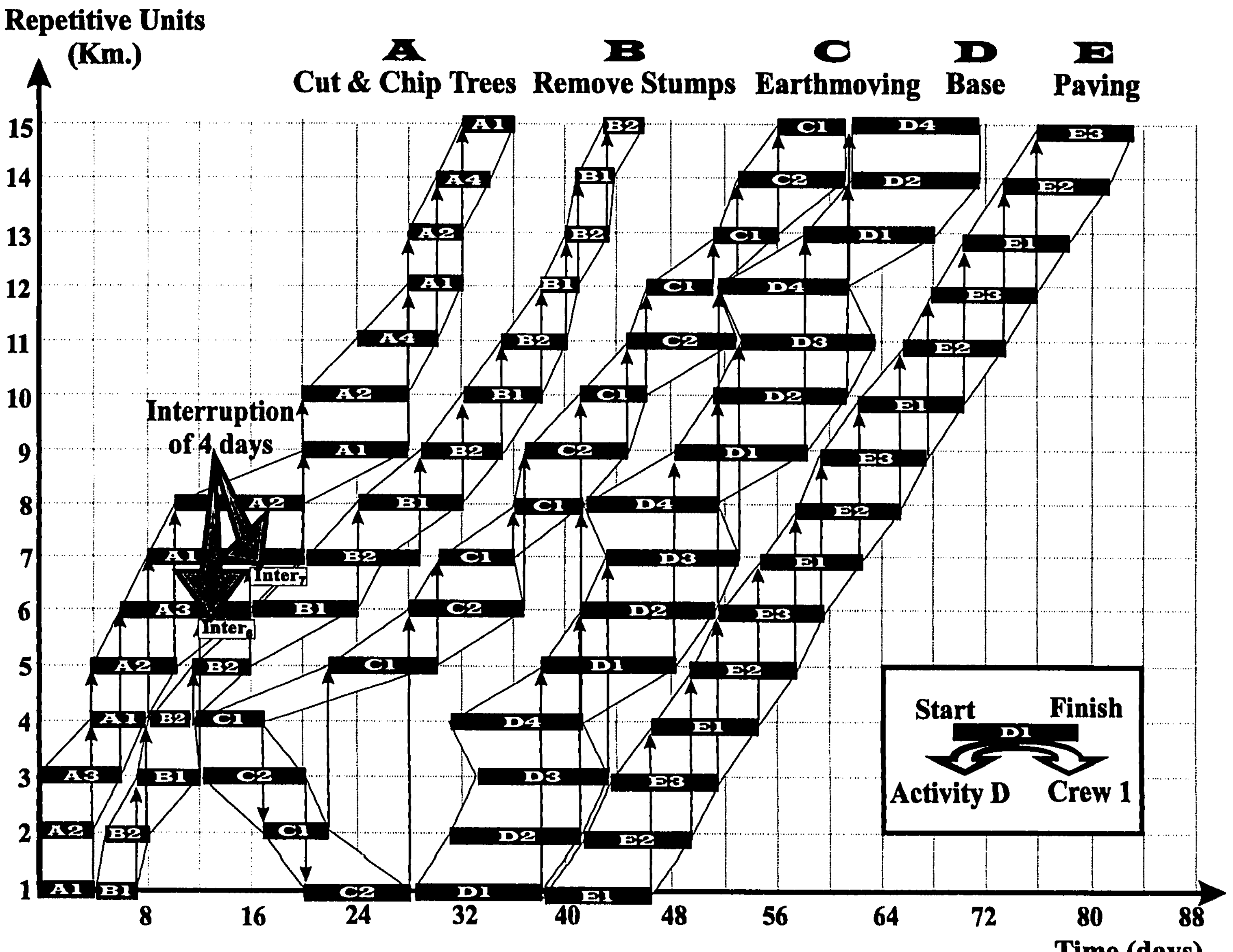


Figure 4.7 Project Schedule With Interruptions

Table 4.3(A) Project Schedule Without Interruption

Repetitive Unit (Km.) j (1)	Repetitive Activity								
	Cut and chip trees			Grub and remove stumps			Earthmoving		
	Start in days S_j (2)	Finish in days F_j (3)	Assigned crew # $Crews_j$ (4)	Start in days S_j (5)	Finish in days F_j (6)	Assigned crew # $Crews_j$ (7)	Start in days S_j (8)	Finish in days F_j (9)	Assigned crew # $Crews_j$ (10)
1	0	4	1	8	11	1	24	32	2
2	0	4	2	9	12	2	21	26	1
3	0	6	3	11	16	1	16	24	2
4	4	8	1	12	15	2	15	21	1
5	4	10	2	15	20	2	26	34	1
6	6	16	3	16	24	1	32	41	2
7	8	20	1	20	29	2	34	40	1
8	10	20	2	24	32	1	40	45	1
9	20	28	1	29	35	2	41	49	2
10	20	28	2	32	38	1	45	50	1
11	24	30	4	35	40	2	49	57	2
12	28	32	1	38	41	1	50	55	1
13	28	32	2	40	43	2	55	60	1
14	30	34	4	41	44	1	57	65	2
15	32	36	1	43	46	2	60	65	1

Table 4.3(B) Project Schedule Without Interruption (Continued)

	Repetitive Activity					
	Base			Paving		
Repetitive Unit (Km.) j (1)	Start in days S_j (2)	Finish in days F_j (3)	Assigned crew # $Crews_j$ (4)	Start in days S_j (5)	Finish in days F_j (6)	Assigned crew # $Crews_j$ (7)
1	32	42	1	42	50	1
2	35	45	2	45	53	2
3	37	47	3	47	55	3
4	35	45	4	50	58	1
5	42	52	1	53	61	2
6	45	55	2	55	63	3
7	47	57	3	58	66	1
8	45	55	4	61	69	2
9	52	62	1	63	71	3
10	55	65	2	66	74	1
11	57	67	3	69	77	2
12	55	65	4	71	79	3
13	62	72	1	74	82	1
14	65	75	2	77	85	2
15	65	75	4	79	87	3

Table 4.4(A) Project Schedule With Interruption

	Repetitive Activity								
	Cut and chip trees			Grub and remove stumps			Earthmoving		
Repetitive Unit (Km.) j (1)	Start in days S_j (2)	Finish in days F_j (3)	Assigned crew # $Crews_j$ (4)	Start in days S_j (5)	Finish in days F_j (6)	Assigned crew # $Crews_j$ (7)	Start in days S_j (8)	Finish in days F_j (9)	Assigned crew # $Crews_j$ (10)
1	0	4	1	4	7	1	20	28	2
2	0	4	2	5	8	2	17	22	1
3	0	6	3	7	12	1	12	20	2
4	4	8	1	8	11	2	11	17	1
5	4	10	2	11	16	2	22	30	1
6	6	16	3	16	24	1	28	37	2
7	8	20	1	20	29	2	30	36	1
8	10	20	2	24	32	1	36	41	1
9	20	28	1	29	35	2	37	45	2
10	20	28	2	32	38	1	41	46	1
11	24	30	4	35	40	2	45	53	2
12	28	32	1	38	41	1	46	51	1
13	28	32	2	40	43	2	51	56	1
14	30	34	4	41	44	1	53	61	2
15	32	36	1	43	46	2	56	61	1

Table 4.4(B) Project Schedule With Interruption (Continued)

Repetitive Unit (Km.) j (1)	Repetitive Activity					
	Base			Paving		
	Start in days S_j (2)	Finish in days F_j (3)	Assigned crew # $Crews_j$ (4)	Start in days S_j (5)	Finish in days F_j (6)	Assigned crew # $Crews_j$ (7)
1	28	38	1	38	46	1
2	31	41	2	41	49	2
3	33	43	3	43	51	3
4	31	41	4	46	54	1
5	38	48	1	49	57	2
6	41	51	2	51	59	3
7	43	53	3	54	62	1
8	41	51	4	57	65	2
9	48	58	1	59	67	3
10	51	61	2	62	70	1
11	53	63	3	65	73	2
12	51	61	4	67	75	3
13	58	68	1	70	78	1
14	61	71	2	73	81	2
15	61	71	4	75	83	3

4.5 Summary

This chapter presented the development of a flexible algorithm for resource-driven scheduling of repetitive activities, included as a function in the *Crew-Formation* class. For each activity in a repetitive unit, the algorithm identifies the scheduled start and finish times as well as the assigned crew. The algorithm provides a schedule that complies with precedence relationships, crew availability and crew work continuity constraints. In addition, it considers the impact of a number of practical factors: 1) type of repetitive activity (i.e. typical or non-typical); 2) multiple crews assigned to work simultaneously on an activity; 3) crew availability period on site; 4) activity interruption; 5) user-specified order of execution among repetitive units; 6) weather impact; and 7) learning curve effect. The scheduling algorithm is carried out in two main stages: the first achieves compliance with precedence relationships and crew availability constraints, and the second further achieves compliance with crew work continuity constraint. A numerical example of a highway project is analyzed to illustrate the use of the algorithm and demonstrate its capabilities. The project example is analyzed with and without activity interruption to illustrate the useful features of the algorithm. The results demonstrate that the consideration of activity interruption can help reduce project duration. The impact of activity interruption on project duration and cost is considered in more details in an optimization algorithm, presented in the next chapter.

CHAPTER 5

SCHEDULING OPTIMIZATION

5.1 Introduction

This chapter focuses on the detailed design of the scheduling optimization functions incorporated in the object-oriented model. It presents the development of two algorithms designed to optimize the scheduling of repetitive construction projects. The first algorithm generates a feasible set of interruption options for each crew formation in the project. The second algorithm identifies the optimum crew formation and interruption option, from a set of possible alternatives, for each repetitive activity in the project. The optimization algorithm utilizes dynamic programming and is capable of generating least cost or minimum duration schedules. It is invoked when the user specifies schedule optimization as shown in Figure 3.10. During scheduling calculation, a repetitive activity sends a message to the project object, inquiring about user-specified option as shown in Figure 3.10. If the user specifies schedule optimization (i.e. minimize project overall cost or project duration), the repetitive activity applies the proposed optimization functions.

5.2 Optimized Scheduling

Minimizing the duration of repetitive construction projects is a more complex process than that of non-repetitive ones due to the compliance with crew work continuity constraint (Moselhi and El-Rayes 1993 (b)). This led to the use of optimization techniques such as linear programming and dynamic programming to optimize scheduling of repetitive construction projects. Linear programming in the form of an objective function and a set of constraints has been proposed by Perera (1982 and 1983) to maximize the construction rate of the activities in a repetitive project, thus enabling a minimum project duration. The set of constraints are specified to satisfy the resource availability and financial limits. The provided optimum solution of this linear programming formulation is in the form of non-integer values. This formulation can be considered inadequate for providing a least cost schedule for a repetitive project for the following reasons: 1) cost is not considered in the optimization process, and therefore the formulation cannot be used to minimize the project overall cost; 2) the formulation cannot be applied to non-typical repetitive projects because it assumes identical durations for the activity in all repetitive units; 3) interruption of crew work continuity is not considered; 4) the produced optimum resource utilization is in the form of non-integer values (e.g. 0.556 crew) which is unrealistic in construction; and 5) the formulation of the constraints for such a problem can be a complex task for non-academic users in the construction industry.

In 1990, Reda proposed a repetitive project model (RPM) which utilizes a linear programming formulation to minimize the project direct cost for a given project duration. In the formulation, the size of the model in terms of the number of constraints depends on the number of repetitive activities in the project. In the numerical example used to illustrate the use of RPM, an objective function and 24 constraints were formulated to minimize the direct cost for an example of 6 activities. The formulation extended the one provided by Perera (1983) to include cost in the optimization process. It was, however, assumed that each activity will have a constant duration in all repetitive units. This assumption limits the application of the model to typical repetitive projects, and thus cannot provide solution to non-typical repetitive projects. In addition, the formulation is limited by the complexity of establishing the constraints, the non-integer outcome and the inability to consider interruptions similar to the formulation provided by Perera (1983).

Dynamic programming was advanced in the 1950's by Bellman (1957). There is no standard mathematical technique for dynamic programming problems. Dynamic programming is an approach to problem solving, and particular mathematical equations have to be developed to fit each problem. Dynamic programming has been applied to provide solutions to shortest route, resource allocation, equipment replacement and inventory theory (Bellman and Dreyfus 1962, Denardo 1982, and Dreyfus and Law 1977).

Dynamic programming was introduced by Selinger (1980) to minimize the duration of repetitive construction projects. The formulation was developed to satisfy two main requirements: 1) maintaining crew work continuity, and 2) determining the optimum resources to minimize the project duration. In 1988, Russell and Caselton (1988) presented an expanded dynamic programming formulation that enables the consideration of interruption to work continuity. These dynamic programming formulations can be applied to minimize the duration of both typical and non-typical repetitive projects, thus providing a flexible methodology to overcome the limitations of linear programming models. However, the application of the two formulations are limited because: 1) the consideration of interruption to work continuity was not considered in the first formulation and was only limited to a user-specified set of interruption options prior to scheduling in the second formulation, and therefore cannot guarantee the optimum solution; 2) the two formulations focus on minimizing project duration and do not account for its impact on the overall cost. In order to overcome the first limitation, the following two sections present an algorithm for generating interruption options during the scheduling process. The second limitation is circumvented by presenting an expanded dynamic programming formulation (section 5.5) that considers cost as an important decision variable in the optimization process.

5.3 Interruption of Crew Work Continuity

The application of crew work continuity constraint during scheduling of repetitive construction activities maximizes the efficiency of resource utilization. Despite the advantages of maintaining crew work continuity, its strict application often leads to a longer project duration (Selinger 1980, Russell and Caselton 1988). Selected interruption of crew work continuity can be used to minimize the duration of repetitive construction projects as shown in Figure 5.1. Russell and Caselton (1988) and Eldin and Senouci (1993) utilized interruption as a second state variable in their dynamic programming formulations in order to minimize project duration. The two formulations, however, required that the user (e.g. a construction planner) provide a prespecified set of possible interruption vectors for each activity in the project prior to scheduling. For example, Russell and Caselton (1988) provided a prespecified set of interruption vectors that included 13 alternatives in their numerical example as shown in Table 5.1. For each crew formation associated with each repetitive activity in the project, the prespecified set of interruption vectors was evaluated during the optimization process in order to identify the optimum vector.

Table 5.1 Interruption Vectors (Russell and Caselton, 1988)

Unit	Interruption Vectors (days)												
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
1	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	1	2	3	4	0	0	0	0	0	0	0	0
3	0	1	2	3	4	1	2	3	4	0	0	0	0
4	0	1	2	3	4	0	0	0	0	1	2	3	4

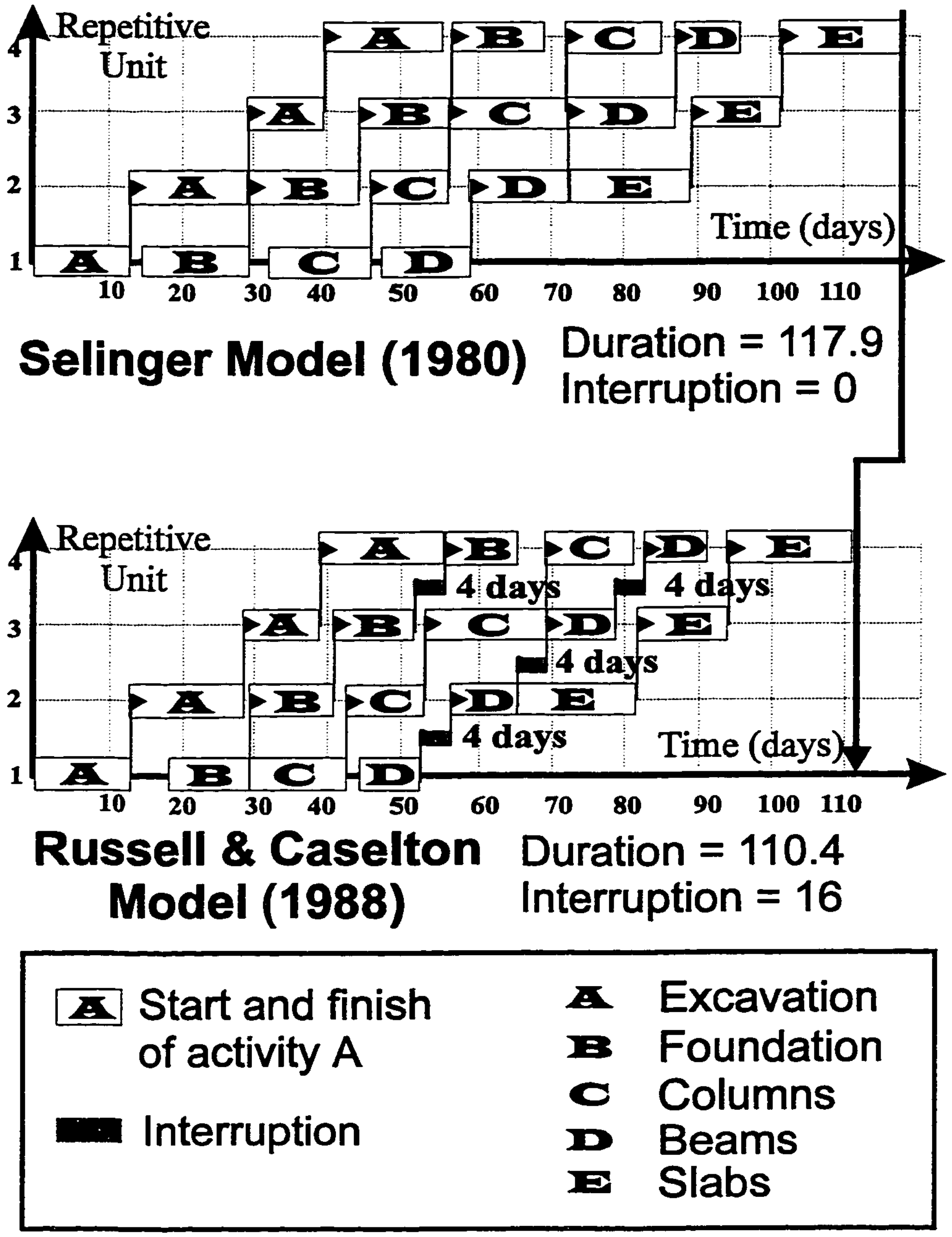


Figure 5.1 Minimizing Project Duration Using Activity Interruption

Available dynamic programming formulations that utilizes interruption vectors as a second state variable in the optimization process (Russell and Caselton 1988) can produce reduction in project duration as shown in Figure 5.1. The application of these formulations, however, is limited because they require the user to provide a prespecified set of interruption vectors before scheduling. This poses a number of limitations:

1) It is impractical to require a construction planner to identify a set of possible interruption vectors for each crew formation of each activity in the project prior to scheduling.

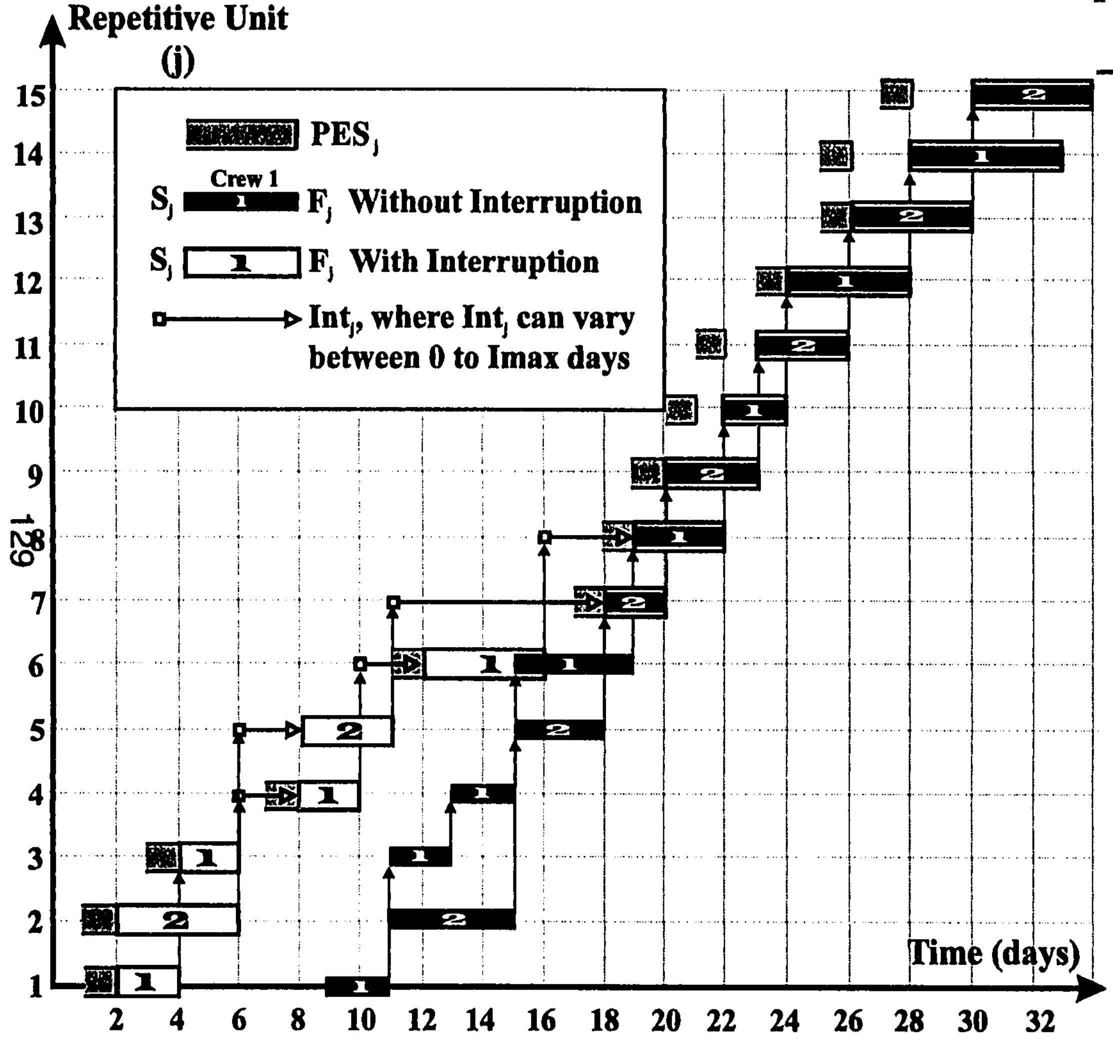
2) The number of feasible interruption vectors that can be considered for a repetitive activity can be unlimited if there is no upper limit on the number of days that a crew is allowed to be interrupted before the start of the activity in a given repetitive unit. Even if such a limit (Imax) is arbitrarily established, the number of interruption vectors that can be considered for a crew formation n associated with repetitive activity i (NIV^i_n) increases exponentially with the increase of the number of repetitive units (J). The application of an Imax value means that a crew can be interrupted by a value that ranges from 0 to Imax days before the start of the activity in a repetitive unit j. For each repetitive unit j, this leads to a total number of possible interruptions of Imax + 1, assuming that interruption can vary from 0 to Imax by an increment of 1 day as shown in Figure 5.2. Accordingly, the total number of interruption vectors that can be considered for crew formation n of activity i (NIV^i_n) that is repeated in J

repetitive units can be calculated as follows:

$$\mathrm{NIV}_n^i = (\mathrm{Imax}+1)^{J-1} \tag{5.1}$$

For example, an upper limit of 6 interruption days (Imax = 6) imposed on an activity that is repeated in 15 units (J = 15) can result in approximately 680 billion interruption vectors that need to be considered for each crew formation of each activity in the project (i.e. $\mathrm{NIV}_n^i = (\mathrm{Imax} + 1)^{J-1} = 7^{14} = 680$ billion). This clearly illustrates that even imposing a reasonable value for Imax (e.g. 6 days) still renders the optimization problem practically infeasible.

3) The arbitrary selection of a maximum interruption days (Imax) before scheduling does not guarantee the optimum solution. As shown in the numerical example presented later in this chapter, the optimum solution may require a number of interruption days greater than that specified by the Imax value. A solution that is based on an arbitrary Imax value, therefore, does not guarantee the optimum solution.



Possible Interruption Vectors for Imax = 6

(1)	...	(7)	(8)	...	(14)	(15)	...	(21)	(22)	....	(7^{14})
0	...	0	1	...	1	2	...	2	3	....	6
0	...	6	0	...	6	0	...	6	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	0	....	6
0	...	0	0	...	0	0	...	0	1	....	6
-	-	-	-	-	-	-	-	-	-	-	-

Figure 5.2 Possible Interruption Vectors For a Repetitive Activity

5.4 An Algorithm for Generating Feasible Interruption Vectors

In order to circumvent the above three limitations of available dynamic programming formulations that considers interruption as a second state variable in the optimization process (Russell and Caselton 1988, and Eldin and Senouci 1993), a flexible and practical algorithm for generating feasible interruption vectors is developed. First, the algorithm is automated and therefore the construction planner need not provide a user-specified set of interruption vectors prior to scheduling. Second, it generates a limited number of feasible interruption vectors and therefore enables a practical and feasible approach for the consideration of interruption during schedule optimization. Third, the algorithm generates all needed interruption vectors during schedule calculation rather than being limited by a prespecified Imax value before scheduling, ensuring the generation and selection of the optimum solution.

5.4.1 Limiting the Number of Interruption Vectors

The present algorithm is designed to identify only relevant interruption vectors that need to be considered during the optimization process. The algorithm utilizes four rules to limit the total number of feasible interruption vectors for a given crew formation n of activity i (NIV^i_n). A closer examination of the impact of interruption on minimizing project duration leads to the identification of the following four rules:

1) The first rule that can be used to limit the number of feasible interruption

vectors (NIV^i_n) is to identify an upper limit on the number of interruption days that need to be considered for the activity in each repetitive unit j ($Imax_j$). For each repetitive unit j, this rule considers that $Imax_j$ is equal to $Idle_j$ generated using the resource-driven scheduling algorithm described in Chapter 4. The resource-driven scheduling algorithm was designed to schedule repetitive construction activities in two stages. The first provides an initial schedule that satisfies logical precedence relationship and crew availability constraints, and the second achieves further compliance with crew work continuity constraint. For example, the activity schedule developed in stage 1 does not ensure compliance with crew work continuity constraint for crew 1 as shown in Figure 5.3. Crew 1 can finish work in unit 3 on day 6 ($F_3 = 6$), however it cannot start work in its next assigned unit (j = 4) until day 8 because of the logical precedence relationship imposed by the completion of the predecessor activity in the same unit ($PES_4 = 8$). This means that crew 1 has to wait idle for 2 days before it can start work in unit 4 ($Idle_4 = 2$) as shown in Figure 5.3. In order to eliminate crew idle time and maintain crew work continuity constraint, the second stage of the resource-driven scheduling algorithm shifts the start and finish times of the activity in selected units by a duration of $shift_j$ as shown in Figure 5.3.

The schedule developed in stage 2 achieves strict compliance with crew work continuity constraint by delaying the start and finish times of the activity in a number of units. This may cause a delay to successor activities and accordingly an extension to project duration as shown in Figure 5.3. In order to minimize such effect, the strict compliance with crew work continuity applied in stage 2 can be relaxed by considering work interruption. This allows the activity in a number of units to start and finish at an earlier time than that established in stage 2, and therefore can lead to reduction in project duration. As such, interruption can be used to shift the schedule of the activity in selected units to start and finish at an earlier time than that developed in stage 2. It should be noted, however, that the activity in each repetitive unit cannot start at an earlier time than that developed in stage 1 because of the logical precedence relationship and crew availability constraints as shown in Figure 5.3. The Figure also illustrates that the application of an interruption of $Idle_j$ before the start of the activity in each unit j actually transforms the schedule of stage 2 to that of stage 1. This shows that the earliest start and finish times of the activity in each unit (i.e. schedule of stage 1) can be achieved by applying an interruption of $Idle_j$ to the schedule of stage 2 as shown in Figure 5.3. As such, this rule recognizes the fact that the upper limit of interruption days before the start of unit j ($Imax_j$) can be considered equal to $Idle_j$ which is generated using the resource-driven scheduling algorithm described in Chapter 4. Based on this rule and assuming that the interruption before unit j (Int_j) can change by an increment of 1 day (i.e. Int_j = 0, 1, 2 for $Imax_j$ = 2), the total number of interruption vectors that can be considered for crew formation n of activity i

(NIV^i_n) can be calculated as follows:

$$NIV^i_n = \prod_{m=1}^{M} NIV^i_{nm} = \prod_{j=2}^{J} (1 + Idle_j) \qquad (5.2)$$

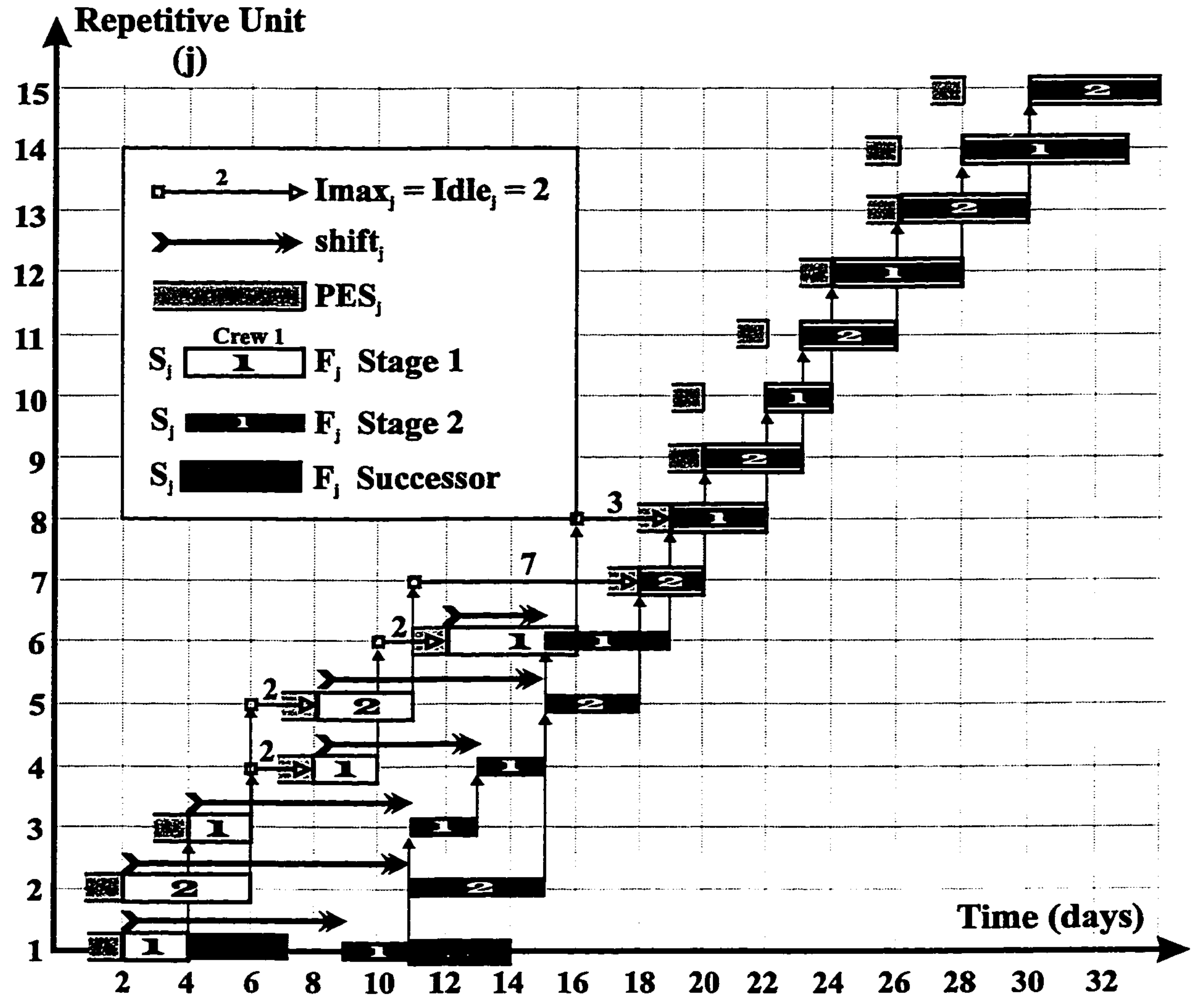


Figure 5.3 Identification of Upper Limit for Interruption (Imax)

The application of this rule transforms the process of generating feasible interruption vectors from an impractical and unlimited problem to a feasible and limited one. For example, an activity such as the one shown in Figure 5.3 can have an unlimited number of possible interruption vectors, however the application of this rule provides an upper limit on the number of interruption days that can be applied before the start of the activity in eah of the 15 repetitive units (i.e. $Imax_j = Idle_j = \{0, 0, 0, 2, 2, 2, 7, 3, 0, 0, 0, 0, 0, 0, 0\}$. This means that the interruption that can be applied before the start of the activity in each unit j (Int_j) in order to minimize the project duration ranges between 0 and $Imax_j$. For this example, the total number of feasible interruption vectors (NIV^i_n) that can be considered is 864 (i.e. $NIV^i_n = (1+2)\ (1+2)\ (1+2)\ (1+7)\ (1+3) = 864$). Comparing this number to that generated earlier considering Imax = 6 (NIV^i_n = 680 billion) clearly illustrates the advantage of this rule in limiting NIV^i_n.

