

**THE SEQUENCE STEP ALGORITHM
A SIMULATION-BASED SCHEDULING ALGORITHM FOR REPETITIVE
PROJECTS WITH PROBABILISTIC ACTIVITY DURATIONS**

by

Chachrist Srisuwanrat

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Civil Engineering)
in The University of Michigan
2009

Doctoral Committee:

Professor Photios G. Ioannou, Chair
Lecturer John G. Everett
Assistant Professor Vineet R. Kamat
Assistant Professor Mark P. Van Oyen

© Chachrist Srisuwanrat 2009
All Rights Reserved

To my wonderful Father, Warin Srisuwanrat, My beloved Grandmother, Sei-Ngek Lim
and my family, Mas, Rasamee, Sumrit, and Rangson Srisuwanrat
For their love, understanding, encouragement, and support.

ACKNOWLEDGMENTS

My academic journey would not have been possible without my beloved and respected advisor, Professor Photios G. Ioannou. Great indebtedness goes to his belief in me and the opportunity he has offered to continue my academic journey at the University of Michigan. It has been an honor to work under his direction. During the development of this research, he has always reinforced to me that we can make a difference, a great one. And here I am, with this research. His interesting guidance, valuable advice, and constructive skepticism have contributed to the achievement of this research. I wish him and his family a blissful and healthy life.

The knowledge and input I learnt from my committee members and their courses have provided profundity and richness to this research. I have combined all the knowledge learnt from Professor Photios G. Ioannou in scheduling and simulation, Dr. John G. Everett in construction management, Dr. Vineet R. Kamat in simulation visualization, and Dr. Mark P. Van Oyen in simulation and simulation system development. Therefore, I dedicate this research to all of my committee.

One committee member, Dr. John G. Everett, has been the best teacher and also friend that anyone could ask for. A great person and valuable asset to the Construction Engineering and Management department at the University of Michigan, he has

continuously supported and encouraged me through these years. His support, humor, and kindness will always be remembered.

I would like to express my sincere thanks to Vineet R. Kamat. An intelligent and diligent researcher, he is a driving force in modern construction research. I believe his hard work and vision will improve the construction industry and will be an excellent example for others, including me. He has given me academic and life experience by offering opportunities to prove and strengthen myself during the course of my journey at the University of Michigan. It has been a privilege to have such opportunities.

Special acknowledgement goes to Professor Julio C. Martinez for his technical support in Stroboscope. His thesis and Stroboscope inspire both my thesis and my application for this research, called “ChaStrobe.” Without his support and Stroboscope, I may have taken a more difficult path in establishing the application. In addition, I would like to thank the following researchers who have paved the foundations of repetitive project scheduling, used to originate new ideas in this research: Robert B. Harris, Robert I. Carr, I-Tung Yang, David W. Johnston, David B. Ashley, David Arditi, James D. Lutz, Keith C. Crandall, Simaan M. AbouRizk, Onur B. Tokdemir, and Khaled A. El-Rayes. Their works are valuable to this research and respected.

During the course of writing this thesis, it would have been difficult for this work to come to fruition without assistance and advice from Krista Osborne, Robin Roots, and Professor Leslie Olsen. Numerous hours of reviewing and editing on this thesis ensued conception culminating in the finalization. I extend my heartfelt appreciation to them.

I want to express my sincere gratitude towards my dad, Warin Srisuwanrat, my grandmother, Sei-Ngek Lim, and the rest of my family. Without their encouragement, I

could not have gone through this long journey. Especially for my dad, his kindness, patience, and love significantly contribute to the accomplishment of this research as well as my life.

Many thanks go to the friends I met at the University of Michigan. Kittinun Sirijaroonchai, one of my best friends and the nicest person I have ever met, has helped me during these years. To Theerathon Tharachai and Visit Likhitrungsilp who were like my brothers at the University of Michigan, I will always appreciate their company and wisdom. Rita Awwad and Mustafa Saadi, two smart and kind people, have always assisted me in reviewing my works and publications. Their friendship is appreciated. I wish them all the success they pursue in both academia and life. Lastly, I want to thank Kanyapat Pakdipadungdan for her love and encouragement. She is my oasis at the end of the day.

TABLE OF CONTENTS

DEDICATION.....	ii
LIST OF FIGURES	xi
LIST OF TABLES	xix
LIST OF APPENDICES	xxi
ABSTRACT.....	xxii
CHAPTER 1 INTRODUCTION.....	1
1.1 Repetitive Construction Projects.....	2
1.2 Resource Constraints	3
1.3 Characteristics of Repetitive Activities and Projects.....	6
1.3.1 Typical and Non-Typical Activities.....	6
1.3.2 Repetitive and Non-Repetitive Activities	7
1.3.3 Deterministic and Non-Deterministic Durations	9
1.3.4 Hard and Soft Logic Dependencies	10
1.3.5 Resource-Sharing Activities	12
1.4 Problem Description	13
1.5 Existing Scheduling Techniques.....	14
1.6 Challenges.....	15
1.7 Research Objectives.....	16
1.8 Conclusion	16
CHAPTER 2 LITERATURE REVIEW	19
2.1 Introduction.....	20
2.2 Existing Techniques.....	21
2.2.1 Critical Path Method (CPM).....	21
2.2.2 Project Evaluation Review Technique (PERT)	24
2.2.3 Line-Of-Balance (LOB).....	24

2.2.4	Other Graphical Approaches.....	29
2.2.5	Linear Programming (LP).....	35
2.2.6	Dynamic Programming (DP)	37
2.2.7	Simulation	41
2.3	Summary	45
CHAPTER 3 REPETITIVE SCHEDULING METHOD.....		48
3.1	Introduction.....	49
3.2	Maintaining Continuity in Graphical Methods	51
3.3	Critical Activities and Controlling Sequence	56
3.3.1	Critical Activities	57
3.3.2	Controlling Sequence.....	59
3.4	Summary	61
CHAPTER 4 SEQUENCE STEP ALGORITHM.....		64
4.1	Sequence Steps.....	65
4.1.1	Example 4.1 Determining sequence steps for a repetitive project with three activities	65
4.2	Two Different Types of Idle Time in Repetitive Activities.....	66
4.3	Confidence Levels and Crew Lead Times	71
4.4	Overview of the Sequence Step Algorithm	72
4.4.1	Example 4.3 Determining crew lead time in a repetitive project with three activities with probabilistic activity duration.....	73
4.5	Flow Chart of the Sequence Step Algorithm	79
4.5.1	Example 4.4 Scheduling a repetitive project with 7 activities.....	85
4.6	Discussion of Results from the Sequence Step Algorithm	94
4.7	Selection of Confidence Levels	97
4.8	Summary	101
CHAPTER 5 SIMULATION MODEL TEMPLATES.....		103
5.1	Simulation Model for Repetitive Projects.....	104
5.2	Simulation Model Templates.....	108
5.2.1	Work Flow Template	110
5.2.2	Resource Flow Template	114

5.3	Example 5.1 Simulation code and model for a repetitive project.....	117
5.3.1	Simulation Code for Model Parameters (MP)	121
5.3.2	Simulation Code for Programming Objects (PO).....	123
5.3.3	Simulation Code for Model Elements (CME)	130
5.3.4	Control Statements (CS)	137
5.4	Summary	145
CHAPTER 6	WORK BREAKS	149
6.1	Introduction of Work Breaks	150
6.2	Candidate Work Break Positions	155
6.2.1	Control Points and Controlling Sequences	155
6.2.2	Relative Production Rates.....	161
6.2.3	Determining Effective Work Break Positions	162
6.3	Determining Work Break Duration	164
6.4	Example 6.1 Repetitive project with work breaks	167
6.5	Simulation Model and Code for Example 6.1	176
6.5.1	Simulation Code for Model Parameters (MP)	178
6.5.2	Simulation Code for Programming Objects (PO).....	180
6.5.3	Simulation Code for Model Elements (CME)	190
6.5.4	Simulation Code for Controlling Statements (CS)	196
6.6	Summary	205
CHAPTER 7	RESOURCE-SHARING ACTIVITIES.....	208
7.1	Considerations in Scheduling Resource-Sharing Activities	210
7.2	Examples of Repetitive Projects with Resource-Sharing Activities X and Y	211
7.2.1	Example 7.1	211
7.2.2	Example 7.2	216
7.2.3	Example 7.3	219
7.2.4	Example 7.4	226
7.2.5	Example 7.5	229
7.2.6	Example 7.6	234
7.3	Summary	239
CHAPTER 8	CHASTROBE APPLICATION	243

8.1	Overview of the ChaStrobe Application.....	244
8.1.1	Inputs for ChaStrobe	246
8.1.2	Simulation Output from ChaStrobe	255
8.1.3	Automation in ChaStrobe	256
8.1.4	Capabilities of ChaStrobe	257
8.2	Examples of Repetitive Projects in ChaStrobe	258
8.2.1	Example 8.1 Simple repetitive project.....	259
8.2.2	Example 8.2 Repetitive project with work breaks	266
8.2.3	Example 8.3 Repetitive project with resource-sharing activities.....	273
8.2.4	Example 8.4 Repetitive project with resource-sharing activities and work breaks	283
8.3	ChaStrobe's Output.....	288
8.3.1	Project Duration Graphs for each Processing SQS.....	289
8.3.2	Static Graphs.....	291
8.3.3	ChaStrobe's Analyzer	294
8.3.4	Schedule in Microsoft Project.....	303
8.4	Summary	304
CHAPTER 9 OPTIMIZATION IN CHASTROBE		307
9.1	Overview of ChaStrobe's Optimization.....	308
9.2	Optimization Input	311
9.2.1	Search Inputs.....	312
9.2.2	Dynamic Code Input	315
9.2.3	Additional Consistent Code	318
9.2.4	Search Parameters	319
9.3	Simulation Code and Model Manipulation.....	321
9.4	Search Output from ChaStrobe's Optimization	324
9.5	Search Methods in ChaStrobe.....	325
9.5.1	The Exhaustive Search.....	325
9.5.2	The Genetic Algorithm	326
9.6	Example 9.1 Optimizing a Repetitive Project.....	331
9.7	Summary	341

CHAPTER 10	CONCLUSTIONS AND RECOMMENDATIONS	346
10.1	Summary	346
10.2	Contributions.....	349
10.3	Recommendations.....	351
APPENDICES		354
BIBLIOGRAPHY		412

LIST OF FIGURES

Figure 1.1 Scheduling repetitive project with different types of constraints	4
Figure 1.2 Typical and non-typical activities	7
Figure 1.3 Repetitive and non-repetitive activities	8
Figure 1.4 A repetitive project with soft logic dependencies	11
Figure 3.1 CPM network for three repetitive units (from Harris and Ioannou 1998).....	51
Figure 3.2 RSM Diagram for Three Units based on Precedence Constraints	52
Figure 3.3 Postponing activities with interruptions in Figure 3.2 to achieve continuous resource utilization.....	53
Figure 3.4 Satisfying precedence and continuity constraints.....	55
Figure 3.5 Different critical activities between CPM and RSM.....	58
Figure 3.6 Production Diagram from RSM with controlling sequence	60
Figure 4.1 Precedence diagram for Example 4.1	65
Figure 4.2 Precedence diagram for Example 4.2.....	66
Figure 4.3 Production diagram showing a delay in Activity B by its Unit Idle Time	68
Figure 4.4 Arrival idle time between the arrival date and the start date of Resource B ...	69
Figure 4.5 SQS-AL's assumption of resource arriving at the beginning of the project ...	71
Figure 4.6 Determining crew lead time from the collected crew idle time and user- specified confidence level.....	72
Figure 4.7 The single unit precedence diagram for Example 4.3	73
Figure 4.8 Determining crew idle time of Resource B in processing SQS2 in Example 4.3	74
Figure 4.9 Changes in the resource arrival dates before and after processing SQS2 for Example 4.3	76

Figure 4.10 Determining crew idle time for Resource C during processing SQS3 for Example 4.3	77
Figure 4.11 Changes in resource arrival dates before and after processing SQS3 for Example 4.3	78
Figure 4.12 Flow Chart of the Sequence Step Algorithm.....	80
Figure 4.13 Replication loop and sequence step loop in SQS-AL	81
Figure 4.14 The single unit precedence diagram for Example 4.4	85
Figure 4.15 The first replication production diagram from processing SQS2 (collecting CIT_B and CIT_C) for Example 4.4	87
Figure 4.16 The first replication production diagram from processing SQS3 (collecting CIT_D , CIT_E , and CIT_F) for Example 4.4	89
Figure 4.17 The first replication production diagram from processing SQS4 (collecting CIT_G) for Example 4.4.....	91
Figure 4.18 The first replication production diagram from processing SQS5 for Example 4.4.....	92
Figure 4.19 Cumulative distributions of project duration at an 80% confidence level	96
Figure 4.20 Decreasing idle time and increasing project duration as SQS-AL progresses with 5 different confidence levels for Example 4.4.....	98
Figure 4.21 Seven density functions of project duration for 5 different confidence levels, CPM, and RSM.....	100
Figure 4.22 Seven different cumulative distributions of project duration from five different confident levels, CPM, and RSM.....	100
Figure 5.1 A precedence diagram for a repetitive project	105
Figure 5.2 A single unit precedence diagram for the repetitive project in Figure 5.1	105
Figure 5.3 Models for activities and resources in a single unit precedence diagram	106
Figure 5.4 Separate models for activities and resources in a single unit precedence diagram	107
Figure 5.5 Using separate models to model a repetitive project with resource-sharing activities	108
Figure 5.6 Work Flow Template and Resource Flow Template.....	109
Figure 5.7 Work flow template (work flow sub-network for Activity ACT).....	111

Figure 5.8 Two work flow sub-networks for Activities A and B	113
Figure 5.9 Resource flow template (resource flow sub-network for Resource RES).....	115
Figure 5.10 Single Unit Precedence Diagram.....	118
Figure 5.11 Simulation model for Example 5.1.....	120
Figure 5.12 Assigning semaphore and duration for ResB_CLT Combi.....	132
Figure 5.13 Assigning semaphore and duration for B_Perform Combi	133
Figure 5.14 Assigning Strength for Link iResB_Stay	135
Figure 5.15 Assigning strength for iResB_Leave Link	136
Figure 5.16 Collecting CIT _B during processing SQS2	137
Figure 6.1 The CPM schedule with 105-day project duration and 75-day idle time.....	151
Figure 6.2 RSM schedule with an increased project duration from 105 to 135 days.....	152
Figure 6.3 The work break B2-B3 reducing project duration from 135 to 115 days	154
Figure 6.4 The control point between Activities A and B at A3-B3	156
Figure 6.5 The control point between Activities A and B at A2-B2	156
Figure 6.6 The controlling sequence (A1 to A4, B3 to B2, C1 to C4, and D4).....	157
Figure 6.7 Determining the controlling sequence for the 1 st replication	158
Figure 6.8 Determining the controlling sequence for the 2 nd replication	159
Figure 6.9 Determining the controlling Sequence for the 3 rd replication	160
Figure 6.10 Single unit precedence diagram for Example 6.1	168
Figure 6.11 CPM schedule with 277-day project duration and 438-day idle time	169
Figure 6.12 SQS-AL schedule without work breaks with 449-day project duration and 1- day idle time.....	169
Figure 6.13 SQS-AL schedule with 2 work breaks with 376-day project duration for Example 6.1	171
Figure 6.14 SQS-AL schedule with 3 work breaks and 342-day project duration for Example 6.1	174
Figure 6.15 Simulation Model for the Example in Figure 6.10.....	177
Figure 7.1 Precedence diagram with independent resource-sharing	211
Figure 7.2 SQS2 Schedule where the simulation model for X is created before Y	212
Figure 7.3 SQS2 Schedule with additional X_Perform semaphore.....	213
Figure 7.4 SQS2 schedule with additional Y semaphore	214

Figure 7.5 Finalized schedule with additional Y semaphore from Figure 7.4.....	215
Figure 7.6 Finalized schedule with additional X semaphore	216
Figure 7.7 Finalized schedule with additional Y semaphore	216
Figure 7.8 Precedence diagram with independent resource-sharing activities in different SQSs.....	217
Figure 7.9 SQS1 to SQS4 Schedules, given CLT_{XY} has not been assigned.....	217
Figure 7.10 Finalized schedule after assigned CLT_{XY} either at the end of processing SQS2, SQS3, or SQS4, from Figure 7.9.....	218
Figure 7.11 Precedence diagram with independent resource-sharing activities in different SQSs.....	219
Figure 7.12 SQS2 schedule.....	220
Figure 7.13 Finalized SQS-AL schedule, given CLT_{XY} is derived from the SQS2 schedule, idle time in RES_{XY} is 1 week	221
Figure 7.14 SQS3 schedule without delaying Activity X in SQS2, developed from Figure 7.12.....	222
Figure 7.15 Finalized SQS-AL schedule using the CIT of RES_{XY} from SQS3, developed from Figure 7.14	222
Figure 7.16 SQS2 shedule with additional Y semaphore	223
Figure 7.17 SQS3 schedule, developed from Figure 7.16.....	223
Figure 7.18 Finalized schedule, developed from Figure 7.17.....	224
Figure 7.19 SQS2 schedule with additional X semaphore	225
Figure 7.20 SQS3 and SQS4 schedules, developed from Figure 7.19	225
Figure 7.21 A precedence diagram with directly dependent resource-sharing activities X and Y.....	226
Figure 7.22 SQS2 schedule.....	227
Figure 7.23 Modifying Activity A's duration and comparing the duration to the combined durations of X and Y.....	228
Figure 7.24 Decreasing project duration in the SQS2 schedule due to additional X semaphore	228
Figure 7.25 Precedence diagram with indirectly dependent resource-sharing Activities X and Y.....	229

Figure 7.26 SQS2 schedule.....	230
Figure 7.27 SQS3 schedule, developed from Figure 7.26	231
Figure 7.28 Finalized schedule, developed from Figure 7.27.....	231
Figure 7.29 SQS2 schedule with additional Y semaphore	232
Figure 7.30 SQS3 schedule with additional Y semaphore, from Figure 7.29	232
Figure 7.31 SQS4 schedule, developed from Figure 7.30.....	233
Figure 7.32 Indirectly dependent resource-sharing Activities X and Y with a slower- production-rate Activity B between them.....	234
Figure 7.33 SQS2 schedule.....	235
Figure 7.34 SQS3, SQS4, and the finalized schedule when using CIT_{XY} from SQS2 and CIT_B from SQS3, developed from Figure 7.33.....	236
Figure 7.35 Finalized schedule when using CLT_B from SQS3 and CLT_{XY} from SQS4, developed from Figure 7.34.....	237
Figure 7.36 Using dedicated resources or work break between Activities X and Y	238
Figure 7.37 Balancing production rates of Resource RES_{XY} to achieve its continuous resource utilization.....	239
Figure 8.1 ChaStrobe's process of modeling and solving problems	245
Figure 8.2 ChaStrobe's interface for Simulation Parameters	247
Figure 8.3 ChaStrobe's Interface for activities' names and precedence constraints on the Precedence Input sheet.....	250
Figure 8.4 ChaStrobe's interface for activities' productivities and work amounts on the Quantity Input Sheet	251
Figure 8.5 Resources' names, confidence levels, amounts for each type, and continuity constraints on the Resource Input Sheet	253
Figure 8.6 ChaStrobe's interface for Utilization Input.....	254
Figure 8.7 ChaStrobe's presentations, analyses, and optimization.....	256
Figure 8.8 Activating ChaStrobe	259
Figure 8.9 A single unit precedence diagram for Example 8.1	260
Figure 8.10 Simulation Parameters for Example 8.1	261
Figure 8.11 Precedence Input for Example 8.1.....	262
Figure 8.12 Quantity Input for Example 8.1.....	263

Figure 8.13 Resource Input for Example 8.1	264
Figure 8.14 Utilization Input for Example 8.1.....	265
Figure 8.15 Production diagram from the 1 st replication in SQS5 for Example 8.1.....	266
Figure 8.16 Single unit precedence diagram for Example 8.2.....	267
Figure 8.17 Simulation Parameters for Example 8.2.....	268
Figure 8.18 Precedence Input for Example 8.2.....	268
Figure 8.19 Quantity Input for Example 8.2.....	269
Figure 8.20 Utilization Input for Example 8.2.....	269
Figure 8.21 Resource Inputs with a different number of work series in ResB, ResC, and ResG for Example 8.2.....	270
Figure 8.22 Production diagram from the 1 st replication in SQS6 for Example 8.2.....	273
Figure 8.23 Single unit precedence diagram for Example 8.3 and 8.4.....	274
Figure 8.24 Single unit precedence diagram with resource nodes for Examples 8.3 and 8.4.....	274
Figure 8.25 Simulation Parameters for Examples 8.3 and 8.4.....	276
Figure 8.26 Precedence Input for Examples 8.3 and 8.4	277
Figure 8.27 Quantity Input for Examples 8.3 and 8.4	277
Figure 8.28 Resource Input for Example 8.3 only	278
Figure 8.29 Utilization Input for Examples 8.3 and 8.4	278
Figure 8.30 Production diagram from the 1 st replication in SQS6 for Example 8.3.....	280
Figure 8.31 An unusual up-and-down pattern of average project idle time in scheduling resource-sharing activities	283
Figure 8.32 Resource Input for Example 8.4, different from Example 8.3	285
Figure 8.33 CIT _{1M} and CIT _{1MN} (before work break at M5-N1) collected from the same processing SQS2	286
Figure 8.34 Additional code stipulating ResMN's working sequence from M1 to M5 and then N1 to N5, and ResXY's working sequence from X1 to X5 and Y1 to Y5	286
Figure 8.35 A typical pattern of decreasing average project idle time in scheduling repetitive projects using SQS-AL	287
Figure 8.36 Production diagram from the 1 st replication in SQS6 for Example 8.4.....	288
Figure 8.37 ChaStrobe's four output features.....	289

Figure 8.38 The average project duration and idle time from processing each SQS for Example 8.1	290
Figure 8.39 Probability density functions of project duration from processing each SQS for Example 8.1.....	290
Figure 8.40 Cumulative distribution functions of project duration from processing each SQS for Example 8.1	291
Figure 8.41 Creating and viewing Static Graphs	292
Figure 8.42 Static graph from the 1 st replication of processing SQS5 for Example 8.1.	294
Figure 8.43 Cumulative distributions of project duration derived from CPM, RSM, and SQS-AL.....	295
Figure 8.44 Probability density functions of project duration derived from CPM, RSM, and SQS-AL.....	296
Figure 8.45 Comparing project duration of RSM and SQS-AL to CPM.....	297
Figure 8.46 Comparing project duration of SQS-AL to RSM.....	300
Figure 8.47 Difference in project duration and idle time between RSM and CPM, SQS-AL and CPM, and SQS-AL and RSM	301
Figure 8.48 The finalized SQS-AL Schedule in Microsoft Project	303
Figure 9.1 Nine steps of optimization process in ChaStrobe.....	309
Figure 9.2 Modifying inputs, updating inputs, and creating simulation code and model	312
Figure 9.3 The Search Input sheet and the current decision variable cells in Row 2	313
Figure 9.4 Dynamic code input with cells in Column D reference to the decision variable Cells on the Search Input sheet, shown in Figure 9.3.....	316
Figure 9.5 Dynamic code indexes and dynamic code positions in the main code.....	317
Figure 9.6 Consistent additional code calculating objective function value, placed after the Control Statements' main code	319
Figure 9.7 Search Parameters with two main search methods, Exhaustive Search and Genetic Algorithm	321
Figure 9.8 Output from optimization using the genetic algorithm	325
Figure 9.9 The Genetic Algorithm in ChaStrobe.....	327
Figure 9.10 Precedence diagram with resource nodes for Example 9.1	332
Figure 9.11 Simulation Parameters for Example 9.1	333

Figure 9.12 Precedence Input for Example 9.1.....	333
Figure 9.13 Quantity Input for Example 9.1.....	334
Figure 9.14 Resource Input with cells referencing to decision variable cells for Example 9.1.....	335
Figure 9.15 Utilization Input for Example 9.1.....	336
Figure 9.16 Dynamic Code Input for the initial decision variables in Figure 9.17	337
Figure 9.17 Initial decision variables for Dynamic Code in Figure 9.16	337
Figure 9.18 Dynamic Code for the initial decision variables in Figure 9.19.....	338
Figure 9.19 Initial decision variables for Dynamic Code in Figure 9.18	338
Figure 9.20 Objective function and user-specified additional output for Example 9.1 ..	339
Figure 9.21 Search Parameters for Example 9.1.....	340
Figure 9.22 GA Results with three best solutions providing an objective value of 590.	341

LIST OF TABLES

Table 4.1 Precedence relationships for Example 4.1	65
Table 4.2 Precedence relationships for Example 4.2	66
Table 4.3 Stochastic durations of repetitive activities for Example 4.3	74
Table 4.4 Cumulative frequency of crew idle time of Resource B during processing SQS2 for Example 4.3.....	75
Table 4.5 Cumulative frequency of crew idle time for Resource C during processing SQS3 for Example 4.3	77
Table 4.6 Unit idle time, the selected Crew Lead Time, Average Project Duration, and Average Total Idle Time.....	78
Table 4.7 Activities' work amounts in each unit for Example 4.4	86
Table 4.8 Activities' production rates for Example 4.4.....	86
Table 4.9 Collected CITs from processing SQS2 and determining CLT for Activities B and C for Example 4.4	88
Table 4.10 Collected CITs from processing SQS3 and determining CLTs for Activities D, E, and F for Example 4.4	90
Table 4.11 Collecting CITs from processing SQS4 and determining CLT for Activity G for Example 4.4.....	92
Table 4.12 The finalized CITs from processing SQS5 for Example 4.4	93
Table 4.13 Assigned CLT and average CIT of activities from processing different sequence steps for Example 4.4.....	93
Table 4.14 The finalized UIT, average project duration, and average project idle time for Example 4.4	94
Table 5.1 Work amounts for each activity in each unit	118

Table 5.2 Daily Production Rates	119
Table 6.1 Scheduling methods, idle time, project duration, and work break	154
Table 6.2 Probability of activities on the controlling sequence.....	160
Table 6.3 Daily production rates and activity work amounts for Example 6.1	168
Table 6.4 CLT1 from the SQS-AL schedule without work breaks for Example 6.1	170
Table 6.5 CLT from the SQS-AL schedule with 2 work breaks for Example 6.1.....	172
Table 6.6 CLT from the SQS-AL schedule with 3 work breaks for Example 6.1.....	174
Table 6.7 Finalized project duration and idle time for Example 6.1	175
Table 8.1 Activities' productivities and work amounts for Example 8.1	260
Table 8.2 Activities' productivities and work amounts for Example 8.2	267
Table 8.3 Durations and variability for activities in Examples 8.3 and 8.4.....	275

LIST OF APPENDICES

Appendix A Determination of the Controlling Sequence.....	355
Appendix B Extension of Simulation Model Templates	374
Appendix C Flow Chart for the Chastrobe Application	383
Appendix D Graphical Formats for Static Graphs.....	406

ABSTRACT

The construction industry and academia have realized the critical path method and other time-based methods were not suitable for repetitive projects, which were resource-driven in nature. Both communities have been attempting to develop a better technique to schedule repetitive projects. Many approaches have been proposed; however, they are capable of solving the problems only to a certain degree of complication. Most of these approaches were limited to deterministic problems. A few probabilistic scheduling methods using simulation techniques were proposed with improvement in capturing the stochastic nature of construction activities; however, none of them guaranteed continuous resource utilization.

The Sequence Step Algorithm (SQS-AL) is a general scheduling algorithm for minimizing the duration of repetitive projects with probabilistic activity durations while achieving continuous resource utilization. SQS-AL consists of two main nested loops: the sequence step loop and the replication loop. For each sequence step, each replication loop is a simulation run that collects crew idle time for activities in that sequence step. The collected crew idle times are, then, used to determine resource arrival dates for user-specified confidence levels, i.e., probabilities of having zero idle time in corresponding activities. The process of collecting the crew idle times and determining crew arrival times for activities on a considered sequence step is repeated from the first to the last

sequence step. The effect of scheduling activities on the crew idle times for following activities is revealed step by step prior to scheduling the following activities. As a result, SQS-AL can guarantee continuous resource utilization for the user-specified confidence levels.

This thesis also presents the application of work breaks, the determination of the controlling sequence, and the scheduling of resource-sharing activities in repetitive projects with probabilistic activity durations. An application, called “ChaStrobe,” was developed on top of the Stroboscope Graphical User Interface to facilitate schedulers in creating simulation model for repetitive projects and scheduling the projects using all concepts presented in the thesis. In addition, ChaStrobe consists of two search methods, the exhaustive search and the genetic algorithm. Using the proposed concepts, the programmability in Stroboscope, and the search methods, ChaStrobe can optimize the scheduling problems of repetitive projects effectively.

CHAPTER 1

INTRODUCTION

The ultimate goal of managing construction projects is completing the project with the least amount of time and at the lowest possible cost. To achieve this objective, establishing an attainable and practical schedule in terms of time, cost, and resource utilization for the project is essential.

The Critical Path Method (CPM), a scheduling method, has been used widely in construction because 1) it offers great simplicity and 2) most scheduling software offers CPM capabilities with an inexpensive price and ease of use. However, industry and academia have realized the fallacies of the method over the past decades. While most profit-driven projects are influenced by the quality of time management, cost management, and resource management, CPM primarily focuses on time. Many heuristic methods have been integrated within CPM to supplement the missing dimensions. Yet, their capabilities are still limited by the underlying concept of CPM. Especially in managing and maximizing resource utilization, CPM performs poorly since it is a pure time-based scheduling approach, not a resource-based approach.

Though several resource-based scheduling techniques have been proposed over the years, many diverse topics of repetitive project scheduling need to be examined such

as probabilistic scheduling, the tradeoff between continuous resource utilization and project duration, and resource allocation.

1.1 Repetitive Construction Projects

Examples of repetitive projects are high-rise buildings, housing projects, tunnels, and highways. High-rise buildings are made up of floors; housing projects are made up of housing units; tunnel projects are made up of tunnel rings; highways are made up of road sections. These projects require resources performing the same or similar activities repetitively from floor to floor, from house to house, from tunnel ring to ring, or from section to section. For example, the same crew installs drywall from floor to floor in a multistory building. Floors, houses, tunnel rings, and sections are referred to as **repetitive units** in such projects.

By definition, repetitive projects are projects that consist of a series of repetitive activities requiring resources working and moving from one unit to another. These units are usually identical or similar depending on the design.

One of the main interests in scheduling repetitive projects is the ability to keep resources working continuously without idle time. Idle time is the period that a resource is being paid but not performing any work. Since resources are paid from the date they start working to the date they finish the work, idle time during employment periods is considered unproductive. Accordingly, activities should be scheduled in such a way that idle time of resources is eliminated or minimized. To do so, resource constraints must be incorporated into the schedule.

1.2 Resource Constraints

There are two types of resource constraints, resource availability constraints and resource continuity constraints. Availability constraints indicate the limited number of resources available to activities during a particular period; therefore, they control the output of those activities. Continuity constraints stipulate that resources, such as crews, need to work continuously and without interruption from the time they first arrive to the job site until they leave.

CPM generally assumes that there are unlimited resources and unconditional utilization of resources. Instead of applying resource constraints on the schedule, only precedence constraints are accounted for in scheduling resources. As a result, the derived schedule from CPM is often impractical and inefficient. Resource availability constraints and resource continuity constraints should not be omitted nor represented by precedence constraints, and, of course, they are not inferior to precedence constraints. Thus, it is necessary to impose availability and continuity constraints onto the schedule.

Figure 1.1 shows the benefits of imposing resource availability and continuity constraints on a repetitive project consisting of 3 units with 3 activities in each unit. Their dependencies are from A to B to C. Moreover, Activities A and C share the same limited resources, allowing only one of them to be performed one at a time. Figure 1.1.a results from applying only precedence constraints without resource constraints. As can be seen, this schedule is feasible because an overlap in resource usage exists between Activities A and C, which should be prevented by resource availability constraints. Moreover, it is inefficient because there is an idle time of 20 days between units in Activity B. This idle time is almost 60% of the entire employment period for Activity B.

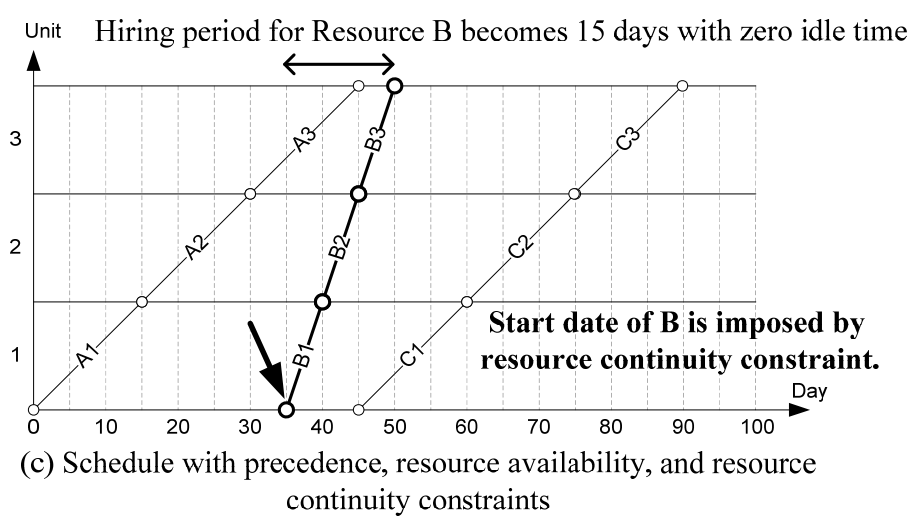
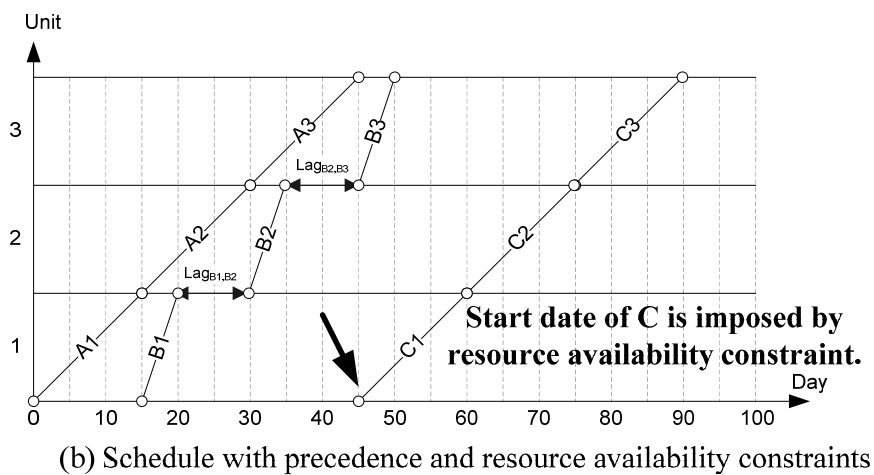
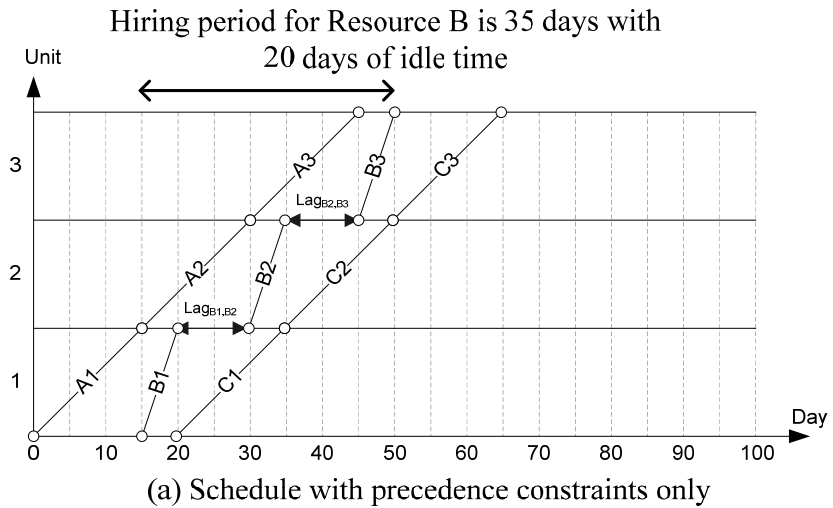


Figure 1.1 Scheduling repetitive project with different types of constraints

The schedule shown in Figure 1.1.b satisfies both precedence and resource availability constraints. The start of Activity C no longer occurs on Day 20 since A and C share the same limited resource. Resource availability constraints resolve the conflict in resource usage between Activities A and C. Now, the schedule becomes practical; however, the schedule is still inefficient because of the remaining idle time in Activity B.

To eliminate idle time, in other words, to satisfy resource continuity constraints, the sum of lags between units of a repetitive activity is calculated. Positive values of the sum of the lags indicate that the continuity constraints are violated, while zero indicates the constraints are satisfied. If the sum is greater than zero, the activity must be postponed by the sum of the lags in order to eliminate the idle time so that the continuity constraints are satisfied.

In Figure 1.1.c, the schedule satisfies precedence, resource availability and resource continuity constraints. Continuity constraints force activities to be scheduled in such a way that their resources work continuously, resulting in a shorter period of employment and zero idle time. As can be seen from Figure 1.1.c, the start date of Activity B is postponed by the sum of the lags (20 days in Figure 1.1.a). Moreover, the employment period of Resource B is decreased from 35 days in Figure 1.1.a to 15 days in Figure 1.1.c.

Figure 1.1 shows that applying resource constraints results in a practical and efficient schedule especially for resource utilization. Nevertheless, project duration increases from 65 to 90 days. If there is enough resource for both Activities A and C, project duration is still increased from 65 to 85 days due to the continuity constraints.

From the example, it is necessary to consider whether to introduce resource continuity constraints to the scheduling of a repetitive project.

The tradeoff between eliminating idle time and increasing project durations must be analyzed to ensure the derived solution is the optimal project cost or close to it. To achieve this goal, characteristics of repetitive activities must be analyzed so that alternative schedules can be established and optimized.

1.3 Characteristics of Repetitive Activities and Projects

Durations of repetitive activities in each unit are rarely identical. The differences may come from design, productivity of resources, availability of resources, scheduling techniques, etc. These factors contribute to activity and resource schedules, defining characteristics of repetitive activities. Five characteristics of repetitive activities are described below to show the need for a sophisticated scheduling technique and tool that must be able to model these characteristics, and schedule the project under precedence and resource constraints.

1.3.1 Typical and Non-Typical Activities

A “typical activity” is defined as a series of sub-activities that have the same work amount in each unit; and they have the same duration for each repetitive unit. In contrast, a “non-typical activity” is a series of sub-activities having different work amounts and, therefore, different durations in different units. Figures 1.2.a and 1.2.b demonstrate two repetitive projects whose activities are typical and non-typical, respectively.

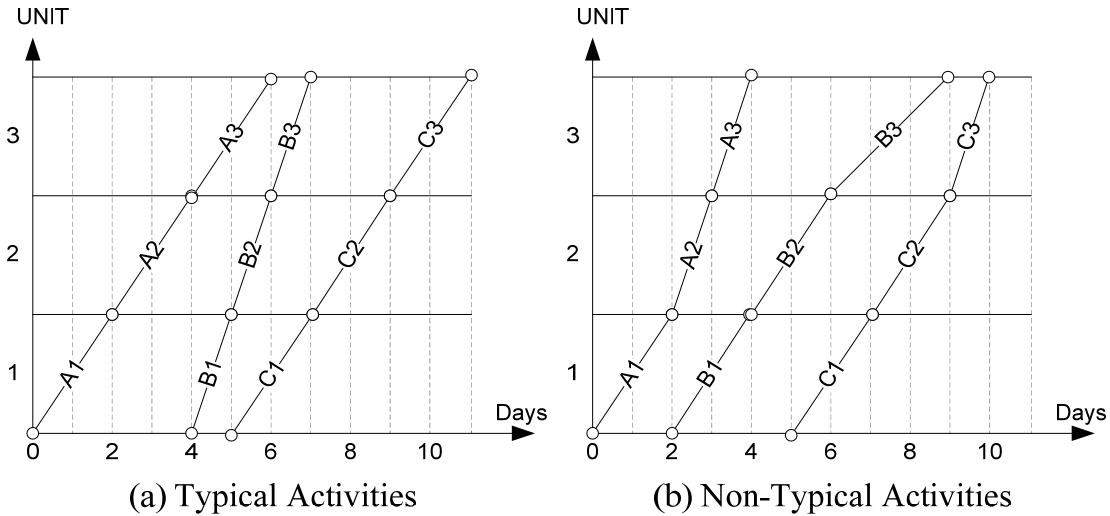


Figure 1.2 Typical and non-typical activities

Many scheduling techniques assume that the durations of sub-activities are the same (typical) so that it allows them to solve the problem easily. However, this assumption is not always practical since activity durations are influenced by many factors such as work amount in each unit and resource productivity for each activity. A developed technique should be able to model both typical and non-typical activities.

1.3.2 Repetitive and Non-Repetitive Activities

Repetitive activities are those activities that need to be performed in every unit in the project. Whether typical or non-typical, if an activity exists in every unit, it is considered a repetitive activity. On the other hand, non-repetitive activities are those activities whose sub-activities do not exist in every unit. The most common situation is when an activity exists only in the beginning of the project (before starting the first unit) and/or in the first unit. For example, excavation is considered a non-repetitive activity for high-rise buildings in which it is only required prior to the construction of the first unit (the 1st floor). Figure 1.3.a is an example of a non-repetitive activity that only exists in

the first unit. Figure 1.3.a1 is the node network for the case and Figure 1.3.a2 is its corresponding production diagram.

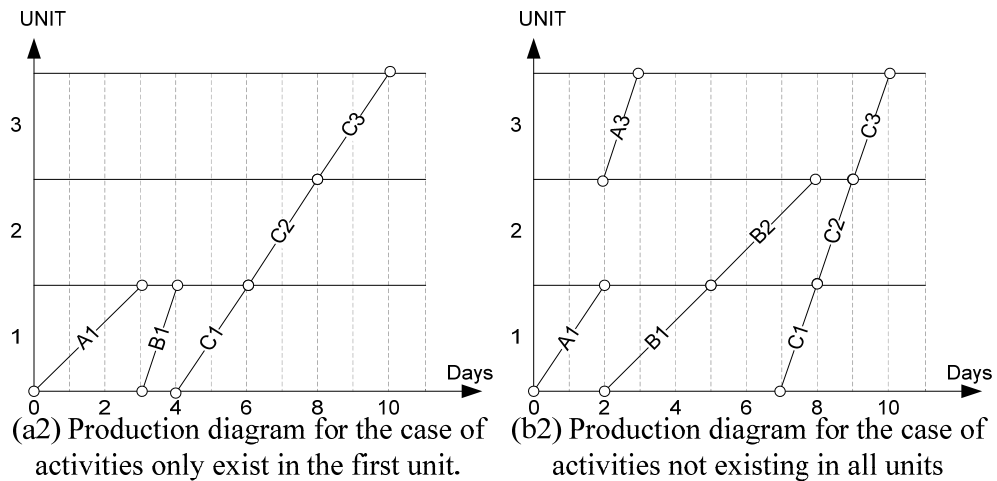
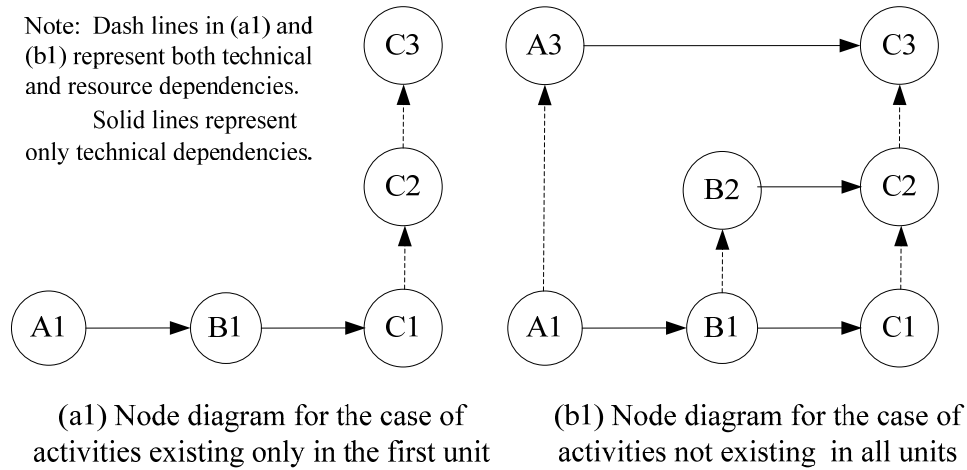


Figure 1.3 Repetitive and non-repetitive activities

A complicated situation arises when sub-activities exist in most units but are omitted in a few (Figure 1.3.b Activities B and C). Precedence and resource constraints for this non-repetitive activity are harder to model using only one unit to represent all the units and to show all dependencies between activities because of their unique dependencies between the non-repetitive activity and other activities. In Figure 1.3.b2, Activities B and C are examples of this case. Moreover, the start date of Activity D3 is

not controlled by its direct predecessor, Activity C, as in Units 1 and 2, because Activity C does not exist in Unit 3. Consequently, Activity D in Unit 3 is controlled by its indirect predecessor (B).

1.3.3 Deterministic and Non-Deterministic Durations

From a modeling perspective, probabilistic scheduling problems are some of the most complicated problems. Both industry and academia have realized the importance of capturing the stochastic nature of construction projects. The difficulty starts from estimating the probabilistic durations of activities, modeling the flows of resources and materials, simulating the dynamic nature of construction activities, and solving the problem probabilistically. Consequently, very few techniques have been proposed to solve repetitive project scheduling problems in a probabilistic manner.

Focusing on resource continuity and non-deterministic parts of repetitive project scheduling problems, prior attempts have failed to solve the problem because: 1) they are based on inappropriate concepts such as CPM and 2) computational limitations such as linear programming. For example, the Program Evaluation Review Technique (PERT) cannot maintain continuity since it schedules activities at their early start date in the same way CPM does. Techniques based on linear and dynamic programming are not effective in solving non-deterministic problems. Simulation cannot ensure continuity because they cannot control or postpone the start date of activities. The deficiencies of these methods are explained in detail in Chapter 2.

Another issue introduced by stochastic activity durations is the need to establish an indicator determining activity criticality. This requires a totally new approach because the traditional concept of a critical path as in CPM does not exist under the continuity

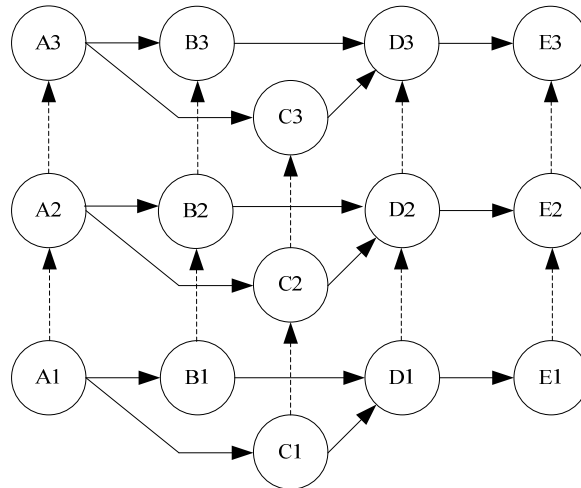
constraints (Harris and Ioannou 1998). The impact of an activity on repetitive projects with regards to project duration and continuity must be studied. The concept of the controlling sequence as defined by Harris and Ioannou (1998) is based on deterministic activity durations. Thus, a modified version of the controlling sequence must be established for the non-deterministic case.

1.3.4 Hard and Soft Logic Dependencies

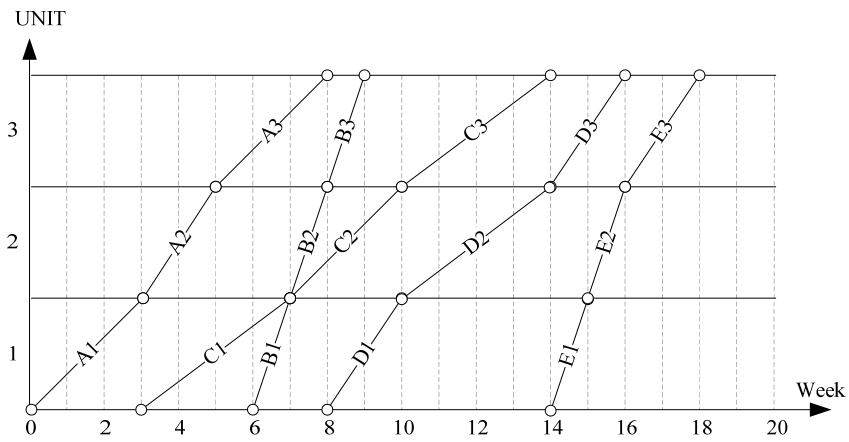
In most scheduling techniques (e.g. CPM, PERT, LOB, and LSM), precedence constraints between activities are defined as hard logic. This means, by applying hard logic to two repetitive activities (e.g., A and C in Figure 1.4), a succeeding sub-activity (e.g., C2) is constrained by 1) the same activity in the previous unit (e.g., C1) and 2) its predecessors in the same unit (e.g., A2).

If activities' interrelationships are defined as hard logic, they are considered to be technically dependent. CPM, PERT, LOB, and LSM are examples of those methods that use hard logic dependency. Unfortunately, this type of dependency in some cases is not a good representative of activity interrelationships, and may unnecessarily limit the flexibility in scheduling activities and allocating resources. For example, in Figure 1.4, a housing project consisting of 3 houses, the order of construction for these 3 houses is not constrained by technical constraints. The construction of these houses can be scheduled in many orders such as Units 1, 2, and 3 as shown in Figure 1.4.b or Units 2, 3, and 1 as shown in Figure 1.4.c. In such a case, constraining repetitive units with hard logic (forcing the order of the Housing Unit 1 to 3) would be unnecessary. Thus, the dependency between repetitive units should be defined in such a way that it allows a

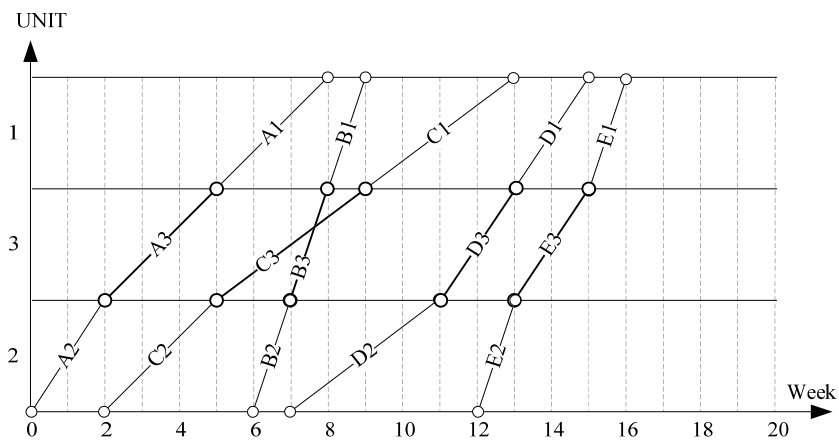
switch in sub-activity orders, when possible. This type of dependency is called “soft logic.”



(a) Node network of a repetitive project



(b) Original production diagram from RSM



(c) Using the advantage of soft logic dependencies

Figure 1.4 A repetitive project with soft logic dependencies

Soft logic is the ability of a crew to define its own orders of units for performing the repetitive work. The comparison of Figures 1.4.b and 1.4.c shows a benefit of applying soft logic constraints to the project. As shown in Figure 1.4.c, reordering the housing units from Units 1, 2, and 3 to Units 2, 3, and 1 results in a shorter project duration by 2 weeks. Accordingly, the idea of soft logic and its benefits need to be studied further.

1.3.5 Resource-Sharing Activities

In CPM and other methods such as LOB and RSM, resource availability constraints are modeled by using precedence constraints. They assume that activities require only one resource each; a resource serving only one activity is called a “dedicated resource.” In practice, however, activities may share the same resources; the resource is called “shared resource” and the activities are called “resource-sharing activities.” In the latter case, precedence constraints would not be able to additionally present the resource availability constraints for the shared resource.

Sharing resources between activities can be highly dynamic and truly resource-driven. Simple scheduling methods such as CPM and graphical methods such as LOB cannot include this activity characteristic into their calculations due to the high dynamic utilization of resources. For mathematical approaches, although it is possible to use dynamic programming to model behaviors of a resource that is shared by many activities, it is more cumbersome and ineffective than using simulation. Resource-driven simulation offers a great advantage when it comes to modeling this characteristic. Moreover, allocating shared resources may be performed promptly by using a conditioning node provided in most simulation systems (e.g., Fork node in Stroboscope). Further studies of

resource-sharing activities and allocating shared resources must be investigated regarding repetitive construction projects.

1.4 Problem Description

Repetitive project planning and scheduling problems confront every construction company. Different companies may employ different strategies to manage their projects; however, one common objective is maximizing the efficiency of resource utilization. This issue of resource efficiency can be viewed in two dimensions: scheduling perspective and operational perspective. This research focuses on the schedule perspective because it determines “how an operation is going to perform,” “when that task is going to be executed,” and “how resources (labor, machine, and money) of the operation are going to be managed.” Even before projects start, schedules could dictate the success or failure of the project; poor schedules result in project loss and delay regardless of how effective the operation may be, since skilled laborers will become idle (waste in time and money), while waiting for another operation to complete. This unproductiveness (caused by ineffective schedule, rather than the operation itself) could, in fact, be avoided. Thus, this research focuses on eliminating the avoidable idle time and unproductive costs, stemming from an ineffective schedule.

Repetitive projects tend to have an avoidable unproductiveness because operations in each unit are influenced by many activities (including sub-activities), resources (including laborers, equipment, and space), and other factors such as funding. Examples of these constraints can be grouped as: 1) technical constraints, 2) resource constraints, and 3) financial constraints. This research focuses on technical constraints and resource constraints in order to achieve a schedule that could eliminate or minimize

idle time in resource utilization. Consequently, this research employs idle time elimination to optimize and balance project time and cost.

In this research, uncertainty in work amount and resource productivity is taken into account when scheduling projects since uncertainty and construction are inseparable. However, this research will not consider the uncertainties in operational accidents, machine breakdown, supplying material delay, disastrous phenomena, financial crises, and other unforeseeable circumstances.

In conclusion, this research can be defined as the problem of probabilistic repetitive project scheduling. The main objective is to develop a tool to assist construction companies to schedule repetitive projects under uncertainty in order to eliminate the avoidable unproductive time and cost.

1.5 Existing Scheduling Techniques

Many techniques have been proposed to solve the problem of scheduling repetitive projects. However, until now, no techniques have effectively and realistically captured all the aforementioned characteristics of repetitive activities and solved the problem. Those that are considered to be advanced techniques are only capable of modeling non-typical and non-repetitive activities, but not probabilistic activity durations and soft logic dependencies. Furthermore, many of those techniques fail to control (maintaining and relaxing) continuity constraints. The limitations of existing techniques are discussed in Chapter 2, Literature Review.

1.6 Challenges

The true challenges of this research stem from the desire to create a well conceptualized and designed mechanism that offers as much flexibility as possible, besides the ability to solve repetitive project scheduling problems. Even though the author recognizes that such a desired flexibility comes with a great cost, he still believes that the challenge is achievable and worth exploring. Theoretically, the challenges are:

- 1) How to establish a generalized algorithm that can be applied to discrete-event simulation systems in order to solve a stochastic repetitive project scheduling problem. While many techniques require a customized tool to solve the problem, the generalized algorithm should be untied to a particular tool. It should be applicable to most discrete-event simulation systems such as Stroboscope, GPSS, and ProModel.
- 2) How to establish a simulation model template that can capture all aforementioned four characteristics of repetitive activities, especially the stochastic nature of repetitive construction activities.
- 3) How to provide flexibility in modifying the model template in (2) so that this proposed technique (the algorithm and the model) can be extended to solve a more complicated problem. The proposed technique must be designed for current and future development.

1.7 Research Objectives

The following are the primary objectives of this research:

- 1) To establish a generalized algorithm that empowers discrete-event simulation to solve the problem of scheduling and planning repetitive construction projects.
- 2) To design a universal simulation model template that can capture the characteristics of repetitive activities and their resources.
- 3) To develop a heuristic algorithm for ordering repetitive units and/or sub-activities in order to benefit from soft logic dependent activities.
- 4) To develop a search algorithm that seeks an optimal (or near optimal) solution according to (1), (2), and (3).
- 5) To develop an application that automates and facilitates the scheduling of repetitive projects according to the proposed (1), (2), (3) and (4).

1.8 Conclusion

This chapter discusses several aspects of repetitive projects. Repetitive projects are commonly found in construction where similar units require repetitive activities from unit to unit. Repetitive activities and their resources are the essence of repetitive project scheduling because their constraints impose directly on project schedules. The constraints in repetitive projects are:

- 1) Precedence constraints
- 2) Resource availability constraints
- 3) Resource continuity constraints

While precedence constraints specify the technological orders of work, resource availability and continuity constraints control the utilization of resources in the project. To obtain a practical and efficient schedule, the above three constraints must be accounted for in project scheduling. They and their application are discussed in this chapter focusing on the resource continuity constraints.

One of the main focuses in scheduling repetitive projects is to maximize resource utilization by keeping resources working continuously without interruption. Resource continuity constraints normally enforce delay in activities to improve continuous resource utilization. Nevertheless, delaying repetitive activities could adversely prolong project duration. Thus, the tradeoff between increasing project duration and decreasing idle time must be considered in order to obtain a practical and efficient schedule. The tradeoff can be analyzed by performing sensitivity analysis between enforcing and relaxing resource continuity constraints. To obtain the most efficient schedule, it is suggested that the analysis of the tradeoff must be incorporated with the fundamental characteristics of repetitive activities.

Four characteristics of repetitive activities are discussed in this chapter to illustrate their potential benefit in optimizing repetitive projects. These characteristics are:

- 1) Typical and non-typical repetitive activities
- 2) Repetitive and non-repetitive activities
- 3) Deterministic and non-deterministic activity durations
- 4) Sharing or non-sharing resources with other activities.

To achieve an optimal schedule, schedulers must truly understand the four characteristics of repetitive activities and carefully analyze the tradeoff between imposing and relaxing resource continuity constraints. This research will coordinate all of these factors in the proposed algorithm to effectively solve the problems of repetitive project scheduling.

CHAPTER 2

LITERATURE REVIEW

This chapter presents advantages and disadvantages of techniques used in scheduling repetitive projects. These techniques are:

- 1) Mathematic Approach. Examples of this approach are:
 - a. Critical Path Method (CPM)
 - b. Program Evaluation and Review Technique (PERT)
 - c. Vertical Production Method (VPM)
- 2) Graphical Approach. Examples of this approach are:
 - a. Line-of-Balance (LOB)
 - b. Linear Scheduling Method (LSM)
 - c. Repetitive Scheduling Method (RSM)
- 3) Linear Programming (LP)
- 4) Dynamic Programming (DP)
- 5) Simulation

These techniques are compared in various aspects of scheduling specifically for repetitive projects based on many researchers' recommendations. At the end, these techniques are summarized.

2.1 Introduction

Many scheduling techniques have been developed for planning and scheduling. Some are for general construction projects, while others are specifically for repetitive projects. The Critical Path Method (CPM) and Program Evaluation and Review Technique (PERT) are examples of techniques suitable for general projects, but not repetitive construction projects. CPM and PERT cannot produce an effective schedule for repetitive projects. On the other hand, many scheduling techniques are designed specifically to schedule repetitive projects. These techniques range from a basic graphical approach, such as Line-of-Balance (LOB), to a more complex approach, such as linear and dynamic programming. Integrations between techniques are commonly established to mutually improve the original concepts such as the integration between LOB and CPM.

The chronological development of repetitive project scheduling techniques can be viewed in these orders:

- 1) Graphical Approach
- 2) Mathematic Approach
- 3) An integration between (1) and (2)
- 4) Advanced mathematical approach (e.g., Linear Programming and Dynamic Programming Simulation)
- 5) An integration between (1) and (4)
- 6) An integration between (1), (4), and (5)

These developments are influenced by the incremental understanding toward the problems of repetitive project scheduling, the need from the industry, and technological changes in computing capability.

2.2 Existing Techniques

The following discussions concentrate on the advantages and disadvantages of the existing techniques and their underlying concepts. Various aspects of repetitive project scheduling and their ultimate objectives are assessed to verify their practical usages and their realistic representatives of construction projects. Comments and suggestions from the original authors and others are presented to emphasize the subjects that need to be improved or included to achieve an effective technique.

2.2.1 Critical Path Method (CPM)

The Critical Path Method (CPM) was developed in the 1950s by James Kelly and Morgan Walker (Senior 1993). The method offers an easy calculation to derive a project schedule and to assess the criticality of activities using its proposed concepts of floats and the critical path, focusing on time. Activities and their interrelationships are depicted in a network by nodes and arrows. Nodes represent the activities and activity information such as title, duration, etc. Arrows represent the interrelationships (precedence constraints) between activities and the lead time between them. After the network is constructed and the activity durations are given, the calculation of critical path, critical activities, and floats can be performed straightforwardly. The derived information informs project managers of the criticality of activities, which allows them to plan in advance how to schedule the activities and manage the project effectively, based on the current schedule. On the other hand, the managers may decide to alter the original schedule to suit the project deadline, the company resources, and so forth.

Most scheduling software (Microsoft Project, Primavera, etc.) offers automation of CPM calculation and network drawing, within seconds after inputting data. These programs facilitate schedulers in altering and updating the schedule purposely for planning and controlling. Accordingly, CPM has been widely used in the construction industry.

However, CPM has been criticized for its incapability of taking resource consideration into account in its calculations. This usually leads to an unfeasible schedule due to the unawareness of resource constraints such as resource availability constraints (Selinger 1980; Kavanagh 1985).

From the perspective of repetitive project scheduling, CPM is incapable of capturing the realistic and stochastic nature of repetitive projects (Selinger 1980; Stradal and Cacha 1982; Chrzanowski and Johnston 1986, Reda 1990; Rahbar and Rowings 1992; Suhail and Neale 1994, Harris and Ioannou 1998. The reasons for such incapability are:

- 1) CPM does not take resources into account in calculating schedules. It is designed primarily for scheduling and monitoring activity and project duration; CPM is a pure time-based schedule technique. CPM cannot ensure the continuous resource utilization of a crew from unit to unit (Senior 1993; Onur 2003). Therefore, it cannot maximize efficiency in resource utilization (Birrell 1980; Stradel and Cacha 1982; Rowings and Rahbar 1992; and Senior 1993).
- 2) CPM calculation does not include nor is it concerned with the imbalanced production rate of resources resulting in inefficient resource utilization.

- 3) CPM is not applicable to non-deterministic activity duration. It is important to recognize that CPM, as a deterministic scheduling method, would schedule projects only to the level of reliability of the input values of the duration of activities. For reliable representation, the productivity data must be expressed in some probabilistic measure (Dhanasekar 2000).
- 4) CPM cannot eliminate idle time, since it schedules activities based on their earliest start dates, (Reda 1990, Huang and Halpin 2000). If a predecessor has a lower production rate than its successor, the successor must wait until the predecessor completes, which results in idle time.
- 5) CPM and its graphical presentation are considered ineffective when applied to repetitive projects having a large number of units. Its calculation becomes tedious and labor intensive (Yang 2002). For example, a repetitive project consisting of 7 activities for 1000 units will require 7000 nodes to represent the network. A network of this size is confusing and unmanageable (Carr and Mayer 1974; Chrzanowski and Johnston 1986; Reda 1990; Yang 2002).

To alleviate the mentioned deficiencies of CPM, the integrations of CPM and other techniques such as Line-Of-Balance (LOB) have been developed during the last couple of decades. Nevertheless, they still cannot handle the stochastic nature in the repetitive projects.

2.2.2 Project Evaluation Review Technique (PERT)

The Project Evaluation and Review Technique (PERT) was introduced to the construction industry in the 1950s. It is a probabilistic scheduling technique using three point estimates of activity durations to determine an estimated project duration. The difference between CPM and PERT is that PERT is capable of scheduling non-deterministic activity durations while CPM cannot.

However, PERT has not been widely used in the construction industry compared to CPM as it requires more data of activity durations, which is often difficult to obtain and justify. Moreover, PERT requires intensive computation compared to CPM. Since the technology of personal computers has been improved in the last couple of decades, the improved technology causes simulation method to supersede PERT (Senior 1993). From a repetitive project perspective, PERT and CPM have the same limitations due to their underlying time-based scheduling calculation and their graphical presentation in precedence networks (Senior 1993; Yang 2002).

2.2.3 Line-Of-Balance (LOB)

The Line-Of-Balance method (LOB) was developed at the Goodyear Company by George E. Fouch in the early 1940's for the purpose of managing and controlling production processes in industrial manufacturing where tasks are repetitive. Then, LOB was applied in the Navy (Miller 1963). LOB's main objective is to balance the size of labors and machines based on their production rates so that their resources are employed at full capacities.

The major benefit of LOB to construction scheduling is that it conveys important production rate and duration information in a graphical format (Onur 2003). LOB shows progress of activities against time in graphical presentation. The accumulated work completed is plotted with work progress on the Y-axis and time on the X-axis. The line representing completed work is termed the “Production Line”. It is evident that LOB offers a better visual presentation than the precedence network, especially for repetitive projects, because the comparisons between activities and between units can be easily perceived in the diagram. The easily interpreted graphics format enhances the viewers understanding of the project and also individual activities. It allows the viewers to detect a potential bottlenecks (Lutz 1990; Onur 2003) by simply observing the production lines.

LOB allows schedulers to observe and adjust the production rate of activities in a production diagram to maximize resource utilization. This process of adjusting production rate is known as “balancing production rates.” LOB provides a means of selecting crew size in order to minimize inefficiency and waste in resource utilization (Lutz 1990). To balance unit production rates, activities are assigned to work at the minimum unit production rate among activities. For example, Activities A, B, and C have unit production rates of 2, 1, and 3 units/day. Thus, to balance these activities, production rates of A, B, and C should be set at a rate of 1 unit/day. If the scheduler desires to expedite the project furthermore, more resources could be assigned to Activities A and B so that they progress as fast as Activity C. For this example, additional resources must be assigned to Activities A and B to speed up their unit production rates to 3 units/day.

The application and graphical presentation (production diagram) of LOB facilitate schedulers in constructing a schedule that satisfies precedence constraints, resource

availability constraints, and resource continuity constraints. The concepts of continuous resource utilization and balancing production rates benefit repetitive projects in several ways. First, the former maximizes the efficiency of resource utilization by eliminating idle time. Since resources are scheduled to work continuously, the project will benefit from the learning phenomenon especially in labor-intensive activities.

Secondly, balancing production rates keeps all activities working at the same pace and possibly reduces project duration. According to the advantages of LOB, many researchers have adopted and adapted these concepts in order to optimize project duration and project cost. The tradeoff between project duration and cost can be analyzed by various techniques, such as integer programming, linear programming, and dynamic programming.

However, the fundamental principal of LOB has several drawbacks, which need to be attuned and improved in order to suit the nature of construction projects. The limitations of LOB and solutions are described below. One of the unrealistic assumptions of LOB is the assumption of activities' constant production rates of activities are constant (Carr and Meyer 1974; Johnston 1981; O'Brien 1985; Arditi and Albulak 1986; Lutz 1990; Russell and Wong 1993; Yang 2002). This assumption implies that 1) work amounts in each unit are identical and 2) productivity of resources in each activity is a constant.

For repetitive construction projects, the amount of works in each unit could be different from unit to unit; it is rarely the same. For example, high-rise buildings consisting of several floors usually have various types of interior finishing on different

floors. Another example is highway projects. Excavation at different locations is likely to result in diverse amounts of soil, according to an existing ground profile.

Senior (1993) stated that the inability to incorporate varying amount of works in repetitive sub-activities is another limitation of LOB. Repetitive projects usually consist of non-typical activities and non-repetitive activities. Non-typical activities are activities having different work amounts in each unit. Non-repetitive activities are activities existing only in a few units. Details of non-typical and non-repetitive activities are discussed in Chapter 1.

In addition, Halpin and Woodhead (1976) stated that construction operations are stochastic by nature; thus, assuming production rates are constant may be erroneous. This unrealistic assumption of constant production rates limits the application of LOB to certain degree of realism and complexity (Neale and Neale 1989; Rowings and Rahbar 1992; Yang 2002). To alleviate the deficiency in LOB, the calculation of LOB must be modified in order for it to be applicable to repetitive projects with non-typical repetitive activities (difference in work amounts) and non-repetitive activities (work not existing in all units).

From a resource utilization perspective, Yang (2002) pointed out the assumption of constant production rates limits LOB in two ways. First, each activity must be performed by the same crew. At a project level, this may not be a serious issue, but at a company level this assumption eliminates the possibility of allocating resources among projects. Second, the hard-logic precedence constraints used in LOB may unnecessarily restrain repetitive activities to work in the same sequence from one unit to another. For example, an interior finishing activity of Unit 1 must perform before the activity of Unit

2. This hard-logic precedence constraint eliminates the flexibility in managing the project at an activity level where sequences of units could be altered to achieve a better balanced production rate. Carr and Mayer (1974) suggested that the working sequence of units within activities, not constrained by resource or technological constraints, should be “a matter of choice rather than dependency”. In other words, the dependency between activities should be modeled by “soft logic” constraints when possible.

Ironically, the benefit of maintaining continuous resource utilization in LOB has also been argued to be a disadvantage of LOB because of the inability to relax the continuity constraints. In a time-cost optimization problem, the tradeoff between continuity and interruption in work must be analyzed and balanced to achieve the minimum project cost. Without the ability to relax the resource continuity constraints, the project duration may be excessively delayed; as a result, the increased indirect cost (from the delay) exceeds the savings in the direct cost (from eliminated idle time).

Since LOB schedules activities in such a way that activities must work continuously, the benefit from allowing the interruption cannot be obtained. It is a fact that idle time in resource utilization is considered waste at the activity level. However, with no consideration of project cost, this may always be valid at the project level (Reda 1990; Lutz 1990). Focusing on continuous resource utilization and project duration, a schedule derived from LOB may increase project cost, although project duration is reduced. This is a time-cost tradeoff problem. While maintaining 100 percent work continuity results in much greater project duration and consequently indirect cost, allowing interruption (relaxing continuity constraints) could reduce the project duration and indirect cost significantly with a negligible penalty costs from allowing idle time.

From a company stand point, dynamically allocating resources among projects may incur idle time or waste at a project level; however, this would provide better resource utilization and cost efficiency at the company level (Lutz 1990). Thus, allowing interruption in resource utilization should also be considered as an option (Yang 2002).

Another shortcoming of LOB is that its graphical presentation becomes confusing when many concurrent activities take place in a particular period. It is designed to model simple repetitive works; thus, LOB's graphical presentation is not readily fit to the complexity of construction projects (Kavanagh 1985; Neale and Raju 1988). Arditi and Albulak (1996) suggested using colored graphics may lessen the problem.

While many suggested the modification of graphics, on the other hand, Sarraj (1990) argues that the diagram is not necessary for the scheduling purpose. He computerizes the LOB algorithm using a mathematic approach that provides all scheduling-related information. Since all information is derived from his program, he claims there is no need to draw any diagram to derive schedules.

An additional concern about LOB in construction was that it has not been used widely because it is not as readily computerized as network methods (Chrzanowski and Johnson 1986; Lutz 1990; Yang 2003). However, this has changed because of the great improvement in computation over the last decades. Examples of computerized LOB combining graphical and analytical methods to solve repetitive project scheduling problems are works from Yang (2002).

2.2.4 Other Graphical Approaches

Similar to LOB, many graphical approaches provide a simple means to schedule repetitive projects. Most of them have similar advantages and disadvantages as does

LOB. Relying heavily on scheduler's judgments and efforts, their uses are limited to a small and simple problem. The differences among existing graphical methods stem from their objectives to solve a specific type of construction project, such as highway, high-rise, and housing projects.

Improved from the original LOB, many graphical methods modified LOB so that they are able to model non-repetitive and non-typical activities in repetitive projects. However, the capability of these LOB-modified graphical methods is still limited to a simplified scheduling problem. Beyond their capability, these methods can not perform a sensitivity analysis nor solve a probabilistic scheduling problem. Consequently, the need to improve and computerize graphical methods has been brought into awareness.

The following sections discuss many existing graphical methods and focus on the concepts. They are introduced in chronological order along with recommendations made by the original authors and others. Rather criticizing the limitations of their methods, it should be understood that these graphical methods were established in the early development of repetitive project scheduling for construction. Their efforts and recommendations are valuable and indispensable for developing a new approach solving repetitive project scheduling problems.

Peer (1974) gave a general idea of construction planning. He suggested activities should be categorized into 4 groups: 1) preparatory (such as approval and planning procedures), 2) main repetitive, 3) interlinked (equivalent to non-critical and/or non-controlling), and 4) external (not considered as a repetitive activity such as fences and site development). These four types of activities give schedulers a mindset of activity

criticality and the span of works (repetitive or non-repetitive). The idea of prioritizing activities in repetitive projects still needs to be studied further.

Peer (1974) suggested that it is not a problem of finding a critical path, but rather specifying a critical path, in other words, determining what should be made critical. Thus, the four categorized sets of activities and their associated costs must be taken into consideration when defining critical activities and balancing production lines. Peer recommended that construction planning input should be based on quantities of work, production rates, and other production characteristics. Moreover, activity durations should be non-deterministic in order to reflect realism of construction activities.

O'Brien (1975) introduced the Vertical Production Method (VPM), a graphical approach similar to LSM. Its main application is scheduling high-rise buildings by considering each floor as a repetitive unit. He addressed the necessity of integrating between non-repetitive and repetitive works for high-rise buildings. VPM employs a node network to represent non-repetitive activities such as excavation existing only in the first floor of a building. For repetitive activities, cumulative production is plotted on the Y-axis while time is plotted on the X-axis, similar to most graphical approaches. Accordingly, two separate schedule formats are used to present a repetitive project. According to Senior (1993), a drawback of VPM is that it requires planners to manually combine CPM to LOB. Therefore, VPM is not suitable for big scale or complex projects.

The Linear Scheduling Method (LSM) was first introduced by Johnston (1981). Its graphical presentation, linear scheduling diagram, is very similar to line-of-balance diagram. The main difference between an LOB diagram and an LSM diagram is that an LOB diagram usually presents a discrete cumulative progress (e.g., floors) while LSM

diagram presents continuous cumulative progress (e.g., miles). Moreover, activities in an LOB diagram are presented by horizontal lines from the start to the finish of the activities while activities in an LSM diagram are presented by production lines with the slope of activity unit production rates. However, since many researchers have adopted the concept of LOB but using LSM diagrams (according to Johnston's work) to present activities, LOB diagrams and LSM diagrams are referred and used interchangeably in construction.

Besides the graphical presentations, Johnston noted a difference between LSM and LOB is in their emphasis. While the application of LOB focuses on balancing production lines, LSM concentrates on planning the activities (Johnston 1981). In his paper, Johnston applied LSM to a highway project. He showed that time-cost optimization can be accomplished by a simple calculation incorporating with the LSM diagram. Comparing to a node network and a bar chart, a LSM diagram provides richer information (such as production rate and production line) allowing schedulers to visualize the process of optimization.

Moreover, Johnston (1981) suggested that seasonal adjustments should be included into LSM diagrams, because holidays and bad weather adversely affect activity productivity. During the course of constructing an LSM diagram, either reduction in production rates or work interruptions must be introduced to the affected activities in order to reflect a realistic schedule.

After the introduction of LOB and LSM, variations between LOB and LSM were developed to improve the means of scheduling repetitive projects. The following works are considered as a new generation of graphical methods. Most of these works are computerized and designed to solve a sophisticated repetitive project scheduling problem.

Thabet (1992) proposed a method called “Horizontal and Vertical Logic Scheduling method” (HVLS), which combines graphical, knowledge-based, and analytical methods in order to schedule multistory buildings. The HVLS application is considered the very first LOB- computerized extension that provides a full benefit of construction information system. It is user-friendly and database-driven and utilizes information extracted from a 3D CAD model.

In Thabet’s thesis, he included space constraints into scheduling in addition to precedence, resource availability, and resource continuity constraints. Thabet suggested that 1) activities in multistory buildings are performed within a limited space and 2) material requires storage area. Ignoring the requirements of work and storage areas may incur a conflict among different trades, decrease productivity, impact safety, and lengthen project duration (Thabet 1992; Thabet and Beliveau 1994).

Effectively incorporating the space and continuity constraints into technical constraints, Thabet grouped technical constraints into 2 categories: horizontal and vertical constraints. The vertical and continuity constraints impose on activities in different units, while the horizontal and space constraints impose on activities in the same unit. During the course of scheduling, these constraints must be satisfied.

The drawbacks of Thabet’s work are 1) not considering non-typical activities and 2) not considering non-repetitive activities (Yang 2002). His solution can solve only a problem with constant production rates and the same work quantity in all units. In reality, production rates are varied and work quantity among units is unlikely exact the same. According to the drawbacks, his work is limited to simplified problems of repetitive project scheduling.

Harris and Ioannou (1998) introduced the concept of the repetitive scheduling method (RSM) and the concept of controlling sequence. Even though Harris and Ioannou did not computerize RSM, their valuable and pioneering thoughts have influenced many others such as Yang (2002) and also this research. RSM is a graphical method that combines graphical and analytical approaches to schedule repetitive projects. It can be applied to a repetitive project consisting of typical, non-typical, repetitive, and non-repetitive activities. Distinctions between controlling activity and critical activity are made based on the introduced concept of controlling sequence and the original concept of critical activity in CPM. Harris and Ioannou pointed out 2 important facts. First, a critical activity may or may not be a part of controlling sequence. Second, a critical path do not exist anymore (Yang 2002) because activities are delayed (resulting in increased floats) to maintain the continuity in resource utilization. Consequently, different critical levels of for activities are established, including controlling-critical, controlling-not-critical, critical-not-controlling, and not-controlling-not-critical activities.

Yang (2002) developed a sophisticated application, named “Repetitive Project Planner,” employing graphical and analytical approaches. Under Ioannou’s supervision, Yang (2002) integrated the concept of RSM to activity graphics formats (lines, bars, and blocks) to model the following realistic characteristics of repetitive project scheduling.

- Variable production rates and variable work quantities.
- Composition of crew size, tools, and equipment.
- Changing in work direction (east-to-west, top-to-bottom, etc.).
- Different precedence relationships (finish-to-start, start-to-start, etc.).
- Specified buffer in time (lead-time) and space (lead-space).

- Allowed interruption if desired.

Yang (2002) explicitly pointed out the similarities and differences in scheduling discrete and continuous repetitive projects. Moreover, criticality of an activity is given based on controlling sequence and the set of critical activities according to RSM.

2.2.5 Linear Programming (LP)

Linear Programming (LP) can be described as a mathematical procedure for minimizing or maximizing a linear function of multiple variables with a defined set of constraints for these variables. LP problems consist of two components: 1) an objective function to be maximized or minimized and 2) constraints defining linear relationships between variables. Given linear relationships between variables, a time-cost tradeoff problem or resource allocation problem can be solved effectively by using LP.

However, the limitation of LP is the assumption of linear relationships among variables (Lutz 1990). Existing linear programming models often simplify the complexity of repetitive projects. According to El-Rayes (1997), most methods employing linear programming assume that all activities are typical activities. For example, these methods often assume that durations of interior activities from the first to the last floor are the same. This assumption is not practical. Since duration is influenced by crew productivity, different work amounts, and so on, the durations of sub-activities (for the same repetitive activity) in different units are rarely the same.

Based on the aforementioned, it can be concluded that linear programming does not suit complex situations common in construction. Senior-Brown (1993) agrees with the conclusion that construction situations are too complex for straightforward mathematical formulation.

The following section presents the applications of LP for repetitive project scheduling proposed by many researchers from different perspectives. Studying the pros and cons of these applications and their objective functions provides insightful information and the possibility of employing LP to search for an optimum solution.

Perera (1983) used LP to determine the maximum rate of construction. The model consists of three main constraints: limited resource constraints, material constraints, and financial constraints. After all constraints are established, the rate of completion is maximized to derive the optimum number of crews for all activities. Since the objective function is maximizing the rate of completion, it does not provide the optimum project cost which in fact should be considered as the main objective (El-Rayes 1997). Moreover, Perera's model cannot model non-typical activities because it assumes identical duration among units of an activity (El-Rayes 1997).

Most importantly, from a resource utilization perspective, Perera's model does not consider the continuity in resource usage (Yang 2002) nor allow work interruption (El-Rayes 1997).

Reda (1990) used LP to optimize the time-cost tradeoff problem for repetitive projects. The proposed model was named the "Repetitive Project Model" (RPM). Compared to other linear programming models, the advantage of Reda's model is that it includes cost as a decision variable in the optimization process (Moselhi and El-Rayes 1993).

The objective of Reda's RPM is minimizing project direct cost for feasible project duration while satisfying the following constraints:

- 1) Maintain a constant production rate for each crew

- 2) Maintain a continuity of work for each crew
- 3) Allow for a time buffer between activities on the same stage
- 4) Allow for a stage buffer between concurrent activities
- 5) Specify feasible project duration

Reda's model guarantees continuity in work flow by using a time buffer between activities, postponing the start date of the succeeding activity. Based on continuity constraints and buffer constraints, work interruption is not allowed; the benefit of allowing interruption is neglect. The lack of considering deliberate interruptions is one of Reda's model drawbacks. The inflexibility in continuous resource utilization and the unrealistic assumption of a constant production rate are the major disadvantages of RPM (Yang 2002).

In addition, the process of formulation in Reda's model could be cumbersome as the number of units increases because the number of constraints depends on the number of repetitive activities. Thus, it is limited by the complexity of establishing the constraints (El-Rayes 1997).

2.2.6 Dynamic Programming (DP)

Dynamic Programming (DP) is a mathematical technique for solving sequential decisions where the next stage is determined by the current stage. DP does not have a specific formulation. The technique aims to find an optimal substructure by dividing the problem at-hand into sub-problems to find an optimal solution in each sub-problem, and thus leading to the optimal solution for the overall problem. Recursive computations are used to inter-relate these stages to resolve the last stage as the optimal solution for the problem.

Two advantages of dynamic programming are: 1) it is not constrained by linear assumptions and 2) it can be easily computerized. DP has been developed to overcome the limitation of linear programming so that the calculation is not constrained by the linear assumption. Thus, DP offers a more realistic model and results (Lutz 1990). Since DP can be easily computerized, the optimizing process and sensitivity analysis are less labor-intensive than presented with graphical approaches.

The following paragraphs discuss the proposed models by various researchers employing dynamic programming (DP). Attention is given to their perspectives of using computerized algorithms and establishing formulas in order to solve or optimize repetitive project scheduling problems. Limitations of their models are also presented.

In 1980, Selinger proposed the “Construction Planning Technique” (CPT) using dynamic programming to minimize project durations. His model is able to maintain continuity in resource usage, and to optimize project durations. Moreover, Selinger’s model can model non-typical activities and different types of activity relationships, such as start-to-start and finish-to-start.

The main disadvantage of Selinger’s model is it only solves problems with serial activities, where activities have only one predecessor and one successor (Yang 2002). This disadvantage limits the usability of his model. Without simplifying construction operations, his model cannot solve repetitive construction projects such as housing projects and high-rise buildings in which activities perform works concurrently. In addition, his model does not consider minimizing project cost.

Yang (2002) pointed out that Selinger’s model can be improved in many ways. Firstly, Selinger (1980) did not consider the potential benefit of allowing interruption.

Secondly, as Yang argues, dynamic programming is not a suitable tool solve and model stochastic problems. Thirdly, the model was designed only for dedicated resources. Fourthly, the learning effect should be formulated and applied to activities whose work continuity is maintained. It is important to understand that these four suggestions could be achieved by modifying Selinger's model; however, the modification would be very complicated and cumbersome, which is the shortcoming of using mathematical approaches when solving a scheduling problem at this level of complication.

Without respect to project cost, Selinger's model was designed to optimize only project duration. Since minimizing project duration may increase project cost, the main objective should be minimizing project cost rather than project duration.

Russell and Caselton (1988) proposed a two-stage variable dynamic programming formulation to schedule repetitive projects. Their model included a set of interrupted durations in the second stage variables, where the first state variables consider possible durations. Consequently, their model offers a more flexible schedule and shorter project duration than Selinger's model does (Yang 2002).

However, it cannot be concluded that Russell and Caselton's solution is better than Selinger's. Since both models are time minimization approaches, neither of their solutions guarantee the minimum project cost, possibly leading to a higher project cost (Moselhi and El-Rayes 1993). Forcing resources to work continuously (not allowing any interruption) without an incorporation of cost (direct and indirect cost) could result in an inefficient outcome. Actually, Russell and Caselton's model does not even guarantee the minimum project duration. In their paper, the solution was mentioned as a "near-optimal solution" (Russell and Caselton 1980). Suggestions for their model are similar to

Selinger's, except for the addition of allowing interruptions. The main disadvantage is still the assumption of serial activities (Russell and Caselton 1988).

Moselhi and El-Rayes (1993) proposed a dynamic programming formulation incorporating cost in optimization processes. Their model minimizes overall project cost by choosing crew formations that provides an optimal project cost. The process of identifying the best crew formation is accomplished by enumerating all the possible combinations between crew formation of predecessor and successor activities, one pair at a time. After accumulated costs are calculated through the completion of the successor activities, crew formation of the predecessor activity that provides the least overall cost is considered the optimum crew formation for the activity. Then, the process moves to the next pair of predecessors and successors. This process is called "finding local optimum predecessor" in their paper. It starts from the first to the last activity, called the "forward stage." After finishing the last activity, the process traces backwards through the previously identified optimum formations to determine the best formation of all the crews. This process is called the "backward stage." For further detail and examples see Moselhi and El-Rayes (1993).

Moselhi and El-Rayes's model is capable of scheduling repetitive projects consisting of both typical and non-typical repetitive activities. Different interrelationships between activities can be defined, as shown in their paper. Overtime cost and allowance of work interruptions (relaxing continuity constraints) are also included in their models.

However, Moselhi and El-Rayes's dynamic programming formulation is limited only to projects with serial activities, and does not consider sharing resources between activities. Only deterministic problems can be solved by their models.

The difficulties in using mathematical optimization techniques are concluded below:

- 1) It is impractical to request schedulers to specify all the possible sets of decision variables (e.g., crew sizes and work interruptions) before project start date (Yang 2002).
- 2) Site managers usually do not have the training to compose hundreds of mathematical constraints, and similarly do not have the training to analyze the numerical outcomes (Yang 2002)
- 3) Mathematic approaches do not provide a generalized form of solution; they require customized input for each new problem (Lutz 1990).
- 4) It is ineffective and, sometimes, infeasible to use LP and DP solving repetitive project scheduling problems with probabilistic activity durations, which are a better representative of the construction activities due to their stochastic nature.

2.2.7 Simulation

Simulation is an analytical technique involving designing and experimenting with an established mathematical-logical model. When relationships between variables are not linear and/or random variables are included in the problem, simulation is a preferable tool to model the relationships and solve the problem. Simulation enables schedulers to study the behaviors of a process without the necessity of formulating a mathematical function of an unpredictable behavior input (Hijazi 1989). In other words, if the problem is too complex for mathematical formulation, it is best analyzed by simulation (Senior 1993). For an example, weather conditions and planning decisions can be modeled in simulation

and evaluated via the outcomes of the simulation. Although the outcomes from simulation are not guaranteed as optimum, since simulation is not an optimization tool, simulation still can be used to derive an optimal or near optimal solution by altering the input and searching for a satisfactory result (Lutz 1990).

The non-deterministic nature of construction projects is one of the most complicated factors that are often simplified or neglected by modelers. Many deterministic approaches such as linear and dynamic programming fail to provide a result with confidence because of the simplification (Yang 2002). On the other hand, simulation is considered an excellent tool for the stochastic problems because the effect of uncertainties in construction projects can be modeled and assessed by using simulation (Halpin and Woodhead 1976; Lutz 1990; Martinez 1996; Yang 2002).

Nevertheless, simulation by itself cannot control or eliminate idle times (Yang 2002; Ioannou and Likhitrungsilp 2005). Therefore, an external algorithm must be developed and implemented in the simulation model in order to effectively solve the problem of repetitive project scheduling (Lutz 1990; Yang 2002).

The following studies have attempted to use simulation to schedule repetitive projects. Their advantages and disadvantages are discussed, focusing on improving the means of scheduling repetitive projects using simulation.

Ashley (1980) used simulation to study the planning of housing projects. His example was carried out by GPSS simulation language. The study optimized the minimum project duration by altering inputs in the simulation. Various simulation runs were executed and then the results were compared. The inputs, decision variables, were

1) the number of machines, 2) the size of crews, 3) the allocation of crew and equipment (Ashley 1980).

Nevertheless, Ashley's model did not consider eliminating idle time. There is no external algorithm implemented in the simulation to achieve continuous resource utilization. Accordingly, Ashley's model schedules the project in the same way as PERT and CPM, based on activities' early start dates.

Kavanagh (1985) developed a scheduling system called SIREN (Simulation of Repetitive Networks). He combined two concepts of the network scheduling technique and the queuing technique into a simulation system (GPSS) to solve repetitive project scheduling problems. Activities (customers) are queued for their resources (servers). This is used as resource constraint. Another type of constraint is precedence constraint. Activity can start when these two constraints are satisfied.

Limitations of Kavanagh's model are discussed in the order of his suggestions on future enhancements for his model as followings:

- Allowing changes in the number of repetitive networks
- Allow changes in crew size
- Control the beginning of each activity
- Model the effect of weather on activity
- Allow users to impose priorities and plan of work on the system.

Kavanagh's suggestions are the primary focus for the Sequence Step Algorithm (SQS-AL) proposed in this research. The enhancement due to the suggestions will improve the efficiency of resource utilization. Since Kavanagh's model cannot control

the start date of an activity, his model has the same limitation as Ashley's; it fails to maintain resource continuity.

Lutz (1990) is another research using simulation to solve scheduling problems of repetitive projects. His work focuses on applying learning phenomenon, cycle monitoring, and buffer monitoring. For the effect of learning phenomenon, Lutz modified Hijazi's work and coded it in MicroCYCLONE to improve the usability of Hijazi's original work.

Lutz's buffer monitoring is used to control the pace of construction processes. Additional queues are introduced into the network to determine when a succeeding process should start, continue working, or wait. The queues are placed between construction processes, preceding process and succeeding process. When a buffer queue (e.g., Buffer A-B) is empty, it indicates the preceding activity (e.g., Activity A) cannot keep up with its succeeding activity (e.g., Activity B). Thus, the succeeding activity will become idle until the queue is not empty.

Nevertheless, Lutz's method does not automatically determine the delay duration needed to achieve continuity in resource utilization. It requires users to provide the maximum numbers of units in the buffer queues prior to the simulation execution. Repeated trial-and-errors in simulation runs are required to derive a satisfactory number of units in the buffer queues. Relying on user input of the specified number of units in a buffer queue, Lutz's model does not guarantee continuity in resource utilization (Yang 2002).

In addition, the means of delaying activities based on the number of units may result in a remaining idle time or an unnecessary delay. The delay (buffer) is better defined in units of time rather than in units of work.

Many researchers have proposed other techniques employing simulation to solve the problems of repetitive project scheduling since 1990. They offer user-friendly interface tools that facilitate the means of constructing simulation models. However, most of these techniques fail to control activity start dates, and thus cannot 1) maintain resource continuity, 2) allow work interruption, and 3) analyze the tradeoff between the first two to optimized project duration and cost.

In order to use simulation to solve repetitive project problems, it is mandatory to consider the problems in the sense of scheduling concepts and underlying nature of repetitive projects rather than the processes of modeling and inputting data.

2.3 Summary

Many approaches have been proposed to solve the problems of scheduling repetitive projects since 1960. Although the Critical Path Method (CPM) can be applied to repetitive projects, it cannot schedule repetitive projects effectively. Because CPM is designed for optimizing project duration, it does not suit the resource-driven nature of repetitive projects. Moreover, constructing a CPM network for a repetitive project can be tedious and cumbersome when the project consists of many repetitive units. Similarly, the Program Evaluation and Review Technique (PERT) has the same limitations as does CPM. Moreover, simulation has superseded PERT due to ease of use and richer information in simulation systems.

Ever since the Line-Of-Balance technique (LOB) was introduced to the construction industry, it has been influencing many researchers to improve the means of scheduling repetitive construction projects. In the early development of graphical techniques, many researchers have adopted the concept of LOB, and focused on balancing crew production and maintaining continuity of work flow. However, since LOB was originated from the manufacturing industry, simplifications were required in order to enable these graphical methods to solve scheduling problems of repetitive construction projects. Consequently, many assumptions of LOB for industrial projects conflict with the nature of construction projects. The main conflict is the assumption of constant production rates, which applies only to typical activities.

After the advent of the personal computer and the improvement in computing capability, many mathematic approaches have been established to solve repetitive project scheduling problems. Sophisticated mathematic models were developed to optimize project duration and cost problems. Advancement of these models was established to mathematically model complicated and realistic construction operations. However, the usability of these methods is only effective to a certain extent. Their limitations are in capturing the stochastic nature of construction activities. Using linear and dynamic programming techniques (LP and DP) in optimizing probabilistic scheduling problems is difficult. As a result, most LP and DP approaches assume activity durations are deterministic.

In contrast to graphical and mathematical methods, simulation has the ability to model variability and uncertainty, inevitable parts of construction. Even more beneficial, since repetitive projects are resource-driven projects, resource-driven simulation systems

are probably one of the most promising tools to solve the problems. Its already-included resource constraints reduce users' effort in constructing resource availability constraints.

However, simulation by itself cannot solve repetitive project scheduling problems effectively; it cannot ensure the resource continuity constraints, and cannot control the degree of continuity in resource utilization. Without implementing an external algorithm to a simulation system, simulation leads to the same results as CPM, where activities are scheduled at their early start dates, and interruptions exist among units. Thus, an external algorithm must be designed to assist the simulation system in order to solve the problems.

Finally, there is a need to establish a comprehensive generalized technique that can 1) effectively capture the realistic characteristics of repetitive construction projects, 2) flexibly maintain and relax the resource continuity constraints, 3) sophisticatedly optimize project cost, and 4) practically offer an ease of use to users.

CHAPTER 3

REPETITIVE SCHEDULING METHOD

This chapter discusses the fundamental concepts of repetitive project scheduling. The Repetitive Scheduling Method (RSM), a graphical approach developed by Harris and Ioannou (1998), is used to clarify and emphasize important aspects of repetitive project scheduling as follows:

- 1) Satisfying precedence and resource availability constraints
- 2) Maintaining continuous resource utilization
- 3) Scheduling and re-scheduling repetitive activities to achieve (1) and (2)
- 4) Determining critical activities
- 5) Identifying controlling sequences and controlling activities

The concepts and aspects of repetitive project scheduling are a prerequisite to development of a new scheduling method. The scheduling processes RSM, a graphical approach, contribute significantly to the establishment of the Sequence Step Algorithm (SQS-AL), a simulation approach, proposed in this research. To schedule repetitive projects with probabilistic activity durations, SQS-AL adopts the RSM sequential process of calculating idle time and determining activity start date, discussed in Chapter 4 Sequence Step Algorithm.

Moreover, the idea of the controlling sequence in RSM, originally for repetitive project scheduling problems with deterministic activity durations, is adapted to suit problems with probabilistic durations in SQS-AL. SQS-AL employ the underlying concept of controlling sequence to determine effective work break positions in repetitive activities, used to reduce project duration.

3.1 Introduction

Construction often involves repetitive projects where several identical (typical) or similar (non-typical) units require resources to perform their activities in specified sequences repeatedly from the first to the last unit. Precedence constraints and resource availability constraints determine both activity start dates and resource arrival dates; activities can start only when their preceding activities in the same unit are completed and the required resources are available. The Critical Path Method (CPM) satisfies these conditions by using precedence constraints to represent both precedence and resource availability constraints. As shown in Figure 3.1, the precedence network presented by Harris and Ioannou (1998) illustrates the application of CPM to a repetitive project consisting of three non-typical units. The precedence constraints among activities in the same unit are technical constraints. For example in Figure 3.1, Activity A1 must be completed before Activity B1 can begin. This precedence constraint specifies that A1 and B1 are technically dependent, but not resource dependent. On the other hand, the precedence constraints among units of the same activity are resource availability constraints and/or technical constraints. For example in Figure 3.1, A1 and A2, sub-activities of Activity A, require the same resource. If the resource required to perform

Activity A, Resource A, is limited, Resource A must finish A1 before starting A2. As shown in Figure 3.1, precedence relationships are used to constrain these conditions.

It should be noted that if sub-activities are technically dependent they are imposed by hard logic relationships requiring both technical and resource constraints to be satisfied. For example, structural work of the first floor must be completed and resources must be available before work in the second floor can start. On the other hand, if sub-activities are not technically dependent, they should be modeled by soft logic relationships requiring only resource constraints to be satisfied. For example in a housing project, excavation activities for the second house can start before the first house if resources are available. The order of the units and the order of construction are not necessarily the same.

Because resource continuity constraints are not included in CPM, CPM cannot prevent interruptions and idle time in resource utilization. From Figure 3.1, Resource B will become idle from day 5 to 7 because Activity B1 is scheduled from day 3 to 4, and B3 from day 8 to 9. Thus, it is necessary to introduce resource continuity constraints to eliminate idle time. Harris and Ioannou (1998) proposed the Repetitive Scheduling Method (RSM), a graphical method that incorporates three types of constraints: precedence, resource availability, and resource continuity constraints. As a result, RSM is able to eliminate idle time and achieve continuous resource utilization.

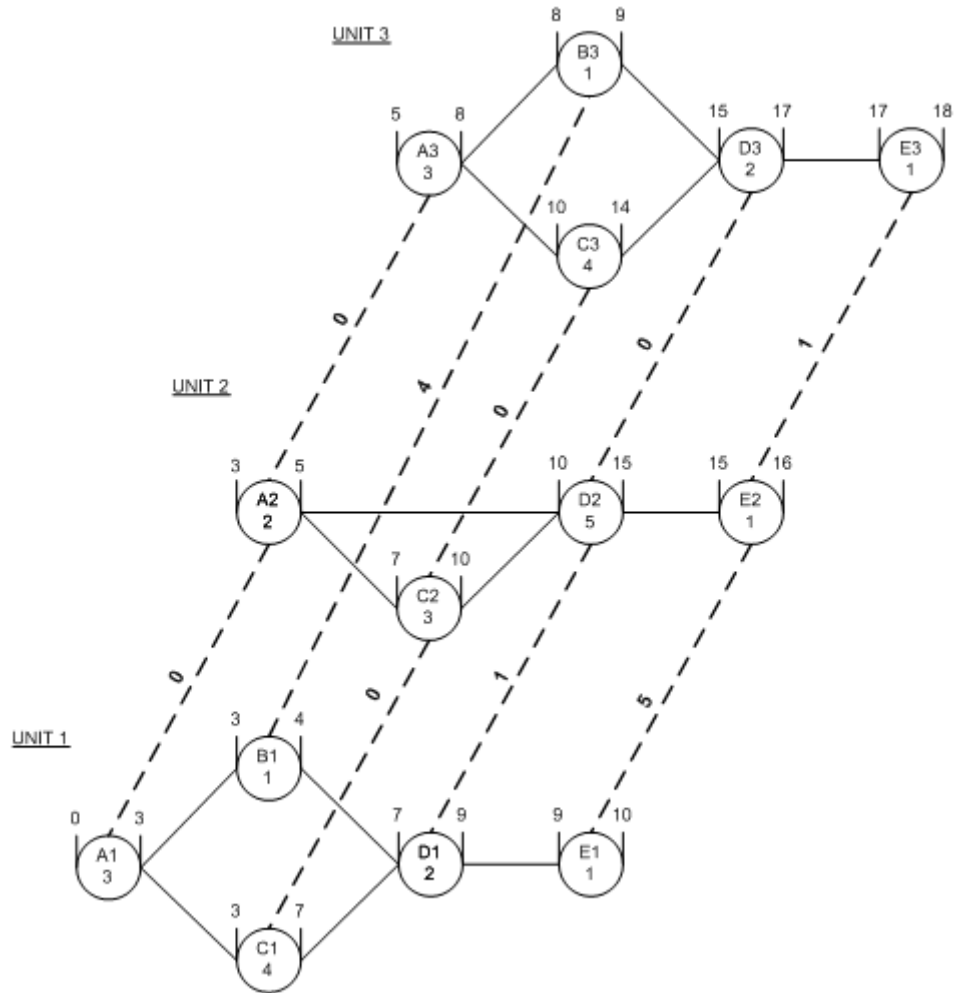


Figure 3.1 CPM network for three repetitive units (from Harris and Ioannou 1998)

3.2 Maintaining Continuity in Graphical Methods

The Repetitive Scheduling Method (RSM) is a graphical method that ensures continuity of resource utilization. Like Line-Of-Balance (LOB), discussed in Chapter 2, RSM uses production lines to represent activities instead of using nodes as the CPM network does. For discrete repetitive projects (e.g., high-rise buildings), construction units are shown on the Y axis and time is shown on the X axis. Figure 3.2 displays production diagram of the repetitive project in Figure 3.1, using the early start dates

imposed by precedence constraints. In Figure 3.2, solid lines indicate activities for which resource continuity constraints have been satisfied.

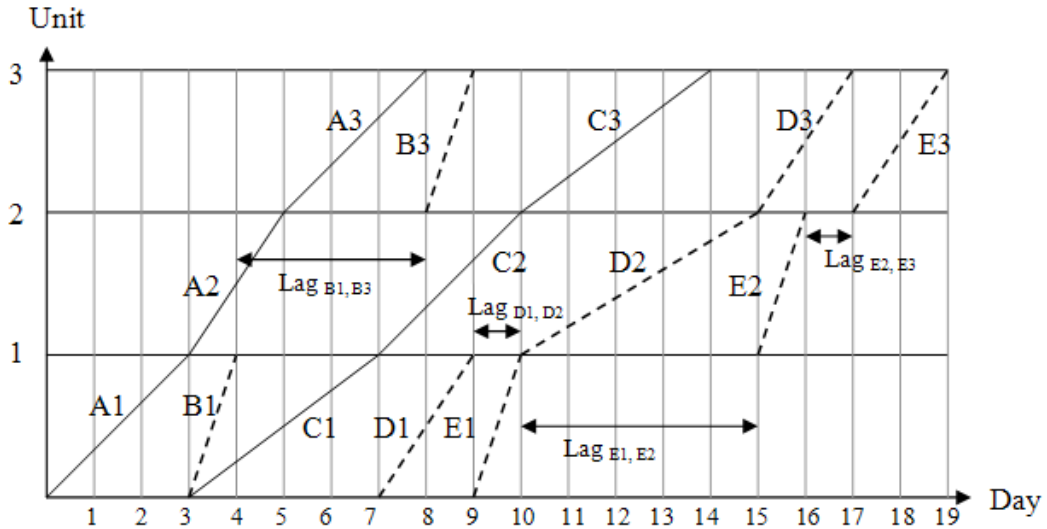


Figure 3.2 RSM Diagram for Three Units based on Precedence Constraints

As shown in Figure 3.2, precedence constraints do not ensure continuous resource utilization in the activities. Interruptions and idle time exist in Activities B, D, and E. There is an interruption of 4 days between B1 and B3, 1 day between D1 and D2, and so forth. In order to eliminate these interruptions, the sub-activities in the first unit (e.g., B1) must be postponed for the period of the sum of all the lags between sub-activities. Thus, activity B1 must be postponed for the amount of duration equal to $Lag_{B1, B3}$; Activity D1 for $Lag_{D1, D2}$; Activity E1 for $Lag_{E1, E2}$ plus $Lag_{E2, E3}$. Figure 3.3 exhibits the result of postponing the discontinuous Activities B, D, and E in Figure 3.2.

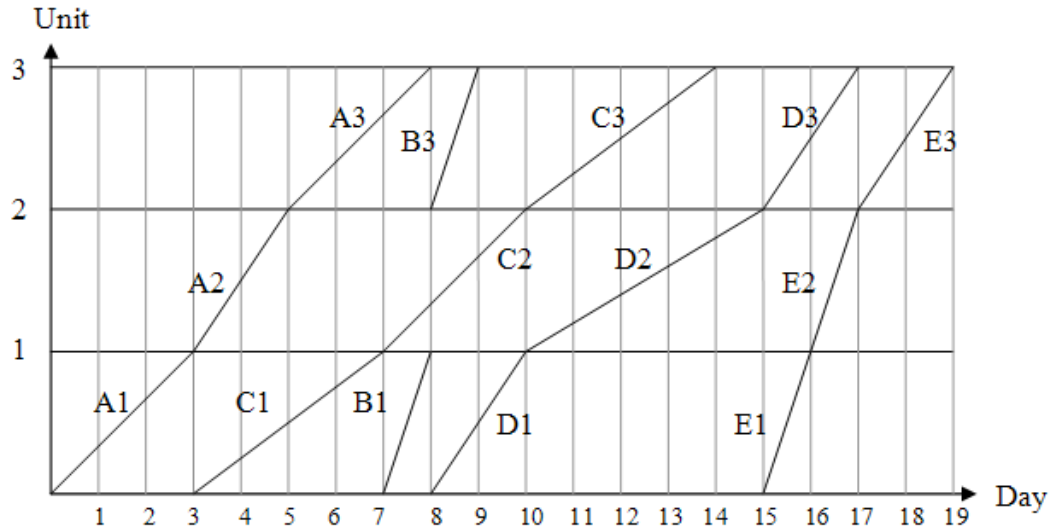


Figure 3.3 Postponing activities with interruptions in Figure 3.2 to achieve continuous resource utilization

After the first sub-activities in B, D, and E are postponed by the sum of lags, as shown in Figure 3.2, their continuities are achieved in Figure 3.3. Clearly, duration of lags must be calculated before determining how many days an activity (the first sub-activity) must be postponed to achieve its continuous resource utilization.

However, postponing activities for the amount of lags originally derived from the early start date schedule may violate precedence constraints. Another example, shown in Figure 3.4, demonstrates this situation. Figure 3.4.a shows the result of using CPM to schedule the example. Interruptions exist between B1 and B3, between E1 and E2, and between E2 and E3.

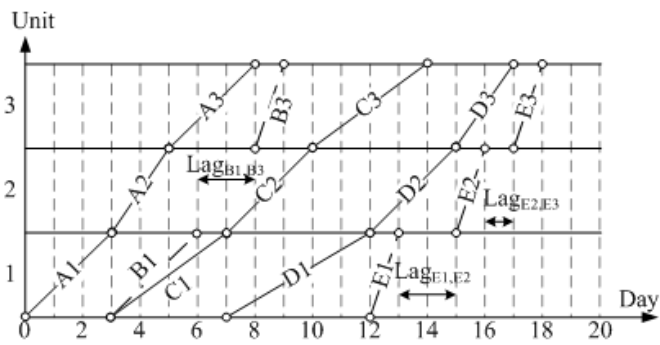
According to the previous example, continuity constraints can be satisfied by postponing the first sub-activity for the sum of lags between sub-activities. Doing so satisfies resource continuity constraints; however, the precedence constraint between B1 and D1 is violated, as depicted in Figure 3.4.b. Thus, it is necessary to re-check the precedence constraints. Rescheduling activities to rectify precedence constraints may be

necessary. To correct the precedence constraints after activities are postponed, the successor whose precedence constraint is violated must be postponed while its predecessor remains the same. In this example, Activity D1 is postponed and, consequently, D2 and D3 are also postponed on the basis of precedence constraints. Figure 3.4.c shows the result from correcting the violation.

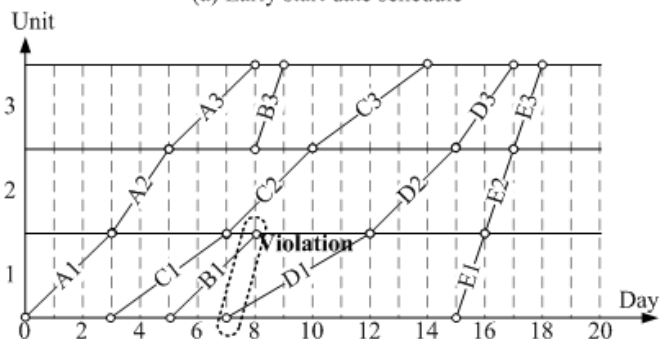
After activities are postponed in Figure 3.4.b, and precedence constraints are corrected in Figure 3.4.c, activities may become discontinuous again. Figure 3.4.c shows that Activity E now has idle time between Units 2 and 3 ($Lag_{E2, E3}$). Thus, activity E1 must be postponed for the amount of the lag to eliminate the interruption. Figure 3.5.d is the finalized schedule for the example where precedence, resource availability, and resource continuity constraints are satisfied.

The example in Figure 3.4 demonstrates that the process of scheduling repetitive activities requires repetitions of postponing activities to satisfy resource continuity constraints. As shown in Figure 3.4, it is difficult to predetermine how many times exactly a repetitive project needs to be scheduled and re-scheduled.

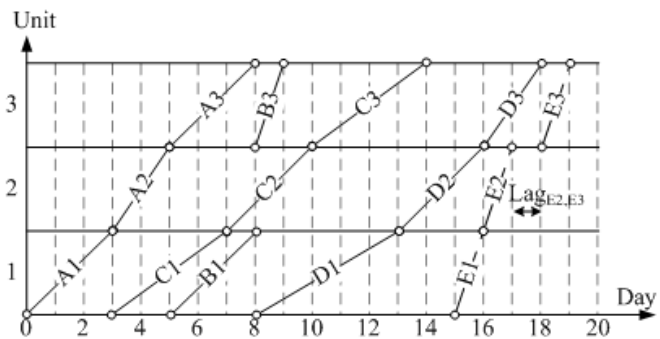
To alleviate the difficulty, RSM schedules activities (e.g., A and B) in a precedence order, one activity at a time. Thus, activities in Figure 3.4 should be scheduled in the order of A, B, C, and so forth. During scheduling an activity (e.g., A), continuity constraints (e.g., between A1 and A2, and between A2 and A3) must be achieved before scheduling its succeeding activities (e.g., B). This process guarantees the satisfaction in precedence and continuity constraints. Moreover, the process of scheduling one activity at a time also eliminates the redundancy in re-scheduling activities, whose work interruptions may be incurred because of the process of scheduling.



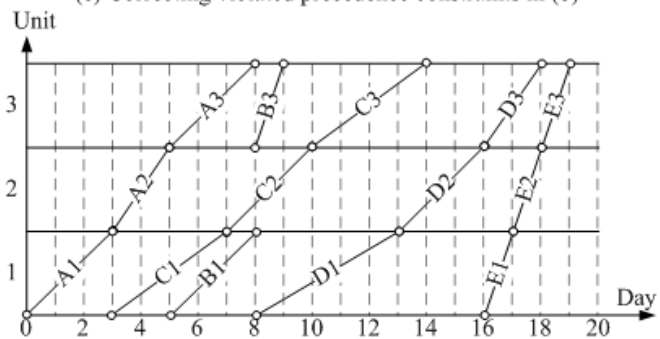
(a) Early start date schedule



(b) Delaying Activities B and C by their idle time in (a)



(c) Correcting violated precedence constraints in (b)



(d) Delaying Activity E by its idle time in (c)

Figure 3.4 Satisfying precedence and continuity constraints

Importantly, the aforementioned idea, a graphical approach, must be improved and systemized in order to derive an algorithm that is applicable to repetitive projects with probability activity durations. The algorithm, namely “Sequence Step Algorithm” (SQS-AL), is a generalized algorithm using sequence steps to systematically schedule repetitive activities. The algorithm is designed based on the current applicability and capability of discrete-event simulation; moreover, the characteristics of repetitive projects are always a concern during the designing stage. Further discussion regarding SQS-AL is provided in Chapter 4.

3.3 Critical Activities and Controlling Sequence

One important and widely used piece of information is the criticality of activities. To successfully manage a construction project, project managers and site engineers must realize the criticality of activities. The criticality indicates the impact of an activity on project duration. If delaying an activity delays project completion, the activity is considered a critical activity (see more detailed discussion in section 3.3.1 Critical Activities). A sequence of critical activities from the project start to finish is called the “critical path.” The sum of activities on a critical path equals the minimum project duration. Accordingly, attention is given to critical activities and critical path to ensure the project will finish on time.

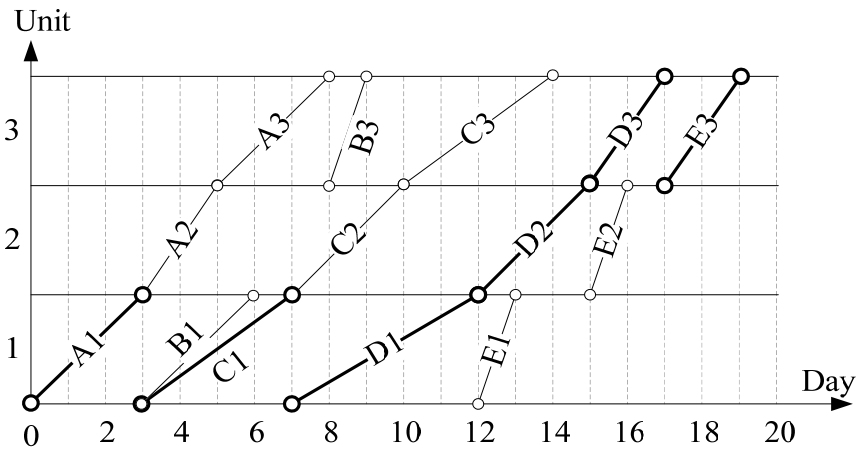
However, in repetitive projects, critical activities do not provide sufficient information for controlling and monitoring. In repetitive projects, the critical path do not determine the minimum project duration because postponing activities to maintain the continuity constraints usually incurs floats between repetitive activities. Accordingly, the critical path does not exist in repetitive projects after the postponement (Harris and

Ioannou 1998). In response to the insufficient information of activity criticality in repetitive projects under continuity constraints, Harris and Ioannou (1998) introduced the concept of the “Controlling Sequence.”

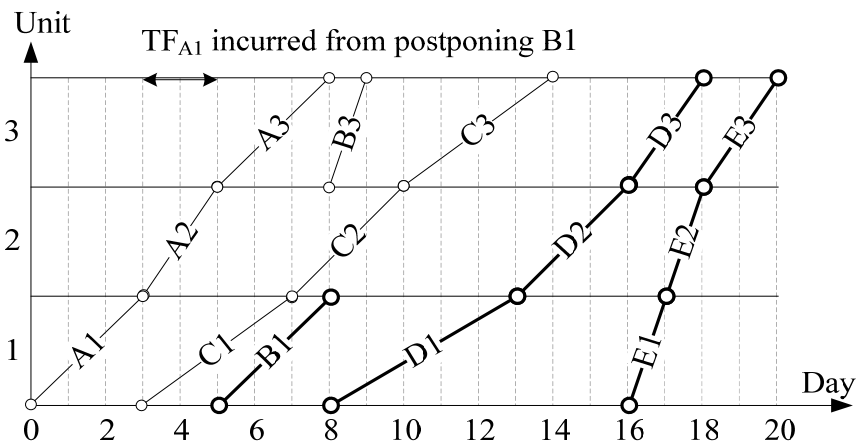
3.3.1 Critical Activities

Critical activities are those activities, if delayed during construction, will delay project completion. In CPM, the calculation of total float is used to determine critical activities. If an activity has zero total float, it is considered a critical activity. The calculation is a backward calculation because it proceeds from the last activity in precedence order to the first. The calculation of float can be applied to any project schedule. For example, in CPM the calculation is performed when activities are at their earliest start positions. On the other hand, in RSM the calculation is performed when activities are in the positions where their continuity constraints are maintained.

It is important to realize that different scheduling methods may result in different sets of critical activities. Comparison between Figure 3.5.a and Figure 3.5.b illustrates critical activities derived from CPM are different from those derived from RSM. Figure 3.5.a is a production diagram where activities are scheduled using CPM; critical activities from CPM are A1, C1, D1, D2, D3, and E3. Figure 3.5.b is an RSM diagram for the same project. Critical activities from RSM as indicated with heavy solid lines in Figure 3.5.b are B1, D1, D2, D3, E1, E2, and E3.



(a) Production diagram from CPM



(b) Production diagram from RSM

Figure 3.5 Different critical activities between CPM and RSM
(Critical activities are indicated by solid heavy lines)

Another difference between CPM and RSM is the implication of critical activities for minimum project duration (Harris and Ioannou 1998). In CPM, the sum of critical activity durations on a critical path equals minimum project duration; however, this is not the case in RSM. As the number of sequence steps increases, critical activities from RSM tend to appear only in the later sequence steps. Figure 3.5.b shows most critical activities are in sequence steps 3, and 4. Typically in RSM, activities in the early sequence steps are not critical because of the floats incurred from postponing activities in RSM. The predecessors of the postponed activities gain float and become non-critical activities. For

example in Figure 3.5, critical activity A1 in CPM becomes non-critical in RSM because postponing activity B1 results in float in activity A1.

Moreover, since critical activities cannot be used to determine minimum project duration for repetitive projects, it can be concluded that critical activities in repetitive projects do not provide sufficient information for planning and control purposes. Consequently, another way of determining the criticality of activity and minimum project duration is needed.

3.3.2 Controlling Sequence

The concept of controlling sequence, developed by Harris and Ioannou (1998), addresses the deficiency of critical activity information in repetitive projects. They realized that the traditional concept of critical activities could not determine the minimum project duration in repetitive projects under resource continuity during planning. Harris and Ioannou introduced two new concepts: Control Points and Controlling Sequence. These are “pre-construction” concepts used in planning and scheduling the project.

A control point between two activities is the precedence constraint that determines the start date of the successor under continuity constraints. For example, in Figure 3.6, the control point between Activities A and B is the precedence constraint in Unit 3. To eliminate idle time in an activity (e.g., B in Figure 3.6), its sub-activities (e.g., B1) are pulled toward the control point (e.g., the start date of B3).

A controlling sequence is an uninterrupted sequence of activities that navigates through activities and control points from the project start to the project finish that determines the planned project duration while ensuring resource continuity (Harris and Ioannou 1998). Figure 3.6 shows the controlling sequence from the previous example in

Figure 3.5.b by using heavy solid lines to represent activities on the controlling sequence, which are E3, D3 to D1, and A3 to A1. This sequence of activities determines the minimum project duration while maintaining continuity constraints of resource utilization. Delaying the start date of an activity on the controlling sequence will either postpone project completion date if resource continuity can still be maintained or interruption in resource utilization if resource continuity cannot be maintained. For example in Figure 3.6, if A1 is delayed, B3 will be delayed, and B1 must be postponed in order to maintain the continuity in Activity B. This is the case where the continuity in B can be maintained. Thus, D1 will be delayed because of the precedence constraints between B1 and D1. Consequently, project completion date is delayed. On the other hand, if A3 is delayed and B1 finishes on time (Day 8), interruption in Activity B is inevitable. Schedulers should realize that activities on a controlling sequence have an impact on both project completion and continuous resource utilization of repetitive projects.

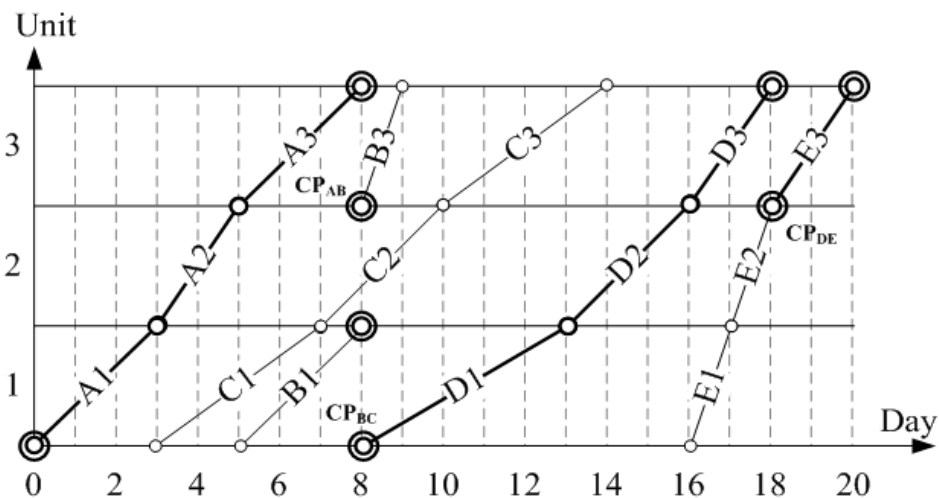


Figure 3.6 Production Diagram from RSM with controlling sequence
(The controlling sequence is indicated by bold lines)

From the planning and scheduling perspective, the controlling sequence provides valuable information, which determines project duration, and provides insight into how to expedite the project. Consider the diagram in Figure 3.6, altering a production rate of an activity on the controlling sequence will change the project duration. For an example, shortening duration of a controlling activity (e.g., A1 in Figure 3.6) results in shorter project duration. On the other hand, shortening duration of a non-controlling activity (e.g., C1 in Figure 3.6) does not affect the project duration.

It is important to realize that accelerating production rates of activities on controlling sequence does not necessarily result in shorter project duration. Because resource continuity constraints are maintained in the RSM, altering production rate of a controlling activity could either allow its successor to start earlier or delay its successor. This is a very complicated issue. Besides, precedence, resource availability, and resource continuity constraints, relative production rates between two dependent activities also contribute to the minimum project duration of a repetitive project. It is recommended to use trial-and-error approach to check whether accelerating an activity produces a shorter project duration or not.

3.4 Summary

In this chapter, the concepts of repetitive project scheduling have been discussed. A graphical method called the "Repetitive Scheduling Method" (RSM) is used to illustrate the processes of deriving a repetitive project schedule that satisfies precedence, resource availability, and resource continuity constraints.

Critical activities are activities that, if delayed, will be delayed. Critical path is a series of critical activities from the start to the finish of the project. While project

duration of non-repetitive projects can be determined by the sum of activities on the critical path, project duration of repetitive projects cannot be calculated by the same method. Critical path does not exist in RSM schedule because of the continuity constraints. Accordingly, Harris and Ioannou (1998) introduced the concepts of control point and controlling sequence to resolve the informative deficiency of critical path in repetitive projects. Controlling sequence in a repetitive project determines the minimum duration of the project. Controlling sequence is a set of sub-activities (e.g., A1, A2) navigating from the project finish date to the project start date through control points. Control point is the point that a predecessor's finish date controls its successor's start date under the continuity constraints.

RSM is an effective and simple method to schedule repetitive projects. Nevertheless, there is a need for improvement. Many subjects need to be taken into account of scheduling repetitive projects as discussed in Chapter 2 Literature Review. RSM and other non-computerized methods should be computerized to lessen human effort in inputting data, updating schedule, and analyzing schedules in terms of project cost. A newly developed method should be capable of capturing the stochastic nature of repetitive projects under the constraints (precedence, resource availability, and resource continuity).

In the next chapter, a new algorithm and technique are proposed; it is called "Sequence Step Algorithm" (SQS-AL). After the algorithm is discussed in Chapter 4 Sequence Step Algorithm, Chapter 5 Simulation Model Templates presents a suggested simulation model for modeling repetitive projects in discrete-event simulation system. The integration between SQS-AL and the proposed simulation model templates provides

a comprehensive approach that is able to solve the complicated scheduling problems of stochastic repetitive projects.

CHAPTER 4

SEQUENCE STEP ALGORITHM

This chapter presents the proposed Sequence Step Algorithm (SQS-AL) and discusses the comparison between SQS-AL, the Critical Path Method (CPM), and the Repetitive Scheduling Method (RSM). SQS-AL is a generalized methodology for scheduling repetitive projects. It is capable of solving both deterministic and stochastic repetitive project scheduling problems. The beginning of this chapter discusses two important components of the algorithm: sequence steps and idle time. SQS-AL uses sequence steps to orderly determine idle time of repetitive activities and schedules the activities.

After the discussion of sequence steps and idle time, an overview of SQS-AL is given along with an example; the purpose of the overview is to show the simplicity of the algorithm. Then, different types of idle time are introduced prior to the discussion of the SQS-AL implementation in simulation. Modeled in Stroboscope, a discrete-event simulation system, an example of a repetitive project with stochastic activity durations is used to demonstrate the application of SQS-AL. In the end, the results derived from SQS-AL are compared to those of CPM and RSM.

4.1 Sequence Steps

Sequence steps are visual columns in a precedence diagram on which activities are placed; they are used mainly for graphical presentation. In a precedence diagram, activities are presented by nodes (circular or rectangular shapes), whereas precedence relationships between activities are presented by links (connecting lines). Positioning nodes (activities) in sequence steps results in an organized precedence diagram.

In a precedence diagram, every activity belongs to a particular sequence step. It is defined to be the left-most visual column in which an activity may be placed and still maintain left-to-right precedence relationships. For example, activities in Figure 4.1 are placed in three sequence steps, from Sequence Step 1 (SQS1) to Sequence Step 3 (SQS3).

4.1.1 Example 4.1 Determining sequence steps for a repetitive project with three activities

Activity Name	Predecessors
A	-
B	A
C	B

Table 4.1 Precedence relationships for Example 4.1

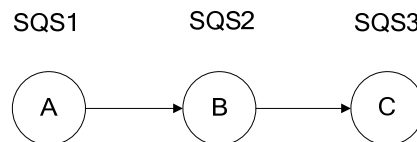


Figure 4.1 Precedence diagram for Example 4.1

Another example is illustrated below. Activities in Figure 4.2 are placed in four sequence steps from SQS1 to SQS4.

4.1.2 Example 4.2 Determining sequence steps for a repetitive project with seven activities

Activity Name	Predecessors
A	-
B	A
C	A
D	B and C
E	B
F	C
G	D, E, and F

Table 4.2 Precedence relationships for Example 4.2

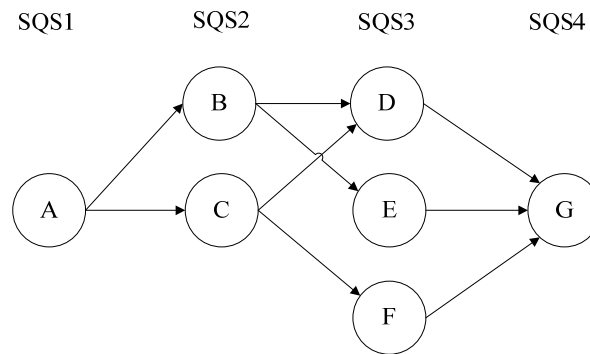


Figure 4.2 Precedence diagram for Example 4.2

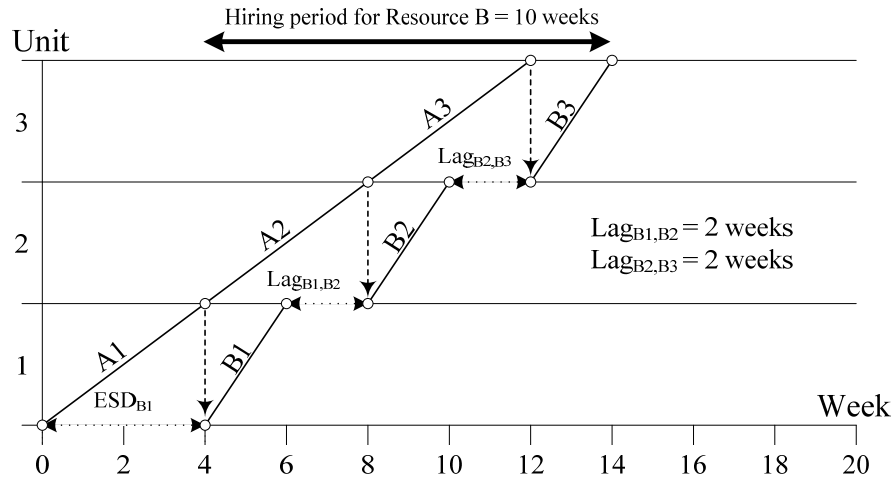
Sequence steps are an important property of a precedence diagram. Indeed, the CPM calculation for precedence networks can be performed very easily by using sequence step order: i.e., left-to-right for the forward pass and right-to-left for the backward pass. As explained later, this property is at the heart of SQS-AL. It is used to specify the order in which idle times of repetitive activities are collected.

4.2 Two Different Types of Idle Time in Repetitive Activities

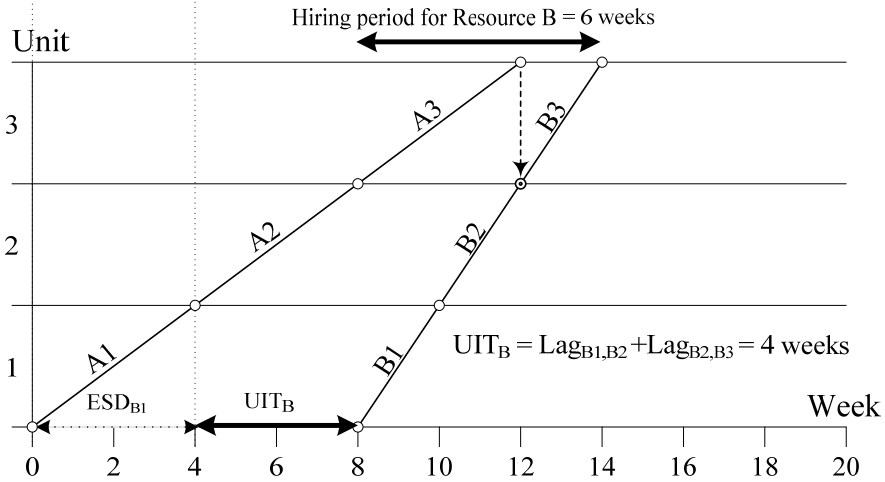
Idle time is the duration that a crew performs no work during their employment period, i.e., getting paid but not producing any output. In SQS-AL, idle time is

categorized into two main types, idle time between units (UIT) and idle time at arrival (AIT). The sum of these two types of idle time for a resource is its total idle time, called “Crew Idle Time” (CIT). From the number of simulation replications, CITs for each resource are collected. After the statistical data of CITs of resources are obtained, SQS-AL uses the statistical data to determine resources’ arrival dates that minimize their idle time. As a result, continuous resource utilization for the resources is improved.

Unit Idle Time (UIT) is the sum of idle time (or duration of lags) between units of a repetitive activity. As can be seen in Figure 4.3.a, UIT of Activity B is the lag between B1 and B2 ($UIT_{B1,B2} = Lag_{B1,B2} = 2$ weeks), and between B2 and B3 ($UIT_{B2,B3} = Lag_{B2,B3} = 2$ weeks). The sum of $UIT_{B1,B2}$ and $UIT_{B2,B3}$ equals UIT_B ($UIT_B = 4$ weeks). Postponing the start date of B from its earliest start date (ESD) by UIT_B (4 weeks) results in zero idle time in B. Given that activity durations are deterministic, ESD and UIT are constant; the calculation of UIT is sufficient to determine the postponement period of an activity in order to eliminate its idle time, specifically UIT. However, this is not the case for projects with stochastic activity durations.



(a) CPM schedule

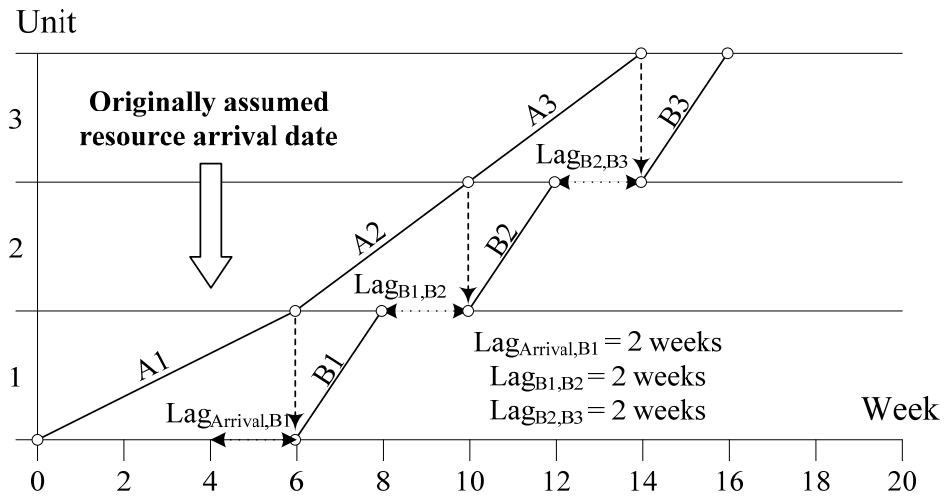


(b) RSM schedule

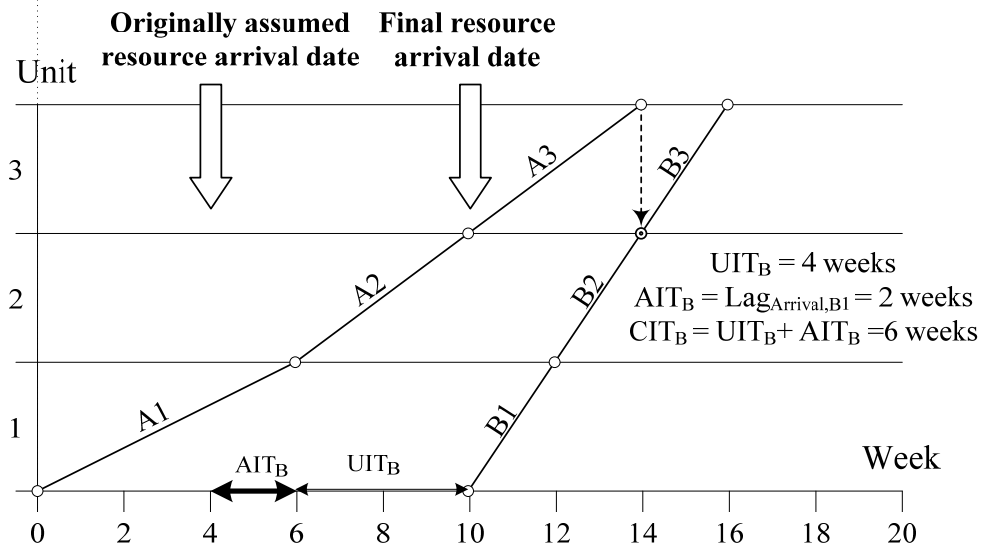
Figure 4.3 Production diagram showing a delay in Activity B by its Unit Idle Time

In probabilistic scheduling problems, activity durations are stochastic, resulting in varying finish dates of activities and therefore varying start dates of their successors. It is possible that resources may arrive to the site as they are scheduled, but not be able to start the work because their predecessors have not yet finished. This situation incurs idle time between the arrival date of the resource and its actual start date in the first unit. This type of idle time is called “Arrival Idle Time” (AIT). An example of AIT is shown in Figure 4.4.

Figure 4.4.a shows what happens to Figure 4.3.a if by chance Activity A1 does not take 4 weeks as planned but actually takes 6 weeks. As shown in Figure 4.4.a, the resource serving Activity B is scheduled to arrive at the end of the fourth week; however, the resource cannot work since A1 has taken longer to do and has not been completed.



(a) CPM schedule



(b) RSM schedule

Figure 4.4 Arrival idle time between the arrival date and the start date of Resource B

For this particular example in Figure 4.4, the resource arrival date for Activity B must be postponed by the sum of UIT and AIT, which is CIT. Figure 4.4.b shows that delaying the arrival date of B from its preliminary scheduled arrival by 6 weeks, which is its crew idle time (CIT_B), will eliminate the entire idle time in B. More precisely, when the start date of an activity varies from the original schedule, both arrival idle time (AIT) and unit idle time (UIT) must be accounted for in crew idle time (CIT).

Note that AIT does exist in CIT for deterministic scheduling problems because in the absence of randomness the idle time between resources' arrival dates and resources' actual start dates is always zero.

In probabilistic scheduling problems, since activity durations vary, it is most convenient that activity start dates and resource arrival dates are measured from the project start date. Accordingly, SQS-AL assumes that all resources, for which no specific arrival date have been scheduled yet, arrive to the site at the beginning of the project (i.e., time zero). This assumption in SQS-AL is very important for calculation purposes, as explained in detail later.

As shown in Figure 4.5, the arrival date of Resource B is assumed to be at the beginning of the project. This assumed arrival date will be changed later to finalize the arrival date of B by using the collected CIT_B and user-desired confidence level, explained later. The finalized arrival date of a resource is called the "Crew Lead Time" (CLT), a duration measured from project start to the arrival date of the resource.

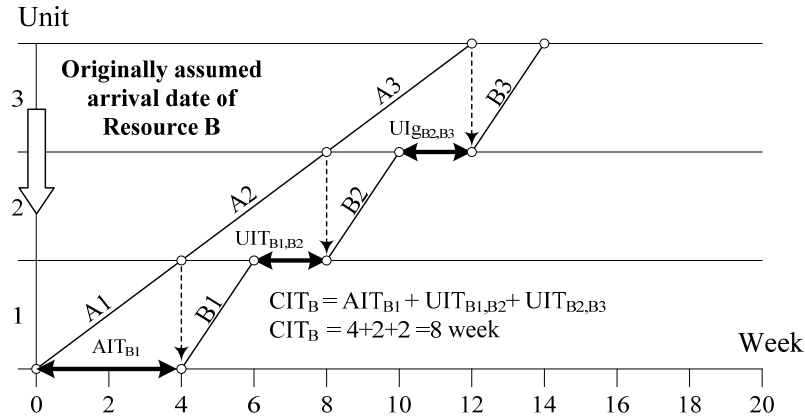


Figure 4.5 SQS-AL’s assumption of resource arriving at the beginning of the project

4.3 Confidence Levels and Crew Lead Times

Confidence levels are user-desired parameters in percent specifying degree of confidence that a particular resource will work continuously without interruptions. In other words, a confidence level is a user-specified probability that a particular resource will have zero idle time based on the data set (i.e., distribution) of that crew’s idle time (CIT), collected from simulation runs. The collected CITs are used to construct a cumulative frequency of the CITs for that particular crew as shown in Figure 4.5. Using the constructed cumulative frequency of CITs (Figure 4.5) and user-desired confidence levels (e.g., 50% and 100% in Figure 4.5), crew lead time (CLT) is chosen from the constructed cumulative frequency of CITs and the corresponding value of the confidence level. Essentially, CLT is the finalized arrival date (e.g., 13 days for 50% confidence level and 22 days for 100% confidence level in Figure 4.5) of the resource measured from the project start date. Clearly, higher confidence levels lead to longer crew lead times.

Figure 4.6 shows an example of cumulative frequency of CIT_B , which is collected from N replications. If Resource B is scheduled to arrive at the site at the end of Day 13

($CLT_B = 13$ days), then there is 50% probability that its crew idle time (sum of idle time, AIT+UIT) will be less than 13 days and its continuous utilization (zero idle time) will be achieved 50% of the time. On the other hand, if CLT_B is set to 22 days, then its crew idle time is guaranteed to be entirely eliminated (based on the collected CITs) and thus its continuous utilization is 100% guaranteed; there is no idle time. Clearly, the accuracy of these predictions depends on the variability of activity durations and the number of replications.

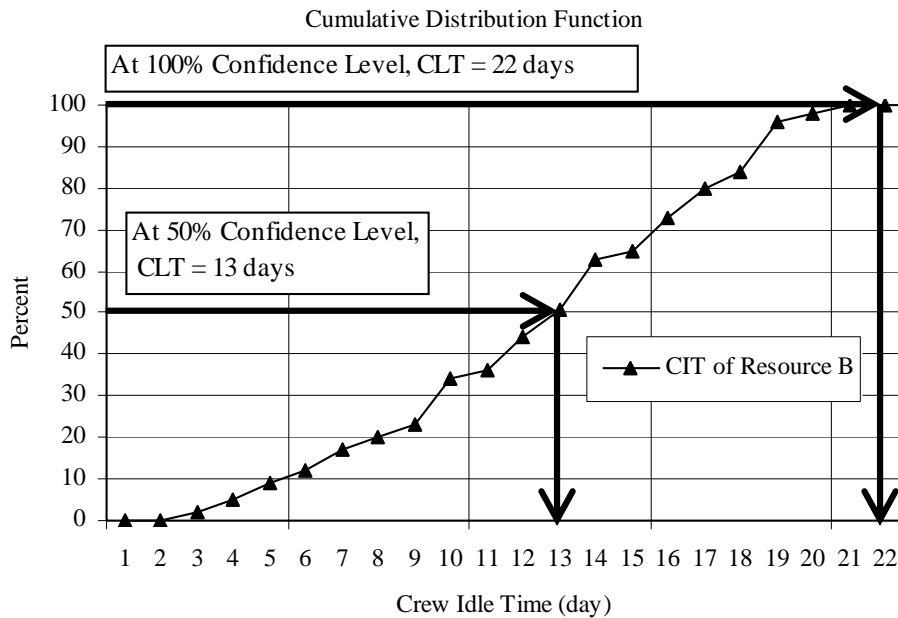


Figure 4.6 Determining crew lead time from the collected crew idle time and user-specified confidence level

4.4 Overview of the Sequence Step Algorithm

Four main components of SQS-AL, 1) sequence steps, 2) idle time, 3) confidence levels, and 4) crew lead time, have been presented in the previous section. This section presents an overview of the concepts of SQS-AL by solving a scheduling problem of a repetitive project with 3 activities consisting of 3 units. The purpose of SQS-AL is to

determine the arrival date for each resource, so it may work continuously without interruption. To determine crew lead time (CLT) of a resource, SQS-AL performs the following four main steps:

- 1) Collect crew idle time (CITs)
- 2) Construct a cumulative frequency distribution of the collected CITs
- 3) Select a confidence level and use it in item (4)
- 4) Determine crew lead time (CLT) from the cumulative CIT using a user-specified confidence level in item (3)

These four steps are combined as “processing a sequence step.” These four steps are repeated for the number of sequence steps in the precedence diagram (to determine arrival dates) plus one extra sequence step (to obtain the finalized idle time, not the arrival dates). To demonstrate SQS-AL’s processing sequence steps, an example of a repetitive project consisting of 3 units with 3 activities in each unit is demonstrated. The node network represents a single unit of the example project shown in Figure 4.3. Activity durations are given in Table 4.3. The formula shown in the table is in Stroboscope language where the first and the second numbers in the bracket are means and standard deviation, respectively.

4.4.1 Example 4.3 Determining crew lead time in a repetitive project with three activities with probabilistic activity duration

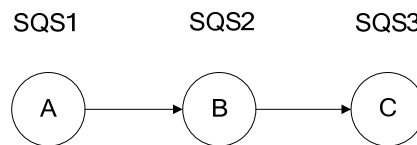


Figure 4.7 The single unit precedence diagram for Example 4.3

Activity	Duration
A	Uniform[20,2]
B	Uniform[10,1]
C	Uniform[15,1.5]

Table 4.3 Stochastic durations of repetitive activities for Example 4.3

In the first step, 3000 replications of the simulation are executed to collect 3000 CITs of activities belonging to the considered sequence step. For this example, it is not necessary to collect CITs for Activity A because Activity A in the first sequence step does not have predecessors or idle time. Thus, the algorithm starts collecting CITs for the activity in Sequence Step 2 (SQS2), which is CITs for Activity B (CIT_B).

In the second step, after 3000 replications are executed, cumulative frequency of CIT_B is constructed using the collected CIT_B in the previous step. For example, Table 4.4 shows the samples of CIT_B collected from 3000 replications. The “Hit” Column shows the number of samples where CIT_B is less than the value in the “CIT” Column. The “% Hit” Column shows the percentage of CIT that is less than the value in the “CIT” Column.

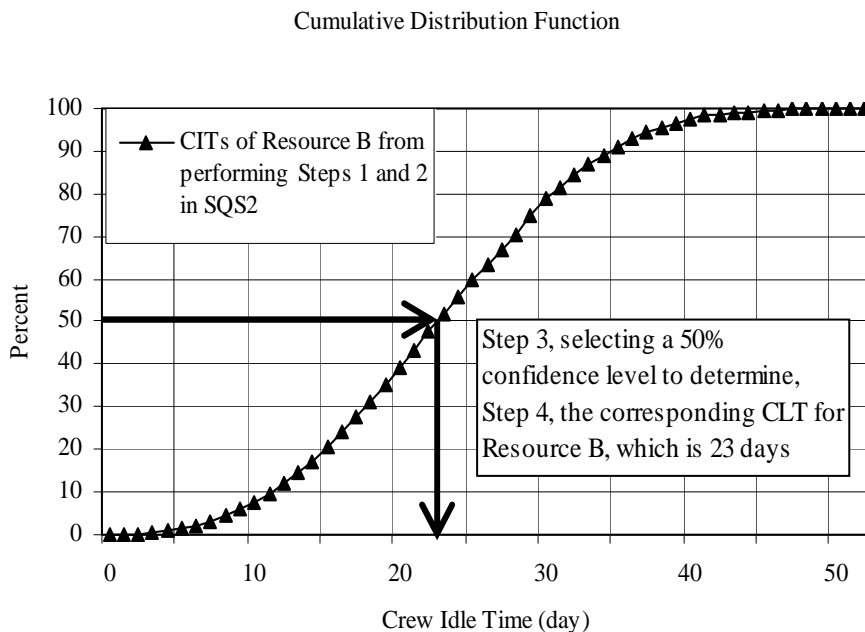


Figure 4.8 Determining crew idle time of Resource B in processing SQS2 in Example 4.3

CIT _B (Days)	Hit	%Hit	CIT _B (Days)	Hit	%Hit	CIT _B (Days)	Hit	%Hit
< 2	0	0	< 19	1059	35.3	< 36	2785	92.83
< 3	11	0.37	< 20	1179	39.3	< 37	2831	94.37
< 4	24	0.8	< 21	1302	43.4	< 38	2867	95.57
< 5	39	1.3	< 22	1426	47.53	< 39	2899	96.63
< 6	57	1.9	< 23	1553	51.77	< 40	2931	97.7
< 7	93	3.1	< 24	1680	56	< 41	2950	98.33
< 8	133	4.43	< 25	1790	59.67	< 42	2959	98.63
< 9	180	6	< 26	1893	63.1	< 43	2968	98.93
< 10	229	7.63	< 27	2006	66.87	< 44	2977	99.23
< 11	292	9.73	< 28	2111	70.37	< 45	2988	99.6
< 12	359	11.97	< 29	2248	74.93	< 46	2991	99.7
< 13	431	14.37	< 30	2360	78.67	< 47	2995	99.83
< 14	518	17.27	< 31	2448	81.6	< 48	2996	99.87
< 15	613	20.43	< 32	2535	84.5	< 49	2998	99.93
< 16	719	23.97	< 33	2609	86.97	< 50	2999	99.97
< 17	826	27.53	< 34	2671	89.03	< 51	2999	99.97
< 18	932	31.07	< 35	2735	91.17	< 52	3000	100

Table 4.4 Cumulative frequency of crew idle time of Resource B during processing SQS2 for Example 4.3

In the third step, CLT_B is selected from the constructed cumulative distribution function by using a user-specified confidence level. From Table 4.4 and Figure 4.8, if the user-specified confidence level is 50%, the corresponding CLT_B will be 23 days. Figure 4.9 shows CLTs of resources before and after processing SQS2. As explained earlier, SQS-AL assumes resources arrive to the site at the beginning of the project. These arrival dates at time zero are temporary arrival dates, which will be finalized after the sequence steps to which they belong are processed.

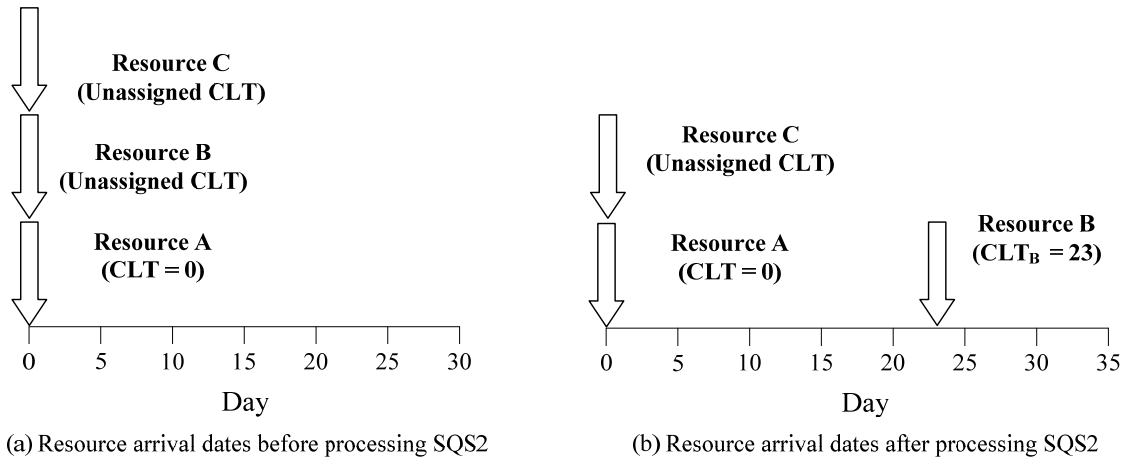


Figure 4.9 Changes in the resource arrival dates before and after processing SQS2 for Example 4.3

After determining CLT_B (the only activity in SQS2), SQS-AL moves onto the next sequence step, and repeats the three steps again, only this time to collect the crew idle time of Activity C (CIT_C).

For SQS3, the first step, collecting CITs, is repeated; another 3000 replications are executed. In the second step, the cumulative distribution function of CIT_C is constructed as shown in Table 4.5. Then, a 50% confidence level is selected in the third step to select the corresponding CLT_C of 31 days in the fourth step.

It is very important to notice that that the cumulative frequency distribution of CIT_C will be different if a different confidence level for Activity B is used.

CIT _C (days)	Hit	%Hit	CIT _C (days)	Hit	%Hit
23	0	0	42	2910	97
24	0	0	43	2924	97.47
25	140	4.67	44	2947	98.23
26	312	10.4	45	2962	98.73
27	519	17.3	46	2969	98.97
28	730	24.33	47	2974	99.13
29	1004	33.47	48	2985	99.5
30	1257	41.9	49	2990	99.67
31	1549	51.63	50	2993	99.77
32	1881	62.7	51	2993	99.77
33	2206	73.53	52	2994	99.8
34	2329	77.63	53	2997	99.9
35	2462	82.07	54	2998	99.93
36	2574	85.8	55	2998	99.93
37	2669	88.97	56	2998	99.93
38	2753	91.77	57	2998	99.93
39	2805	93.5	58	2998	99.93
40	2854	95.13	59	2999	99.97
41	2889	96.3	60	3000	100

Table 4.5 Cumulative frequency of crew idle time for Resource C during processing SQS3 for Example 4.3

Cumulative Distribution Function

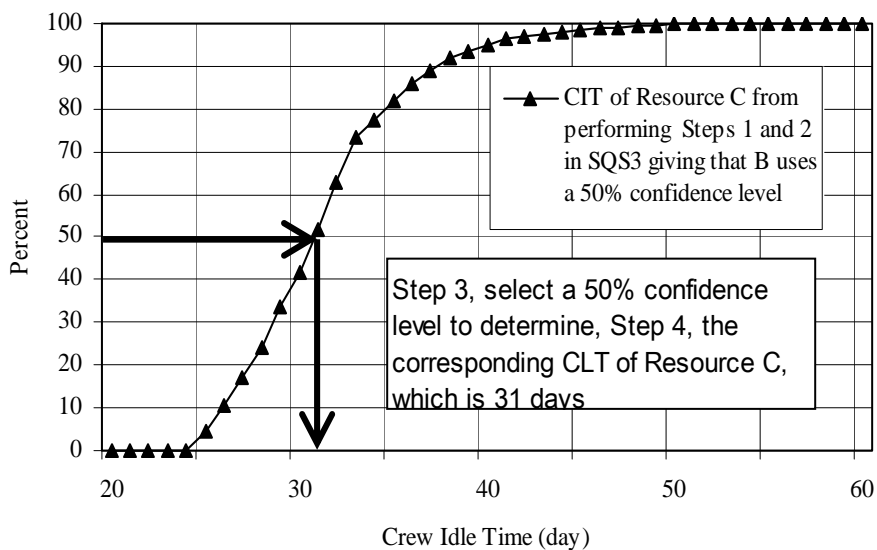


Figure 4.10 Determining crew idle time for Resource C during processing SQS3 for Example 4.3

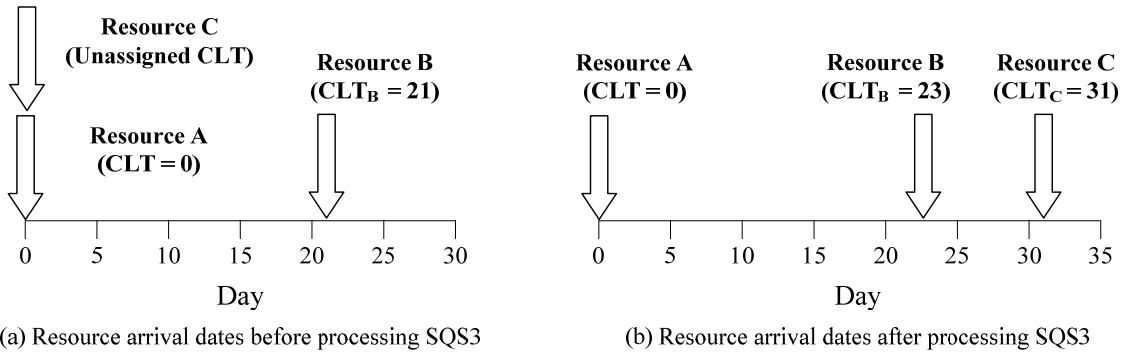


Figure 4.11 Changes in resource arrival dates before and after processing SQS3 for Example 4.3

After CLTs for all activities are determined, the finalized CITs are derived by running an additional 3000 replications (this is an extra SQS). In these replications, Resources A, B, and C are scheduled to the site at the end of Days 0, 23, and 31, respectively. The final results from SQS-AL are shown in Table 4.6.