2) A closer examination of the possible interruption vectors generated based on rule 1 indicates that a second rule can be identified to further reduce the total number of interuption vectors. The second rule is based on the fact that if: 1) two interruption vectors (A and B) lead to the same total number of interruption days to a given crew m ($\sum Int_j$ of A = $\sum Int_j$ of B) and 2) vector A results in an earlier than or equal schedule for the activity in all repetitive units than that of vector B, then vector A can be considered a dominant option and accordingly vector B can be discarded from further consideration.

For example, crew 2 in Figure 5.3 can be interrupted before the start of the activity in units 7 and 5 by a duration that ranges from 0 to 7 days and 0 to 2 days, respectively. For crew 2 in this case, 24 possible interruption vectors can be generated based on the first rule as shown in Table 5.2. A closer examination of these vectors indicates that a number of them can be precluded in order to reduce the size of the problem. For example, each of vectors 3 and 17 cause a total of two interruption days to crew 2 as shown in Table 5.2 and Figure 5.4. Vector 3, however, results in an earlier than or equal schedule for the activity in all repetitive units than that of vector 17 as shown in Figure 5.4, and accordingly vector 17 can be discarded from further consideration. Similarly, each of vectors 9 to 15 can be precluded after comparing it to vectors 3 to 8, respectively as shown in Table 5.2. The same can be applied to vectors 17 to 22 after comparing each to vectors 3 to 8, respectively. Also vector 23 can be eliminated after comparing it to vector 16.

Table 5.2 Interruption Vectors for Crew 2

Assigned Repetitive Unit		Interruption Vector																							
	$Idle_j$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
15	0	-	-	-	-	-	-	-	-	▨	▨	▨	▨	▨	▨	▨	-	▨	▨	▨	▨	▨	▨	▨	-
13	0	-	-	-	-	-	-	-	-	▨	▨	▨	▨	▨	▨	▨	-	▨	▨	▨	▨	▨	▨	▨	-
11	0	-	-	-	-	-	-	-	-	▨	▨	▨	▨	▨	▨	▨	-	▨	▨	▨	▨	▨	▨	▨	-
9	0	-	-	-	-	-	-	-	-	▨	▨	▨	▨	▨	▨	▨	-	▨	▨	▨	▨	▨	▨	▨	-
7	7	0	1	2	3	4	5	6	7	▨	▨	▨	▨	▨	▨	▨	7	▨	▨	▨	▨	▨	▨	▨	7
5	2	0	0	0	0	0	0	0	0	▨	▨	▨	▨	▨	▨	▨	1	▨	▨	▨	▨	▨	▨	▨	2
2	0	-	-	-	-	-	-	-	-	▨	▨	▨	▨	▨	▨	▨	-	▨	▨	▨	▨	▨	▨	▨	-

- indiates no interruption due to $Idle_j = 0$

note: shaded area in Table indicate precluded vectors

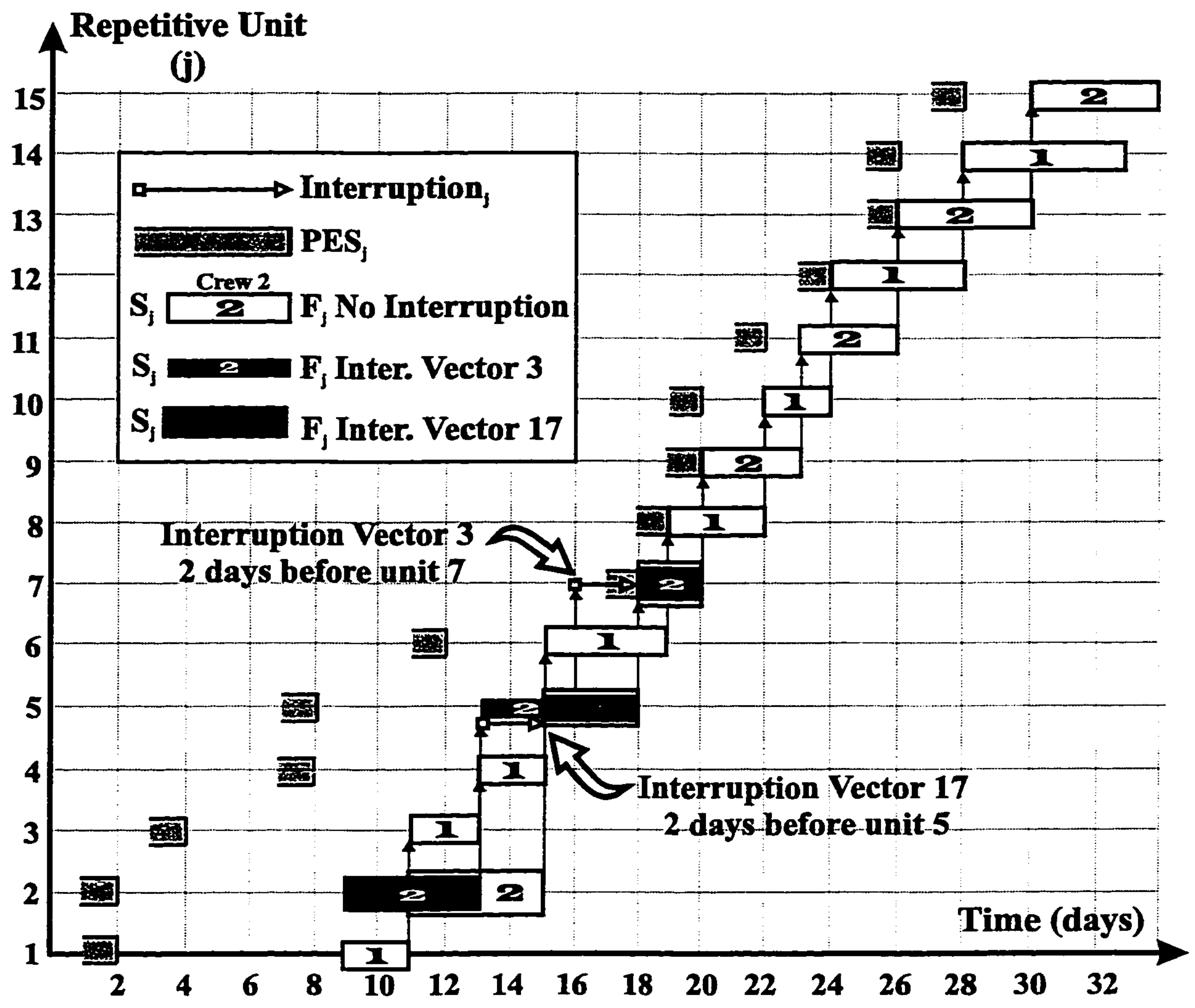


Figure 5.4 Comparing Interruption Vectors 3 and 17 for Crew 2

In general, this rule is based on the fact that if: 1) crew m is scheduled to construct a number of repetitive units that includes units A and B, where the order of executing unit B precedes that of unit A; and 2) crew m can be interrupted by x days before the start of either unit A (option A) or unit B (option

B), then option A provides an earlier than or equal schedule for the activity in all units than that of B. This is true because option A (i.e. interrupting crew m before the start of unit A by x days where $x \leq Idle_A$) leads the activity to start x days earlier in all preceding repetitive units assigned to crew m including unit B, however the opposite is not true as shown in Figure 5.5. Since options A and B cause the same number of interruption days to crew m, and option A provides an earlier than or equal schedule for the activity in all repetitive units than that of option B as shown in Figure 5.5, then option B should be discarded from further consideration.

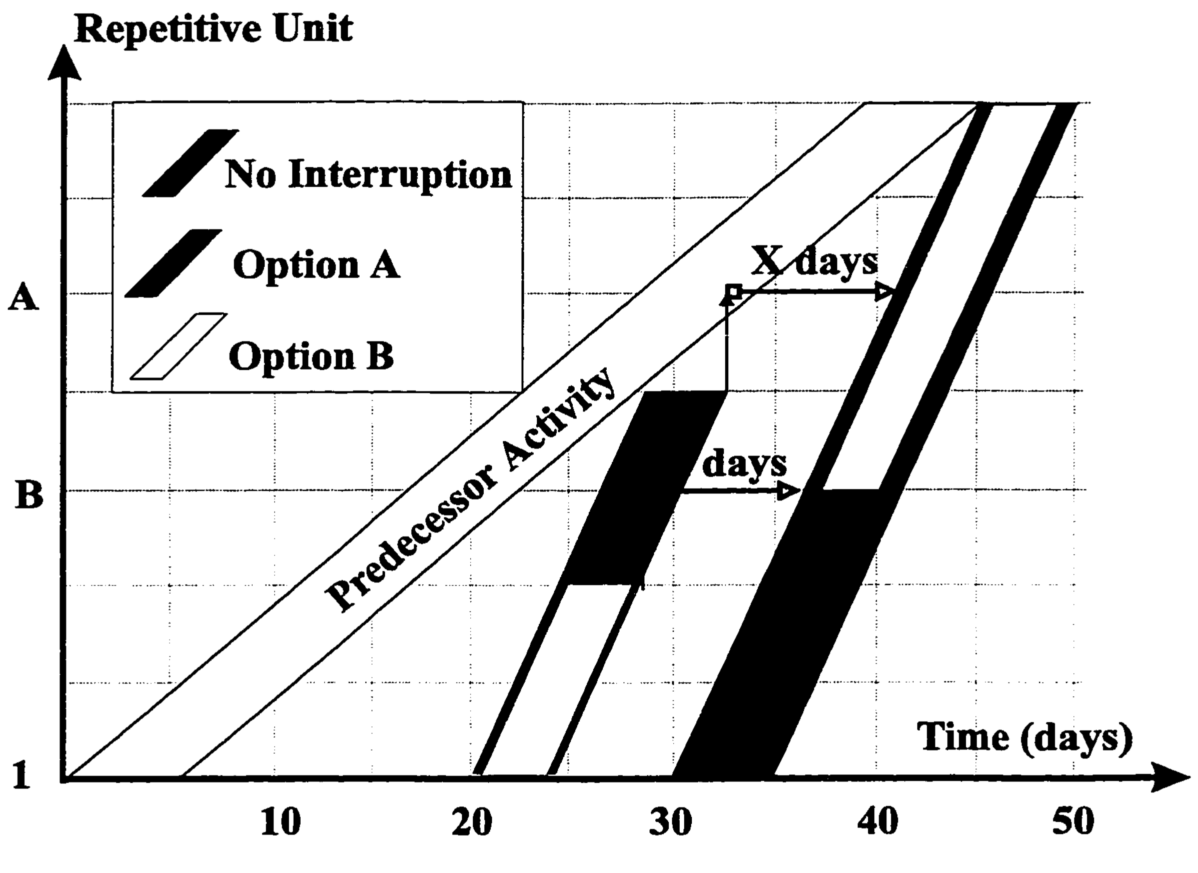


Figure 5.5 Comparing General Interruption Vectors A and B

As such, it is always more effective to interrupt an activity before the start of later assigned units rather than earlier ones. Accordingly, based on this rule and the values of Idle_j recognized in the previous rule, a general form can be identified to generate the needed interruption vectors for crew m that is assigned to construct the activity in a number of repetitive units as shown in Table 5.3.

Table 5.3 Interruption Vectors Based on Rule 2

Unit	Interruption Vectors (days)										
J	0	1	2	...	Idle_J	Idle_J	Idle_J	Idle_J	Idle_J	Idle_J	Idle_J
J-1	0	0	0	0	0	1	...	Idle_{J-1}	Idle_{J-1}	Idle_{J-1}	Idle_{J-1}
...	...	...	...	...	...	...	...	...	...	...	...
2	0	0	0	0	0	0	0	0	1	...	Idle_2
1	0	0	0	0	0	0	0	0	0	0	0

For crew m of crew formation n, the application of this rule transforms the process of identifying the total number of interruption vectors (NIV^i_{nm}) from multiplicative (i.e. $\text{NIV}^i_{nm} = \prod_{j=2}^{J} (1+\text{Idle}_j)_m$ to additive (i.e. $\text{NIV}^i_{nm} = 1 + \sum_{j=2}^{J} (\text{Idle}_j)_m$). This can lead to a significant reduction in the number of interruption vectors that need to be considered for real-life problems. For example, the application of this rule to a crew m that has $\text{Idle}_8 = 9$, $\text{Idle}_6 = 10$,

$Idle_4 = 8$ and $Idle_2 = 6$ reduces its total number of interruption vectors that need to be considered from 6930 (i.e. $NIV^i_{nm} = (1+9)\,(1+10)\,(1+8)\,(1+6) = 6930$) to 34 (i.e. $NIV^i_{nm} = 1 + 9 + 10 + 8 + 6 = 34$). Based on this rule, the total number of interruption vectors that need to be considered for crew formation n of activity i (NIV^i_n) can be calculated as follows:

$$NIV^i_n = \prod_{m=1}^{M} NIV^i_{nm} = \prod_{m=1}^{M} (1+\sum_{j=2}^{J} Idle_j)_m \qquad (5.3)$$

For crew 2 in Figure 5.3, the application of this rule reduces its total number of interruption vectors from 24 (i.e. $NIV^i_{n2} = (1+Idle_7)(1+Idle_5) = (8)(3) = 24$) to 10 (i.e. $NIV^i_{n2} = 1 + Idle_7 + Idle_5 = 1 + 7 + 2 = 10$) as shown in Table 5.2. Similarly for crew 1 in Figure 5.3, the total number of interruption vectors can be reduced from 36 (i.e. $NIV^i_{n1} = (1+Idle_8)(1+Idle_6)(1+Idle_4) = (4)(3)(3) = 36$) to 8 (i.e. $NIV^i_{n2} = 1 + Idle_8 + Idle_6 + Idle_4 = 1 + 3 + 2 + 2 = 8$) as shown in Table 5.4. Accordingly, the application of this rule reduces the total number of interruption vectors that need to be considered for the activity shown in Figure 5.3 from 864 (i.e. NIV^i_n of previous rule) to 80 (i.e. $NIV^i_n = (NIV^i_{n1})\,(NIV^i_{n2}) = (8)(10) = 80$).

3) The third rule that can be used to reduce the number of feasible interruption vectors is using a step greater than 1 day to increment interruption days. For example, the feasible interruption vectors generated based on the previous rule for crew 2 of Figure 5.3 assumes a step of 1 day as shown in Table 5.2.

Increasing this step from 1 to 2 days reduces the total number of interruption vectors for crew 2 (NIV^i_{n2}) from 10 to 6 as shown in Table 5.5. Similarly for crew 1 of Figure 5.3, the increase of the step to 2 days decreases NIV^i_{n1} from 8 to 5 as shown in Table 5.6. Accordingly, this reduces the total number of interruption vectors that can be considered for the activity shown in Figure 5.3 from 80 (i.e. NIV^i_n = (8)(10) = 80) to 30 (i.e. NIV^i_n = (5)(6) = 30). Further increase in the step leads to further reduction in NIV^i_n as follows:

$$NIV^i_n = \prod_{m=1}^{M} NIV^i_{nm} = \prod_{m=1}^{M} \left(1+\sum_{j=2}^{J}\frac{Idle_j}{step}\right)_m \qquad (5.4)$$

Table 5.4 Interruption Vectors for Crew 1 Based on Rule 2

Assigned Unit	Idle$_j$	Interruption Vector (days)							
(j)		(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
14	0	-	-	-	-	-	-	-	-
12	0	-	-	-	-	-	-	-	-
10	0	-	-	-	-	-	-	-	-
8	3	0	1	2	3	3	3	3	3
6	2	0	0	0	0	1	2	2	2
4	2	0	0	0	0	0	0	1	2
3	0	-	-	-	-	-	-	-	-
1	0	-	-	-	-	-	-	-	-

- indicates no interruption because Idle$_j$ = 0

Table 5.5 Interruption Vectors for Crew 2 Using a Step of 2 days

Assigned Unit (j)	$Idle_j$	Interruption Vector (days)					
		(1)	(2)	(3)	(4)	(5)	(6)
15	0	-	-	-	-	-	-
13	0	-	-	-	-	-	-
11	0	-	-	-	-	-	-
9	0	-	-	-	-	-	-
7	7	0	2	4	6	7	7
5	2	0	0	0	0	0	2
2	0	-	-	-	-	-	-

- indicates no interruption because $Idle_j = 0$

Table 5.6 Interruption Vectors for Crew 1 Using a Step of 2 days

Assigned Unit (j)	$Idle_j$	Interruption Vector (days)				
		(1)	(2)	(3)	(4)	(5)
14	0	-	-	-	-	-
12	0	-	-	-	-	-
10	0	-	-	-	-	-
8	3	0	2	3	3	3
6	2	0	0	0	2	2
4	2	0	0	0	0	2
3	0	-	-	-	-	-
1	0	-	-	-	-	-

- indicates no interruption because $Idle_j = 0$

It should be noted, however, that during the optimization process the model will select an optimum interruption vector from the set of generated feasible alternatives. If the optimum interruption vector cannot be found in this set, the closest vector with larger interruption in the set will be selected. As such, the application of this rule ensures the optimum project schedule, however it may provide a schedule that specifies a slightly higher number of interruption days than what is really needed. For example, if a step of 2 days was used for crew 2 (see Table 5.5) and the optimum interruption vector (A) required $Int_7 = 5$ and $Int_5 = 0$, the optimization model will select the closest higher vector (B) from the generated set in Table 5.5 (i.e. $Int_7 = 6$ and $Int_5 = 0$). The selected B vector provides the same project duration as that of A, however it will cause an additional day of interruption than what is really needed (i.e. $\sum_{j=2}^{J} Int_j = 5$ for vector A vector as compared to $\sum_{j=2}^{J} Int_j = 6$ for vector B).

4) The fourth rule is based on the fact that interruption vectors need not be generated and considered for the last and some intermediate activities in the project. The main objective of considering interruption for a repetitive activity is to allow its successor to start at an earlier time in order to reduce project duration as shown in Figure 5.1. Since the last repetitive activity (e.g. activity D in Figure 5.6) has no successors, its interruption does not reduce project duration and therefore it need not be interrupted. Also the interruption of some intermediate activities (e.g. activity C in Figure 5.6) causes their successors to start at a later rather than at an earlier time and therefore they need not be interrupted. In addition, the present model is designed to allow the user to

specify not to interrupt particular repetitive activities in the project for practical considerations. This further reduces the total number of interruption vectors that need to be generated and considered during the optimization process.

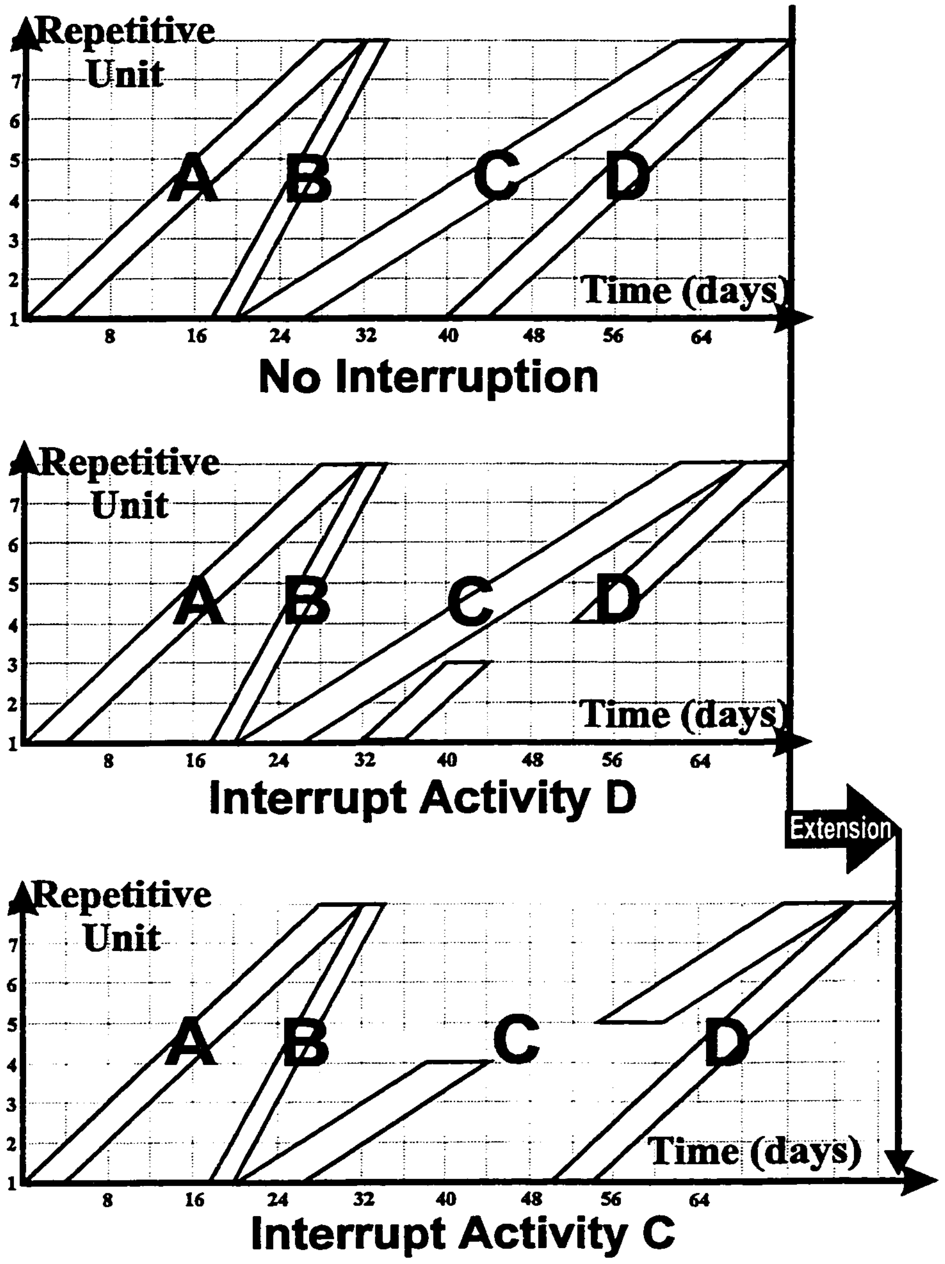


Figure 5.6 Impact of Activity Interruptions on Project Duration

5.4.2 Proposed Interruption Algorithm

The above four rules are used as the basis to develop an algorithm for generating a limited set of interruption vectors for each crew formation C^i_n. For a crew formation C^i_n that consists of m crews working simultaneously, the algorithm is applied in six steps:

1) Based on $Idle_j$ values obtained from the earlier described algorithm for resource-driven scheduling (Chapter 4), calculate the number of interruption vectors that need to be considered for crew m of crew formation C^i_n (NIV^i_{nm}) and the total number of interruption vectors that need to be considered for the entire crew formation C^i_n (NIV^i_n) as follows:

$$NIV^i_{nm} = 1 + \sum_{jm=2}^{J} Idle_{jm} \tag{5.6}$$

$$NIV^i_n = \prod_{m=1}^{M} NIV^i_{nm} \tag{5.7}$$

jm:	repetitive unit j that is assigned to crew m; and
$Idle_{jm}$:	$Idle_j$ values for repetitive unit j that is assigned to crew m.

2) According to the earlier described third rule for limiting NIV^i_n, calculate the required *step* to reduce NIV^i_n to a prespecified smaller number ($MaxNIV^i_n$). In this algorithm, $MaxNIV^i_n$ can be prespecified (e.g. 200 vectors) to keep an upper limit on the size of the problem. Through an iterative process, the

algorithm identifies the required step as the least integer number that satisfies the following inequality relationship:

$$\mathrm{MaxNIV}^{i}_{n} \geq \prod_{m=1}^{M} \mathrm{NIV}^{i}_{nm} \tag{5.8}$$

where, $$\mathrm{NIV}^{i}_{nm} = 1 + \sum_{jm=2}^{J} \frac{Idle_{jm}}{step} \tag{5.9}$$

3) Based on the above identified *step* value, calculate the reduced number of interruption vectors NIV^{i}_{nm} and NIV^{i}_{n} as follows:

$$\mathrm{NIV}^{i}_{nm} = 1 + \sum_{jm=2}^{J} \frac{Idle_{jm}}{step} \tag{5.10}$$

$$\mathrm{NIV}^{i}_{n} = \prod_{m=1}^{M} \mathrm{NIV}^{i}_{nm} \tag{5.11}$$

4) Create a two dimensional interruption array of size J rows x NIV^{i}_{n} columns for crew formation C^{i}_{n}, where each column in the array represents a possible interruption vector r for crew formation C^{i}_{n} as shown in Table 5.7.

5) Initialize the interruption array.

6) Fill in the non-zero elements of the interruption array in order to identify all possible combinations of interruption vectors for crew formation C^{i}_{n}. This is achieved in two stages, starting with the second column of the array as shown

in Table 5.7. In the first stage, interruption vectors are generated for each crew m following the same procedure described in rules 2 and 3 and in similar manner to that summarized in Table 5.5. It should be noted that the generated interruption vectors in the first stage covers the possibility of interrupting only a single crew m of crew formation C^i_n, without considering concurrent interruptions for multiple crews as shown in Table 5.7(A). For example, columns 2 to 5 of stage 1 in Table 5.7(A) provide the possibility of interrupting only crew 1 of Figure 5.3 without interrupting crew 2 of the same figure. Columns 6 to 10 in Table 5.7(A), on the other hand, present the possibility of interrupting only crew 2 of Figure 5.3.

In the second stage, however, the possibility of concurrent interruptions of multiple crews is considered by grenerating all possible combinations of the interruption vectors identified in stage 1. This is achieved by combining each interruption vector of crew m with each interruption vector of other crews working simultaneously in crew formation C^i_n as shown in Table 5.7. For example, the first non-zero interruption vector of crew 1 in stage 1 (column 2 in Table 5.7(A)) is combined with each interruption vector of crew 2 (columns 6 to 10 in Table 5.7) to provide a first set of possible concurrent interruption vectors of crews 1 and 2 as shown in columns 11 to 15 in Table 5.7. As such, stages 1 and 2 of this step provide a limited set of possible interruption vectors for crew formation C^i_n.