SQS	Crew Lead Time (CLT) in days for a 50% Confidence Level			Sum of Idle Time Between Units (UIT) in days			Average Project Duration in days	Average Total Idle Time in days
	A	B	C	A	B	C		
1	0	0	0	0	23	25	50	47
2	0	0	0	0	23	25	50	47
3	0	23	0	0	3	31	56	34
4 (Final)	0	23	31	0	3	2	58	5

Table 4.6 Unit idle time, the selected Crew Lead Time, Average Project Duration, and Average Total Idle Time

As shown in Table 4.6, the average project duration is increased from 50 to 58 days; however, the average idle time significantly decreased from 47 to 5 days. The implication of the results from SQS-AL will be discussed in detail later.

4.5 Flow Chart of the Sequence Step Algorithm

The flow chart of SQS-AL is presented in Figure 4.12 showing SQS-AL's steps in collecting crew idle times (CITs) and determining crew lead time (CLT) in simulation. As displayed in the flow chart, the algorithm contains two nested loops. The inner loop is the replication loop (shown in Figure 4.13.a) whereas the outer loop is the sequence step loop (shown in Figure 4.13.b). According to the overview in Section 4.3, Step 1, collecting CITs, is in the replication loop, while Step 2, constructing cumulative frequency of CITs, Step 3, selecting confidence levels, and Step 4, determining CLT, are in the sequence step loop. Figure 4.12 presents the complete mechanism of SQS-AL including:

- 1) Initializing parameters (such as number of sequence steps)
- 2) Collecting simulation output (such as crew idle time)
- 3) Determining and updating parameters in the simulation model (such as crew lead time)
- 4) Proceeding from one sequence step to the next sequence step

Note that, some simulation parameters and components are introduced here with a brief explanation. They will be explained in more detail in the next chapter. In this section, understanding the complete process of SQS-AL is essential.

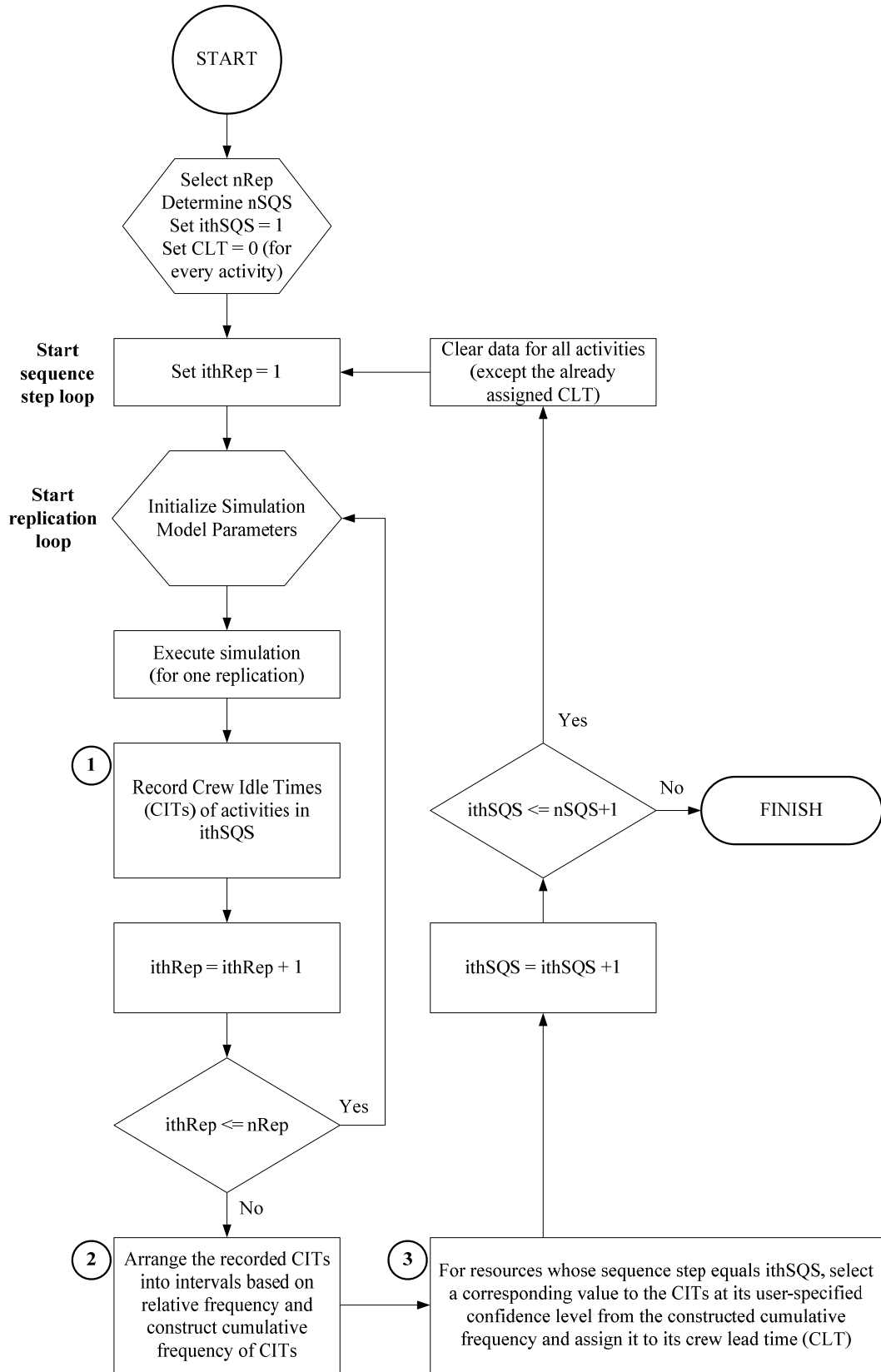
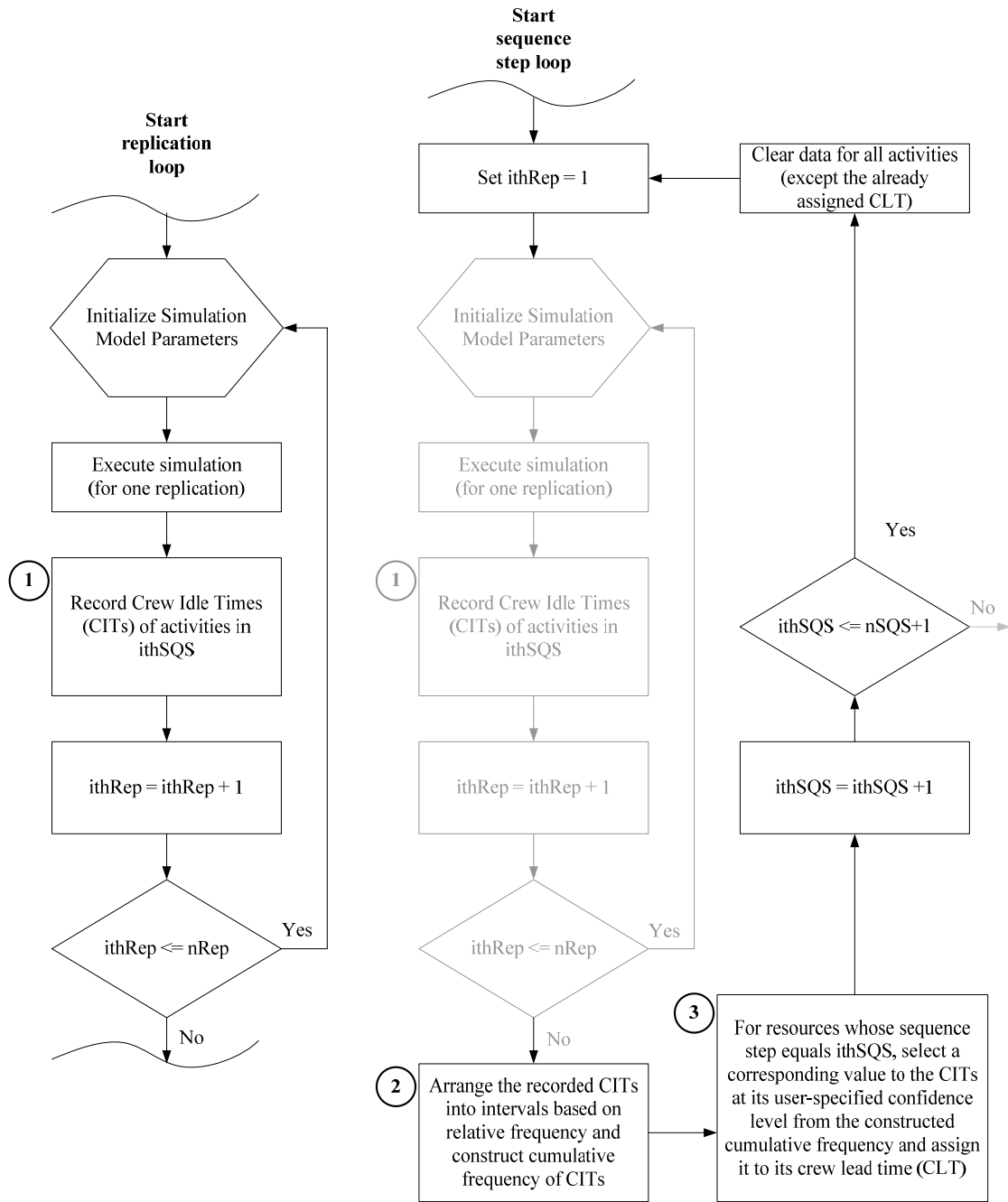


Figure 4.12 Flow Chart of the Sequence Step Algorithm



(a) SQS-AL's Replication Loop (inner loop)

(b) SQS-AL's Sequence Step Loop (outer loop)

Figure 4.13 Replication loop and sequence step loop in SQS-AL

In the initialization of SQS-AL parameters, the following parameters must be determined prior to simulation execution.

1. The total number of sequence step (nSQS).
2. Sequence step for each resource. Determining in which processing sequence steps that CITs of resources must be collected (e.g., CIT_B in SQS2 and CIT_C in SQS3 from Example 4.3).
3. The total number of replications (nRep), a constant value specified by users of how many simulation replications will be executed in order to collect the data set of CITs.
4. Current sequence step (ithSQS), an index indicating the current processing sequence step. In the beginning of SQS-AL, it is set to 0 (ithSQS=1), and increased by one after finishing processing each sequence step.
5. Crew lead times (CLT), the arrival dates for resources such as CLT_B , CLT_C , etc. In the beginning of SQS-AL, all CLTs are set to zero, assuming resources arrive to the site at the project start date.

After the initialization of SQS-AL parameters, SQS-AL sets the replications index (ithRep) to 1, which is also at the beginning of each replication loop, as shown in Figure 4.13. In the replication loop, for each simulation run, the simulation model parameters are initialized as shown in Figure 4.13 in a hexagon titled “Initialize Simulation Model Parameters.” This means that at the beginning of each simulation run the following parameters must be initialized. Note that the names for graphical simulation elements such as Queues, Combis, and SaveValues are in simulation model templates, which will

be discussed later in Chapter 5. These names are included here as references in this section and also between Chapters 4 and 5.

1. The number of repetitive units for each activity (the number of resources assigned in ACT_Remain Queues). The current value of resources in ACT_Remain Queues is the remaining work for that activity (ACT).
2. The number of resources for each type (the number of resources assigned in RES_Offsite Queues). This is the number of crews for a resource (RES) that can work simultaneously.
3. The temporary recorded idle time (svRES_Idle SaveValues)

Before starting a new replication (a simulation run), the number of repetitive units (i.e., the remaining work) in ACT_Remain Queues must be reinitialized. Otherwise, the project will be completed at time zero since all the works are completed from the previous simulation run; therefore there is no amount of work in ACT_Remain Queues.

The temporary idle times of resources (svRES_Idle) are recorded and used only within one replication. At the end of each replication (simulation run), svRES_Idle values for each resource are collected in a permanent storage collector, which is called RES_IdleSQS\$<ithSQS>\$, where “\$<ithSQS>\$” is the current processing step (“\$<...>\$” is Stroboscope’s language which will be explained in Chapter 5). Thus, idle time for each resource (e.g., Resource B) from each sequence step is recorded in RES_IdleSQSi (e.g., B_IdleSQS1, B_IdleSQS2, etc.) for the further calculation.

After simulation initialization, the simulation is executed for one replication. At the end of the execution, svRES_Idle values are assigned to their RES_IdleSQSi and to RES_CIT (crew idle time) of the resources that belong to the current processing step

(ithSQS). Then, ithRep is increased by 1. These processes (simulation initialization, simulation execution, and data collecting and updating) are repeated for nRep replications. As discussed in Section 4.4, Overview of the Sequence Step Algorithm, these processes, called the “replication loop,” are the first step in determining CLTs of resources, which is collecting CITs of the resources in the current sequence step.

After the replication loop is completed (ithRep > nRep), SQS-AL performs two main steps in order to determine CLTs of resources belonging to the current sequence step (ithSQS). These two steps are Steps 2 and 3, discussed in Section 4.4 constructing the cumulative distribution function of CITs and selecting the corresponding CLT. At this point, the crew idle times (CITs) for the resource belonging to the current sequence step (ithSQS) are already collected from nRep replications. Then these CITs are arranged into intervals based on relative frequency, and cumulative frequency of CITs for the resource is constructed. Based on user-specified confidence levels and the constructed cumulative frequency of CIT, crew lead time of the resource is selected and used as the arrival date for the resource.

After CLTs for resources in the current sequence step (ithSQS) are determined, SQS-AL moves to the next sequence step by, first, increasing ithSQS by one, and then goes back to the replication loop again. By repeating the replication loop and the sequence step loop, SQS-AL collects samples of crew idle times (CITs), affected by the already chosen crew lead time (CLTs) for all activities in the previous sequence steps. These CITs for the resources in the current sequence step are used to select their CLTs.

In the next section, another example is used to demonstrate the application of SQS-AL. Additionally, several important aspects of SQS-AL, such as collecting CITs and determining CLTs, are reviewed along with the example.

4.5.1 Example 4.4 Scheduling a repetitive project with 7 activities

An example of a repetitive project consisting of 4 units requiring 7 repetitive activities is used to demonstrate the application of the sequence step algorithm. Each activity is performed by a different crew. The precedence diagram for each of the repetitive units appears in Figure 4.14. As exhibited in Figure 4.14, Activity A is in Sequence Step 1 (SQS1); Activities B and C in SQS2, etc. The amount of work for each activity in each of the 4 units is different as displayed in Table 4.7. For example, the work amounts for Activity A in Units 1, 2, 3, and 4 are 100, 250, 150, and 200 work units, respectively.

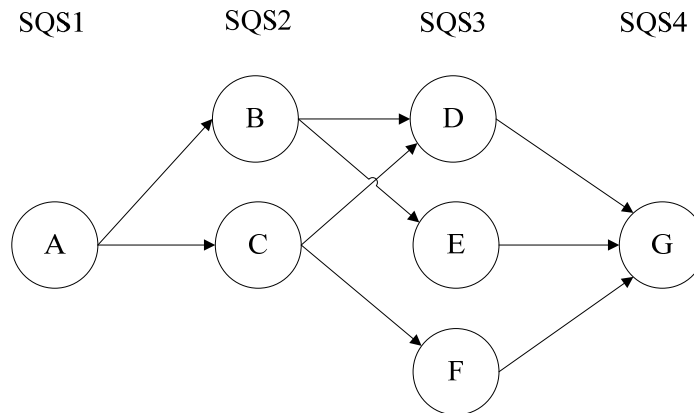


Figure 4.14 The single unit precedence diagram for Example 4.4

	Activity						
	A	B	C	D	E	F	G
Unit	Work Amount (Quantity/Unit)						
1	100	150	200	150	100	150	50
2	250	100	150	200	150	250	200
3	150	200	50	100	50	50	50
4	200	150	200	150	100	100	150

Table 4.7 Activities' work amounts in each unit for Example 4.4

In each of the 4 repetitive units, crew production rates (in work amounts per day) for each of the 7 activities are assumed to follow normal distributions with the means and standard deviations shown in Table 4.8. Consequently, the duration for each activity in each of the 4 repetitive units varies because of the different work amounts and the varying production rates.

Activity	Mean	SD
A	10	1.0
B	20	2.0
C	15	1.5
D	15	1.5
E	25	2.5
F	15	1.5
G	20	2.0

Table 4.8 Activities' production rates for Example 4.4

Figure 4.15 displays the first replication result from processing SQS2. At this point, none of the resources' CLTs are assigned to values other than zero. Therefore, activities start at their early start date and resources arrive at time zero. The result in Figure 4.15 derived from processing SQS2 is equivalent to a result from using simulation to schedule this repetitive project without incorporating any algorithm. Again, note that in processing SQS2, all resources have been scheduled to arrive at the site at the beginning of the project (time zero) for calculation purposes as explained in Sections 4.3 and 4.4.

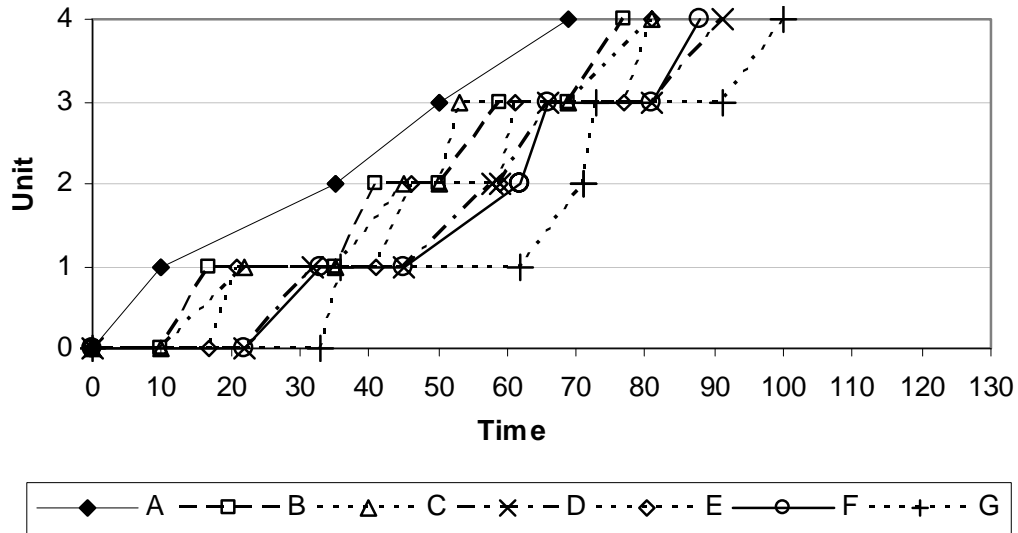


Figure 4.15 The first replication production diagram from processing SQS2 (collecting CIT_B and CIT_C) for Example 4.4

For this example, 10,000 replications were simulated within replication loop for each sequence step. Ten thousand data points of the crew idle times for Activities B and C each were collected by the end of processing SQS2. These 10,000 samples of CIT_B and CIT_C were then arranged by relative frequency in cumulative bins using an interval of 5 time units (arbitrarily selected by the modeler) as shown in Table 4.9. Each row of this table shows a time value, and the percent of the 10,000 crew idle times for activities. For example, out of a total of 10,000 CIT_B samples, 9,427 of them are less than 55 days. Thus, if Activity B is scheduled to start at the end of Day 55 ($CLT_B = 55$ days), the probability of continuous resource utilization for Activity B will be 94.27%. On the other hand, if CLT_B is set to 75, then, continuous resource utilization would be 100%.

Notice that even though the CITs for all activities are shown in Table 4.9, only those for Activities B and C, which are in SQS2, will be used since the current processing sequence step is SQS2. This is why the columns for Activities B and C appear in bold outline.

CIT _B		CIT _C		CIT _D	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 35	0.01	< 25	0.01	< 35	0.00
< 40	1.26	< 30	0.03	< 40	0.11
< 45	19.37	< 35	0.09	< 45	1.57
< 50	65.92	< 40	15.33	< 50	16.93
< 55	94.27	< 45	57.92	< 55	57.04
< 60	99.40	< 50	90.67	< 60	90.10
< 65	99.95	< 55	98.94	< 65	98.66
< 70	99.99	< 60	99.92	< 70	99.88
< 75	100.00	< 65	100.00	< 75	99.98

CIT _E		CIT _F		CIT _G	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 50	0.00	< 40	0.01	< 65	0.01
< 55	0.04	< 45	1.59	< 70	1.26
< 60	3.85	< 50	17.11	< 75	14.22
< 65	35.64	< 55	56.31	< 80	54.52
< 70	80.39	< 60	89.84	< 85	88.30
< 75	97.19	< 65	98.43	< 90	98.38
< 80	99.77	< 70	99.86	< 95	99.89
< 85	99.98	< 75	99.97	< 100	99.99
< 90	100.00	< 80	100.00	< 105	100.00

Table 4.9 Collected CITs from processing SQS2 and determining CLT for Activities B and C for Example 4.4

For this example project, an 80% confidence level has been selected to determine the crew lead times for various resources. Hence, CLT_B is set to 55 days (the first value in Table 4.9 that exceeds 80%). Thus, Activity B cannot start sooner than the end of Day 55, because its resource will not arrive to the site until the end of Day 55. Similarly, at an 80% confidence level, the duration for CLT_C is set to 50 days. Once CLT_B and CLT_C are set to 55 and 50 days, respectively, they are not changed again. Now that the CLTs for the resources utilized by activities SQS2 are determined, SQS-AL moves to the next sequence step, SQS3.

In SQS3, the algorithm collects 10,000 samples of crew idle times (CIT) for resources whose activities are in SQS3; these activities are D, E, and F. Therefore, CIT_D , CIT_E , and CIT_F are collected during processing SQS3. It is important to realize that the collected CITs for Activities D, E, and F reflect:

- 1) The effect of the chosen CLT_B (55 days) and CLT_C (50 days) from the previous processing SQS2.
- 2) The variability of activity durations (A, B, C, D, E, and F).

Figure 4.16 shows the schedule resulting from the first replication in processing SQS3. As shown in Figure 4.16, Activities B and C no longer start at their earliest start date, since their resources have been deliberately delayed by 55 and 50 days from the beginning of the project start date, respectively ($CIT_B = 55$ days and $CIT_C = 50$ days).

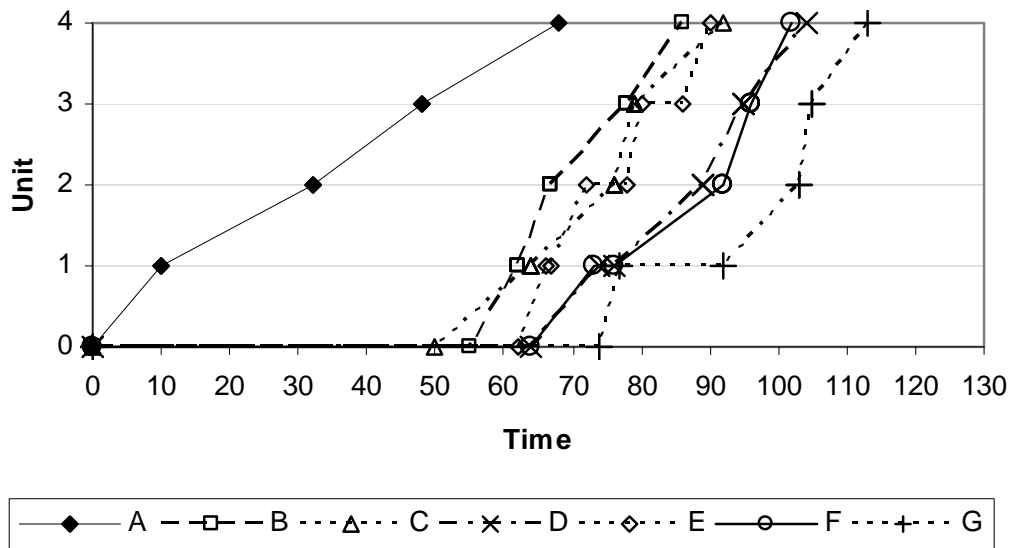


Figure 4.16 The first replication production diagram from processing SQS3 (collecting CIT_D , CIT_E , and CIT_F) for Example 4.4

Executing 10,000 replications in SQS3, the algorithm collects 10,000 samples of each CIT_D , CIT_E , and CIT_F , and then organizes them in cumulative frequency as shown in Table 4.10. At the end of processing SQS3, CLT_D , CLT_E , and CLT_F are set to 70, 80,

and 70 days, respectively according to an 80% confidence level. This means Activities D, E, and F cannot start until the end of days 70, 80, and 70 due to resource availability constraints and the selected crew lead times.

CIT _B		CIT _C		CIT _D	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 0	0.00	< 0	0.00	< 60	0.00
< 5	99.48	< 5	98.92	< 65	62.14
< 10	99.97	< 10	99.95	< 70	99.50
< 15	100.00	< 15	100.00	< 75	99.99
				< 80	100.00

CIT _E		CIT _F		CIT _G	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 65	0.00	< 55	0.00	< 80	0.00
< 70	1.84	< 60	0.05	< 85	0.31
< 75	74.66	< 65	63.24	< 90	38.85
< 80	99.67	< 70	99.39	< 95	94.31
< 85	99.99	< 75	99.99	< 100	94.89
< 90	100.00	< 80	100.00	< 105	99.99
				< 110	100.00

Table 4.10 Collected CITs from processing SQS3 and determining CLTs for Activities D, E, and F for Example 4.4

Compared to Table 4.9, CIT_B and CIT_C in Table 4.10 decrease significantly due to the assigned CLT_B and CLT_C in the previous processing SQS2. For both Resources B and C, there is approximately 99% chance that their idle time will be less than 5 days, shown in Table 4.10, compared to 55 days in Table 4.9.

It is important to note that if CLT_D, CLT_E, and CLT_F were assigned at the end of processing SQS2, their values would be 60, 70, and 60 days (from Table 4.9, processing SQS2), instead of 70, 80, and 70 days (from Table 4.10, processing SQS3). Consequently, their idle time would be most likely to exceed the corresponding idle time

to the user-specified confidence level of 80%. As explained, delaying activities at the end of processing a SQS is likely to change the CITs of resources in the subsequent SQS. Therefore, CITs and CLT for each resource must be collected and determined in the order of SQS in which their activities belong. Figure 4.18 presents the first replication production diagram from processing SQS4, after CLT_D , CLT_E , and CLT_F were assigned.

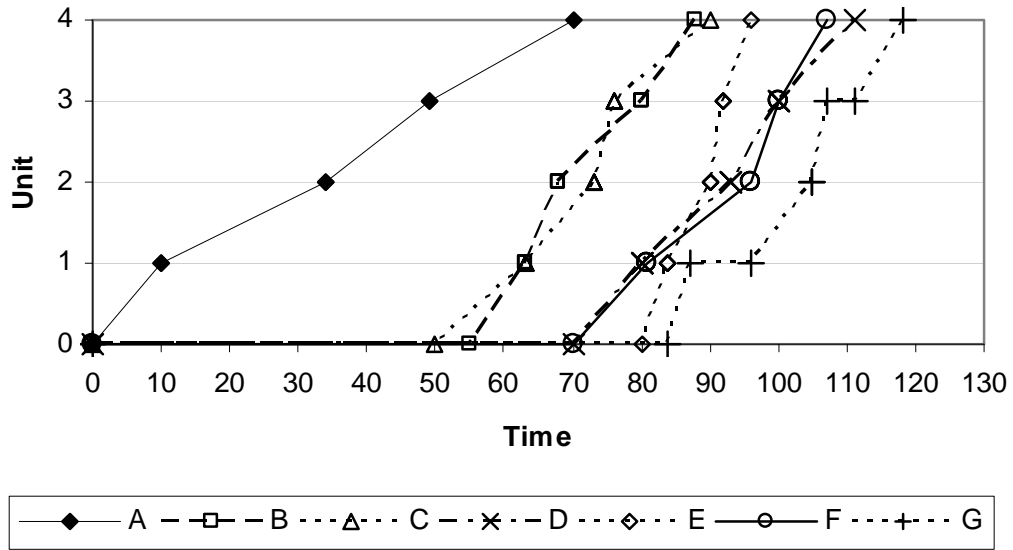


Figure 4.17 The first replication production diagram from processing SQS4 (collecting CIT_G) for Example 4.4

In Figure 4.17, there is no interruption of work for Resources B, C, D, E, and F (at least for this replication). Activities whose CLT is already assigned have idle time close to zero. The already-assigned-CLT resources work continuously from the day they arrive to the day they complete their work. At this current SQS4, only Activity G has idle time (between G1 and G2 in Figure 4.17), since CIT_G and CLT_G have not yet been collected and determined. Table 4.11 shows the current CITs of resources collected from processing SQS4.

CIT _B		CIT _C		CIT _D	
Range	% Frequency	Range	% Frequency	Range	% Frequency
< 0	0.00	< 0	0.00	< 5	0.00
< 5	99.52	< 5	99.06	< 10	100.00
< 10	99.98	< 10	99.92		
< 15	100.00	< 15	100.00		

CIT _E		CIT _F		CIT _G	
Range	% Frequency	Range	% Frequency	Range	% Frequency
< 0	0.00	< 0	0.00	< 90	0.00
< 5	100.00	< 5	100.00	< 95	21.44
				< 100	93.09
				< 105	99.92
				< 110	100.00

Table 4.11 Collecting CITs from processing SQS4 and determining CLT for Activity G for Example 4.4

Table 4.11 shows the CITs of activities after the algorithm finishes processing SQS4. At the end of processing SQS4, samples of CIT_G are collected and arranged as shown in Table 4.11. CLT_G is set to 100 days according to the user-specified 80% confidence level. Thus, CLTs for all activities are determined. Then, an extra sequence step is processed to collect statistics of the finalized CITs, as shown in Table 4.12.

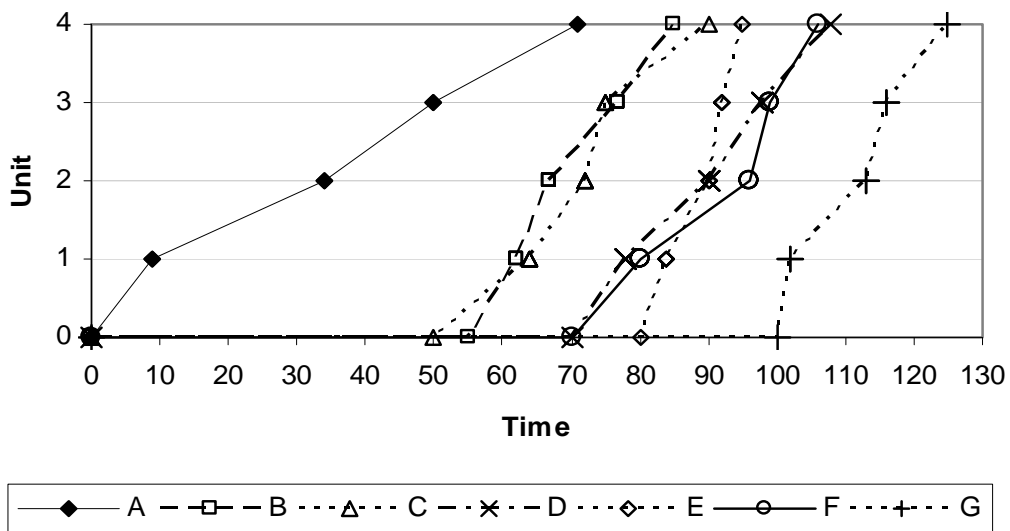


Figure 4.18 The first replication production diagram from processing SQS5 for Example 4.4

CIT _B		CIT _C		CIT _D	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 0	0.00	< 0	0.00	< 5	0.00
< 5	99.39	< 5	99.11	< 10	100.00
< 10	99.95	< 10	99.94		
< 15	100.00	< 15	100.00		

CIT _E		CIT _F		CIT _G	
Range (days)	Frequency (%)	Range (days)	Frequency (%)	Range (days)	Frequency (%)
< 0	0.00	< 0	0.00	< 0	0.00
< 5	100.00	< 5	100.00	< 5	99.90
				< 10	100.00

Table 4.12 The finalized CITs from processing SQS5 for Example 4.4

Table 4.13 shows the assigned crew lead time and the average crew idle time (average CIT from 10,000 samples in each sequence step) each resource spends on the site from the arrival date. Notice that before processing, SQS2, Resources B and C arrive at the site at time 0 (CLT = 0) and thus have average total idle times of 48 and 44 days respectively. After processing SQS2, resources B and C are assigned to arrive at the end of Day 55 (CLT_B) and 50 (CLT_C), respectively, and thus have zero average idle time from then on. As the algorithm proceeds through sequence steps, crew idle time of activities become 0 in the sequence step order (1 to 5) as indicated in Table 4.13.

SQS	Assigned Crew Lead Time (CLT in days)						Average Crew Idle Time (CIT in days)						
	B	C	D	E	F	G	A	B	C	D	E	F	G
1	0	0	0	0	0	0	0	48	44	54	66	54	79
2	0	0	0	0	0	0	0	48	44	54	66	54	79
3	55	50	0	0	0	0	0	0	0	64	73	64	90
4	55	50	70	80	70	0	0	0	0	0	0	0	96
5	55	50	70	80	70	100	0	0	0	0	0	0	0

Table 4.13 Assigned CLT and average CIT of activities from processing different sequence steps for Example 4.4

SQS	Sum of Lags Between Units (UIT in days)						Average Project Duration	Average Project Idle Time
	B	C	D	E	F	G		
1	38	34	30	48	30	45	102	225
2	38	34	30	48	30	45	102	225
3	0	0	1	11	1	16	113	29
4	0	0	0	0	0	12	119	12
5	0	0	0	0	0	0	123	0

Table 4.14 The finalized UIT, average project duration, and average project idle time for Example 4.4

4.6 Discussion of Results from the Sequence Step Algorithm

Table 4.13 shows the average crew idle time (CIT) of each crew during its employment period. Table 4.14, on the other hand, shows the average idle time between units (UIT). The difference is that CIT values in Table 4.13 include not just, but also the idle time from resource arrival dates to the start date of the first unit, which is the arrival idle time (AIT).

It is interesting to note that in Table 4.13, the average CIT for resources drops to zero once their crew lead times are assigned. As some crew lead times are assigned from one SQS to the next, the average CIT for succeeding activities increases. In contrast, in Table 4.14 the average idle time between units (UIT) actually decreases. This is as expected, because the assignment of crew lead times to predecessor activities delays the first start of successor activities further (increasing their AIT), but also reduces idle time between units (decreasing their UIT).

Table 4.14 shows that, as the sequence step algorithm proceeds, the average project duration increases from 102 days to 123 days. The corresponding cumulative distributions for project duration for each sequence step in the algorithm are shown in Figure 4.19. The greatest horizontal time shift in project duration between successive

distributions occurs between SQS2 and SQS3 (Figure 4.19), which gives an increase in average project duration from 102 to 113 days (Table 4.14). The reason for this can be observed in Figure 4.15. Activity A, the slowest activity in the project, contributes the most to discontinuities in the other activities. Thus, to eliminate the discontinuities, it is necessary to shift their resource arrival dates significantly, which in turn increases project duration the most. Figure 4.16 shows that once the crew lead times for Activities B and C have been assigned, the rest of the activities (except G) have almost no work interruption between units, at least in the first replication of the project.

Figure 4.17 shows the result from postponing the start dates of activities in SQS3 (D, E, and F) by their CLTs. As expected, the continuities in the activities are improved while the average project duration increases from 113 days to 119 days.

Finally, crew lead time for Activity G in the last sequence step (SQS5) is assigned ($CLT_G=100$). As shown in Figure 4.18, the continuities in resource utilization of all activities are now maintained, at least in the first replication of SQS5. The final average project duration is 123 days (Table 4.14).

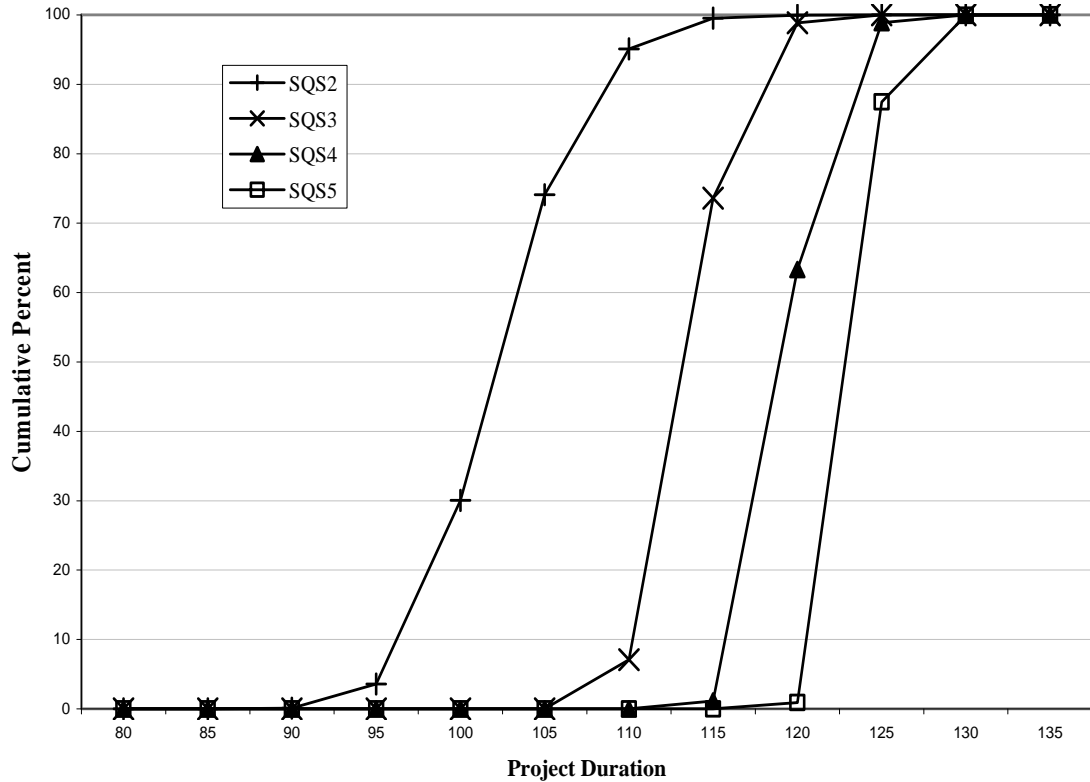


Figure 4.19 Cumulative distributions of project duration at an 80% confidence level

Insightful information of what activity in which sequence steps controls project duration can be revealed by:

- 1) Indicating the greatest shift in the average project duration from one sequence step to the next (SQS2 in Figure 4.19)
- 2) Indicating activities in the sequence step in (1), which are Activities B and C
- 3) Indicating controlling sequence activities in the sequence step in (1) from the final schedule, which is Activity C, and its direct predecessor on the same sequence step, which is Activity A
- 4) Studying production rates of activities in the sequence steps in (3)
- 5) Studying crew idle time of resources before and after assigning crew lead time

Then, the decision of which activities to accelerate could be made with a selected confidence level of resource continuity in order to reduce the project duration.

As shown in Figure 4.20, it is interesting that as the algorithm proceeds from one sequence step to another, most samples of project duration fit into a smaller number of intervals: 6 intervals in SQS2, 4 intervals in SQS3, 3 intervals in SQS4, and 2 intervals in SQS5. The variability in activity duration has become less impacting on project duration because successor activities have been postponed and result in more floats for predecessor activities. Consequently, project duration is only influenced by those activities in the later sequence steps (e.g., SQS3 and SQS4). For this example at an 80% confidence level, project duration is dominated by Activities F1 and F2 in SQS3 and G1 to G4 in SQS4.

4.7 Selection of Confidence Levels

An important question in the optimal use of the sequence step algorithm is how to select an appropriate confidence level for the occurrence of crew work interruptions to balance the increase in project duration. This is an important issue because high confidence levels (to virtually eliminate idle time) can lead to a significant increase in project duration.

To address this issue, Figure 4.20 shows 5 lines that relate average total crew idle time (total CIT in crew-days) to average project duration (in days). Each line corresponds to a different confidence level: 20%, 40%, 60%, 80%, and 100%. Moreover, each line consists of four points that correspond, from left to right (or top to bottom), to the four sequence steps (SQS2 to SQS5). Thus, the expected total idle times between units decrease as the algorithm proceeds from one sequence step to the next. Clearly, selecting

a greater confidence level decreases the expected crew idle time, but it also increases the expected project duration. Thus, allowing the possibility of some interruption can decrease the project duration. The idea of allowing or scheduling interruption is discussed in Chapter 6, Work Breaks.

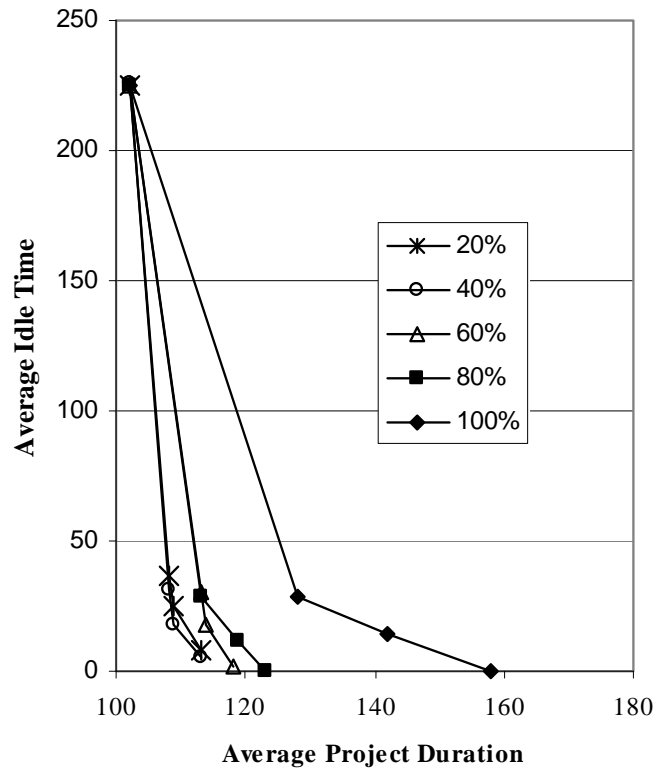


Figure 4.20 Decreasing idle time and increasing project duration as SQS-AL progresses with 5 different confidence levels for Example 4.4

For all confidence levels in Figure 4.20, the most dramatic reduction in average total crew idle time occurs between SQS2 and SQS3 from the first point to the second point of each line. From that point on, the reduction in average idle time from SQS3 to SQS4 is relatively small, and results in even higher increases in average project duration as the confidence level increases. It is only the case for a confidence level of 100% that the project duration increases significantly from processing SQS3 to SQS4, and from SQS4 to SQS5.

It is very important to notice that irrespective of confidence levels, the average total idle times at the completion of the algorithm (bottom point in each line) are very small. In particular, the expected idle time for confidence levels of 60%, 80%, and 100% is practically zero. Yet, the expected project durations for confidence levels of 60%, 80%, and 100% increase from 118 to 123 to 158 days. It is likely that an optimal confidence level is between 40% and 80%.

Figure 4.21 shows the corresponding probability density functions for project duration for different confidence levels. It is confirmed in this figure that there is little difference between confidence levels of 60% and 80%, but there is substantial difference between 80% and 100%. Figure 4.22 shows the cumulative distributions of project duration when the project is scheduled using CPM, RSM, and SQS-AL at different confidence levels. For the CPM case, activities in each replication are allowed to start as early as their predecessors allow. Thus, project duration from CPM tends to be the shortest with a large span of distribution. For the RSM case, activities in each replication are scheduled with perfect hindsight so as to eliminate crew idle time. This produces a slight increase in expected project duration over the CPM case, but completely eliminates idle time. The difference between the RSM expected project duration and that for an 80% confidence level of SQS-AL represents the value of perfect information about the true activity durations that will be experienced during construction. If these were known ahead of time or were effectively controlled, then the project could be scheduled with even shorter crew lead times and have even shorter project duration.

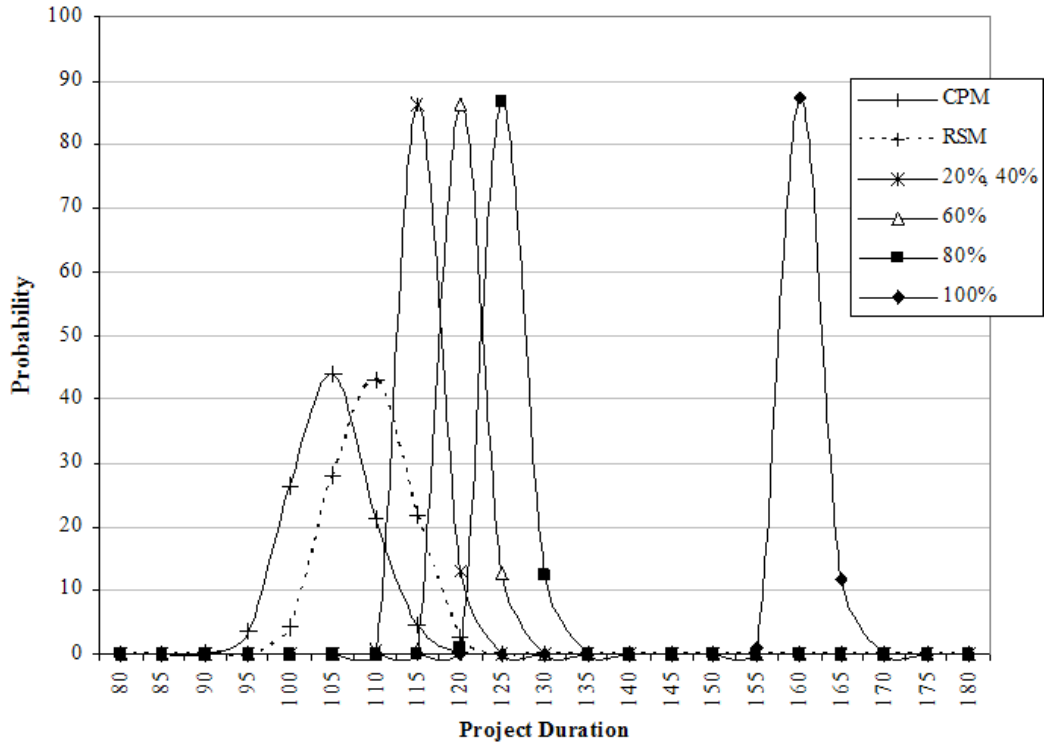


Figure 4.21 Seven density functions of project duration for 5 different confidence levels, CPM, and RSM

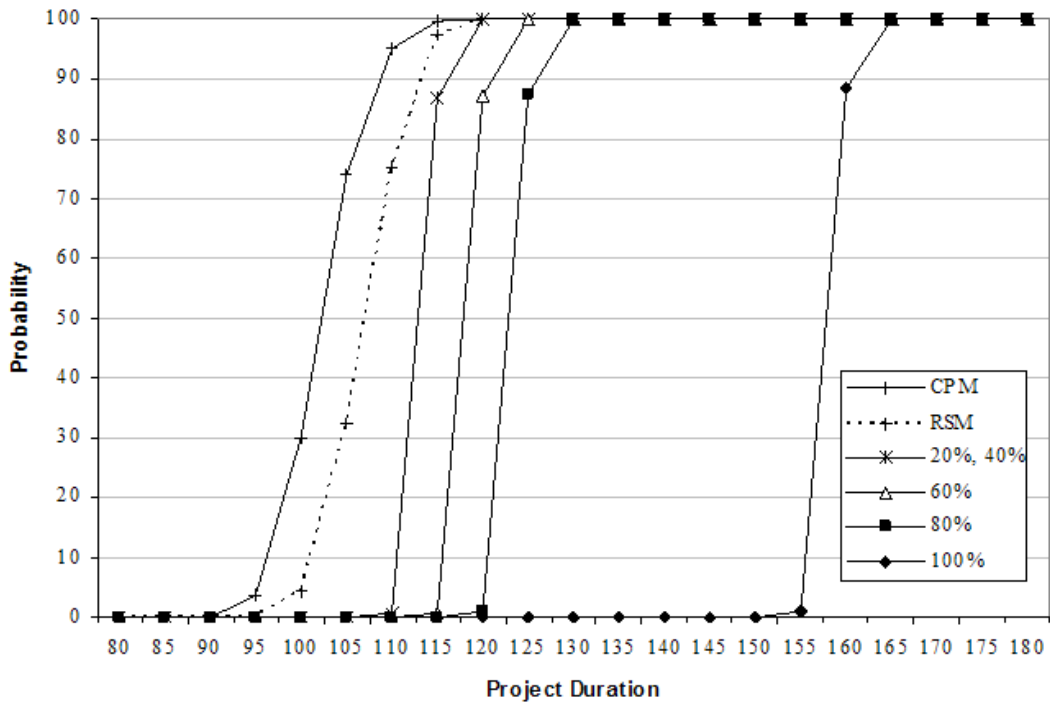


Figure 4.22 Seven different cumulative distributions of project duration from five different confidence levels, CPM, and RSM

4.8 Summary

The Sequence Step Algorithm (SQS-AL) is a generalized algorithm that solves scheduling problems of probabilistic repetitive projects. It can be implemented in discrete-event simulation systems, programming environments such as Visual Studio dot Net, or in spreadsheet applications like MS Excel.

SQS-AL consists of two nested loops: the replication loop (inner loop) and the sequence step loop (outer loop). Within the inner replication loop, three different types of idle time are collected in order to determine arrival dates of resources that provide continuous resource utilization for each resource. These three types of idle time are:

- 1) Unit idle time (UIT), which is the idle time between units in the same activity.
- 2) Arrival idle time (AIT), which is the idle time between resource arrival date and the start date in the first unit.
- 3) Crew idle time, which is the sum of UIT and AIT for each resource, representing total idle time for that resource.

Crew idle times (CITs) for each resource are collected from simulation replications in order to construct the cumulative distribution function of the CITs. Then, resource arrival date can be determined by selecting the corresponding CITs to user-specified confidence levels. This determined resource arrival date is called “Crew Lead Time” (CLT) measured from project start date. CLT is used to postpone the start date of activities (the arrival date of their resources) to achieve continuous resource utilization via resource availability constraints.

When to determine crew lead time (CLT) is crucial. CLT of a resource must be determined in the order of sequence steps, because delaying the arrival date of resources

changes the crew idle time (CIT) for activities in subsequent sequence steps. Accordingly, the processes of collecting CITs and determining CLT must be performed in sequence step order. These processes, called “processing sequence step,” moves from one sequence step to the next sequence step. This is performed in the outer sequence step loop. When the algorithm completes processing the last sequence step, CLTs for all resources are determined. Then, an extra processing sequence step is executed in order to obtain the final schedule and idle time in resources.

In the next chapter, two simulation model templates are introduced. These simulation templates are a composed set of simulation model elements used to represent repetitive activities and their resources with respect to SQS-AL. The templates can be implemented in most discrete-event simulation applications such as Stroboscope and GPSS. For the purpose of demonstration, the repetitive project example used in this chapter is modeled and implemented in Stroboscope using the simulation model templates.

CHAPTER 5

SIMULATION MODEL TEMPLATES

Simulation modeling for repetitive projects is complicated and involves many components. To model repetitive projects successfully, the following components must be taken into account when constructing the simulation model.

- 1) Precedence constraints
- 2) Variability in activity durations
- 3) Resource availability constraints
- 4) Resource continuity constraints, or computational algorithm solving the problem of resource continuity constraints
- 5) Collection of statistical outputs from simulation, such as idle time

This chapter introduces simulation model templates used to model repetitive projects in compliance with discrete-event simulation. These templates and their concepts can be applied to most discrete-event simulation systems. Using the templates results in an organized simulation model and also eases and expedites the processes of constructing and modifying simulation models for repetitive projects. Considering the aforementioned components of modeling repetitive projects, two simulation model templates, 1) Work Flow template and 2) Resource Flow template, are designed to systematically model the constraints and realistically capture the nature of activities and resources in repetitive

projects. The work flow template is for modeling precedence constraints, variability in activity duration, and work in repetitive units. On the other hand, the resource flow template is for modeling resource availability constraints and collecting data related to resource continuity constraints. Collecting resource-related data, the resource flow provides the data to the Sequence Step Algorithm (SQS-AL) in order to solve scheduling problems of repetitive projects.

To demonstrate the application of the work flow and resource flow templates, Example 4.2 from the previous chapter is modeled in Stroboscope. The simulation model and code are shown in detail with discussion of the usability, flexibility, and extensibility of the templates.

5.1 Simulation Model for Repetitive Projects

One benefit of simulation is the capability of modeling repetitive activities using only one or several simulation elements to represent the entire number of repetitive units for the activities. Different units of the same activity are simulated repeatedly by the same simulation elements. Therefore, repetitive projects consisting of multiple units can be represented by a single unit model. For example, the precedence diagram shown in Figure 5.1 can be replaced by the model in Figure 5.2. As can be seen, the greater the number of repetitive units, the larger the precedence diagram in Figure 5.1 becomes. On the other hand, the size of the single-unit model in Figure 5.2 is not subject to the number of repetitive units.

Another advantage of using a single unit model is that it facilitates the processes of constructing and modifying a simulation model. Since multiple units of the same repetitive activity commonly share the same attributes such as productivity, work

condition, resources, they can be represented by a single unit model. Accordingly, it is recommended to model repetitive projects using single unit model as shown in Figure 5.2, instead of Figure 5.1.

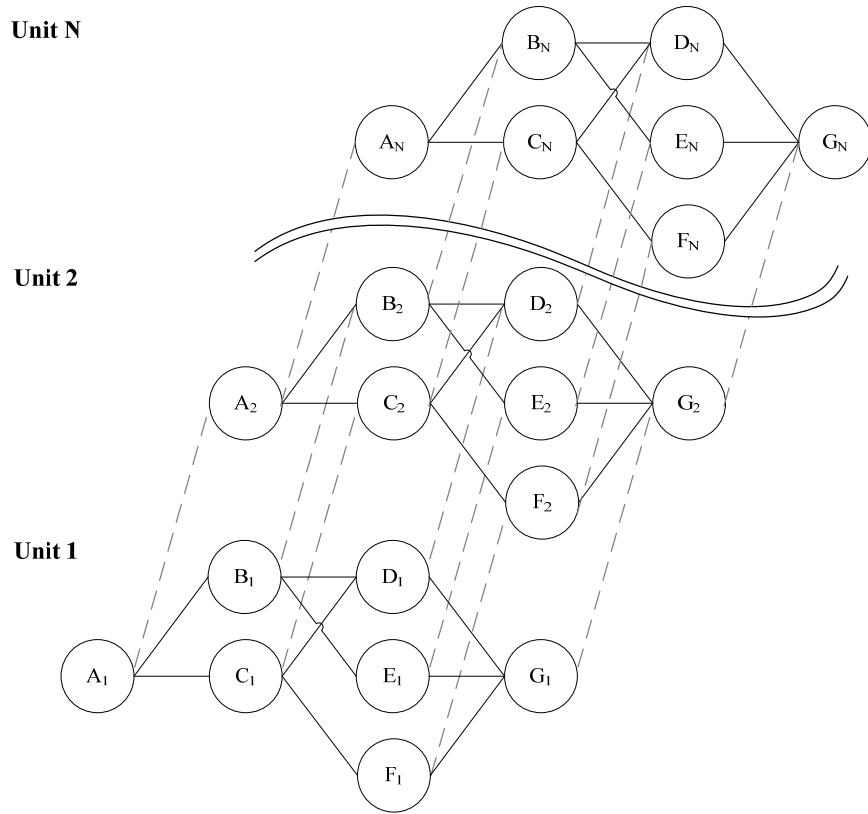


Figure 5.1 A precedence diagram for a repetitive project

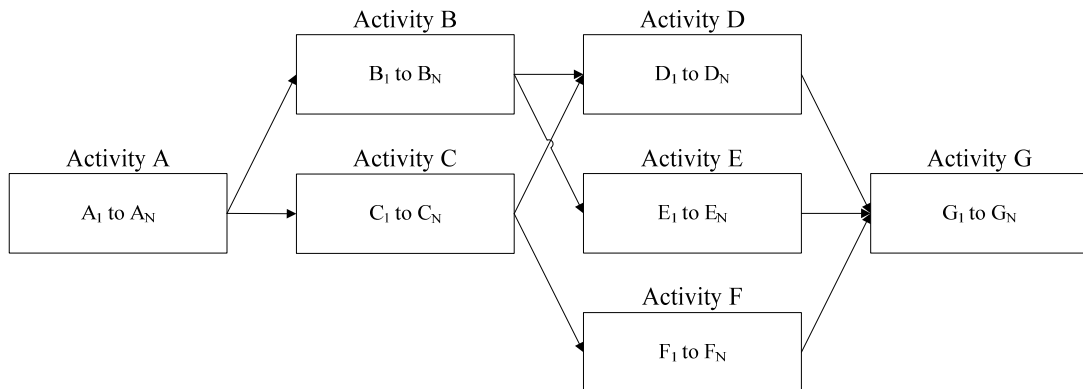


Figure 5.2 A single unit precedence diagram for the repetitive project in Figure 5.1

As shown in Figure 5.2, repetitive activities are represented effectively by a single unit model. Nevertheless, the model in Figure 5.2 only focuses on activities; information or model of resources is missing in the figure. When resource utilization is a concern, resources must be taken into account for scheduling repetitive projects. As mentioned earlier, different resource statuses must be distinguishable, and a model for resources should promptly provide this information. Hence, separating a model for resources from the model for activities is recommended. By doing so, tracking the progress of activities and distinguishing resource statuses should be modeled and coded by two separate sets, but dependent (or connected) models. Figure 5.3 shows a single unit precedence diagram with separate models for resources and work for each activity. In the figure, rounded rectangles of work flow are used primarily for tracking work progress where as rounded rectangles of resource flow are used primarily for distinguishing the status of resources. As shown in Figure 5.4, the simulation models for resources can be removed from the simulation models for activities.

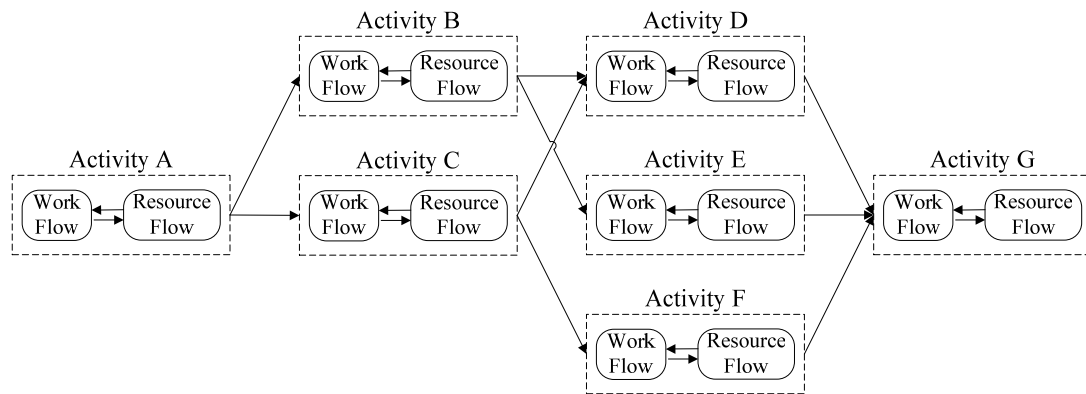


Figure 5.3 Models for activities and resources in a single unit precedence diagram

After the simulation models for activities and resources are separated into two sets: 1) work flow and 2) resource flow, constraints belonging to each set of the

simulation model (activities and resources) can become separate and distinct. Precedence constraints are modeled in work flow sub-networks, whereas resource availability and continuous constraints are modeled in resource flow sub-networks. As a result, the simulation model becomes more organized and systematic.

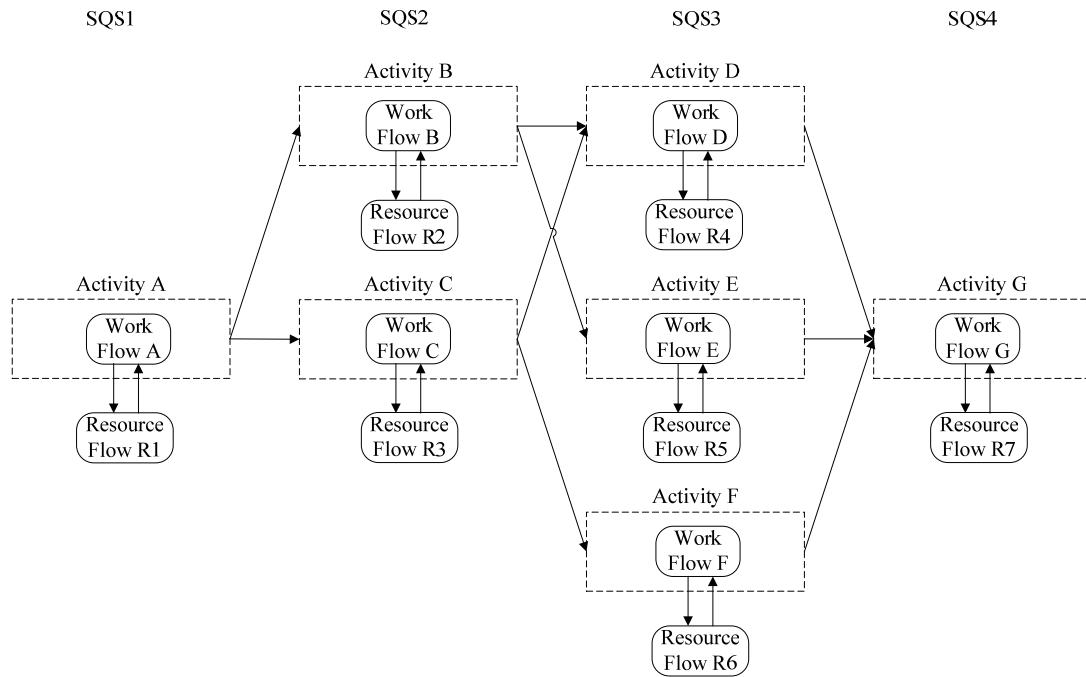


Figure 5.4 Separate models for activities and resources in a single unit precedence diagram

To illustrate the possibilities, Figure 5.5 shows an example of a repetitive project where same activities within the same repetitive unit share the same resource (e.g., Activity B and C) or require more than one resource. Using separate models for activities and resources allows the construction of complicated relationships between repetitive activities and resources easily. For example, in Figure 5.5, Activities B and C (work flow sub-networks for B and C) require the same Resource R2. Activity D requires resource R3 and shares Resource R6 with Activity G. Modeling multiple resource utilization and sharing resources, called “shared resource,” between activities are discussed later in

Chapter 7. At this point, the following discussion will concentrate on models where each repetitive activity requires its own unique resource (or crew), called “dedicated resource.”

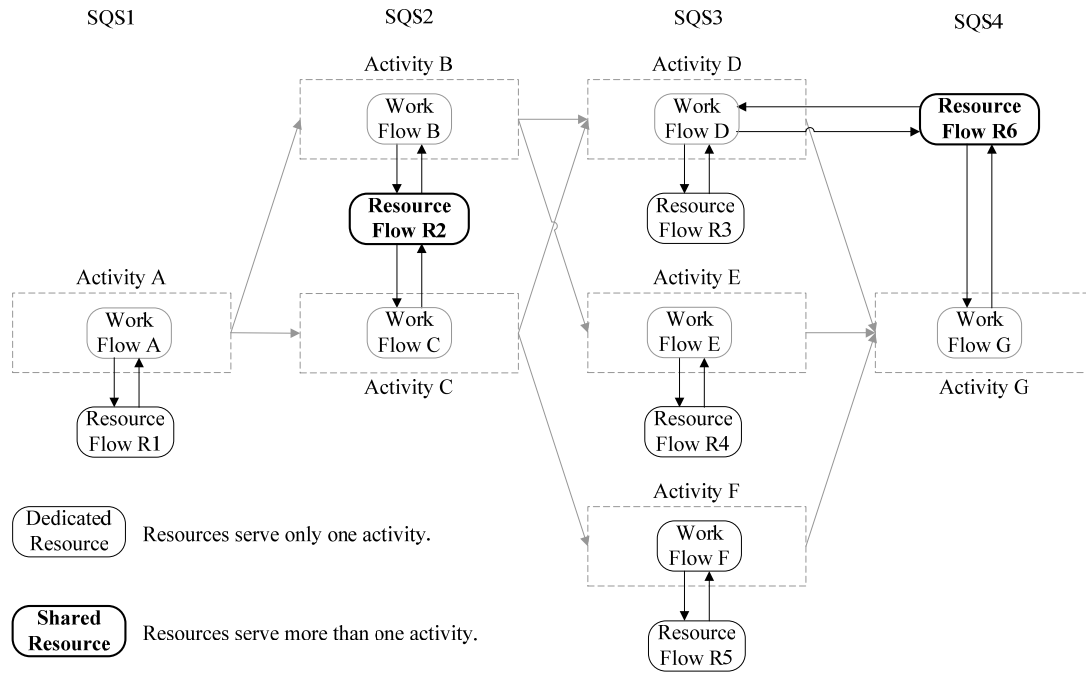


Figure 5.5 Using separate models to model a repetitive project with resource-sharing activities

5.2 Simulation Model Templates

To apply SQS-AL, a simulation model must be able to track work progress, and to distinguish the different states of the resources. It is important to track the current progress of repetitive activities so that precedence constraints between activities and units are not violated. Similarly, distinguishing different statuses of resources provides useful information of resource utilization, which is used to determine crew lead time (CLT). The different states of resources are:

- 1) Unemployment period (not on jobsite, not getting paid)
- 2) Employment period (on jobsite, getting paid)

- 3) Unproductive period (on jobsite, getting paid, but not working, which is idle)
- 4) Productive period (on jobsite, getting paid, working)

The above different states are represented in the work flow template and resource flow template, as explained later. In Figure 5.6, the work flow template and the resource flow template represent Activity ACT and Resource RES, respectively. The self-explanatory labels on simulation elements (Queues, Combis, etc) show that the work flow template tracks work progress via ACT_Remain and ACT_Complete Queues. Activity duration is simulated by ACT_Perform Combi. On the other hand, the resource flow template distinguishes resource status via RES_Idle and RES_Offsite Queues. Duration of crew lead time (CLT) is represented by RES_CLT Combi.

In Figure 5.6, Activity ACT requires Resource RES; thus, the resource flow sub-networks for RES is linked to the work flow sub-networks by iRES_ACT and iiRES_ACT Links. Figure 5.6 shows a simple relationship between one activity and one resource, where the resource is dedicated to only one activity.

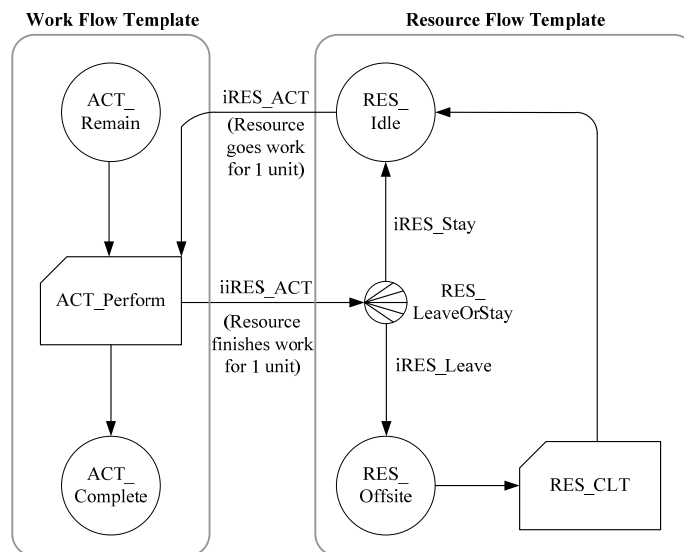


Figure 5.6 Work Flow Template and Resource Flow Template

5.2.1 Work Flow Template

The main functionalities of the work flow template are:

- 1) Tracking work progress of repetitive activities
- 2) Modeling precedence constraints
- 3) Simulating activity durations.

The work flow template can be modeled and coded using two queues (ACT_Remain and ACT_Complete), and one Combi (ACT_Perform). In the work flow template, there are two types of simulation resources: rq_ACT and RES. Resource rq_ACT could be considered as an entity indicating flow of work or permission to perform the work, or it could also be considered a “unit” (i.e., a repetitive unit, such as a floor.)

Resource RES is the resources required to perform the work such as labor and equipment. The rq_ACT resource is the main entity moving in the work flow template, while Resource RES moves back and forth between work flow and resource templates. The model for Resource RES will be discussed later in Section 5.2.2, Resource Flow Template. It suffices to mention that for a repetitive activity to start, it requires 1) permission to start, which is Resource rq_ACT, and 2) resources such as labor and equipment, which is Resource RES.

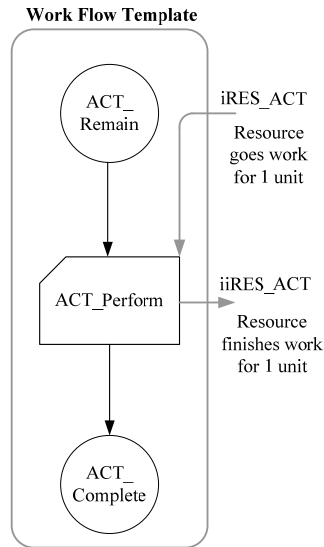


Figure 5.7 Work flow template (work flow sub-network for Activity ACT)

Tracking work progress in the work flow sub-network can be accomplished by examining:

- 1) The number of rq_ACT resources in ACT_Remain Queue
- 2) The number of rq_ACT resources in ACT_Complete Queue
- 3) The number of instances in ACT_Perform Combi

Whereas the number of rq_ACT resources in ACT_Remain Queue represents the remaining units of work, the number of rq_ACT in ACT_Complete Queue represents the completed units of work (units). For example, if Activity ACT represents laying dry wall in a 10-story building, 10 units of rq_ACT will be initiated in ACT_Remain at the start of simulation representing the 10 stories. In other words, there are 10 remaining units of work for ACT. At the end of each replication, the number of rq_ACT in ACT_Remain will become zero, because all the work units in Activity ACT have already been completed.

In contrast, the number of `rq_ACT` in `ACT_Complete` is zero in the beginning of simulation, since no work has been completed. Then, at the end of simulation, the number of `rq_ACT` in `ACT_Complete` will be equal to the total number of units, or 10 units for the given example above. For `ACT_Perform Combi`, the presence of `rq_ACT` resource and `RES` resource together in `ACT_Perform` indicates an on-going unit (e.g., Resource `RES` is working on a particular unit of Activity `ACT`.) An on-going unit in `ACT_Perform` is an instance of the `ACT_Perform`. Thus, the number of instances in `ACT_Perform Combi` is the number of on-going units of work.

Activity durations are coded in `ACT_Perform combi` in work flow template. `ACT_Perform` is a conditional activity that controls the start of works and durations. `ACT_Perform Combi` represents the performance of activity `ACT` in a particular unit or one `rq_ACT` resource. Every time `ACT_Perform` starts, it draws one `rq_ACT` resource from `ACT_Remain Queue`, holds the resource for the duration of the activity, and at the completion releases the `rq_ACT` resource to `ACT_Complete Queue`. Thus, each `rq_ACT` resource moves from `ACT_Remain` (not started), to `ACT_Perform combi` (in process), and finally to `ACT_Complete` (completed).

Precedence constraints in the work flow template are implemented through simulation code. Although it is possible to use links to model precedence constraints, using simulation code reduces the number of links in the simulation model, and keeps the model organized. Moreover, it provides flexibility in coding constraints. Figure 5.8 shows two sub-networks conforming to the work flow template for Activity A and its successor Activity B. Notice that these two sub-networks are not connected by any links. The precedence relationship between A and B is implemented in `B_Perform Combi`

through a “semaphore.” Used in Stroboscope, the semaphore is a logical start control or conditional statement. Thus, the semaphore for B_Perform combi prevents Activity B from starting until its predecessor Activity A, in the same unit, has been completed. As mentioned earlier, the number of completed work units for Activity A is the number of resource rq_A in A_Complete Queue.

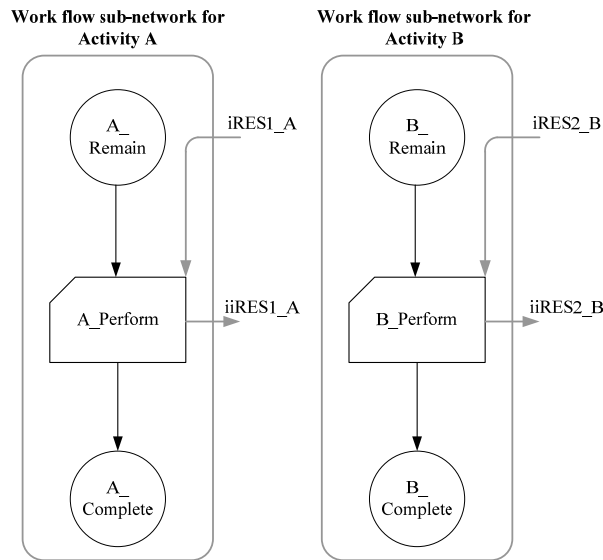


Figure 5.8 Two work flow sub-networks for Activities A and B

In particular, the logical expression for the semaphore for Activity B in Figure 5.8 compares the number of Resource rq_B in B_Complete Queue (completed units of B) and the number of instances in B_Perform (on-going work units of B) to the number of Resource rq_A in A_Complete Queue (completed units of A). The simulation code in Stroboscope language for this semaphore is

```
B_Complete.CurCount + B_Perform.CurInst < A_Complete.CurCount
```

CurCount is a predefined Stroboscope variable that returns the number of resources currently residing in the referred-to queue; CurInst returns the current number

of instances (on-going work) in the referred-to Combi. Thus, the above statement is true when the number of completed units in Activity A is greater than the completed units of B plus the number of on-going units in B. If 1) semaphore is true, 2) B_Remain Queue is not empty, and 3) all resources required by Activity B are available, B_Perform Combi will draw one Resource rq_B from B_Remain Queue and one Resource B from the resource flow sub-network for Resource B via $iRES2_B$ link in order to start work. As a result, an instance of B_Perform will be created at this particular moment with duration that equals the time required to finish this particular work unit.

5.2.2 Resource Flow Template

The main functionalities of the resource flow template are distinguishing different states of resources and collecting the statistical data of the resources. To be specific, the resource work template is able to:

- 1) Present the idle state of resources during employment
- 2) Present the idle state of resources during unemployment
- 3) Record the period of crew idle time (CIT)
- 4) Postpone resource arrival date (using the determined crew lead time, CLT)
- 5) Determine whether to keep or lay off the resource so that (1) and (2) are distinguishable

Resource flow template is shown in Figure 5.9. It consists of two queues (RES_Offsite and RES_Idle), one combi (RES_CLT), and one fork (RES_LeaveOrStay). There is only one type of resource circulating in the resource flow, which is Resource RES.

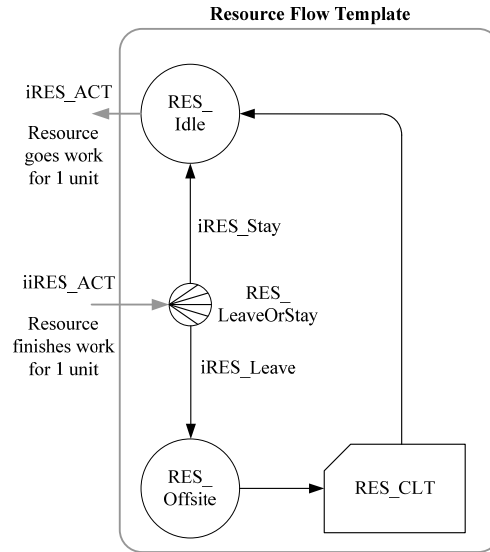


Figure 5.9 Resource flow template (resource flow sub-network for Resource RES)

RES_Offsite is a queue where RES resource resides when not on site (unemployed). At the start of simulation, RES resource is initialized in the RES_Offsite Queue, and will return to this queue after all work units have been completed. Importantly, the time resources spend in RES_Offsite is not considered as idle time because resources in it are unemployed. In contrast, the time resources spend in RES_Idle is considered idle time. Crew idle time (CIT) represents the sum of arrival idle time (AIT) and unit idle time (UIT).

RES_Idle is a queue where RES resource is on the site waiting, not performing any work. This situation occurs when its activity ACT_Perform cannot start because its predecessors in the same repetitive unit have not finished yet. The total time resource RES spends in RES_Idle Queue is the crew idle time (CIT) for the RES resource. This duration (CIT) represents the sum of arrival idle time (AIT) and unit idle time (UIT). The total time RES spends in RES_Idle, (CIT), is given by:

$$RES_Idle.AveWait \times RES_Idle.TotCount$$

AveWait and TotCount are predefined Stroboscope variables. AveWait is the average visit time (duration) spent by resources in the referred-to queue. TotCount is the number of times that resources entered the queue. Accordingly, the product of these two variables is the total idle time resources spend in the referred-to queue.