Table 5.7(A) Possible Interruption Vectors

Repetitive Unit	$Idle_j$	Assigned Crew	Interruption Array														
			Stage 1										Stage 2				
			Crew 1					Crew 2									
			1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	3	1	0	2	3	3	3	0	0	0	0	0	2	2	2	2	2
7	7	2	0	0	0	0	0	2	4	6	7	7	2	4	6	7	7
6	2	1	0	0	0	2	2	0	0	0	0	0	0	0	0	0	0
5	2	2	0	0	0	0	0	0	0	0	0	2	0	0	0	0	2
4	2	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0
3	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

- indiates no interruption due to $Idle_j = 0$

Table 5.7(B) Possible Interruption Vectors

Repetitive Unit	$Idle_j$	Assigned Crew	Interruption Array														
			Stage 2														
			16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
15	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
14	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
13	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	3	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
7	7	2	2	4	6	7	7	2	4	6	7	7	2	4	6	7	7
6	2	1	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2
5	2	2	0	0	0	0	2	0	0	0	0	2	0	0	0	0	2
4	2	1	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2
3	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

- indiates no interruption due to $Idle_j = 0$

5.5 Optimization Procedure

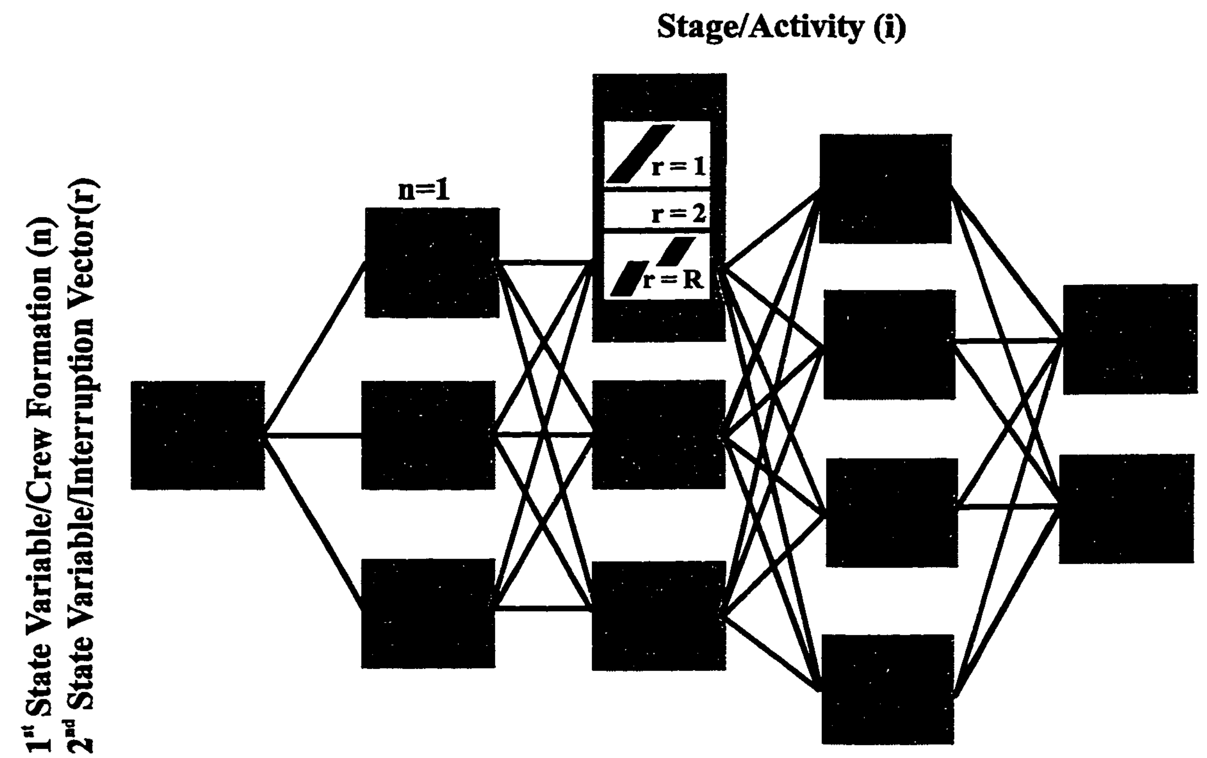
Existing dynamic programming formulations are capable of identifying, from a set of possible alternatives, the optimum crew size for each activity in a repetitive project (Selinger, 1980; and Russell and Caselton, 1988). As stated earlier, the optimization criterion of these formulations is, however, limited to the minimization of the overall duration of the project. While this may lead to the minimization of the indirect cost of the project, it does not guarantee its overall minimum cost. In practice, the overall cost of a project is frequently regarded to be highly important. This is particularly true for contractors performing in a highly competitive environment. The objective of the present optimization procedure is to incorporate cost in the optimization process.

Minimizing the overall project cost is a major priority for the project participants. Therefore, it must be directly considered as an optimization objective rather than indirectly as a byproduct of the minimization of the project duration (Selinger, 1980; and Russell and Caselton, 1988). This is important because while reducing a project duration reduces its indirect cost, this reduction may increase its direct cost, which in many cases results in higher overall project cost. In fact, accelerating a project can be achieved by expediting a number of selected activities and/or introducing interruption options to crew work continuity. Expediting selected activities can be achieved by increasing the crew size or introducing an overtime policy, that could be coupled with impact costs resulting from higher premium rates for overtime hours and reduced productivity due to site congestion or crowding and overtime effect. The interruption option is also associated with additional costs resulting from idle crew time and disruption of learning curve effect.

The problem of minimizing the overall cost of repetitive projects is one of establishing the delicate balance between the reduction in indirect costs as a result of time reduction, and the additional direct costs that may result from such acceleration. To enable a comparison among alternative crew formations, a procedure similar to that used in time-cost trade-off analysis (Ahuja, 1984) is followed. The time is considered to have a money value that is represented by the project indirect cost per day. Hence, any time reduction in project duration can be translated to savings in project indirect costs, and can easily be compared to the additional direct costs arising from the time reduction or schedule acceleration. This analysis is used in the local optimization problem to identify the local optimum predecessor crew formation and associated interruption vector during the dynamic programming procedure, if the user-specified optimization criterion is to minimize the project overall cost.

The problem of selecting the optimum crew formation and interruption vector for each repetitive activity in the project is formulated as a dynamic programming problem that consists of I stages and two state variables as shown in Figure 5.7.

The repetitive activities of the project are represented by I stages. For each stage (i.e. repetitive activity) there are two possible state variables (i.e. crew formations and interruption vector) as shown in Figure 5.7. For each repetitive activity in this model, the user is required to specify the possible alternatives for crew formations, but interuption vectors are automatically generated by the model (see section 5.4.2). The objective of the optimization procedure is to identify for each repetitive activity (i.e. stage) the optimum crew formation (1st state variable) and the optimum interruption vector (2nd state variable) from a set of possible alternatives. The dynamic programming procedure is executed in two paths: first in a forward then in a backward path.


Stage/Activity (i)
n=1
r = 1
r = 2
r = R
1st State Variable/Crew Formation (n)
2nd State Variable/Interruption Vector(r)

Recursive Relationship: $COC^{i}_{n}* = \min [DC^{i}_{n} + IC^{i}_{n} + COC^{*i-1}_{n}]$

Figure 5.7 Dynamic Programming Formulation

5.5.1 Forward Path

The forward path consists of a local optimization problem that is repeated for each pair of crew formation n and interruption vector r associated with each repetitive activity i (C^i_{nr}). The local optimization problem is applied for each C^i_{nr}, starting with the second and ending with the last repetitive activity i as shown in Figure 5.7. In the local optimization problem, for a given crew formation n and an associated interruption vector r for an activity i (C^i_{nr}), the impacts of all possible crew formations and associated interruption vectors of the predecessor activity on C^i_{nr} are compared, and the local optimum crew formation and associated interruption vector of the predecessor C^{i-1}_{nr}, that provides the minimum overall cost of C^i_{nr}, is selected (Moselhi and El-Rayes, 1993(a) and (b)).

In the present object-oriented model, the local optimization problem is designed as a number of functions inside the *Crew-Formation* class. These functions are executed each time a repetitie activity receives a message from its predecessor invoking it to proceed with forward path optimization. This message contains information concerning: a) possible start time of the receiving activity in each repetitive unit j (PES_j) based on the completion time of the predecessor activity and the precedence relationship type and lag; b) cumulative direct cost of the predecessor activities (PDC^i_{nr}); and c) predecessor crew formation and associated interruption vector (C^{i-1}_{nr}). Each time a repetitive activity receives this message, it invokes the forward optimization functions inside *Crew-Formation* class. As shown in Figure 5.8, the forward path of the optimization procedure is performed in the following steps:

1) For each crew formation n of activity i (C^i_n), perform loop 1 in Figure 5.8 which consists of the following substeps:

1.1) Based on the composition of crew formation C^i_n and the PES_j vector received from the predecessor, calculate finish time of the activity in each repetitive unit (F^i_{nj}) as well as $Idle^i_{nj}$ vector, using the resource driven scheduling algorithm described earlier in Chapter 4.

1.2) Examine if user-specification allows interruption for C^i_n and if the current activity i is not the last repetitive activity in the project (i = I). If the two conditions are true proceed with the next step. If it is not, create an artificial interruption vector r where all its elements are equal to zero, indicating that no interruption is allowed, and then skip the next step and move on to step 2.1). It should be noted that as stated in the earlier described fourth rule, the last repetitive activity in the project need not be interrupted since its interruption does not minimize overall project duration (see Figure 5.6). Accordingly and in compliance with the fourth rule, the optimization procedure does not generate interruption vectors for all crew formations of the last repetitive activity in the project in order to limit the size of the optimization problem.

1.3) Based on Idle^i_{nj} vector, generate all feasible interruption vectors for crew formation C^i_n, using the earlier described interruption algorithm (section 5.4.2).

2) For each interruption vector r generated in step 1.3) (C^i_{nr}), perform the nested loop 2 of Figure 5.8 which consists of the following substeps:

2.1) Based on the pair of crew formation n and interruption vector r, and PES_j vector, calculate the finish time for activity i in each repetitive unit j (F^i_{nrj}), using the earlier described algorithm for resource-driven scheduling (Chapter 4). Then calculate the direct cost DC^i_{nr} of crew formation n and interruption vector r C^i_{nr} as follows:

$$DC^i_{nr} = \sum_{j=1}^{J} (MC^i_n . Q^i_j + LC^i_n . D^i_{nj} + EC^i_n . D^i_{nj}) + RC^i_{nr} \qquad (5.12)$$

where,

Q^i_j:	quantity of work of activity i in repetitive unit j;
D^i_{nj}:	duration of repetitive unit j in days for crew formation C^i_n;
MC^i_n:	material cost rate in \$/unit of measurement for crew formation C^i_n;
LC^i_n:	labor cost rate in \$/day for crew formation C^i_n;
EC^i_n:	equipment cost rate in \$/day for crew formation C^i_n; and
RC^i_{nr}:	interruption cost in \$ for C^i_{nr}.

Calculate the cumulative direct cost of the predecessor activities PDC^i_{nr}. This value is obtained by adding the direct cost of the predecessor crew formation and associated interruption vector DC^{i-1}_{nr}, and the cumulative direct cost of all its local optimal predecessors as follows:

$$PDC^i_{nr} = DC^{i-1}_{nr} + PDC^{i-1}_{nr*} \qquad (5.13)$$

Calculate the indirect cost IC^i_{nr} of the project up to crew formation C^i_n as follows:

$$IC^i_{nr} = ICR.F^i_{nr(j=J)} \qquad (5.14)$$

where,

ICR :	indirect cost rate in \$/day of the project;
and $F^i_{nr(j=J)}$:	finish date of the last section (i.e. $j = J$) for crew formation and associated interruption vector C^i_{nr}.

Then, calculate the cumulative overall cost COC^i_{nr} of the project up to crew formation n and associated interruption vector r C^i_{nr} as follows :

$$COC^i_{nr} = DC^i_{nr} + PDC^i_{nr} + IC^i_{nr} \qquad (5.15)$$

2.2) Examine if interruption vector r has already been generated and stored in a data file for crew formation C^i_n. If it has not been generated in previous loops, store it at the end of the data file as a new possible pair of crew formation n and interruption vector r for activity i (C^i_{nr}), along with finish time vector (F^i_{nrj}) and cumulative overall cost (COC^i_{nr}) obtained from the previous step. Since this pair (C^i_{nr}) has not been stored before in the data file, identify its predecessor pair of crew formation and interruption vector (C^{i-1}_{nr}) as the local optimum predecessor pair and store them in the data file. This initial assumption of local optimum predecessors can be modified in later iterations of loops 1 and 2, if new pairs of predecessors are found to provide better results for the optimization criterion than that of the stored ones.

If interruption vector r has already been stored in the data file in a previous iteration, compare either the value of COC^i_{nr} or F^i_{nrJ} obtained in step 2.1) of the current iteration to that stored in the data file. If COC^i_{nr} or F^i_{nrJ} of the current iteration is less than that stored, identify and replace in the data file the current value as the local optimum solution and its predecessor pair of crew formation and interruption vector as the new local optimum predecessor pair C^{i-1}_{nr}. It should be noted that if the user-specified optimization criterion is to minimize project overall cost compare the two COC^i_{nr} values, however if the criterion is to minimize project duration F^i_{nrJ} values are used instead.

The above loop 1 and nested loop 2 in Figure 5.8 are repeated each time activity i receives a message from its predecessor invoking it to proceed with forward path optimization. This message is sent to activity i from each pair of crew formation n and interruption vector r of the predecessor activity C^{i-1}_{nr}. After receiving messages from all possible pairs of C^{i-1}_{nr} of the predecessor and applying the above described forward path procedure, a limited set of possible interruption vectors is created and stored in a data file for each crew formation n of activity i C^i_n. In addition, for each pair of crew formation and interruption vector C^i_{nr}, the local optimum predecessor pair C^{i-1}_{nr}, that yields either the minimum overall cost or duration for the pair C^i_{nr} is identified and stored in the data file, to be recalled later for determining the global optimum solution for the project.

5.5.2 Backward Path

The procedure in this path involves no computations and its main objective is to scan the local optimum conditions, identified and stored in the forward path, and select the optimum crew formation and interruption vector for each repetitive activity in a backward path from the last activity to the first. First, this is done by selecting the global optimum crew formation and associated interruption vector for the last repetitive activity, which yields the minimum overall cost or duration of the project. This cost or duration, being cumulative, represents the overall cost or duration of the project. Then starting from the last activity and tracing backwards, the predecessor local optimum crew formation and associated interruption vector C^{i-1}_{nr}, that lead to this global optimum condition is identified to be the global optimum crew formation for the predecessor. This process of recalling, and scanning the predecessor local optimum crew formation and interruption vector propagates backwards towards the first repetitive activity in the project.

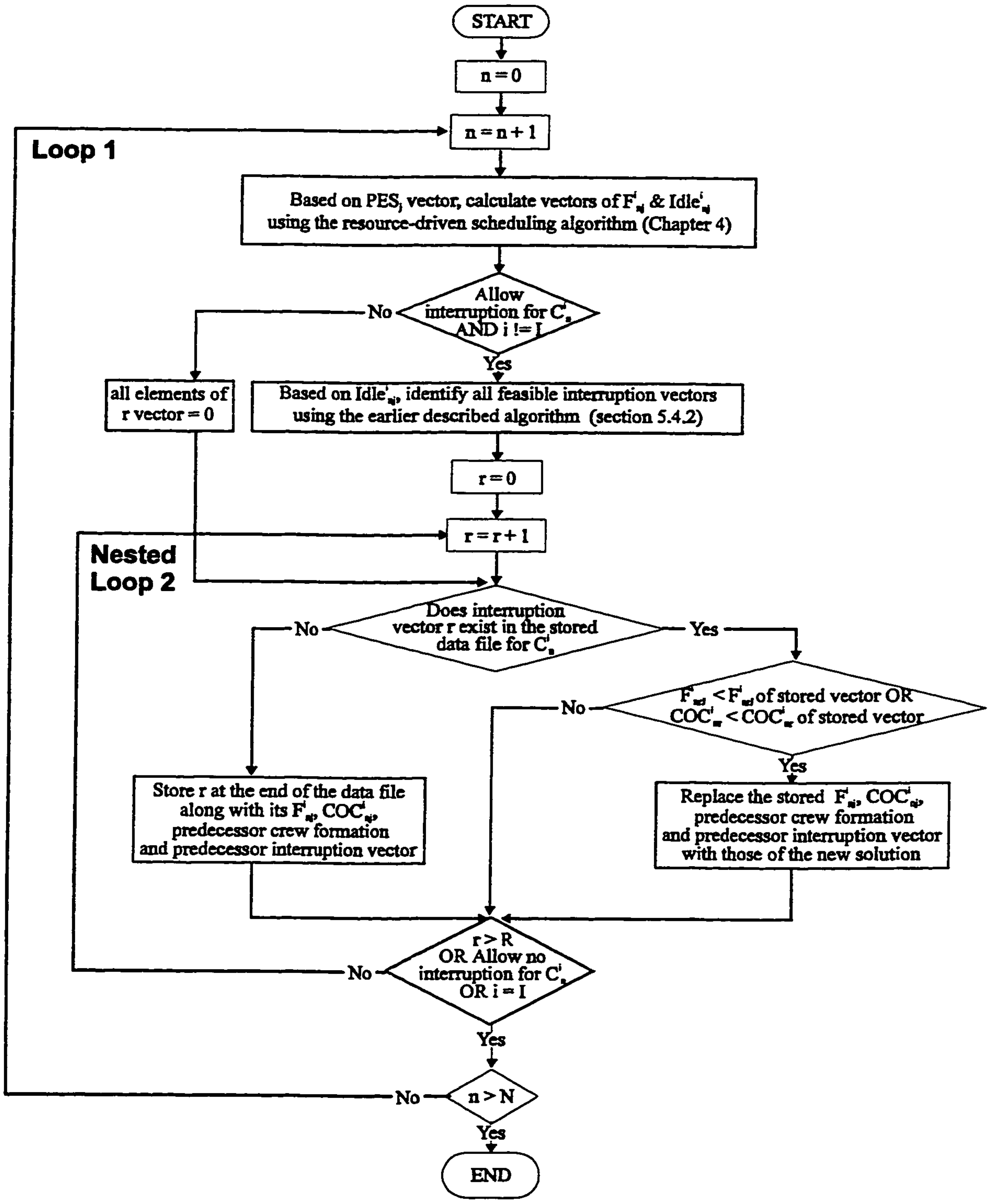

START
n = 0
n = n + 1
Loop 1
Based on PES$_j$ vector, calculate vectors of F$^i_{sj}$ & Idle$^i_{sj}$ using the resource-driven scheduling algorithm (Chapter 4)
Allow interruption for C^i_s AND i != I
No
Yes
all elements of r vector = 0
Based on Idle$^i_{sj}$, identify all feasible interruption vectors using the earlier described algorithm (section 5.4.2)
r = 0
r = r + 1
Nested Loop 2
Does interruption vector r exist in the stored data file for C^i_s
No
Yes
F$^i_{srj}$ < F$^i_{srj}$ of stored vector OR COC$^i_{sr}$ < COC$^i_{sr}$ of stored vector
No
Yes
Store r at the end of the data file along with its F$^i_{sj}$, COC$^i_{sj}$, predecessor crew formation and predecessor interruption vector
Replace the stored F$^i_{sj}$, COC$^i_{sj}$, predecessor crew formation and predecessor interruption vector with those of the new solution
r > R OR Allow no interruption for C^i_s OR i = I
No
Yes
n > N
No
Yes
END

Figure 5.8 Forward Path of the Optimization Procedure

5.6 Numerical Example

The 3-span concrete bridge example, originally introduced by Selinger (1980), is considered. A few modifications are introduced in order to illustrate the capabilities of the present model and validate its results. The project consists of four similar sections or units, each includes the following repetitive activities: excavation, foundations, columns, beams, and slabs. Each repetitive activity is performed by a single crew progressing from the first to the fourth section sequentially as shown in Figure 5.9. The logical precedence relationships among succeeding activities are finish to start with no lag time. No cost data were included in the original example and the objective of the optimization process, then, was to minimize the overall duration of the project.

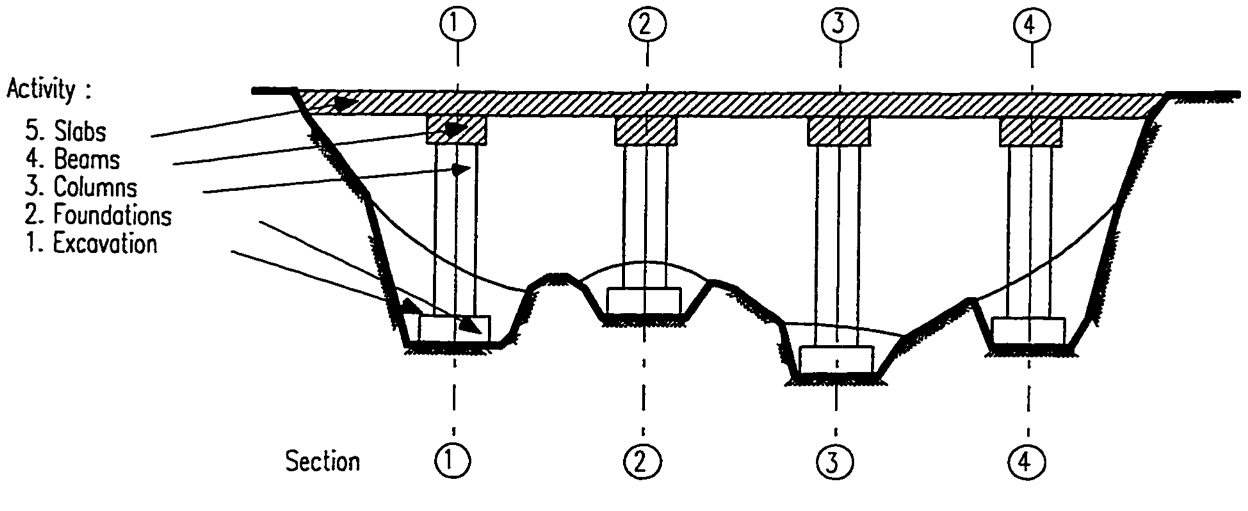


Figure 5.9 Bridge Example (Selinger 1980)

In order to incorporate cost in the optimization process, and in an effort to use realistic data, crew daily output and direct cost rates for each activity are obtained from Means Construction Cost Data (1991) as indicated by the activity code numbers listed in the second column of Table 5.8, and the indirect costs of the project are assumed to be $1000/day (i.e. approximately 10% of the direct cost). The base crew data presented in Table 5.8 are produced by transforming Means labor and equipment cost rates from $/unit of measurement to $/day in compliance with the required input data for the present model (Equation 5.12). To enable a comparison with Selinger's formulation (1980), output rates are used to calculate quantities of work in such a way to reproduce identical durations to those originally used by Selinger (1980). As shown in Table 5.8, work quantities are expressed in units of measurement (i.e. cubic yard), and hence could be used as a direct input to the present model (Equation 5.12).

For each activity, a set of possible crew formations is introduced as shown in Table 5.9. For each set, the base crew data presented in Table 5.8 is considered to represent the normal productivity achieved by a normal crew formation, accelerated daily outputs of other crew formations are obtained by increasing the normal output using the same ratios reported in the original example (Selinger 1980), as shown in the third and fourth columns of Table 5.9. For estimating the cost rates of these crew formations, it is assumed that the increased output is achieved by assigning overtime hours to the base crew. The overtime hours OT^i_n

shown in the fifth column, are in addition to the regular daily 8 working hours, and are calculated based on the following equation:

$$OT^i_n = (8.\frac{P^i_n}{BP^i}) - 8 \qquad (5.16)$$

where,

P^i_n : daily output (in c.y./day) for crew formation n of activity i including overtime;

BP^i : daily output of base crew (c.y./day) without overtime for activity i.

In this numerical example, the applcation of equation 5.16 assumes that crew productivity is similar in both regular and overtime working hours. While overtime policies can exhibit different premiums for the different crews and equipment and can readily be accounted for in the present model, for simplicity, an overtime premium of 50% is assumed in the present example for direct labor and equipment rates. The labor and equipment cost rates (LC^i_n and EC^i_n) in $/day including overtime premiums (eighth and tenth columns of Table 5.9) are calculated based on the following equations :

$$LC^i_n = BLC^i + (\frac{BLC^i}{8}.(1.5).OT^i_n) \qquad (5.17)$$

$$EC^i_n = BEC^i + (\frac{BEC^i}{8}.(1.5).OT^i_n) \qquad (5.18)$$

where,

BLC^i :	labor cost rate of base crew (in \$/day) without overtime premiums;
and BEC^i :	equipment cost rate of base crew (in \$/day) without overtime premiums for each activity i as presented in Table 5.8.

In order to illustrate the use of the present model, demonstrate its capabilities and validate its results, the numerical example is analyzed twice. The first analysis provides a least cost solution and the second presents a minimum duration solution for the project.

5.6.1 Least Cost Solution

This solution is obtained by specifying that the optimization criterion is to minimize the project overall cost. The objective of this solution is to illustrate the use of the model and show the significance of incorporating cost in the optimization procedure. The optimum solution is determined based on the dynamic programming procedure described earlier. As stated, the solution is performed in two paths: a forward path to identify the local optimum predecessor pair of crew formation and interruption vector, and a bakward path to determine the overall optimum pair of crew formation and interruption vector for each activity at the project level.

In this solution, no activity interruptions are considered in order to allow for a comparison with the results available in the literature (Selinger 1980). As such, the second state variable (i.e. interruption vectors) need not be considered in this solution. This means that the forward and backward paths of this analysis focus only on crew formations and does not consider interruption vectors. The forward stage starts by calculating the cost associated with the planned start and finish dates for the first crew formation of the first activity (excavation) as shown in Table 5.10. The local optimization problem is, then, repeated for each possible crew formation associated with each activity, starting from foundation and ending with slabs. In each iteration, the impact of each possible predecessor crew formation on the overall cost is calculated. These costs are then compared and the predecessor crew formation yielding the minimum overall cost is identified as the local optimum predecessor.

For each crew formation of the foundation activity, there is only one possible predecessor crew formation, and thus it is considered to be the local optimum predecessor as shown in Table 5.10. For crew formation 1 of the columns activity, however, there are three possible predecessors (crew formations 1, 2, and 3 of foundation). As shown in Table 5.10, the impact of each predecessor on the planned dates and costs is calculated as described earlier in the forward path of the optimization procedure. The overall cost of the three alternatives are compared and the predecessor yielding the minimum overall cost is identified as the local optimum predecessor. In this case, crew formation 3 of foundation activity is identified to be the local optimum predecessor for crew formation 1 of columns. Similarly, the local optimization problem is performed for crew formations 2 and 3 of the columns activity as shown in Table 5.10. For the two remaining activities (beams and slabs), the local optimization problem is repeated for each crew formation in a similar manner. The local optimum predecessor and its impact on the dates and costs of each crew formation are stored as shown in Table 5.11, to be recalled later during the backward stage.

In the backward stage, crew formations are scanned starting off with the last activity (i.e slabs), and progressing backwards to beams, columns, foundations, and excavation. The least project overall cost, represented by the cumulative overall cost assigned to the last activity, identifies crew number 2 as the overall optimum crew formation for slabs. The local optimum predecessor crew formation number 4 associated with this option is traced backwards to identify the overall optimum option for activity beams. The scanning progresses backwards, in a similar manner to identify the overall optimum crew formations for all remaining activities as shown in Table 5.11.

Table 5.8 Basic Input Data

Repetitive activity (1)	Quantities in Cubic Meter				Base Crew Data			
	Section 1 (2)	Section 2 (3)	Section 3 (4)	Section 4 (5)	Output in m^3/day (6)	Material cost in \$/m^3 (7)	Labor cost in \$/day (8)	Equipment cost in \$/day (9)
Excavation	1147	1434	994	1529	91.75	0	340	566
Foundation	1032	1077	943	898	53.86	92	1902	436
Columns	104	86	129	100	5.73	479	1875	285
Beams	85	92	101	80	5.66	195	1850	148
Slabs	0	138	114	145	7.76	186	1878	149

Table 5.9 Possible Crew Formations' Data

Repetitive activity (1)	Crew form-ation (2)	Selinger crew size (3)	Output in m^3/day (4)	Over-time in hrs (5)	Material cost in \$/m^3 (6)	Labor cost in \$/day (7)	Equipment cost in \$/day (8)
Excavation	1	6	91.75	0.00	0	340	566
Foundation	1	10	89.77	5.33	92	3804	874
	2	8	71.81	2.67	92	2853	655
	3	6	53.86	0.00	92	1902	436
Columns	1	10	5.73	0.00	479	1875	285
	2	12	6.88	1.60	479	2438	371
	3	14	8.03	3.20	479	3000	456
Beams	1	7	9.90	6.00	195	3931	315
	2	6	8.49	4.00	195	3238	259
	3	5	7.07	2.00	195	2544	204
	4	4	5.66	0.00	195	1850	148
Slabs	1	9	8.73	1.00	186	2230	177
	2	8	7.76	0.00	186	1878	149

Table 5.10 Calculation of the Local Optimization Problem

Repetitive activity (1)	Crew formation (2)	Predecessor crew formation (3)	Early Start in Working Days: Section 1 (4)	Section 2 (5)	Section 3 (6)	Section 4 (7)	Finish (8)	Indirect cost in $ (9)	Own direct cost in $ (10)	Cumulative predecessor direct cost in $ (11)	Overall cost in $ (12)	Local optimum predecessor (13)
Excavation	1	-	0.0	12.5	28.1	39.0	55.6	55625	50397	-	106022	-
Foundation	1	1	21.6	33.1	45.1	55.6	65.6	65624	567433	50397	683453	1
	2	1	13.7	28.1	43.1	56.2	68.7	68751	554555	50397	673702	1
	3	1	12.5	31.7	51.7	69.2	85.8	85829	533063	50397	669288	1
Columns	1	1	33.1	51.2	66.3	88.8	106.3	106325	359046	617830	1083200	3
		2	28.1	46.2	61.3	83.8	101.3	101325	359046	604951	1065322	
		3	36.0	54.1	69.2	91.7	109.2	109165	359046	583459	1051669	
	2	1	33.1	48.2	60.7	79.5	94.1	94125	372283	617830	1084237	3
		2	28.6	43.7	56.2	75.0	89.6	89586	372283	604951	1066820	
		3	41.5	56.6	69.2	87.9	102.5	102498	372283	583459	1058240	
	3	1	33.1	46.0	56.8	72.9	85.4	85411	381634	617830	1084873	3
		2	32.5	45.4	56.2	72.3	84.8	84824	381634	604951	1071408	
		3	46.0	59.0	69.7	85.8	98.3	98305	381634	583459	1063397	

Table 5.11 Calculation of the Optimum Solution using the Proposed Model

Repetitive activity (1)	Crew formation (2)	Early Start in Working Days					Indir-ect cost in $ (8)	Own direct cost in $ (9)	Cumulative predecessor direct cost in $ (10)	Overall cost in $ (11)	Local optimum predecessor (12)
		Section 1 (3)	Section 2 (4)	Section 3 (5)	Section 4 (6)	Finish (7)					
Excavation	1**	0.0	12.5	28.1	39.0	55.6	55625	50397	-	106022	-
Foundation	1	21.6	33.1	45.1	55.6	65.6	65624	567433	50397	683453	1
	2	13.7	28.1	43.1	56.2	68.7	68751	554555	50397	673702	1
	3**	12.5	31.7	51.7	69.2	85.8	85829	533063	50397	669288	1
Columns	1**	36.0	54.1	69.2	91.7	109.2	109165	359046	583459	1051669	3
	2	41.5	56.6	69.2	87.9	102.5	102498	372283	583459	1058240	3
	3	46.0	59.0	69.7	85.8	98.3	98305	381634	583459	1063397	3
Beams	1	81.1	89.7	99.0	109.2	117.2	117196	222702	942505	1282402	1
	2	76.5	86.5	97.3	109.2	118.5	118534	216710	942505	1277748	1
	3	69.9	81.9	94.9	109.2	120.4	120408	208320	942505	1271232	1
	4**	60.5	75.5	91.7	109.5	123.6	123590	195673	942505	1261768	1
Slabs	1	-	94.7	110.5	123.6	140.2	140228	183441	1138178	1461846	4
	2**	-	91.7	109.5	124.2	142.9	142935	177687	1138178	1458799	4

** Overall optimum crew formation

In order to study the sensitivity of the minimum overall cost solution to the project indirect cost, the same numerical example is analyzed after setting the project indirect cost at different values ranging from \$500/day to \$4000/day. For each project indirect cost, a set of overall optimum crew formations is identified for the five repetitive activities of the project. As shown in Table 5.12, three different sets of optimum crew formations are obtained in this analysis. The first set (1,3,1,4,2) is obtained when the project indirect cost is within the range of \$500/day to \$2000/day. The second (1,2,2,4,1) and the third (1,2,3,3,1) sets are obtained when the project indirect costs are within the ranges of \$2500/day to \$3500/day, and \$4000/day or more, respectively. As expected, the third set of overall optimum crew formations is identical to that obtained by Selinger (1980). For this numerical example, the minimum overall cost solution coincides with the minimum duration solution when the project indirect cost is \$4000/day or more (i.e. approximately 34% or more of the direct cost). This illustrates that the minimum duration solution provided by existing dynamic programming models does not guarantee the minimum overall cost solution.