Note that the total time a resource on the site is the sum of duration the resource spends in RES_Idle and its corresponding activity in ACT_Perform.

RES_CLT Combi, representing an abstract activity, is included in the resource flow template to postpone the arrival of a resource by using a determined duration of RES_CLT, crew lead time. As a result, the duration of RES_CLT (CLT) delays the start date of the corresponding activity. As discussed in Chapter 4, Sequence Step Algorithm, CLT of resources is set to zero at the start of SQS-AL; all resources are assumed to arrive to the site at the beginning of the project. Accordingly, the duration of RES_CLT Combi, which is CLT, is initially set to zero, before processing its activity's sequence step.

From one replication to the next, SQS-AL collects CITs of resources belonging to that particular sequence step. From one sequence step to the next, SQS-AL constructs cumulative frequency of the collected CITs, determines CLT, and assigns the CLT to the duration of RES_CLT. The CLT value assigned to duration of RES_CLT is the time to hold resource RES in the simulation in order to postpone the resource arrival date to the site. Therefore, the postponement of the activity start date is stipulated by the resource availability constraints for Resource RES.

Note that the unemployment period for the RES is the total time the RES spends in RES_Offsite Queue and RES_CLT Combi.

To determine whether a resource should be kept on site or laid off, a Stroboscope simulation element, called “Fork,” is used. RES_LeaveOrStay (or RES_F) Fork is a decision point determining either to send RES resource to RES_Idle Queue (holding the resource) or to send it to RES_Offsite Queue (laying off the resource). Whether to keep or lay off the resource depends on whether there is any remaining work (rq_ACT) in ACT_Remain Queue. If the following statement is true, the RES_LeaveOrStay fork will send the RES back to RES_Idle Queue.

```
ACT_Remain.CurCount > 0
```

On the other hand, if the above statement is false, meaning all the work is completed, the RES_LeaveOrStay Fork will send RES to RES_Offsite.

After a resource is laid off, it is not allowed to re-enter the site, achieved by comparing between 1) the current number of completed units (*CurCount*) in ACT_Complete Queue and 2) the total number of units. Resources are allowed to enter the site only if the CurCount of their corresponding activities is not equal to the total number of units. This condition is shown below.

```
ACT_Complete.CurCount != the total number of units
```

5.3 Example 5.1 Simulation code and model for a repetitive project

Example 4.2 from Chapter 4 is used to demonstrate the application of the discussed model templates and SQS-AL. The example is implemented in Stroboscope. The simulation model and code in Stroboscope language are given with explanation. Several important aspects of the simulation model and SQS-AL are re-deliberated along with the given simulation code.

The example is a repetitive project consisting of 4 units with 7 activities in each. Each activity is performed by a dedicated resource named after that activity. Figure 5.10 presents a single unit precedence diagram for this example, showing that repetitive activities in Sequence Steps 1 to 4 (SQS1 to SQS4). For example, Activity A is in SQS1; B and C are in SQS2, etc.

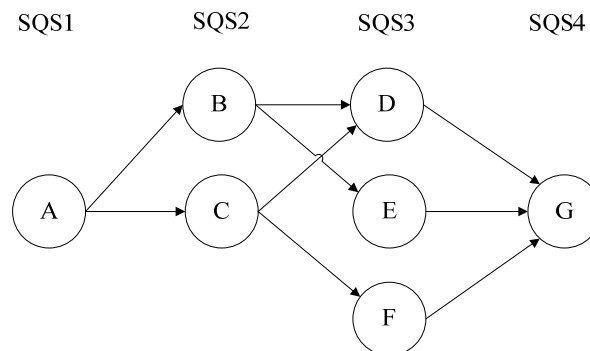


Figure 5.10 Single Unit Precedence Diagram

Table 5.1 presents the work amount of each activity in each of the 4 units, whereas Table 5.2 presents the production rates of each particular activity. Production rates of activities are assumed to follow a normal distribution with the means and standard deviations displayed in Table 5.2.

	Activity						
	A	B	C	D	E	F	G
Unit	Work Amount (Quantity/Unit)						
1	100	150	200	150	100	150	50
2	250	100	150	200	150	250	200
3	150	200	50	100	50	50	50
4	200	150	200	150	100	100	150

Table 5.1 Work amounts for each activity in each unit

Activity	Mean	SD
A	10	1.0
B	20	2.0
C	15	1.5
D	15	1.5
E	25	2.5
F	15	1.5
G	20	2.0

Table 5.2 Daily Production Rates

Figure 5.11 is the simulation model for this example. As can be seen, Figure 5.11 resembles Figure 5.10, a single unit precedence diagram. Simulation model templates are used to represent activities and resources, instead of nodes, in Figure 5.10.

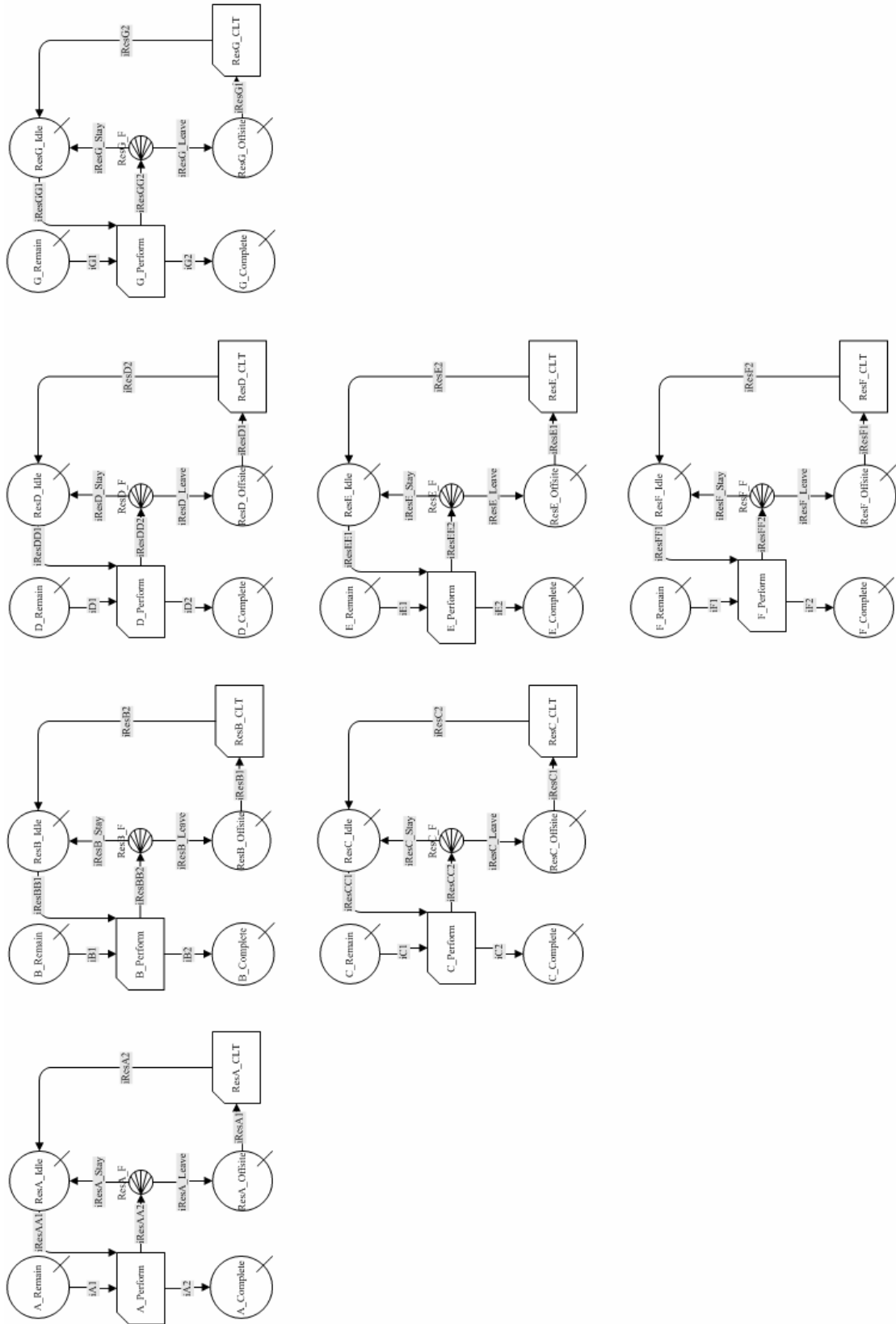


Figure 5.11 Simulation model for Example 5.1

5.3.1 Simulation Code for Model Parameters (MP)

The Simulation code in the following sections is encoded in Stroboscope GUI's Model Parameters (MP).

5.3.1.1 Variables controlling replication and sequence step loops (MP.Loops)

Variables that controls SQS-AL's replication loop and sequence step loop are stored in SaveValue, which is a storage class in Stroboscope language. "nRep" is the total number of replications executed in processing each sequence step, whereas "nSQS" is the total number of sequence steps. The ithRep and ithSQS indicate the current replication and processing sequence step, respectively.

```
SAVEVALUE nRep 3000;  
SAVEVALUE nSQS 4;  
SAVEVALUE ithRep* 1;  
SAVEVALUE ithSQS* 1;
```

Note that in Stroboscope the asterisk sign "*" indicates that the referred-to SaveValue (e.g., ithRep and ithSQS) is persistent throughout simulation runs, meaning the SaveValue with an asterisk sign will not be reset to its initial value (e.g., 1 for ithRep and ithSQS for the code above) at the beginning of each replication. Since it is necessary to track the current replication (ithRep) and current sequence step (ithSQS), these SaveValues must be persistent and are declared so by the asterisk sign at the end of their names.

5.3.1.2 Work amounts (MP.ACT.Quantity)

For each activity, the work amount in each repetitive unit is stored in one-dimensional arrays. The number 4 in the code below is the total number of units in this

project; it defines the size of the array. The figures in parenthesis are the work amount for each activity in each unit; these figures initialize the member of the array. So each of the following statements defines and initializes the arrays holding the work amount for each activity.

```
ARRAY A_Quantity 4 {100 250 150 200};  
ARRAY B_Quantity 4 {150 100 200 150};  
ARRAY C_Quantity 4 {200 150 50 200};  
ARRAY D_Quantity 4 {150 200 100 150};  
ARRAY E_Quantity 4 {100 150 50 100};  
ARRAY F_Quantity 4 {150 250 50 100};  
ARRAY G_Quantity 4 {50 200 50 150};
```

Note that indexing arrays in the Stroboscope language is zero-based. Thus, the index of the first value in an array is zero, which is the same as in the C language.

5.3.1.3 Confidence levels (MP.RES.ConfidenceLevel)

Confidence levels for determining the CLT for each resource are stored in the SaveValues shown below. For this example, a 50 percent confidence level is used for all resources.

```
SAVEVALUE ResA_ConfidenceLevel 0.5 ;  
SAVEVALUE ResB_ConfidenceLevel 0.5 ;  
SAVEVALUE ResC_ConfidenceLevel 0.5 ;  
SAVEVALUE ResD_ConfidenceLevel 0.5 ;  
SAVEVALUE ResE_ConfidenceLevel 0.5 ;  
SAVEVALUE ResF_ConfidenceLevel 0.5 ;  
SAVEVALUE ResG_ConfidenceLevel 0.5 ;
```

5.3.1.4 Additional variables (MP.AdditionVariable)

There are two additional variables, which include `iSQS` and `nthBinInterval` used later. `iSQS` is used to automatically generate Stroboscope code, whereas `nthBinInterval` is used to determine the corresponding crew lead time (CLT). Both additional variables are stored in `SaveValues`. Note that `iSQS` is used for coding purpose, while `ithSQS` is used to track the current index of sequence step.

```
SAVEVALUE iSQS 1;
SAVEVALUE nthBinInterval 0;
```

5.3.2 Simulation Code for Programming Objects (PO)

The simulation code in the following subsections is encoded in Stroboscope GUI's Programming Objects.

5.3.2.1 Permission to the site (PO.RES.Semaphore)

One important decision (i.e., `RES_CLT_Semaphore`) is whether to allow resources to enter the site (i.e., letting resources in `RES_CLT_Combis`) or to keep them off the site (i.e. holding resources in `RES_Offsite Queues`). Resources are allowed to enter the site only if their corresponding activities must not yet have been completed. For example, the number of units in `ACT_Complete Queues` must not equal the total number of units (4 units for this example). Without this decision (or condition), resources will always re-enter the site, even though all the work has been completed, and rest in `RES_Idle` until the simulation ends because there is no work left for them to work. The condition (`RES_CLT_Semaphore`) of allowing a resource to enter the site is stored in `Variables`, updated automatically by Stroboscope. It is encoded in Programming Objects, and used in `RES_CLT`'s semaphore in resource flow sub-networks.

```

VARIABLE ResA_CLT_Semaphore ' A_Complete.CurCount != 4' ;
VARIABLE ResB_CLT_Semaphore ' B_Complete.CurCount != 4' ;
VARIABLE ResC_CLT_Semaphore ' C_Complete.CurCount != 4' ;
VARIABLE ResD_CLT_Semaphore ' D_Complete.CurCount != 4' ;
VARIABLE ResE_CLT_Semaphore ' E_Complete.CurCount != 4' ;
VARIABLE ResF_CLT_Semaphore ' F_Complete.CurCount != 4' ;
VARIABLE ResG_CLT_Semaphore ' G_Complete.CurCount != 4' ;

```

5.3.2.2 Precedence constraints (PO.ACT.Semaphore)

Precedence constraints (e.g., B_Perform_Semaphore) are stored in Variables and used in ACT_Perform's semaphore work flow sub-networks. Due to precedence constraints, an activity can start only if its current number of completed units (e.g., B_Complete.CurCount) plus its current number of on-going units (e.g., B_Perform.CurInst) is less than its predecessor's number of completed units (e.g., A_Complete.CurCount). For this example, all precedence constraints are shown below:

```

VARIABLE B_Perform_Semaphore
    'B_Complete.CurCount + B_Perform.CurInst
    < A_Complete.CurCount' ;
VARIABLE C_Perform_Semaphore
    'C_Complete.CurCount + C_Perform.CurInst
    < A_Complete.CurCount' ;
VARIABLE D_Perform_Semaphore
    'D_Complete.CurCount + D_Perform.CurInst
    < B_Complete.CurCount
    & D_Complete.CurCount + D_Perform.CurInst
    < C_Complete.CurCount' ;
VARIABLE E_Perform_Semaphore
    'E_Complete.CurCount+ E_Perform.CurInst

```

```

        < B_Complete.CurCount ' ;

    VARIABLE F_Perform_Semaphore

        'F_Complete.CurCount +F_Perform.CurInst
        < C_Complete.CurCount ' ;

    VARIABLE G_Perform_Semaphore

        'G_Complete.CurCount+G_Perform.CurInst
        < D_Complete.CurCount
        &
        G_Complete.CurCount+G_Perform.CurInst
        < E_Complete.CurCount
        &
        G_Complete.CurCount+G_Perform.CurInst
        <F_Complete.CurCount ' ;

```

5.3.2.3 Activity Duration (PO.ACT.Duration)

Activity durations (e.g., A_Perform_Duration) are stored in Variables and used in ACT_Perform's duration in the work flow sub-networks. ACT_Perform's duration varies from unit to unit due to amount of works in each unit and variability in production rates. Therefore, activity durations are a function of work amount of the current unit and production rate. The duration of ACT_Perform Combis is essentially the work amount in a unit divided by the production rate of that activity. Duration variables of activities are shown below:

```

    VARIABLE A_Perform_Duration

        A_Quantity[4-A_Remain.CurCount]*1/Normal[10,1];

    VARIABLE B_Perform_Duration

        B_Quantity[4-B_Remain.CurCount]*1/Normal[10,1];

    VARIABLE C_Perform_Duration

```

```

        C_Quantity[4-C_Remain.CurCount]*1/Normal[15,1.5];
VARIABLE D_Perform_Duration
        D_Quantity[4-D_Remain.CurCount]*1/Normal[15,1.5];
VARIABLE E_Perform_Duration
        E_Quantity[4-E_Remain.CurCount]*1/Normal[25,2.5];
VARIABLE F_Perform_Duration
        F_Quantity[4-F_Remain.CurCount]*1/Normal[15,1.5];
VARIABLE G_Perform_Duration
        G_Quantity[4-G_Remain.CurCount]*1/Normal[20,2];

```

5.3.2.4 Decision of keeping or laying off resource (PO.RES.Strength)

The decisions of keeping or laying off resources are stored in variables (e.g., ResA_Leave_Strength) and used in resource flow sub-networks. After a resource completes one unit of work, it will consider whether to leave the site. If there is no remaining work for the resource, it will leave. The resources' decisions whether to leave or stay are shown below:

```

VARIABLE ResA_Leave_Strength '(A_Remain.CurCount == 0)? 1:0';
VARIABLE ResB_Leave_Strength '(B_Remain.CurCount == 0)? 1:0';
VARIABLE ResC_Leave_Strength '(C_Remain.CurCount == 0)? 1:0';
VARIABLE ResD_Leave_Strength '(D_Remain.CurCount == 0)? 1:0';
VARIABLE ResE_Leave_Strength '(E_Remain.CurCount == 0)? 1:0';
VARIABLE ResF_Leave_Strength '(F_Remain.CurCount == 0)? 1:0';
VARIABLE ResG_Leave_Strength '(G_Remain.CurCount == 0)? 1:0';

```

Note that these decision variables are used in the Strengths of two links, iRES_Leave and iRES_Stay in the resource flow sub-networks. See Section 5.3.3.5, CME.RES.Stay.Strength, and Section 5.3.3.6, CME.RES.Leave.Strength for more detail.

5.3.2.5 Temporary SaveValues for crew idle time (PO.RES.TempIdleTime)

The following SaveValues are for temporarily recording the total idle time in each resource in each replication. They are defined in Programming Objects and reset at the beginning of each replication.

```
SAVEVALUE svResB_Idle 0;  
SAVEVALUE svResC_Idle 0;  
SAVEVALUE svResD_Idle 0;  
SAVEVALUE svResE_Idle 0;  
SAVEVALUE svResF_Idle 0;  
SAVEVALUE svResG_Idle 0;
```

These svRES_Idle SaveValues (temporary storages) of idle time will be assigned to their corresponding BinCollectors (permanent storages), declared in Section 5.3.2.8 PO.RES.CIT.SQS, for a specific sequence step, at the end of each replication (see Section 5.3.4.5 CS.RES.CIT.SQS). After the assignment, these SaveValues are, then, reset prior to the execution of a new replication.

5.3.2.6 Crew idle time (PO.RES.CIT)

Samples of crew idle time (CIT) used to determine crew lead time (CLT) are stored in BinCollectors. BinCollectors are data holders that keep statistics of the numbers they receive in intervals, specified by users. Every resource that is scheduled by SQS-AL in order to achieve continuous resource utilization must have at least one RES_CIT BinCollector, which is RES_CIT1.

BINCOLLECTOR statement requires 4 arguments, which are: 1) the name of the collector, 2) the number of intervals, 3) the lower bound of the collected data, and 4) the upper bound of the collected data. For this example in Figure 5.11, RES_CIT1

BinCollectors collect CITs from a range of 0 to 300 days. This range is divided into 60 intervals, meaning that CIT samples are grouped on a five-day basis from 0 to 5 days, 5 to 10 days, etc., assuming five work days per week.

Since crew idle time of resources is collected from one replication to another, the data in RES_CIT1 BinCollectors must be persistent, and thus an asterisk, “*”, is required when their names are declared.

```
BINCOLLECTOR ResB_CIT1* 60 0 300 ;
BINCOLLECTOR ResC_CIT1* 60 0 300 ;
BINCOLLECTOR ResD_CIT1* 60 0 300 ;
BINCOLLECTOR ResE_CIT1* 60 0 300 ;
BINCOLLECTOR ResF_CIT1* 60 0 300 ;
BINCOLLECTOR ResG_CIT1* 60 0 300 ;
```

5.3.2.7 Crew lead time (PO.RES.CLT.Duration)

The crew lead time (CLT) for each resource is stored in a persistent SaveValue (requiring an asterisk sign “*” after its name), and used to set the duration RES_CLT Combi in resource flow sub-network. The initial value for CLT is zero according to the sequence step algorithm. The RES_CLT_Duration Variables are created mainly to systemize the simulation code. They are used in RES_CLT Combis, discussed in section

5.3.3.1 CME.RES.CLT.Duration.

```
SAVEVALUE ResB_CLT1* 0 ;
SAVEVALUE ResC_CLT1* 0 ;
SAVEVALUE ResD_CLT1* 0 ;
SAVEVALUE ResE_CLT1* 0 ;
SAVEVALUE ResF_CLT1* 0 ;
SAVEVALUE ResG_CLT1* 0 ;
```



```

VARIABLE ResB_CLT_Duration ResB_CLT1;
VARIABLE ResC_CLT_Duration ResC_CLT1;
VARIABLE ResD_CLT_Duration ResD_CLT1;
VARIABLE ResE_CLT_Duration ResE_CLT1;
VARIABLE ResF_CLT_Duration ResF_CLT1;
VARIABLE ResG_CLT_Duration ResG_CLT1;

```

Note that the number one at the end of RES_CLT1 indicates the first lead time measured from the project start date. In Chapter 6 where work breaks in resource schedules are discussed, a resource with one work break will have RES_CLT1 (measured from project start date) and RES_CLT2 (measured from the time the resource takes the break).

5.3.2.8 Crew idle time for each sequence step (PO.RES.CIT.SQS)

The code below of BinCollectors for crew idle time (CIT) associated with sequence steps (SQS) is for the purpose of recording the changes in CITs from one SQS to another. Note that the iSQS in the below code is used with “\$<...>\$” (explained later) to automatically create simulation code, while ithSQS is used to track the current index of sequence step.

```

ASSIGN iSQS 1;
WHILE 'iSQS<=nSQS+1';
    BINCOLLECTOR bcltResA_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResB_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResC_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResD_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResE_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResF_IdleSQS<iSQS>$* 60 0 300;
    BINCOLLECTOR bcltResG_IdleSQS<iSQS>$* 60 0 300;

```

```

        BINCOLLECTOR bcltProjectDurationSQS$<iSQS>$* 60 0 300;
        BINCOLLECTOR bcltProjectIdleTimeSQS$<iSQS>$* 60 0 300;
        ASSIGN iSQS iSQS+1;
WEND; /iSQS

```

Note that the “\$<Argument>\$” is a preprocessor operator of Stroboscope used to automatically generate simulation code. The Stroboscope evaluates the argument in the operator, and replaces the operator and argument before executing the statement. This preprocessor operator is useful when there is a consistent pattern of simulation code (Martinez 1995). For more information about the “\$<Argument>\$” operator, see The Stroboscope Simulation Language, Chapter 15, Statement Preprocessing and Automatic Code Generation.

5.3.3 Simulation Code for Model Elements (CME)

The simulation code in this section (CME) can be easily encoded in simulation model elements (Visio shapes) such as Combis and Links on drawings in Stroboscope GUI. To enter the following code, users can double click on a shape of Combis or Links (e.g., Figure 5.12 is the GUI for ResB_CLT Combi) and enter the code as shown in this section. The following statements specify attributes of simulation elements (i.e., Combis and Links) in Stroboscope GUI where the code is stored. These statements are:

- 1) SEMAPHORE for Semaphore Combis
- 2) DURATION for Duration in Combis
- 3) STRENGTH for Strength in Links
- 4) ONFLOW for OnFlow in Links

5.3.3.1 Semaphore in RES_CLT Combis (CME.RES.Semaphore)

The Semaphores of RES_CLT Combis are shown below. They are stored in simulation model elements (Visio shapes for Combis in Stroboscope GUI), as shown in Figure 5.12. When using the Stroboscope GUI for simulation model elements as shown in Figure 5.12, certain statements of Stroboscope must be omitted because Stroboscope GUI will automatically create those statements such as in Figure 5.12 for the SEMEPHORE statement in this section and DURATION statement in the next section. Details of this semaphore are given in Section 5.3.2.1, PO.RES.Semaphore.

```
SEMAPHORE ResB_CLT ResB_CLT_Semaphore;  
SEMAPHORE ResC_CLT ResC_CLT_Semaphore;  
SEMAPHORE ResD_CLT ResD_CLT_Semaphore;  
SEMAPHORE ResE_CLT ResE_CLT_Semaphore;  
SEMAPHORE ResF_CLT ResF_CLT_Semaphore;  
SEMAPHORE ResG_CLT ResG_CLT_Semaphore;  
SEMAPHORE ResH_CLT ResH_CLT_Semaphore;  
SEMAPHORE ResJ_CLT ResJ_CLT_Semaphore;
```

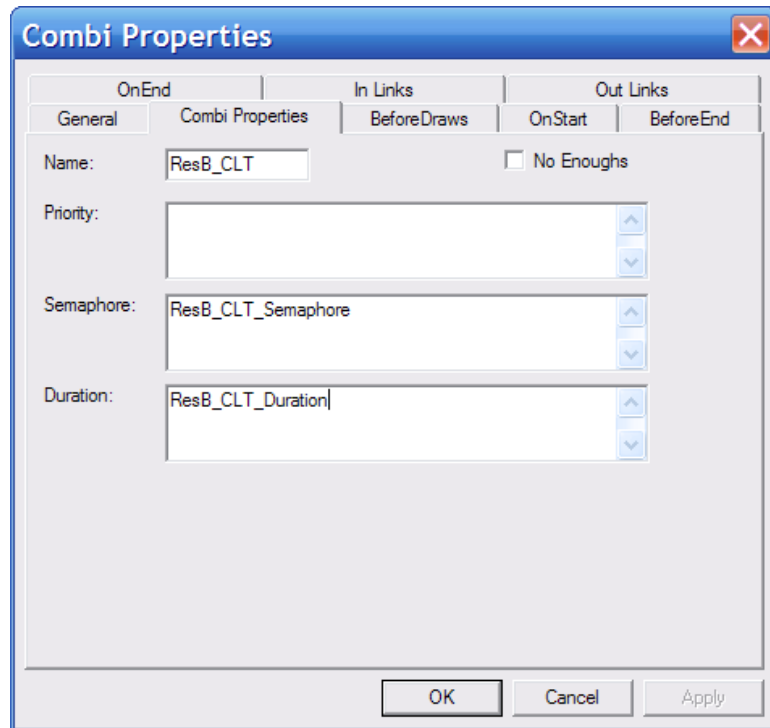


Figure 5.12 Assigning semaphore and duration for ResB_CLT Combi

5.3.3.2 Duration in RES_CLT Combis (CME.RES.CLT.Duration)

Durations of RES_CLT Combis in resource flow are shown below, stored in simulation model elements (Visio shapes for Combis in Stroboscope GUI), as shown in Figure 5.12. Details of these durations are given in Section 5.3.2.7, PO.RES.CLT.Duration.

```

DURATION ResA_CLT ResA_CLT_Duration;
DURATION ResB_CLT ResB_CLT_Duration;
DURATION ResC_CLT ResC_CLT_Duration;
DURATION ResD_CLT ResD_CLT_Duration;
DURATION ResE_CLT ResE_CLT_Duration;
DURATION ResF_CLT ResF_CLT_Duration;
DURATION ResG_CLT ResG_CLT_Duration;
DURATION ResH_CLT ResH_CLT_Duration;

```

```
DURATION ResJ_CLT ResJ_CLT_Duration;
```

5.3.3.3 Semaphore in ACT_Perform Combis (CME.ACT.Semaphore)

Semaphores of ACT_Perform Combis in work flow are shown below. They are stored in simulation model elements, as shown in Figure 5.13. Details of these semaphores are given in Section 5.3.2.7, PO.RES.CLT.Duration.

```
SEMAPHORE A_Perform A_Perform_Semaphore;  
SEMAPHORE B_Perform B_Perform_Semaphore;  
SEMAPHORE C_Perform C_Perform_Semaphore;  
SEMAPHORE D_Perform D_Perform_Semaphore;  
SEMAPHORE E_Perform E_Perform_Semaphore;  
SEMAPHORE F_Perform F_Perform_Semaphore;  
SEMAPHORE G_Perform G_Perform_Semaphore;
```

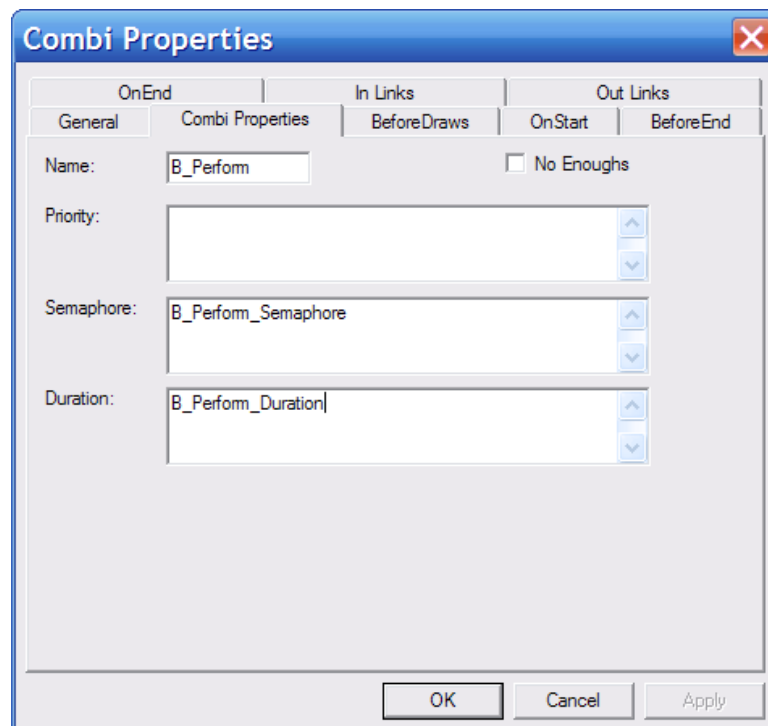


Figure 5.13 Assigning semaphore and duration for B_Perform Combi

5.3.3.4 Duration in ACT_Perform Combis (CME.ACT.Duration)

Durations of ACT_Perform Combis in work flow sub-networks are shown below. These ACT_Perform_Duration Variables are stored in simulation model elements, as shown in Figure 5.13. Details of these durations are given in Section 5.3.2.3, PO.ACT.Duration.

```
DURATION A_Perform A_Perform_Duration;  
DURATION B_Perform B_Perform_Duration;  
DURATION C_Perform C_Perform_Duration;  
DURATION D_Perform D_Perform_Duration;  
DURATION E_Perform E_Perform_Duration;  
DURATION F_Perform F_Perform_Duration;  
DURATION G_Perform G_Perform_Duration;
```

5.3.3.5 Strength in iRES_Stay Links (CME.RES.Stay.Strength)

Strengths of iRES_Stay links in work flow sub-networks are shown below. They are stored in simulation model elements, as shown in Figure 5.14. Details of these strengths are given in Section 5.3.2.4 PO.RES.Strength. Notice, the RES_Leave_Strength Variables are preceded by an exclamation sign returning the opposite value of the Variables. Accordingly, the opposite of resource leaving the site is resource staying on the site.

```
STRENGTH iResA_Stay !ResA_Leave_Strength;  
STRENGTH iResB_Stay !ResB_Leave_Strength;  
STRENGTH iResC_Stay !ResC_Leave_Strength;  
STRENGTH iResD_Stay !ResD_Leave_Strength;  
STRENGTH iResE_Stay !ResE_Leave_Strength;  
STRENGTH iResF_Stay !ResF_Leave_Strength;  
STRENGTH iResG_Stay !ResG_Leave_Strength;
```

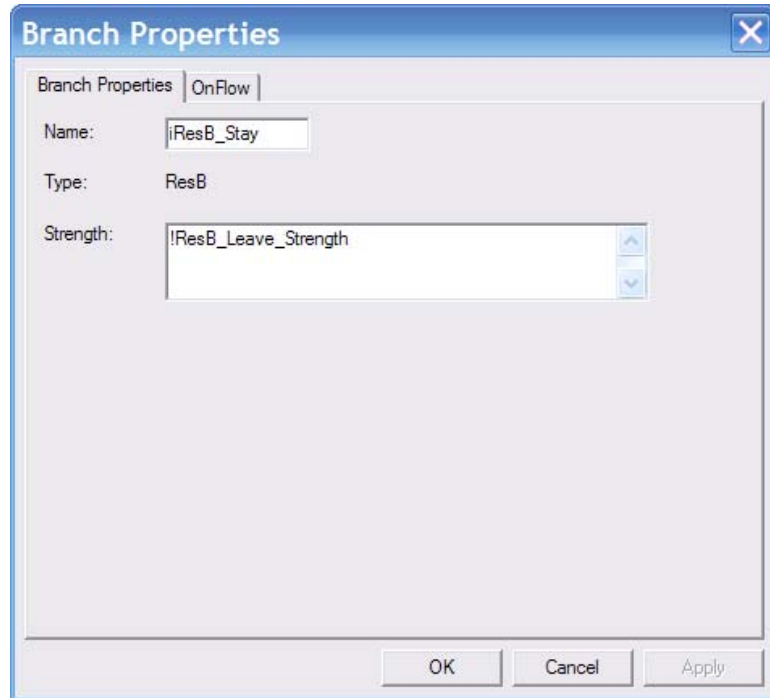


Figure 5.14 Assigning Strength for Link iResB_Stay

5.3.3.6 Strength in iRES_Leave (CME.RES.Leave.Strength)

As opposed to the strength in iRES_Stay Link, the Strength of iRES_Leave is RES_Leave_Strength Variable. Details of these strengths are given in Section 5.3.2.4, PO.RES.Strength.

```
STRENGTH iResA_Leave ResA_Leave_Strength;  
STRENGTH iResB_Leave ResB_Leave_Strength;  
STRENGTH iResC_Leave ResC_Leave_Strength;  
STRENGTH iResD_Leave ResD_Leave_Strength;  
STRENGTH iResE_Leave ResE_Leave_Strength;  
STRENGTH iResF_Leave ResF_Leave_Strength;  
STRENGTH iResG_Leave ResG_Leave_Strength;
```

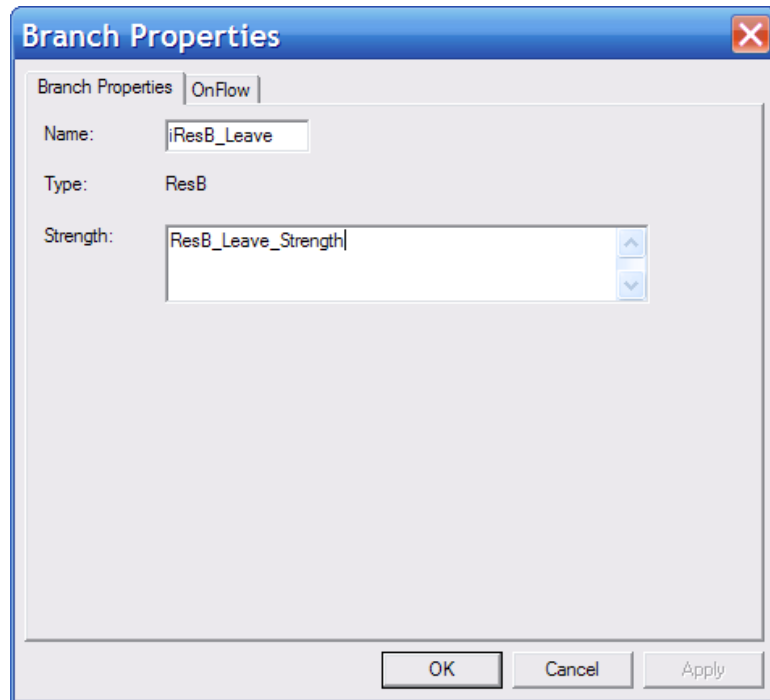


Figure 5.15 Assigning strength for iResB_Leave Link

5.3.3.7 OnFlow in iRES_Leave Links (CME.RES.Leave.OnFlow)

Crew idle time of resources is collected only when SQS-AL is processing the sequence step of the activities they serve. For example, CITs for Resource B is collected from simulation runs during processing SQS2 because Activity B is in SQS2.

```
ONFLOW iResB_Leave COLLECT ResB_CIT1
    PRECOND 'ithSQS==2' ResB_Idle.AveWait*ResB_Idle.TotCount;
ONFLOW iResC_Leave COLLECT ResC_CIT1
    PRECOND 'ithSQS==2' ResC_Idle.AveWait*ResC_Idle.TotCount;
ONFLOW iResD_Leave COLLECT ResD_CIT1
    PRECOND 'ithSQS==3' ResD_Idle.AveWait*ResD_Idle.TotCount;
ONFLOW iResE_Leave COLLECT ResE_CIT1
    PRECOND 'ithSQS==3' ResE_Idle.AveWait*ResE_Idle.TotCount;
ONFLOW iResF_Leave COLLECT ResF_CIT1
    PRECOND 'ithSQS==3' ResF_Idle.AveWait*ResF_Idle.TotCount;
```



```
ONFLOW iResG_Leave COLLECT ResB_CIT1
```

```
PRECOND 'ithSQS==4' ResG_Idle.AveWait*ResG_Idle.TotCount;
```

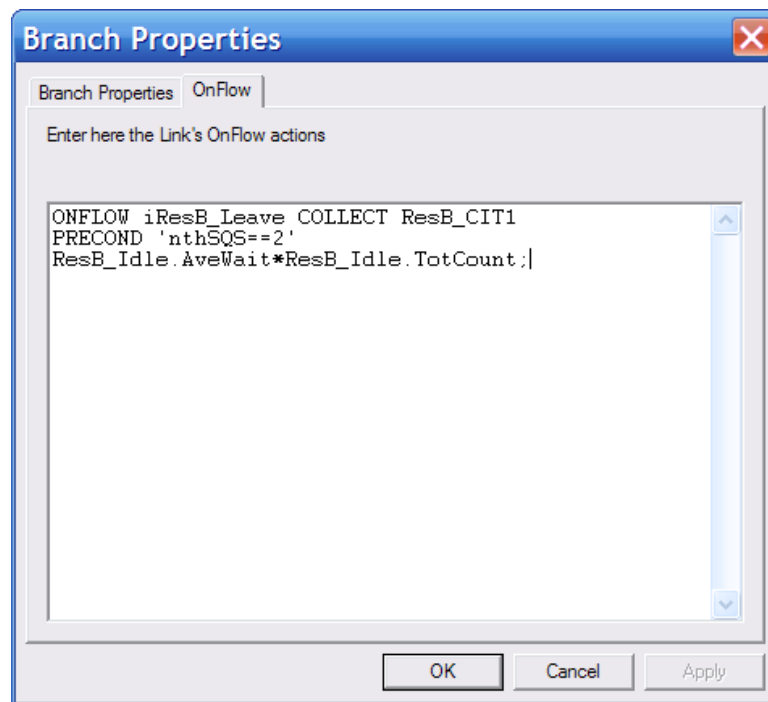


Figure 5.16 Collecting CIT_B during processing SQS2

5.3.4 Control Statements (CS)

Simulation code in the following sections is encoded in Stroboscope GUI's Control Statements.

5.3.4.1 Sequence step and replication loops (CS.Loops)

The following two While-Loops statements control SQS-AL's sequence step loop and replication loop, respectively. As discussed in Chapters 4 and 5, the replication loop is for collecting crew idle time of resources whose activities are in the current processing sequence step (ithSQS). The sequence step loop is for determining crew lead time for the resources. An extra sequence step is added to obtain the final results of project duration, project idle time, and idle time in resource utilization.

```

WHILE 'ithSQS <= nSQS+1';           / Start Sequence Step Loops.
    WHILE 'ithRep <= nRep ';       / Start Replication Loops.
        CLEAR;                     / Clear temporary data.

```

Crew lead times for all resources are determined when SQS-AL finishes processing the last sequence step. Nevertheless, an extra sequence step is added to obtain the final results of project duration, project idle time, and idle time in resource utilization.

The CLEAR statement is executed at the beginning of each replication to clear the results from a previous simulation run. This statement sets all non-persistent SaveValues to their initial values and clears all data and statistics from non-persistent Collectors and BinCollectors.

5.3.4.2 Initializing work amounts (CS.ACT.INIT)

At the beginning of each replication, the work amount in units for each activity is initialized in ACT_Remain Queues in Work Flow Networks. INIT is a Stroboscope's statement used to create and place resources in specified Queues. As discussed in Section 5.2.1, Work Flow Template, the amount of Resources rq_ACT, in ACT_Remain Queue is the number of remaining units needed to be completed, which is 4, as shown in the code below.

```

INIT A_Remain 4; / The resource name in this Queue is "rq_A".
INIT B_Remain 4; / The resource name in this Queue is "rq_B".
INIT C_Remain 4; / The resource name in this Queue is "rq_C".
INIT D_Remain 4; / The resource name in this Queue is "rq_D".
INIT E_Remain 4; / The resource name in this Queue is "rq_E".
INIT F_Remain 4; / The resource name in this Queue is "rq_F".
INIT G_Remain 4; / The resource name in this Queue is "rq_G".

```

5.3.4.3 Initialization of resources (CS.RES.INIT)

At the beginning of each replication, resources are initialized in RES_Offsite Queues in resource flow sub-networks. As discussed in Section 5.2.2, Resource Flow Template, resources in RES_Offsite Queues are considered unemployed, prior to the start of the first unit and after the end of the last unit.

```
INIT ResA_Offsite 1; / The resource name in this Queue is "ResA".  
INIT ResB_Offsite 1; / The resource name in this Queue is "ResB".  
INIT ResC_Offsite 1; / The resource name in this Queue is "ResC".  
INIT ResD_Offsite 1; / The resource name in this Queue is "ResD".  
INIT ResE_Offsite 1; / The resource name in this Queue is "ResE".  
INIT ResF_Offsite 1; / The resource name in this Queue is "ResF".  
INIT ResG_Offsite 1; / The resource name in this Queue is "ResG".
```

5.3.4.4 Executing simulation (CS.Simulate)

After initializing the work amount in ACT_Remain and the resources in RES_Offsite, the simulation starts after Stroboscope executes the SIMULATE statement. Then, the simulation replication will end when all activities are completed, i.e., when there is no rq_ACT resource in each and every one of the ACT_Remain Queues.

```
SIMULATE; /Run one replication
```

Note that the crew idle time for each resource is collected at the end of each replication during simulation runs, while crew lead time is determined at the end of processing each sequence step. Therefore, the simulation code collecting crew idle time is encoded in simulation model elements, discussed in Section 5.3.3.7, CME.RES.Leave.OnFlow, while the code determining crew lead time is coded in Control Statements, discussed in Section 5.3.4.6, CS.RES.CLT.

5.3.4.5 Recording CITs in bcltRES_IdleSQS BinCollectors (CS.RES.CIT.SQS)

At the end of the specified number of replications for each sequence step, the following data are collected.

- 1) Crew idle time of resources from processing the current sequence step
- 2) Project duration from processing the current sequence step
- 3) Sum of crew idle time and project idle time from processing the current sequence step

These collected data are for tracking changes in crew idle time, project duration, and project idle time. They are valuable for analysis of the impact of an assigned crew lead time on activities, resources, and the project. The following code is for calculating the crew idle time (e.g., svResB_Idle) for each resource (e.g., Resource B) responding to each sequence step processing. The svRES_Idle SaveValues are temporary storage for crew idle time, declared in Programming Object (5.3.2.5 PO.RES.TempIdleTime). The bcltRES_IdleSQS\$<iSQS>\$ BinCollectors (e.g., bcltResB_IdleSQS1) are permanent storage for crew idle time from each sequence step.

```
ASSIGN svResB_Idle ResB_Idle.AveWait*ResB_Idle.TotCount;
ASSIGN svResC_Idle ResC_Idle.AveWait*ResC_Idle.TotCount;
ASSIGN svResD_Idle ResD_Idle.AveWait*ResD_Idle.TotCount;
ASSIGN svResE_Idle ResE_Idle.AveWait*ResE_Idle.TotCount;
ASSIGN svResF_Idle ResF_Idle.AveWait*ResF_Idle.TotCount;
ASSIGN svResG_Idle ResG_Idle.AveWait*ResG_Idle.TotCount;
ASSIGN iSQS 1;
WHILE 'iSQS<=nSQS+1';
    IF 'ithSQS==$<iSQS>$';
        COLLECT bcltResB_IdleSQS$<iSQS>$ svResB_Idle;
```

```

        COLLECT bcltResC_IdleSQS$<iSQS>$ svResC_Idle;
        COLLECT bcltResD_IdleSQS$<iSQS>$ svResD_Idle;
        COLLECT bcltResE_IdleSQS$<iSQS>$ svResE_Idle;
        COLLECT bcltResF_IdleSQS$<iSQS>$ svResF_Idle;
        COLLECT bcltResG_IdleSQS$<iSQS>$ svResG_Idle;
        COLLECT bcltProjectDurationSQS$<iSQS>$ SimTime;
        COLLECT bcltProjectIdleTimeSQS$<iSQS>$
                'svResA_Idle+ svResB_Idle+ svResC_Idle
                +svResD_Idle+ svResE_Idle+ svResF_Idle
                +svResG_Idle';
    ENDIF;
    ASSIGN iSQS iSQS+1;
WEND; /iSQS
ASSIGN ithRep ithRep+1; / Increase ithRep by 1
WEND; /ithRep

```

Remember that `iSQS` is used for coding purpose, while `ithSQS` is used to track the current index of sequence step. For the code above, the small nested loop controlled by `iSQS` is not the sequence step loop.

The last line of the code is the end of SQS-AL's replication loop. When Stroboscope reaches this line, it will check whether the current replication index (`ithRep`) is less than or equal to the total number of replications (`nRep`). The corresponding `WHILE` statement for this `WEND` of `ithRep` (replication loop) is in Section 5.3.4.1, `CS.Loops`. SQS-AL exits the replication loop when `ithRep` equals to `nRep`, and enters the sequence step loop.

5.3.4.6 Determining crew lead time for activities belonging to ithSQS (CS.RES.CLT)

After SQS-AL exits the replication loop and enters the sequence step loop, the crew lead time of resources serving activities in the current processing sequence step will be determined. For example, after the total number of replications is executed in processing SQS2, SQS-AL determines CLTs of Resources ResB and ResC serving Activities B and C in SQS2, respectively. The code below shows the determination of the crew lead time for Resource Res_B.

```
IF 'ithSQS==2'; / Resource Res_B serves activity B in SQS2.
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResB_CIT1,nthBinInterval]
        <ResB_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResB_CLT1 BinHigh[ResB_CIT1,nthBinInterval];
ENDIF;
```

As shown in the code above, if the current sequence step is SQS2 (ithSQS==2), SQS-AL will determine CLT_B . The *nthBinInterval* SaveValue is used to indicate the current index of ResB_CIT1 BinCollector. The nthBinInterval is increased by 1 until the cumulative frequency value of the indexed element is greater than the user-specified confidence level for that particular resource. To obtain the cumulative frequency, *PctAtOrBelowBin* function is used; it returns the percentage of values collected in the indexed element and all lower elements. To obtain the value of crew lead time, BinHigh function is used; it returns the upper bound of the indexed element.

For the example in Table 4.4, the element in ResB_CIT1 BinCollector having a PctAtOrBelowBin[ResB_CIT1,nthBinInterval] greater than 80% is in the range of 55

days. Accordingly, `PctAtOrBelowBin[ResB_CIT1, nthBinInterval]` returned 94.27%, whereas `BinHigh[ResB_CIT1, nthBinInterval]` returned 55 days. The following simulation code is for obtaining the crew lead time for Resources ResC, ResD, ResE, ResF, and ResG.

The code below is for determining CLT_C (ResC_CIT1) of Resource ResC serving Activity C in SQS2.

```

IF 'ithSQS==2'; / Resource Res_C serves activity C in SQS2.
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResC_CIT1,nthBinInterval]
        <ResC_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResC_CLT1 BinHigh[ResC_CIT1,nthBinInterval];
ENDIF;

```

The code below is for determining CLT_D (ResD_CIT1) of Resource ResD serving Activity D in SQS3.

```

IF 'ithSQS==3'; / Resource Res_D serves activity D in SQS3.
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResD_CIT1,nthBinInterval]
        <ResD_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResD_CL1 BinHigh[ResD_CIT1, nthBinInterval];
ENDIF;

```

The code below is for determining CLT_E (ResE_CIT1) of Resource ResE serving Activity E in SQS3.

```

IF 'ithSQS==3 ';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResE_CIT1 , nthBinInterval]
        <ResE_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResE_CLT1 BinHigh[ResE_CIT1,nthBinInterval];
ENDIF;

```

The code below is for determining CLT_F (ResF_CIT1) of Resource ResF serving Activity F in SQS3.

```

IF 'ithSQS==3 ';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResF_CIT1 ,
nthBinInterval]<ResF_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResF_CLT1 BinHigh[ResF_CIT1,nthBinInterval];
ENDIF;

```

The code below is for determining CLT_G (ResG_CIT1) of Resource ResG serving Activity G in SQS4.

```

IF 'ithSQS==4';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResG_CIT1,nthBinInterval]
        <ResG_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResG_CLT1 BinHigh[ResG_CIT1,nthBinInterval];
ENDIF;

```



```
ASSIGN ithSQS ithSQS+1;  
WEND; / ithSQS
```

The last line of the code is the end of the sequence step loop. When Stroboscope reaches this line, it will check whether the current index of the sequence step (ithSQS) is greater than the total number of sequence steps plus one (nSQS+1). If ithSQS is not greater than nSQS+1, SQS-AL will begin processing the next sequence step. On the other hand, if ithSQS is greater than nSQS+1, SQS-AL will stop. At the end of SQS-AL, crew lead times for all resources are determined, and the schedule is finalized. Project duration and project idle time are recorded from the processing of every sequence step, as well as the idle time of all resources. Results from SQS-AL are comprehensively discussed in Chapter 8, ChaStrobe Application.

5.4 Summary

This chapter discusses two simulation model templates and a comprehensive simulation model and code for the Sequence Step Algorithm (SQS-AL). First, the proposed simulation model templates are introduced in details related to repetitive projects. The advantages of the simulation model templates are also discussed. The implementation of SQS-AL in simulation and cooperation between SQS-AL and the templates is presented. An example of a repetitive project is used to demonstrate the application of SQS-AL in simulation and also the proposed simulation model templates.

To facilitate the means of collecting simulation data for SQS-AL, the two templates are established: 1) Work Flow template, and 2) Resource Flow template. The Work Flow template is for modeling repetitive activities, whereas the Resource Flow template is for modeling utilization of resources. The design of the two templates

facilitates the creation of organized and systemized simulation models for applying the Sequence Step Algorithm (SQS-AL) to repetitive projects. The two templates also provide users with the simulation outputs produced by SQS-AL.

The Work Flow template is designed to produce an efficient and effective simulation model of repetitive activities. The use of the Work Flow template results in the following:

- Tracking work progress of repetitive activities. Different work statuses of repetitive activities in each unit are categorized in:
 - a. Remaining work, represented by ACT_Remain Queues
`ACT_Remain.CurCount` indicates the number of uncompleted units.
 - b. On-going work, represented by ACT_Perform Combis
`ACT_Perform.CurInst` indicates the number of in-progress units.
 - c. Completed work, represented by ACT_Complete Queues
`ACT_Complete.CurCount` indicates the number of completed units.
- Modeling precedence constraints. The precedence constraints for each repetitive activity are modeled through its semaphore by comparing its work progress to those of its preceding activities, promptly derived from tracking activities' work progress
- Simulating activity durations

The Resource Flow template is designed to represent the simulation model of resources serving repetitive activities. The use of the Resource Flow template results in the following:

- Differentiating states of resources. Different states of resources are categorized in:
 - a. Unemployed. Resources are unemployed, and thus they are not on the construction site, represented by RES_Offsite Queues. During the period of unemployment, resources are not considered idle.
 - b. Being Idle during employment. The duration that resources are employed but performing no work is considered idle time. The idle time of resources is presented by the time resources spend in RES_Idle Queues.
 - c. Being productive during employment. The duration that resources perform work is considered productive time. The productive time of resources is presented by the time resources spend in ACT_Perform Combis.
- Recording the duration of crew idle time (CIT). Crew idle time of resources is derived from the total duration resources spend in RES_Idle Queues, which is $RES_Idle.AveWait \times RES_Idle.TotCount$
- Postponing resource arrival date. Resources are scheduled to the site according to their calculated crew lead time (CLT), which is the duration of RES_CLT Combis.
- Determining when to lay off resources. The decision whether to lay off a resource is determined by checking the number of incomplete units of its activity, represented by ACT_Remain.CurCount.

As stated above, the proposed simulation model templates offer the ability of collecting data and distinguishing different statuses of activities and resources. The proposed simulation model templates provide instantaneous data required by SQS-AL.

The last example in this chapter shows the application of the templates, and also explains the simulation code and model for SQS-AL.

CHAPTER 6

WORK BREAKS

To minimize idle time and achieve continuous resource utilization, the Sequence Step Algorithm (SQS-AL) postpones the arrival date of resources. Results from the postponement are 1) cost reduction from minimizing resource idle time and 2) productivity improvement from the learning-curve phenomenon. However, minimizing idle time comes at the cost of prolonged project duration; as a result, penalty cost is incurred from the increased project duration.

Penalty cost from the increased project duration could exceed the savings from the minimized resource idle time. To mitigate this problem, relaxation of resource continuity constraints should be considered. Carefully examining and balancing the tradeoff between satisfying and relaxing continuity constraints can optimize both project duration and cost. Chapter 4 demonstrates that the tradeoff can be analyzed using different confidence levels of resources. Relaxing continuity constraints using lower confidence levels results in a shorter project duration. However, this approach is effective only to a certain extent. Relaxing continuity constraints using confidence levels may not effectively reduce idle time and project duration simultaneously. Thus, introducing another form of relaxation focusing on reducing the increased project duration without incurring idle time is necessary.

This chapter introduces the application of work breaks within repetitive activities, which is another form of relaxation of resource continuity constraints. A work break is a deliberate interruption in the work of a resource which specifies when resources should leave the site and for how long. With work breaks, resources will leave the site at the beginning of the work break and return at the end of the work break. It is essentially a predetermined interruption, intentionally scheduled in a resource calendar. This deliberate interruption in resource utilization (work break) can be used to relax continuity constraints. Applying work breaks properly in repetitive activities could result in an earlier start date of the repetitive activities, which in turn shortens project duration.

The application and the calculation of work breaks in this chapter focus on reducing project duration. The calculation of work breaks uses the concepts from the Repetitive Scheduling Method (RSM) and the Sequence Step Algorithm (SQS-AL). The concept of controlling sequence in RSM is used to determine work break position. In this chapter, the calculation is shown in detail with an example, including simulation model and code.

6.1 Introduction of Work Breaks

Scheduling repetitive activities with the Repetitive Scheduling Method (RSM) or the Sequence Step Algorithm (SQS-AL) is likely to increase project duration. These methods postpone repetitive activities from their early start date in order to keep resources working continuously, without interruption. These methods provide a schedule with minimized idle time.

Nevertheless, these methods result in lengthened project durations. In certain cases, cost savings from maintaining continuous resource utilization might not favorably

compensate for the penalty cost from the lengthened duration. Extending project duration from an early schedule should be carefully analyzed.

Figure 6.1 is a production diagram showing the early schedule for four activities (A, B, C, and D) that repeat over 4 identical units. The resource performing Activity B has a total idle time of 30 days from 3 interruptions ($Lag_{B1,B2}$, $Lag_{B2,B3}$, and $Lag_{B3,B4}$). Similarly, Activity D has a total idle time of 45 days, caused by 3 interruptions. Therefore, this early schedule (CPM) results in 75 days of total idle time, where project duration is 105 days.

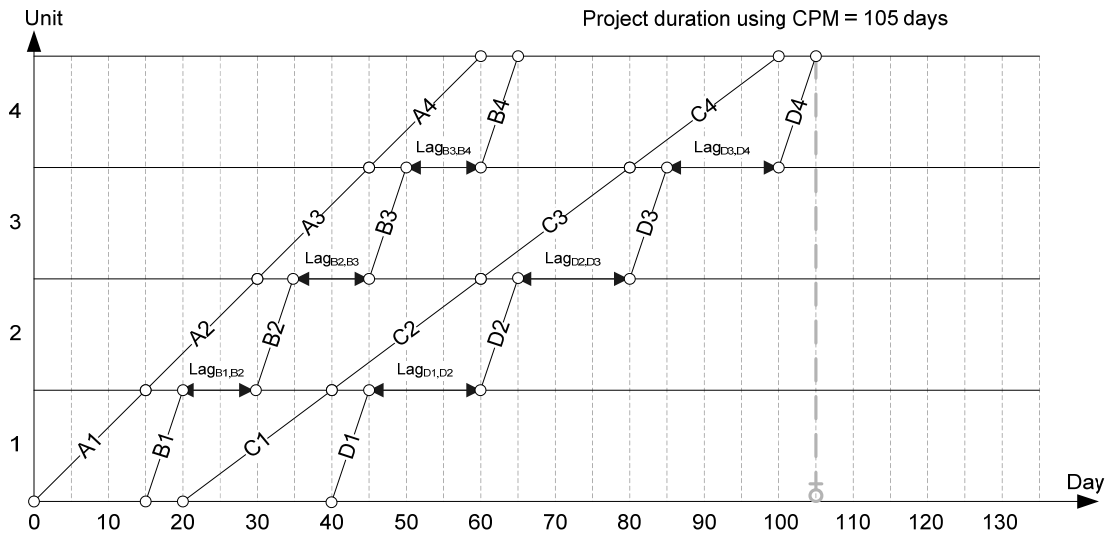


Figure 6.1 The CPM schedule with 105-day project duration and 75-day idle time

Figure 6.2 displays a schedule using RSM eliminating idle time by postponing Activities B and D from their early start dates. A comparison between Figure 6.1 (derived from CPM) and Figure 6.2 (derived from RSM) shows the resource continuity constraints in the RSM schedule eliminate all 75 days of resource idle time but increase project duration from 105 to 135 days.

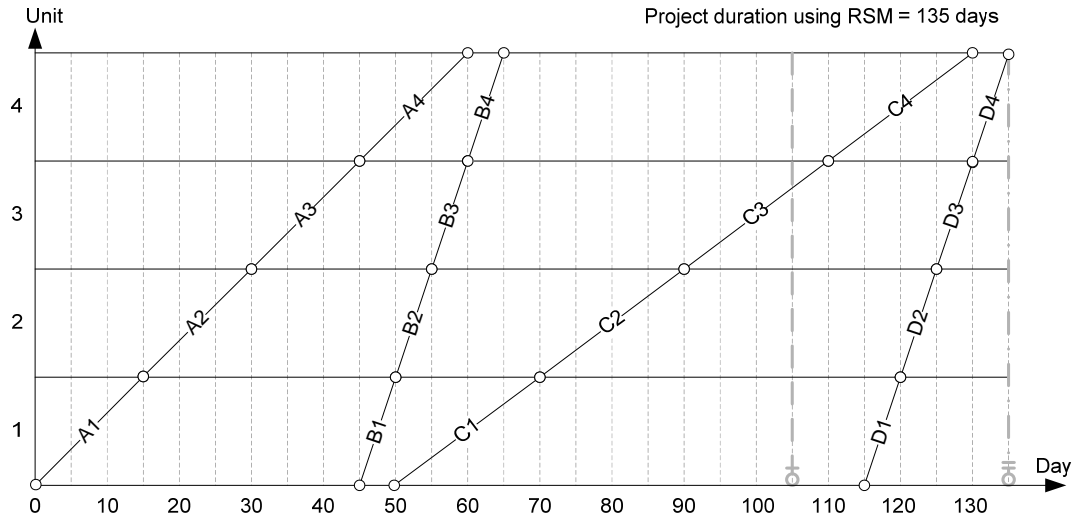


Figure 6.2 RSM schedule with an increased project duration from 105 to 135 days

Typically, the cost savings due to the elimination of resource idle time is far greater than the additional cost from the increased project duration. Yet, in certain cases this generalization is not always true. Delaying project completion may increase indirect project costs, lead to lost opportunities, and result in a lower overall profit. Accordingly, the tradeoff between eliminating resource idle time and increasing project duration must be analyzed thoroughly. The same holds for the possibility of eliminating idle time without increasing project duration.

One of the approaches that can be applied to RSM and SQS-AL in order to minimize cost and project duration is introducing work breaks in repetitive activities. A work break is a time period when a resource is scheduled to temporarily leave the project; resources do not earn wages during the work break. Remember, a work break is a scheduled interruption, while idle time is not.

During idle time, resources are on site and being paid, although they do not produce output. Accordingly, the costs associated with work breaks and idle time are completely different, as well as their resulting discontinuities. Work breaks usually incur

cost of transporting resources (labor and equipment), cost of hiring and firing labor, cost of setting up and dismantling equipment, etc. Additionally, the impact of setup time after work breaks must also be evaluated. This chapter focuses on the underlying tradeoff between project idle time and project duration.

Figure 6.3 illustrates the tradeoff between project idle time and project duration by introducing one work break in Activity B. In Figure 6.3, introducing a 20-day work break between B2 and B3 reduces project duration from 135 days (Figure 6.2) to 115 days (Figure 6.3). Notice that in order to maintain resource continuity between B1 and B2, the start of B1 must be postponed from its early start date of day 15 (Figure 6.1) to day 25 (Figure 6.3). Overall, the introduction of a 20-day work break in activity B reduces project duration also by 20 days, from 135 days in Figure 6.2 (the RSM schedule without a work break) to 115 days in Figure 6.3 (the RSM schedule with one work break). This schedule is only 10 days longer than the project duration of 105 days in the CPM schedule, while at the same time it eliminates the 75 days of resource idle time. Thus, the schedule with the work break results in a better solution, summarized in Table 6.1.

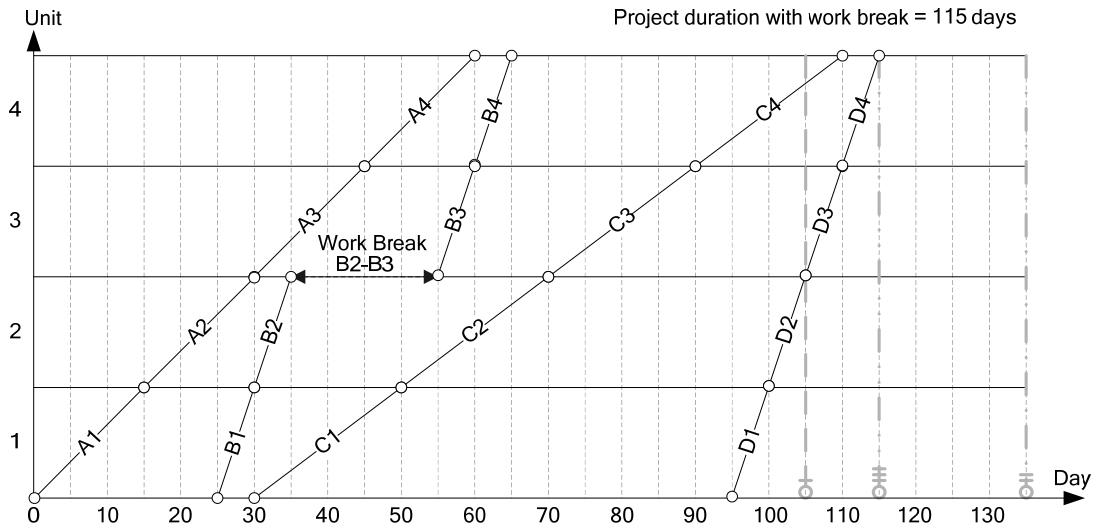


Figure 6.3 The work break B2-B3 reducing project duration from 135 to 115 days

Method	Project Duration (days)	Total Resource Idle Time (days)	Work Break Position
CPM	105	75	None
RSM	135	0	None
RSM	115	0	B2-B3

Table 6.1 Scheduling methods, idle time, project duration, and work break

From this example the following questions must be answered prior to scheduling work breaks:

- Which activities should be considered for the introduction of a work break (e.g., B)?
- Between which repetitive units should the work break be introduced (e.g., B2-B3)?
- What should be the start date of that activity in the first unit (e.g. B1)?
- How long should the duration of the work break be?

These questions will be answered in this chapter for both deterministic and probabilistic activity durations. An example project consisting of 9 activities with

probabilistic durations repeating over 10 similar work units is used for the purpose of demonstration.

6.2 Candidate Work Break Positions

Work break positions (e.g., B2-B3 in Figure 6.3) specify when resources will leave the site (e.g., the completion of B2 in Figure 6.3), and then return (e.g., the start of B3) to continue the work. To determine candidate work break positions, the following concepts derived from repetitive scheduling method (RSM) are used:

- 1) Control points and controlling sequences
- 2) Relative production rates

6.2.1 Control Points and Controlling Sequences

A control point between two repetitive activities is the precedence constraint that determines the start date of a succeeding repetitive activity under resource continuity constraints. In Figure 6.4, the control point between A and B is at the end of A3 and the start of B3 because it is the point that specifies the *earliest* start of repetitive Activity B, (day 35) under resource continuity constraints.

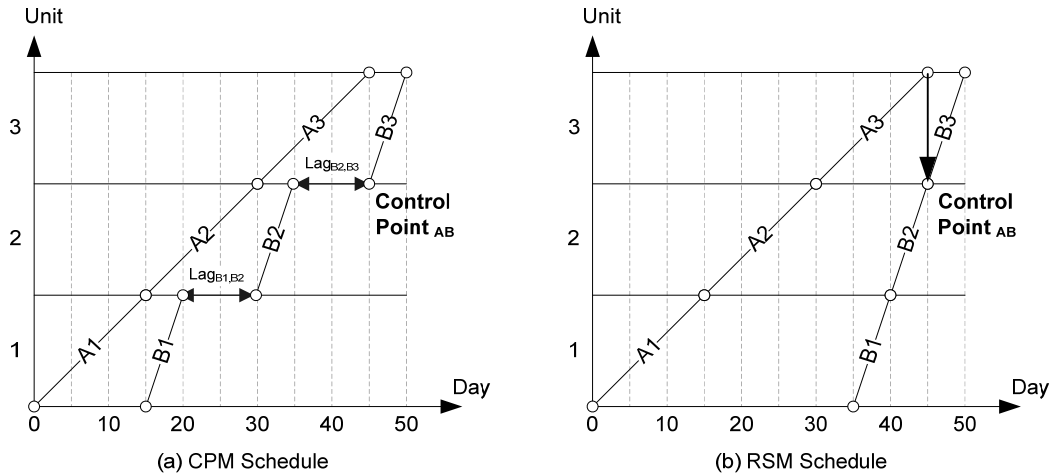


Figure 6.4 The control point between Activities A and B at A3-B3

Another example is shown in Figure 6.5. In the figure, the control point between A and B is at the end of A2 and the start of B2 since it specifies the *earliest* start date of repetitive Activity B (day 15 in Figure 6.5.b) under resource continuity constraints.

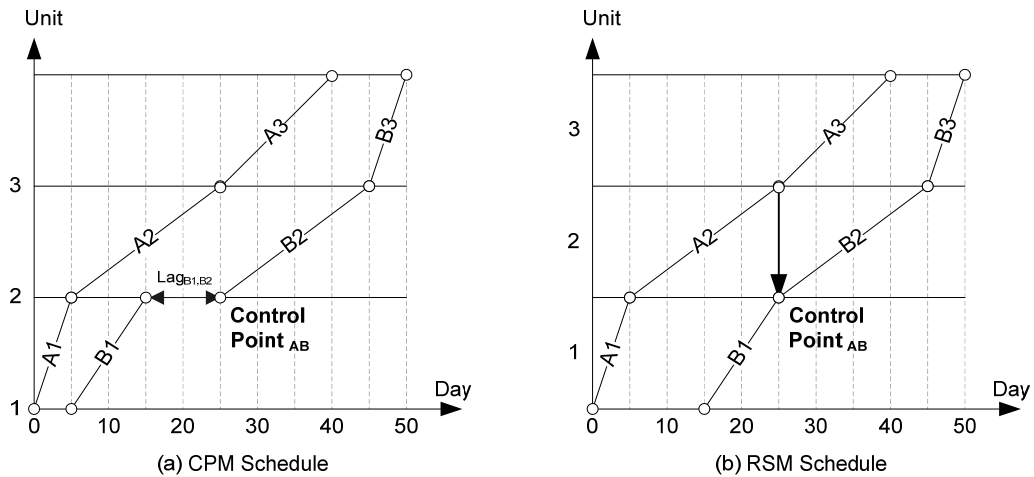


Figure 6.5 The control point between Activities A and B at A2-B2

A controlling sequence is a series of activities that controls project duration under resource continuity constraints. It can be determined visually in a production diagram by navigating from project completion to project start through control points. Figure 6.6 shows the control points and controlling sequence (in bold lines) from the example in

Figure 6.1. Identifying a controlling sequence of a repetitive project when activity durations are deterministic is relative easy. However, for activities with probabilistic durations, the means of identifying a controlling sequence is slightly complicated.

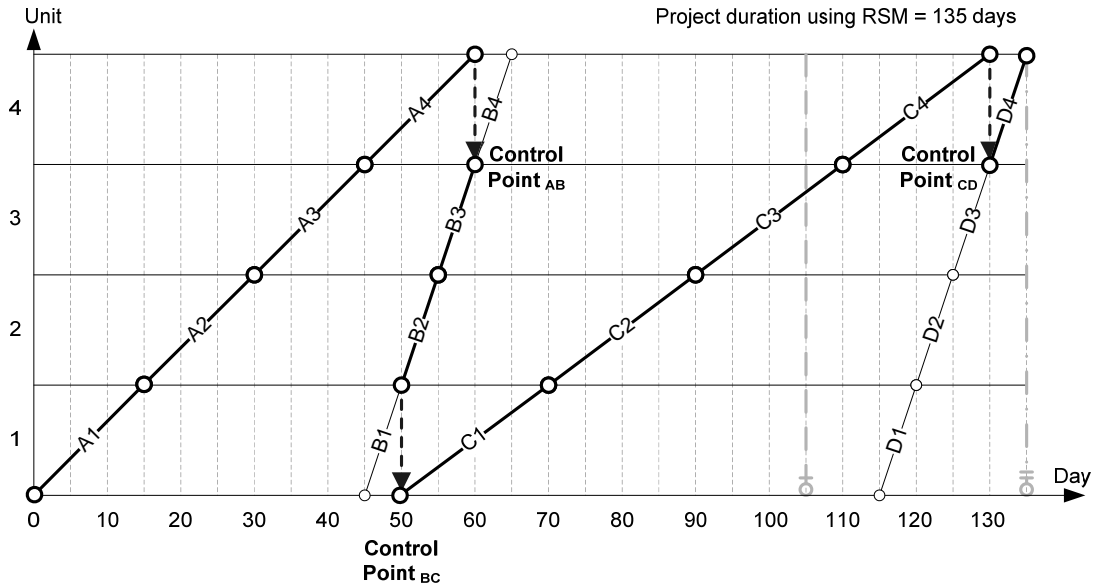


Figure 6.6 The controlling sequence (A1 to A4, B3 to B2, C1 to C4, and D4)

The implication of the controlling sequence, shown in Figure 6.6, reveals that the durations of activities on the controlling sequence determine the minimum project duration under resource continuity constraints. To shorten the project duration, work breaks are introduced in these activities to relax the continuity constraints. The conclusion from this observation is that *only repetitive activities on controlling sequences should be considered as candidates for introducing work breaks*. Activities not on the controlling sequences can be ignored.

For a project with probabilistic activity durations, lags between activities must be calculated before navigating from project completion to project start through the minimum lags between activities in order to identify controlling sequence. Figure 6.7, 6.8, and 6.9 are the results from three replications of an example project with

probabilistic activity durations. These results and explanations demonstrate the use of minimum lags in identifying controlling sequence and candidate work break positions.

Controlling sequence activities in Figure 6.7 are A1 to A4, B3 to B2, C1 to C4, and D4. Therefore, the candidate work break position for Activity A are A1-A2, A2-A3, A3-A4; for Activity B are B1-B2, B2-B3, B3-B4; for Activity C are C1-C2, C2-C3, C3-C4; and for activity D are D3-D4. Thus, there are 10 possible work break positions. Work break position B1-B2 is considered a possible work break since activity B2 is on the controlling sequence. This is also the same for position B3-B4 due to B3 and for position D3-D4 due to D4.

These work break positions are the candidate work breaks considering they are on controlling sequence. However, not all of them are effective work break positions. Effective work break positions will be discussed in Section 6.2.2, Relative Production Rates.

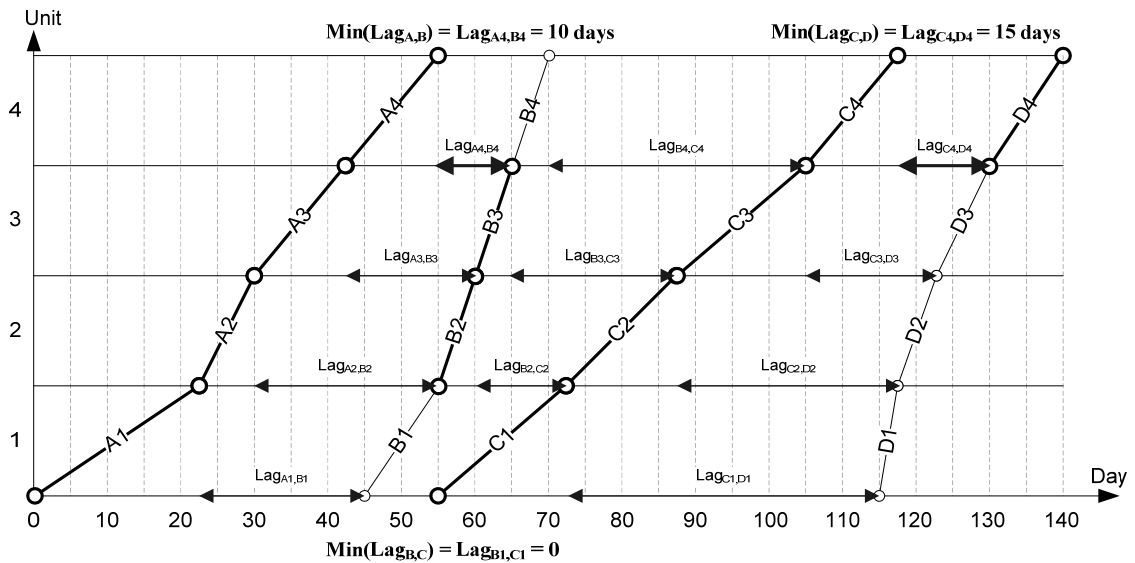


Figure 6.7 Determining the controlling sequence for the 1st replication

In Figure 6.8, controlling-sequence activities are A1 to A4, B3 to B2, C1 to C3, and D3 to D4. Therefore, the candidate work breaks are on Activity A are A1-A2, A2-A3, A3-A4; on Activity B are **B1-B2**, B2-B3, **B3-B4**; on Activity C are C1-C2, C2-C3, **C3-C4**; and on Activity D are **D2-D3**, D3-D4. Thus, there are 11 possible work break positions.

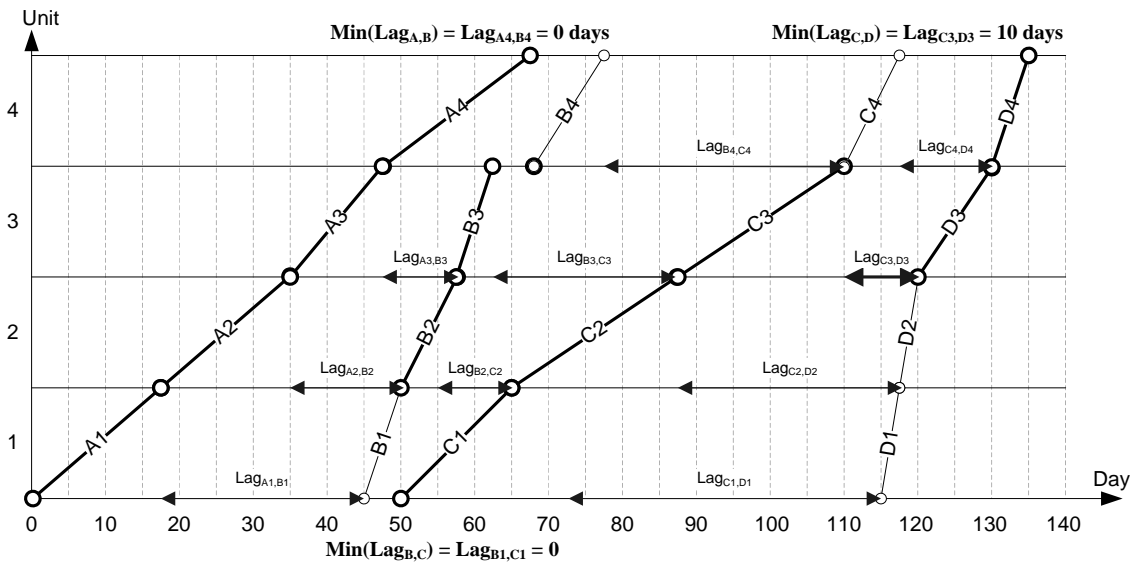


Figure 6.8 Determining the controlling sequence for the 2nd replication

In Figure 6.9, controlling-sequence activities are A1 to A2, B2, C2, and D2 to D4. Therefore, the candidate work breaks are on Activity A are A1-A2, A2-A3, A3-A4; on Activity B are B1-B2 and B2-B3; on Activity C are **C1-C2** and **C2-C3**; and on Activity D are **D1-D2**, D2-D3, D3-D4. Thus, there are 10 possible work break positions.