To enable a cost comparison between the present model and existing ones, the cost data assumed in the present example were used to estimate the overall project cost based on the optimum crew formations identified by Selinger (1980) and Russell and Caselton (1988) as shown in Tables 5.13 and 5.14. For simplicity, the costs of the interruption vectors reported in the numerical example of Russell and Caselton (1988) are neglected and considered not available.

Table 5.12 Impact of Project Indirect Cost

Project indirect cost in $/day (1)	Optimum crew formations (2)	Project duration in days (3)	Project indirect cost in $ (4)	Project direct cost in $ (5)	Project overall cost in $ (6)
500	1,3,1,4,2	142.93	71468	1315865	1387332
1000	1,3,1,4,2	142.93	142935	1315865	1458799
1500	1,3,1,4,2	142.93	214402	1315865	1530266
2000	1,3,1,4,2	142.93	285869	1315865	1601733
2500	1,2,2,4,1	123.56	308899	1356348	1665247
3000	1,2,2,4,1	123.56	370679	1356348	1727026
3500	1,2,2,4,1	123.56	432459	1356348	1788806
4000	1,2,3,3,1	117.88	471527	1378345	1849872

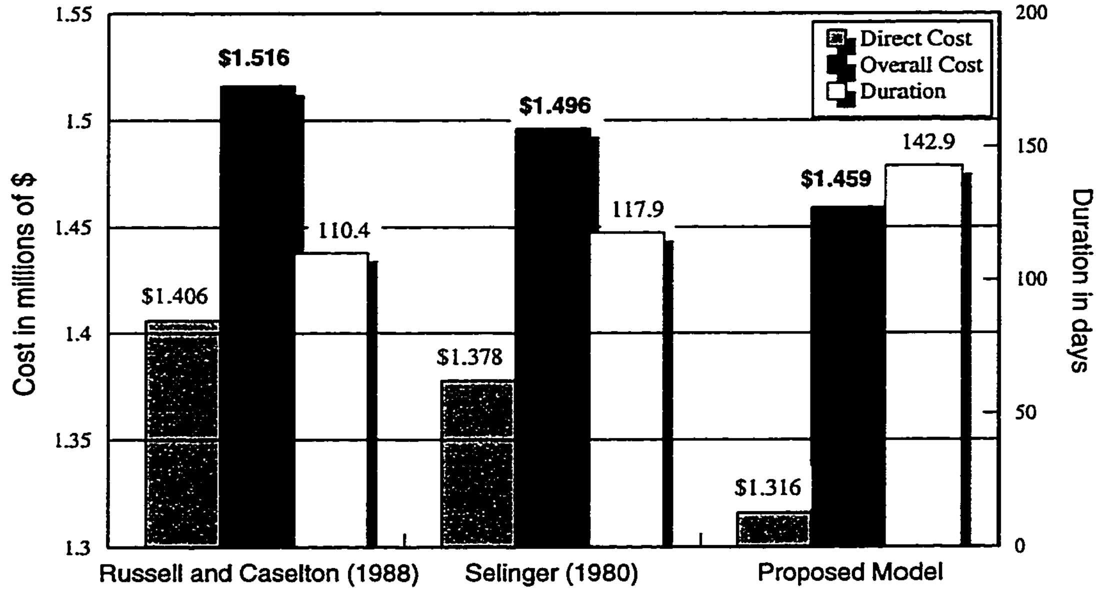
Table 5.13 The Optimum Solution of Selinger (1980)

Repetitive activity (1)	Crew formation (2)	Early start in working days					Indirect cost in $ (8)	Own direct cost in $ (9)	Cumulative predecessor direct cost in $ (10)	Overall cost in $ (11)
		Section 1 (3)	Section 2 (4)	Section 3 (5)	Section 4 (6)	Finish (7)				
Excavation	1	0.0	12.5	28.1	39.0	55.6	55625	50397	-	106022
Foundation	2	13.7	28.1	43.1	56.2	68.7	68751	554555	50397	673702
Columns	3	32.6	45.5	56.2	72.3	84.8	84824	381634	604951	1071409
Beams	3	47.3	59.3	72.3	86.6	97.9	97862	208320	986585	1292767
Slabs	1	-	72.3	88.1	101.2	117.9	117882	183441	1194904	1496227

Table 5.14 The Optimum Solution of the Two-State-Variable Formulation (Russell and Caselton, 1988)

Repetitive activity (1)	Crew form-ation (2)	Early start in working days					Indirect cost in $ (8)	Own direct cost in $ (9)	Cumulative predecessor direct cost in $ (10)	Overall cost in $ (11)
		Section 1 (3)	Section 2 (4)	Section 3 (5)	Section 4 (6)	Finish (7)				
Excavation	1	0.0	12.5	28.1	39.0	55.6	55625	50397	-	106022
Foundation	1	17.6	29.1	41.1	55.6	65.6	65624	567433	50397	683453
Columns	3	29.1	42.1	52.8	68.9	81.4	81400	381634	617830	1080864
Beams	1	43.0	55.6	68.9	83.0	91.0	91000	222702	999464	1313166
Slabs	1	-	64.9	80.7	93.7	110.4	110400	183441	1222166	1516000

As expected, the optimum project duration and cost obtained by the three models were different, each identifying a unique combination of optimum crew formations. A comparison of these results is shown in Figure 5.10. It is clear that the use of any of the three models may lead the contractor to formulate a different crew formation strategy to execute the project. While the two previous dynamic programming formulations can result in minimum project duration, they do not ensure minimum overall cost as demonstrated in the example. Unlike these previous formulations, the present model incorporates cost as a decision variable in the optimization process and allows for the minimization of the project overall cost. This should prove useful to owners and contractors alike, and contribute to cost-effective delivery of constructed projects.



Note: Project indirect cost considered in this solution = $1000/day

Figure 5.10 Least Cost Solution

5.6.2 Minimum Duration Solution

This solution is obtained by specifying that the optimization criterion in the model is to minimize project duration. In order to validate the results of the present model, the current example was analyzed after specifying that activity interruptions are not allowed similar to the assumptions of the Selinger model (1980). The results were identical to those obtained by Selinger (1980). The validation process utilized here is to verify whether or not the developed algorithms can duplicate results generated by others, considering the same assumptions and conditions.

A second minimum duration solution was performed, considering interruption vectors, for the same numerical example in order to illustrate the capabilities of the present model. In the literature, a minimum duration solution has been provided for the same numerical example by two dynamic programming formulations: Selinger (1980) and Russell and Caselton (1988) as shown in Figure 5.11. The first formulation did not consider interruption and provided a minimum project duration of 117.9 days. Russell and Caselton (1988) provided an expanded formulation in order to allow the consideration of a prespecified set of interruption vectors in the optimization process. The latter formulation provided an improvement in minimizing project duation to 110.4 days at the expense of interrupting crew work continuity for 16 days as shown in Figure 5.11. Unlike available dynamic programming formulations that considers a prespecified set of interruption vectors as a second state variable (Russell and Caselton 1988, and Eldin and Senouci

1993), the present model does not require the user to arbitrarly specify interruption. Instead, the interruption vectors are generated automatically in the model, using the interruption algorithm described in section 5.4.2. As shown in Figure 5.11, this provided a better solution for the same numerical example, leading to: 1) further reduction in project duration beyond the minimum solutions provided by others (Selinger 1980, and Russell and Caselton 1988); and 2) less interruption days than those obtained using the formulation of Russell and Caselton (1988).

In addition to providing a superior solution than that of existing dynamic programming formulations, the model provides an added practicality and advantage in three main aspects. First, the incorporated algorithm in the model is automated and therefore the construction planner need not provide a prespecified set of interruption vectors prior to scheduling. Second, it generates a limited number of feasible interruption vectors and therefore enables a practical and feasible approach for the consideration of interruptions during schedule optimization. Third, the algorithm generates all needed interruption vectors during schedule calculation rather than being limited by a prespecified Imax value prior to scheduling, ensuring the generation and selection of the optimum solution as shown in Figure 5.11.

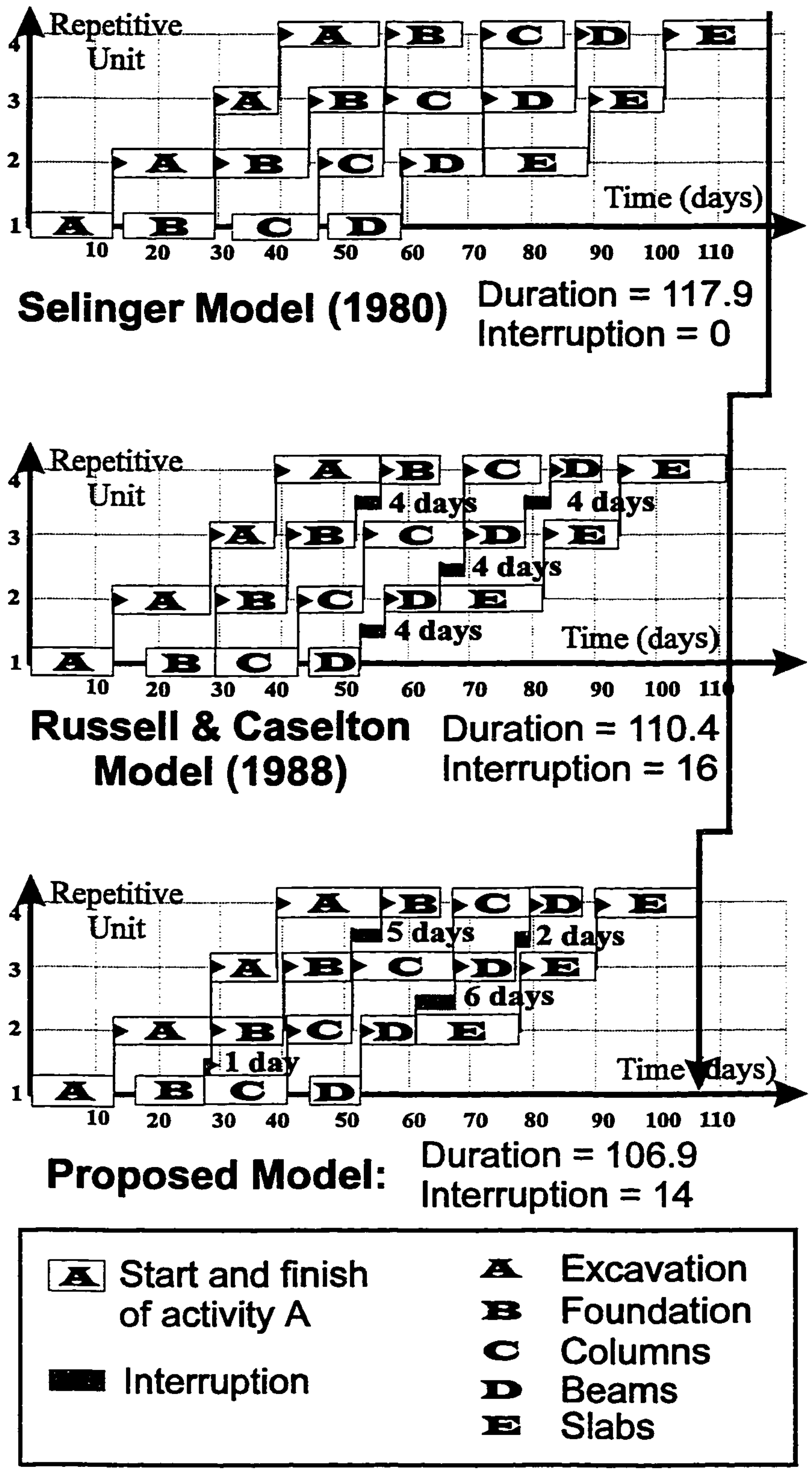


Figure 5.11 Least Duration Solution

5.7 Summary

This chapter presented the development of the scheduling optimization functions incorporated in the present object-oriented model. In order to optimize the scheduling of repetitive construction projects in this model, an interruption algorithm and an optimization procedure were developed. The interruption algorithm generates feasible interruption vectors for each crew formation in the project and provides added advantage over available formulations that consider arbitrary user-specified interruption vectors prior to scheduling (Russell and Caselton 1988, and Eldin and Senouci 1993): 1) the algorithm is automated and therefore the construction planner need not provide a user-specified set of interruption vectors prior to scheduling; 2) it generates a limited number of feasible interruption vectors and therefore enables a practical and feasible approach for the consideration of interruption during schedule optimization; and 3) the algorithm generates all needed interruption vectors during schedule calculation rather than being limited by a prespecified Imax value before scheduling, ensuring the generation and selection of the optimum solution.

The optimization procedure is based on a dynamic programming formulation. Unlike available formulations, the present formulation is capable of incorporating cost in the optimization process, thus offering valuable support to project team members in minimizing the overall cost of the project. For each repetitive activity in the project, the present model assists the planner in selecting the optimum crew formation and interruption vector from a set of possible alternatives. As such, the model can be used to evaluate the impact of different project acceleration strategies (i.e multiple crews, increased crew size, overtime policies, or additional shifts) on the overall cost. In order to demonstrate the use of the model and illustrate its capabilities, a project example from the literature was analyzed twice. The first analysis provides a least cost solution ad the second presents a minimum duration solution for the project. In both analyses, the present model provided a better solution for the same project example (i.e. further savings in project overall cost or reduction in project duration) than that generated by available models (Selinger 1980, and Russell and Caselton 1988).

CHAPTER 6

IMPLEMENTATION OF THE SCHEDULING MODEL: LSCHEDULER

6.1 Introduction

This chapter presents the implementation stage of the present object-oriented model for scheduling of repetitive construction projects. This stage represents the third and last stage in model development (Rumbaugh et al 1991). The model is implemented using Borland C++ integrated development environment (IDE) (Borland C++ 1996) as a 32-bit windows application that is named *LSCHEDULER*. *LSCHEDULER* runs on Microsoft Windows 95 and NT and supports user-friendly interface. The first letter (*L*) of the name *LSCHEDULER* is used to represent "*L*"inear construction projects *SCHEDULER* and also to represent "*L*"east project cost/duration *SCHEDULER*. *LSCHEDULER* consists of two main modules: scheduling module and user interface module as shown in Figure 6.1. The two modules are combined in *LSCHEDULER* using the Borland C++ IDE *Project Manager* which can be used to manage and combine program source files (.H, .C, and/or .CPP files) to produce target files (.EXE, and/or .DLL files). Each of the two modules consists of a number of header files (.H) and code files (.CPP) as shown in Figure 6.1.

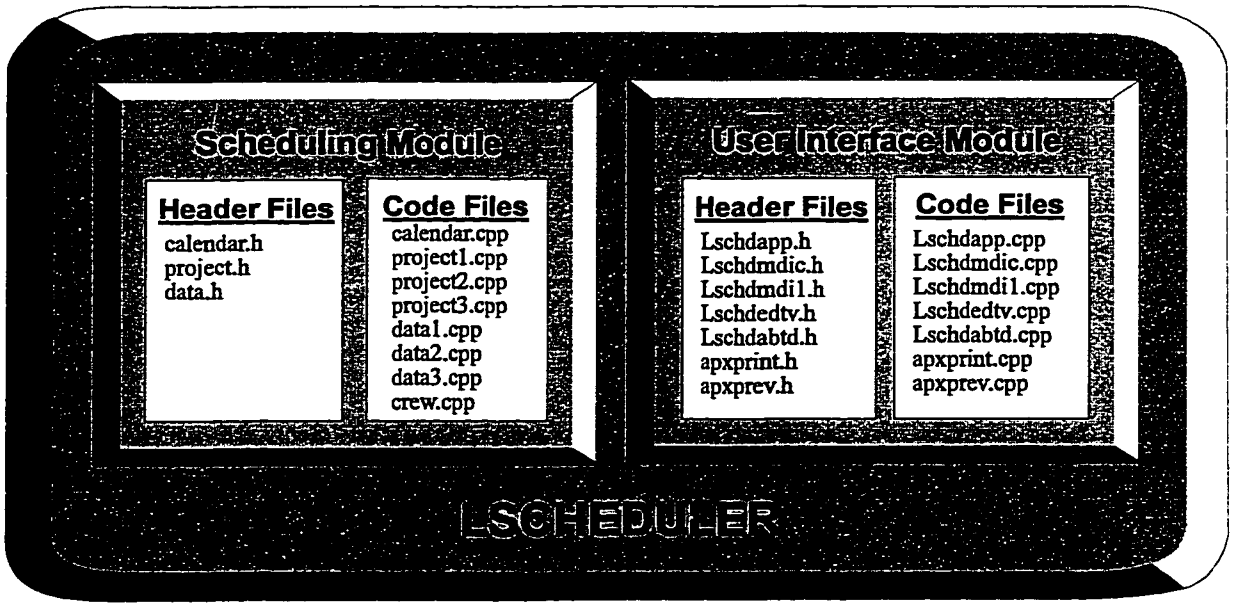


Figure 6.1 *LSCHEDULER* Modules

6.2 Scheduling Module

The scheduling module includes all header and code files, representing the implementation stage of the present object-oriented model. For each class in the model, the implementation stage provides: 1) a class declaration, and 2) a class definition (Rumbaugh et al 1991). The class declaration is a simple C++ computer code representing a list of all data members and member functions of the class. The class definition is a detailed C++ computer code for each member function of the class. In C++, class declarations are included in header files (.H) and class definitions in code files (.CPP). In this scheduling module, class declarations are included in three header files (project.h, data.h, and calendar.h) as shown in Appendix I, and class definitions in eight code files (project1.cpp, project2.cpp, project3.cpp, data1.cpp, data2.cpp, data3.cpp, crew.cpp, and calendar.cpp) as shown in Figure 6.1.

As described earlier in Chapter 3, the present object-oriented model consists of 10 main classes (*Date*, *Project*, *Project-Data, Activity*, *Repetitive* activity, *Non-Repetitive* activity, *Regular-Relation*, *Repetitive-Relation*, *Hetero-Relation*, and *Crew-Formation*) as shown in Figure 3.6. The declaration of *Date* class is included in "calendar.h" header file, and the definitions of its member functions are included in "calendar.cpp" file. The declaration of *Project* class is included in "project.h" header file, and the definitions of its member functions are included in three separate files (project1.cpp, project2.cpp, and project3.cpp). "Project1.cpp" file includes the function definition of saving the project data to a binary file, while "project2.cpp" file includes the function definition of opening and reading a project data file. "Project3.cpp" file includes the definitions of all the remaining functions for *Project* class.

The declaration of the remaining eight classes in the present model are included in "data.h" header file. The definitions of the member functions for these classes are included in four separate files (data1.cpp, data2.cpp, data3.cpp and crew.cpp). “Data1.cpp” file includes the definitions of the member functions for five classes: *Activity, Non-Repetitive* activity, *Regular-Relation, Repetitive-Relation, and Hetero-Relation*. "Data2.cpp" and “data3.cpp” files include the definitions of regular scheduling functions, and scheduling optimization functions, respectively, for *Repetitive* activity class. "Crew.cpp" file includes the definitions of member functions for *Crew-Formation* class.

6.3 User Interface Module

The Borland C++ integrated development environment (IDE) incorporates a tool for building user interface called *AppExpert* (Borland C++ 1996). The user interface module of *LSCHEDULER* is developed using *AppExpert*, and it includes seven header files (lschdapp.h, lschdmdic.h, lschdmdi1.h, lschdedtv.h, lschdabtd.h, apxprint.h, and apxprev.h) and seven code files (lschdapp.cpp, lschdmdic.cpp, lschdmdi1.cpp, lschdedtv.cpp, lschdabtd.cpp, apxprint.cpp, and apxprev.cpp) as shown in Figure 6.1. The user interface of *LSCHEDULER* incorporates menus, a tool bar, a status bar, dialog boxes, and multiple document interface (MDI) windows as shown in Figure 6.2.

6.3.1 Menus, Tool Bar, and Status Bar

Menus are lists of commands that the user can choose from to perform a specific function. The main menus of a windows application are often listed in a menu bar at the top of the screen. *LSCHEDULER* contains a menu bar at the top of the screen that consists of nine main menus: *File, Project, Activity, Relationship,* *Edit, Search, Display, Window*, and *Help* as shown in Figure 6.2. Each of these main menus includes a list of menu items, each of which can perform a specific command. For example, *File* main menu includes a list of ten menu items: *New, Open, Close, Save, Save As, Print, Print Setup, Send*, and *Exit* as shown in Figure 6.3. It should be noted that the name of some menu items are followed by three periods "...", which indicates that if the user executes one of these menu items the program will display a dialog box on the screen to allow the user to specify needed input data. For example, if the user executes "*Open...*" menu item (see Figure 6.3), the program will display the Open dialog box shown later in Figure 6.14. The menu items included in each of the nine main menus of *LSCHEDULER* are shown in Figure 6.3 to Figure 6.12. For each menu item, the function and the associated dialog box, if any, are summarized in Table 6.1 to Table 6.11.

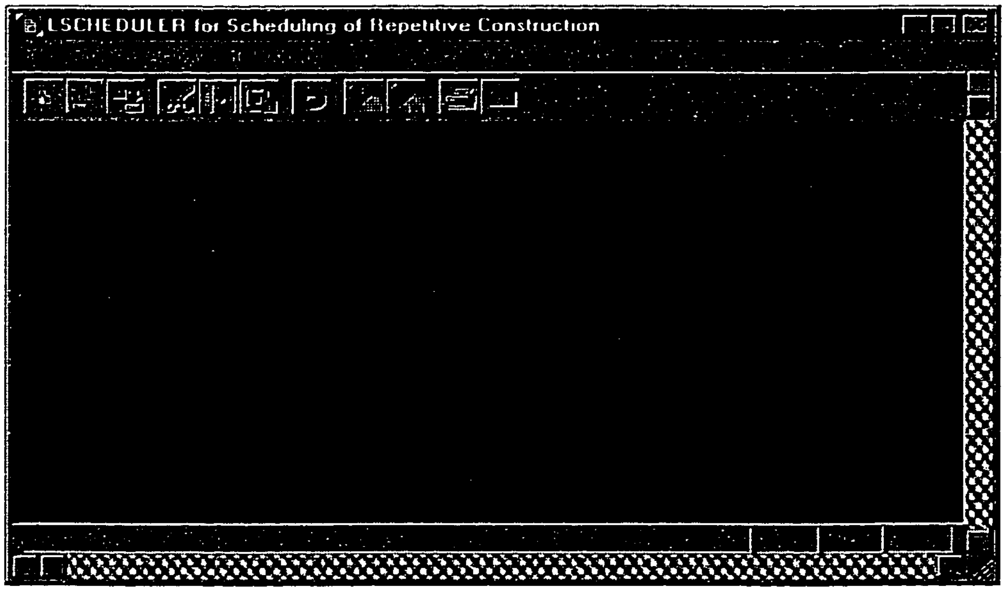
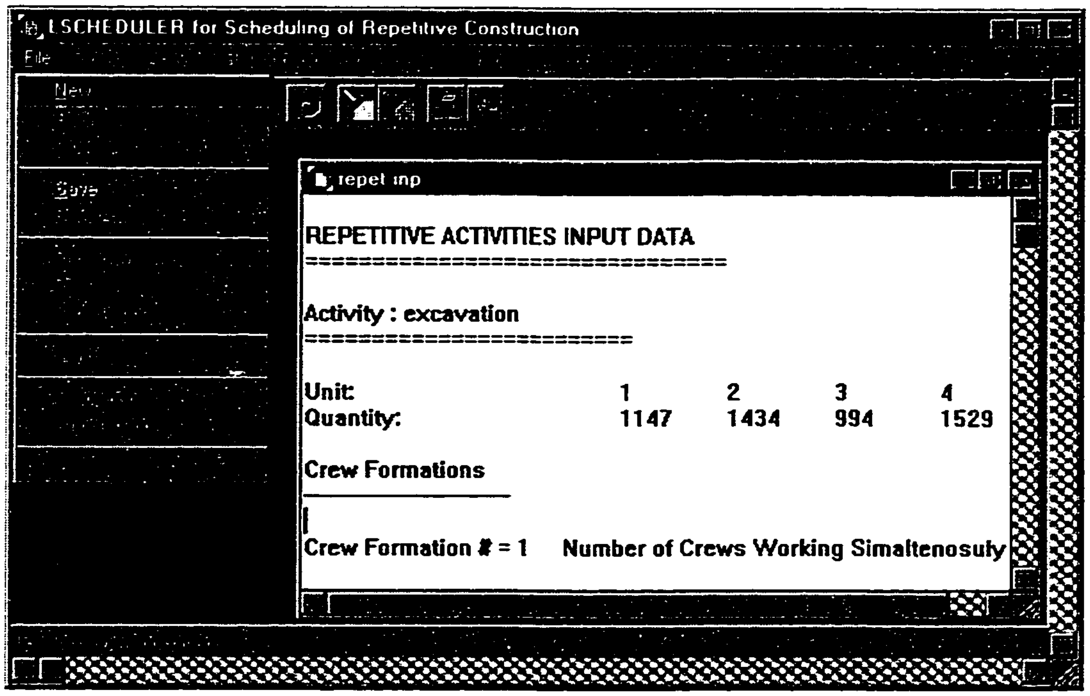


Figure 6.2 Developed Windows Application: *LSCHEDULER*



REPETITIVE ACTIVITIES INPUT DATA
```
==============================

Activity : excavation
=======================

Unit:                    1       2       3       4
Quantity:                1147    1434    994     1529

Crew Formations

Crew Formation # = 1     Number of Crews Working Simaltenosuly
```

Figure 6.3 File Menu

Table 6.1 File Menu Functions

Menu Item	Prompted Dialog Box	Function
New	-	Creates a new document and displays it in a new window.
Open...	Figure 6.14	Opens an existing document and displays it as shown in Figure 6.3.
Close	-	Closes the window of the active document.
Save	-	Saves changes made to the active document.
Save As...	Figure 6.15	Saves a copy of the document in another file.
Print Preview...	-	Displays full pages as they will be printed.
Print...	Figure 6.16	Prints the active document.
Print Setup...	Figure 6.17	Changes the printer set-up.
Send...	-	Sends the active document through electronic mail.
Exit	-	Quits *LSCHEDULER*.

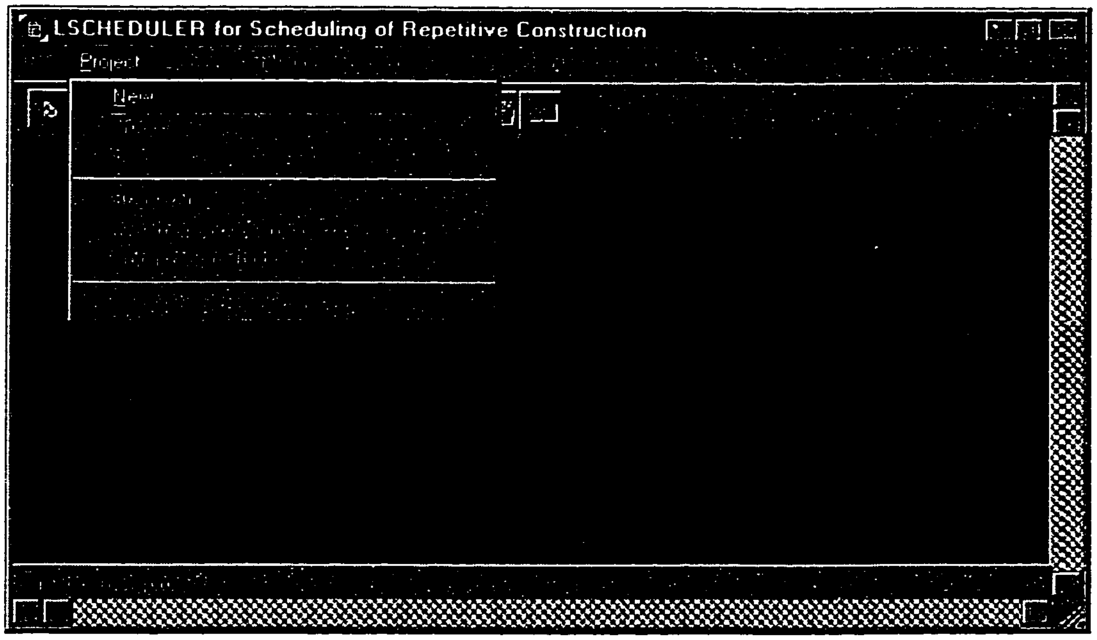


Figure 6.4 Project Menu

Table 6.2 Project Menu Functions

Menu Item	Prompted Dialog Box	Function
New	-	Creates a new project.
Open...	Figure 6.18	Opens an existing project.
Save...	-	Saves changes made to the current project.
Start Date...	Figure 6.19	Specifies the start date of the project.
Weather and Learning Curve Options...	Figure 6.20	Specifies whether to consider weather impact and/or learning curve effect in scheduling calculation or not.
Optimization Options...	Figure 6.21	Specifies the scheduling optimization criterion which can be: a) perform regular scheduling without optimization; b) minimize project overall cost; or c) minimize project duration.
Calculate Schedule and Cost	-	Performs scheduling calculation.

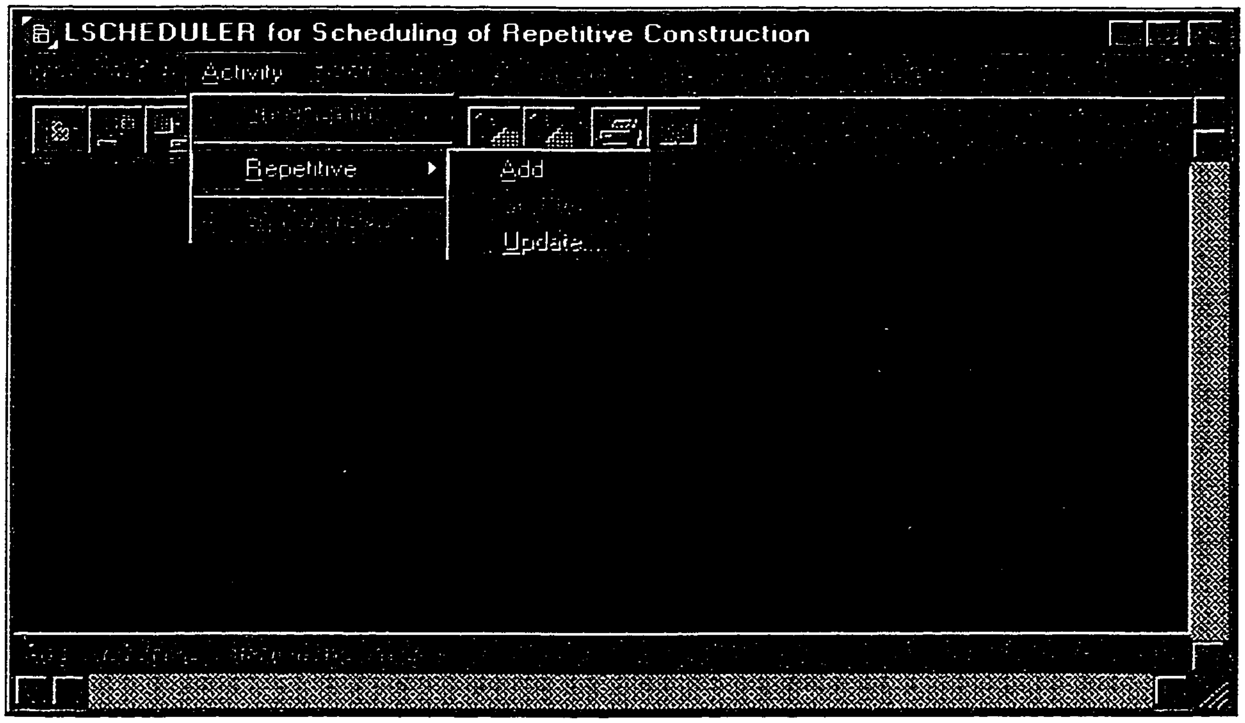


Figure 6.5 Activity Menu

Table 6.3 Activity Menu Functions

Menu Item	Prompted Dialog Box	Function
Add...	Figure 6.22 Figure 6.23	Adds a new activity (i.e. non-repetitive, repetitive, or subcontractor) to the project and inputs its data.
Modify...	-	Modifies the input data of an existing activity.
Update...	-	Updates the actual start and finish dates of an activity.

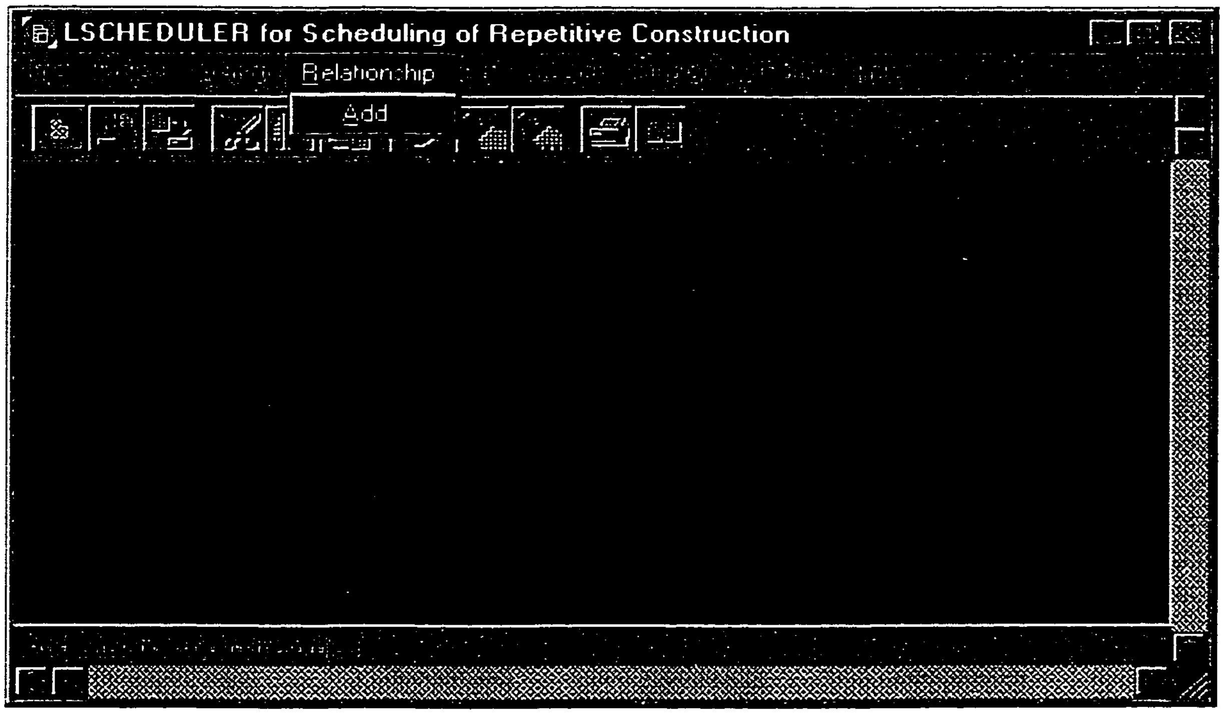
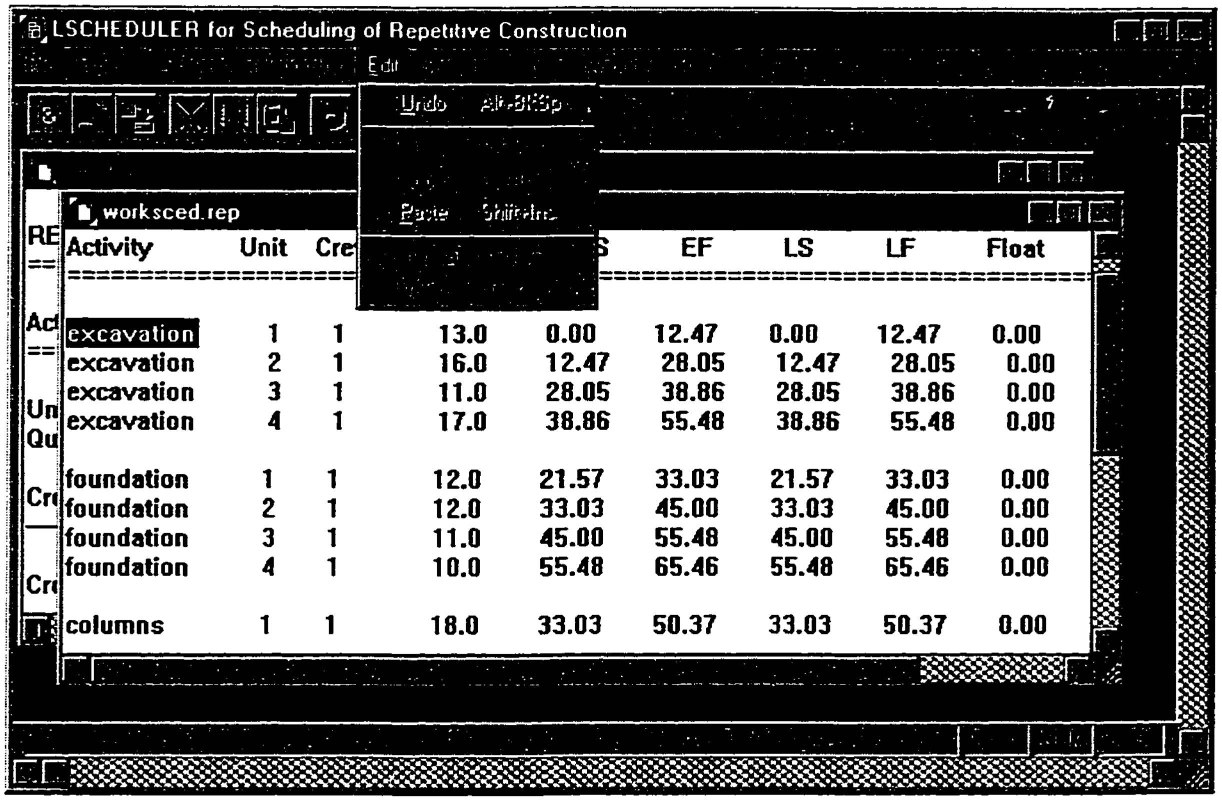


Figure 6.6 Relationship Menu

Table 6.4 Relationship Menu Functions

Menu Item	Prompted Dialog Box	Function
Add...	Figure 6.28	Adds a new precedence relationship to the project and specifies its type, lag in days, and predecessor and successor activities.

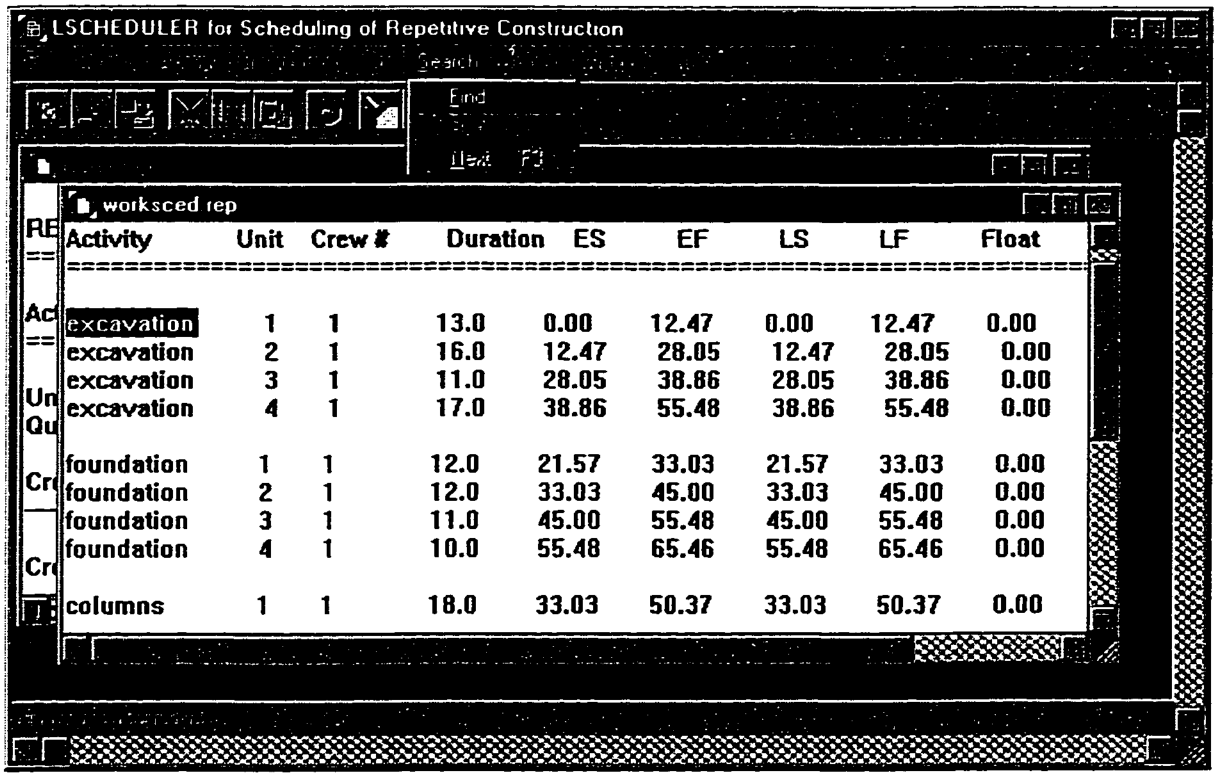


```
Activity      Unit  Cre                 S      EF      LS      LF     Float
===========================================================================

excavation      1    1      13.0     0.00   12.47    0.00   12.47    0.00
excavation      2    1      16.0    12.47   28.05   12.47   28.05    0.00
excavation      3    1      11.0    28.05   38.86   28.05   38.86    0.00
excavation      4    1      17.0    38.86   55.48   38.86   55.48    0.00

foundation      1    1      12.0    21.57   33.03   21.57   33.03    0.00
foundation      2    1      12.0    33.03   45.00   33.03   45.00    0.00
foundation      3    1      11.0    45.00   55.48   45.00   55.48    0.00
foundation      4    1      10.0    55.48   65.46   55.48   65.46    0.00

columns         1    1      18.0    33.03   50.37   33.03   50.37    0.00
```

Figure 6.7 Edit Menu

Table 6.5 Edit Menu Functions

Menu Item	Prompted Dialog Box	Function
Undo	-	Reverses the last edit operation.
Cut	-	Cuts the selected items and puts them on the clipboard.
Copy	-	Copies the selected items and puts them on the clipboard.
Paste	-	Inserts the clipboard contents at the insertion point.
Clear All	-	Clears the entire active document.
Delete	-	Deletes the selected items.

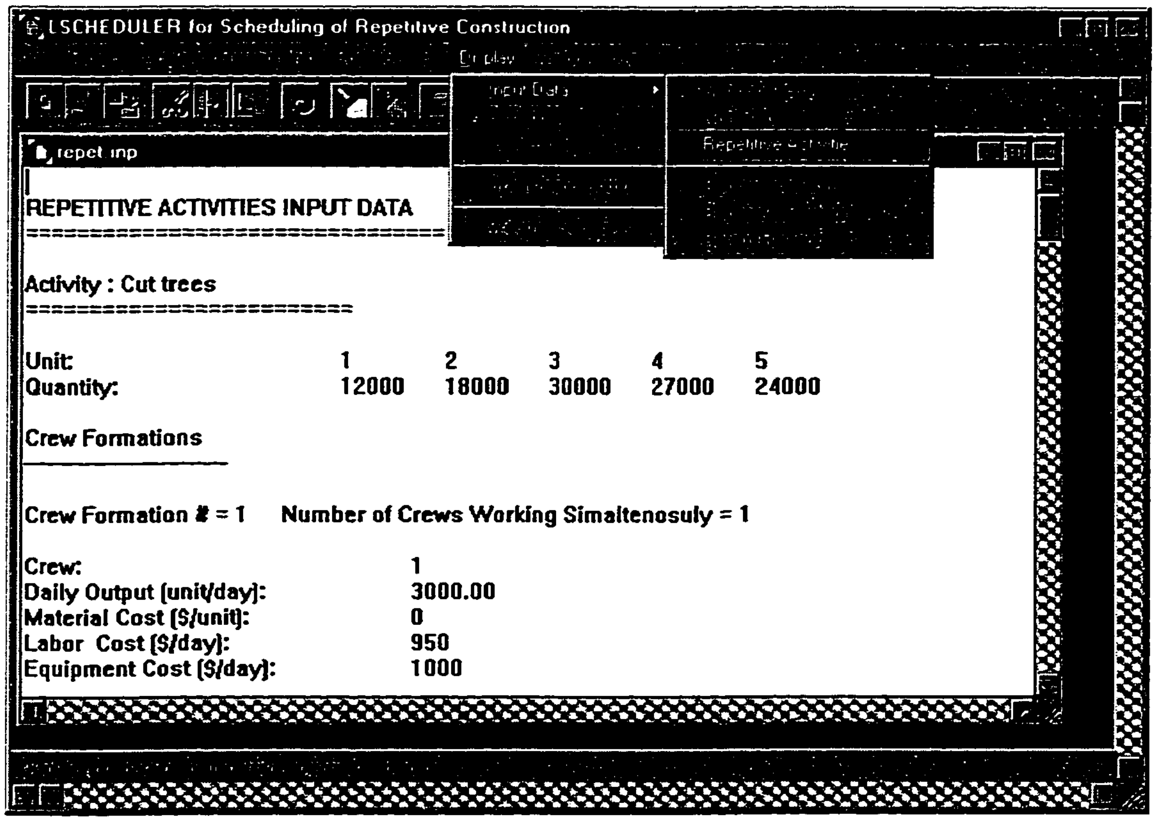


```
Activity      Unit  Crew #   Duration   ES      EF      LS      LF      Float
==============================================================================

excavation     1     1       13.0      0.00    12.47   0.00    12.47   0.00
excavation     2     1       16.0      12.47   28.05   12.47   28.05   0.00
excavation     3     1       11.0      28.05   38.86   28.05   38.86   0.00
excavation     4     1       17.0      38.86   55.48   38.86   55.48   0.00

foundation     1     1       12.0      21.57   33.03   21.57   33.03   0.00
foundation     2     1       12.0      33.03   45.00   33.03   45.00   0.00
foundation     3     1       11.0      45.00   55.48   45.00   55.48   0.00
foundation     4     1       10.0      55.48   65.46   55.48   65.46   0.00

columns        1     1       18.0      33.03   50.37   33.03   50.37   0.00
```

Figure 6.8 Search Menu

Table 6.6 Search Menu Functions

Menu Item	Prompted Dialog Box	Function
Find...	-	Finds the specified text.
Replace...	-	Finds the specified text and changes it.
Next	-	Finds the next match.

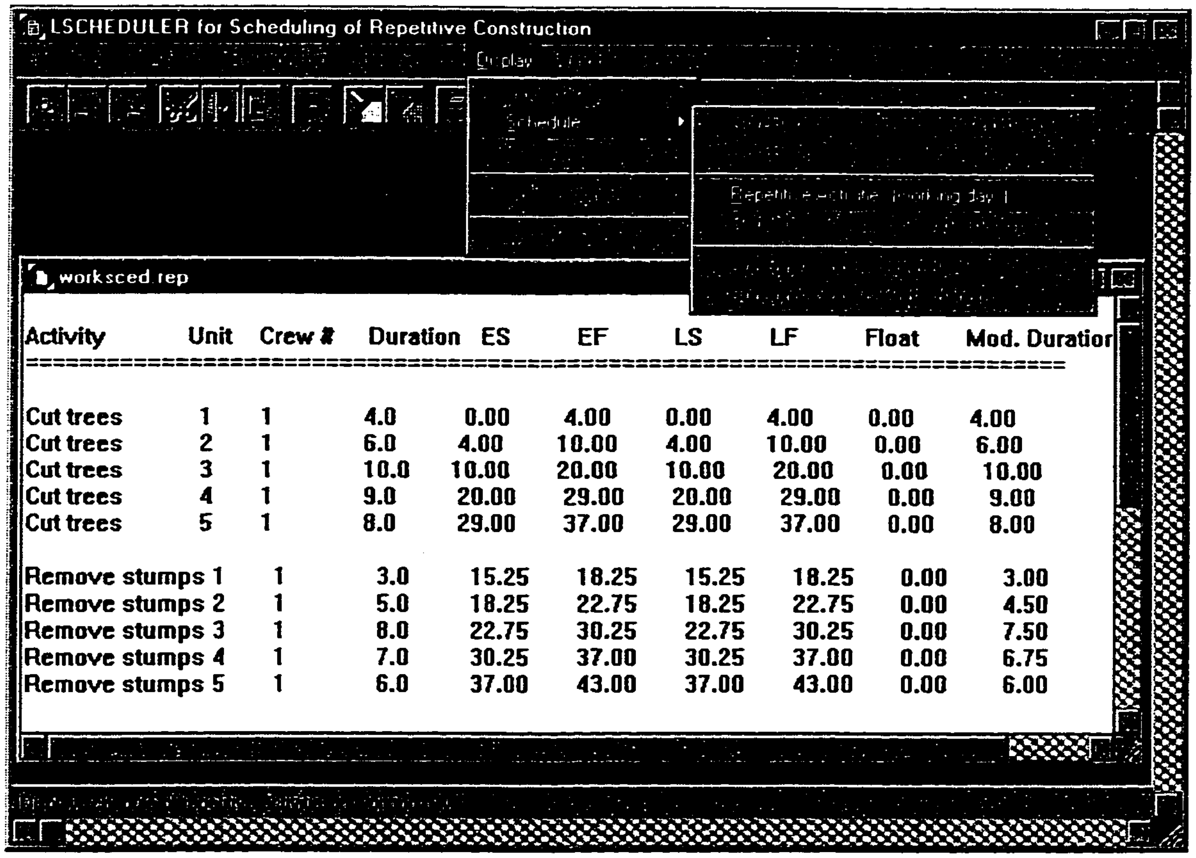


```
REPETITIVE ACTIVITIES INPUT DATA
================================

Activity : Cut trees
========================

Unit:                 1        2        3        4        5
Quantity:             12000    18000    30000    27000    24000

Crew Formations
_______________

Crew Formation # = 1    Number of Crews Working Simaltenosuly = 1

Crew:                         1
Daily Output (unit/day):      3000.00
Material Cost ($/unit):       0
Labor  Cost ($/day):          950
Equipment Cost ($/day):       1000
```

Figure 6.9 Display Input Data Menu

Table 6.7 Display Input Data Menu Functions

Menu Item	Prompted Dialog Box	Function
Subcontractors	-	Displays input data of subcontracted activities.
Non-Repetitive Activities	-	Displays input data of non-repetitive activities.
Repetitive Activities	-	Displays input data of repetitive activities.
Regular Relation	-	Displays input data of precedence relationships between non-repetitive activities.
Repetitive Relation	-	Displays input data of precedence relationships between repetitive activities.
Hetero Relation	-	Displays input data of relationships between repetitive and non-repetitive activities.

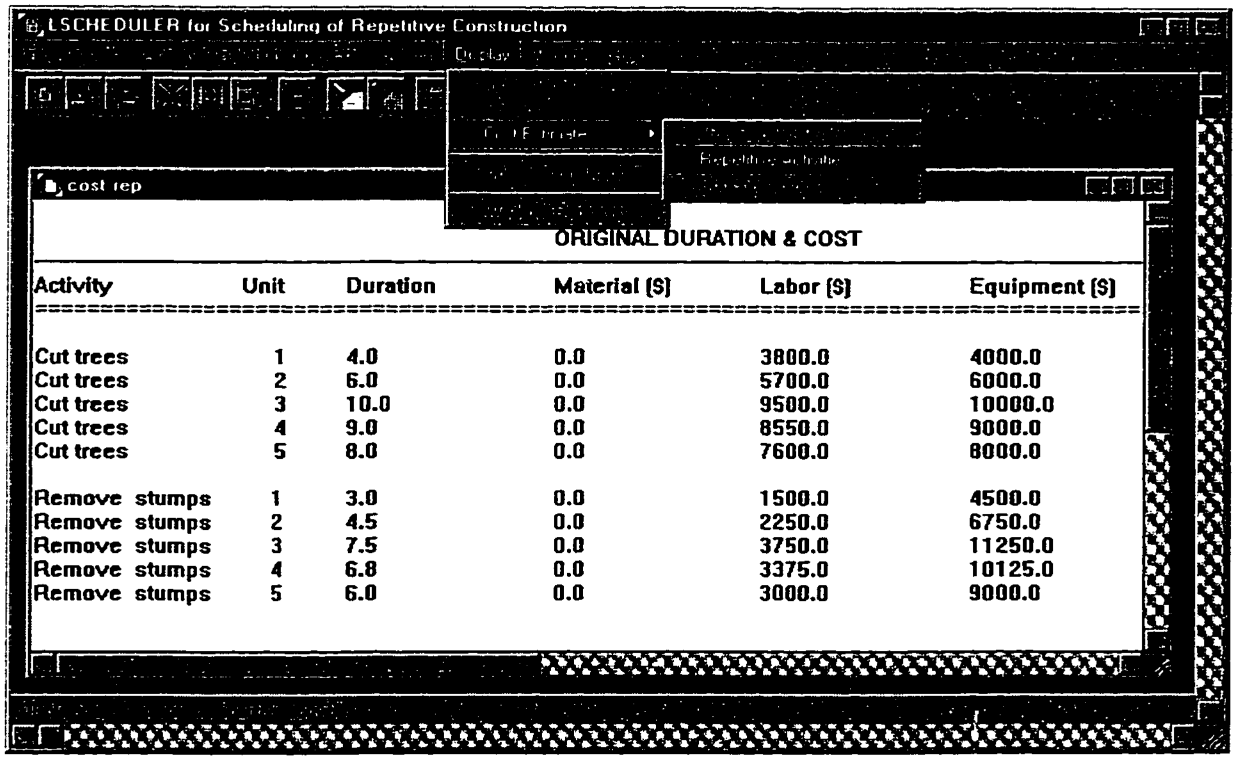


```
Activity        Unit  Crew #  Duration  ES     EF     LS     LF     Float  Mod. Duratio
=============================================================================

Cut trees        1     1      4.0      0.00   4.00   0.00   4.00   0.00   4.00
Cut trees        2     1      6.0      4.00   10.00  4.00   10.00  0.00   6.00
Cut trees        3     1      10.0     10.00  20.00  10.00  20.00  0.00   10.00
Cut trees        4     1      9.0      20.00  29.00  20.00  29.00  0.00   9.00
Cut trees        5     1      8.0      29.00  37.00  29.00  37.00  0.00   8.00

Remove stumps 1        1      3.0      15.25  18.25  15.25  18.25  0.00   3.00
Remove stumps 2        1      5.0      18.25  22.75  18.25  22.75  0.00   4.50
Remove stumps 3        1      8.0      22.75  30.25  22.75  30.25  0.00   7.50
Remove stumps 4        1      7.0      30.25  37.00  30.25  37.00  0.00   6.75
Remove stumps 5        1      6.0      37.00  43.00  37.00  43.00  0.00   6.00
```

Figure 6.10 Display Schedule Menu

Table 6.8 Display Schedule Menu Functions

Menu Item	Prompted Dialog Box	Function
Non-Repetitive Activities	-	Displays schedule of non-repetitive activities in working days or calendar dates.
Repetitive Activities	-	Displays schedule of repetitive activities in working days or calendar dates.
Subcontractors	-	Displays schedule of subcontracted activities in working days or calendar dates.



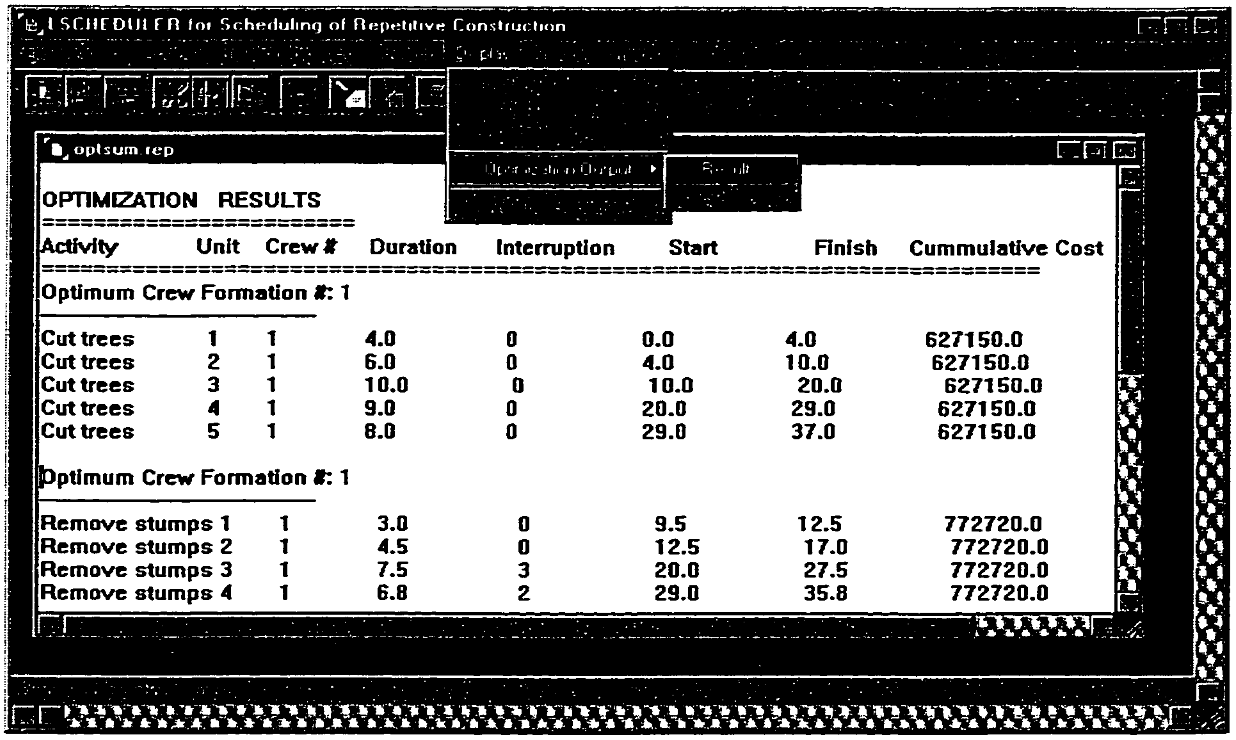
ORIGINAL DURATION & COST

```
Activity          Unit   Duration      Material [$]    Labor [$]     Equipment [$]
==================================================================================

Cut trees           1    4.0           0.0             3800.0        4000.0
Cut trees           2    6.0           0.0             5700.0        6000.0
Cut trees           3    10.0          0.0             9500.0        10000.0
Cut trees           4    9.0           0.0             8550.0        9000.0
Cut trees           5    8.0           0.0             7600.0        8000.0

Remove  stumps      1    3.0           0.0             1500.0        4500.0
Remove  stumps      2    4.5           0.0             2250.0        6750.0
Remove  stumps      3    7.5           0.0             3750.0        11250.0
Remove  stumps      4    6.8           0.0             3375.0        10125.0
Remove  stumps      5    6.0           0.0             3000.0        9000.0
```

Figure 6.11 Display Cost Estimates Menu

Table 6.9 Display Cost Estimates Menu Functions

Menu Item	Prompted Dialog Box	Function
Non-Repetitive Activities	-	Displays cost estimates of non-repetitive activities.
Repetitive Activities	-	Displays cost estimates of repetitive activities.
Subcontractors	-	Displays cost estimates of subcontracted activities.