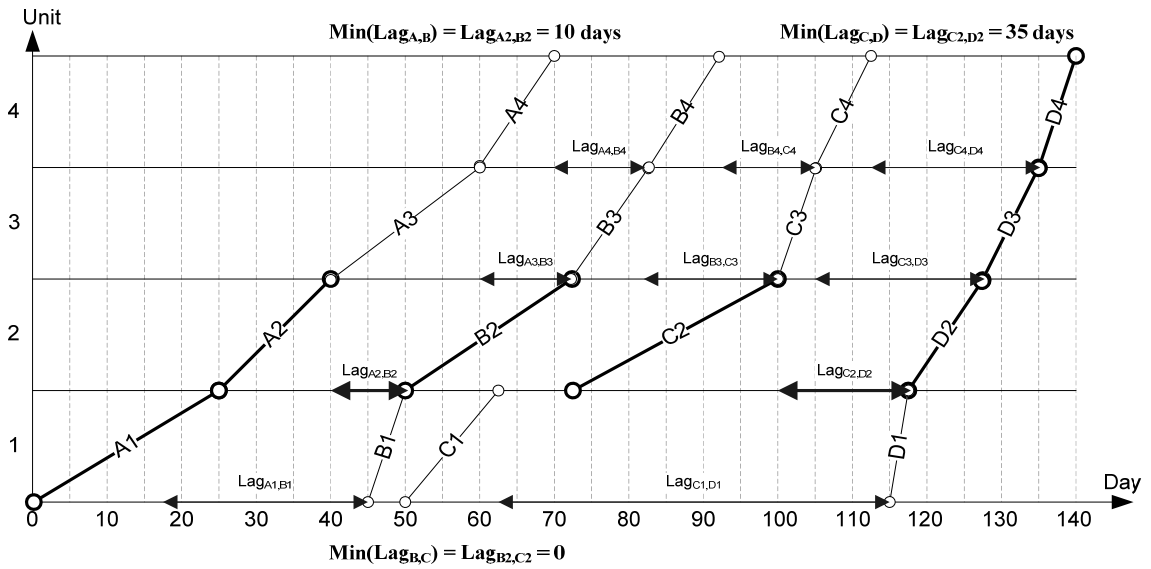


Figure 6.9 Determining the controlling Sequence for the 3rd replication

According to these three replications (Figures 6.7 to 6.9), the probability that activities will be on the controlling sequence are determined, shown in Table 6.2. Then, these probabilities can be used in order to determine work break positions.

Activity	# Controlling Sequence (out of 3 replications)	% Controlling Sequence
A1	3	100
A2	3	100
A3	3	100
A4	3	100
B1	0	0
B2	3	100
B3	2	66.66
B4	0	0
C1	2	66.66
C2	3	100
C3	2	66.66
C4	1	33.33
D1	0	0
D2	1	33.33
D3	2	66.66
D4	3	100

Table 6.2 Probability of activities on the controlling sequence

From Table 6.2, Activities A1 to A4, B2, C2, and D4 are always on the controlling sequence. Based on the probability that activities might be on the controlling sequence, the number of possible work break positions is reduced. Considering only controlling-sequence activities significantly reduces the number of candidate work break positions. Nevertheless, not every work break position on controlling-sequence activities could shorten project duration. Figure 6.6 validates this claim; introducing a work break on either controlling-sequence Activity A or C will lengthen rather than shorten the project. On the other hand, the introduction of a work break in Activity B does shorten project duration as shown in Figure 6.6.

The effectiveness of work breaks in shortening project duration depends on the relative production rates between activities on the controlling sequence as explained below.

6.2.2 Relative Production Rates

Positions of control points in the Repetitive Scheduling Method (RSM) are subject to the relative value of production rates between activities. According to RSM, the relative value of production rates between predecessors and successors is characterized as either *converging* or *diverging*.

A converging relationship occurs when the production rate of a successor is greater than that of its predecessor. The control point between the two repetitive activities is at the completion of the predecessor's last unit and the start of the successor's last unit. For example, in Figure 6.6, the production rate of B is greater than that of A, meaning Activities A and B have a converging relationship. Therefore, the control point between A and B is at the end of A4 and the start of B4.

A diverging relationship, on the other hand, occurs when the production rate of a successor is less than its predecessor. The control point between the two activities is at the end of the predecessor's first unit and the start of the successor's first unit. For example in Figure 6.6, the production rate of Activity C is less than that of B, meaning they have a diverging relationship. Therefore, the control point between B and C is at the end of B1 and the start of C1. The concepts of controlling sequence and relative production rates will be discussed further to identify and validate the effectiveness of candidate work break positions.

6.2.3 Determining Effective Work Break Positions

To confirm whether introducing a work break in a controlling-sequence activity can shorten the project, it is necessary to compare the production rate of that activity to the production rates of its direct predecessor and successor. Considering, for example, Activity B in Figure 6.6, the converging relationship between A and B and the diverging relationship between B and C enable a work break in B to shorten project duration. Therefore, a controlling-sequence activity having a converging relationship to its predecessor and a diverging relationship to its successor are the critical criteria in determining an effective work break in shortening project duration, given that the three activities (Activities A, B, and C in Figure 6.6) are on the same controlling sequence.

As shown in Figure 6.6, the converging relationship between Activities A and B allows Activities B1 and B2 to start earlier if a work break is introduced at the end of B2. Clearly, the relative production rate between A1-A2 and B1-B2 is still converging. The control point between A and B is at the end of A2 and the start of B2. Thus, the introduction of the work break in B creates a new control point that allows B1 to start

earlier. However, the fact that B1 is now scheduled earlier does not guarantee that the work break in B will result in shorter project duration. It is necessary to check the relative production rates between B and its successor on the same controlling sequence, i.e., Activity C.

The second requirement is the diverging relationship between B and C which enables Activity C to start and finish earlier if a work break is introduced to Activity B. Since the control point between B and C is at the start date of C1, starting B1 earlier moves the control point toward the project start date, resulting in an earlier start date for C1. Consequently, Activity C would finish sooner, resulting in shorter project duration.

It is important to realize that work breaks do not shorten the project duration directly; instead, they relax the continuity constraints, allowing controlling-sequence activities to start and finish sooner. For example in Figures 6.2 and 6.3, project duration is shortened because the controlling sequence Activity C starts earlier in Figure 6.3 than it does in Figure 6.2. Clearly, the reduction in project duration is due to the earlier start and finish date of Activity C, the successor to the activity to which the work break is introduced (i.e. Activity B).

In summary, an activity to which work break should be introduced must meet the following criteria.

- 1) It must be on a controlling sequence.
- 2) It must have a converging relationship with its direct predecessor on the same controlling sequence.
- 3) It must have a diverging relationship with its direct successor on the same controlling sequence.

The application of these rules reduces the number of candidate work break positions to just a handful. The next step is to determine work break duration for each work break position that fits the aforementioned criteria.

6.3 Determining Work Break Duration

After all the possible work break positions are filtered down to only those that may indeed shorten project duration, each candidate work break must be analyzed in order to derive a new schedule and project duration. For an activity with a work break, a new schedule must include 1) activity start date in the first unit and either 2.1) work break duration or 2.2) work break end date.

Work break duration specifies a fixed duration for work break. The sooner the resource completes the last unit before the break, the sooner it will return to the site; the later it completes the last unit, the later it will return.

On the other hand, work break end date specifies a fixed date when the work break is to end. A resource will always return on the same date. Accordingly, the work break duration for a fixed-date work break varies, depending on when the resource finishes the last unit prior to the break.

To construct a new project schedule for a particular work break position, the resource continuity constraints for that activity must be split into two sets: one before the work break and one after the work break. For example, in Figure 6.3, a work break is introduced at the end of B2. This means that Resource B is scheduled to work continuously from B1 to B2 and then takes a break at the end of B2. After the break, Resource B will return and work continuously from B3 to B4. Thus, for a deterministic problem,

Arrival date = {ESD} + {sum of idle times (lags) strictly before the work break}

Work break duration = {sum of idle times (lags) at and after the work break}

Therefore, Activity B1 must be postponed from its early start date by the sum of idle times before the work break ($Lag_{B1,B2}$) in order to achieve continuous resource utilization from Activity B1 to B2, as shown in Figure 6.3. After the completion of B2, the resource will take a break for a period equal to the sum of idle times at and after the work break location ($Lag_{B2,B3} + Lag_{B3,B4}$) to achieve continuity between Activities B3 and B4.

As a result from introducing a work break between Activities B2 and B3 (B2-B3), the increased project duration decreases from 135 days (Figure 6.2) to 115 days (Figure 6.3). For the example shown in Figure 6.2, it is obvious that the B2-B3 work break position minimizes project duration the most. However, each work break position must be tested individually. This is usually the case for projects with probabilistic activity durations.

To model probabilistic duration activities with a work break, crew idle time (CIT) is split into two: CIT before the break (CIT1) and CIT after the break (CIT2). CIT1 equals the sum of arrival idle time (AIT) and unit idle time before the break (UIT1), and CIT2 equals unit idle time at and after the break (UIT2). Therefore, the calculation for a fixed-duration work break is

$$CIT1 = \{idle\ time\ between\ arrival\ date\ and\ start\ date\ in\ the\ first\ unit\} \\ + \{sum\ of\ idle\ times\ (lags)\ strictly\ before\ the\ work\ break\}$$

which is

$$CIT1 = AIT + UIT1$$

For the duration of the work break (CIT2)

$$CIT2 \text{ (fixed duration)} = \{ \text{sum of idle times (lags) at and after the work break} \}$$

which is

$$CIT2 \text{ (fixed duration)} = UIT2$$

After CIT1 and CIT2 are collected from a number of replications, the corresponding crew lead times (CLT1 and CLT2 respectively) can be determined. These modifications of CIT and CLT are required in order to model work breaks and resource continuity constraints under the variability in probabilistic activity durations.

For example, assuming that the durations of activities in the previous example are probabilistic durations, when the Sequence Step Algorithm (SQS-AL) is processing SQS2, CIT1_B and CIT2_B are collected from each replication. These CITs are different from one replication to the next because of the variability in activity durations. After a user-specified number of replications are simulated, the collected samples of CIT1 and CIT2 are summarized in histograms based on relative frequency. At this point, the user does not have a single deterministic value of CIT1 and CIT2, which is used to determine CLT1 and CLT2, respectively. Instead the user must choose a desired confidence level (discussed in Chapter 4) to select the corresponding crew lead times for before and after the break (CLT1 and CLT2) from the corresponding histograms for CIT1 and CIT2.

For a fixed-duration work break, CLT1 is the start date of the activity in the first unit (e.g., B1), whereas CLT2 is the selected work break duration. After the completion of the last unit prior to the break (e.g. the finish date of B2), Resource B will take a break for a duration of CLT2_B. Delaying the work in the unit right after the break by CLT2_B,

ensures that when Resource B returns it should be able to work continuously and without idle time with probability equal to a chosen confidence level.

Using the calculation of fixed-duration work breaks, the work break has a fixed duration equal to CLT2, but the start and end of the break are uncertain. Nevertheless, in certain cases, it may be desirable to define CIT2 as a specific end date of work break. In order to determine such a specific return date, the calculation of CIT2 for a fixed-date work break is

$$CIT2 \text{ (fixed date)} = \{idle \text{ time between arrival date and start date in the first unit}\} \\ + \{sum \text{ of all idle times (lags)}\} + \{durations \text{ of activity before the break}\}$$

which is

$$CIT2 \text{ (fixed date)} = AIT + UIT1 + UIT2 + T1$$

where, T1 is the sum of activity durations before the break.

6.4 Example 6.1 Repetitive project with work breaks

The methodology for the optimal introduction of work breaks will be demonstrated using an example project comprised of 9 activities with probabilistic durations that repeat over 10 non-identical work units. The simulation model, the sequence step algorithm, and the introduction of work breaks have been implemented using the Stroboscope, discrete-event simulation system.

The precedence network for each of the 10 work units in the example project is shown in Figure 6.10. The quantities of work for each activity, however, are different in each unit as shown in Table 6.3. Moreover, each activity is performed by a different crew that has its own uncertain production rate. The production rate for each crew follows a normal distribution with the mean and standard deviation shown in Table 6.3. Thus,

activity durations vary from unit to unit because of different work quantities and because of the variability in production rates.

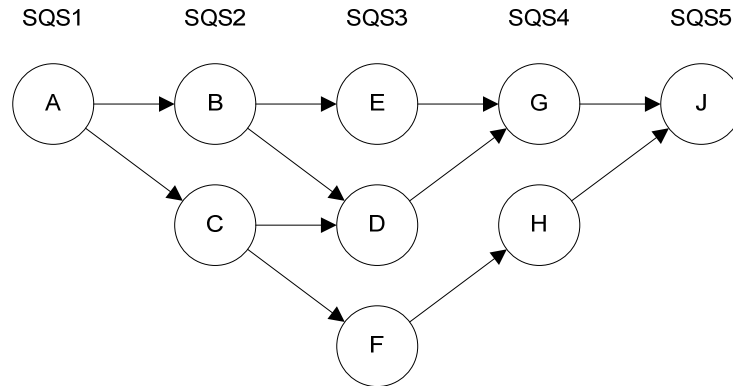


Figure 6.10 Single unit precedence diagram for Example 6.1

Activity Name	Resource Production Rate		Repetitive Unit										
	Mean	SD	1	2	3	4	5	6	7	8	9	10	
			Work Amount										
A	20	2.0	200	200	200	200	200	200	200	400	400	400	400
B	30	3.0	150	150	100	100	100	100	100	100	100	100	100
C	30	3.0	250	200	200	250	300	200	350	400	200	350	
D	15	1.5	300	400	400	450	300	300	250	250	250	400	
E	20	2.0	150	150	150	150	150	150	200	200	200	200	
F	25	2.5	350	400	300	350	150	200	400	250	300	250	
G	30	3.0	150	150	150	150	300	250	300	300	300	450	
H	20	2.0	200	300	300	200	250	400	300	400	300	250	
J	15	1.5	200	200	200	200	200	200	300	300	300	300	

Table 6.3 Daily production rates and activity work amounts for Example 6.1

For this example, 1000 replications were simulated for processing each SQS. A confidence level of 80% was used to choose CLT1 for activities without a work break and CLT1 and CLT2 for activities with a work break. In this example, CLT2, if required, represents a specific return date after a work break.

Figure 6.11 shows the production diagram of an early start schedule from the first replication (out of 1000 replications). Crews are assumed to arrive and start work in the first unit exactly when needed (no idle time) and activities begin as soon as their

predecessors in the same unit are completed. This produces the shortest possible average project duration of 277 days. As expected, allowing activities to start as early as possible results in a large average idle time, 438 days in Figure 6.11.

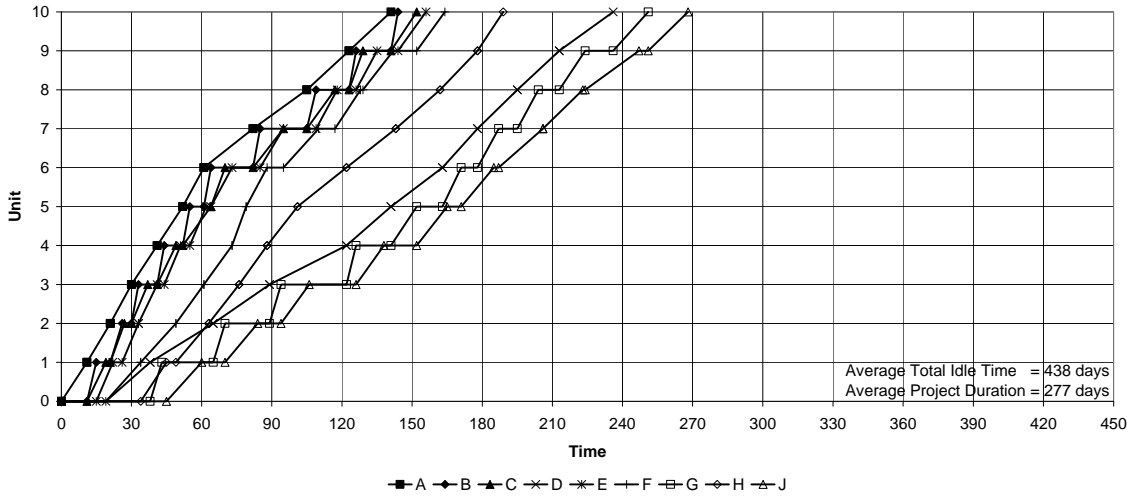


Figure 6.11 CPM schedule with 277-day project duration and 438-day idle time

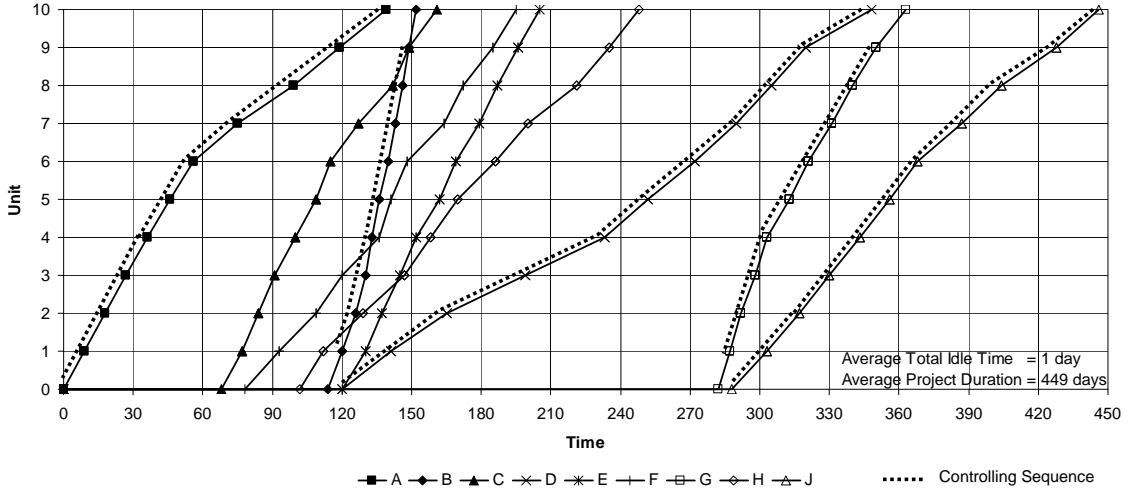


Figure 6.12 SQS-AL schedule without work breaks with 449-day project duration and 1-day idle time

Activity	Crew Arrival Date (CLT1)	Work Break Location	Crew Return Date (CLT2)
A	0	-	-
B	114	-	-
C	68	-	-
D	120	-	-
E	120	-	-
F	78	-	-
G	282	-	-
H	102	-	-
J	288	-	-

Table 6.4 CLT1 from the SQS-AL schedule without work breaks for Example 6.1

Figure 6.12 shows the project schedule derived from SQS-AL. To eliminate idle time, start dates of activities are delayed by their respective CLT1, as shown in Table 6.4, using a confidence level of 80%. The resulting average total idle time for the project is only 1 day but the average project duration has increased to 449 days. Thus, to eliminate resource idle time, project duration increased by 172 days or about 60%. It may be hard to justify to management or the project owner to delay project completion that much in order to eliminate idle time.

The decrease in project duration due to the introduction of one or more work breaks will now be investigated. As shown in Figure 6.12, there are 5 repetitive activities on the controlling sequence (A, B, D, G, and J) and only these activities are candidates for work breaks. Once the activities on the controlling sequence are identified, their relative production rates are compared to indicate activities that fit the aforementioned criteria. According to the criteria, only Activities B and G (in Figure 6.12) satisfy the two necessary conditions for shortening project duration, i.e., a converging relationship with their predecessor and a diverging relationship with their successor. Activity B, for

example, has a converging relationship with Activity A (its predecessor) and a diverging relationship with Activity C (its successor).

To determine the best work break position for Activity B, yielding the greatest decrease in project duration, nine possible positions are evaluated, which are at the finish dates of B1, B2, and so on until B9. For each candidate work break position, crew idle times before and after the prospective work break (CLT1 and CLT2) are calculated to determine the start date of B1 and the specific return date for the resource after the break.

The same process is performed for Activity G at the finish dates of G1, G2, and so on until G9. After all the candidate break positions (in Activities B and G) are evaluated, the results indicate that the two best work break locations in B and G are at the completion of B5 and G7, as shown in Figure 6.13. A work break at the completion of G7 shortens project duration by 39 days, whereas a work break at the completion of B5 shortens project duration by 34 days.

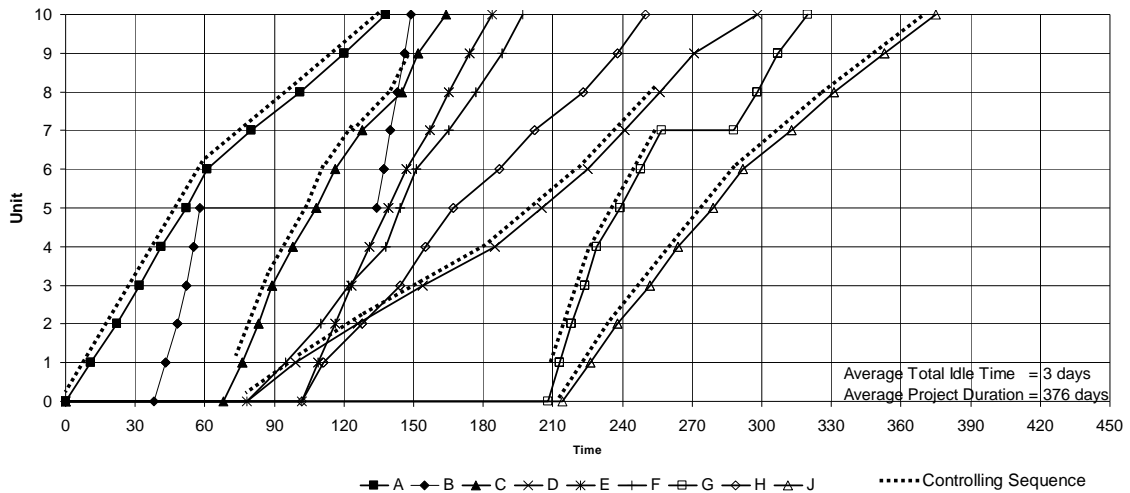


Figure 6.13 SQS-AL schedule with 2 work breaks with 376-day project duration for Example 6.1

Activity	Crew Arrival Date (CLT1)	Work Break Location	Crew Return Date (CLT2)
A	0	-	-
B	38	B5-B6	134
C	68	-	-
D	78	-	-
E	102	-	-
F	78	-	-
G	208	G7-G8	288
H	102	-	-
J	214	-	-

Table 6.5 CLT from the SQS-AL schedule with 2 work breaks for Example 6.1

Applying the two work breaks together shortens the average project duration by 73 days, from 449 to 376 days. The average total idle time increases slightly from 1 to 3 days. Table 6.5 shows CLT1 and CLT2 for each activity after introducing the B5-B6 and G7-G8 work breaks.

It is important to note that introducing a work break in Activity G does not change the status of Activity B on the controlling sequence, and vice versa. Consequently, work break positions satisfying the criteria remain the same, either applying a break in B or G first. Therefore, it is allowable to introduce work breaks in both activities in one step, without re-determining the controlling sequence. Otherwise, after introducing a break in Activity G only (as the most effective in reducing project duration), it is necessary to identify new candidate work break positions on the new controlling sequence.

In Figure 6.13, Resource B is scheduled to work continuously from B1 to the end of B5 and then take a break. After the break ends, Resource B will return to the site to start work on B6 on day 134. Resource G is scheduled to work continuously from G1 to the end of G7. Then resource G should take a break and come back on day 288. The work

break durations for B and G are not fixed, but depend on when the activities finish the unit prior to their respective breaks.

A comparison between Figures 6.12 and 6.13 shows Activity B is no longer on the controlling sequence after the introduction of the two work breaks. On the other hand, Activity G is still on the controlling sequence. For the new schedule, Activity C becomes an activity on the new controlling sequence, as shown in Figure 6.13.

A comparison of the relative production rates of the five activities on the new controlling sequence in Figure 6.13 indicates that only activities C and G have converging relationships with their predecessors and diverging relationships with their successors. Only candidate work break positions on Activity C are tested, since Activity G has just been evaluated for a work break in the previous step.

At this point it should be noted that sometimes the converging or diverging relationships between activities on the controlling sequence are not evident. For example in Figure 6.13, the converging relationship between Activity C and its predecessor, Activity A, is not apparent. In such cases, it is necessary to test the work breaks in C in order to determine whether they would shorten the project duration.

In Figure 6.13, there are nine possible work break locations for Activity C, i.e., at the end of C1, C2, and so on to the end of C9. The best work break position is at the completion of C4, shortening the project duration by 34 days. The resulting project schedule with work breaks in Activities B5, C4, and G7, is shown in Figure 6.14. The crew lead times, CLT1 and CLT2, for the three work break positions are displayed in Table 6.6. In summary, the three work breaks reduce the average project duration to 342 days with an average total idle time of 3 days. It should be noted that CLT2_G changes

from 288 days (Table 6.5) to 254 days (Table 6.6) due to the work break in Activity C. The work break at C4-C5 allows its successor activities (including G8) to start earlier, which in turn reduces project duration by 34 days.

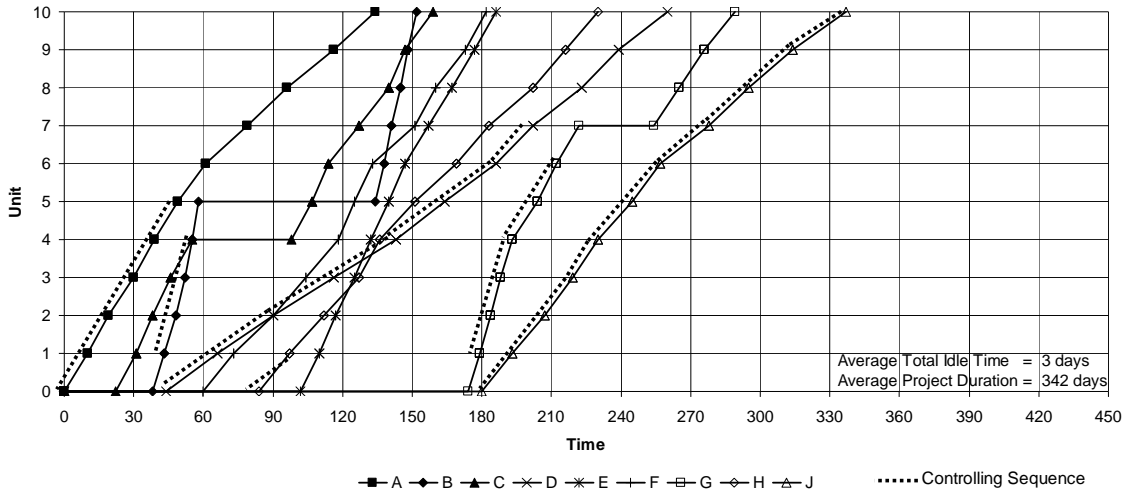


Figure 6.14 SQS-AL schedule with 3 work breaks and 342-day project duration for Example 6.1

Activity	Crew Arrival Date (CLT1)	Work Break Location	Crew Return Date (CLT2)
A	0	-	-
B	38	B5-B6	134
C	22	C4-C5	98
D	44	-	-
E	102	-	-
F	60	-	-
G	174	G7-G8	254
H	84	-	-
J	180	-	-

Table 6.6 CLT from the SQS-AL schedule with 3 work breaks for Example 6.1

Method	Average Project duration	Average Project Idle Time	Work Break Positions
CPM	277	438	-
SQS-AL	449	1	-
SQS-AL	410	3	G7
SQS-AL	376	3	G7,B5
SQS-AL	342	3	G7,B5,C4

Table 6.7 Finalized project duration and idle time for Example 6.1

Table 6.7 is a summary of the average project duration and the average total idle time from CPM, and SQS-AL with zero, one, two and three work breaks. Clearly, SQS-AL is effective in reducing the average crew idle time from 444 days to 1 day to save the cost associated with resource idle time. However, satisfying continuity constraints also increases project duration significantly from 277 to 449 days (from 9 to 15 months).

To reduce the increased project duration, relaxation of the continuity constraints using work breaks is an option. As illustrated in the example, applying work breaks reduces project duration significantly, while the change in project idle time is negligible. The introductions of work breaks at the end of G7, B5, and C4, reduce average project duration by 39, 34, and 34 days, respectively. In the final schedule, the three work breaks result in an average project duration of 342 days (11.5 months) with idle time of just 3 days. In other words, the average project duration increases by 2.5 months from the CPM schedule while the 15 months of average crew idle time is almost completely eliminated.

Moreover, a comparison between the schedules without work breaks (Table 6.4 and Figure 6.12) and the schedule with three work breaks (Table 6.6 and Figure 6.12) shows that the introduction of the work breaks favorably decreases the project duration by 3.5 months. Thus, the optimal introduction of work breaks is an effective strategy for shortening the increased project duration that results from resource continuity constraints.

6.5 Simulation Model and Code for Example 6.1

The finalized schedule with 3 work breaks from Example 6.1 is used to demonstrate the implementation of work breaks in SQS-AL and simulation. The example is implemented in Stroboscope; the simulation model and code in Stroboscope language are given with explanation. Several important aspects of the simulation model and SQS-AL are explained along with the given simulation code. The precedence diagram is presented in Figure 6.10, and activities' work amounts in each unit are given in Table 6.3. Production rates of activities are assumed to follow a normal distribution with the means and standard deviations displayed in Table 6.3. Figure 6.11 is the simulation model for this example. As can be seen, Figure 6.15 resembles Figure 6.10, a single unit precedence diagram. Simulation model templates are used to represent activities and resources, instead of nodes as in Figure 6.10.

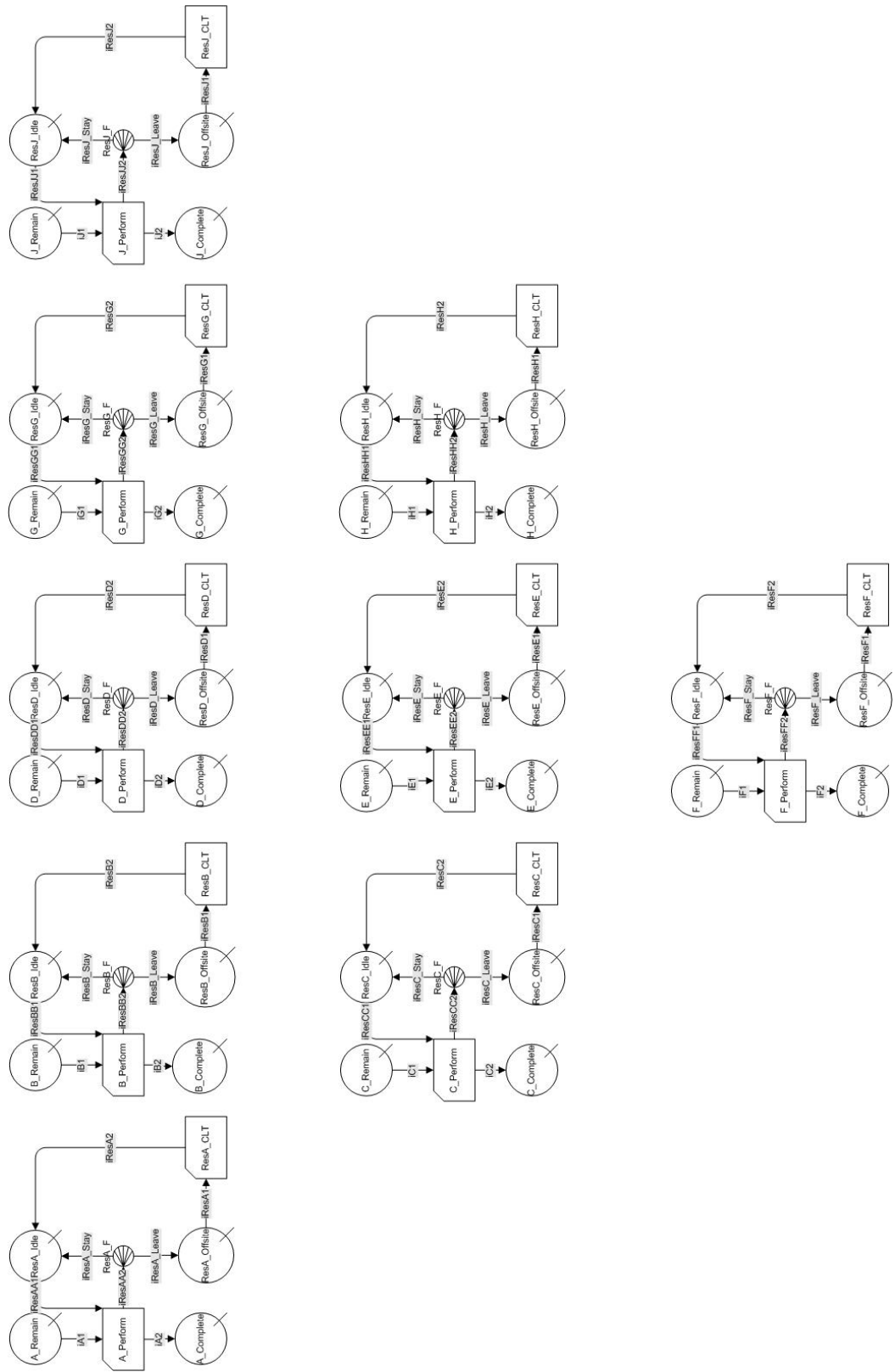


Figure 6.15 Simulation Model for the Example in Figure 6.10

6.5.1 Simulation Code for Model Parameters (MP)

The Simulation code in the following sections is encoded in Stroboscope GUI's Model Parameters (MP).

6.5.1.1 Variables controlling replication and sequence step loops (MP.Loops)

Variables that controls SQS-AL's replication loop and sequence step loop are stored in SaveValue, which is a storage class in Stroboscope language. The nRep is the total number of replications executed in processing each sequence step, whereas nSQS is the total number of sequence steps. ithRep and ithSQS indicate the current replication and processing sequence step, respectively.

```
SAVEVALUE nRep 1000;  
SAVEVALUE nSQS 5;  
SAVEVALUE ithSQS* 1;  
SAVEVALUE ithRep* 1;
```

Note that in Stroboscope the asterisk sign "*" indicates that the referred-to SaveValue (e.g., ithRep and ithSQS) is persistent throughout simulation runs, meaning the SaveValue with asterisk sign will not be reset to its initial value at the beginning of each replication. Since it is necessary to track the current replication (ithRep) and current sequence step (ithSQS), there SaveValues must be persistent and are declared so by the asterisk sign at the end of their names.

6.5.1.2 Work amounts (MP.ACT.Quantity)

For each activity, the work amount in each repetitive unit is stored in one-dimensional arrays. The number 10 in the code below is the total number of units in this project. The figures in parenthesis are the work amounts for each activity in each unit;

these figures initialize the member of the array. So each of the following statement defines and initializes the arrays holding the work amount for each activity.

```
ARRAY A_Quantity 10 {200 200 200 200 200 200 400 400 400 400};
ARRAY B_Quantity 10 {150 150 100 100 100 100 100 100 100 100};
ARRAY C_Quantity 10 {250 200 200 250 300 200 350 400 200 350};
ARRAY D_Quantity 10 {300 400 400 450 300 300 250 250 250 400};
ARRAY E_Quantity 10 {150 150 150 150 150 150 200 200 200 200};
ARRAY F_Quantity 10 {350 400 300 350 150 200 400 250 300 250};
ARRAY G_Quantity 10 {150 150 150 150 300 250 300 300 300 450};
ARRAY H_Quantity 10 {200 300 300 200 250 400 300 400 300 250};
ARRAY J_Quantity 10 {200 200 200 200 200 200 300 300 300 300};
```

Note that indexing arrays in the Stroboscope language is zero-based. Thus, the index of the first member in an array is zero, which is the same as in the C language.

6.5.1.3 Confidence levels (SMC.RES.ConfidenceLevel)

Confidence levels for each resource are stored in SaveValues shown below. For this example, an 80 percent confidence level is used for all resources.

```
SAVEVALUE ResA_ConfidenceLevel 0.8;
SAVEVALUE ResB_ConfidenceLevel 0.8;
SAVEVALUE ResC_ConfidenceLevel 0.8;
SAVEVALUE ResD_ConfidenceLevel 0.8;
SAVEVALUE ResE_ConfidenceLevel 0.8;
SAVEVALUE ResF_ConfidenceLevel 0.8;
SAVEVALUE ResG_ConfidenceLevel 0.8;
SAVEVALUE ResH_ConfidenceLevel 0.8;
SAVEVALUE ResJ_ConfidenceLevel 0.8;
```

6.5.1.4 Additional variables (MP.AdditionalVariable)

Two additional SaveValues are included and used later, iSQS and nthBinInterval. The iSQS is used to automatically generate Stroboscope code, whereas the nthBinInterval is used to determine corresponding crew lead time (CLT).

```
SAVEVALUE iSQS 1;  
  
SAVEVALUE nthBinInterval 0;
```

6.5.2 Simulation Code for Programming Objects (PO)

The Simulation code in the following subsections is encoded in Stroboscope GUI's Programming Objects (PO).

6.5.2.1 Permission to enter the site (PO.RES.Semaphore)

One important decision (i.e., RES_CLT_Semaphore) is whether to allow resources to enter the site (i.e., letting them in RES_CLT Combis) or keep them off the site (i.e., holding them in RES_Offsite Queues). Resources are allowed to enter the site only if their corresponding activities must not yet have been completed. For example, the number of units in ACT_Complete Queues must not equal the total number of units (4 units for this example). Without this decision (or condition), resources will always re-enter the site, even though all the work has been completed, and rest in RES_Idle until the simulation ends because there is no work left for them to work. The condition (RES_CLT_Semaphore) of allowing a resource to enter the site is stored in Variables, updated automatically by Stroboscope. It is encoded in Programming Objects, and used in RES_CLT's semaphore in resource flow sub-networks.

```
VARIABLE ResA_CLT_Semaphore ' A_Complete.CurCount != 10' ;  
  
VARIABLE ResB_CLT_Semaphore ' B_Complete.CurCount != 10' ;
```

```

VARIABLE ResC_CLT_Semaphore ' C_Complete.CurCount != 10 ' ;
VARIABLE ResD_CLT_Semaphore ' D_Complete.CurCount != 10 ' ;
VARIABLE ResE_CLT_Semaphore ' E_Complete.CurCount != 10 ' ;
VARIABLE ResF_CLT_Semaphore ' F_Complete.CurCount != 10 ' ;
VARIABLE ResG_CLT_Semaphore ' G_Complete.CurCount != 10 ' ;
VARIABLE ResH_CLT_Semaphore ' H_Complete.CurCount != 10 ' ;
VARIABLE ResJ_CLT_Semaphore ' J_Complete.CurCount != 10 ' ;

```

6.5.2.2 Precedence constraints (PO.ACT.Semaphore)

Precedence constraints (i.e., RES_Perform_Semaphore) are stored in Variables and used in ACT_Perform's semaphore work flow sub-networks. Due to precedence constraints, an activity (e.g., Activity B) can start only if its current number of completed units (e.g., B_Complete.CurCount) plus its current number of on-going units (e.g., B_Perform.CurInst) is less than its predecessor's number of completed units (e.g., A_Complete.CurCount). For this example, all precedence constraints are shown below:

```

VARIABLE B_Perform_Semaphore
    'B_Complete.CurCount + B_Perform.CurInst
    < A_Complete.CurCount' ;
VARIABLE C_Perform_Semaphore
    'C_Complete.CurCount + C_Perform.CurInst
    < A_Complete.CurCount ;
VARIABLE D_Perform_Semaphore
    'D_Complete.CurCount + D_Perform.CurInst
    < B_Complete.CurCount
    &
    D_Complete.CurCount + D_Perform.CurInst
    < C_Complete.CurCount' ;
VARIABLE E_Perform_Semaphore

```

```

        'E_Complete.CurCount + E_Perform.CurInst
        < B_Complete.CurCount' ;

VARIABLE F_Perform_Semaphore

        'F_Complete.CurCount + F_Perform.CurInst
        < C_Complete.CurCount' ;

VARIABLE G_Perform_Semaphore

        'G_Complete.CurCount + G_Perform.CurInst
        < D_Complete.CurCount
        & G_Complete.CurCount + G_Perform.CurInst
        < E_Complete.CurCount' ;

VARIABLE H_Perform_Semaphore

        'H_Complete.CurCount + H_Perform.CurInst
        < F_Complete.CurCount' ;

VARIABLE J_Perform_Semaphore

        'J_Complete.CurCount + J_Perform.CurInst
        < G_Complete.CurCount
        &
        J_Complete.CurCount + J_Perform.CurInst
        < H_Complete.CurCount' ;

```

6.5.2.3 Activity duration (PO.ACT.Duration)

Activity durations (e.g., A_Perform_Duration) are stored in Variables, and used in ACT_Perform's duration in work flow sub-networks. ACT_Perform's duration varies from unit to unit due to amount of work in each unit and variability in production rates. Therefore, activity durations are a function of work amount of the current unit and production rate. The duration of ACT_Perform Combis is essentially the work amount in a unit divided by the production rate of that activity. Duration variables of activities are shown below:

```

VARIABLE A_Perform_Duration
    A_Quantity[10-A_Remain.CurCount]*1/Normal[20,2];
VARIABLE B_Perform_Duration
    B_Quantity[10-B_Remain.CurCount]*1/Normal[30,3];
VARIABLE C_Perform_Duration
    C_Quantity[10-C_Remain.CurCount]*1/Normal[30,3];
VARIABLE D_Perform_Duration
    D_Quantity[10-D_Remain.CurCount]*1/Normal[15,1.5];
VARIABLE E_Perform_Duration
    E_Quantity[10-E_Remain.CurCount]*1/Normal[20,2];
VARIABLE F_Perform_Duration
    F_Quantity[10-F_Remain.CurCount]*1/Normal[25,2.5];
VARIABLE G_Perform_Duration
    G_Quantity[10-G_Remain.CurCount]*1/Normal[30,3];
VARIABLE H_Perform_Duration
    H_Quantity[10-H_Remain.CurCount]*1/Normal[20,2];
VARIABLE J_Perform_Duration
    J_Quantity[10-J_Remain.CurCount]*1/Normal[15,1.5];

```

6.5.2.4 Decision of keeping or laying off resource (PO.RES.Strength)

Decisions of keeping or laying off resources are stored in variables (i.e., RES_Leave_Strength) and used in resource flow sub-networks. After a resource completes one unit of work, it will consider whether to leave the site. If there is no remaining work for the resource, it will leave. The resources' decision whether to leave or stay of resources is shown below:

```

VARIABLE ResA_Leave_Strength '(A_Remain.CurCount == 0) ? 1:0 ' ;
VARIABLE ResD_Leave_Strength '(D_Remain.CurCount == 0) ? 1:0 ' ;
VARIABLE ResE_Leave_Strength '(E_Remain.CurCount == 0) ? 1:0 ' ;

```

```
VARIABLE ResF_Leave_Strength '(F_Remain.CurCount == 0) ? 1:0 ' ;
```

For resources with work breaks, additional variables (e.g., ResB_nthBreak) below are required to track how many times the resources have already left the site. These variables are used to determine 1) when resources are allowed to leave the site and 2) which crew lead time (CLT1 or CLT2) will be used, if the resources enter the site.

```
SAVEVALUE ResB_nthBreak 0;
```

```
SAVEVALUE ResC_nthBreak 0;
```

```
SAVEVALUE ResG_nthBreak 0;
```

For resources with work breaks, the decisions of keeping or laying off resources are shown below. For example, ResB is allowed to leave the site when:

- 1) Five units of B have been completed. Then ResB will take the break.
- 2) Ten units of B have been completed. Then ResB will leave the site. At this point, ResB has already taken one break (ResB_nthBreak = 1) after completing B5.

```
VARIABLE ResB_Leave_Strength  
      '(B_Complete.CurCount == 5 & ResB_nthBreak == 0)  
      | (B_Remain.CurCount == 0 )  
      ? 1:0 ' ;
```

For ResC, it will take a break at the completion of Activity C in Unit 4, and it will permanently leave the project at the end of Activity C in Unit 10.

```
VARIABLE ResC_Leave_Strength  
      '(C_Complete.CurCount == 4 & ResC_nthBreak == 0)  
      | (C_Remain.CurCount == 0 )  
      ? 1:0 ' ;
```

For ResG, it will take a break at the completion of Activity G in Unit 7, and it will permanently leave the project at the end of Activity G in Unit 10.


```

VARIABLE ResG_Leave_Strength
      ' (G_Complete.CurCount == 7 & ResG_nthBreak == 0)
      | (G_Remain.CurCount == 0)
      ? 1:0 ' ;

```

6.5.2.5 Temporary SaveValues for crew idle time (PO.RES.TempIdleTime)

The following SaveValues are for temporarily recording the total idle time in each resource in each replication.

```

SAVEVALUE svResB_Idle 0;
SAVEVALUE svResC_Idle 0;
SAVEVALUE svResD_Idle 0;
SAVEVALUE svResE_Idle 0;
SAVEVALUE svResF_Idle 0;
SAVEVALUE svResG_Idle 0;
SAVEVALUE svResH_Idle 0;
SAVEVALUE svResJ_Idle 0;

```

At the end of each replication, these SaveValues (temporary storages) of idle time will be assigned to their corresponding BinCollectors (permanent storages) for a specific sequence step (see 6.5.2.8 PO.RES.CIT.SQS). After the assignment, these SaveValues are, then, reset prior to the execution of a new replication.

6.5.2.6 Crew idle time (PO.RES.CIT)

Samples of crew idle time (CLTs) used to determine CLT are stored in BinCollectors. BinCollectors are data holders that keep statistics of the numbers they receive in intervals, specified by users. Every resource that is scheduled by SQS-AL to achieve continuous utilization must have at least one BinCollector, which is RES_CLT1.

BINCOLLECTOR statement requires 4 arguments, which are: 1) the name of the collector, 2) the number of intervals, 3) the lower bound of the collected data, and 4) the upper bound of the collected data. For this example, RES_CIT1 BinCollectors collect CITs from a range of 0 to 300 days. This range is divided into 300 intervals, meaning that CIT samples are grouped on a daily basis.

Since crew idle time of resources is collected from one replication to another, the data in RES_CIT1 and RES_CIT2 BinCollectors must be persistent, and thus an asterisk, “*”, is required when their names are declared. Resources without work breaks (ResA, ResD, ResE, ResF, ResH, and ResJ) have only CIT1 BinCollector, used to determine CLT1, as shown below.

```
BINCOLLECTOR ResD_CIT1* 300 0 300 ;  
BINCOLLECTOR ResE_CIT1* 300 0 300 ;  
BINCOLLECTOR ResF_CIT1* 300 0 300 ;  
BINCOLLECTOR ResG_CIT1* 300 0 300 ;  
BINCOLLECTOR ResH_CIT1* 300 0 300 ;  
BINCOLLECTOR ResJ_CIT1* 300 0 300 ;
```

Resources with work breaks (ResB, ResC, and ResG) have CIT1 and CIT2 BinCollectors that are used to determine their CLT1 and CLT2, respectively. Moreover, the last time resources leave the site must be recorded for calculation purposes in order to determine CLT2. See Section 6.5.3.7 (CME.RES.Leave.OnFlow) for the calculation of CLT2.

For ResB,

```
BINCOLLECTOR ResB_CIT1* 300 0 300 ;  
BINCOLLECTOR ResB_CIT2* 300 0 300 ;  
SAVEVALUE tempResB_CIT1 0 ;
```

```
SAVEVALUE tempResB_CIT2 0 ;  
SAVEVALUE ResB_LastTimeLeaveSite 0 ;
```

For ResC,

```
BINCOLLECTOR ResC_CIT1* 300 0 300 ;  
BINCOLLECTOR ResC_CIT2* 300 0 300 ;  
SAVEVALUE tempResC_CIT1 0 ;  
SAVEVALUE tempResC_CIT2 0 ;  
SAVEVALUE ResC_LastTimeLeaveSite 0 ;
```

For ResG,

```
BINCOLLECTOR ResG_CIT1* 300 0 300 ;  
BINCOLLECTOR ResG_CIT2* 300 0 300 ;  
SAVEVALUE tempResG_CIT1 0 ;  
SAVEVALUE tempResG_CIT2 0 ;  
SAVEVALUE ResG_LastTimeLeaveSite 0 ;
```

6.5.2.7 Crew lead time (PO.RES.CLT.Duration)

The crew lead time (CLT) for each resource is stored in a persistent SaveValue (requiring an asterisk sign after its name), and used to set the duration of RES_CLT Combis in resource flow sub-network. CLT's initial value is zero according to the sequence step algorithm. The RES_CLT_Duration Variables are created mainly to systemize the simulation code. They are used in RES_CLT Combis, discussed in Section 6.5.3.2, CME.RES.CLT.Duration.

The arrival date of resources without work breaks equals their CLT1, shown in the simulation code below.

```
SAVEVALUE ResD_CLT1* 0 ;  
SAVEVALUE ResE_CLT1* 0 ;
```

```

SAVEVALUE ResF_CLT1* 0;

SAVEVALUE ResH_CLT1* 0;

SAVEVALUE ResJ_CLT1* 0;

VARIABLE ResD_CLT_Duration ResD_CLT1;

VARIABLE ResE_CLT_Duration ResE_CLT1;

VARIABLE ResF_CLT_Duration ResF_CLT1;

VARIABLE ResH_CLT_Duration ResG_CLT1;

VARIABLE ResJ_CLT_Duration ResG_CLT1;

```

Crew lead times (CLT1 or CLT2) for resources with work breaks are chosen, based on their RES_nthBreak Variables (e.g., ResB_nthBreak), to delay the arrival of the resources. For example, if ResB has never taken a break (ResB_nthBreak = 0), CLT1_B will be chosen to delay the arrival of ResB, when ResB attempts to enter the site. On the other hand, if ResB has already taken one work break (ResB_nthBreak = 1), CLT2_B will be chosen to delay the 2nd arrival of ResB.

```

VARIABLE ResB_CLT_Duration
    'ResB_nthBreak == 0 ? ResB_CLT1 :
    ResB_nthBreak == 1 ? ResB_CLT2 : 0' ;

VARIABLE ResC_CLT_Duration
    'ResC_nthBreak == 0 ? ResC_CLT1 :
    ResC_nthBreak == 1 ? ResC_CLT2 : 0' ;

VARIABLE ResG_CLT_Duration
    'ResG_nthBreak == 0 ? ResG_CLT1 :
    ResG_nthBreak == 1 ? ResG_CLT2 : 0' ;

```

6.5.2.8 Crew idle time for each sequence step (PO.RES.CIT.SQS)

The code below for BinCollectors for crew idle times (CITs) associated with sequence steps (SQS) is for the purpose of recording the changes in CITs from one SQS

to another. Note that the `iSQS` in the code below is used with “`$<...>$`” (explained later) to automatically create simulation code.

```
ASSIGN iSQS 1;
WHILE 'iSQS<=nSQS+1';
  BINCOLLECTOR bcltResA_IdleSQS$iSQS>*300 0 300 ;
  BINCOLLECTOR bcltResB_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltResC_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltResD_IdleSQS$iSQS>*300 0 300 ;
  BINCOLLECTOR bcltResE_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltResF_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltResG_IdleSQS$iSQS>*300 0 300 ;
  BINCOLLECTOR bcltResH_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltResJ_IdleSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltProjectDurationSQS$iSQS>* 300 0 300 ;
  BINCOLLECTOR bcltProjectIdleTimeSQS$iSQS>* 300 0 300 ;
  ASSIGN iSQS iSQS+1;
WEND; /iSQS
ASSIGN iSQS 1;
```

Note that the “`$<Argument>$`” is a preprocessor operator of Stroboscope used to automatically generate simulation code. The Stroboscope evaluates the argument in the operator, and replaces the operator and argument before executing the statement. This preprocessor operator is useful when there is a consistent pattern of simulation code (Martinez 1995). For more information about the “`$<Argument>$`” operator, see *The Stroboscope Simulation Language*, Chapter 15, Statement Preprocessing and Automatic Code Generation.

6.5.3 Simulation Code for Model Elements (CME)

The simulation code in this section 5.3.3 is encoded in graphical model elements in Stroboscope GUI. For the code in this section, the statements specify attributes of simulation elements in Stroboscope GUI where the code is stored. These statements are:

- SEMAPHORE for Semaphore Combis
- DURATION for Duration in Combis
- STRENGTH for Strength in links
- ONFLOW for OnFlow in links

6.5.3.1 Semaphore in RES_CLT Combis (CME.RES.Semaphore)

The Semaphores of RES_CLT Combis are shown below. They are stored in simulation model elements, as shown in Figure 5.12. Details for these semaphores are given in Section 6.5.2.1 PO.RES.Semaphore.

```
SEMAPHORE ResB_CLT ResB_CLT_Semaphore;  
SEMAPHORE ResC_CLT ResC_CLT_Semaphore;  
SEMAPHORE ResD_CLT ResD_CLT_Semaphore;  
SEMAPHORE ResE_CLT ResE_CLT_Semaphore;  
SEMAPHORE ResF_CLT ResF_CLT_Semaphore;  
SEMAPHORE ResG_CLT ResG_CLT_Semaphore;  
SEMAPHORE ResH_CLT ResH_CLT_Semaphore;  
SEMAPHORE ResJ_CLT ResJ_CLT_Semaphore;
```

6.5.3.2 Duration in RES_CLT Combis (CME.RES.CLT.Duration)

Durations of RES_CLT Combis in resource flow are shown below. Details of these durations are given in section 6.5.2.7, PO.RES.CLT.Duration.

```
DURATION ResA_CLT ResA_CLT_Duration;
```

```

DURATION ResB_CLT ResB_CLT_Duration;
DURATION ResC_CLT ResC_CLT_Duration;
DURATION ResD_CLT ResD_CLT_Duration;
DURATION ResE_CLT ResE_CLT_Duration;
DURATION ResF_CLT ResF_CLT_Duration;
DURATION ResG_CLT ResG_CLT_Duration;
DURATION ResH_CLT ResH_CLT_Duration;
DURATION ResJ_CLT ResJ_CLT_Duration;

```

6.5.3.3 Semaphore in ACT_Perform Combis (CME.ACT.Semaphore)

Semaphores of ACT_Perform Combis in work flow are shown below. They are stored in graphical simulation elements, as shown in Figure 5.13. Details of these semaphores are given in Section 6.5.2.7, PO.RES.CLT.Duration.

```

SEMAPHORE A_Perform A_Perform_Semaphore;
SEMAPHORE B_Perform B_Perform_Semaphore;
SEMAPHORE C_Perform C_Perform_Semaphore;
SEMAPHORE D_Perform D_Perform_Semaphore;
SEMAPHORE E_Perform E_Perform_Semaphore;
SEMAPHORE F_Perform F_Perform_Semaphore;
SEMAPHORE G_Perform G_Perform_Semaphore;
SEMAPHORE H_Perform H_Perform_Semaphore;
SEMAPHORE J_Perform J_Perform_Semaphore;

```

6.5.3.4 Duration in ACT_Perform Combis (CME.ACT.Duration)

Durations of ACT_Perform Combis in work flow sub-networks are shown below. These ACT_Perform_Duration Variables are stored in simulation model elements, as shown in Figure 5.13. Details of these duration Variables are given in section 6.5.2.3 PO.ACT.Duration.

```

DURATION A_Perform A_Perform_Duration;
DURATION B_Perform B_Perform_Duration;
DURATION C_Perform C_Perform_Duration;
DURATION D_Perform D_Perform_Duration;
DURATION E_Perform E_Perform_Duration;
DURATION F_Perform F_Perform_Duration;
DURATION G_Perform G_Perform_Duration;
DURATION H_Perform H_Perform_Duration;
DURATION J_Perform J_Perform_Duration;

```

6.5.3.5 Strength in iRES_Stay Links (CME.RES.Stay.Strength)

Strengths of iRES_Stay links in work flow sub-networks are shown below. They are stored in simulation model elements, as shown in Figure 5.14. Details of these strengths are given in Section 5.3.2.4, PO.RES.Strength. Note that RES_Leave_Strength Variables are preceded by an exclamation sign returning the opposite value of the Variables. Accordingly, the opposite of a resource leaving the site is the resource staying on the site.

```

STRENGTH iResA_Stay !ResA_Leave_Strength;
STRENGTH iResB_Stay !ResB_Leave_Strength;
STRENGTH iResC_Stay !ResC_Leave_Strength;
STRENGTH iResD_Stay !ResD_Leave_Strength;
STRENGTH iResE_Stay !ResE_Leave_Strength;
STRENGTH iResF_Stay !ResF_Leave_Strength;
STRENGTH iResG_Stay !ResG_Leave_Strength;
STRENGTH iResH_Stay !ResH_Leave_Strength;
STRENGTH iResJ_Stay !ResJ_Leave_Strength;

```


6.5.3.6 Strength in iRES_Leave Links (CME.RES.Leave.Strength)

As opposed to the strength in iRES_Stay Links, strength of iRES_Leave is RES_Leave_Strength Variable. Details of these strengths are given in Section 6.5.2.4, PO.RES.Strength.

```
STRENGTH iResA_Leave ResA_Leave_Strength;  
STRENGTH iResB_Leave ResB_Leave_Strength;  
STRENGTH iResC_Leave ResC_Leave_Strength;  
STRENGTH iResD_Leave ResD_Leave_Strength;  
STRENGTH iResE_Leave ResE_Leave_Strength;  
STRENGTH iResF_Leave ResF_Leave_Strength;  
STRENGTH iResH_Leave ResH_Leave_Strength;  
STRENGTH iResJ_Leave ResJ_Leave_Strength;
```

6.5.3.7 OnFlow in iRES_Leave (CME.RES.Leave.OnFlow)

Crew idle time of resources is collected when SQS-AL is processing the sequence step of the activities they serve. For example, CIT1 of Resource D is collected from simulation runs during processing SQS3 because Activity D is in SQS2.

The simulation code, determining CLT, for resources without work breaks (ResD, ResE, ResF, and ResJ) is similar to Section 5.3.3.7 (CME.RES.Leave.OnFlow)

```
ONFLOW iResD_Leave COLLECT ResD_CIT1  
    PRECOND 'ithSQS==3' ResD_Idle.AveWait*ResD_Idle.TotCount;  
ONFLOW iResE_Leave COLLECT ResE_CIT1  
    PRECOND 'ithSQS==3' ResE_Idle.AveWait*ResE_Idle.TotCount;  
ONFLOW iResF_Leave COLLECT ResF_CIT1  
    PRECOND 'ithSQS==3' ResF_Idle.AveWait*ResF_Idle.TotCount;  
ONFLOW iResH_Leave COLLECT ResH_CIT1  
    PRECOND 'ithSQS==4' ResH_Idle.AveWait*ResH_Idle.TotCount;
```

```
ONFLOW iResJ_Leave COLLECT ResJ_CIT1
      PRECOND 'ithSQS==4' ResJ_Idle.AveWait*ResJ_Idle.TotCount;
```

The simulation code for the resources with work breaks (ResB, ResC, and ResG) are modified as follows.

For ResB,

```
ONFLOW iResB_Leave ASSIGN ResB_nthBreak ResB_nthBreak+1;
/CLT for Break B-5
ONFLOW iResB_Leave COLLECT ResB_CIT1
      PRECOND 'ithSQS==ResB_CLT1_SQS & ResB_nthBreak ==1'
      ResB_Idle.AveWait*ResB_Idle.TotCount ;
ONFLOW iResB_Leave ASSIGN tempResB_CIT1
      PRECOND 'ithSQS==ResB_CLT1_SQS & ResB_nthBreak ==1'
      ResB_Idle.AveWait*ResB_Idle.TotCount ;
/CLT for Break B-10
ONFLOW iResB_Leave COLLECT ResB_CIT2
      PRECOND 'ithSQS==ResB_CLT2_SQS & ResB_nthBreak==2'
      'ResB_Idle.AveWait*ResB_Idle.TotCount -
      tempResB_CIT1+ResB_LastTimeLeaveSite' ;
ONFLOW iResB_Leave ASSIGN tempResB_CIT2
      PRECOND 'ithSQS==ResB_CLT2_SQS & ResB_nthBreak==2'
      'ResB_Idle.AveWait*ResB_Idle.TotCount - tempResB_CIT1' ;
ONFLOW iResB_Leave ASSIGN ResB_LastTimeLeaveSite SimTime;
```

For ResC,

```
/CLT for ARRIVAL of ResC
ONFLOW iResC_Leave ASSIGN ResC_nthBreak ResC_nthBreak+1;
/CLT for Break C-4
ONFLOW iResC_Leave COLLECT ResC_CIT1
```

```

        PRECOND 'ithSQS==ResC_CLT1_SQS & ResC_nthBreak ==1'
        ResC_Idle.AveWait*ResC_Idle.TotCount ;
ONFLOW iResC_Leave ASSIGN tempResC_CIT1
        PRECOND 'ithSQS==ResC_CLT1_SQS & ResC_nthBreak ==1'
        ResC_Idle.AveWait*ResC_Idle.TotCount ;
/CLT for Break C-10
ONFLOW iResC_Leave COLLECT ResC_CIT2
        PRECOND 'ithSQS==ResC_CLT2_SQS & ResC_nthBreak==2'
        'ResC_Idle.AveWait*ResC_Idle.TotCount -
        tempResC_CIT1+ResC_LastTimeLeaveSite' ;
ONFLOW iResC_Leave ASSIGN tempResC_CIT2
        PRECOND 'ithSQS==ResC_CLT2_SQS & ResC_nthBreak==2'
        'ResC_Idle.AveWait*ResC_Idle.TotCount - tempResC_CIT1' ;
ONFLOW iResC_Leave ASSIGN ResC_LastTimeLeaveSite SimTime;
For ResG,
/CLT for ARRIVAL of ResG
ONFLOW iResG_Leave ASSIGN ResG_nthBreak ResG_nthBreak+1;
/CLT for Break G-7
ONFLOW iResG_Leave COLLECT ResG_CIT1
        PRECOND 'ithSQS==ResG_CLT1_SQS & ResG_nthBreak ==1'
        ResG_Idle.AveWait*ResG_Idle.TotCount ;
ONFLOW iResG_Leave ASSIGN tempResG_CIT1
        PRECOND 'ithSQS==ResG_CLT1_SQS & ResG_nthBreak ==1'
        ResG_Idle.AveWait*ResG_Idle.TotCount ;
/CLT for Break G-10
ONFLOW iResG_Leave COLLECT ResG_CIT2

```

```

PRECOND 'ithSQS==ResG_CLT2_SQS & ResG_nthBreak==2'
'ResG_Idle.AveWait*ResG_Idle.TotCount -
tempResG_CIT1+ResG_LastTimeLeaveSite' ;
ONFLOW iResG_Leave ASSIGN tempResG_CIT2
PRECOND 'ithSQS==ResG_CLT2_SQS & ResG_nthBreak==2'
'ResG_Idle.AveWait*ResG_Idle.TotCount - tempResG_CIT1' ;
ONFLOW iResG_Leave ASSIGN ResG_LastTimeLeaveSite SimTime;

```

6.5.4 Simulation Code for Controlling Statements (CS)

Simulation code in the following sections is encoded in Stroboscope GUI's Controlling Statements (CS).

6.5.4.1 Sequence step and replication loops (CS.Loops)

The following two While-Loops statements control SQS-AL's sequence step loop and replication loop, respectively. As discussed in Chapters 4 and 5, the replication loop is for collecting crew idle times of resources whose activities are in the current processing sequence step (ithSQS). The sequence step loop is for determining crew lead time for the resources. An extra sequence step is added to obtain the final results of project duration, project idle time, and idle time in resource utilization.

```

WHILE 'ithSQS <= nSQS+1';           / Start Sequence Step Loop.
    WHILE 'ithRep <= nRep ';         / Start Replication Loop.
        CLEAR;                       / Clear temporary data.

```

Crew lead times for all resources are determined when SQS-AL finishes processing the last sequence step. Nevertheless, an extra sequence step is added to obtain the final results of project duration, project idle time, and idle time in resource utilization.

The CLEAR statement is executed at the beginning of each replication to clear the results from a previous simulation run. This statement sets all non-persistent SaveValues to their initial values and clears all data and statistics from non-persistent Collectors and BinCollectors.

6.5.4.2 Initializing work amounts (CS.ACT.INIT)

At the beginning of each replication, the work amount in units for each activity is initialized in ACT_Remain Queues in Work Flow Networks. INIT is a Stroboscope's statement used to create and place resources in specified Queues. As discussed in section 5.2.1 Work Flow Template, the amount of Resource rq_ACT in ACT_Remain is the number of remaining units needed to be completed, which is 10 as shown in the code below.

```
INIT A_Remain 10;  
INIT B_Remain 10;  
INIT C_Remain 10;  
INIT D_Remain 10;  
INIT E_Remain 10;  
INIT F_Remain 10;  
INIT G_Remain 10;  
INIT H_Remain 10;  
INIT J_Remain 10;
```

6.5.4.3 Initializing resources (CS.RES.INIT)

At the beginning of each replication, resources are initialized in RES_Offsite Queues in resource flow sub-networks.

```
INIT ResA_Offsite 1;  
INIT ResB_Offsite 1;
```

```
INIT ResC_Offsite 1;  
INIT ResD_Offsite 1;  
INIT ResE_Offsite 1;  
INIT ResF_Offsite 1;  
INIT ResG_Offsite 1;  
INIT ResH_Offsite 1;  
INIT ResJ_Offsite 1;
```

6.5.4.4 Executing simulation (CS.Simulate)

After initializing the work amount in ACT_Remain and the resources in RES_Offsite, the simulation starts after Stroboscope executes the SIMULATE statement. Then, the simulation replication will end when all activities are completed, i.e., when there is no rq_ACT resource in each and every one of the ACT_Remain Queues.

```
SIMULATE; /Run one replication
```

Note that crew idle times of resources are collected during simulation runs, while crew lead time is determined after the end of processing each sequence step.

6.5.4.5 Recording CIT in beltRES_IdleSQS BinCollectors(CS.RES.CIT.SQS)

At the end of the specified number of replications for each sequence step, the following data is collected.

- 1) Crew idle times of resources for processing each sequence step
- 2) Project duration for processing each sequence step
- 3) Sum of crew idle times and project idle time for processing each sequence step

The collected data are for tracking changes in crew idle times, project durations, and project idle times. These data are valuable for analysis of the impact of an assigned

crew lead time on activities, resources, and the project. The following code is for collecting the data responding to each sequence step processing.

```
ASSIGN svResB_Idle ResB_Idle.AveWait*ResB_Idle.TotCount;
ASSIGN svResC_Idle ResC_Idle.AveWait*ResC_Idle.TotCount;
ASSIGN svResD_Idle ResD_Idle.AveWait*ResD_Idle.TotCount;
ASSIGN svResE_Idle ResE_Idle.AveWait*ResE_Idle.TotCount;
ASSIGN svResF_Idle ResF_Idle.AveWait*ResF_Idle.TotCount;
ASSIGN svResG_Idle ResG_Idle.AveWait*ResG_Idle.TotCount;
ASSIGN svResH_Idle ResH_Idle.AveWait*ResH_Idle.TotCount;
ASSIGN svResJ_Idle ResJ_Idle.AveWait*ResJ_Idle.TotCount;
ASSIGN iSQS 1;
WHILE 'iSQS<=nSQS+1';
    IF 'ithSQS==<iSQS>';
        COLLECT bcltResB_IdleSQS<iSQS>$ svResB_Idle;
        COLLECT bcltResC_IdleSQS<iSQS>$ svResC_Idle;
        COLLECT bcltResD_IdleSQS<iSQS>$ svResD_Idle;
        COLLECT bcltResE_IdleSQS<iSQS>$ svResE_Idle;
        COLLECT bcltResF_IdleSQS<iSQS>$ svResF_Idle;
        COLLECT bcltResG_IdleSQS<iSQS>$ svResG_Idle;
        COLLECT bcltProjectDurationSQS<iSQS>$ SimTime;
        COLLECT bcltProjectIdleTimesSQS<iSQS>$
            'svResA_Idle+ svResB_Idle+ svResC_Idle
            +svResD_Idle+ svResE_Idle+ svResF_Idle
            +svResG_Idle';
    ENDIF;
    ASSIGN iSQS iSQS+1;
WEND; /iSQS
ASSIGN ithRep ithRep+1; / Increase ithRep by 1
```

```
WEND; /ithRep
```

The last line of the code is the end of SQS-AL's replication loop. When Stroboscope reaches this line, it will check whether the current replication index (ithRep) is less than or equal to the total number of replications (nRep), explained in section 6.5.4.1, CS.Loops. SQS-AL exits the replication loop, and enters the sequence step loop, when the ithRep SaveValue is greater than nRep SaveValue.

Remember that iSQS is used for coding purpose, while ithSQS is used to track the current index of sequence step. For the code above, the small nested loop controlled by iSQS is not the sequence step loop.

6.5.4.6 Determining crew lead time for activities belonging to ithSQS (CS.RES.CLT)

After SQS-AL exits replication loop and enters sequence step loop, the crew lead time of the resources serving activities in the current processing sequence step will be determined. For example, after the total number of replications is executed in processing SQS2, SQS-AL determines CLT of Resource B serving Activity B in SQS2, respectively. See Section 5.3.4.6 in Chapter 5 for the explanation of the simulation code.

The following simulation code is for the resources with work breaks (ResB, ResC, and ResG). Each of these resources has two CLTs, CLT1 (for the arrival date) and CLT2 (for the return date after the break). Therefore, there are two set of loops, determining CLT1 and CLT2.

For CLT1_B,

```
IF 'ithSQS==2';  
    ASSIGN nthBinInterval 0;  
    WHILE PctAtOrBelowBin[ResB_CIT1,nthBinInterval]  
        <ResB_ConfidenceLevel;
```



```

        ASSIGN nthBinInterval nthBinInterval+1;

    WEND;

    ASSIGN ResB_CLT1 BinHigh[ResB_CIT1,nthBinInterval];

ENDIF;

For CLT2B,

IF 'ithSQS==2';

    ASSIGN nthBinInterval 0;

    WHILE PctAtOrBelowBin[ResB_CIT2,nthBinInterval]
        <ResB_ConfidenceLevel;

        ASSIGN nthBinInterval nthBinInterval+1;

    WEND;

    ASSIGN ResB_CLT2 BinHigh[ResB_CIT2,nthBinInterval];

ENDIF;

For CLT1C,

IF 'ithSQS==2';

    ASSIGN nthBinInterval 0;

    WHILE PctAtOrBelowBin[ResC_CIT1,nthBinInterval]
        <ResC_ConfidenceLevel;

        ASSIGN nthBinInterval nthBinInterval+1;

    WEND;

    ASSIGN ResC_CLT1 BinHigh[ResC_CIT1,nthBinInterval];

ENDIF;

For CLT2C,

IF 'ithSQS==2';

    ASSIGN nthBinInterval 0;

    WHILE PctAtOrBelowBin[ResC_CIT2,nthBinInterval]
        <ResC_ConfidenceLevel;

        ASSIGN nthBinInterval nthBinInterval+1;

```

```

        WEND;

        ASSIGN ResC_CLT2 BinHigh[ResC_CIT2,nthBinInterval];
ENDIF;

For CLT1G,

IF 'ithSQS==4';

    ASSIGN nthBinInterval 0;

    WHILE PctAtOrBelowBin[ResG_CIT1,nthBinInterval]
        <ResG_ConfidenceLevel;

        ASSIGN nthBinInterval nthBinInterval+1;

    WEND;

    ASSIGN ResG_CLT1 BinHigh[ResG_CIT1,nthBinInterval];

ENDIF;

For CLT2G,

IF 'ithSQS==4';

    ASSIGN nthBinInterval 0;

    WHILE PctAtOrBelowBin[ResG_CIT2,nthBinInterval]
        <ResG_ConfidenceLevel;

        ASSIGN nthBinInterval nthBinInterval+1;

    WEND;

    ASSIGN ResG_CLT2 Round[BinHigh[ResG_CIT2,nthBinInterval],0];

ENDIF;

```

The following simulation code is for resources without work breaks (ResD, ResE, ResF, ResH, and ResJ), having only CLT1. Detail of the code is explained in Section 5.3.4.6, CS.RES.CLT.

```

For CLT1D,

IF 'ithSQS==3';

    ASSIGN nthBinInterval 0;

```

```

        WHILE PctAtOrBelowBin[ResD_CIT,nthBinInterval]
            <ResD_ConfidenceLevel;
            ASSIGN nthBinInterval nthBinInterval+1;
        WEND;
        ASSIGN ResD_CLT BinHigh[ResD_CIT,nthBinInterval];
    ENDIF;

For CLT1E,
IF 'ithSQS==3';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResE_CIT,nthBinInterval]
        <ResE_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResE_CLT BinHigh[ResE_CIT1,nthBinInterval];
ENDIF;

For CLT1F,
IF 'ithSQS==3';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResF_CIT,nthBinInterval]
        <ResF_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResF_CLT BinHigh[ResF_CIT1,nthBinInterval];
ENDIF;

For CLT1H,
IF 'ithSQS==4';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResH_CIT,nthBinInterval]

```

```

        <ResH_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResH_CLT BinHigh[ResH_CIT,nthBinInterval];
ENDIF;
For CLT1j,
IF 'ithSQS==5';
    ASSIGN nthBinInterval 0;
    WHILE PctAtOrBelowBin[ResJ_CIT,nthBinInterval]
        <ResJ_ConfidenceLevel;
        ASSIGN nthBinInterval nthBinInterval+1;
    WEND;
    ASSIGN ResJ_CLT BinHigh[ResJ_CIT,nthBinInterval];
ENDIF;
ASSIGN ithSQS ithSQS+1;
WEND; / ithSQS
REPORT;

```

The last line of the code is the end of sequence step loop. When Stroboscope reaches this line, it will check whether the current index of the sequence step (ithSQS) is greater than the total number of sequence steps plus one (nSQS+1). If ithSQS is not greater than nSQS+1, SQS-AL will begin processing the next SQS. In addition, this example project with work breaks is also modeled using ChaStrobe, presented in Chapter 8, Example 8.2.

6.6 Summary

To minimize idle time, the Sequence Step Algorithm (SQS-AL) postpones the arrival date of resources. Results from minimizing idle time usually provide cost reduction and productivity improvement. However, project duration may be lengthened significantly by scheduling resources to work continuously. Therefore, tradeoffs between maintaining and relaxing resource continuity constraints must be carefully studied in terms of project cost and duration.

This chapter presents an application and calculation of work breaks in repetitive projects. Work breaks can be used to relax resource continuity constraints in repetitive activities, which in turn may shorten project duration. Work breaks, deliberate interruptions, are not considered idle time, since work breaks are carefully predetermined as part of the project schedule. Resources are informed of the breaks in advance; therefore, they can be allocated to other projects at the beginning of the breaks and return when the breaks end.

Under uncertainty, work breaks relax the resource continuity constraints, while still minimizing crew idle time. Optimal work breaks relaxing resource continuity constraints can be determined by using the concepts of control points, the controlling sequence, and relative production rates, proposed by Harris and Ioannou (1998). The concepts are discussed in this chapter and also in Chapter 3, Repetitive Scheduling Method.

To determine effective work breaks, positions and durations of work breaks must be carefully analyzed and calculated. The following rules offer useful guidance in determining candidate positions of work breaks:

- 1) Controlling Sequences and activities on the sequence must be determined prior to considering applying a work break.
- 2) Only repetitive activities on the controlling sequence are considered as candidates for work break positions.
- 3) The repetitive activities in (2) must have a converging relationship with their direct predecessor on the same controlling sequence.
- 4) The repetitive activities in (2) must have a diverging relationship with their direct successor on the same controlling sequence.
- 5) When there is more than one possible work break position, it is necessary to test all the possible positions.

To test the effectiveness of a work break position, SQS-AL is applied with the split activity treated as two separate activities, and duration for each candidate work break position is calculated. Then, each work break (position and duration) is applied to the project schedule to check the changes in project duration and idle time. The best work break position should result in the greatest decrease in project duration and a small increase project idle time.

For an activity with a work break, a new schedule must include both 1) activity start date in the first unit and either 2.1) a work break duration or 2.2) a work break end date. Fixed-duration work breaks are used when schedulers want resources to take a break for a certain period. On the other hand, fixed-date work breaks are used when schedulers want resources to return to the site on a specific date.

To schedule an activity with a work break, resource continuity constraints are split into two sets: before the work break and after the work break. The crew lead time before work break (CLT1) can be derived from

$$CIT1 = \{idle\ time\ between\ arrival\ date\ and\ start\ date\ in\ the\ first\ unit\} \\ + \{sum\ of\ idle\ times\ (lags)\ strictly\ before\ the\ work\ break\}$$

For crew lead time after work break, two calculations for two types of work breaks are presented below.

For a fixed-duration work break,

$$CIT2\ (fixed\ duration) = \{sum\ of\ idle\ times\ (lags)\ at\ and\ after\ the\ work\ break\}$$

For a fixed-date work break,

$$CIT2\ (fixed\ date) = \{idle\ time\ between\ arrival\ date\ and\ start\ date\ in\ the\ first\ unit\} \\ + \{sum\ of\ all\ idle\ times\ (lags)\} + \{durations\ of\ activity\ before\ the\ break\}$$

In this chapter, an application of work breaks in repetitive projects is presented with examples. The examples provide evidence that applying work breaks could shorten project duration while maintaining resource continuity constraints. The implementation of work breaks in simulation using the concepts of SQS-AL is demonstrated and attests the versatility of SQS-AL and the simulation model templates.

CHAPTER 7

RESOURCE-SHARING ACTIVITIES

This chapter discusses the scheduling of resources that serve more than one activity in a repetitive project. These types of resources serving many activities are called “shared resources,” while resources serving only one activity are called “dedicated resources.” Activities sharing one or more resources are called “resource-sharing activities.” Shared resources can be scheduled in various ways depending on the activities they serve. It is possible that a resource-sharing activity might use one dedicated resource solely for itself and also share a shared resource with another activity. Without shared resources in the simulation model, the dedicated resource can be scheduled using the Sequence Step Algorithm (SQS-AL) discussed so far to satisfy resource continuity constraints. However, when shared resources are included in the simulation model, violations in resource continuity constraints for both dedicated and shared resources may be incurred.

Satisfying resource continuity constraints for shared resources is quite complicated. The continuity constraints of shared resources are not affected only by the resource-sharing activities’ precedence constraints. They could also be compromised by the precedence and continuity constraints of the predecessors and successors of the resource-sharing activities. This includes both direct and indirect predecessors and

successors. Apparently, the calculation of crew idle time and crew lead time may not adequately tackle this highly dynamic and complicated utilization of shared resources under continuity constraints. Many considerations and careful analysis are required to solve this challenging problem.

To schedule resource-sharing activities and shared resources effectively, relationships between resource-sharing activities and their predecessors and successors must be accounted for in scheduling. The order of which activities are scheduled first among resource-sharing activities and their dependents may result in different schedules, as well as efficiency. These additional considerations of scheduling order must be incorporated in the schedule in order to successfully maintain continuous utilization of resources in the project. In many cases, maintaining resource continuous utilization of shared resources may not be possible, depending on the relationships and relative production rates between resource-sharing activities and their dependents.

This chapter uses several examples to explore many possible situations that exist in scheduling repetitive projects with resource-sharing activities. Suggestions and solutions are given and vary from case to case. There is no substitute for careful analysis and sound judgment. Examining and testing variation of schedules are mandatory so that potential problems, ineffectiveness and conflict in the schedules are foreseen and corrected. Especially for repetitive projects with resource-sharing activities, corrective modification in project schedules is usually required to improve the schedules or alleviate existing problems and ineffectiveness.

7.1 Considerations in Scheduling Resource-Sharing Activities

Many characteristics of activities, resources, and discrete-event simulation have a great influence on the effectiveness of the sequence step algorithm in scheduling repetitive projects with resource-sharing activities. The following characteristics outline imperative considerations that must be taken into account in scheduling such projects:

- The precedence relationships between activities sharing the same resource, either directly or indirectly dependent.
- The sequence steps of resource-sharing activities, either in the same or different sequence steps.
- The sequence steps of resource-sharing activities' direct predecessors and successors.
- The order in which each activity is modeled in the simulation model.
- The processing sequence step in which resources' crew lead time (CLT) is determined.
- The relative order that CLT of shared resources serving resource-sharing activities and the order that CLT of dedicated resources serving the dependents of the resource-sharing activities are determined.
- The idle time in the early start date schedule and the changes in the idle time as SQS-AL proceeds from the first to the last sequence step.
- The impact from assigning shared resources' CLT on the idle time and start date of their activities and their activities' dependents.

7.2 Examples of Repetitive Projects with Resource-Sharing Activities X and Y

The following examples demonstrate different problems that schedulers may encounter when using the sequence step algorithm to schedule repetitive projects with resource-sharing activities.

Activities X and Y in the following examples are resource-sharing activities, sharing the same Resource RES_{XY} . For brevity, a schedule derived during processing a sequence step is referred to as the SQS Schedule. For example, the schedule derived during processing sequence step 2 is “SQS2 Schedule.”

7.2.1 Example 7.1

Figure 7.1 displays a single unit precedence diagram of a repetitive project consisting of 3 units requiring 4 activities in each unit. Activities A and B require dedicated Resource RES_A and RES_B , respectively, while Activities X and Y share the same Resource RES_{XY} .

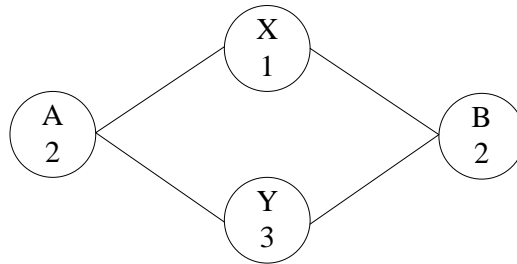


Figure 7.1 Precedence diagram with independent resource-sharing Activities X and Y in the same SQS

Figure 7.2 is a production diagram of the project in Figure 7.1, presenting the SQS2 schedule. Crew lead times (CLT) and crew idle times (CIT) of activities during processing SQS2 are shown in the dialog box. The values of CLT indicate the current

value of CLT (CLT_A), and the dash sign indicates an unassigned value of CLT (CLT_B and CLT_{XY}). The values of CIT indicate the idle time.

Figure 7.2 shows the SQS2 Schedule, which is the same as the SQS1 Schedule, because Activity A does not have predecessors and thus has no idle time. Activities X and Y compete for the same Resource RES_{XY} . The bold lines emphasize the work order for RES_{XY} . In Figure 7.2, after A1 finishes, Activity X1 starts. The reasons that X1 starts before Y1, in other words X1 successfully obtains RES_{XY} , are:

- 1) The simulation model of Activity X was created before the model of Activity Y; thus, by default, Activity X has a higher priority than Y and is first activity to obtain RES_{XY} .
- 2) Only one RES_{XY} is available; thus, Activities X and Y cannot work on the same day.

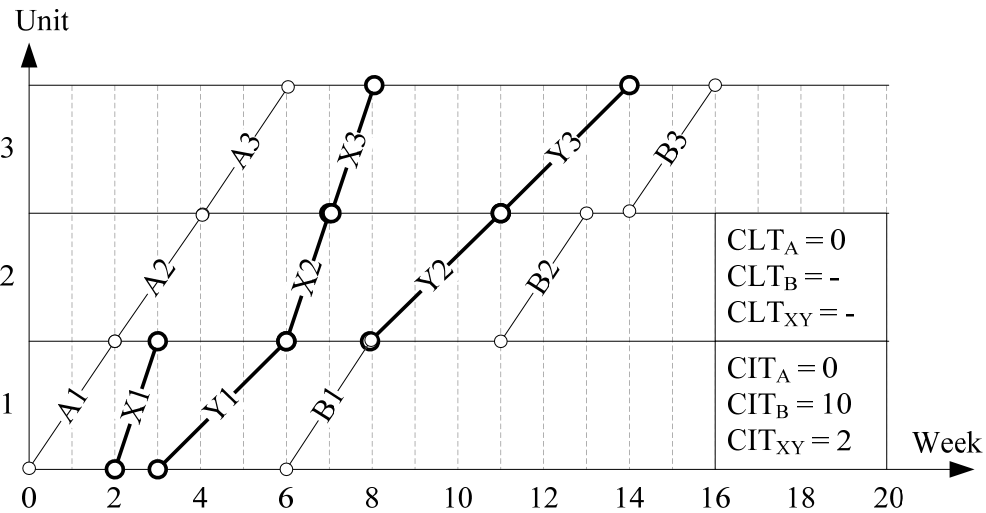


Figure 7.2 SQS2 Schedule where the simulation model for X is created before Y

In contrast to the schedule in Figure 7.2, if Activity Y is modeled before Activity X, the production diagram would appear like Figure 7.3. In Figure 7.2, the working sequence of RES_{XY} is less organized than that in Figure 7.3. In Figure 7.2, RES_{XY} works

on Activities X and Y for one unit each, then works for two units of X, and finally works for two units of Y. This schedule is considered unorganized and may cause some confusion to RES_{XY} . On the other hand, the schedule in Figure 7.3 is more organized and simpler; RES_{XY} is scheduled to finish Activity Y for all units before starting Activity X.

Idle time and project duration in Figures 7.2 and 7.3 are different in many ways: 1) the work order of RES_{XY} , 2) the idle time in Activity B, and 3) project duration. The idle time of RES_B is 4 weeks in Figure 7.2, while it is zero in Figure 7.3. The project duration is 16 weeks in Figure 7.2, while it is 18 weeks in Figure 7.3. The important issue to learn here is that the priority with which resource-sharing activities are modeled in simulation software affects the schedule, project duration, and idle time in ways that may not be obvious.

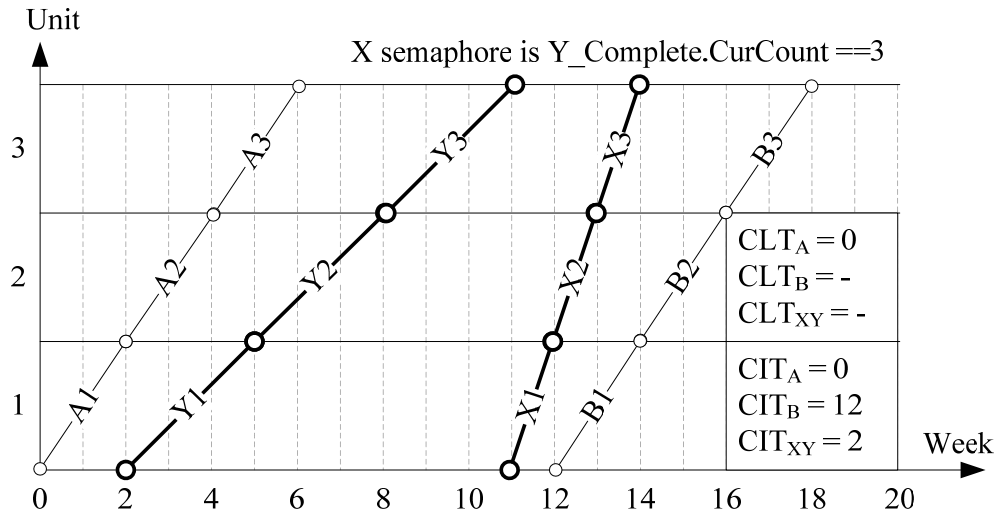


Figure 7.3 SQS2 Schedule with additional X_Perform semaphore

As opposed to Figure 7.3, Figure 7.4 stipulates RES_{XY} to finish Activity X in all units before starting on Y. This additional constraint prohibits RES_{XY} from serving Y1 before three units of Activity X have been completed. As a result, idle times of RES_{XY} and RES_B in the SQS2 Schedule are 2 weeks.

To eliminate idle time in RES_{XY} , crew idle time of RES_{XY} must be collected during processing $SQS2$, which is the sequence step of Activities X and Y. In Figure 7.4, CIT_{XY} is 4 weeks. (Remember, $SQS-AL$ assumes resources arrive to the site at the beginning of the project.) Since this is a deterministic example, CLT equals CIT . Therefore, RES_{XY} is scheduled to arrive at the site at the end of the 4th week.

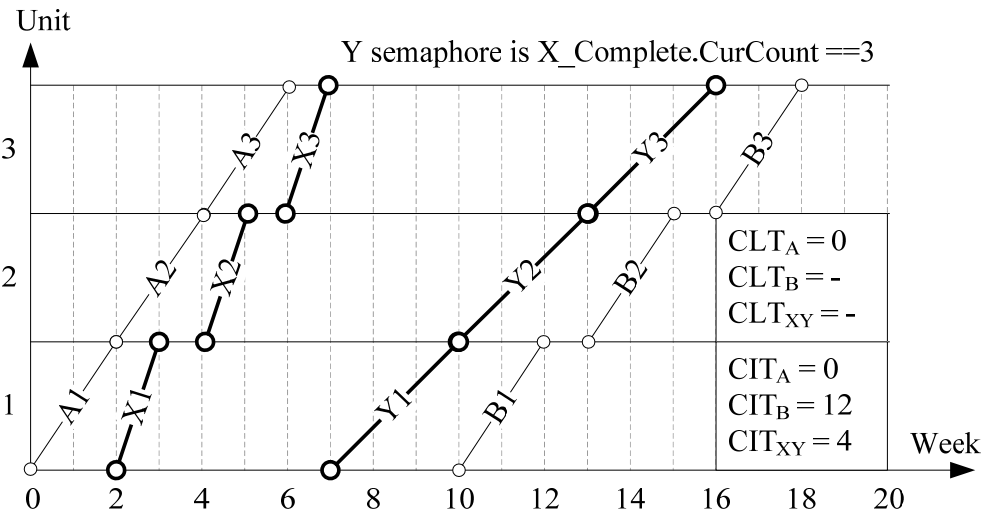


Figure 7.4 SQS2 schedule with additional Y semaphore

Figure 7.5 is the final schedule developed from Figure 7.4. Project duration in Figure 7.5 is 18 weeks. For this project, project duration is the same, either stipulating RES_{XY} to finish Activity X first or Y first. This situation may occur when resource-sharing activities are not technically dependent; however it is not always true depending on the precedence constraints between resource-sharing activities and their dependents (both direct and indirect predecessors and successors.) When switching the working sequence of resource-sharing activities does not affect the project duration, schedulers can benefit from the variations of resource allocation in many different ways. Schedulers should consider the variations as an opportunity to improve the project schedule and to maximize resource utilization.

It is important to notice that even though the schedules in Figures 7.3 and 7.5 result in exactly the same duration of 18 weeks, they require the presence of Resource XY at the site in different time intervals, i.e., Weeks 2 to 14 for Figure 7.3 and Week 4 to 16 for Figure 7.5. So, the two schedules are not interchangeable and one may be preferred over the other because of other work elsewhere for RES_{XY} (e.g., in other projects).

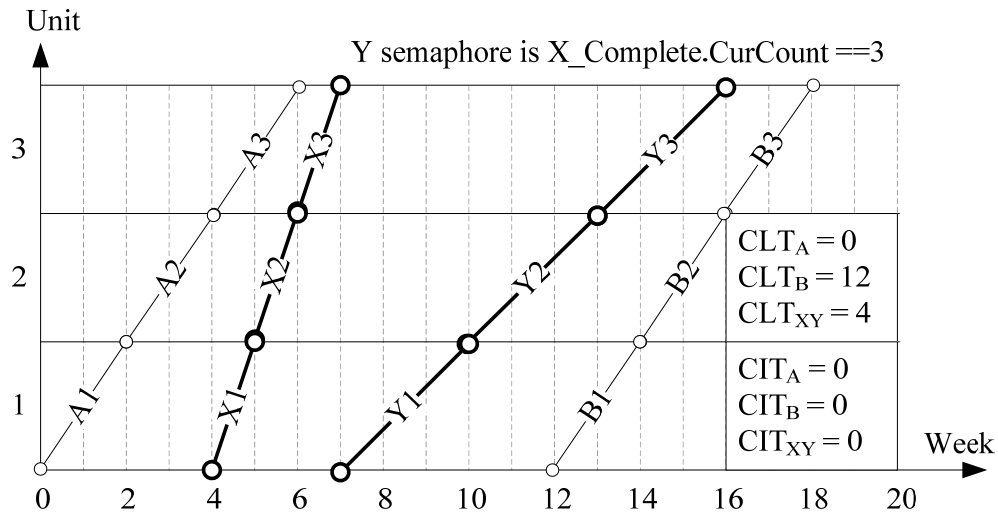


Figure 7.5 Finalized schedule with additional Y semaphore from Figure 7.4

Figures 7.6 and 7.7 show two alternative schedules by conditioning RES_{XY} to work alternately between X and Y. The differences between Figures 7.6 and 7.7 are from 1) the sequences of which the models for Activities X or Y are created, and 2) the additional semaphore.

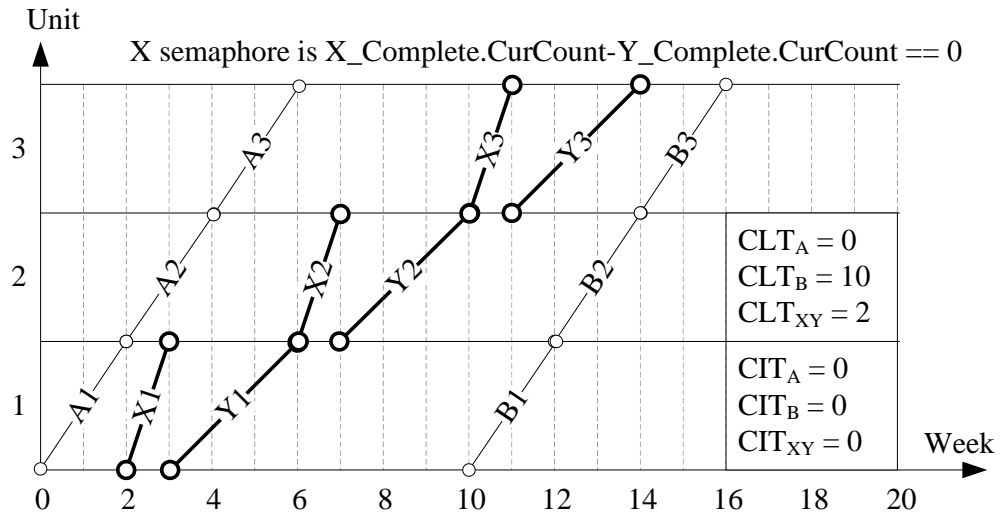


Figure 7.6 Finalized schedule with additional X semaphore

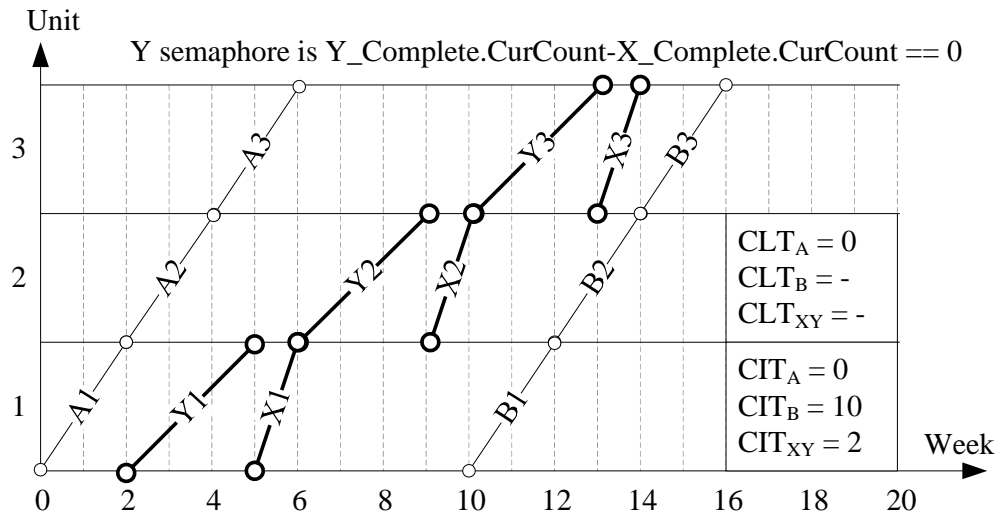


Figure 7.7 Finalized schedule with additional Y semaphore

7.2.2 Example 7.2

Figure 7.8 is a precedence diagram of a project with resource-sharing Activities X and Y that are independent of each other and are in different sequence steps. Discussion in this example and the next example illustrates an adverse impact of the orders of activities' sequence steps on SQS-AL's effectiveness.

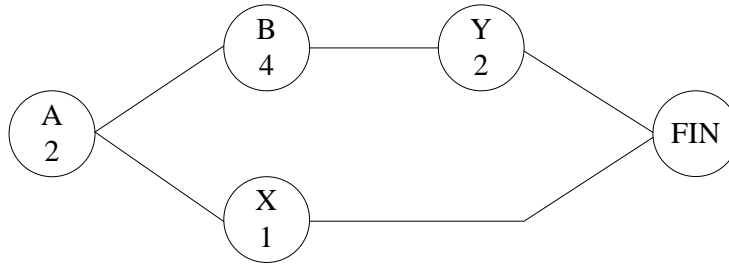


Figure 7.8 Precedence diagram with independent resource-sharing activities in different SQSs

Figure 7.9 is the SQS2 Schedule for Example 7.2. In this figure, RES_{XY} has idle time. To eliminate the idle time, SQS-AL collects CIT_{XY} during processing the sequence step to which activities served by RES_{XY} belong. However in Example 2, RES_{XY} serves two Activities X and Y that are in 2 different sequence steps, SQS2 and SQS4. Thus, it is essential to answer this question: “In which processing sequence step should resource’s crew idle time be collected and its crew lead time be determined when its resource-sharing activities are in different sequence steps?”

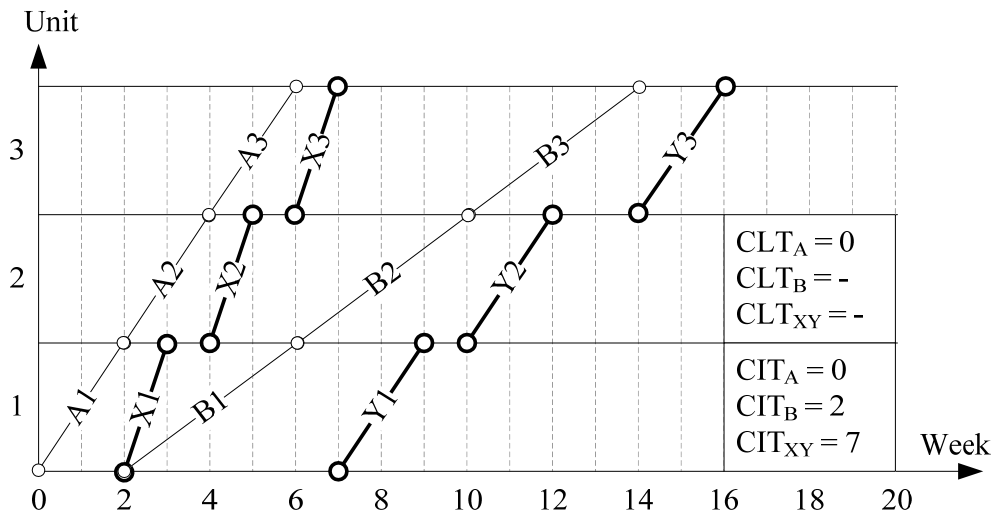


Figure 7.9 SQS1 to SQS4 Schedules, given CLT_{XY} has not been assigned

Given CLT of RES_{XY} has not been assigned in Figure 7.9, crew idle time collected from processing SQS2, SQS3, and SQS4 are the same, since there is no postponement in RES_B at the end of processing SQS2. Accordingly, CIT_{XY} from SQS2, SQS3, and SQS4 are the same. Note that Activities A and B are already in their final position, since there is no idle time existing in their resource schedule. Moreover, delaying the arrival of RES_{XY} does not affect their schedule.

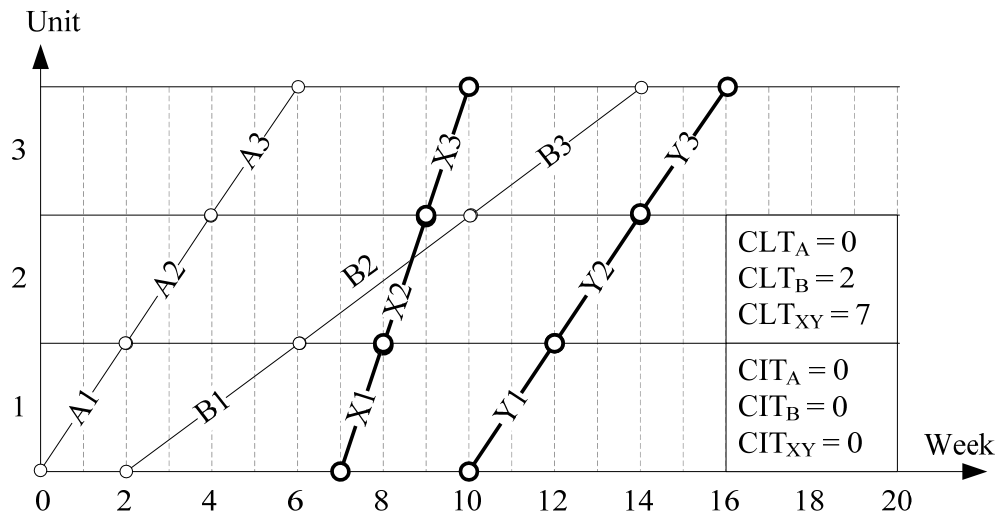


Figure 7.10 Finalized schedule after assigned CLT_{XY} either at the end of processing SQS2, SQS3, or SQS4, from Figure 7.9

Even though Example 7.2 does not depict the problem in choosing a processing sequence step to collect CIT, it shows collecting CITs and determining CLT in different sequence steps can result in the same outcome, for this example. However, in the next example, choosing different sequence steps will result in a different outcome. Therefore, the answer to which sequence step is the best for processing sequence step of the shared resource serving resource-sharing activities in different sequence steps can not be easily determined. The answer to this question requires schedulers to study the relationships between resource-sharing activities and their dependents.

7.2.3 Example 7.3

Figure 7.11 is another example of scheduling resource-sharing activities that are in different sequence steps. The precedence diagram shown in Figure 7.11 is similar to Figure 7.8 in Example 7.2 except the production rates of A and B are changed. The slow production rate of A relative to B and X incurs idle time in B and X. It is interesting that the change in a predecessor (Activity A) of resource-sharing Activities X and Y has a great impact on the schedule and effectiveness of SQS-AL in solving this problem.

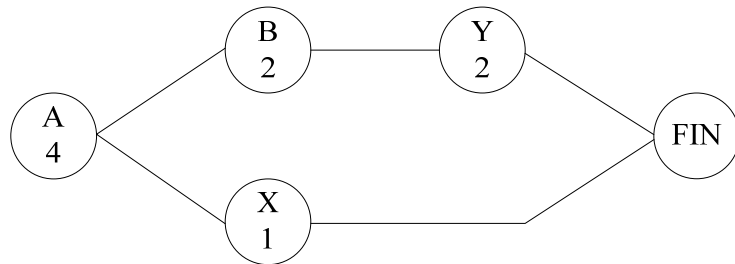


Figure 7.11 Precedence diagram with independent resource-sharing activities in different SQSs

Figure 7.12 shows a production diagram for Example 7.3. Comparing between Figures 7.10 and 7.12, a significant difference between Examples 7.2 and 7.3 is that in Activity B, the predecessor of resource-sharing Activity Y, has idle time. Postponing B at the end of processing SQS2 is likely to affect Activity Y in the next sequence step, SQS3.

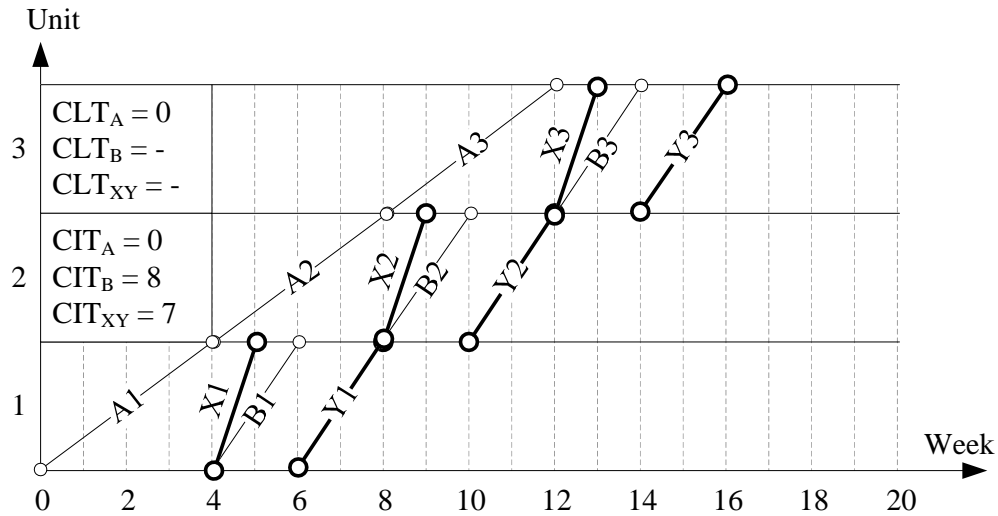


Figure 7.12 SQS2 schedule

In the SQS2 Schedule shown in Figure 7.12, the CIT_{XY} is 7 weeks. The idle time between the arrival date and first unit for RES_{XY} is 5 weeks ($AIT = 5$ week), and the idle time between units is 2 weeks ($UIT = 2$ weeks). If CIT_{XY} is collected in processing SQS2, CLT_{XY} will be 7 weeks; therefore, RES_{XY} will be scheduled to arrive at the site on the 7th week. However, this CLT_{XY} of 7 weeks does not eliminate idle time in RES_{XY} completely. It results in a remaining idle time of 1 week ($UIT = 1$ week), as shown in Figure 7.13. Figure 7.13 shows the result of scheduling RES_{XY} at the end of processing SQS2. CIT_{XY} from processing SQS2 is not effective, because postponing X delays Activity B in SQS2, which in turn delays the start date of Y to the end of the 10th week, causing idle time in RES_{XY} in the 9th week.

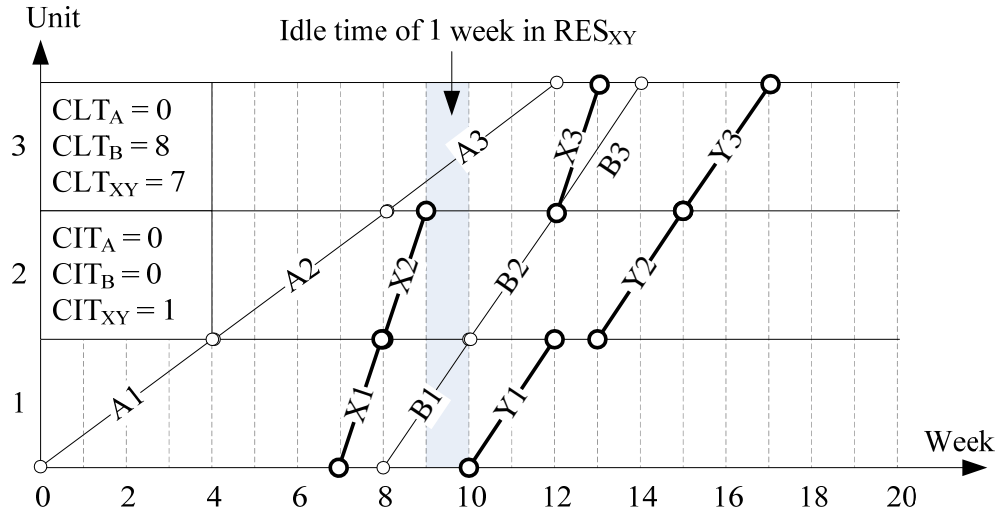


Figure 7.13 Finalized SQS-AL schedule, given CLT_{XY} is derived from the SQS2 schedule, idle time in RES_{XY} is 1 week

Figure 7.14 is the SQS3 Schedule with no postponement in RES_{XY} at the end of processing SQS2. In the SQS3 Schedule under the given condition shown in Figure 7.14, CIT_{XY} is 8 weeks. Comparing between Figures 7.12 and 7.14 shows postponing Activity B at the end of processing SQS2 induces more idle time in RES_{XY} . Accordingly, it can be concluded that CIT_{XY} should not be collected during processing SQS2, but during processing SQS3 instead. If CIT_{XY} is collected from processing SQS3 (Figure 7.14), RES_{XY} will be scheduled to the site at the beginning of the 9th week resulting in zero idle time in RES_{XY} , as shown in Figure 7.15.

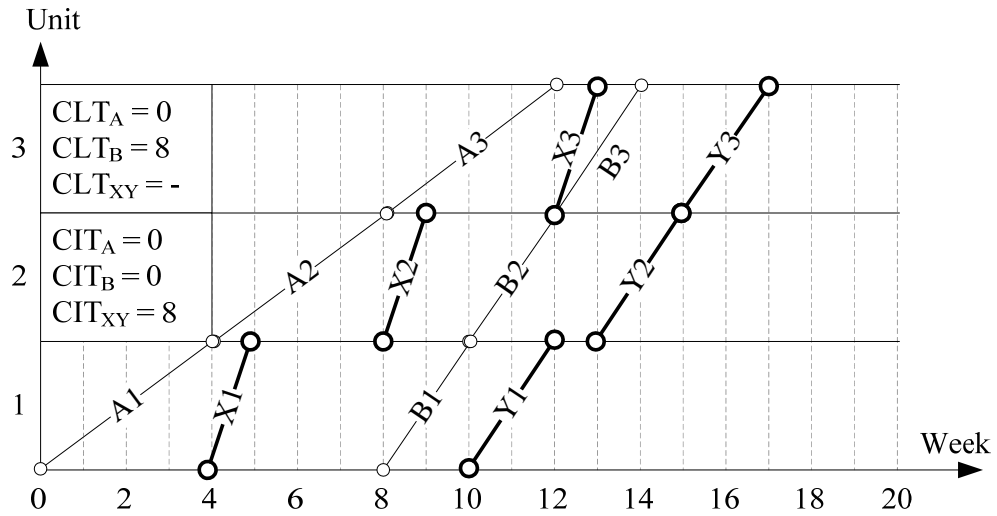


Figure 7.14 SQS3 schedule without delaying Activity X in SQS2, developed from Figure 7.12

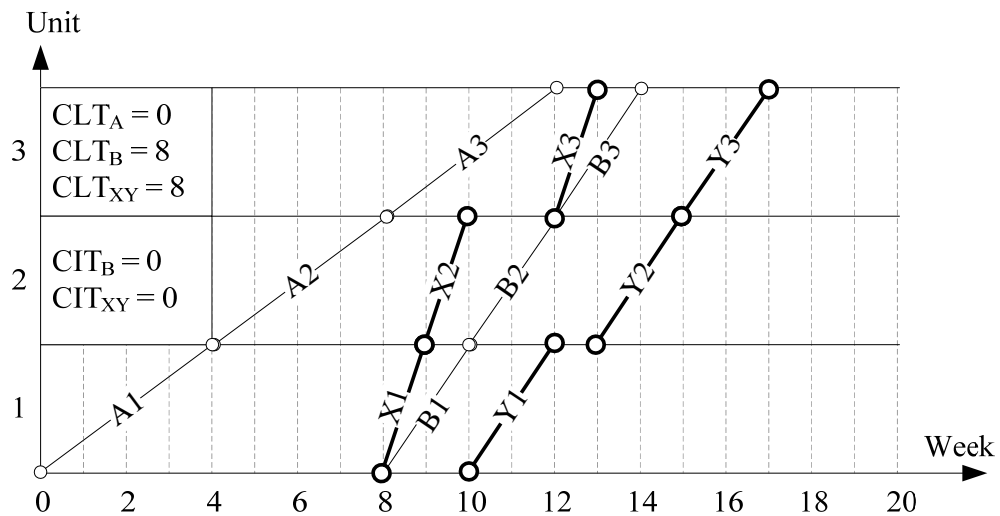


Figure 7.15 Finalized SQS-AL schedule using the CIT of RES_{XY} from SQS3, developed from Figure 7.14

Examples 7.2 and 7.3 illustrate the effect of choosing the sequence step at which to collect the CIT of shared resources when resource-sharing activities are in different sequence steps. In Example 7.2, it does not matter whether the earlier sequence step (SQS2) or the later sequence step (SQS4) of resource-sharing Activities X and Y

respectively is chosen. In contrast, in Example 7.3, choosing the earlier sequence step (SQS2) would result in idle time of 1 week in the shared RES_{XY} .

Figures 7.16 to 7.18 display an alternative schedule that uses an additional Y semaphore for Activity Y to specify the working sequence of RES_{XY} . The Y semaphore conditions RES_{XY} to finish three units of X before starting on Activity Y. As a result of applying the additional Y semaphore, the finalized schedule in Figure 7.18 shows zero idle time with project duration of 19 weeks.

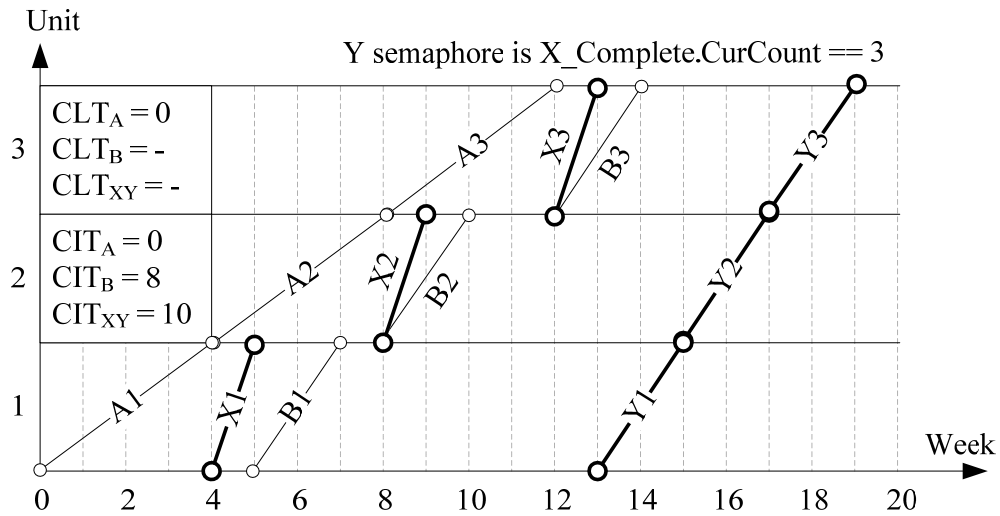


Figure 7.16 SQS2 schedule with additional Y semaphore

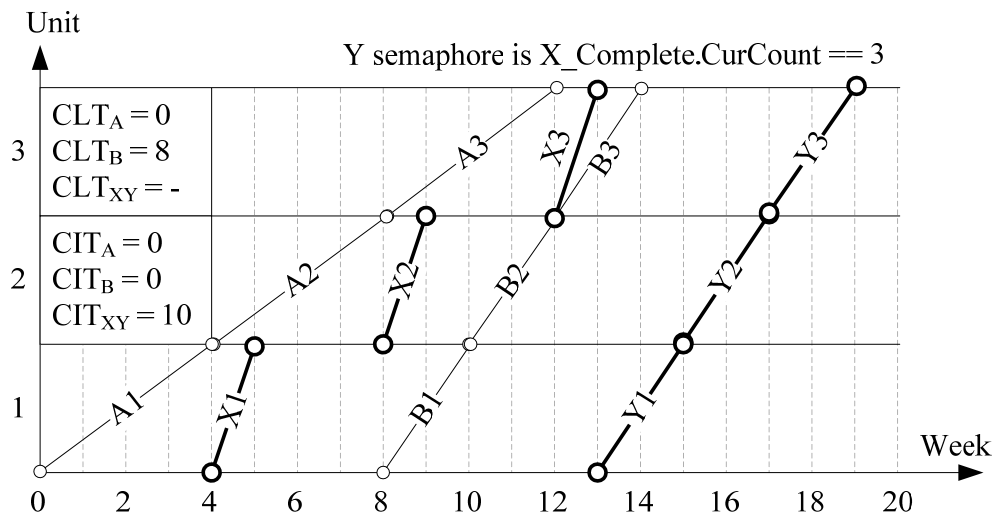


Figure 7.17 SQS3 schedule, developed from Figure 7.16

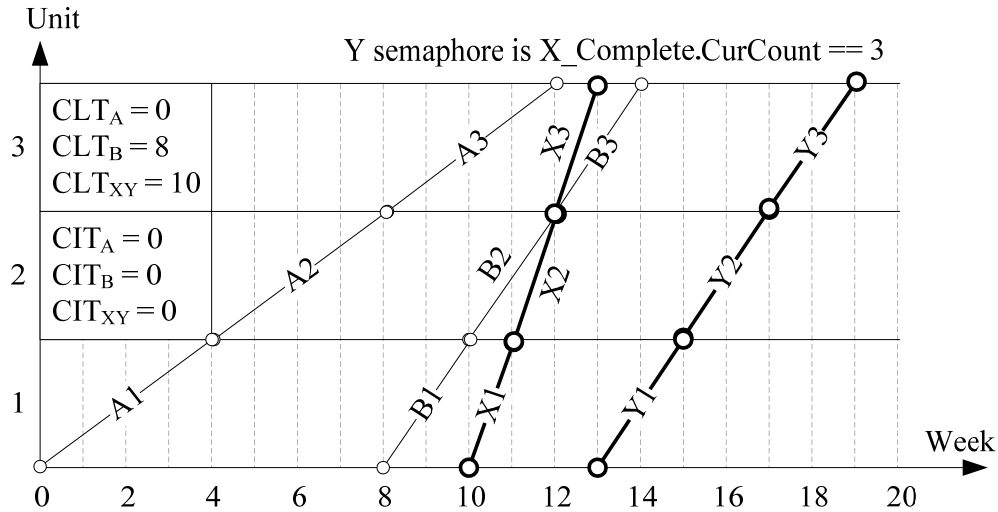
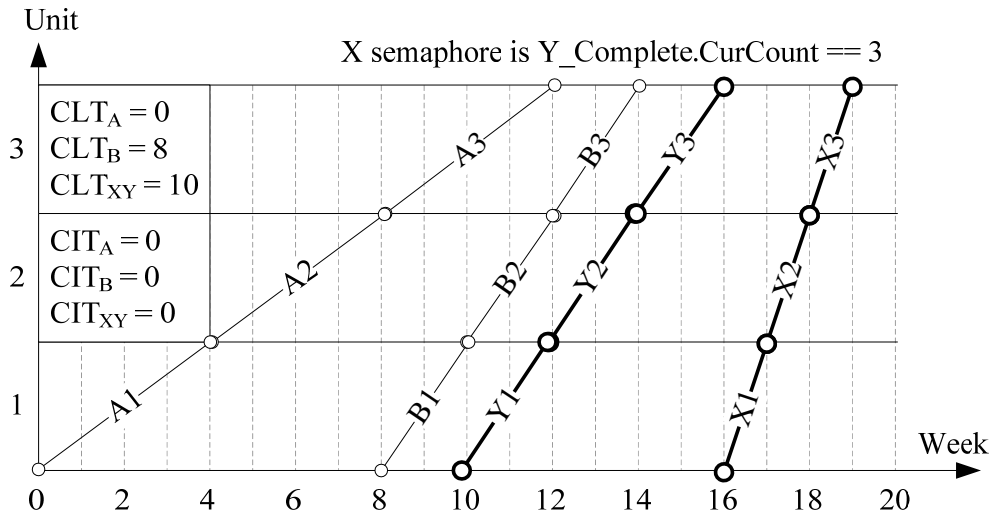
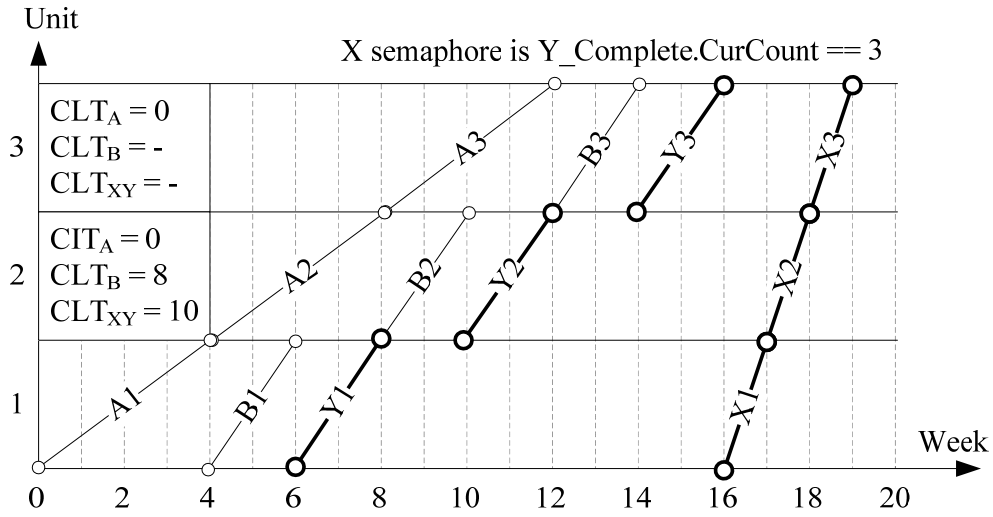


Figure 7.18 Finalized schedule, developed from Figure 7.17

Note that the schedule with an additional semaphore (i.e., Figure 7.18) is two weeks longer than that of the schedule without one (i.e., Figure 7.15). However, the schedule with the stipulated working sequence of RES_{XY} in Figure 7.18 is straightforward and easy to manage the resource. On the other hand, the schedule in Figure 7.15 may incur confusion especially when the number of repetitive units is large.

Figures 7.19 and 7.20 display another alternative schedule by applying an additional X semaphore to specify the working sequence of RES_{XY} . The semaphore in this schedule forces RES_{XY} to finish three units of Y before starting on Activity X. The finalized schedule in Figure 7.20 shows zero idle time with a project duration of 19 weeks, the same as in the previous alternative schedule shown in Figure 7.18. Applying an additional semaphore in resource-sharing activities could alleviate the problems of choosing between different sequence steps of resource-sharing activities, as exhibited in Figures 7.18 and 7.20.



Note that the two alternative schedules shown in Figure 7.18 and 7.20 result in the same project duration and the hiring period for RES_{XY} . However, the schedule in Figure 7.20 has a higher probability of having interruptions in RES_{XY} when working on Activity Y due to the parallel production diagram between Activities B and Y. There are three possible interruption points. Whenever the production rate of Y (e.g., Y1) is higher than B (e.g., B2), it will incur an interruption in Activity Y.

B1 is not completed at the end of Week 10, RES_{XY} has to wait before working on Activity Y.

7.2.4 Example 7.4

Figure 7.21 is a precedence diagram of a repetitive project with four repetitive activities. Activities A and B use dedicated Resources RES_A and RES_B, respectively; Activities X and Y share the same resource RES_{XY}. These two resource-sharing activities X and Y are directly dependent.



Figure 7.21 A precedence diagram with directly dependent resource-sharing activities X and Y

Figure 7.22 is the production diagram of the project before processing any sequence steps. The simulation model of Activity X in SQS2 is modeled first before the simulation model of Activity Y; therefore, Activity X has higher priority than Activity Y. For this example, it is unnecessary to postpone any activities in order to eliminate idle time because there is no idle time in any resources. Nevertheless, it is beneficial to study this example to gather insightful information regarding activity duration and idle time.

As shown in Figure 7.21, each unit of Activities A and X takes 3 weeks to complete. Since the production rates of A and X are the same, there is no idle time in Activity X. Comparing Activities X and Y, the production rate of X is slower than Y, which should cause idle time in Y. However, this is not the case, since they share the same Resource RES_{XY}, and they are directly dependent. When there is an idle time or

interruption between units (e.g., X1 and X2), RES_{XY} will work on another resource-sharing activity (e.g., Y1).

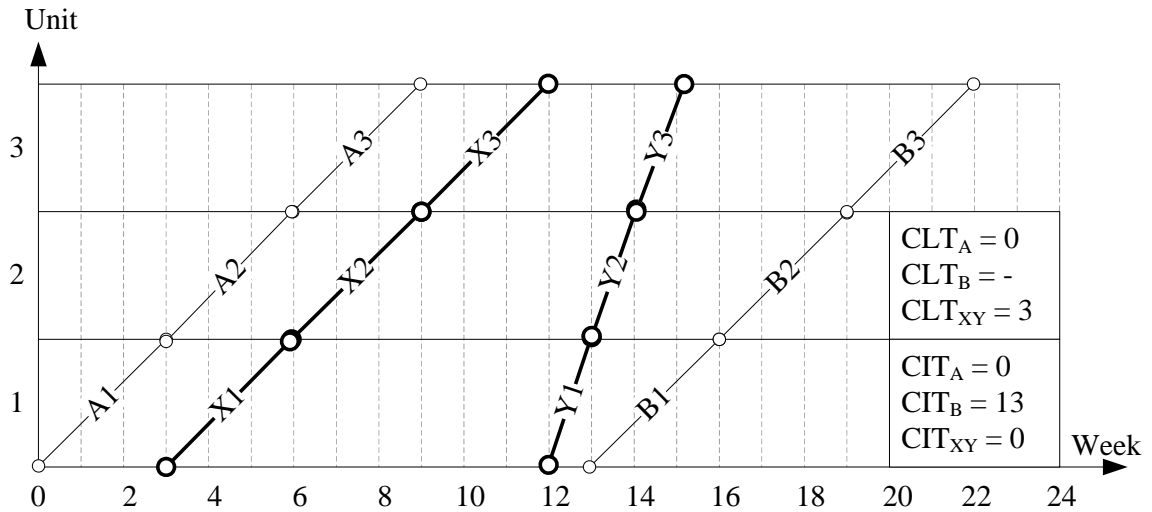


Figure 7.22 SQS2 schedule

For this example, idle time in RES_{XY} is always zero as long as the combined production rate of X and Y (4 weeks) is equal to or slower than the production rate of A. When comparing between Figure 7.22 and Figure 7.23, if production rate of Activity A is slower than X as in Figure 7.23, Activity Y will get the resource RES_{XY} whenever activity X cannot start. As a result, the idle time of RES_{XY} between units in Activity X is eliminated by the work in Activity Y.

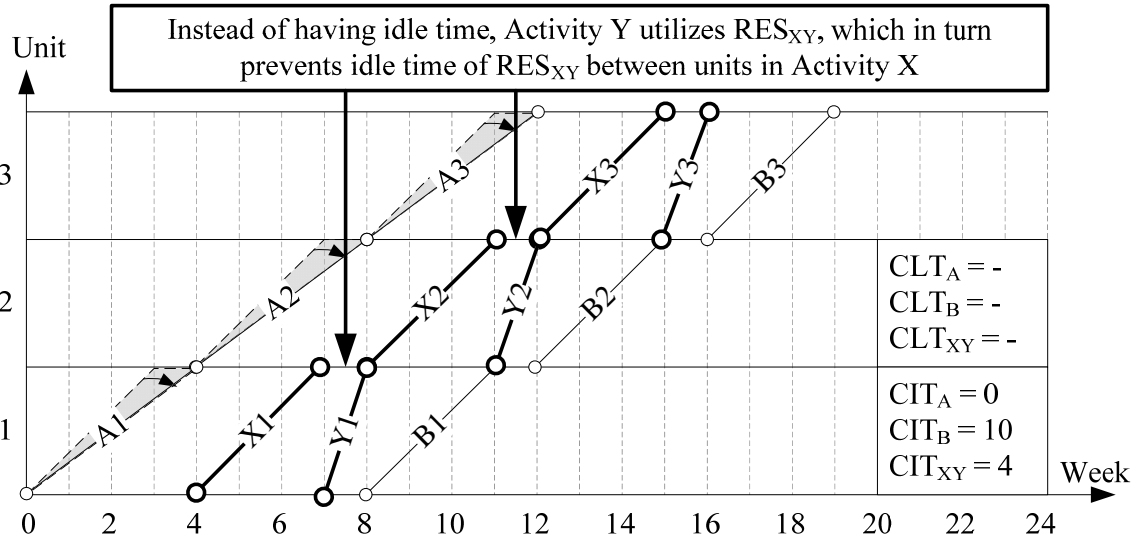


Figure 7.23 Modifying Activity A's duration and comparing the duration to the combined durations of X and Y

Figure 7.24 is an alternative schedule from the original duration shown in Figure 7.22. Figure 7.24 illustrates that project duration is reduced significantly from 22 to 18 weeks by introducing additional precedence constraints (X semaphore) between X and Y, forcing RES_{XY} to work alternately between X and Y.

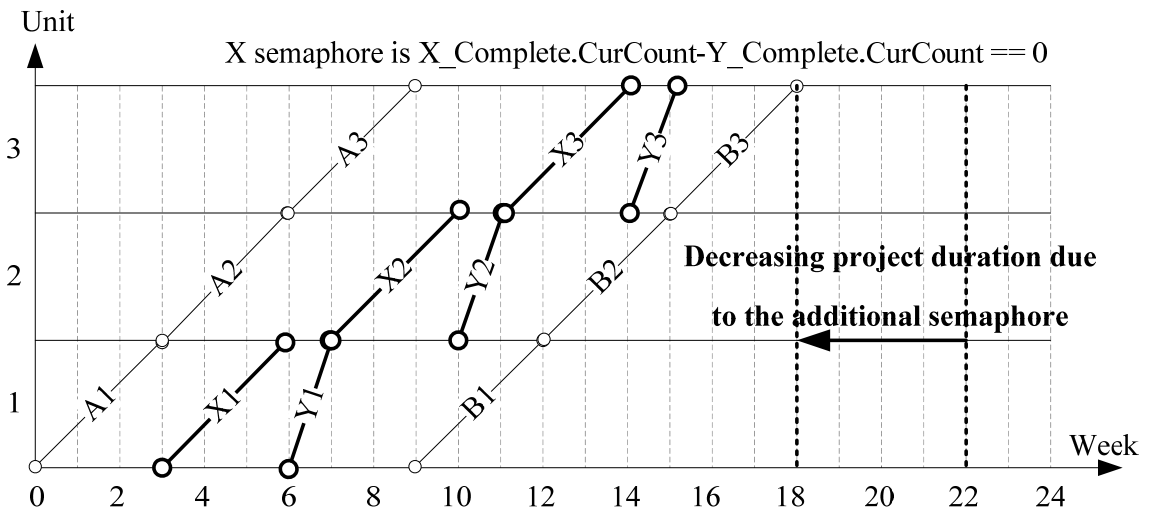


Figure 7.24 Decreasing project duration in the SQS2 schedule due to additional X semaphore

7.2.5 Example 7.5

Figure 7.25 is a precedence diagram of a repetitive project consisting of 3 units requiring 4 repetitive activities. Activities A and B utilize two dedicated resources RES_A and RES_B ; X and Y use the same Resource RES_{XY} . Resource-sharing Activities X and Y are indirectly dependent having Activity B between them as shown in Figure 7.25.

As explained below, this is an interesting example because it illustrates that the working sequence of resource-sharing activities (i.e., X and Y) could be affected by another activity (i.e., Activity B) whose precedence places it between the two resource-sharing activities.



Figure 7.25 Precedence diagram with indirectly dependent resource-sharing Activities X and Y

Figure 7.26 is the SQS2 Schedule of the project, where CIT_{XY} is 5 weeks. Figure 7.27 is the SQS3 Schedule, where CLT_{XY} is 5 weeks from the previous processing SQS2, and CIT_B is 15 weeks. As shown in Figures 7.26 and 7.27, Activity Y1 takes place between X2 and X3, and thus absorbs the idle time between X2 and X3. However, it is likely that RES_{XY} will follow a different working sequence after CLT_B is determined due to precedence constraints between Activities B and Y. Postponing the arrival date of RES_B will delay Activity B' successor, Activity Y.

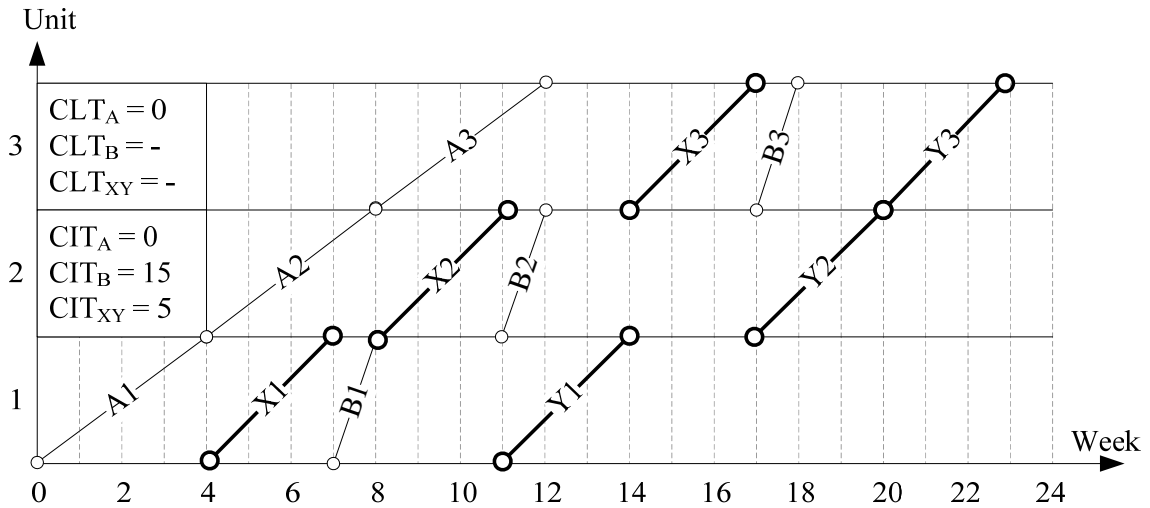


Figure 7.26 SQS2 schedule

Comparing Figures 7.27 to 7.28, the working sequence of RES_{XY} changes from X1, X2, Y1, X3, Y2, Y3 in the SQS3 Schedule to X1, X2, X3, Y1, Y2, Y3 in the SQS4 Schedule. Hence, CLT_B (15 weeks) derived from the SQS3 Schedule (Figure 7.27) may not be the best, because assigning CLT_B of 15 weeks changes the working sequence of the shared resource RES_{XY} serving Activity B's dependents. The difference in working sequence of RES_{XY} adversely results in: 1) idle time in RES_{XY} and 2) an unnecessary delay in RES_B , as shown in Figure 7.28. Thus, the blind application of SQS-AL results in a suboptimal schedule.

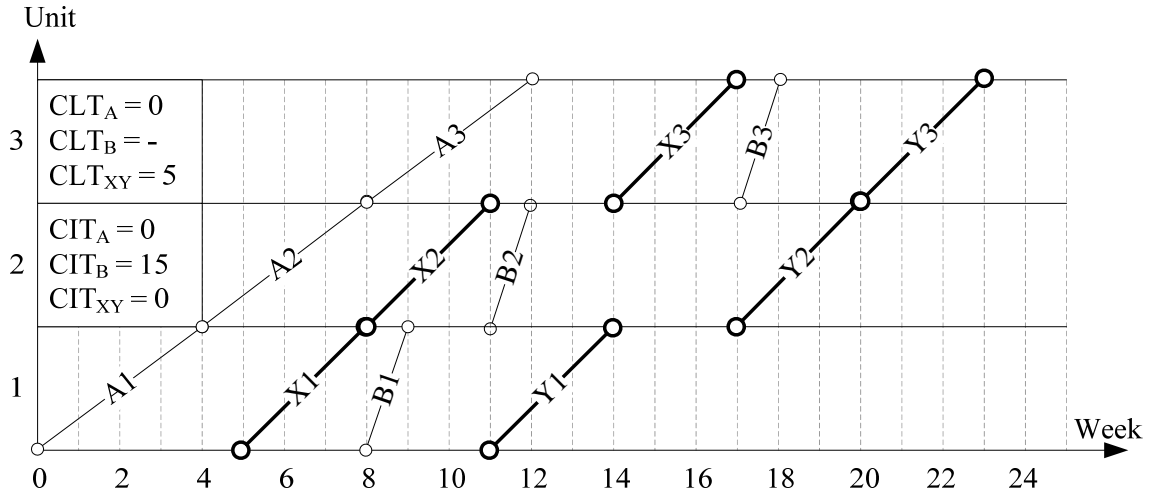


Figure 7.27 SQS3 schedule, developed from Figure 7.26

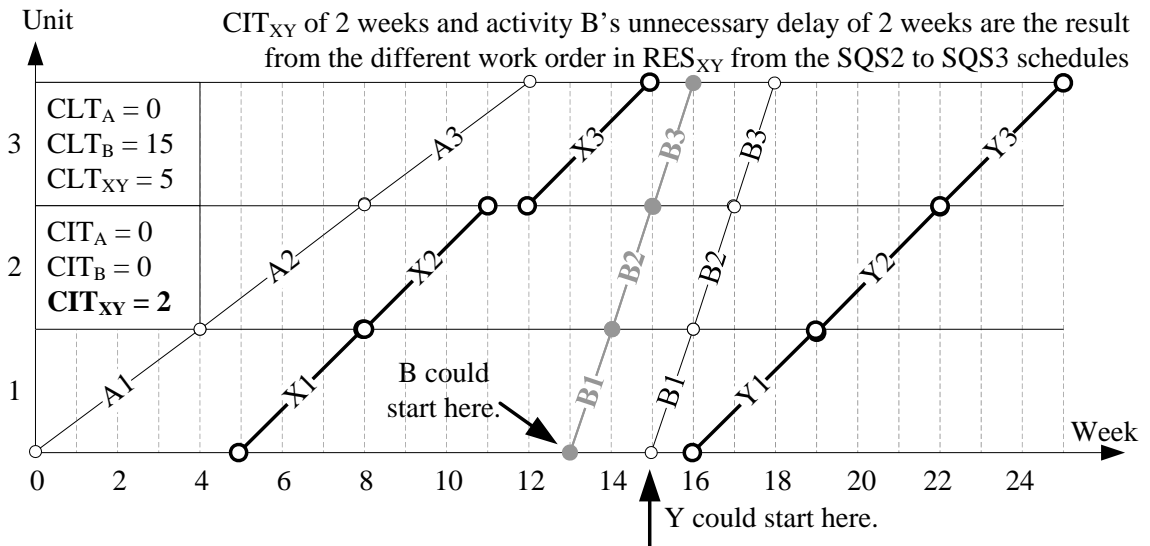


Figure 7.28 Finalized schedule, developed from Figure 7.27

Figure 7.29 is the SQS2 Schedule with an additional semaphore for Activity Y that prevents Y from starting unless all units of X are finished; the resulting schedule is shown in Figure 7.29. The working sequence for RES_{XY} is fixed, and Activity Y cannot absorb the idle time between X2 and X3 in SQS3, as it does in Figure 7.27. As a result of adding the additional semaphore in Figure 7.29, CIT_{XY} increases to 6 weeks for the SQS2 Schedule, compared to 5 weeks in Figure 7.26.

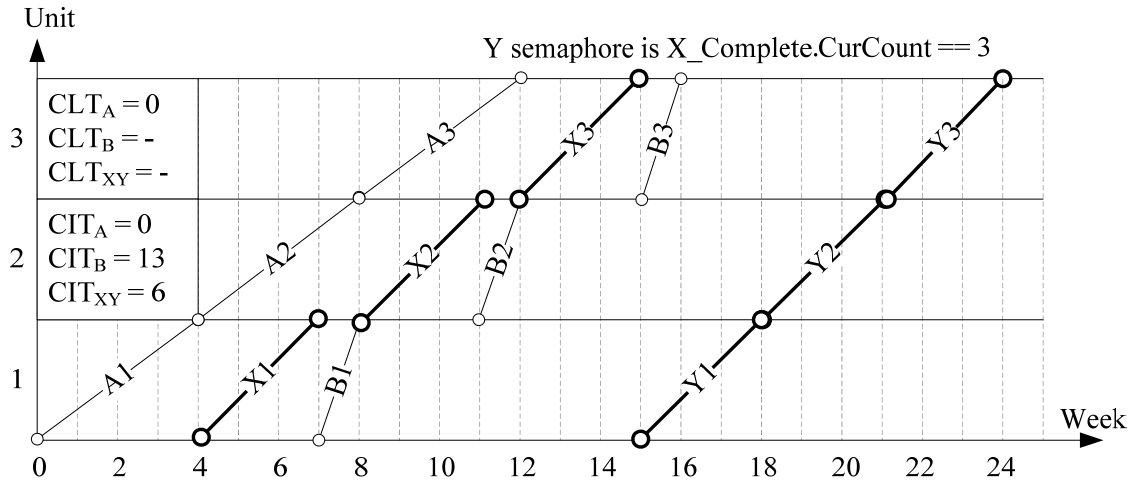


Figure 7.29 SQS2 schedule with additional Y semaphore

Figure 7.30 is the SQS3 Schedule with the additional semaphore and CLT_{XY} is 6 weeks. CIT_B in the SQS3 Schedule with the semaphore is 13 weeks (Figure 7.30), which is different from CIT_B in the SQS2 Schedule (15 weeks) without the semaphore (Figure 7.27).

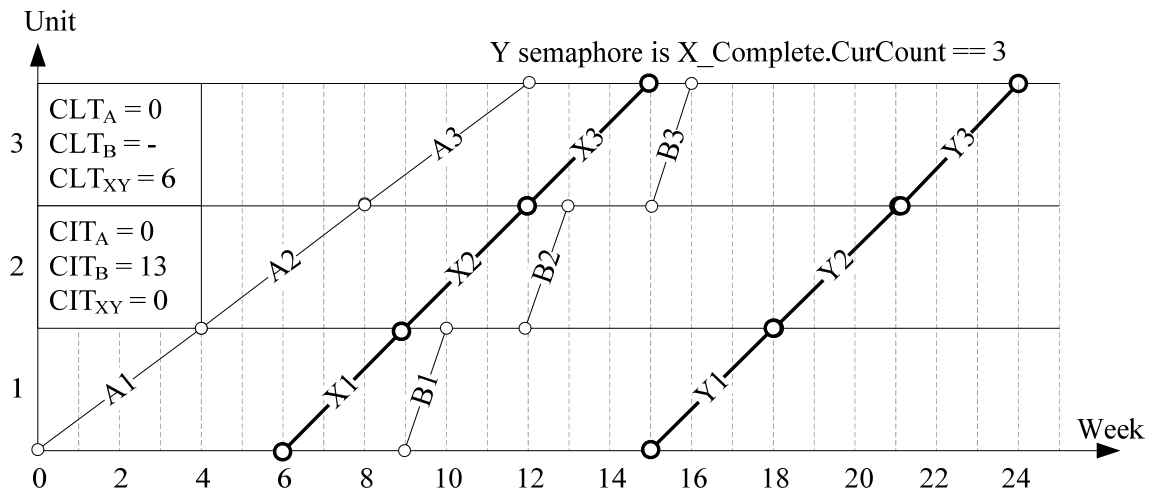


Figure 7.30 SQS3 schedule with additional Y semaphore, from Figure 7.29

A comparison between the finalized schedules with the semaphore (Figure 7.31) and without the semaphore (Figure 7.28) shows that allowing RES_{XY} to start work in Y as early as possible results in an unfavorable impact on the schedule derived by SQS-AL.

This is caused by the unexpected change in working sequence for RES_{XY} from processing one sequence step (Figure 7.27) to another (Figure 7.28). By introducing the additional Y semaphore, the working sequence for RES_{XY} is fixed ensuring consistency in the working sequence from processing one sequence step (Figure 7.29) to another (Figure 7.30). As a result, CIT_{XY} from the SQS2 Schedule (Figure 7.29) is compatible to the working sequence for RES_{XY} in the finalized schedule (Figure 7.31).

Comparing between the finalized schedule in Figure 7.31 where RES_{XY} has a specified working sequence and in Figure 7.28 where RES_{XY} has an unspecified working sequence, the schedule from Figure 7.31 is preferable because its project duration is shorter by 1 week. Moreover, CIT_{XY} in Figure 7.31 is zero, while CIT_{XY} in Figure 7.28 is 2 weeks. Project duration in Figure 7.31 is less susceptible to delay in B1 than that in Figure 7.28. There is a buffer of 1 week between Activities B and Y. As long as an unexpected delay in B1 is less than 1 week, project duration should be completed on the deadline. On the other hand in Figure 7.28, if there is an unexpected delay in RES_B from its CLT_{XY} (Week 15), project duration will most likely be delayed.

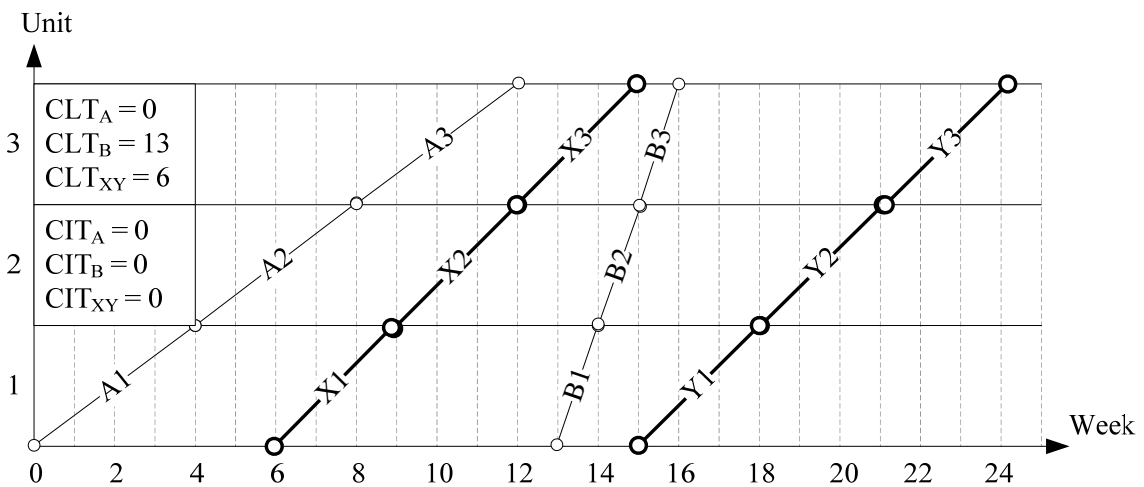


Figure 7.31 SQS4 schedule, developed from Figure 7.30

As discussed above, the schedule shown in Figure 7.31 is more favorable than Figure 7.28. However, the schedule shown in Figure 7.28 has one advantage; it provides a higher confidence level in Activity B due to the buffer between Activities X and B. In most circumstances, the schedule in Figure 7.31 should be chosen. An exception is when the penalty cost from an interruption (or idle time) in Activity B or Resource RES_B is far more significant than penalty cost from 1) delaying project completion and/or 2) idle time in resource RES_{XY} .

7.2.6 Example 7.6

Example 6 is similar to Example 5, except the production rates of the four activities are changed to demonstrate the adverse effect of idle time in resource-sharing activities on their dependents. Figure 7.32 shows a precedence diagram and activity durations, and Figure 7.33 shows the production diagram from the SQS2 Schedule.



Figure 7.32 Indirectly dependent resource-sharing Activities X and Y with a slower-production-rate Activity B between them

In Figure 7.33, the production rates of resource-sharing Activities X and Y are faster than that for Activity B between them. In other words, resource-sharing Activities X and Y are indirectly dependent separated by Activity B whose production rate is relatively slow compared to that of X and Y. As a result, there is idle time in Activity Y.

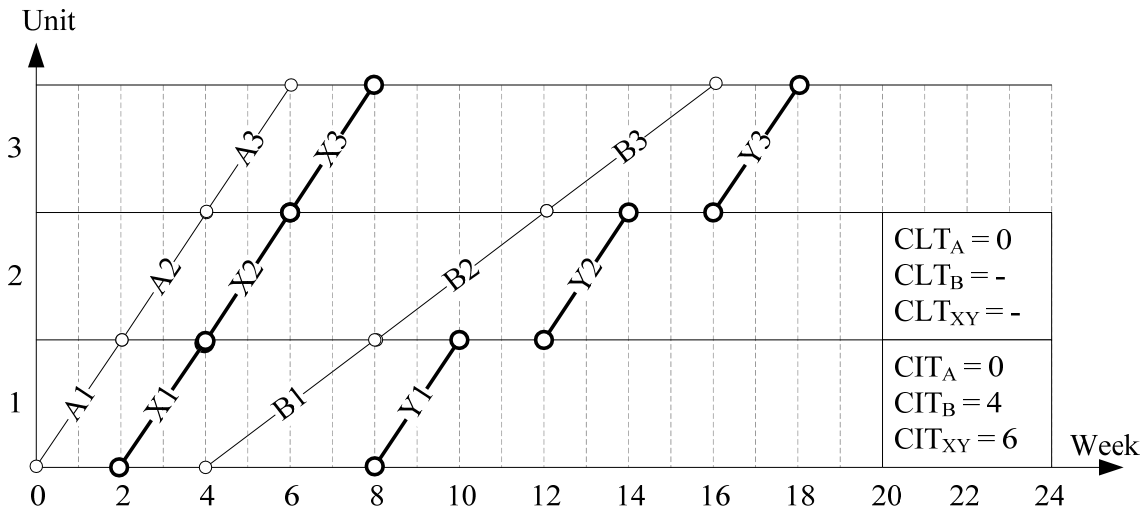


Figure 7.33 SQS2 schedule

Figure 7.34 is the SQS3, SQS4, and the finalized schedules, where CIT_{XY} of 4 weeks is derived from processing SQS2, in Figure 7.33. That is at the end of processing SQS2, RES_{XY} is scheduled to arrive at the site at the end of the 6th week, resulting in CIT_B of 8 weeks in SQS3. Therefore, at the end of processing SQS3, a CLT_B of 8 weeks is assigned. In SQS4, there is no resource that has not yet been assigned crew lead time. Accordingly, there is no change in the schedule from SQS3 to SQS5 (final schedule), although there is idle time of 4 weeks in Activity Y. In addition, it is obvious that the project duration is unnecessarily delayed without eliminating any idle time between units in Activity Y.

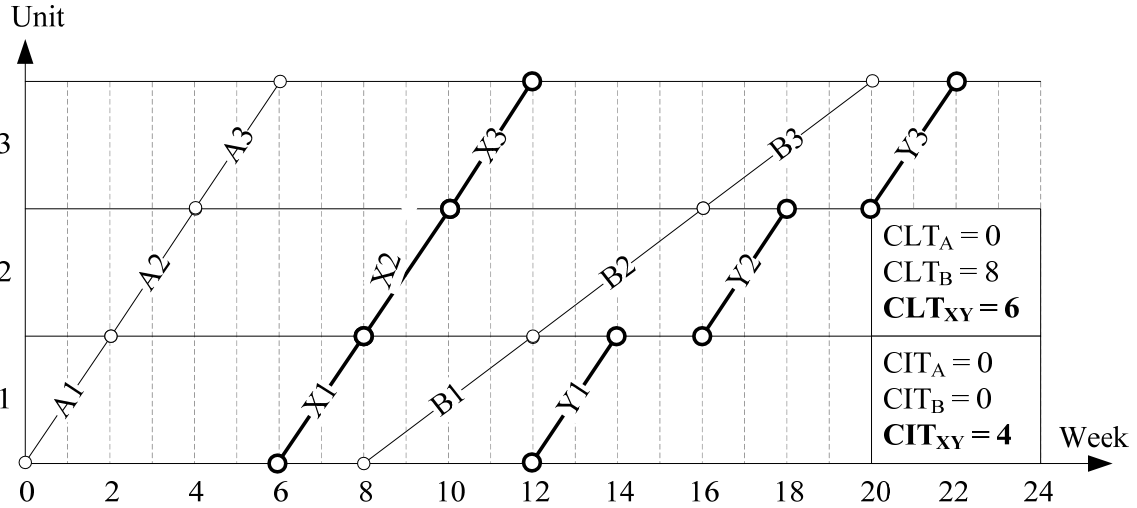


Figure 7.34 SQS3, SQS4, and the finalized schedule when using CIT_{XY} from SQS2 and CIT_B from SQS3, developed from Figure 7.33

As shown in Figure 7.34, delaying the arrival date of RES_{XY} does not have the intended effect of eliminating interruptions in Activity Y, since delaying Activity X will also delay its successor Activity B which causes the idle time and interruption in Y. As a result, Activity Y is delayed and has the same idle time as before (Figure 7.34).

In this case, the idle time of 4 weeks in RES_{XY} cannot be eliminated because of:

- 1) Precedence constraints among Activities X, B, and Y
- 2) The relatively slower production rate of Activity B compared to X and Y
- 3) Continuity constraints between resource-sharing Activities X and Y

Figure 7.35 is an alternative schedule for Example 7.6, showing the finalized schedule, where CLT_{XY} is derived from processing SQS4 (not SQS2), after CLT_B in processing SQS3. Remember, CLT_B could be derived from either SQS2 (before CLT_B) or SQS4 (after CLT_B). Consequently, postponing the arrival of RES_{XY} delays Activity X, after already scheduling RES_B , causes AIT_B of 4 weeks. This AIT_B of 4 weeks exists between RES_B 's arrival date and the first start date of B1, as shown in Figure 7.35.