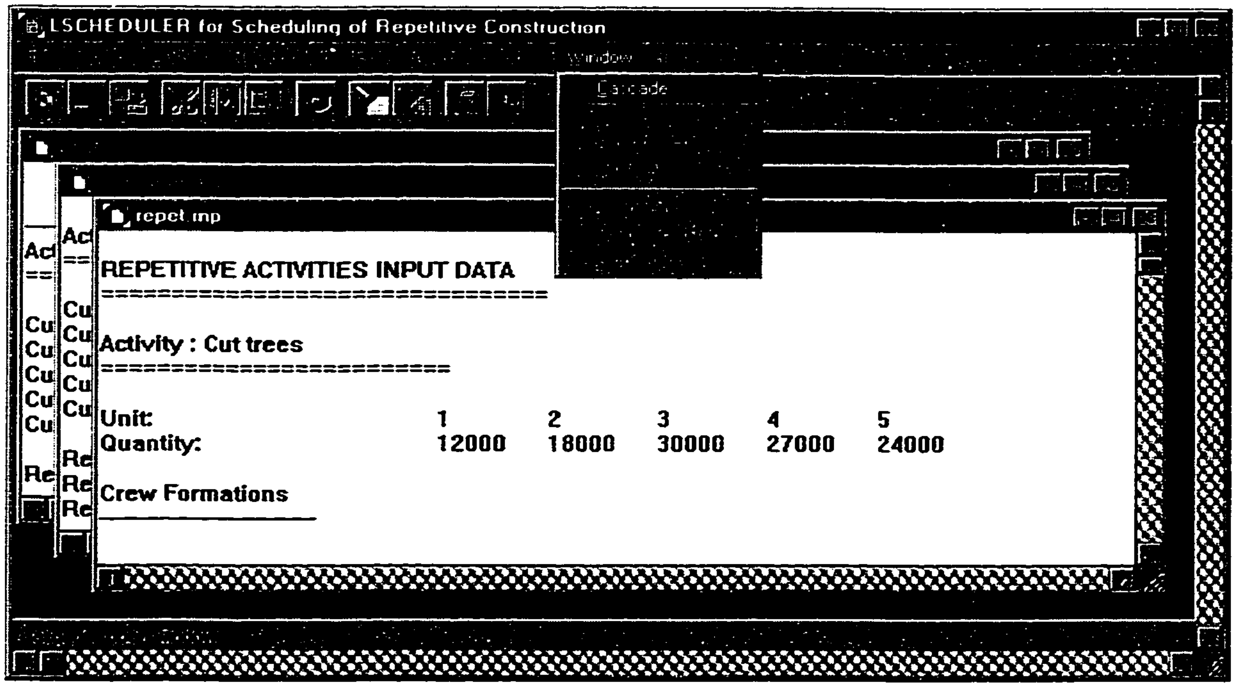


```
OPTIMIZATION   RESULTS
=======================
Activity        Unit  Crew #   Duration   Interruption   Start      Finish   Cummulative Cost
=========================================================================================
Optimum Crew Formation #: 1
---------------------------
Cut trees        1     1       4.0        0              0.0        4.0      627150.0
Cut trees        2     1       6.0        0              4.0        10.0     627150.0
Cut trees        3     1       10.0       0              10.0       20.0     627150.0
Cut trees        4     1       9.0        0              20.0       29.0     627150.0
Cut trees        5     1       8.0        0              29.0       37.0     627150.0

Optimum Crew Formation #: 1
---------------------------
Remove stumps 1        1       3.0        0              9.5        12.5     772720.0
Remove stumps 2        1       4.5        0              12.5       17.0     772720.0
Remove stumps 3        1       7.5        3              20.0       27.5     772720.0
Remove stumps 4        1       6.8        2              29.0       35.8     772720.0
```

Figure 6.12 Display Optimization Output Menu

Table 6.10 Display Optimization Output Menu Functions

Menu Item	Prompted Dialog Box	Function
Results	-	Displays summarized results of scheduling optimization.
Analysis	-	Displays detailed analysis of scheduling optimization.



```
REPETITIVE ACTIVITIES INPUT DATA
================================

Activity : Cut trees
========================

Unit:                      1      2      3      4      5
Quantity:                  12000  18000  30000  27000  24000

Crew Formations
```

Figure 6.13 Window Menu

Table 6.11 Window Menu Functions

Menu Item	Prompted Dialog Box	Function
Cascade	-	Cascades open windows.
Tile		Tiles open windows.
Arrange Icons	-	Arranges iconic windows along the bottom.
Close All	-	Closes all open windows.

LSCHEDULER incorporates a tool bar at the top of the screen below the menu bar as shown in Figure 6.13. The tool bar includes a number of icons, each representing a shortcut for executing a specific command. For example, when the user clicks on the first left icon on the tool bar (see Figure 6.13), *LSCHEDULER* opens a new document. In addition to the menus and tool bar, *LSCHEDULER* incorporates a status bar at the bottom of the application's main window. The purpose of a status bar is to display information about the function of currently highlighted menu item. For example, when the user highlights the menu item of *Project* then *New*, the status bar displays a message "Create a new project" as shown in Figure 6.4.

6.3.2 Dialog Boxes

In order to facilitate user data input, *LSCHEDULER* incorporates a number of dialog boxes. The main dialog boxes of *LSCHEDULER* are shown in Figure 6.14 to Figure 6.28.

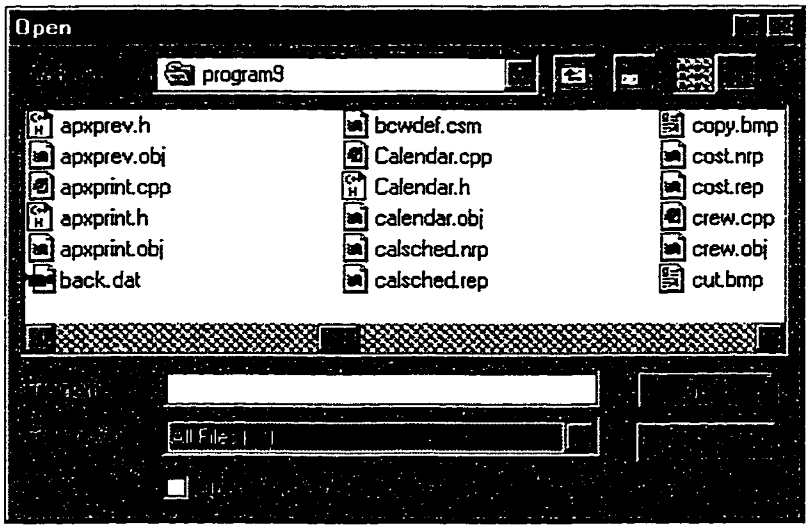
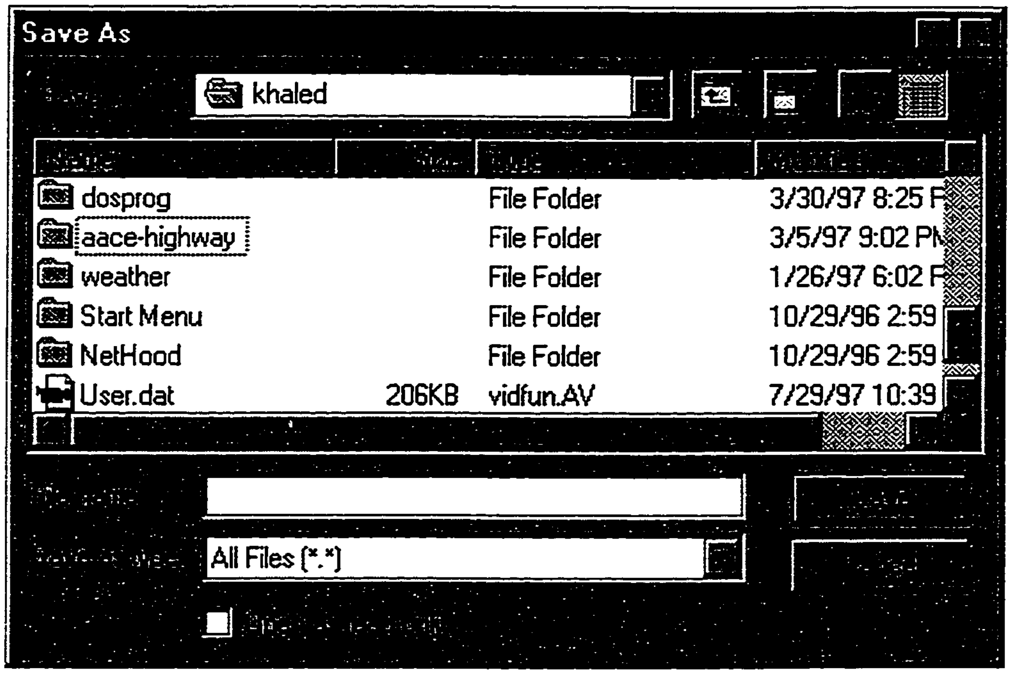


Figure 6.14 Open Dialog Box



dosprog		File Folder	3/30/97 8:25
aace-highway		File Folder	3/5/97 9:02 P
weather		File Folder	1/26/97 6:02
Start Menu		File Folder	10/29/96 2:59
NetHood		File Folder	10/29/96 2:59
User.dat	206KB	vidfun.AV	7/29/97 10:39

Figure 6.15 Save As Dialog Box

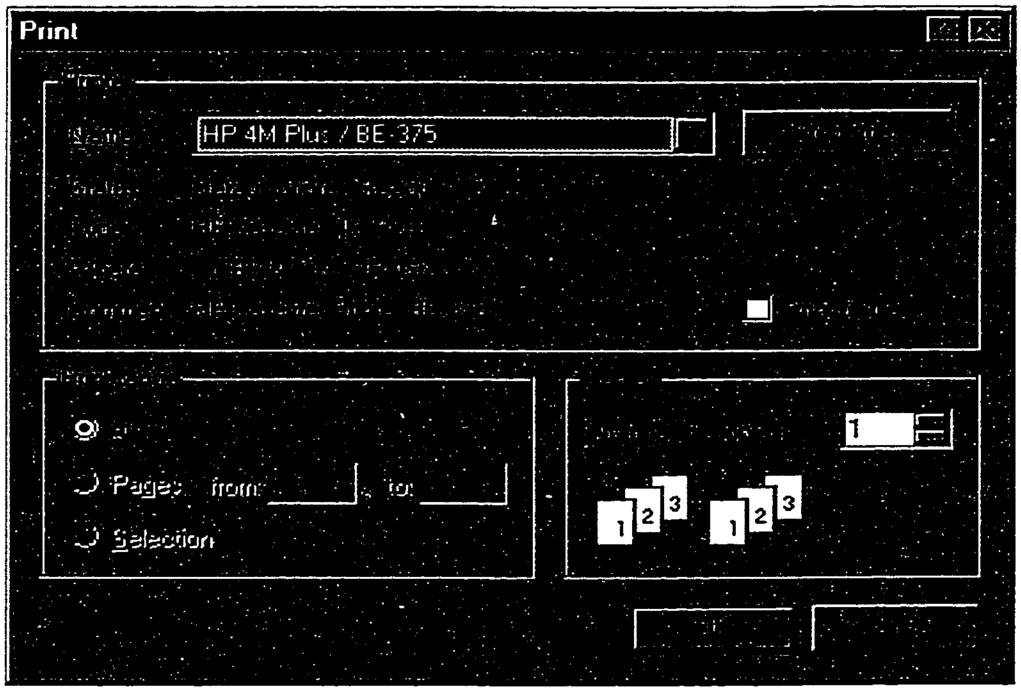
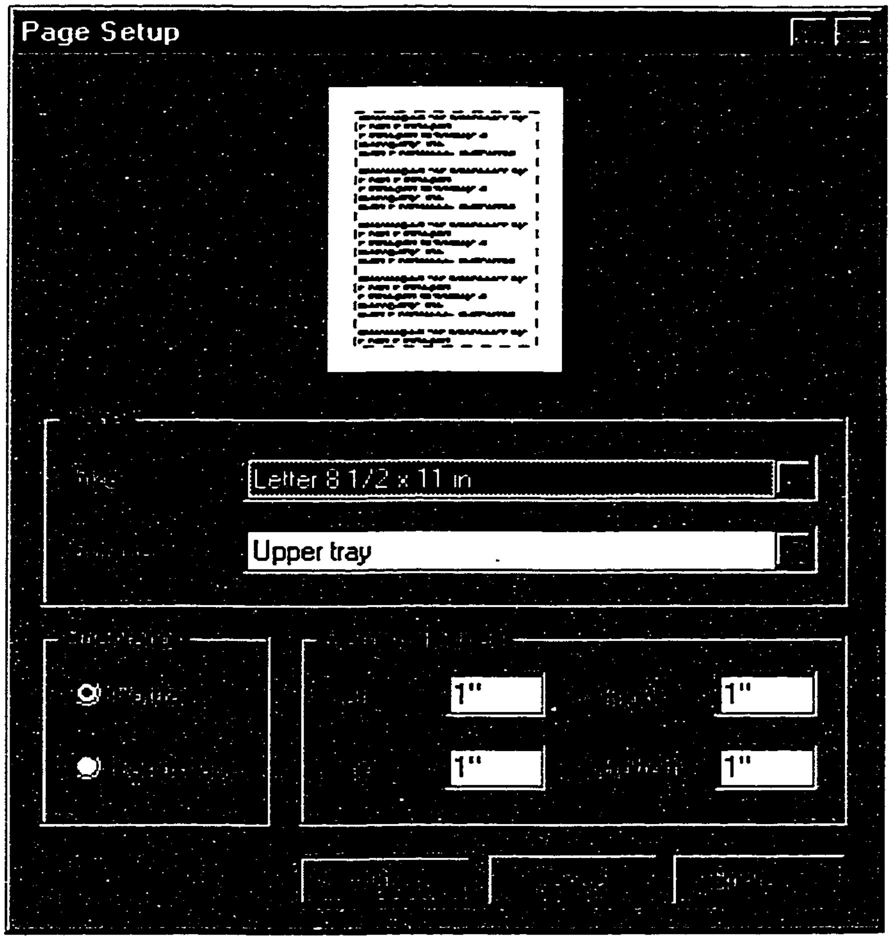


Figure 6.16 Print Dialog Box


Page Setup

Letter 8 1/2 x 11 in

Upper tray

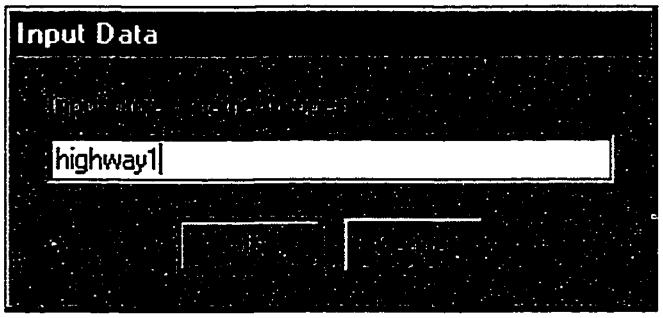
1"

1"

1"

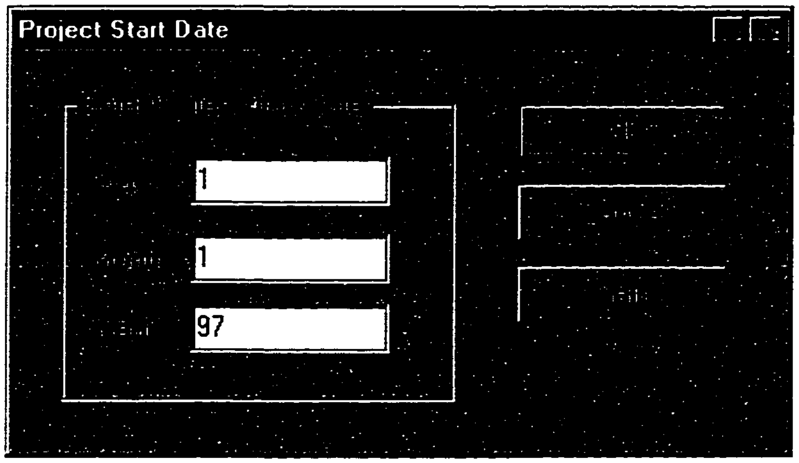
1"

Figure 6.17 Print Setup Dialog Box


Input Data

highway1

Figure 6.18 Open Project Dialog Box


Project Start Date

1

1

97

Figure 6.19 Project Start Date Dialog Box

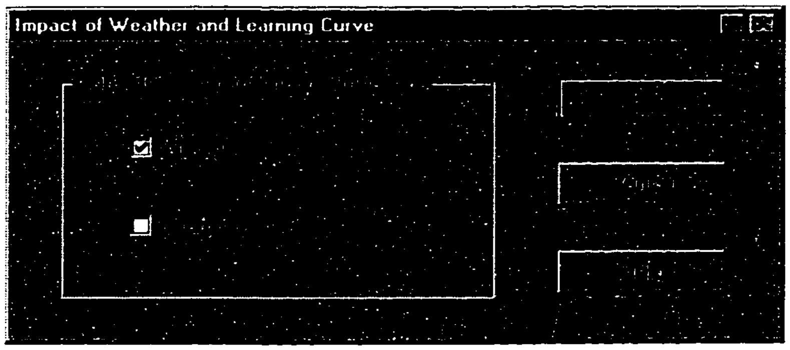
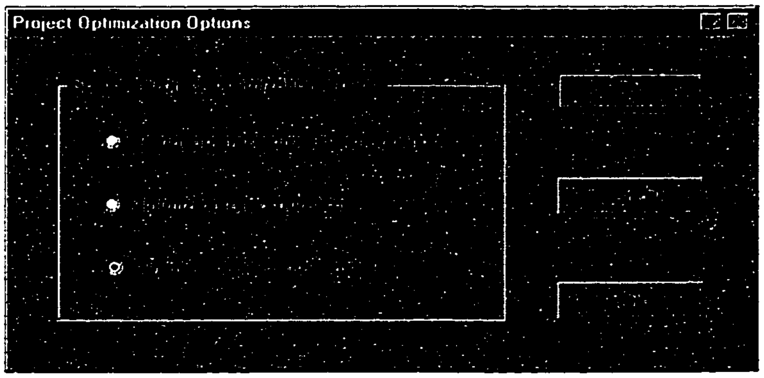
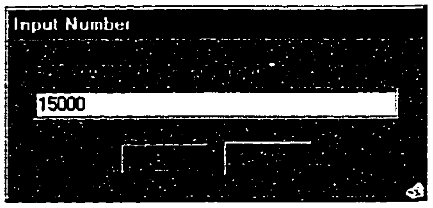

Impact of Weather and Learning Curve

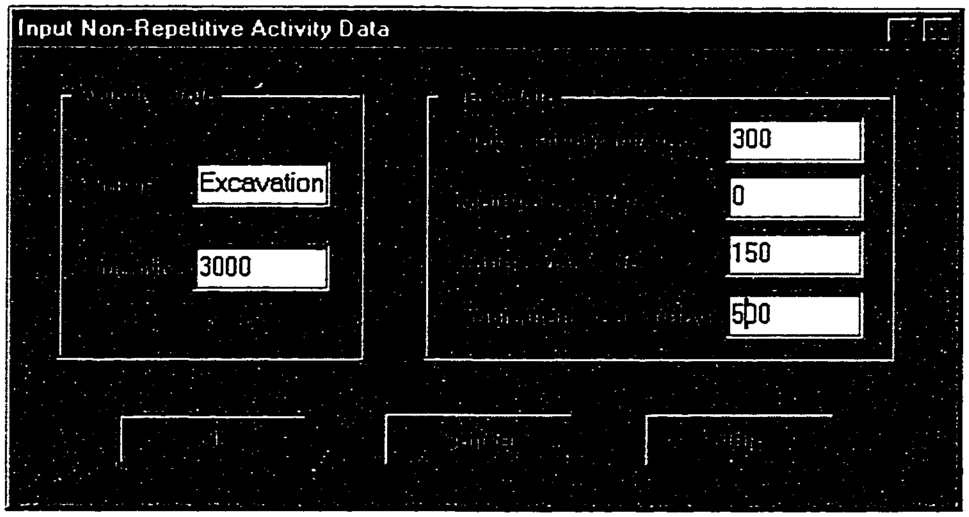
Figure 6.20 Weather and Learning Curve Dialog Box


Project Optimization Options


Input Number

15000

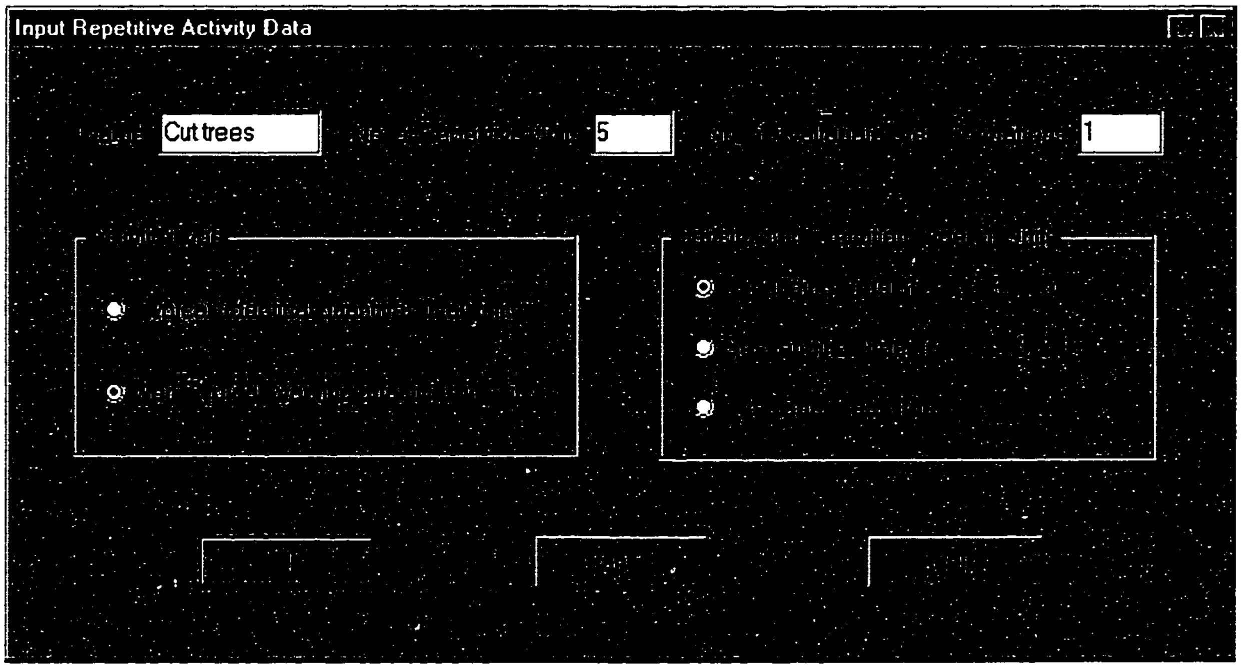
Figure 6.21 Scheduling Optimization Dialog Box



Input Non-Repetitive Activity Data

[illegible]	Excavation
[illegible]	3000
[illegible]	300
[illegible]	0
[illegible]	150
[illegible]	500

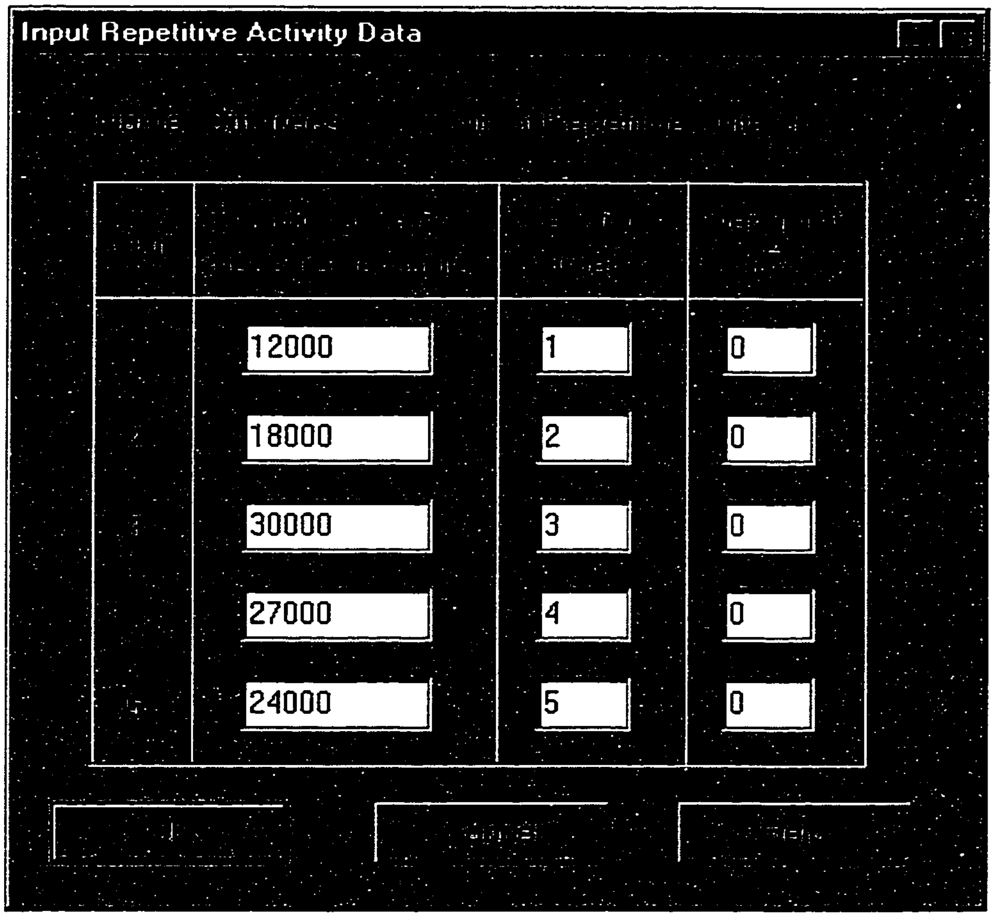
Figure 6.22 Non-Repetitive Activity Dialog Box



Input Repetitive Activity Data

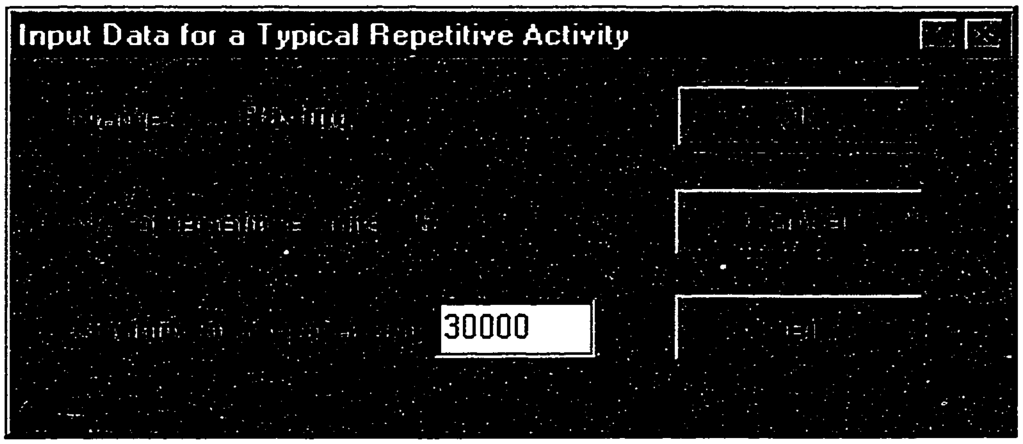
[illegible]	Cut trees
[illegible]	5
[illegible]	1

Figure 6.23 Repetitive Activity Dialog Box


Input Repetitive Activity Data

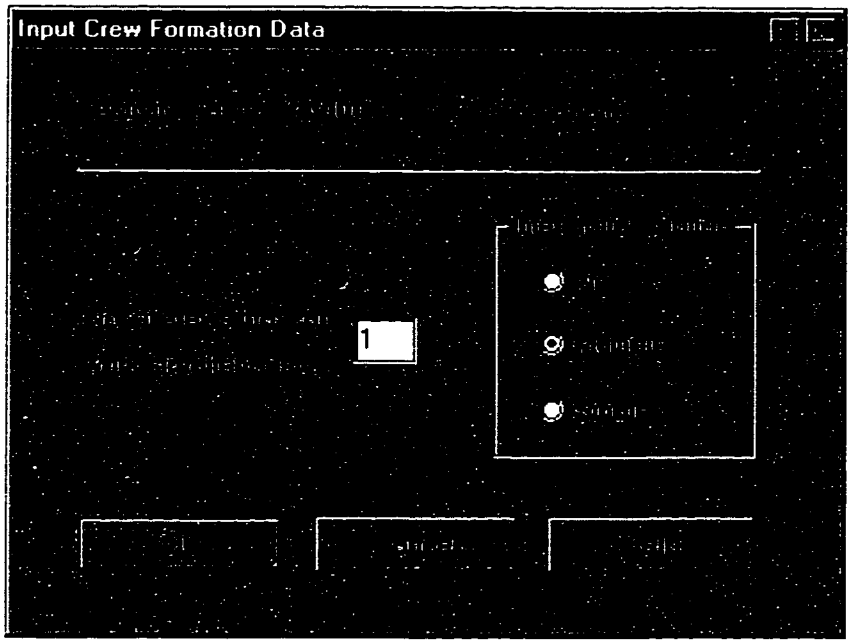
12000	1	0
18000	2	0
30000	3	0
27000	4	0
24000	5	0

Figure 6.24 Non-Typical Repetitive Activity Dialog Box


Input Data for a Typical Repetitive Activity

30000

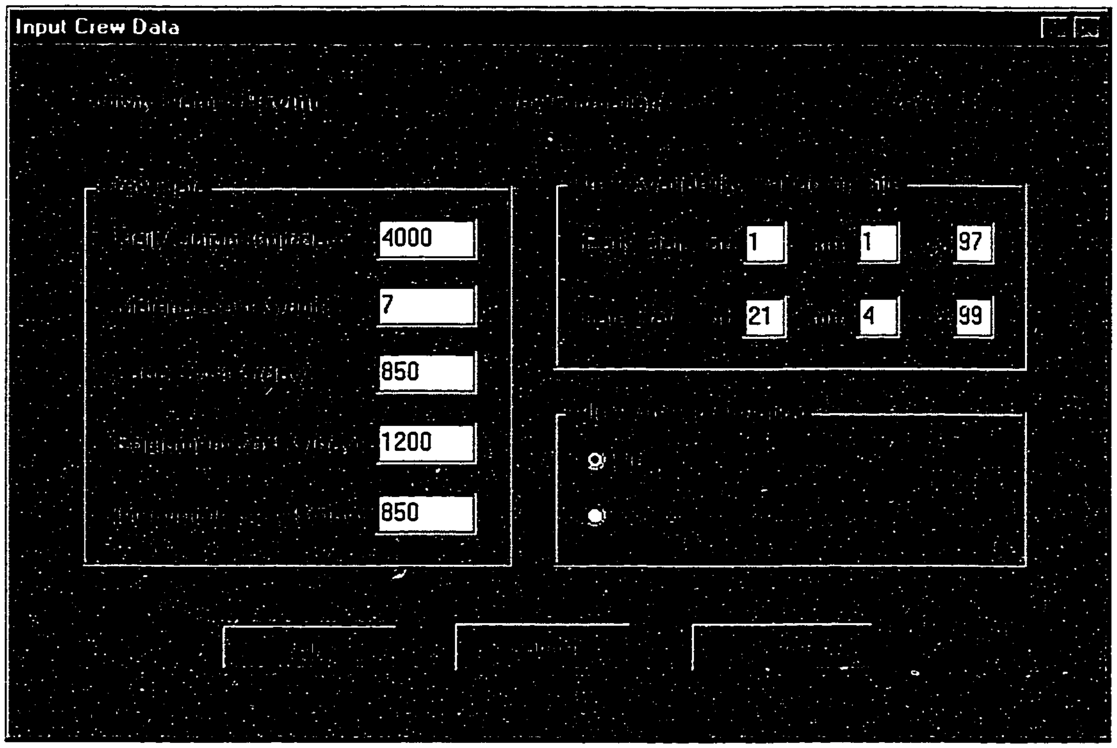
Figure 6.25 Typical Repetitive Activity Dialog Box



Input Crew Formation Data

1

Figure 6.26 Crew Formation Dialog Box



Input Crew Data

4000

7

850

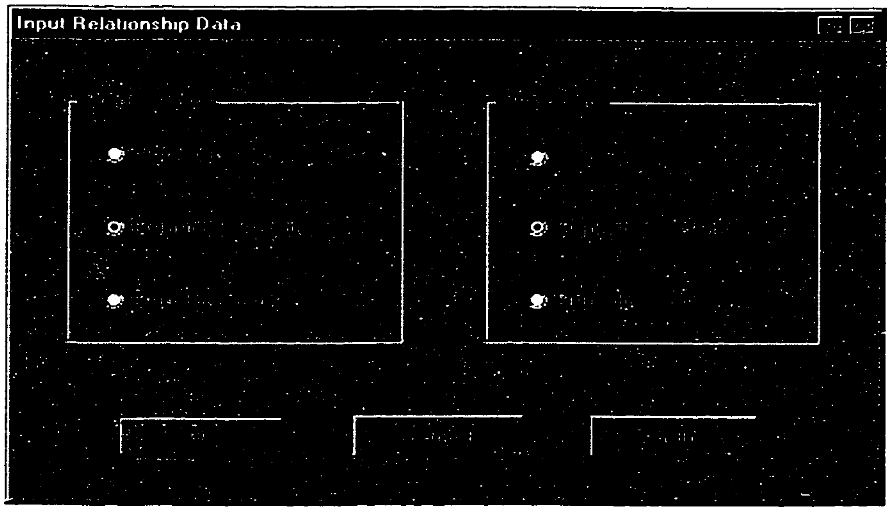
1200

850

1 1 97

21 4 99

Figure 6.27 Crew Dialog Box



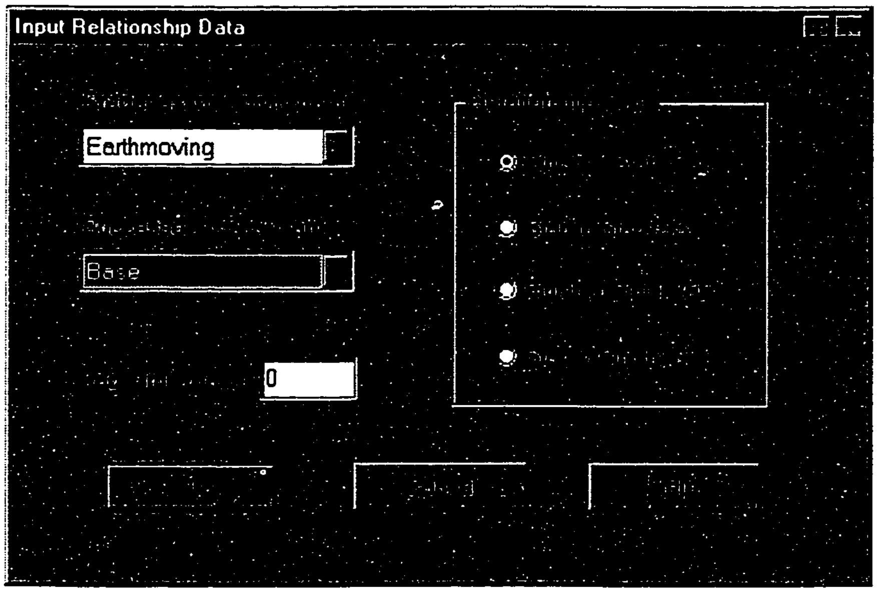


Figure 6.28 Relationship Dialog Box

6.3.3 MDI Windows

LSCHEDULER supports multiple document interface (MDI) as shown in Figure 6.13. MDI enables multiple child windows to be created and constrained within the boundaries of the main application window (Borland C++ 1996). The MDI windows are used to facilitate the display of scheduling results in *LSCHEDULER*.