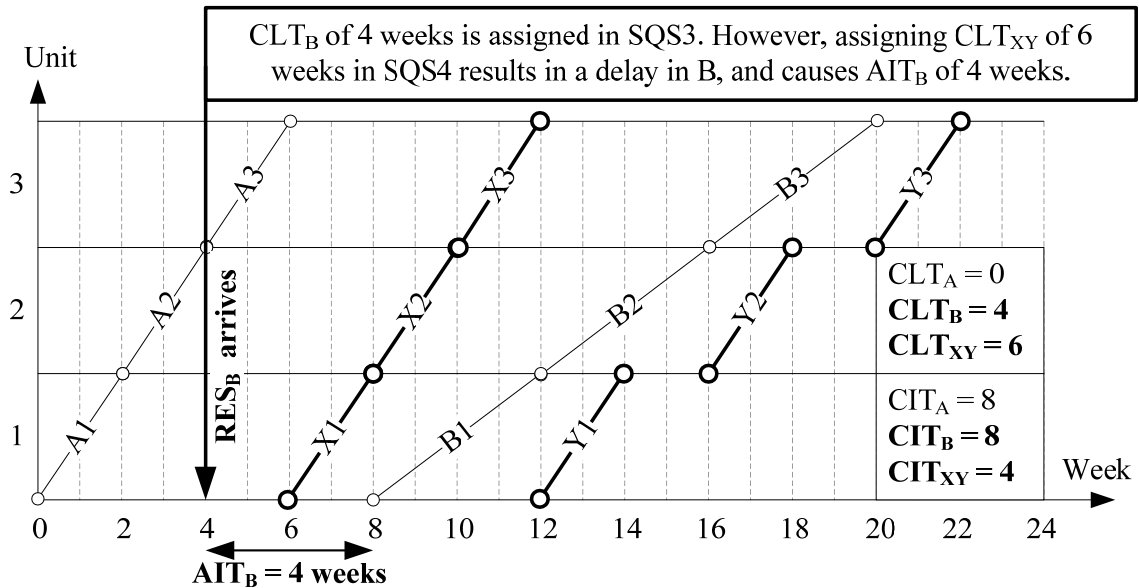


Figure 7.35 Finalized schedule when using CLT_B from SQS3 and CLT_{XY} from SQS4, developed from Figure 7.34

In accordance with the finalized schedules in Figures 7.34 and 7.35, it can be concluded that the idle time in the shared Resource RES_{XY} cannot be eliminated, as long as Activities X and Y still share the same resource or continuity constraints of RES_{XY} remains the same. To eliminate idle time in RES_{XY}, three alternatives can be applied.

The first alternative is to assign dedicated resources to Activity X and Activity Y, as shown in Figure 7.36.a. A dedicated RES_X is assigned to Activity X, whereas another dedicated RES_Y is assigned to Activity Y. The second alternative is to introduce a work break of 4 weeks between Activities X and Y, as shown in Figure 7.36.b.

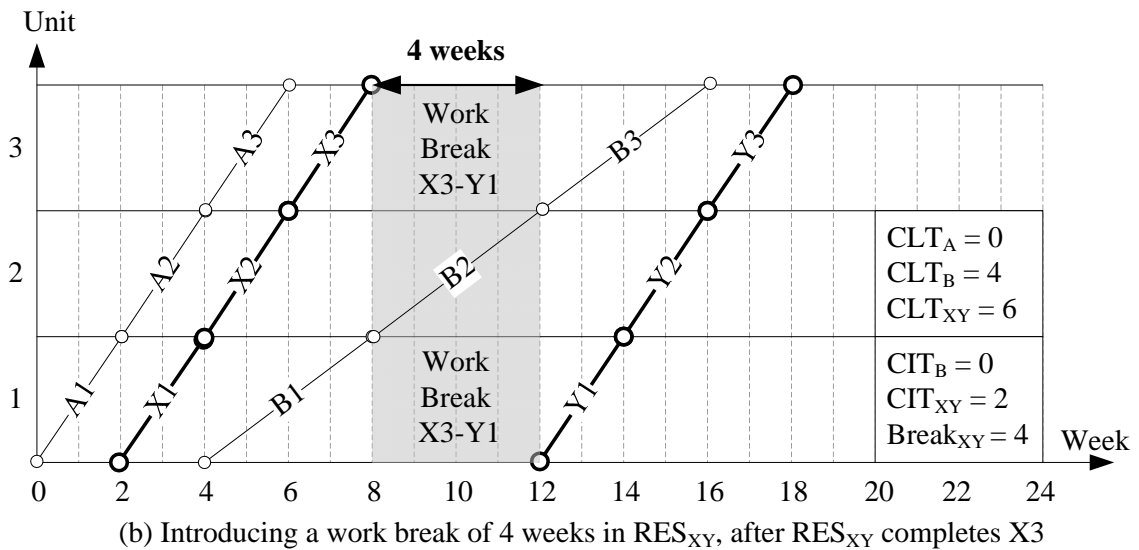
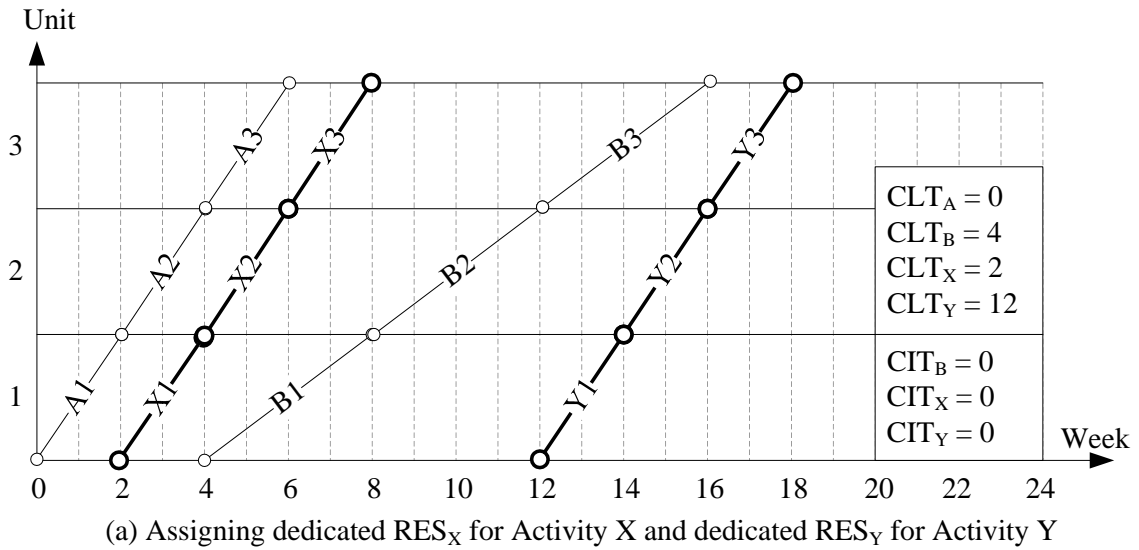
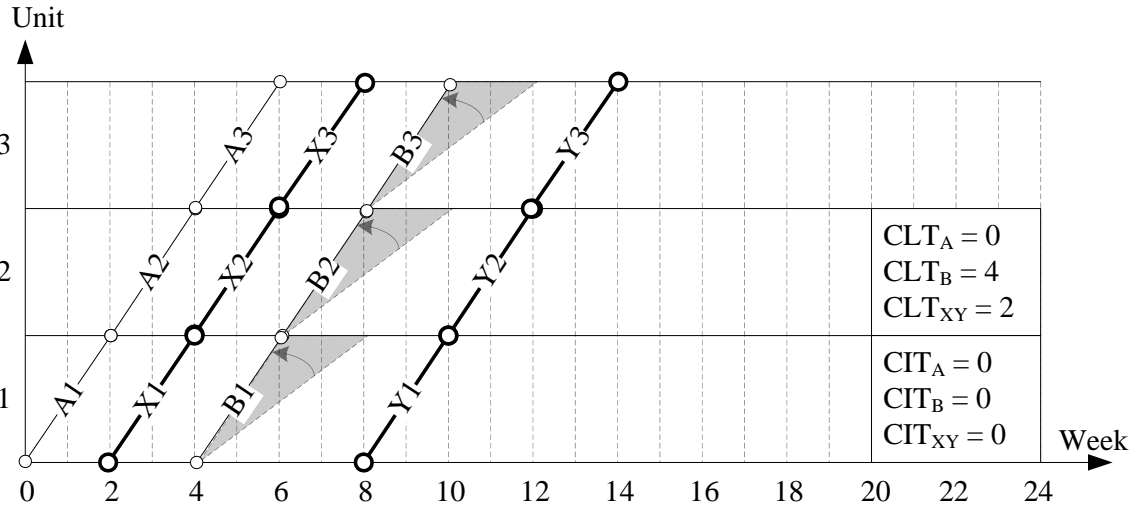
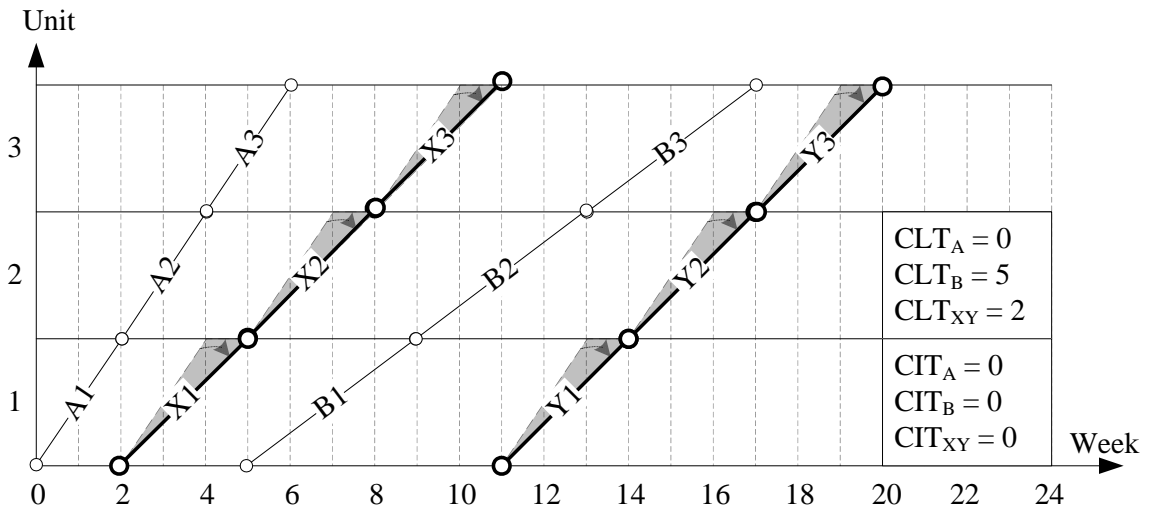


Figure 7.36 Using dedicated resources or work break between Activities X and Y

The third alternative is to balance the production rates for Activities X, B, and Y as shown in Figure 7.37. Figure 7.37.a illustrates that the production rate for RES_B is increased in order to balance the production rates between RES_B and RES_{XY} . On the other hand, Figure 7.37.b illustrates that the production rate of RES_{XY} is decreased in order to balance the production rates between RES_{XY} and RES_B .



(a) Increasing production rate of RES_B to balance the production rates of RES_B and RES_{XY}



(b) Decreasing production rate of RES_{XY} to balance the production rates of RES_B and RES_{XY}

Figure 7.37 Balancing production rates of Resource RES_{XY} to achieve its continuous resource utilization

7.3 Summary

This chapter discusses various topics and complexity in scheduling repetitive projects consisting of resource-sharing activities. Examples in this chapter demonstrate the difficulty in maintaining continuous resource utilization in such projects. The

complicated relationships and interactions among precedence, resource availability, and resource continuity constraints diminish the effectiveness of SQS-AL when used blindly.

For SQS-AL to effectively schedule repetitive projects with resource-sharing activities, schedulers must imperatively consider the following questions:

- Are resource-sharing activities directly or indirectly dependent?
- Are there any activities between the resource-sharing activities? What are their production rates compared to the resource-sharing activities' production rates?
- Are resource-sharing activities in the same sequence step? If they are not in the same sequence step, in which processing sequence step should their shared resources' crew lead time be determined?
- What is the order of which simulation models for resource-sharing activities are created in the simulation? Does the order affect resource idle time and schedule?
- Could the working sequence of resources be changed because of determination of resources' CLT? Does the change impact the effectiveness of the already determined CLT?
- Is it possible to eliminate idle time in resource utilization?
- Could a work break be introduced to relax the resource continuity constraints?
- Is it necessary to replace a shared resource by many dedicated resources?

The questions above are a good start in analyzing repetitive projects with resource-sharing activities. Many examples are used to point out different problems associated with the above questions in scheduling such projects. The examples and their

solutions in this chapter show how to tackle the difficulties in scheduling resource-sharing activities and shared resources, case by case. However, neither the above questions nor the example in this chapter are exhaustive. The combinations of these questions and examples result in even larger number of possibilities.

Using SQS-AL to solve scheduling problems of repetitive projects consisting of resource-sharing activities definitely requires thorough analysis of the aforementioned considerations and careful examination of the simulation results. The trial-and-error approach may be necessary to derive an optimal schedule; different scheduling scenarios must be tested to improve both activity and resource schedules. To do so, certain decisions (e.g., considering different sequence steps for shared resources) must be first outlined and their options (e.g., SQS2, SQS3, and SQS4) must be listed prior to the testing in simulation and optimization. The following suggestions should be treated as decision variables in scheduling resource-sharing activities.

- It is recommended to use only dedicated resources first in order to study the proximity (i.e., start dates and working period) of activities sharing the same resource and their interaction in the schedule with their dependents.
- Activities, both resource-sharing and not-sharing, should be modeled in the order of their precedence relationships. Then, the decision of which resource-sharing activities to start first should be determined later after studying the production diagram carefully.
- Working sequence should be specified so that the order of which activities are performed is certain, not by chance. This also prevents an unexpected outcome in the schedule such as idle time from probabilistic activity durations

and their impact on the effectiveness of SQS-AL. Examples of specifying working sequence for shared resources are 1) finishing one resource-sharing activity at a time (e.g., X1, X2, X3, and then Y1, Y2, and Y3) and 2) working alternately between resource-sharing activities (e.g., X1, Y1, X2, Y2, X3, and Y3), given that Activities X and Y share the same resource and are not dependent.

- When resource-sharing activities are in different sequence steps (e.g., X is in SQS2 and Y is in SQS4), it is important to check their effect on the schedule. If using different sequence steps results in different schedule, all sequence steps from the SQS of the earlier-SQS activity (i.e., SQS2) to the later-SQS activity (i.e., SQS4) must be tested for the shared resource.
- When there is a slow-production-rate activity between resource-sharing activities, it is recommended to consider assigning work breaks or balancing the production rates. If the proximity between resource-sharing activities is close (their schedules overlap each other or almost), balancing production rate among activities are recommended. On the other hand, if the proximity is large, introducing work breaks between resource-sharing activities is suggested.

Chapter 8 presents the ChaStrobe application, which facilitates the trial-and-error approach by expediting the processes of simulation code and model creation. In addition, the application offers comprehensive analysis of simulation results.

CHAPTER 8

CHASTROBE APPLICATION

Commencing with Chapters 4 to 7, the Sequence Step Algorithm (SQS-AL) and various topics of repetitive project scheduling have been discussed. In Chapter 4, the fundamental concepts of SQS-AL are introduced. The chapter shows that repetitive activities should be scheduled in sequence step order so that the effect of postponing an activity on its successors' idle time and interruption are unveiled, and thus taken into the account of scheduling the successors. Accordingly, SQS-AL is involved in the repetitive processes of executing simulation, collecting data, and scheduling activities. These repetitions are demonstrated in Chapters 4 and 5. In Chapter 5, the implementation of SQS-AL in simulation system, Stroboscope, is discussed. At the end of Chapter 5, it becomes apparent that it is beneficial to create an application automating SQS-AL's systematic process in solving the problems of repetitive project scheduling.

In Chapters 6 and 7, two advanced ideas of scheduling repetitive projects are discussed: work breaks in repetitive activities (Chapter 6) and resource-sharing activities (Chapter 7). Chapter 6 shows benefits of applying work breaks (deliberated interruptions) in repetitive activities to shorten increased project duration due to the assigned crew lead times. Suggestions of which activities into which a work break should be introduced are given; however, trial-and-error approach of scheduling work breaks between units in a

repetitive activity is apparently inevitable. The modifying simulation model and code is required and repeated several times in order to derive an optimal schedule.

Chapter 7 shows that scheduling resource-sharing activities could be extremely complicated. The complexity of scheduling repetitive projects with resource-sharing activities varies from case to case depending on 1) relationships between activities sharing the same resource and 2) relationships between the resource-sharing activities and their dependents. Similar to the trial-and-error approach in Chapter 6, the repetition of modifying simulation models and code is mandatory in order to derive an optimal schedule for repetitive projects with resource-sharing activities.

From Chapters 6 and 7, it is recommended to establish an application that facilitates the means of generating and modifying simulation models and code conforming to SQS-AL (Chapter 4) and the suggested simulation model templates (Chapter 5). The application must be able to facilitate the trial-and-error approach so that modifying simulation models and executing the simulation to optimize a schedule become easier. According to the aforementioned needs and goals, ChaStrobe has been developed.

8.1 Overview of the ChaStrobe Application

ChaStrobe is an application that automates the process of creating simulation code and models conforming to the concepts of the Sequence Step Algorithm (SQS-AL) and the simulation model templates, discussed in Chapters 4 and 5. ChaStrobe is programmed in Visual Basic for Applications (VBA) for Microsoft Visio. It takes input from users, and generates the simulation model in a Stroboscope Graphic User Interface file (Stroboscope GUI), which is a Visio file with Stroboscope add-ons. When users execute

the generated simulation model, the Stroboscope add-ons will convert the simulation model in Stroboscope GUI to Stroboscope code. Then, the converted code is combined with the simulation code in the Stroboscope GUI and send to the Stroboscope Integrated Development Environment (Stroboscope IDE) for simulation execution. Figure 8.1 shows the processes of using ChaStrobe and Stroboscope to model repetitive projects and solve repetitive project scheduling problems using SQS-AL and Stroboscope.

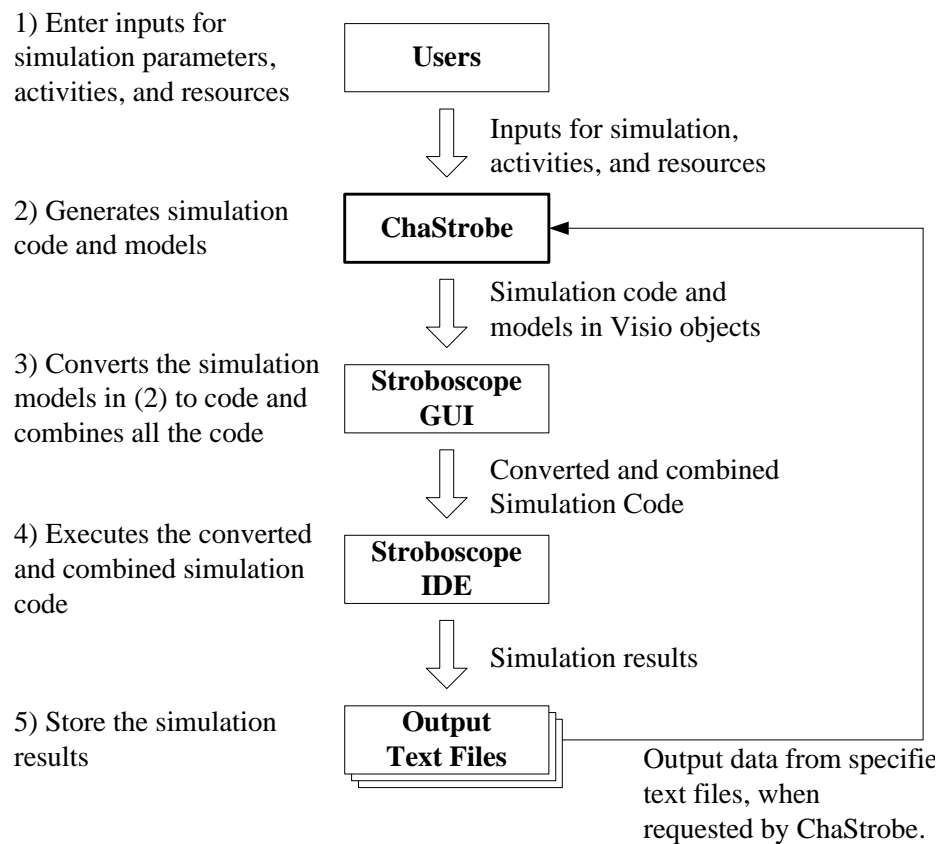


Figure 8.1 ChaStrobe’s process of modeling and solving problems

As shown in Figure 8.1, a user enters inputs for simulation parameters, activities, and resources in ChaStrobe. Then, the user asks ChaStrobe to create a simulation model and code inside the Stroboscope GUI. This process of creating simulation code and

model in Stroboscope GUI normally takes seconds depending on the size of the simulation model (e.g., the number of activities and the number of sequence steps) and the complexity of the repetitive project (e.g., the existence of work breaks and resource-sharing activities). After the model is created, both the graphical model and code can be examined by the user within the Stroboscope GUI. At this point, the simulation code and model can be modified by the user prior to simulation execution.

As depicted in Figure 8.1, between the Stroboscope GUI and IDE, after a user requests to execute the simulation from within the Stroboscope GUI, the graphical simulation elements such as Queues, Combis, and Links are converted to simulation code and then combined to the simulation code, created by ChaStrobe and the code added by users before simulation execution. Then, the combined code is sent to and executed by the Stroboscope IDE. Results from the execution are stored in various text files for further analysis.

8.1.1 Inputs for ChaStrobe

There are four main types of inputs in the ChaStrobe application: 1) simulation parameters, 2) project inputs, 3) optimization parameters, and 4) graphical presentation parameters. Simulation parameters (e.g., the number of replications) and project inputs (e.g., activities and resources) are the most important inputs used to construct simulation code and model. Accordingly, this chapter focuses on simulation parameters and project inputs, while Chapter 9, Optimization in ChaStrobe, focuses on optimization parameters. Graphical presentation parameters for ChaStrobe's Static Graphs are explained in detail in Appendix D.

The following is a description of ChaStrobe's inputs.

- 1) Simulation Parameters are simulation setup input, variables, and constants for establishing and running simulation, not related to specific activities and resources. Figure 8.2 shows ChaStrobe’s user interface for the Simulation Parameters.

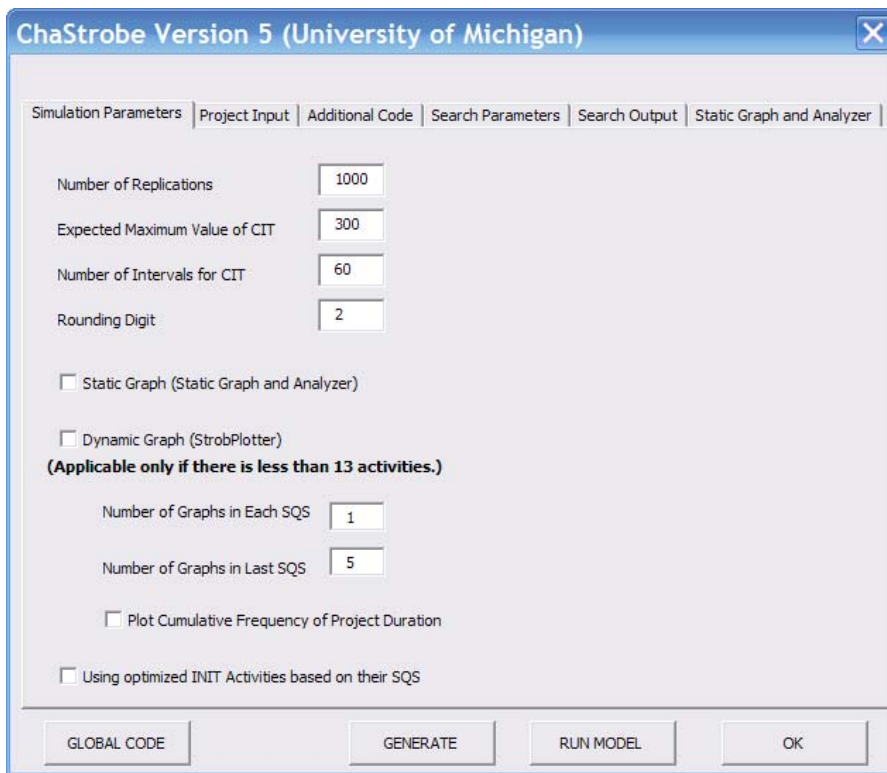


Figure 8.2 ChaStrobe’s interface for Simulation Parameters

The Simulation Parameters in Figure 8.2 are described below:

- a. “Number of Replications” specifies the total number of replications for each SQS-AL processing sequence step. This number is assigned to nRep to control SQS-AL replication loop, discussed in Section 5.3.1.1 (MP.Loops).
- b. “Expected Maximum Value of CIT” specifies the upper bound of BinCollectors such as RES_CIT1, discussed in Section 5.3.2.6 (PO.RES.CIT).

- c. “Number of Intervals for CIT” determines the number of intervals in BinCollectors in (b), discussed in Section 5.3.2.6 (PO.RES.CIT).
- d. “Rounding Digit” specifies the number of decimals used in Stroboscope’s Round function to round decimal places.
- e. “Static Graph” option informs ChaStrobe whether to create Static Graphs at the end of simulation. When this option is checked (true), ChaStrobe generates simulation code storing data, which is used later to create Static Graphs. Details for constructing a Static Graphs are given in Appendix D.
- f. “Dynamic Graph” option informs ChaStrobe whether to create production diagrams during simulation runs. When this option is checked (true), ChaStrobe creates simulation code for the Dynamic Graph.
- g. “Number of (Dynamic) Graphs in Each SQS” indicates how many dynamic graphs will be plotted during processing each sequence step. The data for each dynamic graph are derived from each replication, starting from the first replication. Note that ChaStrobe uses this parameter only if the option of Dynamic Graph in item (f) is checked.
- h. “Number of (Dynamic) Graphs in Last SQS” indicates how many dynamic graphs will be plotted in the last sequence step. Similar to “Number of Graphs in Each SQS,” ChaStrobe uses this parameter only if the option of Dynamic Graph is checked.
- i. “Plot Cumulative Frequency of Project Duration” option informs ChaStrobe whether to create simulation code used in plotting the cumulative frequency of project durations from processing each SQS.

- j. “Using optimized INIT Activities based on their SQS” option is for optimizing SQS-AL’s processing sequence steps in simulation. When this option is checked, only activities in all previous and the current sequence steps are initiated with the number of units, while activities in the later SQS than the current SQS are not initialized. This option is used to expedite SQS-AL, when there is no desire to study changes in crew idle time, project duration, project idle time etc., between different processing sequence steps.
- 2) Project Inputs are data related to activities and resources used to create simulation code and model. Activity data are activities’ names, precedence constraints, production rates, and work amounts. Resource data are resources’ names, resource utilization, resource continuity constraints, and work breaks. Figures 8.3 to 8.6 show ChaStrobe’s user interface for Project Input.
- a. The Precedence Input sheet (Figure 8.3) collects 1) the number of activities, 2) activities’ names, and 3) precedence constraints. ChaStrobe determines the number of activities in the Precedence Input sheet by counting non-blank cells in Column A, starting from Row 2. Then, it collects activities’ names, used to create work flow sub-networks for each activity. For precedence constraints, ChaStrobe determines the number of predecessors by counting non-blank cells in the column direction, starting from Column 2 in the same row of each particular activity.

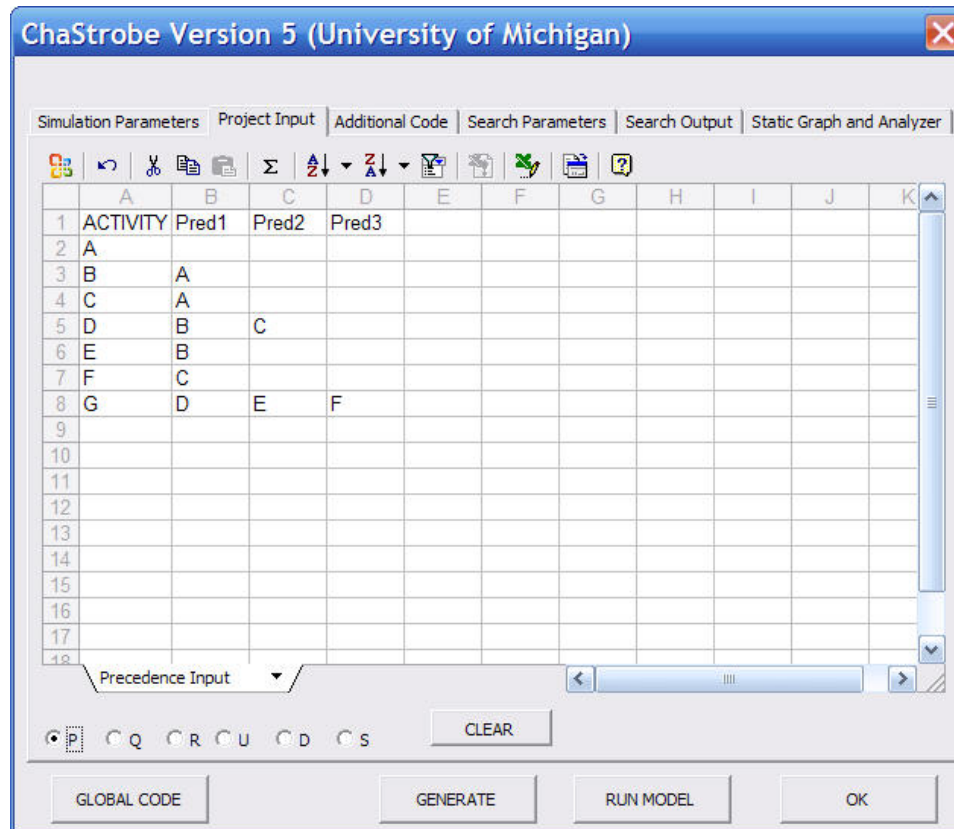


Figure 8.3 ChaStrobe's Interface for activities' names and precedence constraints on the Precedence Input sheet

- b. The Quantity Input sheet in Figure 8.4 collects 1) conversion of productivity of activities in the unit of time per the unit of work (e.g., hr/m), and 2) work amount for each repetitive unit and each activity in units of work per repetitive unit (e.g., m/floor). Therefore the unit of their product is in the unit of time (e.g., hr). On the Quantity Input sheet, ChaStrobe counts the number of non-blank cells in Row 1 in the column direction, starting from Column C, to determine the number of units in the project. Therefore, the project in Figure 8.4 consists of 4 units. ChaStrobe uses the number of units to: 1) create arrays of work amount in simulation code and 2) initialize the number of work units. The conversion of

productivity in Column B (e.g., in Cell B2, 1/Normal[10,1]) must be Stroboscope's expression. Activity durations are derived from the product of the conversion in Column B and the work amount in Columns C, D, E, and F. For more information about simulation code generated from data in the Quantity Input sheet, see Section 5.3.1.2 (MP.ACT.Quantity) for arrays of work amounts, Section 5.3.4.2 (CS.ACT.INIT) for initializing the number of repetitive units, and Section 5.3.2.3 (PO.ACT.Duration) for activities' durations.

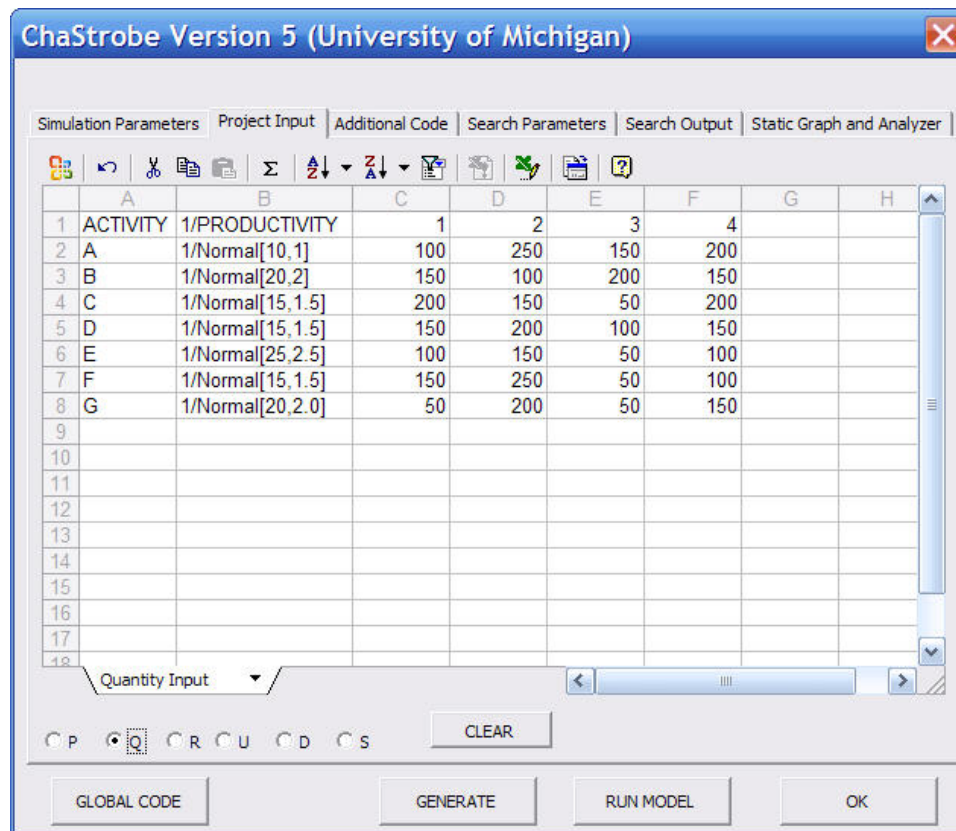


Figure 8.4 ChaStrobe's interface for activities' productivities and work amounts on the Quantity Input Sheet

- c. The Resource Input sheet (Figure 8.5) collects data related to resources in repetitive projects. Resource-related inputs for each resource in Resource

Input sheet are 1) resources' names, 2) confidence levels, 3) the total number of resources for each type, and 4) continuity constraints. ChaStrobe first determines the number of resources by counting non-blank cells in Column A starting from Row 2. Then, it records the resources' names in Column A and stores resource's properties in different columns. Confidence levels and the number of resources for each resource type are collected from Columns B and C, respectively.

Continuity constraints for each resource are constructed using notation of *Break*. Breaks indicate series of continuous utilization of resources. For example, in Figure 8.5, resource ResB's Break 1 in Column H is B-4, meaning that ResB is scheduled to work continuously from the day it arrives to the day it finishes Activity B4. On the other hand, blank cells in Column H indicate resources are not constrained by continuity constraints; therefore, the early start date is used to schedule the unconstrained resources. ChaStrobe determines the number of breaks by counting non-blank cells in the column direction, starting from Column H.

Simulation code for confidence levels is discussed in Section 5.3.1.3 (MP.RES.ConfidenceLevel) and Section 5.3.4.6 (CS.RES.CLT). For work breaks in continuous resource utilization, see simulation code in Section 5.3.3.7 (CME.RES.Leave.OnFlow) for a project without work breaks, and Section 6.5.3.7 for the project with work breaks.

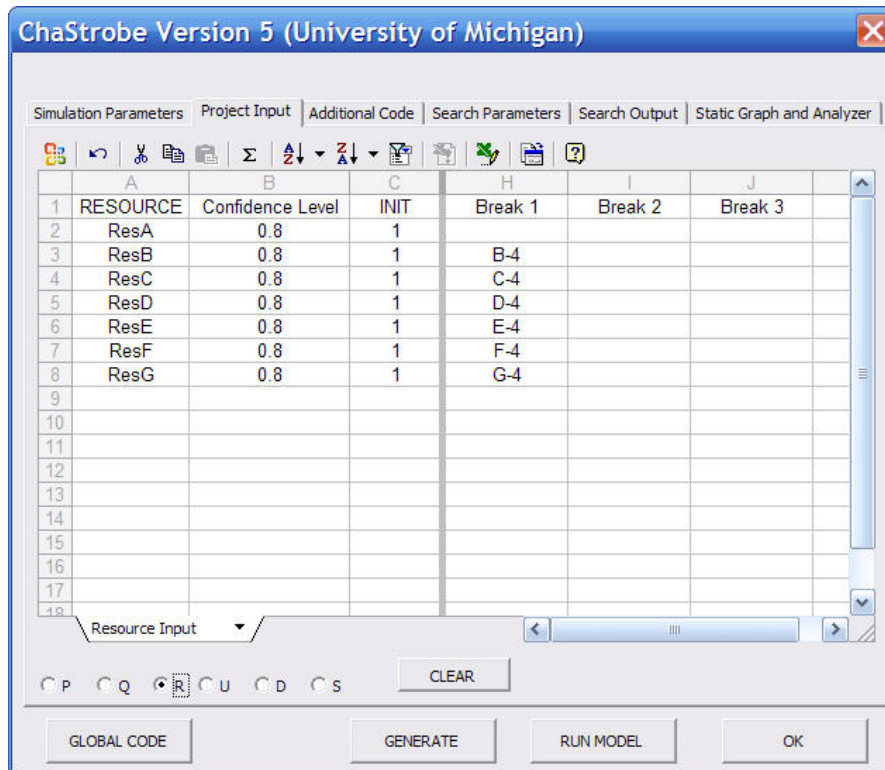


Figure 8.5 Resources' names, confidence levels, amounts for each type, and continuity constraints on the Resource Input Sheet

- d. The Utilization Input sheet is used to assign resource(s) to activities. ChaStrobe uses row index in identifying activities in the Utilization Input and Precedence Input sheets; therefore, the order of activities on both sheets must be the same. Column A on the Utilization Input sheet is only for reference purposes; omitting activities' names in Column A on the Utilization Input sheet does not result in an error. For example, as shown in Figure 8.6, after ChaStrobe reads resource's name (ResA) in Row 2 Column B on the Utilization Input sheet, it assigns Resource ResA to an activity declared in Row 2 on the Precedence Input sheet, which is Activity A as shown in Figure 8.4. Then, ChaStrobe moves to Row 2 Column C on the Utilization Input sheet; however, since there is no

resource's name in Row 2 Column C, ChaStrobe will move to Row 3 Column B. Then, it will assign Resource ResB to an activity declared in the same row index, which is Row 3, on the Precedence Input sheet, which is Activity B as shown in Figure 8.4.

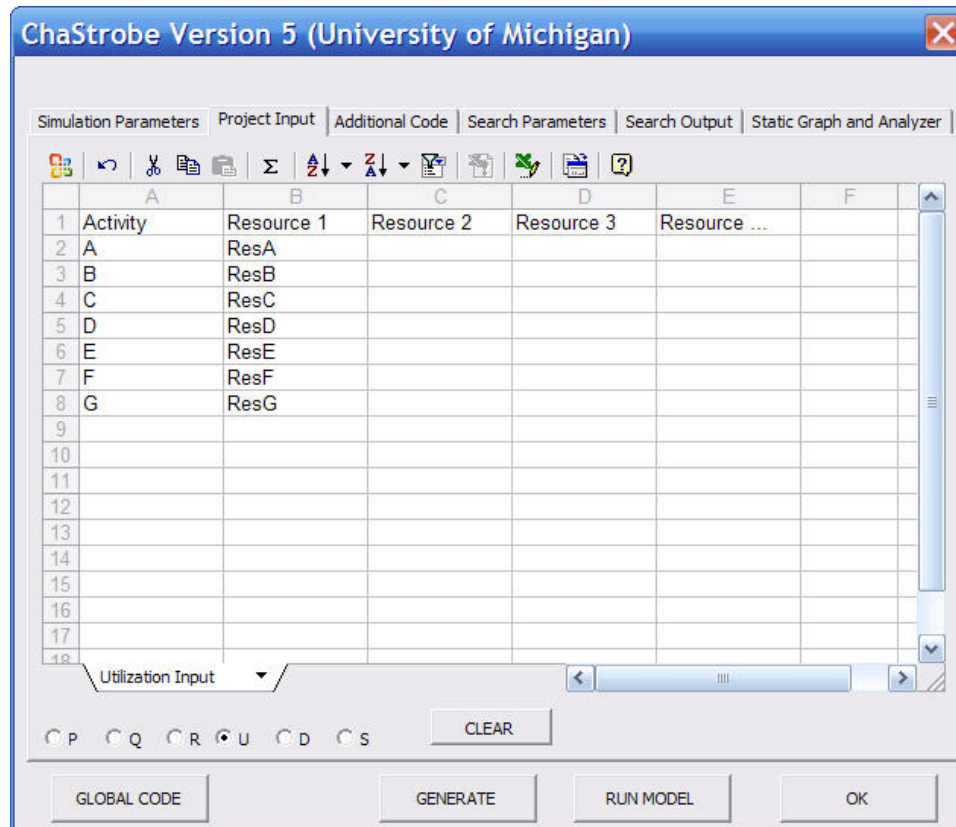


Figure 8.6 ChaStrobe's interface for Utilization Input

Note that the Dynamic Code Input and Search Input sheets are for optimization, discussed in Chapter 9, Optimization in ChaStrobe.

After Simulation Parameters and Project Input (Precedence, Quantity, Resource, and Utilization Input) are entered, simulation code and model for the problem are ready to be created. To generate the simulation model, users click the "GENERATE" button (Figure 8.6). To execute the simulation, users click the "RUN MODEL" button.

8.1.2 Simulation Output from ChaStrobe

ChaStrobe stores simulation results related to activities, resources, and projects in six different text files. After simulation execution, the six text files are ready for viewing. Their main functions are explained below.

- 1) tempChaStrobe_Analyzer.txt stores simulation results related to activities, such as start date and duration, for the finalized schedule in each replication. ChaStrobe uses the activity data in tempChaStrobe_Analyzer to analyze and compare the results among SQS-AL, CPM, and RSM. Details of ChaStrobe Analyzer are discussed in Section 8.10.
- 2) tempChaStrobe_CLT.txt stores resources' crew lead times.
- 3) tempChaStrobe_GA.txt stores objective function values and user-specified output from simulation execution, used for optimization, discussed in detail in Chapter 9, Optimization in ChaStrobe.
- 4) tempChaStrobe_StaticGraph.txt stores data of activities from the first replication when processing each sequence step. ChaStrobe's Analyzer uses this data to create example production diagrams.
- 5) tempChaStrobe_StaticGraphProb.txt stores simulation results of probability of project duration for processing each sequence step. ChaStrobe uses this data to create cumulative frequency of project duration from processing each sequence step.
- 6) tempChaStrobe_StaticGraphSQS.txt stores simulation results of project duration and project idle time from processing each sequence step.

8.1.3 Automation in ChaStrobe

ChaStrobe uses Visual Basic for Applications (VBA) and Object and Linking Embedding Automation (OLE) in order to control MS Visio, MS Excel, and MS Project, to 1) create graphical presentations, 2) execute computational analyses, and 3) perform optimization. Upon users' requests, ChaStrobe retrieves the data stored in output text files, performs the necessary tasks, and returns the results when it finishes. Figure 8.7 explains the processes of analyzing the data from output files, and also applications used to present the results.

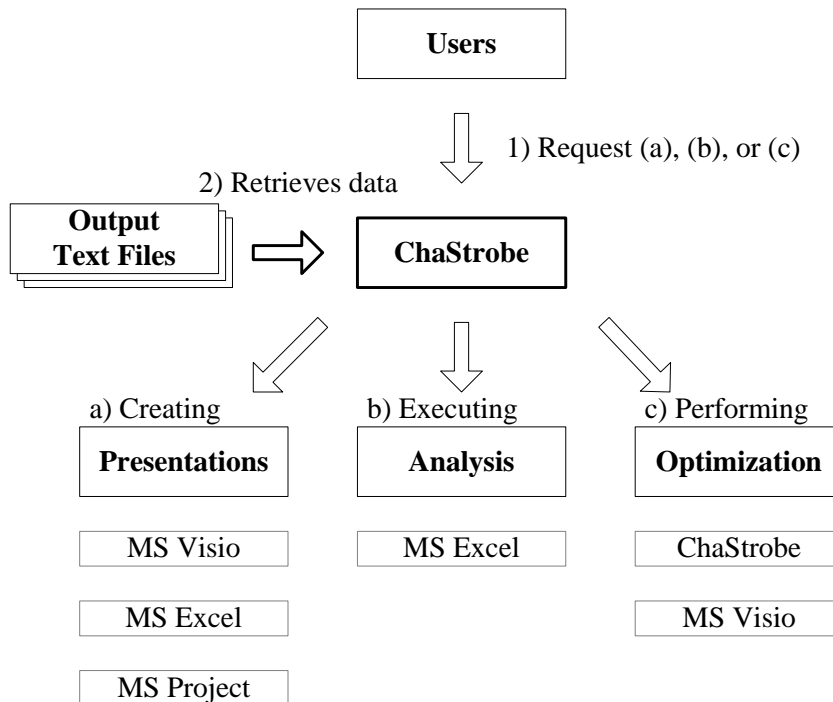


Figure 8.7 ChaStrobe's presentations, analyses, and optimization

Figure 8.7 indicates that the graphical presentations are displayed in MS Visio, MS Excel, and MS Project, depending on the type of presentation. Analysis in MS Excel focuses on the comparison of results among CPM, RSM, and SQS-AL. For the optimization in ChaStrobe, both ChaStrobe and MS Visio with Stroboscope GUI are

required as shown in Figure 8.7; it is a recursive procedure of creating simulation models, executing the models in Stroboscope GUI, retrieving output from tempChaStrobe_GA, and then performing optimization-related calculations in ChaStrobe. ChaStrobe's Optimization and ChaStrobe's Automation are discussed in Chapter 9.

8.1.4 Capabilities of ChaStrobe

According to the aforementioned ChaStrobe's features, the ChaStrobe application is capable of:

- 1) Generating simulation model and code conforming to SQS-AL for repetitive projects with:
 - a. Probabilistic activity durations (Chapters 4 to 7)
 - b. Dedicated resources (Chapters 4 and 5)
 - c. Work breaks (Chapter 6)
 - d. Resource-sharing activities (Chapter 7)
- 2) Automatically and dynamically modifying simulation code and model by using Dynamic Simulation Modeling and Dynamic Coding features in ChaStrobe, discussed in Chapter 9.
- 3) Retrieving output from specified text files to:
 - a. Create graphical presentations from the output
 - b. Perform comparison analyses of the Critical Path Method (CPM), the Repetitive Scheduling Method (RSM), and the Sequence Step Algorithm (SQS-AL)
 - c. Obtain objective function values for optimization in ChaStrobe: 1) Genetic Algorithm Optimization, and 2) Exhaustive Search

- 4) Creating graphical presentations as follows:
 - a. Production diagrams from different processing sequence steps
 - b. Cumulative distribution function of project duration based on CPM, RSM, and SQS-AL
 - c. Density functions of project durations from CPM, RSM, and SQS-AL
 - d. Comparisons among CPM, RSM, and SQS-AL in project duration, and project idle time
- 5) Performing optimization using:
 - a. Genetic Algorithm
 - b. Exhaustive Search

8.2 Examples of Repetitive Projects in ChaStrobe

Four examples of repetitive projects demonstrate the use of the ChaStrobe application. Ordered from simple to complicated, the four examples are:

- 1) Example 8.1 Simple repetitive project. In this example, each activity utilizes one dedicated resource. There are no work breaks or resource-sharing activities.
- 2) Example 8.2 Repetitive project with work breaks. This example focuses on using ChaStrobe to model repetitive activities with work breaks. There is no resource-sharing activity.
- 3) Example 8.3 Repetitive project with resource-sharing activities. This example focuses on using ChaStrobe to model resource-sharing activities. In this example, the difficulty in scheduling repetitive projects with resource-sharing activities is shown.

To solve Example 8.3 effectively, the inputs in Example 8.3 must be modified as shown in Example 8.4. However, Example 8.3 is shown for the purposes of comparison between two schedules derived from the input in Examples 3 and 4.

- 4) Example 8.4 Repetitive project with work breaks and resource-sharing activities. This is a very complicated scheduling problem of repetitive projects, since both work breaks and resource-sharing activities are included in the simulation model.

To activate ChaStrobe, right click on the Visio drawing with ChaStrobe macro. Then, select CHASTROBE, as shown in Figure 8.8 below.

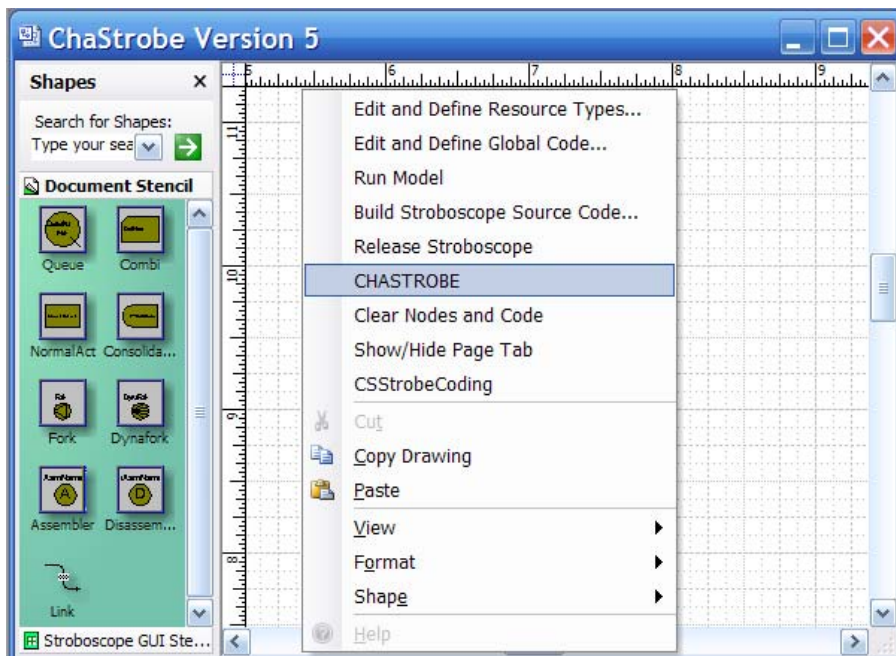


Figure 8.8 Activating ChaStrobe

8.2.1 Example 8.1 Simple repetitive project

Example 8.1 is the example used in Section 5.3. Each activity requires only one dedicated resource, and there is no work break. This example shows how to use

ChaStrobe to model a simple repetitive project. The comparison between the simulation code in section 5.3 and the input in the following section should be beneficial to users when they desire to modify the default code and model created by ChaStrobe.

Figure 8.9 displays a single unit precedence diagram for Example 8.1, whereas Table 8.1 shows each activity's productivity and work amount in each unit.

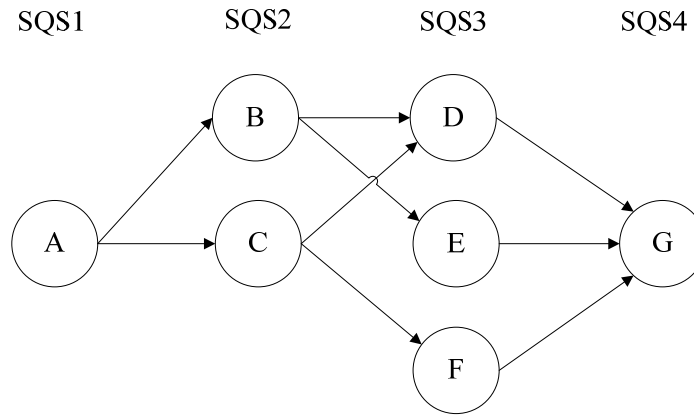


Figure 8.9 A single unit precedence diagram for Example 8.1

Activity	Productivity	Unit			
		1	2	3	4
		Work Amount			
A	Normal[10,1]	100	250	150	200
B	Normal[20,2]	150	100	200	150
C	Normal[15,1.5]	200	150	50	200
D	Normal[15,1.5]	150	200	100	150
E	Normal[25,2.5]	100	150	50	100
F	Normal[15,1.5]	150	250	50	100
G	Normal[20,2.0]	50	200	50	150

Table 8.1 Activities' productivities and work amounts for Example 8.1

The following sections show the input for Example 8.1 in ChaStrobe. Figure 8.10 shows Simulation Parameters for the example. The total number of replications is 3000; therefore, nRep SaveValue equals 3000 as shown in Section 5.3.1.1 (MP.Loops). The Maximum Value in RES_CIT is set at 300, meaning the upper bound of RES_CIT BinCollectors is set to 300 days. The number of intervals in these BinCollectors is 60 as

shown in the Intervals in RES_CIT bincollector in Figure 8.10. The Maximum Value in RES_CIT and the Intervals in RES_CIT bincollector are used in the BinCollectors' declaration in Programming Objects. See Sections 5.3.2.6 (PO.RES.CIT) and 5.3.2.8 (PO.RES.CIT.SQS) for more detail.

In Figure 8.10, the option of Static Graph is checked (true) so that ChaStrobe will create simulation code for storing data necessary to construct Static Graphs in tempChaStrobe_StaticGraph.txt.

Figure 8.10 Simulation Parameters for Example 8.1

Note that, the total number of processing sequence steps is automatically determined by ChaStrobe, which is 4 in this example. Thus, nSQS SaveValue is 4 as shown in Section 5.3.1.1 (MP.Loops).

Figure 8.11 shows the input of activities' names and predecessors entered in ChaStrobe's Precedence Input sheet. Activities' names are entered in Column A starting from Row 2, while predecessors of activities are entered in the row direction starting

from Column B. ChaStrobe interprets the input on the Precedence Input sheet to create work flow sub-networks for each activity and also their precedence constraints. Note that precedence constraints are encoded in activities' semaphores, as shown in Section 5.3.2.2 (PO.ACT.Semaphore).

	A	B	C	D	E	F	G	H	I	J	K
1	ACTIVITY	Pred1	Pred2	Pred3							
2	A										
3	B	A									
4	C	A									
5	D	B	C								
6	E	B									
7	F	C									
8	G	D	E	F							
9											
10											
11											
12											
13											
14											
15											
16											
17											
18											

Figure 8.11 Precedence Input for Example 8.1

Figure 8.12 shows the input of activities' production rates and work amounts in each unit; the input is entered in the Quantity Input sheet. The data on the Quantity Input sheet is used mainly for 1) determining the total number of units, 2) creating Arrays storing activities' work amounts in each unit, and 3) constructing simulation code for activities' durations. On the Precedence Input sheet, ChaStrobe collects the inverse value of productivity from Column B. It is mandatory to enter the input of activities in the same order as activities in the Precedence Input sheet, because ChaStrobe uses the row index as a reference to collect activities' properties (production rates and work amounts).

For the work amount, ChaStrobe collects data in the row direction starting from Column C. In order to determine the number of units, ChaStrobe counts non-blank cells in Row 1, starting from Column C. Hence, ChaStrobe recognizes 4 units in this example.

Note that the simulation code for the work amount is explained in Section 5.3.1.2 (MP.ACT.Quantity), and the simulation code for activities' productivity is explained in Section 5.3.2.3 (PO.ACT.Duration).

	A	B	C	D	E	F	G	H
1	ACTIVITY	1/PRODUCTIVITY	1	2	3	4		
2	A	1/Normal[10,1]	100	250	150	200		
3	B	1/Normal[20,2]	150	100	200	150		
4	C	1/Normal[15,1.5]	200	150	50	200		
5	D	1/Normal[15,1.5]	150	200	100	150		
6	E	1/Normal[25,2.5]	100	150	50	100		
7	F	1/Normal[15,1.5]	150	250	50	100		
8	G	1/Normal[20,2.0]	50	200	50	150		
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								

Figure 8.12 Quantity Input for Example 8.1

Figure 8.13 shows the resource-related input on the Resource Input sheet. The Resource Input sheet is one of the most important input sheets, since it collects resource data and resource continuity constraints. Starting from Row 2, ChaStrobe collects resources' names and confidence levels from Columns A and B, respectively. For Example 8.1, there are 7 resources, as shown in Figure 8.13. The confidence level for each resource is 80%. The number of resources for each resource type is in Column C. For Example 8.1, there is only one set of resource for each type, as shown in Figure 8.13.

To model continuity constraints, ChaStrobe collects user-specified breaks in activities and units, defined in “Activity’s Name - Unit’s ID”. The breaks in Column H inform ChaStrobe how to model the continuity constraints for each resource. The breaks indicate when resources are scheduled to leave the site. During each period that resources remain on the site, resource continuity constraints are applied. Consequently, the breaks implicitly indicate a series of continuous resource utilization for the resource. For more explanation about breaks and series of continuous resource utilization, see Chapter 7, Work Breaks.

	A	B	C	H	I	J	K
1	RESOURCE	Confidence Level	INIT	Break 1	Break 2	Break 3	...
2	ResA	0.8	1				
3	ResB	0.8	1	B-4			
4	ResC	0.8	1	C-4			
5	ResD	0.8	1	D-4			
6	ResE	0.8	1	E-4			
7	ResF	0.8	1	F-4			
8	ResG	0.8	1	G-4			
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							

Figure 8.13 Resource Input for Example 8.1

Note that simulation code for collecting CIT and determining CLT are discussed in Section 5.3.3.7 (CME.RES.Leave.OnFlow) and Section 5.3.4.6 (CS.RES.CLT). Simulation code for confidence levels is explained in Section 5.3.1.3 (MP.RES.ConfidenceLevel), and simulation code for the number of resources for each type is explained in Section 5.3.2.7 (CS.RES.INIT).

Figure 8.14 shows data of resource utilization in the Utilization Input sheet for Example 8.1. These data in the Utilization Input sheet are used to connect work flow sub-networks to resource flow sub-networks. ChaStrobe links sub-networks of activities in Column A to sub-networks of resources in Columns B, C, and so forth.

	A	B	C	D	E	F	G
1	Activity	Resource 1	Resource 2	Resource 3	Resource ...		
2	A	ResA					
3	B	ResB					
4	C	ResC					
5	D	ResD					
6	E	ResE					
7	F	ResF					
8	G	ResG					
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							

Figure 8.14 Utilization Input for Example 8.1

After entering Simulation Input and Project Input, the entered inputs are ready for the generation of simulation code and model. To generate simulation model and code, users click the “GENERATE” button. To execute the simulation, users click the “RUN MODEL” button. Figure 8.15 is the production diagram from the first replication in SQS5. As shown in the figure, all activities have no interruption between units at least in this replication. More details about this example, solution, and simulation model can be found in Chapters 4 and 5.

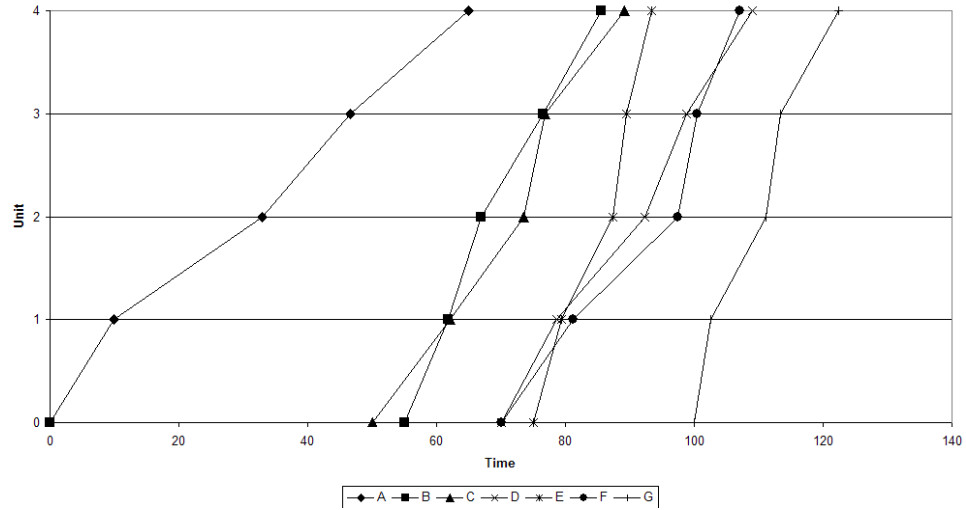


Figure 8.15 Production diagram from the 1st replication in SQS5 for Example 8.1

8.2.2 Example 8.2 Repetitive project with work breaks

Example 8.2 demonstrates how to use ChaStrobe to model repetitive projects with multiple work breaks. Example 8.2 is the same example in Chapter 6, which the final schedule consists of three work breaks at the end of Activities B in Unit 5, C in Unit 4, and G in Unit 7. Entering project inputs for Example 8.2 is done in the same way as shown in Example 8.1, except for the input for work breaks on the Resource Input sheet.

Figure 8.16 presents a single unit precedence diagram for Example 8.2; precedence relationships from the figure are entered in the Precedence Input sheet as shown in Figure 8.18. Table 8.2 displays the work amount for each unit and the productivity; these two sets of data are entered in the Quantity Input sheet as shown in Figure 8.19. Figure 8.20 presents input for resource utilization on the Utilization Input sheet for Example 8.2. Simulation parameters for Example 8.2 are shown in Figure 8.17.

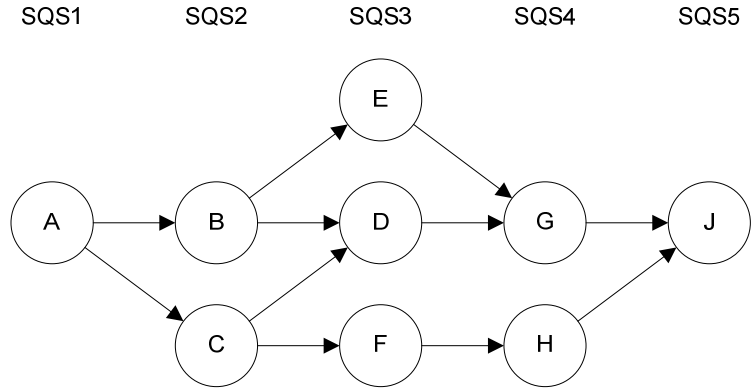


Figure 8.16 Single unit precedence diagram for Example 8.2

Act	Resource Production Rate	Unit									
		1	2	3	4	5	6	7	8	9	10
		Work Amount									
A	Normal[20,2]	200	200	200	200	200	200	400	400	400	400
B	Normal[30,3]	150	150	100	100	100	100	100	100	100	100
C	Normal[30,3]	250	200	200	250	300	200	350	400	200	350
D	Normal[15,1.5]	300	400	400	450	300	300	250	250	250	400
E	Normal[20,2]	150	150	150	150	150	150	200	200	200	200
F	Normal[25,2.5]	350	400	300	350	150	200	400	250	300	250
G	Normal[30,3.0]	150	150	150	150	300	250	300	300	300	450
H	Normal[20,2]	200	300	300	200	250	400	300	400	300	250
J	Normal[15,1.5]	200	200	200	200	200	200	300	300	300	300

Table 8.2 Activities' productivities and work amounts for Example 8.2

Simulation Parameters | Project Input | Additional Code | Search Parameters | Search Output | Static Graph and Analyzer

Number of Replications: 1000

Expected Maximum Value of CIT: 500

Number of Intervals for CIT: 100

Rounding Digit: 0

Static Graph (Static Graph and Analyzer)

Dynamic Graph (StrobPlotter)

(Applicable only if there is less than 13 activities.)

Number of Graphs in Each SQS: 1

Number of Graphs in Last SQS: 5

Plot Cumulative Frequency of Project Duration

Using optimized INIT Activities based on their SQS

Figure 8.17 Simulation Parameters for Example 8.2

Simulation Parameters | Project Input | Additional Code | Search Parameters | Search Output | Static Graph and Analyzer

	A	B	C	D	E	F	G	H	I	J	K
1	ACTIVITY	Pred1	Pred2	Pred3							
2	A										
3	B	A									
4	C	A									
5	D	B	C								
6	E	B									
7	F	C									
8	G	D	E								
9	H	F									
10	J	G	H								
11											
12											
13											
14											
15											
16											
17											
18											

Precedence Input

Figure 8.18 Precedence Input for Example 8.2

	A	B	C	D	E	F	G	H	I	J	K	L
1	ACT	1/PRODUCTIVITY	1	2	3	4	5	6	7	8	9	10
2	A	1/Normal[20,2]	200	200	200	200	200	200	400	400	400	400
3	B	1/Normal[30,3]	150	150	100	100	100	100	100	100	100	100
4	C	1/Normal[30,3]	250	200	200	250	300	200	350	400	200	350
5	D	1/Normal[15,1.5]	300	400	400	450	300	300	250	250	250	400
6	E	1/Normal[20,2]	150	150	150	150	150	150	200	200	200	200
7	F	1/Normal[25,2.5]	350	400	300	350	150	200	400	250	300	250
8	G	1/Normal[30,3.0]	150	150	150	150	300	250	300	300	300	450
9	H	1/Normal[20,2]	200	300	300	200	250	400	300	400	300	250
10	J	1/Normal[15,1.5]	200	200	200	200	200	200	300	300	300	300

Figure 8.19 Quantity Input for Example 8.2

	A	B	C	D	E	F	G	H
1	Activity	Resource 1	Resource 2	Resource 3	Resource ...			
2	A	ResA						
3	B	ResB						
4	C	ResC						
5	D	ResD						
6	E	ResE						
7	F	ResF						
8	G	ResG						
9	H	ResH						
10	J	ResJ						

Figure 8.20 Utilization Input for Example 8.2

Figures 8.21.a and 8.21.b present two different inputs on the Resource Input sheet for Example 8.2, one with workout work breaks (Figure 8.21.a) and another one with work breaks (Figure 8.21.b). Figure 8.21.a is present here to show the difference between the inputs for resources with and without work breaks. The schedule derived from the

Resource Input sheet in Figure 8.21.a does not have work breaks, while the schedule derived from the Resource Input sheet in Figure 8.21.b has three work breaks. In Figure 8.21.a, resources are scheduled to work continuously from the start of their activities in Unit 1 to the completion in Unit 10. There is no work break specified in Figure 8.21.a. The solid borders in Figure 8.21.a indicate cells that will be modified in order to include work breaks into their resource schedule, as shown in Figure 8.21.b.

1	RESOURCE	Confidence Level	INIT	Break 1	Break 2	Break 3	...
2	ResA	0.8	1				
3	ResB	0.8	1	B-10			
4	ResC	0.8	1	C-10			
5	ResD	0.8	1	D-10			
6	ResE	0.8	1	E-10			
7	ResF	0.8	1	F-10			
8	ResG	0.8	1	G-10			
9	ResH	0.8	1	H-10			
10	ResJ	0.8	1	J-10			

(a) One continuous work series for each resource

1	RESOURCE	Confidence Level	INIT	Break 1	Break 2	Break 3	...
2	ResA	0.8	1				
3	ResB	0.8	1	B-5	B-10 date		
4	ResC	0.8	1	C-4	C-10 date		
5	ResD	0.8	1	D-10			
6	ResE	0.8	1	E-10			
7	ResF	0.8	1	F-10			
8	ResG	0.8	1	G-7	G-10 date		
9	ResH	0.8	1	H-10			
10	ResJ	0.8	1	J-10			

(b) Two continuous work series for Resources ResB, ResC, and ResG

Figure 8.21 Resource Inputs with a different number of work series in ResB, ResC, and ResG for Example 8.2

In contrast to Figure 8.21.a, Resource Input in Figure 8.21.b specifies three work breaks at the end of Activities B5, C4, and G7. The solid borders in Figure 8.21.b emphasize cells modified from Figure 8.21.a in order to specify work breaks in the resource schedule. According to the input on the Resource Input sheet in Figure 8.21.b, Resource ResB is scheduled to work continuously from the start of B1 to the completion of B5, and then takes a break. After the break, ResB will return to the site to work on a specified-return date, which will be determined by SQS-AL. After the break, ResB will work continuously from the start of B6 to the completion of B10. Resources ResC and ResG are scheduled to work and take a break in the same way as ResB; ResC will take a break at the completion of C4; ResG will take a break at the completion of G7. Accordingly, there are three work breaks and three specified-return dates.

There are two types of work breaks in ChaStrobe: 1) a work break with a fixed return date and 2) a work break with a fixed duration (see Chapter 6, Work Breaks, for more detail). A work break with a fixed return date determines when the resource returns to the site after the break (e.g., returning back on May 20). Using this fixed-return-date option, resources will return to the site on the same date. Therefore, work break duration for a fixed-return-date work break itself varies depending on when the resource finishes the unit before the break and the fixed return date after the break. To inform ChaStrobe to construct simulation code and model for a fixed-return-date work break, users must include the keyword “date” after activities’ names and units’ IDs, as shown in Figure 8.21.b (i.e., Cells I2, I3 and I9).

Note that the keyword for work break types must be included in the cells specifying the continuous series of sub-activities after the break (e.g., B10 in Cell I3 in

Figure 8.21.b), since the break determines when the resource will return to work on the next unit (e.g., B6). The first continuous series of sub-activities for Activity B in Figure 8.21.b is from B1 to B5, and the second continuous series is from B6 to B10.

A work break with a fixed duration specifies how long the resource will take a break (e.g., taking a break for 15 days). Using the fixed-duration work break option, resources always take the exact same work break duration. The later resources finish the unit before the break, the later the resources return to the site. To specify a fixed duration of a work break in ChaStrobe, users include the keyword “duration” after activities’ names and units’ IDs.

Note that if users do not include either keyword “date” or “duration,” ChaStrobe will use the fixed-duration work break option, since the simulation code and model for the fixed-duration option is similar to those for repetitive projects without work breaks. For more detail about work breaks and calculation, see Chapter 6, Work Breaks.

After entering Simulation Input and Project Input, the entered inputs are ready for simulation code and model generation. To generate the simulation model, users click the “GENERATE” button. To execute the simulation, users click the “RUN MODEL” button. Figure 8.22 is the production diagram from the first replication in SQS5. As shown in the figure, all activities have no interruption between units at least in this replication. The three specified work breaks the lags between B5 and B6, between C4 and C5, and between G7 and G8 in the figure. More details about this example, solution, and simulation model can be found in Chapter 6.

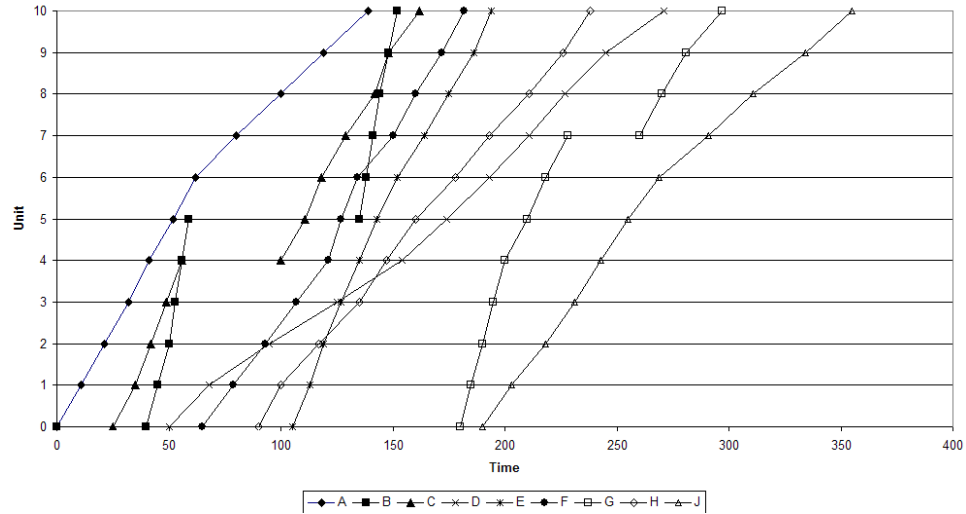


Figure 8.22 Production diagram from the 1st replication in SQS6 for Example 8.2

8.2.3 Example 8.3 Repetitive project with resource-sharing activities

Example 8.3 demonstrates how to use ChaStrobe to model repetitive projects with resource-sharing activities. ChaStrobe can model, solve, and optimize scheduling problems of repetitive projects with resource-sharing activities and probabilistic activity durations. Nevertheless, schedulers should be aware of complicated situations that may arise due to the existence of resource-sharing activities, as discussed in Chapter 7, Resource-Sharing Activities. Example 8.3 illustrates how the complexity of resource-sharing activities could diminish the effectiveness of SQS-AL, discussed in detail in Chapter 7.

Figure 8.23, a single unit precedence diagram, is not a good representative for repetitive projects with resource-sharing activities, because the information of resource utilization is missing; the relationships between shared resources and resource-sharing activities are not presented in Figure 8.23. Therefore, Figure 8.24, a precedence diagram

with resource nodes, is established to present the missing information of resource utilization.

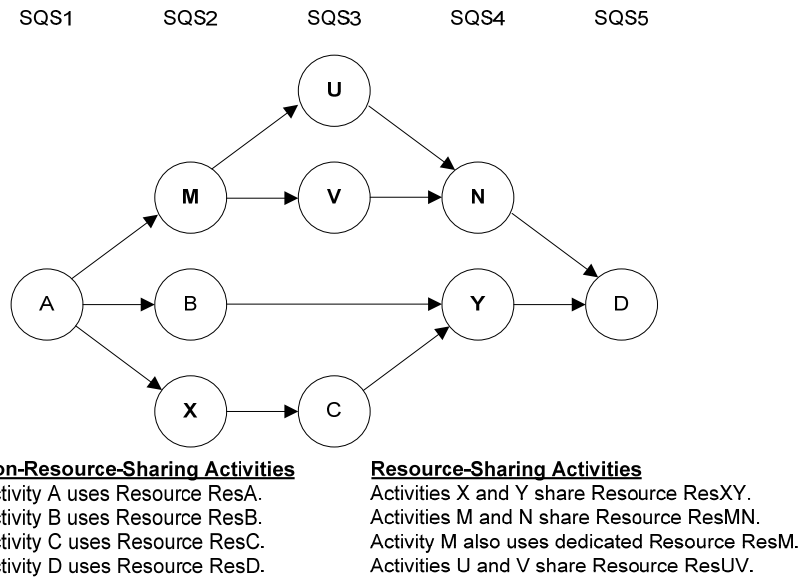


Figure 8.23 Single unit precedence diagram for Example 8.3 and 8.4

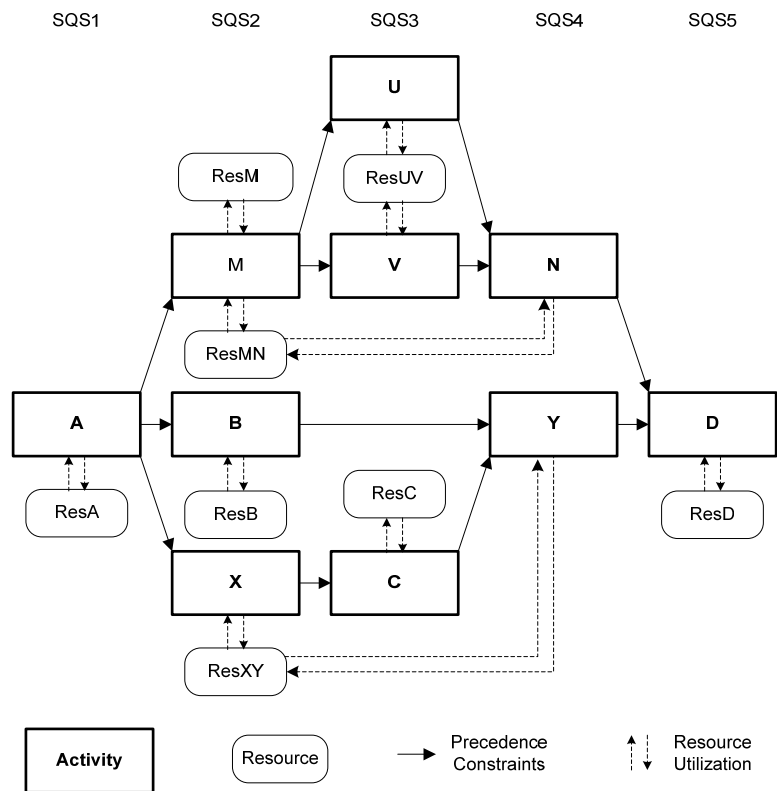


Figure 8.24 Single unit precedence diagram with resource nodes for Examples 8.3 and 8.4

Table 8.3 displays activity durations and variability in the durations. For Examples 8.3 and 8.4, each activity duration is the product of duration and variability following a normal distribution with a standard deviation of 10% (e.g., Normal[1,0.1] in Table 8.3, which 1 is a mean value and 0.1 is a standard deviation).