For example, when the user executes the menu item of *Display* then *Input Data* then *Repetitive Activities*, *LSCHEDULER* creates a child window within its boundaries to display repetitive activities input data as shown in Figure 6.9. The displayed results in the child window can be viewed on screen using the vertical and horizontal scroll bars. The vertical scroll bar is placed on the right side of the window and enables up and down movement within the window, and the horizontal scroll bar is located on the bottom of the window and enables left and right movement. In addition, the displayed results in the child window can be printed out to obtain a hard copy of the scheduling results.

6.4 Input and Output

LSCHEDULER is capable of performing regular scheduling as well as optimized scheduling for repetitive construction projects. In order to perform scheduling calculations in *LSCHEDULER*, the user need to provide necessary input data at the project level and at the activity level including activity relationships as shown in Figure 6.29. At the project level, the user is required to specify: 1) start date of the project (see Figure 6.19); 2) whether or not to consider impact of weather and/or learning curve on scheduling (see Figure 6.20); and 3) whether to perform regular scheduling, minimize project overall cost, or minimize project duration (see Figure 6.21).

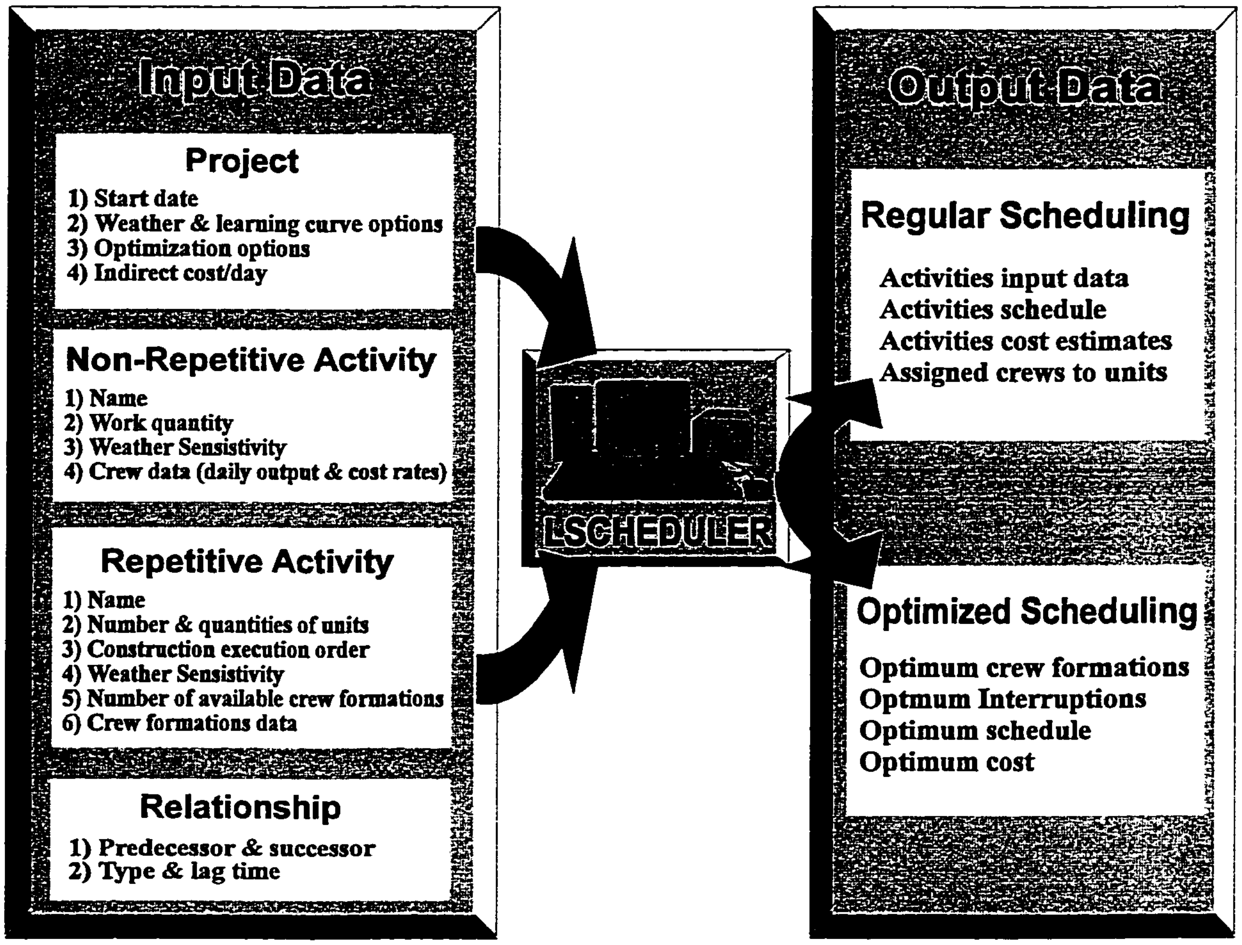


Figure 6.29 ***LSCHEDULER*** **Input and Output**

At the non-repetitive activity level, if any, the user is required to input (see Figure 6.22): 1) activity name; 2) work quantity in units of measurement (e.g. m^3); 3) crew daily output in units/day (e.g. m^3/day); 4) crew cost rates (i.e. material, labor, and equipment); and 5) activity sensitivity to weather. At the repetitive activity level, the needed input data are: 1) activity name; 2) number of repetitive units; 3) work quantity of each repetitive unit; 4) activity sensitivity to weather; and 5) number of available crew formations (see Figure 6.23, Figure 6.24 and

Figure 6.25). For each crew formation, the user need to specify: 1) whether or not to allow interruption of crew work continuity; and 2) number of crews that can work simultaneously (see Figure 6.26). For each of these crews, the user is required to input: 1) crew daily output; 2) crew cost rates (i.e. material, labor, equipment; and interruption); 3) crew availability period on site; and 4) whether or not to allow interruption of crew work continuity (see Figure 6.27). With respect to activity relationship, the needed input data are: 1) predecessor activity name; 2) successor activity name; 3) relationship type; and 4) relationship lag in days (see Figure 6.28).

Given the above input data, *LSCHEDULER* can be used to perform regular scheduling or optimized scheduling for repetitive construction projects. For regular scheduling, *LSCHEDULER* output data is in the form of project schedule or cost estimates. Project schedule can be in working days or in calendar dates and can be displayed for repetitive and non repetitive activities (see Figure 6.10). For a repetitive activity that can be constructed by multiple crews simultaneously, the schedule identifies and displays which crew can be assigned to construct which repetitive unit following the resource-driven scheduling algorithm earlier described in Chapter 4. Project cost estimates provide an estimate of material, labor, equipment and total cost for each repetitive and non-repetitive activity in the project (see Figure 6.11).

For optimized scheduling, *LSCHEDULER* output data can be in the form of a summarized optimization results or a detailed optimization analysis. The summarized optimization results displays the optimum crew formation and interruption vector for each repetitive activity. It also displays the optimum project schedule and cost as shown in Table 6.14. The detailed optimization results displays, for each crew formation of each activity, the generated interruption vectors and the local optimum pair of predecessor crew formation and interruption vector. It should be noted that *LSCHEDULER* can consider the impact of weather and/or learning curve effect for regular as well as optimized scheduling.

6.5 Assumptions and Limitations

LSCHEDULER is developed as a scheduling tool for repetitive construction projects, and therefore its application is limited to this category of projects. In addition, the main limitations and assumptions of *LSCHEDULER* are:

1) Similar to available scheduling software systems, the application of *LSCHEDULER* is limited to the project scheduling stage, and therefore cannot be used during the planning stage to determine project work break-down structure, scope and size of construction activities and repetitive units, etc. Such planning decisions should be made by the project planner before

LSCHEDULER can be used.

2) The application of *LSCHEDULER* is limited to a single repetitive construction project at a time, and therefore it cannot be used to schedule multiple projects sharing the same resource pool.
3) Factors affecting construction productivity considered in *LSCHEDULER* are limited to weather impact and learning curve effect.
4) Estimates of activity durations and costs are assumed to be deterministic.

6.6 Application Example

An example of a highway construction project is analyzed in order to illustrate the use of *LSCHEDULER* and demonstrate its capabilities. The project involves the construction of a three lane highway for a stretch of 5 kilometers, and consists of five consecutive activities: *cut trees*, *remove stumps*, *earthmoving*, *base* and *paving*. The precedence relationships among these sequential activities are finish to start with no lag time. The project can be divided into 5 repetitive sections, each has a length of one kilometer. The five construction activities of the project are repeated at each of the 5 sections or kilometers of the project. The quantities of work for the five activities in each repetitive section are estimated as shown in Table 6.12.

Table 6.12 Quantities of Work

Activity (1)	Unit (2)	km. 1 (3)	km. 2 (4)	km. 3 (5)	km. 4 (6)	km. 5 (7)
Cut trees	m^2	12000	18000	30000	27000	24000
Remove stumps	m^2	12000	18000	30000	27000	24000
Earthmoving	m^3	6000	7000	8600	6500	6000
Base	m^2	30000	30000	30000	30000	30000
Paving	m^2	30000	30000	30000	30000	30000

The contracting method adopted for this project is bidding on cost and time (Herbsman et al 1995). Bidding on cost and time is one of the most popular contracting methods currently being used by many of the state highway agencies across North America in an attempt to reduce the construction duration of highway projects. The use of this method has steadily increased after it was recommended by the Federal Highway Agency in the United States in 1991 (Herbsman 1993, and Herbsman et al 1995). In this method, contractors are asked to bid on both the project cost and duration, and the major criterion for winning is the lowest total combined bid. The total combined bid combines the project cost and the money value of its duration as follows:

$$TCB = A + B = A + (D.DRUC) \tag{6.1}$$

where, TCB = total combined bid in \$, A = project cost in \$, B = money value of project duration in \$, D = project duration in days and DRUC = daily road-user cost in \$/day. DRUC is estimated by transportation agencies to represent the economic benefits of the road to the public and local economy. In this highway project example, the daily road-user cost (DRUC) is estimated at \$14,000/day. The contractor estimates the project indirect cost rate (ICR) to be \$1000/day, and thus DRUC+ICR = \$15,000/day. This value of DRUC+ICR is used to replace ICR as a direct input in the optimization procedure (Equation 5.14) to represent the daily money value for project duration.

In order to minimize the total combined bid for this project, the contractor has to select the optimum crew formation from a set of available alternatives as shown in Table 6.13. For example, a contractor may have to select a crew formation, for the paving activity, from the available alternatives of: a) crew B-25 that consists of: 8 laborers, 3 equipment operators, 1 asphalt paver, 1 tandem roller and 1 pneumatic wheel roller (Means 1993) with no overtime policy; or b) crew B-25 with 3 hours overtime as shown in Table 6.13. Each crew formation is associated with a specific daily output and direct cost. The selection of a particular crew formation for each activity in a highway project, therefore, affects the value of the project total combined bid. In view of the available crew formations for each activity in this project, there are 72 possible combinations of crew formations to construct the project. For example, one combination is to select crew formations 1, 2, 2, 2 and 1 for the five sequential activities, respectively. Each combination is associated with a unique total combined bid for the project, and the contractor has to select the one that yields the minimum total combined bid.

Table 6.13 Crew Formations

Activity (1)	Crew form-ation (2)	Output in m^2/day (4)	Material cost in \$/m^2 (6)	Labor cost in \$/day (7)	Equipment cost in \$/day (8)
Cut trees	1	3000	0	950	1000
Remove stumps	1	4000	0	500	1500
	2	5000	0	1000	3000
	3	6000	0	1500	4000
Earthmoving	1	800	0	630	1700
	2	1100	0	1000	3400
	3	1200	0	1500	4000
Base	1	3000	6	850	1100
	2	3200	6	1000	1300
	3	3400	6	1100	1450
	4	3600	6	1250	1600
Paving	1	4000	7	850	1200
	2	4500	7	1200	1700

LSCHEDULER is used to minimize the total combined bid for this project. In *LSCHEDULER*, the necessary data are input at the project and activity levels. At the project level, the start date of the project is set on January 1, 1997 as shown in Figure 6.19, and the optimization criterion is specified to be minimize project overall cost as shown in Figure 6.21. At the activity level, activity name, number of units, quantities of work, and number of available crew formations are input as shown in Figure 6.23 and Figure 6.24 for each of the five activities of the project. For each crew formation of each activity, interruption option and number of crews that can work simultaneously are specified as shown in Figure 6.26. For each of these crews, crew data are input as shown in Figure 6.27. With respect to activity relationships, the predecessor and successor names, and relationship type and lag are input as shown in Figure 6.28 for each of the project relationships.

After completing all necessary data input, scheduling calculations are initiated by executing "calculate schedule and cost" menu item from the *Project* menu shown in Figure 6.4. This prompts *LSCHEDULER* to apply the optimization procedure earlier described in Chapter 5 to identify the optimum crew formation and associated interruptions for each activity in the project. The scheduling optimization results can be viewed on screen by executing "display optimization output results" menu item as shown in Figure 6.12. The same results can also be sent out to the printer by executing "print" menu item from the *File* menu shown in Figure 6.3. For each activity in this project, the identified optimum crew formation and interruptions as well the optimum schedule and cost are summarized in the *LSCHEDULER* optimization results report as shown in Table 6.14.

Table 6.14 *LSCHEDULER* Optimization Results

OPTIMIZATION RESULTS

Activity	Unit	Crew #	Duration	Interruption	Start	Finish	Cumulative Cost
Optimum Crew Formation #: 1							
Cut trees	1	1	4.0	0.0	0.0	4.0	627150.0
Cut trees	2	1	6.0	0.0	4.0	10.0	627150.0
Cut trees	3	1	10.0	0.0	10.0	20.0	627150.0
Cut trees	4	1	9.0	0.0	20.0	29.0	627150.0
Cut trees	5	1	8.0	0.0	29.0	37.0	627150.0
Optimum Crew Formation #: 1							
Remove stumps	1	1	3.0	0.0	9.5	12.5	772720.0
Remove stumps	2	1	4.5	0.0	12.5	17.0	772720.0
Remove stumps	3	1	7.5	3.0	20.0	27.5	772720.0
Remove stumps	4	1	6.8	2.0	29.0	35.8	772720.0
Remove stumps	5	1	6.0	2.0	37.0	43.0	772720.0
Optimum Crew Formation #: 2							
Earthmoving	1	1	5.5	0.0	12.5	18.0	991008.2
Earthmoving	2	1	6.4	0.0	17.1	23.5	991008.2
Earthmoving	3	1	7.8	4.0	27.5	35.3	991008.2
Earthmoving	4	1	5.9	1.0	35.8	41.7	991008.2
Earthmoving	5	1	5.5	2.0	43.0	48.5	991008.2
Optimum Crew Formation #: 4							
Base	1	1	8.3	0.0	18.7	27.0	2187712.8
Base	2	1	8.3	0.0	27.0	35.3	2187712.8
Base	3	1	8.3	0.0	35.3	43.7	2187712.8
Base	4	1	8.3	0.0	43.7	52.0	2187712.8
Base	5	1	8.3	0.0	52.0	60.3	2187712.8
Optimum Crew Formation #: 1							
Paving	1	1	7.5	0.0	30.3	37.8	3427087.8
Paving	2	1	7.5	0.0	37.8	45.3	3427087.8
Paving	3	1	7.5	0.0	45.3	52.8	3427087.8
Paving	4	1	7.5	0.0	52.8	60.3	3427087.8
Paving	5	1	7.5	0.0	60.3	67.8	3427087.8

6.7 Summary

This chapter presented the implementation stage of the present object-oriented model for scheduling of repetitive construction projects. The model was implemented using C++ programming language as a 32-bit windows application that is named *LSCHEDULER*. *LSCHEDULER* runs on Microsoft Windows 95 and NT and consists of two main modules: scheduling module and user interface module. The scheduling module includes all header and code files, representing the implementation stage for each of the 10 classes identified in the present object-oriented model. For each class, a C++ computer code was developed to represent data members and member functions identified in the analysis and design stages. The user interface module includes header and code files that were developed to provide a user-friendly interface. The user interface of *LSCHEDULER* incorporates menus, a tool bar, a status bar, dialog boxes, and multiple document interface (MDI) windows. An application example of a highway construction project was analyzed to illustrate the use of *LSCHEDULER* and demonstrate its capabilities.

CHAPTER 7

CONCLUSIONS

7.1 Conclusions

An object-oriented model has been developed for scheduling of repetitive construction projects. The model development consists of three stages: analysis, design, and implementation. The first stage is the analysis stage and its purpose is to outline the model classes and their relationships and the sequence of operations that occur in the the model. The analysis stage started by conducting a field-study of a real-life housing project in order to investigate the nature of the scheduling process for repetitive construction and various factors affecting it. The information obtained from this study and from a comprehensive literature review were used to identify the necessary classes in the present object-oriented model. In this stage, 10 classes were identified in order to model the scheduling process of repetitive construction projects.

The second stage in the the development of the present object-oriented model was the the design stage. For each of the 10 classes identified in the present model, this stage provided a detailed design of the data members and member functions of the class. During the design stage, a number of algorithms and procedures were developed in order to perform the necessary scheduling claculations including: 1) a resource-driven scheduling algorithm for repetitive activities; 2) an interruption algorithm; and 3) an optimization procedure.

The algorithm for resource-driven scheduling of repetitive activities was developed as one of the member functions of the *Crew-Formation* class. For each activity in a repetitive unit, the algorithm identifies the scheduled start and finish times as well as the assigned crew. The algorithm provides a schedule that complies with precedence relationships, crew availability and crew work continuity constraints. In addition, it considers the impact of a number of practical factors: 1) type of repetitive activity (i.e. typical or non-typical); 2) multiple crews assigned to work simultaneously on an activity; 3) crew availability period on site; 4) activity interruption; 5) user-specified order of execution among repetitive units; 6) weather impact; and 7) learning curve effect. The scheduling algorithm is carried out in two main stages: the first achieves compliance with precedence relationships and crew availability constraints, and the second further achieves compliance with crew work continuity constraint.

The interruption algorithm was developed as one of the member functions of the *Crew-Formation* class. The algorithm generates feasible interruption vectors for each crew formation in the project and provides added advantage over available formulations that consider arbitrary user-specified interruption vectors prior to scheduling (Russell and Caselton 1988, and Eldin and Senouci 1993): 1) the algorithm is automated and therefore the construction planner need not provide a user-specified set of interruption vectors prior to scheduling; 2) it generates a limited number of feasible interruption vectors and therefore enables a practical and feasible approach for the consideration of interruption during schedule optimization; and 3) the algorithm generates all needed interruption vectors during schedule calculation rather than being limited by a prespecified value prior to scheduling that can be way off optimum conditions, and thus the developed algorithm ensures the generation and selection of the optimum solution.

The optimization procedure is based on a dynamic programming formulation. It was developed as a number of member functions of the *Repetitive* ativity class. Unlike available formulations, the present formulation is capable of incorporating cost in the optimization process, thus offering valuable support to project team members in minimizing the overall cost of the project. For each repetitive activity in the project, the present model assists the planner in selecting the optimum crew formation and interruption vector from a set of possible alternatives. As such, the model can be used to evaluate the impact of different project acceleration strategies (i.e multiple crews, increased crew size, overtime policies, or additional shifts) on the overall cost.

The third and last stage in the the development of the present object-oriented model was the the implementation stage. The model was implemented using C++

programming language as a 32-bit windows application that is named *LSCHEDULER*. *LSCHEDULER* runs on Microsoft Windows 95 and NT and provides a user-friendly interface. The user interface of *LSCHEDULER* incorporates menus, a tool bar, a status bar, dialog boxes, and multiple document interface (MDI) windows.

7.2 Research Contributions

The contributions of this study can be summarized as:

1) The development of an object-oriented model for scheduling of repetitive construction projects. The model enables the integration of repetitive and non-repetitive scheduling techniques in an efficient operating environment.

2) The development of a practical and flexible algorithm for resource-driven scheduling of repetitive activities. The algorithm complies with precedence relationships, crew availability, and crew work continuity constraints and considers a number of practical factors commonly encountered in scheduling construction projects.

3) The development of an interruption algorithm that enables an automated generation of feasible interruption vectors for each crew formation in the project during scheduling. The algorithm circumvents the limitations of

available formulations that consider arbitrary user-specified interruption vectors prior to scheduling.

4) The development of an optimization procedure for generating least cost or least duration schdules for all types of repetitive construction projects.

5) The development of a scheduling calculation procedure that enables the consideration of weather impact and learning curve effect on scheduling of repetitive construction projects.

6) The development of a prototype software system: *LSCHEDULER*. *LSCHEDULER* is a 32-bit windows application that runs on Microsoft Windows 95 and NT and provides user-friendly interface. *LSCHEDULER* enables regular scheduling as well as optimized scheduling of repetitive construction projects, and it incorporates the newly developed algorithms and procedures described in this study.

7.3 Recommendation for Future Research

This study has presented an object-oriented model for regular as well as optimized scheduling of repetitive construction projects. The model is flexible and can be applied to all types of repetitive projects. However in order to expand the potential applications of the model, the following recommendations for future research can be made:

1) The optimization procedure adopted in this model can be expanded to consider risk and uncertainty in estimating activity duration and costs. This can be achieved by transforming the currently applied deterministic dynamic programming formulation to a stochastic one.

2) The resource-driven scheduling algorithm in this model can be extended to account for resource sharing among multiple projects. This can be useful to a contractor managing multiple projects that share the same resource pool.

3) The estimation of crew productivity factors in this model can be expanded to account for other factors in addition to the currently considered weather and learning curve effects. Additional factors that can be considered include but not limited to: overtime policy, space congestion, and change orders.

REFERENCES

AbouRizk, S. M., and Halpin D. W. (1990). "Probabilistic Simulation Studies for Repetitive Construction Processes." **Journal of Construction Engineering and Management**, ASCE, 116(4), 575-594.

Ahuja, H. N. (1984). **Project Management**. John Wiley & Sons, Inc., New York.

Antill, J. M., Woodhead, R. W. (1990). "**Critical Path Methods in Construction Practice**," John Wiley & Sons, Fourth Edition, New York.

Al Sarraj, Z. M. (1990). "Formal Development of Line-of-Balance Technique," **Journal of Construction Engineering and Management**, ASCE, 116(4), 689-704.

Arditi, D., and Albulak M. Z. (1979). "Comparison of Network Analysis with Line of Balance in a Linear Repetitive Construction Project," **Proceedings of the Sixth INTERNET Congress**, Vol. 2, Garmisch-Partenkirchen, W. Germany, 13-25.

Arditi, D., and Albulak M. Z. (1986). "Line-of-Balance Scheduling in Pavement Construction," **Journal of Construction Engineering and Management**, ASCE, 112(3), 411-424.

Arditi, D. (1985). "Construction Productivity Improvement," **Journal of Construction Engineering and Management**, ASCE, 111(1), 1-14.

Ashley, D. B. (1980). "Simulation of Repetitive Unit Construction," **Journal of the Construction Division**, ASCE, 106(CO2), 185-194.

Baldwin, J.R., Manthei, J.M., and Rothbart, H., and harris, R.B. (1971). "Causes of delay in the construction industry," **Journal of the Construction Division**, ASCE, 97(CO2), 177 -187.

Bellman, R. E. (1957). **Dynamic Programming**. Princeton University Press, Princeton, New Jersey.

Bellman, R. E., and Dreyfus S. E. (1962). **Applied Dynamic Programming**. Princeton University Press, Princeton, New Jersey.

Benjamin, N. H., and Greenwald, T. W. (1973). "Simulating effects of weather on construction," **Journal of the Construction Division**, ASCE, 99(CO1), 175 -190.

Birrell, G. E. (1980). "Construction Planning-Beyond the Critical Path," **Journal of the Construction Division**, ASCE, 106(CO3), 389-407.

Booch, G. (1994). **Object-Oriented Analysis and Design with Applications**, Redwood City, CA, Benjamin/Cummings.

Borland C++ (1996). Borland International Inc., Scotts Valley, CA.

Carlson, J. G. H. (1973). "Cubic Learning Curves: Precision Tool for Labor Estimating." **Manufacturing Engineering and Management**, 67(11), 22-25.

Carr, R. I., and Meyer, W. L. (1974). "Planning Construction of Repetitive Building Units," **Journal of the Construction Division**, ASCE, 100(CO3), 403-412.

Canadian Construction Association Business and Contractor Relation Committee (1974)." A Study of Project Planning and Progress Control in the Canadian Construction Industry."

Cantwell, F. A. (1987), "A model for Scheduling and Analyzing Construction Weather Delays," **A Report in Civil Engineering**, Pennsylvania State University.

Choromokos, J., and Davies, E. W. (1972). "CPM Use in Large Construction Firms: A Top Management Survey," **Proceedings of the 1972 INTRNET Conference**, Stockholm, Sweden.

Chrzanowski, E. N., and Johnston D. W. (1986). "Application of Linear Scheduling," **Journal of Construction Engineering and Management**, ASCE, 112(4), 476-491.

Cooper, L., and Cooper M. W. (1981). **Introduction to Dynamic Programming.** Pergamon Press, New York.

Cost Repetition Maintenance. (1963). *Rep.* ST/ECE/HOU/7, United Nations Committee on Housing, Building and Planning, United Nations, New York, N.Y.

Davies, E. W. (1974). "CPM Use in Top 400 Construction Firms," **Journal of the Construction Division**, ASCE, 100(CO1), 39-49.

Denardo, E. V. (1982). **Dynamic Programming; Models and Applications.** Prentice-Hall, Inc., New Jersey.

Department of the Navy, (June, 1969), "Planning Advanced Bases," NAVFAC page 385, Naval Facilities Engineering Command, Washington, D.C.

Diekmann, J. R., Horn, D. L., and O'Conor, M. H. (1982). "Utilization of Learning Curves in Damage for Delay Claims," **Project Management Quarterly**, 67-71.

Dressler, J. (1974). "Stochastic Scheduling of Linear Construction Sites," **Journal of the Construction Division**, ASCE, 100(CO4), 571-588.

Dreyfus, S. E., and Law A. M. (1977). **The Art and Theory of Dynamic Programming**. Academic Press, New York.

Drewin, F. J., (1982). **Construction Productivity**. American Elsevier Publishing Company, New York, N.Y., 101-104.

Eldin, N. N., and Senouci A. B. (1994). "Scheduling and Control of Linear Projects," **Canadian Journal of Civil Engineering**, CSCE, 21, 219-230.

El-Rayes, K. and Moselhi, O. (1997)(a) "Resource-Driven Scheduling of Repetitive Activities on Construction Projects," accepted for publication (24 Oct., 1996) in the **Journal of Construction Management and Economics.**

El-Rayes, K., and Moselhi, O. (1997)(b) "Optimized Scheduling for Highway Construction," **Transactions of the 41st Annual Conference of American Association of Cost Engineers International**, Dallas, Texas, July 13-16, 1997.

El-Rayes, K., and Moselhi, O. (1997)(c) "Computer-Assisted Scheduling for Repetitive Construction," **Proceedings of the 25th CSCE Annual Conference**, Hotel Delta, Sherbrooke, Quebec, May 27-30, 1997.

El-Rayes, K., and Moselhi, O. (1996) "Optimum Resource Utilization For Bidding on a Cost/Time Highway Contract," **Proceedings of the Third Canadian Conference on Computing in Civil and Building Engineering**, Canadian Society of Civil Engineering, Montreal, Quebec, Canada, August 26-28, 1996.

El-Rayes, K., and Moselhi, O. (1994) "LSCHEDULER: for Scheduling of Repetitive Projects," **Proceedings of PMI's 25th Anniversary Seminar/Symposium**, Project Management Institute, Vancouver, Canada, October 17-19, 1994.

Effect of Repetition on Building Operations and Processes On Site. (1965). *Rep.* ST/ECE/HOU/14, United Nations Committee on Housing, Building and Planning, United Nations, New York, N.Y.

Effect of Temperature on Productivity. (1974) National Electrical Contractors Association, Inc., Washington, D.C.

Everett, J. G., and Farghal S. (1994). "Learning Curve Predictors For Construction Field Operations." **Journal of Construction Engineering and Management**, ASCE, 120(3), 603-616.

Gates, M., and Scarpa, A. (1972). "Learning and Experience Curves," **Journal of the Construction Division**, ASCE, 98(CO1), 79-101.

Gates, M., and Scarpa, A., Errata (1976). "Learning and Experience Curves," **Journal of the Construction Division**, ASCE, 102(CO4), p. 689.

Gates, M., and Scarpa, A., (1978). "Optimum Number of Crews." **Journal of the Construction Division**, ASCE, 104(CO2), 123-132.

Grimm, C. T., and Wagner, N. K. (1974). "Weather Effects on Mason Productivity," **Journal of the Construction Division**, ASCE, 100(3), 319-335.

Handa, V. K., and Barcia R. M. (1986). "Linear Scheduling Using Optimal Control Theory," **Journal of Construction Engineering and Management**, ASCE, 112(3), 387-393.

Harbert, J. A. (1976). "Development of the Combined PERT and the LOB (CPL) Chart," **Proceedings of the Fifth INTERNET Congress**, Vol. Friday, Birmingham, U.K.

Harris, F. C., and Evans J. B. (1977). "Road Construction--Simulation Game for Road Construction," **Journal of the Construction Division**, ASCE, 103(CO3), 405-414.

Harris, R. B. (1978). **Precedence and Arrow Networking Techniques for Construction**. John Wiley & Sons, New York.

Harris, F., and McCaffer R. (1989). **Modern Construction Management**. 3rd. ed., BSP Professional Books, Oxford, England.

Hegazy, T., Fazio, P., and Moselhi O. (1993). "BAL: An Algorithm for Scheduling and Control of Linear Projects." **1993 AACE Transactions**, AACE.

Herbsman, Z. J., Chen W. T., and Epstein W. C. (1995) "Time Is Money: Innovative Contracting Methods in Highway Construction,", **Journal of Construction Engineering and Management**, ASCE, 121(3), pp. 273-281.

Herbsman, Z. J., (1995) "A+B Bidding Method-Hidden Success Story for Highway Construction," **Journal of Construction Engineering and Management**, ASCE, 121(4), pp. 430-437.

Hijazi, A. M., AbouRizk, S. M., and Halpin D. W. (1992). "Modeling and Simulating Learning Development in Construction." **Journal of Construction Engineering**

and Management, ASCE, 118(4), 685-700.

Hinze, J., and Couey J. (1989). "Weather in Construction Contracts," **Journal of Construction Engineering and Management**, ASCE, 115(2), 270-285.

Johnston, D. W. (1981). "Linear Scheduling Method for Highway Construction," **Journal of the Construction Division**, ASCE,107(CO2), 247-261.

Kavanagh, D. P. (1985). "SIREN: A Repetitive Construction Simulation Model," **Journal of Construction Engineering and Management**, ASCE, 111(3), 308-323.

Kleinfield, I. (1976). "Manpower Use in High-Rise Residential Construction," **Journal of the Construction Division**, ASCE,102(CO2), 379-383.

Koehn, E., and Brown, G. (1985). "Climatic Effects on Construction," **Journal of Construction Engineering and Management**, ASCE, 111(2), 129-137.

Koehn, E., and Meilhede, D. (1981). "Cold Weather Construction Costs and Accidents," **Journal of the Construction Division**, ASCE, 107(CO4), 585-595.

Laufer, A. and Cohenca, D. (1990). "Factors affecting construction planning outcomes," **Journal of Construction Engineering and Management**, ASCE, 116(CO1), 135 -156.

Levine, H. A., Aliberti E. M., and Ford, B. P. (1976). "The Application of Line of Balance on an International Project," **Proceedings of the Fifth INTERNET Congress**, Vol. Thursday, Birmingham, U.K.

Lumsden, P. (1968). **The Line of Balance Method**. Pergamon, Telgamon Press Ltd., London, England.

Mangin, J. C. (1979). "Scheduling Methods for Repetitive Tasks in Second Stage Building Construction," **Proceedings of the Sixth INTERNET Congress**, Book II, Garmisch-Partenkirchen, W. Germany, 287-295.

Mansur, F. (1990). "Line of Balance Method," **A Major Technical Report**, Centre for Building Studies, Concordia University, Montreal, Quebec, Canada.

Mawdesley, M. J., Askew, W. H., Lees, J., Taylor, J., and Stevens, C. M. (1989). "Time Chainage Charts for Scheduling Linear Projects," **Proceedings of Computing in Civil Engineering**, Atlanta, Ga.

Mckee, K. E. (1981). "Construction Productivity Improvement," **Journal of the**

Construction Division, ASCE, 107(CO1), 35-47.

Means building construction cost data. (1991), 49th annual edition, R. S. Means Co., Inc., Kingston, Mass.

Melin, J. W., and Whiteaker, B. (1981). "Fencing a Bar-Chart," **Journal of the Construction Division**, ASCE, 107(CO3), 497-507.

Melin, J. W. (1984). "Fenced Bar-Charts for Repetitive Projects," paper presented at the ASCE, Spring Convention, Atlanta, Ga.

Moselhi, O., Gong, D., and El-Rayes, K. (1997) "Estimating Weather Impact on Duration of Construction Activities," **Canadian Journal of Civil Engineering**, 24(3), 359-366.

Moselhi, O., Gong, D., and El-Rayes, K. (1995) "A DSS for Estimating Weather Impact on Construction Productivity," **Proceedings of the First Construction Specialty Conference**, Canadian Society of Civil Engineering, Ottawa, Ontario, Canada, June 1-3, 1995, pp. 369-376.

Moselhi, O., and El-Rayes, K. (1993)(a) "Scheduling of Repetitive Projects With Cost Optimization," **Journal of Construction Engineering and Management**, ASCE, 119(4), 681-697.

Moselhi, O., and El-Rayes, K. (1993)(b) "Least Cost Scheduling for Repetitive Projects," **Canadian Journal of Civil Engineering**, 20(5), 834-844.

Moselhi, O., and El-Rayes, K. (1993)(c) "An OOP Model for Scheduling of Repetitive Projects," **Proccedings of Fifth International Conference on Computing in Civil and Building Engineering**, American Society of Civil Engineers, Anaheim, California June 7-9, 1993, pp. 939-946.

Moselhi, O., and Nicholas M. J. (1990). "Hybrid Expert System for Construction Planning and Scheduling," **Journal of Construction Engineering and Management**, ASCE, 116(2), 221-238.

Nicholas (1989), "Hybrid Expert System for Construction Planning and Scheduling," **A thesis in the Centre for Building Studies**, Concordia University, Montreal.

Nunnally, S. W. (1987) **Construction Methods and Management**. 2nd edition, Prentice-Hall of Canada Ltd., Toronto.

O'Brien, J. J. (1965). **CPM in Construction Management**. McGraw-Hill Inc., New

York.

O'Brien, J. J. (1969). **Scheduling Handbook**. McGraw-Hill Inc., New York.

O'Brien, J. J. (1975). "VPM Scheduling for High Rise Buildings," **Journal of the Construction Division**, ASCE, 101(CO4), 895-905.

O'Brien, J. J. (1984). "Network Scheduling Variations for Repetitive Work," paper presented at the ASCE, Spring Convention, Atlanta, Ga.

O'Brien, J. J., Kreitzberg, F. C., and Mikes W. F. (1985). "Network Scheduling Variations for Repetitive Work," **Journal of Construction Engineering and Management**, ASCE, 111(2), 105-116.

Oldrich, S., and Cacha, J. (1982). "Time Space Scheduling Method," **Journal of the Construction Division**, ASCE, 108(CO3), 445-457.

Oglesby, C. H., Parker, H. W., and Howell, G. A. (1989). **Productivity Improvement in Construction**. McGraw-Hill Inc., New York.

Paulson, B. C. (1975). "Estimation and Control of Construction Labor Costs," **Journal of the Construction Division**, ASCE, 101(CO3), 628-633.

Pedersen, H. (1972). "Network Planning of Repetitive Processes in Housing Construction Industry," **Proceedings of the Third INTERNET Congress**, Book II, Stockholm, Sweden, 381-392.

Perera, S. (1982). "Network Planning of Projects Comprising Repetitive Activities," **Proceedings of the IAHS Conference on the Impact of Economy and Technology**, Vienna, Austria, Nov. 15-18, 1982, 927-985.

Perera, S. (1983). "Resource Sharing in Linear Construction," **Journal of Construction Engineering and Management**, ASCE, 109(1), 102-111.

Programming House Building by Line of Balance. (1966). The National Building Agency, London, England.

Reda, R. (1990). "RPM: repetitive project modelling," **Journal of Construction Engineering and Management**, ASCE, 116(2), 316-330.

Rist, P. T. (1972). "Cascade Charts in Construction," **Proceedings of the Third INTERNET Congress**, Book II, Stockholm, Sweden, 415-422.

Roech, W. (1972). "Network Planning and Velocity-Diagrams in Housing

Construction Industry," **Proceedings of the Third INTERNET Congress**, Book II, Stockholm, Sweden, 415-422.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. (1991). **Object-Oriented Modelling and Design**, Englewood Cliffs, Prentice Hall, New Jersey.

Russell, A. D., and Caselton, W. F. (1988). "Extensions to linear scheduling optimization," **Journal of Construction Engineering and Management**, ASCE, 114(1), 36-52.

Russell, A. D., (1990). "REPCON: An Innovative Construction Management System." **Managing Projects (International Symposium on Building Economics and Construction Managent)**, International Council for Building Research Studies and Documentation (Austria), V. Ireland and T. Uher, eds., 6, 405-416.

Russell, A. D., and Wong, W. C. M. (1993). "New Generation of Planning Structures." **Journal of Construction Engineering and Management**, ASCE, 119(2), 196-214.

Sanders S. R., and Thomas, H. R. (1991). "Factors Affecting Masonry-Labor Productivity," **Journal of Construction Engineering and Management**, ASCE, 117(4), 626-644.

Selinger, S. (1980). "Construction planning for linear projects," **Journal of the Construction Division**, ASCE, 106(CO2), 195-205.

Smith, G. R., and Hancher D. E. (1989). "Estimating Precipitation Impacts for Scheduling," **Journal of Construction Engineering and Management**, ASCE, 115(4), 552-566.

Stradal, O. and Cacha, J. (1982). "Time Space Scheduling Method." **Journal of the Construction Division**, ASCE, 108(CO3).

Suhail, S. A., and Neale, R. H. (1994). "CPM/LOB: New Methodology to Integrate CPM and Line of Balance." **Journal of Construction Engineering and Management**, ASCE, 120(3), 667-684.

Tanner, J. P. (1985). "The Learning Curve" **Production Engineering**, 32(5), 72-78.

Thabet, W. Y., and Beliveau, Y. J. (1994). "Modelling Work Space to Schedule Repetitive Floors in Multistory Buildings." **Journal of Construction Engineering**

and Management, ASCE, 120(1), 96-116.

Thomas, H. R., Mathews C. T., and Ward J. G. (1986). "Learning Curve Models of Construction Productivity," **Journal of Construction Engineering and Management**, ASCE, 112(2), 245-258.

Thomas, H. R., Yiakoumis T. (1987). "Factor Model of Construction Productivity," **Journal of Construction Engineering and Management**, ASCE, 113(4), 623-639.

U.S. Army Cold Regions Research and Engineering Laboratory, (1986) "Firms face frigid facts" **Engineering News Record**, March 20, 1986, 168.

Vorster, M. C., and Bafna, T. (1992). "Formal Development of Line-of-Balance Technique, Al Sarraj, Z. M. (1990)" a Discussion in **Journal of Construction Engineering and Management**, ASCE, 118(1), 210-211.

Wales, R. J., AbouRizk, S. M. (1996). "An Integrated Simulation Model for Construction," **Simulation Practice and Theory Journal**, ELSEVIER, 3(1996), 401-420.

Wiener, R. S., Pinson, L. J. (1988). **An Introduction to Object-Oriented Programming and C++**. Adisson-Wesley Publishing Company, Inc.

Wright, T.P. (1936). "Factors Affecting the Cost of Airplanes," **Journal of Aeronautical Science**, (Feb.), 124-125.

APPENDIX I: HEADER FILES

PROJECT HEADER FILE "PROJECT.H"

```
//==========================================================================
//============ Project header file "project.h"
//==========================================================================


#include "data1.h"

//---------- Global defines
#define  TRUE  1

//---------- Prototypes definitions
int my_compare(const void *value1, const void *value2);


//==================== Node class

class node
{
   friend class Project;

   private:
         node *left, *right;
         Project_data* data;
};



//============== Project class

class Project
{
   friend class Activity;
   friend class Non_Rep_Activity;
   friend class Sub_Activity;
   friend class Repetitive;
   friend class Crew_Formation;
   friend class TKhaledApp;
private:
         int no_activities, no_non_rep, no_repetitives, no_subs;
         int no_relations, no_rep_relations, no_hetro_relations;
         int no_rep_quantities, no_proj_formations, no_proj_crews;
         float project_duration, project_cost;
         Activity **project_activities;
         Relation **project_relations;
         Rep_Relation **project_rep_relations;
         Hetro_Relation **proj_hetro_relations;
```

```
        Non_Rep_Activity **project_non_rep;
        Repetitive **project_repetitives;
        Sub_Activity **project_subs;
        Date project_start;
        int weather_sensitivity;
        int learning_curve;
        int optimization;
        int cost_day;
        node* root;
        node* left;
        node* right;

public:
        Project() { root = left = right = 0;
                    project_duration = project_cost = 0.0;
                    no_activities =  no_non_rep = 0;
                    no_subs = no_repetitives =  no_rep_quantities = 0;
                    no_relations =no_rep_relations = no_hetro_relations = 0;
                    no_proj_formations = no_proj_crews = 0;
                    project_relations  = new Relation* [20];
                    project_rep_relations = new Rep_Relation* [20];
                    proj_hetro_relations = new Hetro_Relation* [20];
                    project_activities = new Activity* [40];
                    project_non_rep = new Non_Rep_Activity* [20];
                    project_repetitives = new Repetitive* [20];
                    project_subs = new Sub_Activity *[20];
                    weather_sensitivity = learning_curve = 0;
                    optimization = 0;
                    cost_day = 0;
                    Date project_start;
                  }
        void insert( Project_data* a);
        void modify_project_duration(float duration) { project_duration = duration; }
        void modify_project_cost(float cost) { project_cost = project_cost + cost; }
        void sensitivity();
        void SetWthrLearnOption (int WthrCheckBox, int LearningCheckBox);
        int GetWeatherOption() {return weather_sensitivity;}
        int GetLearnOption() {return learning_curve;}
        int GetOptimizationOption() {return optimization;}
        int GetDailyCost() {return cost_day;}
        void SetOptimizationOption(int i, int c) {optimization = i; cost_day = c;}
        int GetStDay();
        int GetStMonth();
        int GetStYear();
        void add_activity(Activity* );
        void add_non_rep(Non_Rep_Activity* );
        void add_rep(Repetitive* );
        void add_sub(Sub_Activity* );
        void add_relation(Relation* );
        void add_rep_relation(Rep_Relation* );
        void add_hetro_relation(Hetro_Relation* );
        int modify_activity();
        int update_activity();
        void Set_project_start(Date & d1) { project_start == d1; }
        int is_present ( char* a );
        int repetitive_is_present ( char* a );

        Activity* address_activity ( char* a );  // return address of activity from its name
        Non_Rep_Activity* address_nonrep_activity ( char* a );
        Repetitive* address_rep_activity ( char* a );  // return address of rep. activity from its name
        int start_scheduling();
        void schedule();
        void sort();
        void clear( node* n = 0, int first = 1); // Call clear without any parameters
        int save_file(char * s);
        int open_file(char * s);
        ~Project() { clear();
                delete[] project_activities;
                delete[] project_non_rep;
                delete[] project_repetitives;
                delete[] project_subs;
                delete[] project_relations;
                delete[] project_rep_relations;
                delete[] proj_hetro_relations;
                }
};
```

DATA HEADER FILE "DATA.H"

```
//==================================================================
//=========== Data Hierarchy Header File "data.h"
//==================================================================



//====================== Class Project_data

class Project_data
{
        friend class Project;

protected:
        char name[40];
        Project *project_address;
public:
         Project_data( char * n) {   strcpy(name,n); }
         virtual void set_addresses() {}
         char * get_name() {return name;}
         Project_data() { name[0] = '\0'; project_address = NULL;}
         virtual ~Project_data() {}
};



//========================= Class Relation

class Relation : public Project_data
{
        friend class Activity;
        friend class Project;
protected:
        char pred_name[40], succ_name[40];
        Activity *Apred;
        Activity *Asucc;
        int type, lag;
public:
        Relation() { Apred = Asucc = NULL; type = 1; lag = 0; pred_name[0] ='\0'; succ_name[0] = '\0'; }
        Relation(char* n, Activity *p, Activity *s, int t, int l);
        virtual void send_mess_succ(float*, float* );
        virtual void send_mess_pred(float*, float* );
        virtual void send_mess_rep_succ(float* , float*, float, int) {}
        virtual void send_mess_rep_pred(float* , float*) {}
        void display_data();
        virtual void set_addresses();
        virtual ~Relation() {}
};
```

```
//================================ Class Rep_Relation

class Rep_Relation : public Relation
{
        int no_relations;
public:
        Rep_Relation() : Relation () { no_relations = 0;}
        Rep_Relation(char* n, Activity *p, Activity *s, int t, int l);
        void build_rep_succ_PES(int );
        void send_mess_succ(float*, float* ) {}
        void send_mess_pred(float*, float* ) {}
        void send_mess_rep_succ(float* , float*, float, int, int, int);
        void send_mess_rep_pred(float* , float*);
        void identify_options(int, int);
        void display_data();
        void set_rep_addresses();
        virtual ~Rep_Relation() {}
};


//================================ Class Hetro_Relation

class Hetro_Relation : public Relation
{
        int pred_unit_no, succ_unit_no;

public:
        Hetro_Relation() : Relation () { pred_unit_no = succ_unit_no = 0;}
        Hetro_Relation(char* n, Activity *p, Activity *s, int t, int l, int pred_no, int succ_no);
        void send_mess_succ(float*, float* );
        void send_mess_pred(float*, float* );
        void display_data();
        void set_hetro_addresses();
        virtual ~Hetro_Relation() {}
};


//================================ Class Activity

class Activity : public Project_data
{
        friend class Project;
        friend class Relation;
        friend class Rep_Relation;
        friend int my_compare(const void *value1, const void *value2);
protected:
        Relation *Rsucc[5], *Rpred[5];
        int no_pred, no_succ,mess_pred,mess_succ;
        float possible_ES, possible_LF;
        int actual_start, actual_finish;
        int weather_type;
public:
        Activity() : Project_data()
                {
                 no_pred = no_succ =mess_pred = mess_succ = weather_type = 0;
                 possible_ES = possible_LF = 0.0;
                 actual_start = actual_finish = 0;
                 for (int i = 0; i < 5; i++) Rsucc[i] = Rpred[i] = NULL;
                }
        Activity(char* n) : Project_data(n)
                {
                no_pred = no_succ =mess_pred = mess_succ = weather_type = 0;
                possible_ES = possible_LF = 0.0;
                actual_start = actual_finish = 0;
                for (int i = 0; i < 5; i++) Rsucc[i] = Rpred[i] = NULL;
                }
        virtual void add_succ_relation(Relation *R);
        virtual void add_pred_relation(Relation *R);
        virtual void modify_ES(int&, int&, float&, float& ){}
        virtual void modify_LS(int&, int&, float&, float& ){}
        virtual void set_to_zero() {}
        virtual void enter_cost_data() {}
        virtual int get_no_units() { return 0; }
        virtual void add_rep_succ_relation(Rep_Relation *) {}
        virtual void add_rep_pred_relation(Rep_Relation *) {}
        virtual void increment_mess_rep_pred() {}
        virtual void increment_mess_rep_succ() {}
        virtual void increment_mess_pred() {}
        virtual void increment_mess_succ() {}
        virtual void modify_ES_array(int, float, float, float, int) {}
        virtual void modify_LF_array(int, float, float, float, int) {}
        virtual void start_forward_calculations(float, int, int, int) {}
        virtual void start_backward_calculations() {}
        virtual void identify_optimum_options (int, int) {}
        virtual int modify_my_data(char *, int, int, int, int, int) {return 0;}
        int update_my_data();
        virtual void display_calendar_dates(){}
        virtual void display_working_days(){}
        virtual void display_cost() {}
        virtual void display_update(){}
        virtual void build_crew_PES_arrays(int){}
        void set_addresses();
        virtual void set_rep_addresses() {}
        virtual ~Activity() {}
};
```

```
//======================== Class Non_Rep_Activity

class Non_Rep_Activity : public Activity
{
        friend struct NonRepTransferStruct;

        int  mat_cost_unit, lab_cost_day, equip_cost_day;
        float prod_factor;
        int quantity;
        int  productivity;

public:
        Non_Rep_Activity() : Activity()
                { mat_cost_unit = lab_cost_day = equip_cost_day = 0;
                quantity = 0; productivity = 1; prod_factor = 1.0; }
        Non_Rep_Activity(char* n,int q, int p);
        void modify_ES(int&, int&, float&, float& );
        void modify_LS(int&, int&, float&, float& );
        void enter_cost_data(int, int, int);
        void set_to_zero() { mess_pred =mess_succ = 0; possible_ES = possible_LF= 0.0;
                        prod_factor = 1.0;}
        void display_calendar_dates();
        void display_working_days();
        void display_cost();
        void display_data();
        void display_update();
        void set_rep_addresses() {}
        Non_Rep_Activity* return_my_address2() { return this; }
        int modify_my_data(char *, int, int, int, int, int);
        virtual ~Non_Rep_Activity() {}
};


//======================== Class Crew_Formation

class Crew_Formation
{
        friend class Repetitive;
        friend class Project;
        friend class CrewFormationDialog;
        friend class CrewDialog;
        friend struct CrewFormationTransferStruct;
        friend struct CrewTransferStruct;

        int no_crews, current_no_inter, file_no_inter, inter_type;
        int *mat_cost_unit, *lab_cost_day, *equip_cost_day;
        int *early_avail_date, *late_avail_date, *assigned_crew;
        int *inter_OK, *inter_cost_day, *Imax, **interruption_array;
        float *ES_array, *LF_array, *duration;
        float *productivity, *prod_factor;
```

```
public:
          Crew_Formation()
                    {
                    no_crews = 1;
                    current_no_inter = file_no_inter = 0;
                    inter_type = 2;
                    mat_cost_unit = lab_cost_day = equip_cost_day = NULL;
                    early_avail_date = late_avail_date = assigned_crew = NULL;
                    inter_OK = inter_cost_day = Imax = NULL;
                    interruption_array = NULL;
                    duration = ES_array = LF_array = NULL;
                    productivity = prod_factor = NULL;
                    }
          void build_data_arrays(int no_units);
          void schedule_stage1 (Project * proj, int no_units, int *quantity, float *PES, int *execution_order,
                               float *F, float *shift, int *Inter);
          void   forward_schedule(Project  *  proj,  int  no_units,  int  *quantity,  float  *PES,  int
                               *execution_order, int *Inter);
          void calculate_multiple_interruptions(int no_units, int *execution_order);
          void calculate_single_interruptions(int no_units, int *execution_order);
          void clear_inter_array(int no_units);
          void operator==(Crew_Formation&); // overloaded assignment operator
          ~Crew_Formation();

};


//========================== Class Repetitive

class Repetitive : public Activity
{
          friend struct RepDataTransferStruct;
          friend struct TypRepTransferStruct;
          friend struct NonTypRepTransferStruct;
          friend struct CrewFormationTransferStruct;
          friend struct CrewTransferStruct;
          friend class TypRepDialog;
          friend class NonTypRepDialog;
          friend class CrewFormationDialog;
          friend class CrewDialog;
          friend class Project;

protected:
          int no_units, chosen_formation, chosen_inter_vector, no_crew_formations;
          int mess_rep_pred, mess_rep_succ, no_rep_pred, no_rep_succ;
          int *quantity, *interruption_vector, *execution_order;
          float *PES;
          Rep_Relation *Rep_succ[5], *Rep_pred[5];
          Crew_Formation *crew_formations;
```

```
public:
        Repetitive() : Activity()
         {
                no_units = 1;
                chosen_formation = chosen_inter_vector = 0;
                no_crew_formations = 1;
                mess_rep_pred = mess_rep_succ = no_rep_pred = no_rep_succ = 0;
                quantity = interruption_vector = execution_order = NULL;
                PES = NULL;
                for (int i = 0; i < 5; i++) Rep_succ[i] = Rep_pred[i] = NULL;
                crew_formations = NULL;
         }
        Repetitive(char *n, int no);
        void build_quantity_array();
        void accept_quantities(int i, int quant, int order, int inter);
        void accept_quantities(int i, int quant);
        void build_available_crews(int no_options);
        void accept_crew_formation_info(int, int, int);
        void accept_crew_info(int, int, int, int, int, int, int, int, int, int);
        int get_no_units() { return no_units; }
        int modify_my_data();
        void add_rep_succ_relation(Rep_Relation *R) { Rep_succ[no_rep_succ] = R; no_rep_succ++;}
        void add_rep_pred_relation(Rep_Relation *R) { Rep_pred[no_rep_pred] = R; no_rep_pred++;}
        void increment_mess_rep_pred() { mess_rep_pred++; }
        void increment_mess_rep_succ() { mess_rep_succ++; }
        void increment_mess_pred() { mess_pred++; }
        void increment_mess_succ() { mess_succ++; }
        void modify_ES_array(int, float, float, float, int);
        void optimize_stage1(float pred_cost, int pred_crew, int pred_inter);
        void modify_LF_array(int, float, float, float, int);
        void start_forward_calculations(float, int, int, int);
        void start_backward_calculations();
        void start_forward_optimization();
        void identify_optimum_options(int, int);
        void last_rep();
        void set_to_zero();
        void display_calendar_dates();
        void display_working_days();
        void display_cost();
        void display_data();
        void display_inter_analysis();
        void display_inter_results();
        void set_rep_addresses();
        void operator==(Repetitive&); // overloaded assignment operator
        ~Repetitive() {
                        if ( quantity != NULL ) delete[] quantity;
                        if (interruption_vector != NULL) delete[] interruption_vector;
                        if (execution_order != NULL) delete[] execution_order;
                        if (PES != NULL) delete[] PES;
                        if ( crew_formations != NULL) delete[] crew_formations;
                        }
};
```

```
//========================== Class Sub_Activity
class Sub_Activity : public Activity
{
        int  duration, material_cost, labor_cost, equip_cost;

public:
        Sub_Activity() : Activity()
                { duration = material_cost = labor_cost = equip_cost = 0; }
        Sub_Activity(char* n,int d);
        void modify_ES(int&, int&, float&, float& );
        void modify_LS(int&, int&, float&, float& );
        void enter_cost_data();
        void set_to_zero()
            { mess_pred =mess_succ = 0;
                possible_ES = possible_LF= 0.0;}
        void display_data();
        void display_calendar_dates();
        void display_working_days();
        void display_cost();
        void display_update();
        int modify_my_data() {return 0;}
        void set_rep_addresses() {}
};
```

CALENDAR HEADER FILE "CALENDAR.H"

```
//=================================================================
//================ Calendar Header File "Calendar.h"
//=================================================================


//==============================Julian date class

class Julian
{
public:
   int da, yr;
   Julian() {}
   Julian(int d, int y) { da = d; yr = y; }
   long operator-(Julian&);
   friend int operator<(Julian& jd1, Julian& jd2)
          { return jd1.yr < jd2.yr ? 1:
                   jd2.da < jd2.da ? 1: 0; }
};


//==================================Date class

class Date
{
   int  rem, mo, da, yr;
   char weekday[5];
public:
   Date() { mo = 1; da = 1; yr = 97; rem = 0; weekday[0] = '\0'; }
   Date(int m, int d, int y);
   Date(Julian);              // constructor coversion function
   operator Julian();         // member conversion function
   void display( char * a);
   Date operator+=(int);      // overloaded += operator to add integer to a date
   Date operator+(int);       // overloaded + operator to add integer to a date & count for weekends
   Date operator-=(int);      // overloaded -= operator to subtract integer from a date
   Date operator-(int);       // overloaded - operator to subtract integer from a date & count for weekends
   int operator-=(Date);      // overloaded -= operator to get duration between two dates in calendar days
   int operator-(Date);       // overloaded -  operator to get duration between two dates
   void operator==(Date&); // overloaded assignment operator
   friend int operator>(Date& d1,Date& d2)
          { return d1.yr > d2.yr ? 1 :
                (d1.yr == d2.yr) && (d1.mo > d2.mo) ? 1 :
                (d1.yr == d2.yr) && (d1.mo == d2.mo) && (d1.da > d2.da) ? 1 : 0; }
   int GetDay() {return da;}
   int GetMonth() {return mo;}
   int GetYear() {return yr;}
   float prod_day();        // productivity loss function for one day
   float prod_period(Date);  // productivity loss function over a period
};
```