ACT	Variability	Unit				
		1	2	3	4	5
		Duration				
A	Normal[1,0.1]	40	45	40	40	45
M	Normal[1,0.1]	15	15	10	10	10
B	Normal[1,0.1]	50	40	50	50	40
X	Normal[1,0.1]	20	30	25	20	20
U	Normal[1,0.1]	15	20	15	25	20
V	Normal[1,0.1]	40	40	45	45	40
C	Normal[1,0.1]	15	15	15	15	15
N	Normal[1,0.1]	20	25	30	20	25
Y	Normal[1,0.1]	20	20	20	20	20
D	Normal[1,0.1]	45	35	40	40	30
A	Normal[1,0.1]	40	45	40	40	45

Table 8.3 Durations and variability for activities in Examples 8.3 and 8.4

Figure 8.25 presents Simulation Parameters for Examples 8.3 and 8.4. Notice that the options of Static Graphs and Dynamic Graphs are checked, because it is necessary to analyze the final solution as well as changes in project duration and project idle time from processing one sequence step to another.

Simulation Parameters	Project Input	Additional Code	Search Parameters	Search Output	Static Graph and Analyzer
Number of Replications	<input type="text" value="1000"/>				
Expected Maximum Value of CIT	<input type="text" value="1000"/>				
Number of Intervals for CIT	<input type="text" value="200"/>				
Rounding Digit	<input type="text" value="0"/>				
<input checked="" type="checkbox"/> Static Graph (Static Graph and Analyzer)					
<input checked="" type="checkbox"/> Dynamic Graph (StrobPlotter)					
(Applicable only if there is less than 13 activities.)					
Number of Graphs in Each SQS	<input type="text" value="1"/>				
Number of Graphs in Last SQS	<input type="text" value="1"/>				
<input type="checkbox"/> Plot Cumulative Frequency of Project Duration					
<input type="checkbox"/> Using optimized INIT Activities based on their SQS					

Figure 8.25 Simulation Parameters for Examples 8.3 and 8.4

Figures 8.26, 8.27, and 8.28 present the Precedence Input, Quantity Input, and Utilization Input for Examples 8.3 and 8.4. The differences in the inputs between Examples 8.3 and 8.4 are how resources are scheduled. In Example 8.3, resource work orders are not fixed, and there is no work break. On the other hand, working sequences of resource-sharing activities in Example 8.4 are fixed, and there are scheduled three work breaks. The fixed work orders in Example 8.4 ensure that the resource performs work in the same working sequence from processing its sequence step to the final sequence step. The three work breaks in Example 8.4 eliminate idle time in ResMN serving Activities M and N, which are separated by Activity U having a slower production rate, compared to M and N.

	A	B	C	D	E	F	G	H	I	J	K
1	ACTIVITY	Pred1	Pred2	Pred3							
2	A										
3	M	A									
4	B	A									
5	X	A									
6	U	M									
7	V	M									
8	C	X									
9	N	U	V								
10	Y	B	C								
11	D	N	Y								
12											
13											
14											
15											
16											
17											
18											

Figure 8.26 Precedence Input for Examples 8.3 and 8.4

	A	B	C	D	E	F	G	H
1	ACT	1/PRODUCTIVITY	1	2	3	4	5	
2	A	Normal[1,0.1]	40	45	40	40	45	
3	M	Normal[1,0.1]	15	15	10	10	10	
4	B	Normal[1,0.1]	50	40	50	50	40	
5	X	Normal[1,0.1]	20	30	25	20	20	
6	U	Normal[1,0.1]	15	20	15	25	20	
7	V	Normal[1,0.1]	40	40	45	45	40	
8	C	Normal[1,0.1]	15	15	15	15	15	
9	N	Normal[1,0.1]	20	25	30	20	25	
10	Y	Normal[1,0.1]	20	20	20	20	20	
11	D	Normal[1,0.1]	45	35	40	40	30	
12								
13								
14								
15								
16								
17								
18								

Figure 8.27 Quantity Input for Examples 8.3 and 8.4

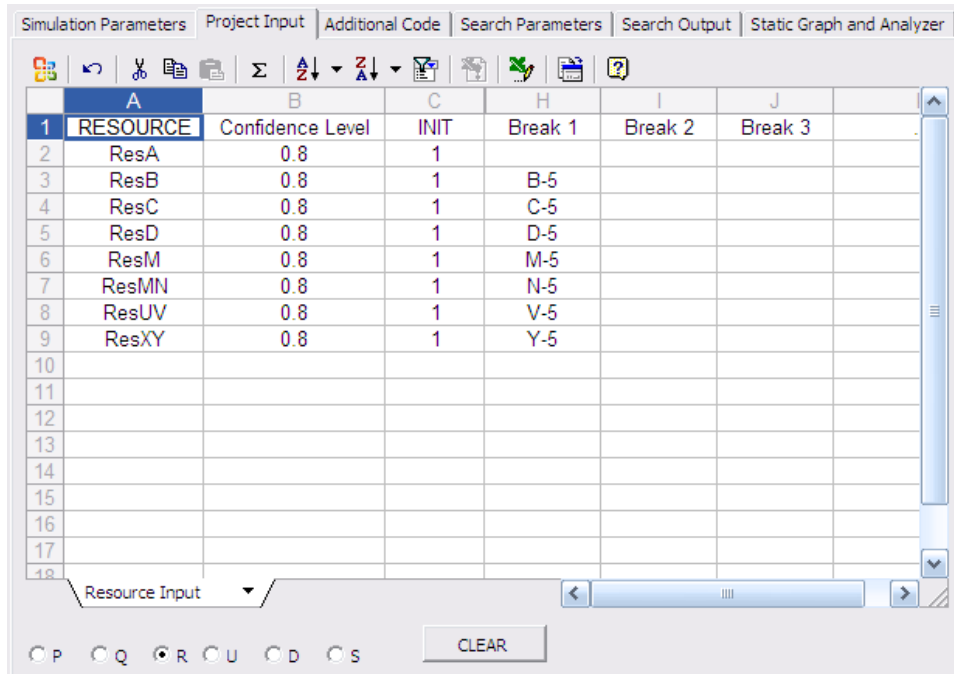


Figure 8.28 Resource Input for Example 8.3 only

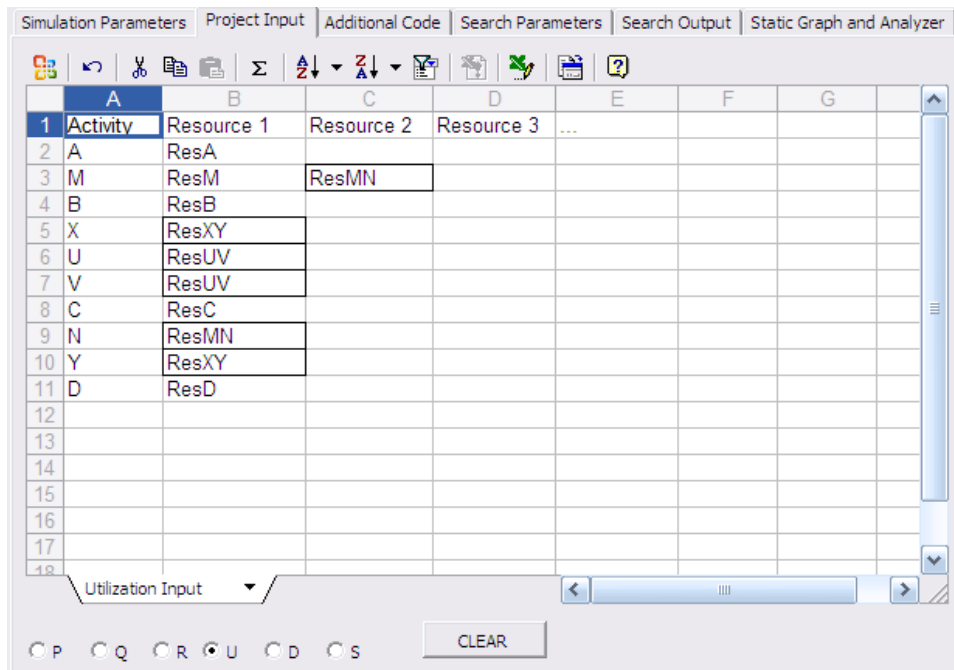


Figure 8.29 Utilization Input for Examples 8.3 and 8.4

After entering Simulation Input and Project Input, the entered inputs are ready for simulation model and code generation. To create the simulation, users click the

“GENERATE” button. To execute the simulation, users click the “RUN MODEL” button.

It is important to realize that the inputs for Example 8.3 are not effective because idle time still remains after solving the problem using SQS-AL with the given inputs. The main three reasons are:

- 1) Working sequences of resources are not consistent from the processing SQS which their CITs are collected, to later SQS which their corresponding CLTs are employed. Accordingly, the collected CITs do not provide an effective CLT that can eliminate idle time. The problem of inconsistency in working sequences and solutions are presented in Chapter 7. Note that this problem could exist in repetitive projects with or without resource-sharing activities.
- 2) Idle time in shared resources cannot be completely eliminated due to a slow production rate of a dependent activity (e.g., Activity U in Figure 8.24) between two indirectly dependent resource-sharing activities (e.g., Activities M and N in Figure 8.24). The problem of insistence of idle time due to a slow production rate of an activity between two resource-sharing activities is discussed in Chapter 7 along with examples and solutions. This problem only exists in repetitive projects with indirectly dependent resource-sharing activities having a slow-production-rate dependent activity between the resource-sharing activities.
- 3) Crew lead times (CLT) of Resources ResM and ResMN, serving Activities M are determined in different processing sequence steps. Idle time will be incurred in the resource that arrives to the site first (ResM in this case),

because ResM and ResM must be available in order to perform resource-sharing Activity M. By default, ChaStrobe collect CITs and determines CLT for shared resources in the latest SQS in which their activities belong. Accordingly, ChaStrobe determines CLT for ResM in SQS2, in which Activity M belongs, while CLT for ResMN is determined in SQS4, in which Activity N belongs. Therefore, Activity M cannot start until ResMN arrives; as a result, ResM has to wait for ResMN.

To solve Example 8.3 effectively, the inputs in Example 8.3 must be modified as explained later in Example 8.4. However, the ineffective inputs for Example 8.3 are shown for the purposes of comparison between two schedules derived from different inputs in Examples 8.3 and 8.4. The production diagram from the 1st replication in the SQS6 is shown in Figure 8.30.

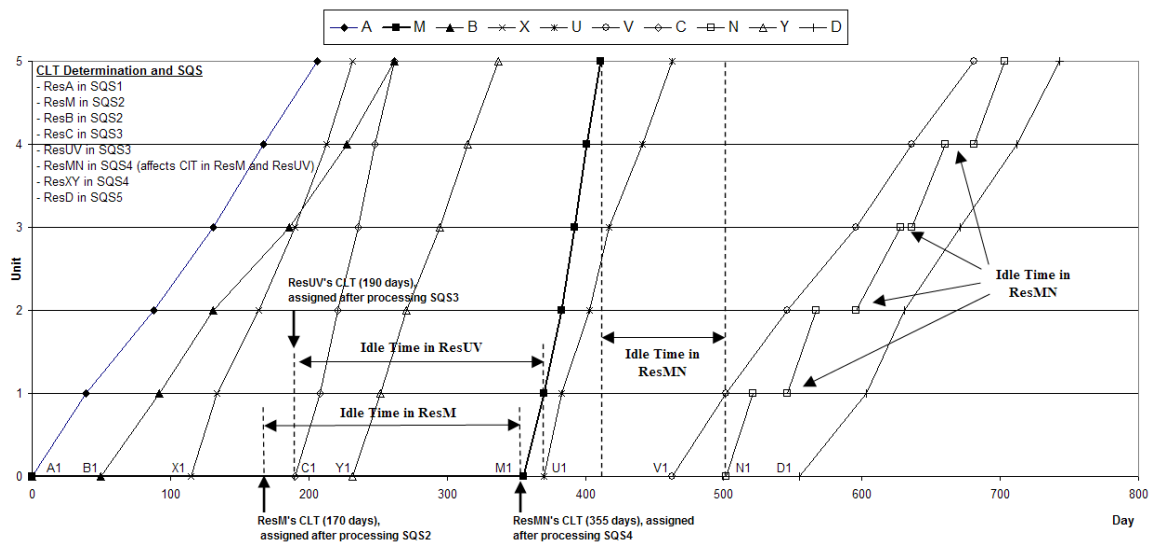


Figure 8.30 Production diagram from the 1st replication in SQS6 for Example 8.3

In Figure 8.30, descriptions are given to point out problems of using SQS-AL to schedule repetitive projects with resource-sharing activities. As discussed in detail in

Chapter 7 and earlier, the difference in processing sequence steps for ResM (i.e., SQS2) and ResMN (i.e., SQS4) causes the idle time in ResM. As shown in Figure 8.30, CLT for ResM is determined after processing SQS2, assigning CLT_M of 170 days, while CLT for ResMN is still undetermined. After processing SQS4, SQS-AL determines CLT_{MN} of 355 days, causing a further delay in the start date of Activity M from CLT_M to CLT_{MN} due to the availability of ResMN. As a result, ResM arrives to the site on its CLT_M (170 days), and must wait until ResMN arrives on CLT_{MN} (355 days), adversely resulting in 185 days of idle time in ResM.

Similar to the idle time in ResM, the idle time in ResUV is caused by the fact that CLT for ResUV and ResMN are determined after processing two different sequence steps. CLT_{UV} is determined after processing SQS2, and is set to 190 days. However, after processing SQS4, SQS-AL determines CLT_{MN} of 355 days, causing a further delay in Activity U due to the precedence constraints between Activities M and U. As a result, ResUV arrives to the site on its CLT_{UV} (190 days), and must wait until ResMN arrives on CLT_{MN} (355 days), causing 165 days of unnecessary idle time in ResUV.

It is important to point out that the idle times in ResM and ResUV are caused by the impact of processing sequence steps for resource-sharing activities and their dependents on the effectiveness of SQS-AL. This impact changes the working sequence of resources and activity start dates from which CLTs are determined. Changing in working sequence and activity start dates incur change in the idle time; as a result, the collected CITs prior to the change are not a good representative for the “new” idle time after the change. Therefore, the CLTs determined earlier cannot effectively eliminate the new idle time.

Another complication of scheduling resource-sharing activities is exhibited in Figure 8.30 between indirectly dependent resource-sharing Activities M and N, and their dependent activities U and V. It is important to realize that Activity U, having a slow production rate, causes 1) the lags between Activities M and N and 2) idle time in ResMN, as shown in Figure 8.30. This idle time between M5 and N1 is inevitable. Delaying ResMN's arrival date further will result in the same amount of idle time in ResMN, since Activities U and V are direct successors of Activity M and direct predecessors of Activity N. Moreover, the slow production rate of Activity U also causes idle time in ResMN serving Activity N.

Figure 8.31 presents the average project duration versus idle time from processing each sequence step (SQS1 to SQS6) for Example 8.3. Figure 8.31 shows an unusual up-and-down average project idle time, which indicates difficulty in solving Example 8.3. In a typical case, the average project idle time should decrease as SQS-AL proceeds from one sequence step to the next. However, Figure 8.31 shows the average project idle time increases after processing SQS2 (assigning CLT_B and CLT_M) and processing SQS4 (assigning CLT_{MN} and CLT_{XY}). The reasons for the increase in the idle time are the complication of scheduling resource-sharing activities using SQS-AL. To solve this complication, the inputs in Example 8.3 must be modified as presented later in Example 8.4.

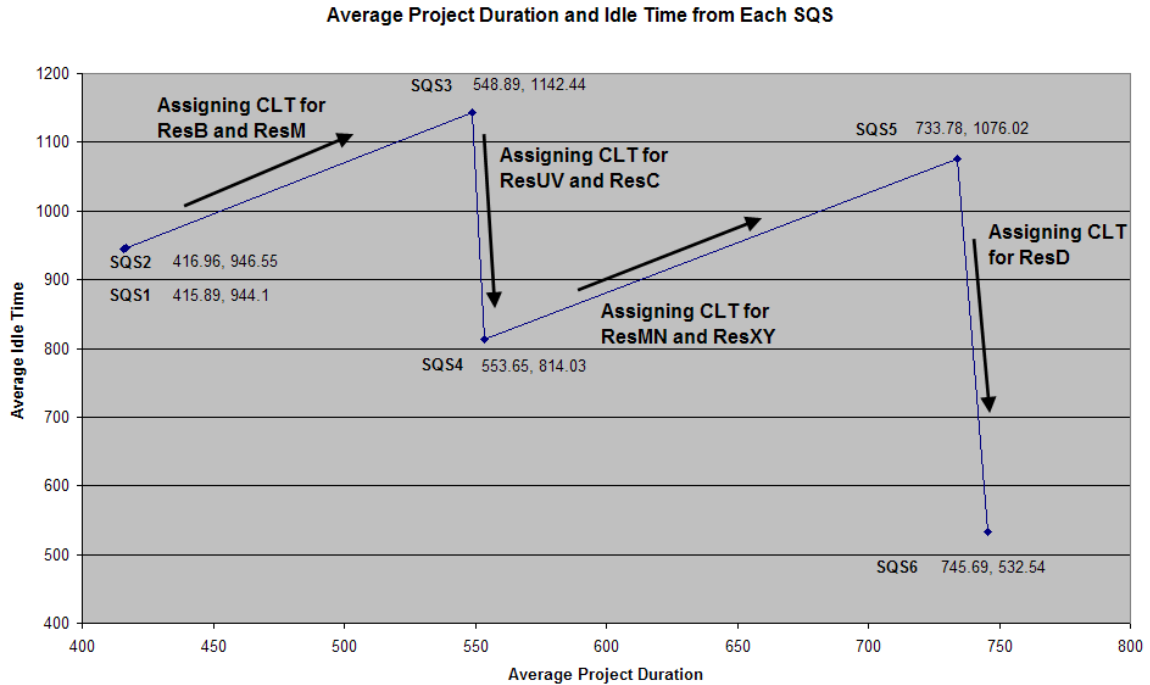


Figure 8.31 An unusual up-and-down pattern of average project idle time in scheduling resource-sharing activities

8.2.4 Example 8.4 Repetitive project with resource-sharing activities and work breaks

Example 8.4 is the same problem as Example 8.3. However, several modifications are made so that SQS-AL can effectively solve the problem. To alleviate the difficulties of scheduling the resource-sharing activities in Example 8.3, the following additional conditions are included in the schedule and simulation model for Example 8.4:

- A work break for Resource ResMN is scheduled between M5 and N1, meaning ResMN will work continuously from M1 to M5 (Cell H7 in Figure 8.32), and then take a break. Next, ResMN will return to the site, and work continuously from N1 to N5 (Cell I7). Figure 8.32 presents the modified inputs on the Resource Input sheet, which is different from Example 8.3. A

work break is scheduled right after M5, as shown in the solid border cells in Figure 8.32.

- The CIT1 for ResM and ResMN (before the break at M5-N1) are specified to be collected from processing the same SQS; CIT1_M and CIT1_{MN} are collected during processing SQS2, while CIT2_{MN} (the crew idle time for Resource MN after the break) is collected during processing SQS4. Figure 8.31 shows the simulation code in Model Parameters specifying the processing SQS for CIT1_M and CIT1_{MN}.
- The consistency in ResMN's working sequence from processing one SQS to another is guaranteed. An additional semaphore for ResMN stipulating its working sequence is included, which is N_Perform_Additional_Semaphore shown in Figure 8.34. This additional semaphore prevents ResMN from serving Activity N, unless five units of Activity M have been completed. This Activity N's additional semaphore is encoded in ChaStrobe's Additional Code for Programming Objects.
- The consistency in ResXY's work orders from one processing SQS to another is guaranteed. An additional semaphore for ResXY, stipulating ResXY's working sequence, is included which is Y_Perform_Additional_Semaphore. This additional semaphore, shown in Figure 8.34, prevents ResXY from serving Activity Y, unless five units of Activity X have been completed. This Activity Y's additional semaphore is encoded in ChaStrobe's Additional Code for Programming Objects.

Note that Precedence Input, Quantity Input, and Utilization Input for Example 8.4 are the same as in Example 8.3. Hence, for the inputs in Example 8.4, see Figure 8.26 for Precedence Input, Figure 8.27 for Quantity Input and Figure 8.29 for Utilization Input.

	A	B	C	H	I	J	K
1	RESOURCE	Confidence Level	INIT	Break 1	Break 2	Break 3	...
2	ResA	0.8	1				
3	ResB	0.8	1	B-5			
4	ResC	0.8	1	C-5			
5	ResD	0.8	1	D-5			
6	ResM	0.8	1	M-5			
7	ResMN	0.8	1	M-5	N-5		
8	ResUV	0.8	1	V-5			
9	ResXY	0.8	1	Y-5			
10							
11							
12							
13							
14							
15							
16							
17							
18							

Resource Input

P Q R U D S

CLEAR

Figure 8.32 Resource Input for Example 8.4, different from Example 8.3

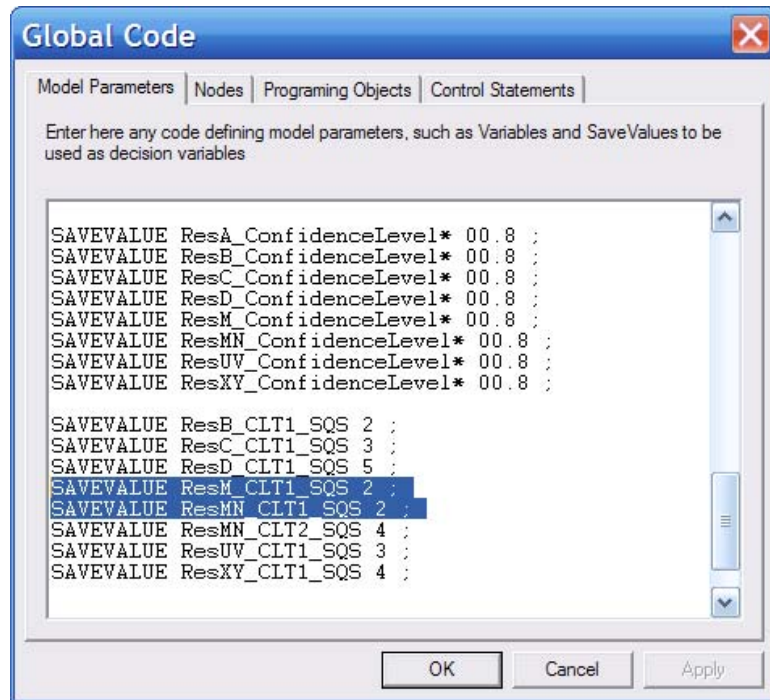


Figure 8.33 CIT1_M and CIT1_{MN} (before work break at M5-N1) collected from the same processing SQS2



Figure 8.34 Additional code stipulating ResMN's working sequence from M1 to M5 and then N1 to N5, and ResXY's working sequence from X1 to X5 and Y1 to Y5

After modifying the input, the simulation model is ready to be created. To create the simulation model, users click the “GENERATE” button. To execute the simulation, users click the “RUN” button.

Figure 8.35 presents the average project duration and idle time from processing each SQS. As shown in the figure, project idle time decreases as SQS-AL proceeds from one SQS to another, which is a typical pattern of decreasing idle time that should be achieved from SQS-AL, not the up-and-down pattern shown in Figure 8.31. Therefore, the additional conditions included in the input for Example 8.4 resolve the ineffectiveness of the input in Example 8.3. Figure 8.36 is the production diagram from the first replication in SQS6. As shown in the figure, all activities have no interruption between units at least in this replication, excepting for Activity C between Units 4 and 5.

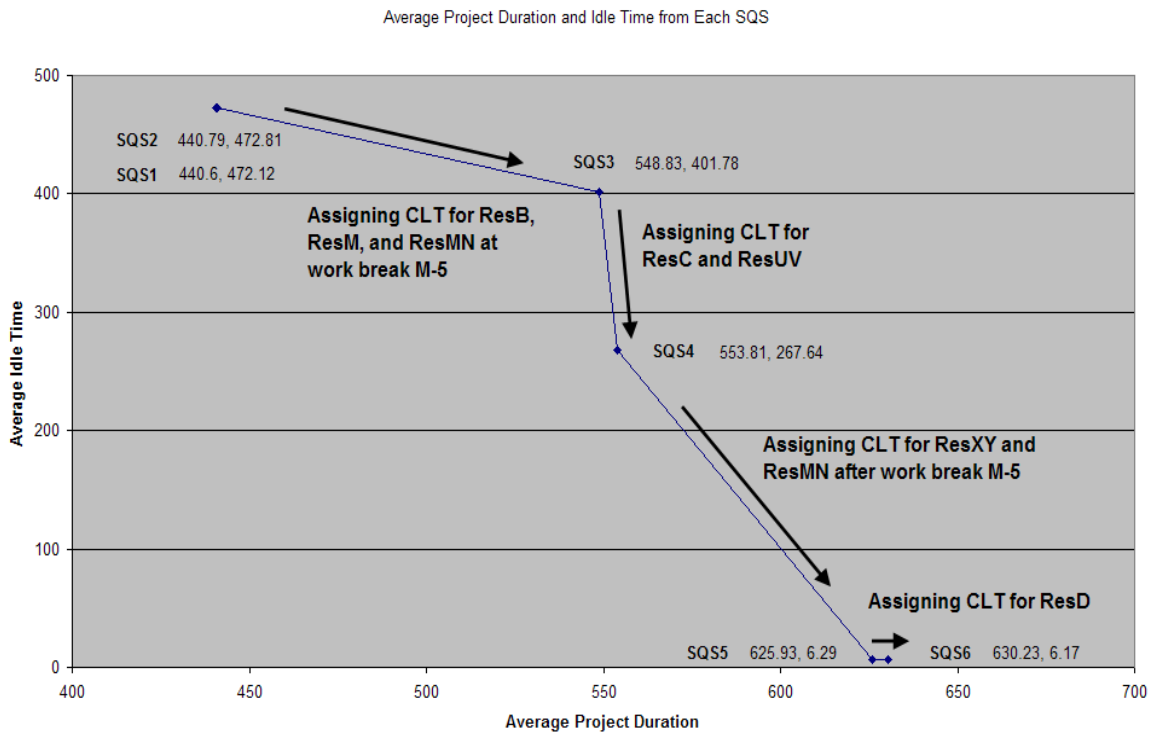


Figure 8.35 A typical pattern of decreasing average project idle time in scheduling repetitive projects using SQS-AL

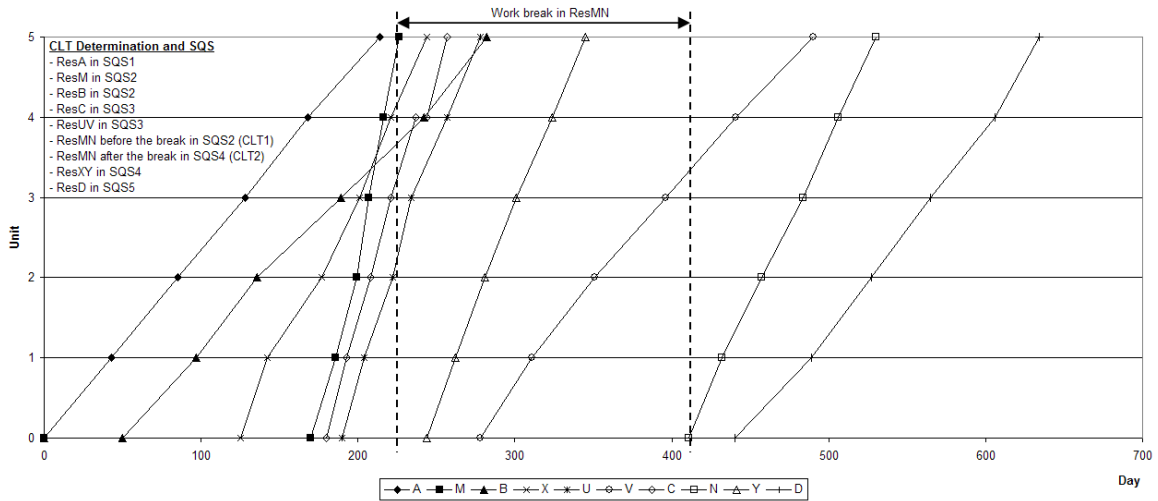


Figure 8.36 Production diagram from the 1st replication in SQS6 for Example 8.4

8.3 ChaStrobe's Output

After simulation execution, ChaStrobe offers four output features used to create a graphical presentation and to analyze the output from the simulation. As shown in Figure 8.37, the four features are:

- 1) Creating graphs for project duration and its probabilities from processing each SQS
- 2) Creating static graphs from processing each SQS
- 3) Analyzing the output by comparing the results from SQS-AL to CPM and RSM, regarding project duration and idle time
- 4) Creating a schedule in MS Project using the analyzed data from SQS-AL schedule in (3)

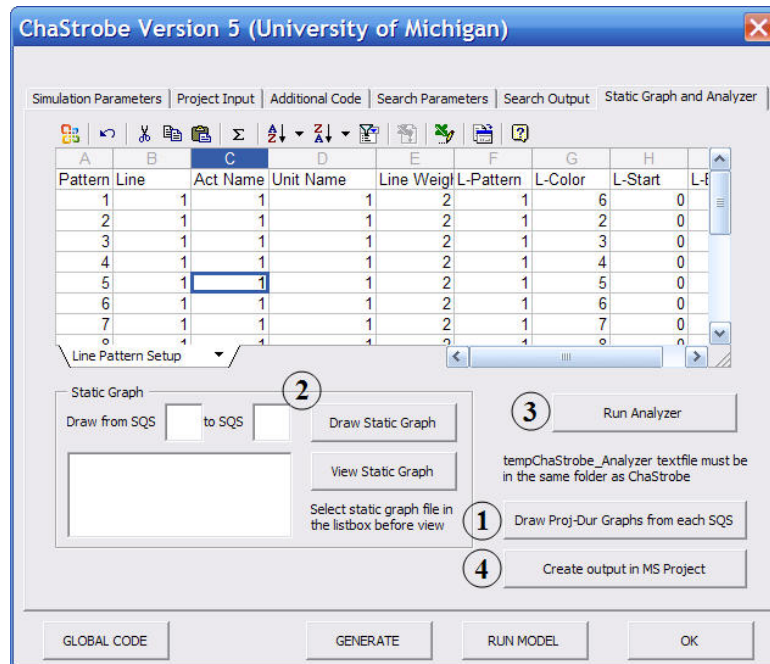


Figure 8.37 ChaStrobe’s four output features

8.3.1 Project Duration Graphs for each Processing SQS

At the end of simulation execution, ChaStrobe stores the simulation outputs for 1) the probability of project duration from processing each SQS in a text file titled tempChaStrobe_StaticGraphProb.txt and 2) project durations and project idle times in a text file titled tempChaStrobe_StaticGraphSQS.txt. To create a graphical presentation from the text files, users click the “Draw Proj-Dur Graphs from each SQS” button, and then ChaStrobe will generate three graphs, related to project duration and idle time from processing each SQS in an automatically-created Excel file, called Analyzer.xls. The three graphs, shown in Figures 8.38 to 8.40, are generated automatically from the stored data in tempChaStrobe_StaticGraphProb.txt and tempChaStrobe_StaticGraphSQS.txt.

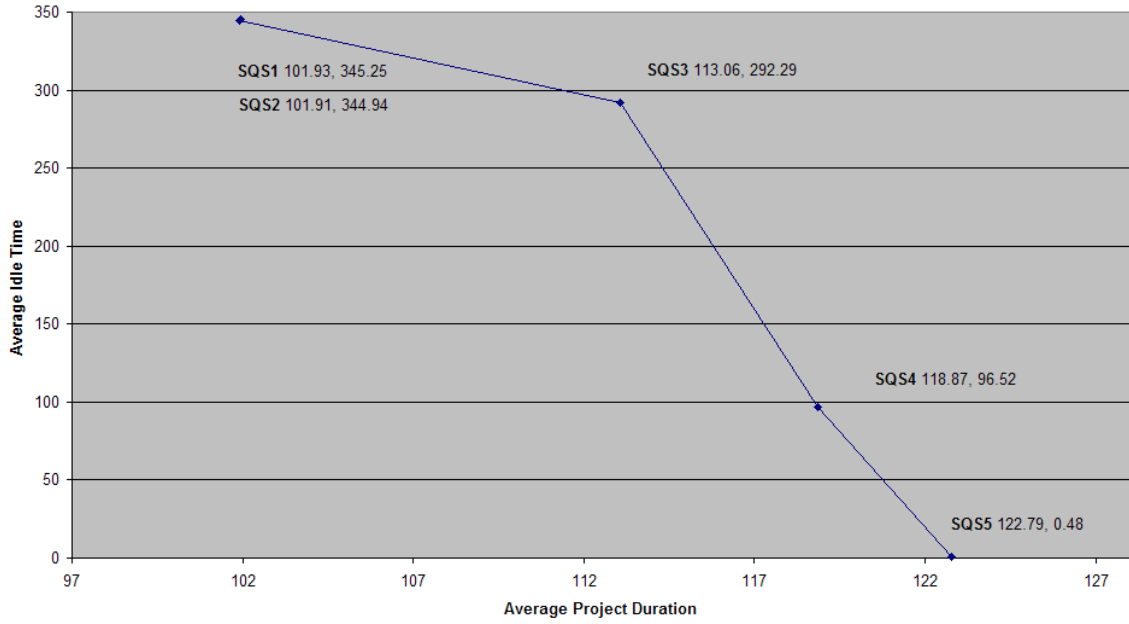


Figure 8.38 The average project duration and idle time from processing each SQS for Example 8.1

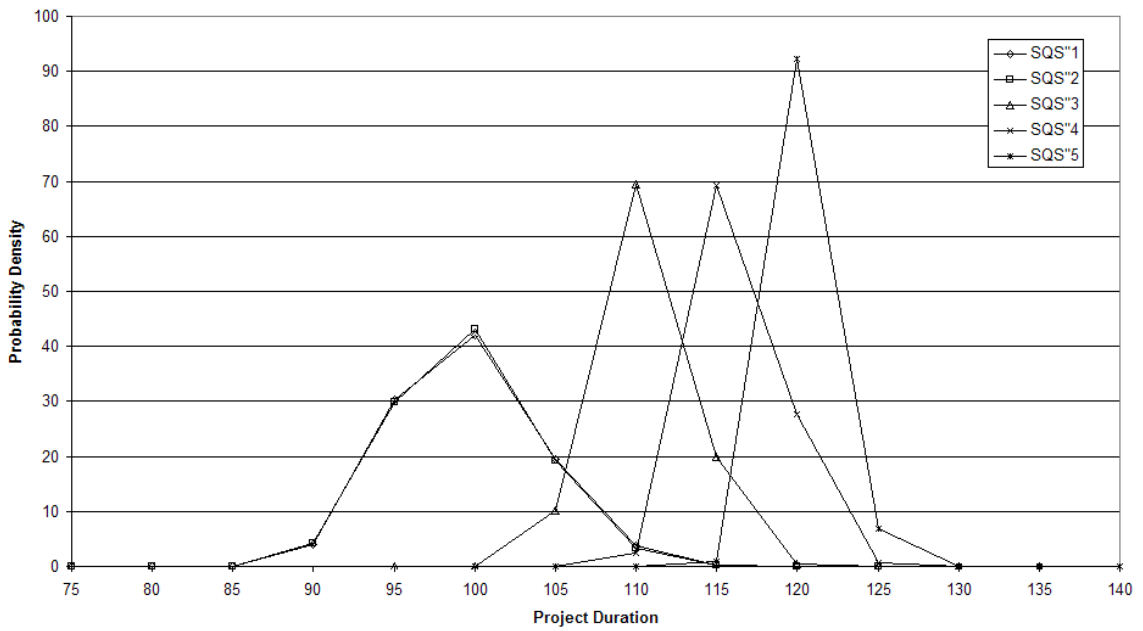


Figure 8.39 Probability density functions of project duration from processing each SQS for Example 8.1

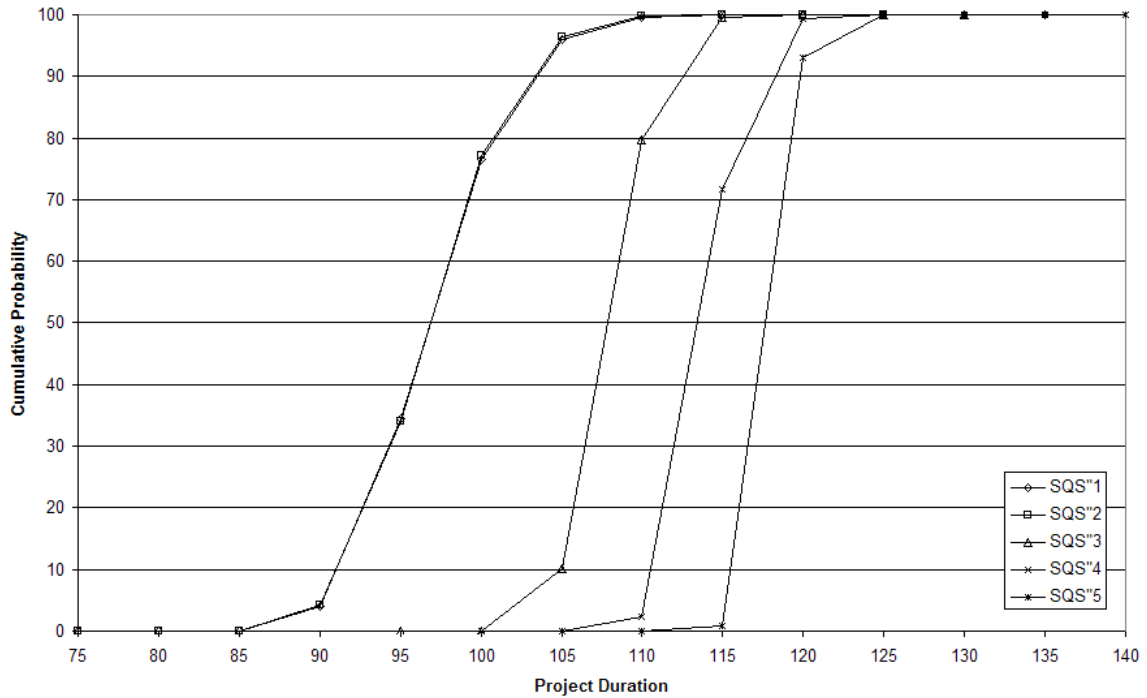


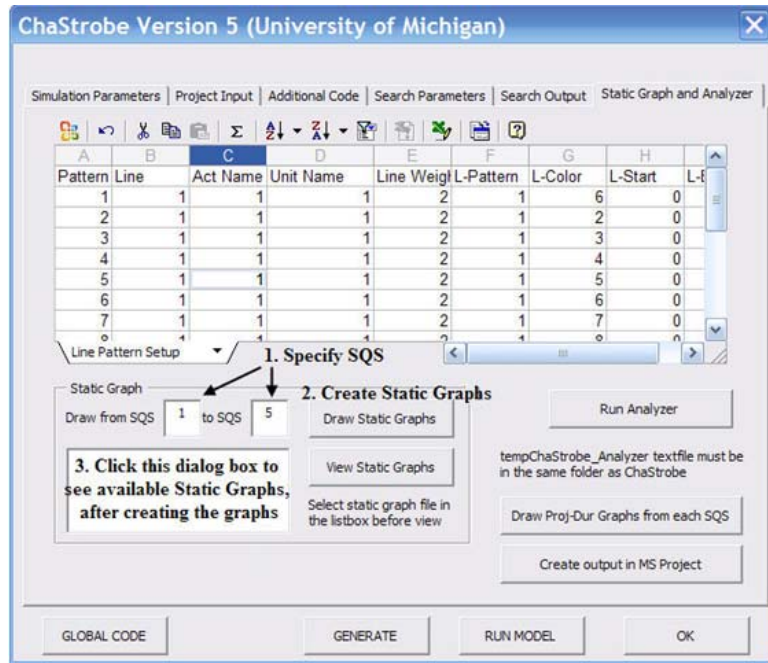
Figure 8.40 Cumulative distribution functions of project duration from processing each SQS for Example 8.1

8.3.2 Static Graphs

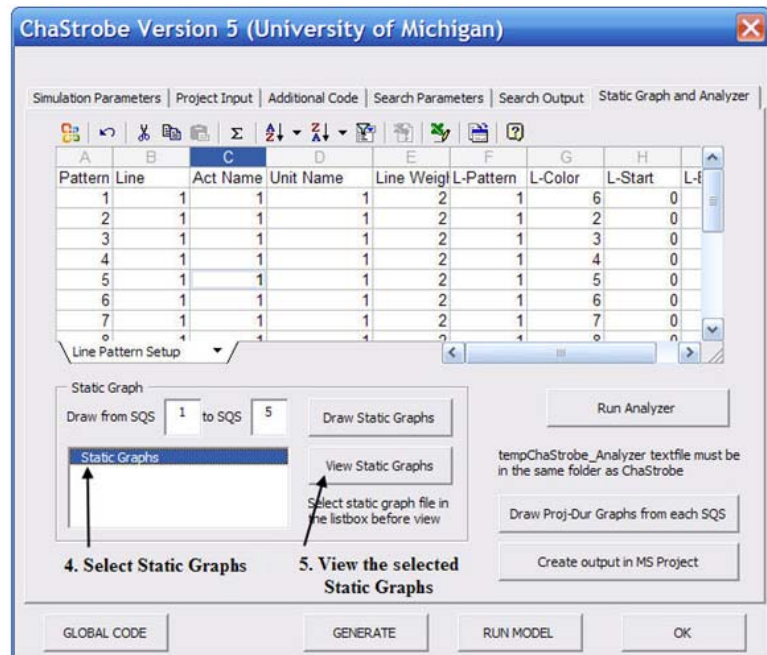
During a simulation run in the first replication of each SQS, ChaStrobe stores the data of activities' start dates and durations in tempChaStrobe_StaticGraph.txt. This data is used for creating Static graphs, which are the production diagrams of the 1st replication from processing each SQS. To create the Static graphs, users first specify the data from which processing SQS will be used, as shown in Figure 8.41.a. Then, users click the “Draw Static Graphs” button. Next, ChaStrobe will request a filename for the Static Graphs in the Visio file from the users.

From the Static Graphs input in Figure 8.41.a, ChaStrobe creates the production diagrams of the 1st replication from SQS1 to SQS5. To view the available created production diagram files, users click on the empty dialog box next to the “View Static

Graphs” button, and then ChaStrobe will show the available files in ChaStrobe’s Static Graphs, as shown in Figure 8.41.b.



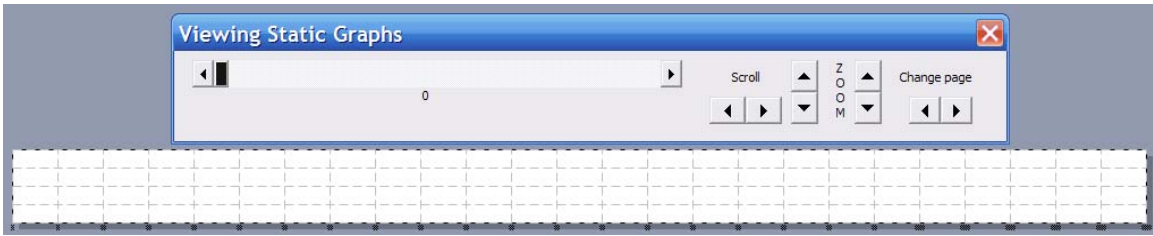
(a) Creating Static Graphs by clicking the “Draw Static Graphs” button



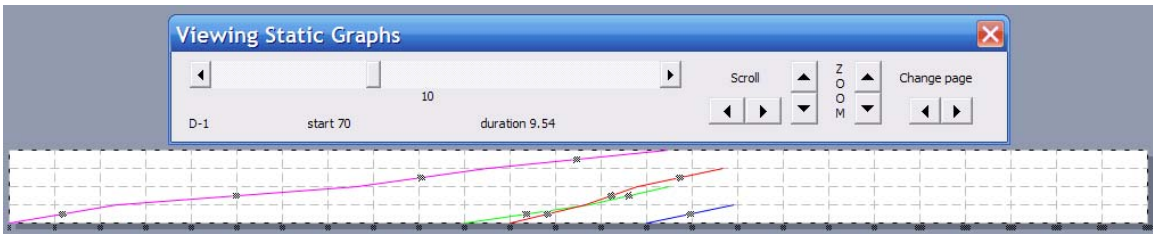
(b) Selecting and viewing Static Graphs

Figure 8.41 Creating and viewing Static Graphs

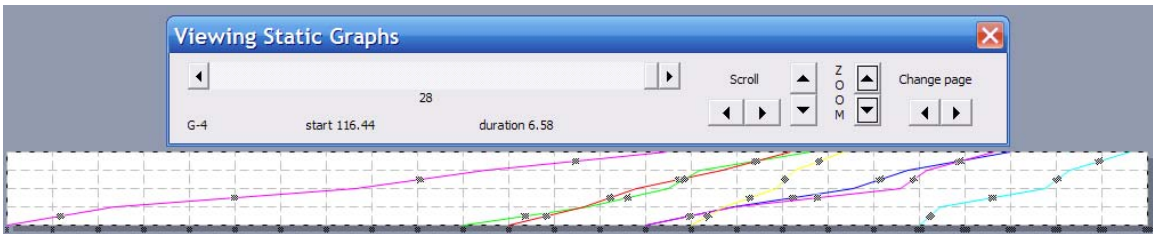
After a file for Static Graphs is selected, as shown in Figure 8.41.b, users click the “View Static Graphs” button to view the selected file. Figures 8.42.a to 8.42.c show three production diagrams (different in the timeframe selected by the users) created from the last SQS. Users use the “Viewing Static Graphs” form to manipulate the timeframe as shown in Figure 8.42. The timeframe for the production diagram in Figure 8.42.a is at the beginning of the project (Time = 0). The timeframe in Figure 8.42.b is at the beginning of the 10th activity (D1). The timeframe in Figure 8.42.c is at the beginning of the 28th activity (G4). Users can use Static Graphs to study the work orders of repetitive activities in the project, and also compare the changes in the activities’ working sequences from one processing SQS to another. By clicking the “Change Page” button, ChaStrobe will show a different production diagram from a different processing SQS.



(a) Production diagram at the beginning of the project



(b) Production diagram at the beginning of the 10th activity (D1)



(c) Production diagram at the beginning of the 28th activity (G4)

Figure 8.42 Static graph from the 1st replication of processing SQS5 for Example 8.1

8.3.3 ChaStrobe's Analyzer

While processing the last sequence step, ChaStrobe stores activities' start dates and durations from each replication in tempChaStrobe_Analyzer.txt. This data is then used by ChaStrobe's Analyzer to analyze and compare the results from SQS-AL, CPM, and RSM, regarding project duration and idle time. To activate ChaStrobe's Analyzer, users click the "Run Analyzer" button as shown in Figure 8.38. After ChaStrobe analyzes the data in tempChaStrobe_Analyzer.txt, the following graphical presentations are created:

- 1) Cumulative Distributions of Project Duration, shown in Figure 8.43
- 2) Probability Density Functions of Project Duration, shown in Figure 8.44
- 3) Comparing Project Duration of RSM and SQS-AL to CPM, shown in Figure 8.45
- 4) Comparing Project Duration of SQS-AL to RSM, shown in Figure 8.46
- 5) Difference in Project Duration and Idle Time between RSM and CPM, between SQS-AL and CPM, and between SQS-AL and RSM, shown in Figure 8.47

Note that the discussion for cumulative distributions (Figure 8.43) and probability density function (Figure 8.44) of project duration derived from CPM, RSM, and SQS-AL is given in Chapter 4.

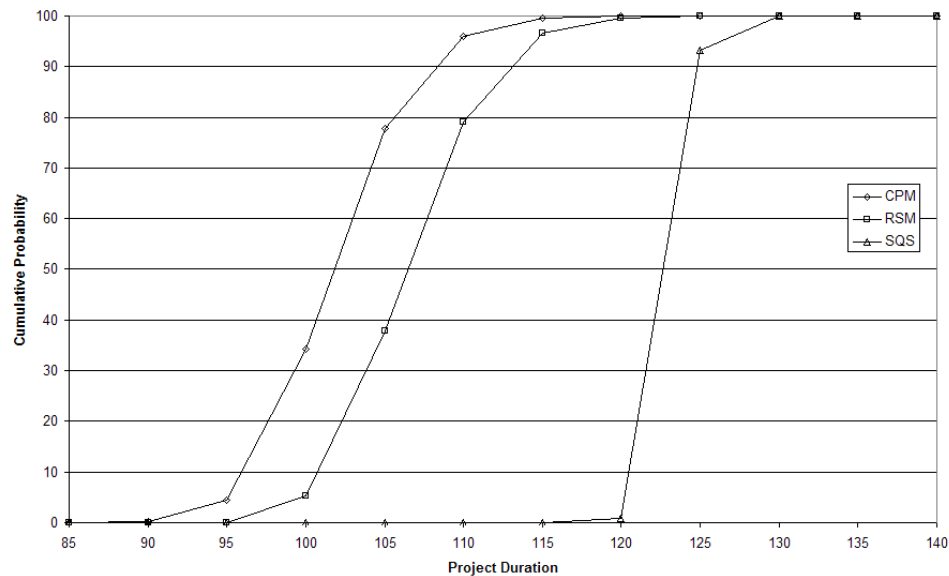


Figure 8.43 Cumulative distributions of project duration derived from CPM, RSM, and SQS-AL

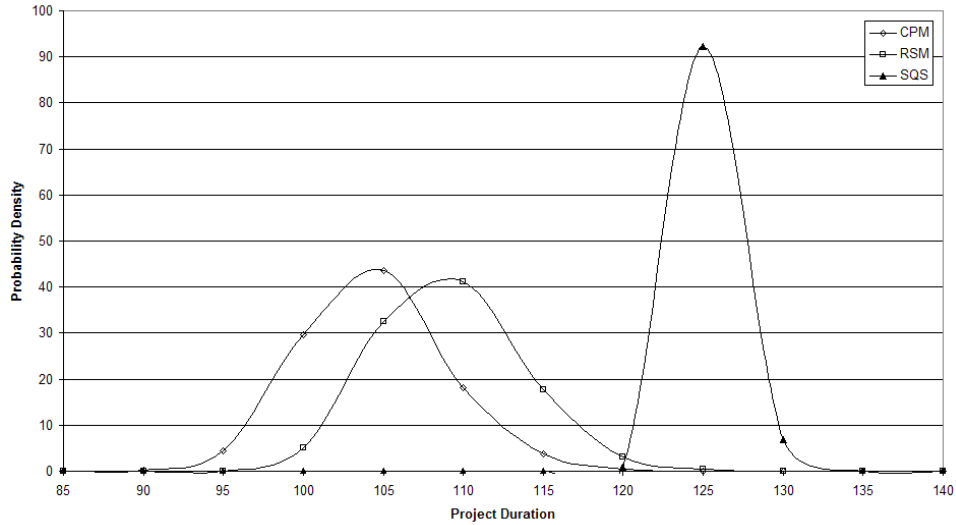


Figure 8.44 Probability density functions of project duration derived from CPM, RSM, and SQS-AL

Note that the explanations (in texts and shapes) in Figures 8.45 to 4.46 are added into the figures in order to discuss about the application of CPM, RSM, and SQS-AL; they are not automatically created by ChaStrobe’s Analyzer. The discussions provide insight information and how to interpret the corresponding figure.

Figure 8.45 shows the duration derived from CPM on the X axis and that from RSM and SQS-AL on the Y axis. It is important to remember that schedules and their project durations derived from RSM are based on perfect hindsight, i.e., it is assumed that activity durations are known before scheduling the project. With such perfect hindsight, project duration from RSM changes according to the sampled activity durations and the resulting necessary delays in activity start dates to achieve continuity. When activity durations and the idle time (existing in the CPM schedule for the same replication) happen to be small, project duration from RSM also turns out small; when the activity durations and the idle time happen to be large, project duration from RSM also turns out large. According to the assumption of perfect hindsight and variability in this example

project, project durations from RSM ranges from 97 to 117 days, approximately, as shown in Figure 8.45.

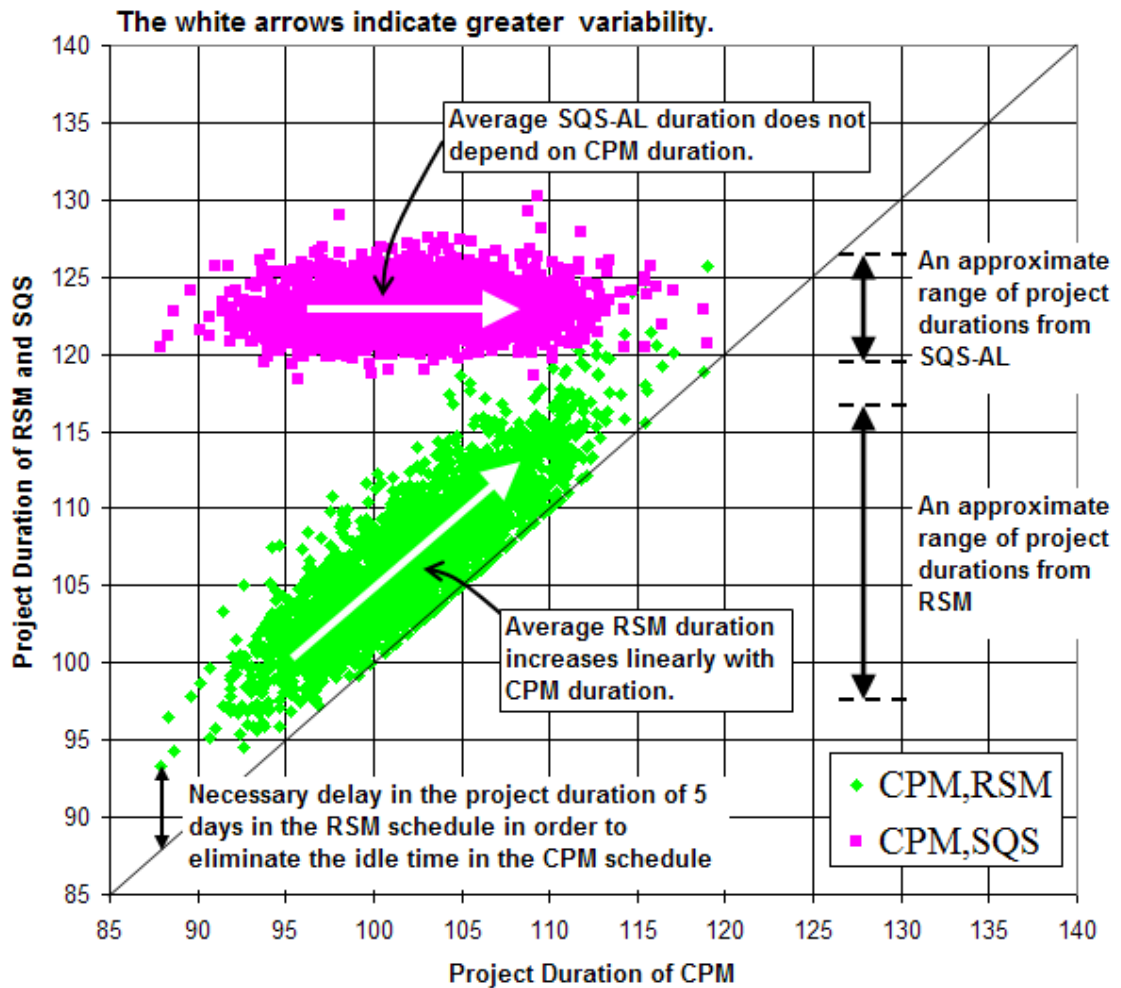


Figure 8.45 Comparing project duration of RSM and SQS-AL to CPM

It is important to notice that all “CPM,RSM” points are above or on the diagonal line in the figure. This is as expected because CPM produces an early start schedule which is guaranteed to be the shortest. RSM, on the other hand, delays the start date of some activities to achieve continuity. Hence, RSM produce greater project duration than does CPM, in most cases. In some cases, it happens by chance that CPM and RSM produce the same project duration as shown by “CPM,RSM” points on the diagonal line;

however, this does not mean they result in the exactly same schedule. It is possible that schedules from CPM on a “CPM,RSM” point on the diagonal line result idle time, while schedules from RSM on the same point do not. The same project duration from CPM and RSM does not necessarily mean they result in neither the same schedule nor the same amount of idle time.

In general, the vertical distance between “CPM,RSM” points the diagonal line in Figure 8.45 is the necessary delay in project completion determined by RSM in order to eliminate idle time with the shortest project duration. It is apparent that the delays caused by resource continuity in RSM follow the same distribution for any value of CPM duration.

As opposed to RSM, SQS-AL schedules projects without perfect hindsight. Thus, it includes floats (time buffers) into the schedule by using crew lead times (CLTs) in order to eliminate idle time with a certain degree of confidence (i.e., a given confidence level). CLT for a resource is a fixed arrival date of that resource (or its activity’s start date) which is determined without knowing activity durations in advance. With an 80% confidence level, used for this example in Figure 8.45, SQS-AL postpones activities from their early start dates and results in project durations ranging of 119 to 127 days, approximately. The project duration from SQS-AL do not vary as a function of CPM duration due to the chosen CLTs. The project duration from SQS-AL fits into a smaller range than that of RSM because SQS-AL with the selected 80% confidence level delays activity start dates (creating floats) to prevent idle time 80% of the time. Therefore, the already-delayed project completion from SQS-AL schedule is not affected as much by activity durations or idle time existing in CPM as the RSM schedule. The vertical

distance between “CPM,RSM” points and the diagonal line is the remaining part of CLTs (unused buffers) that cannot be removed. As variability increases, the greater activity durations and idle time in the CPM schedule, the greater amount of the chosen CLTs is used, the smaller the vertical distance between “CPM,SQS” points and the diagonal line is.

Figure 8.46 shows the comparison of project duration derived from RSM and SQS-AL. The vertical distance between “RSM,SQS” points and the diagonal line is caused by the fact that RSM delays activities knowing the exact activity durations for individual schedule (each point in the figure), whereas SQS-AL has to choose one set of CLT values to be used for all schedules (all points in the figure). In other words, the vertical distance is a part of the float included into the schedule by SQS-AL in order to prevent idle time for a certain degree of confidence, which may be larger than necessary. This vertical distance can also be interpreted as an unnecessary delay (waste in time) in the project duration that is included because of variability. Without variability in project durations, RSM and SQS-AL will result in the same project duration.

It is interesting to note that the difference between the vertical distance between “CPM,SQS” points and the diagonal line in Figure 8.45 and the vertical distance between “RSM,SQS” points in Figure 8.46 is that the vertical distance for “CPM,SQS” includes both necessary and unnecessary delay in the project duration, while the vertical distance for “RSM,SQS” only presents the unnecessary delay.

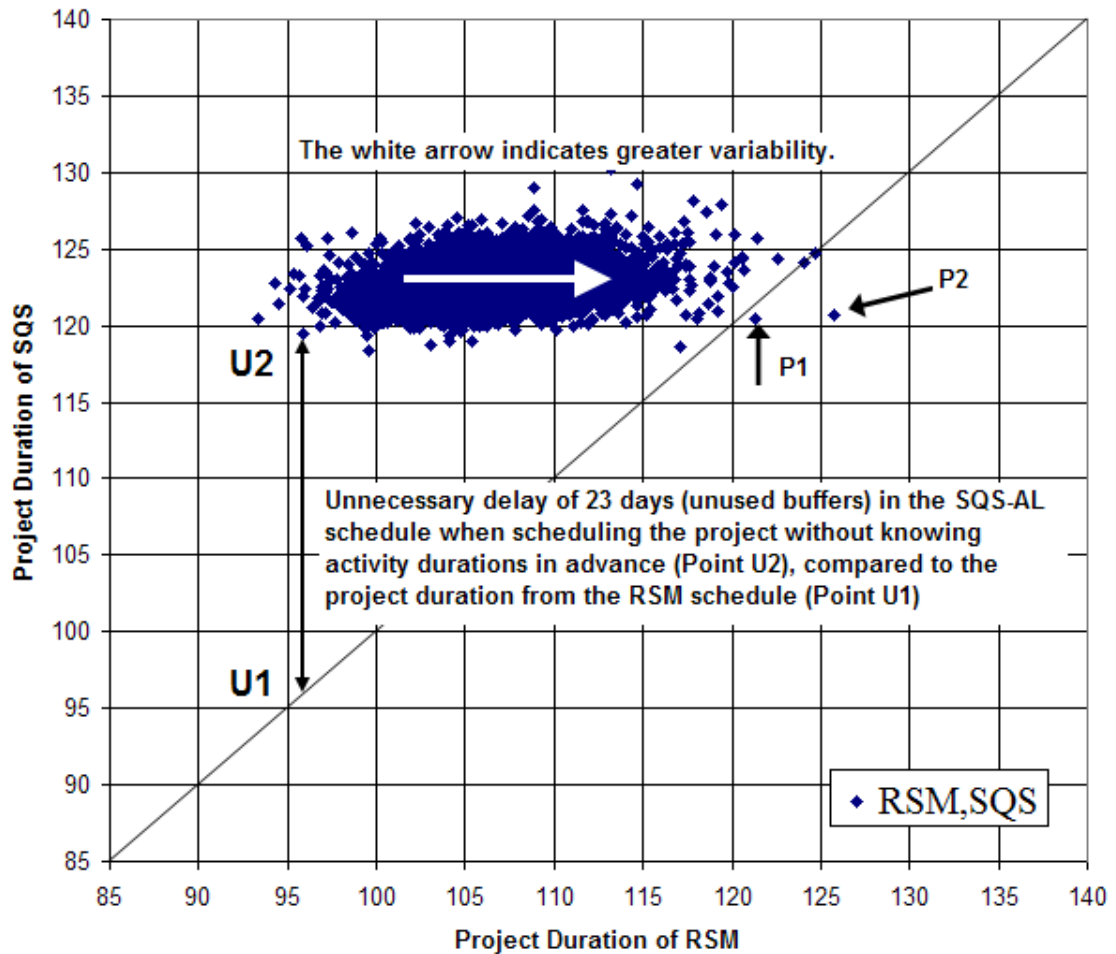


Figure 8.46 Comparing project duration of SQS-AL to RSM

The data points for “RSM,SQS” (e.g., P1 and P2 in Figure 8.46) below the diagonal line indicate that the chosen CLTs are not large enough for these schedules. As a result, it is certain that there is idle time in these schedules. Remember, the project duration derived from RSM is the minimum project duration without idle time. Therefore, it is impossible to have a shorter project duration than that of RSM without idle time. Note that the opposite of this claim is not always true. It is possible that idle time exists in the SQS-AL schedule even though its project duration is greater than that of the RSM schedule.

Figure 8.47 shows the difference in project duration on the X axis and the difference in idle time on the Y axis. This figure provides insightful comparison among CPM, RSM, and SQS-AL in terms of 1) project duration, 2) idle time, and 3) the relationships between project duration and idle time.

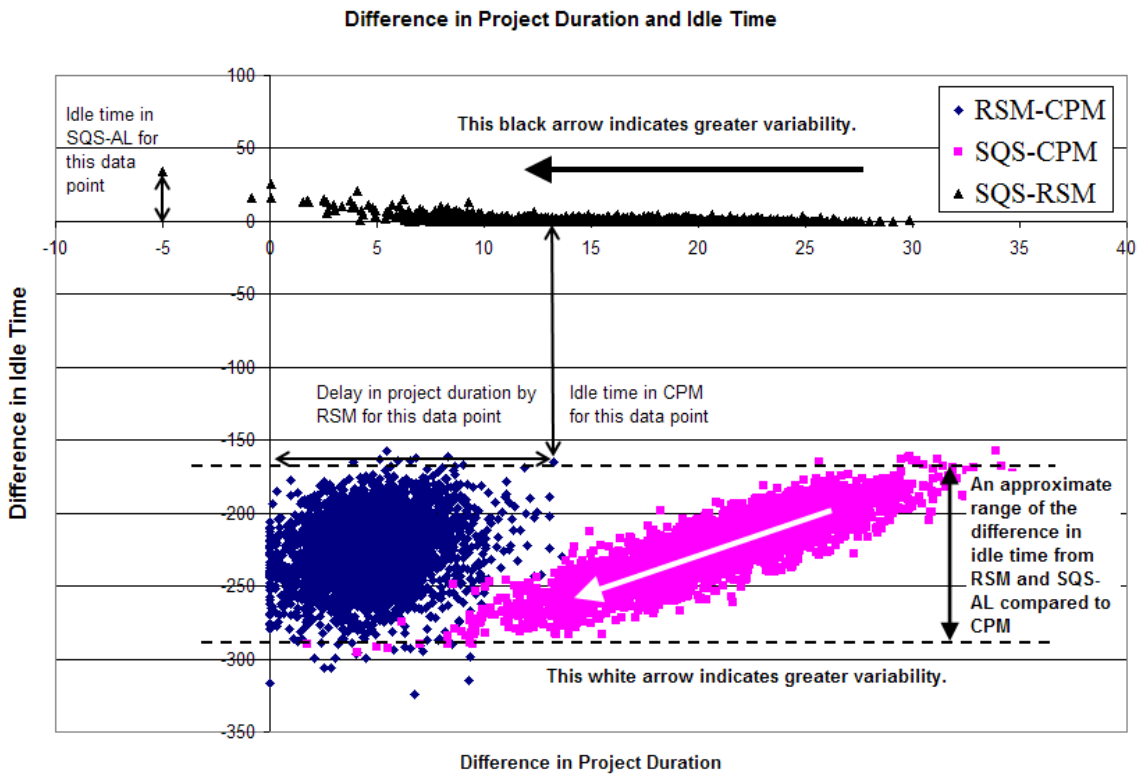


Figure 8.47 Difference in project duration and idle time between RSM and CPM, SQS-AL and CPM, and SQS-AL and RSM

The data points for “RSM-CPM” in Figure 8.47 shows that this example project requires necessary delay in project duration ranging from 0 to 10 days in order to eliminate the idle time in the CPM schedule ranging from 175 to 285 days, approximately. As shown clearly by the scatter of the data points, the relationship between 1) the difference in project duration and 2) the difference in idle time from CPM and RSM do not exhibit any linear dependence (covariance).

Regarding to project duration, the project duration derived from CPM is the shortest project duration (without eliminating idle time) since CPM schedules activities to start as early as possible. Hence, it is impossible to have data points of “RSM-CPM” or “SQS-CPM” on the negative side of the X axis. Similarly, project duration derived from RSM is the minimum project duration without idle time. Since there is no idle time in the RSM schedule, the difference in idle time between RSM and CPM, “RSM-CPM,” is the idle time in the CPM schedule. Accordingly, the difference in project duration between RSM and CPM is the necessary increase in project duration in order to eliminate all idle time (which is the difference in idle time between RSM and CPM).

The data points for “SQS-CPM” in Figure 8.47 show that as variability increases, the performance of SQS-AL schedules improve relatively to that of CPM schedules. As clearly shown in the figure by the white arrow, there is a linear dependency (covariance) between the difference in project duration and the difference in idle time from CPM and SQS-AL. Scheduling this example project with SQS-AL causes increase in project duration ranging from 10 to 30 days (also shown in the vertical distance between “CPM,SQS” points and the diagonal line in Figure 8.45) in order to eliminate the idle time in the CPM schedule ranging from 175 to 285 days.

The data points for “SQS-RSM” shows that as variability increases, the difference in project durations derived from RSM and SQS-AL becomes smaller, while the idle time in SQS-AL schedule becomes larger. Since there is no idle time in the RSM schedule, the difference in the idle time between SQS-AL and RSM is the idle time in the SQS-AL schedule. The horizontal distance between “SQS-RSM” points and the Y axis in Figure

8.47 is an unnecessary delay in SQS-AL schedule as also shown in Figure 8.46, discussed earlier.

8.3.4 Schedule in Microsoft Project

After ChaStrobe’s Analyzer performs the analyses and records the results in Analyzer.xls, users can create a schedule in Microsoft Project by clicking the “Create output in MS Project” button. Figure 8.48 shows the finalized SQS-AL schedule for Example 8.1 in a MS Project file, created automatically. For each activity in the created schedule, the name of the activity is placed inside the bar, whereas the name(s) of resource(s) serving that activity is placed on the left side. The number on the right side of the bar indicates the probability of the activity being on the controlling sequence.

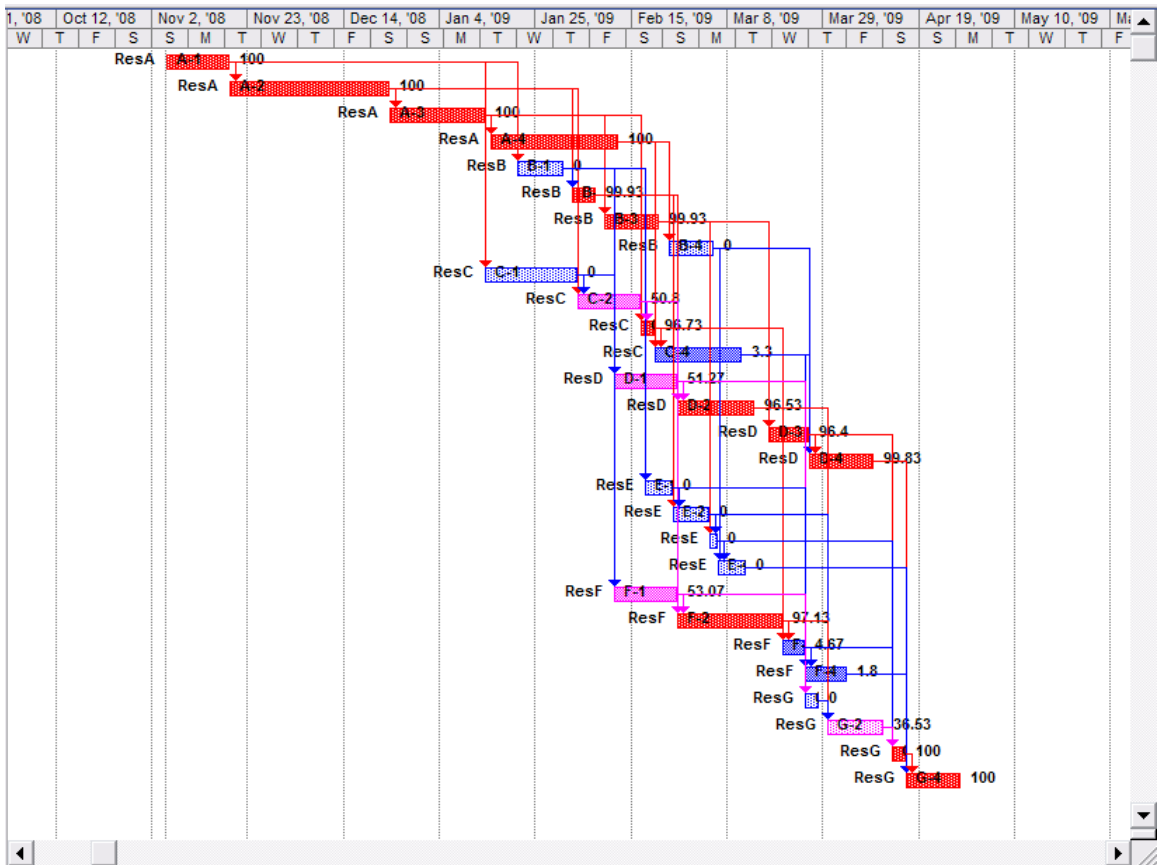


Figure 8.48 The finalized SQS-AL Schedule in Microsoft Project

8.4 Summary

This chapter presents the ChaStrobe application, which is built on top of the add-on that serves as the Stroboscope Graphical User Interface (Stroboscope GUI). ChaStrobe takes input from users, and creates simulation model and code based on the sequence step algorithm (SQS-AL) and the simulation model. After the simulation creation, simulation model and code can be viewed or modified by users prior to simulation execution, which is performed by the Stroboscope Integrated Development Environment (Stroboscope IDE) via either Stroboscope GUI or ChaStrobe.

To model and solve problems of repetitive project scheduling, ChaStrobe takes two main types of input, Simulation Parameters and Project Input. Simulation Parameters are variables and constants used in simulation code in order to control the mechanism of the simulation, such as the number of replications and sequence steps, and BinCollectors' intervals.

Project inputs are activity-related and resource-related data used to create simulation model for repetitive projects, conforming to SQS-AL and the simulation model templates. Accordingly, Project Input consists of four main input types:

- 1) Precedence Input collects activities' names and predecessors.
- 2) Quantity Input collects activities' production rates and work amounts in each unit.
- 3) Resource Input collects resources' names, confidence levels, number of resources, and continuity constraints (including work breaks).
- 4) Utilization Input collects activities' resource utilization.

Four examples of repetitive projects are used to demonstrate how to enter data in ChaStrobe and solve the problem of repetitive projects with probabilistic activity durations. The main characteristics of these four example projects are:

- Example 8.1 Simple repetitive project (no work breaks nor resource-sharing activities)
- Example 8.2 Repetitive project with work breaks (no resource-sharing activities)
- Example 8.3 Repetitive project with resource-sharing activities (no work breaks)
- Example 8.4 Repetitive project with work breaks and resource-sharing activities

As presented in Section 8.3, ChaStrobe offers four features to create graphical presentations and analyze the simulation results. Changes in project duration and idle time from one processing SQS to another can be analyzed, as well as the probability of project duration. Moreover, the simulation results derived from SQS-AL are compared to CPM and RSM. The graphical presentations created by ChaStrobe after simulation execution are:

- Average project duration and idle time from each SQS
- Probability density functions of project duration from each SQS
- Cumulative distribution functions of project duration from each SQS
- Production diagram in Static Graphs
- Cumulative distributions of project duration, derived from CPM, RSM, SQS-AL

- Probability density functions of project duration, derived from CPM, RSM, SQS-AL
- Comparison between the project duration of RSM and SQS-AL to CPM
- Comparison between the project duration of SQS-AL to RSM
- Difference in project duration and idle time between RSM and CPM, between SQS-AL and CPM, and between SQS-AL and RSM.
- Activity and resource schedule in Microsoft Project

The ChaStrobe application is a powerful tool used to solve scheduling problems of repetitive projects with probability activity durations. ChaStrobe provides a systematic and automatic simulation code and model generation for repetitive projects, modeled in Stroboscope GUI. Employing ChaStrobe's capabilities and automation, schedulers benefit from the application in both the early (generating simulation model and code) and late (creating and analyzing simulation results) stages of problem solving.

In the following chapter, optimization of repetitive projects is presented. The optimization processes are performed automatically by ChaStrobe's optimization features. When many alternatives of activity and resource schedules are available, users can use ChaStrobe's optimization to test a large number of combined alternatives in order to optimize the problem.

CHAPTER 9

OPTIMIZATION IN CHASTROBE

In the previous chapter, the ChaStrobe application and the collaboration between ChaStrobe and Stroboscope are presented. ChaStrobe's simulation code and model generation reduces users' time and effort in modeling and solving the problems of repetitive project scheduling. By modifying Project Input in ChaStrobe, users can examine various alternative schedules in order to improve their projects.

Nevertheless, when there are many alternatives in scheduling activities and resources, the combinations of these alternatives may result in a large number of possible simulation model and code. Manually entering, modifying, and executing the input for such large combinations can become time-consuming, tedious, and cumbersome, even using the ChaStrobe application. Consequently, the large number of alternative schedules prohibits schedulers to test all the possibilities; as a result, a selected schedule may defer greatly from an optimum schedule.

In response to the difficulty, automation of simulation code and model manipulation is implemented in ChaStrobe with two search methodologies, the exhaustive search and the genetic algorithm. These search methodologies are used to optimize problems, modeled in Stroboscope Graphical User Interface (Stroboscope GUI). To manipulate the simulation code and model in Stroboscope GUI, ChaStrobe offers

three levels of simulation code and model manipulation. These three levels of simulation manipulation range from simple simulation code modification to complicated code and model modification. With ChaStrobe's automation, simulation code and model are automatically created, modified, and executed. Then, ChaStrobe retrieves the simulation results (stored in tempChaStrobe_GA.txt) and employs either the exhaustive search or the genetic algorithm to optimize the problem.

9.1 Overview of ChaStrobe's Optimization

The optimization process in ChaStrobe consists of nine steps, shown in Figure 9.1. In order to optimize a problem of repetitive project scheduling in ChaStrobe, users must first complete Step 1, entering project-related inputs (Simulation Parameters and Project Inputs are discussed in Chapter 8) and Step 2, entering optimization-related inputs (discussed in this chapter). In Step 1, users provide inputs about the repetitive project (such as precedence constraints), and also determine how simulation code and model will be constructed (such as the number of replications). These inputs are Project-Related Inputs, including ChaStrobe's Simulation Parameters and Project Input, explained in Chapter 8. After users enter the project-related inputs, ChaStrobe and Stroboscope should be able to create a valid simulation code and model from the inputs and execute the simulation without an error.

In Step 2, users provide ChaStrobe information about the considered alternatives (such as different confidence levels for resource utilization), additional simulation code (such as an additional precedence constraint for a specific scenario), and how to optimize the problem (such as objective functions). These inputs are optimization-related inputs,

including ChaStrobe’s Dynamic Code Input, Search Input, an objective function, and Search Parameters. Details of optimization-related inputs are discussed in Section 9.2.

In Step 3, after entering the project-related and optimization-related inputs, users choose between the exhaustive search and the genetic algorithm (discussed in Section 9.5) to initiate the optimization process.

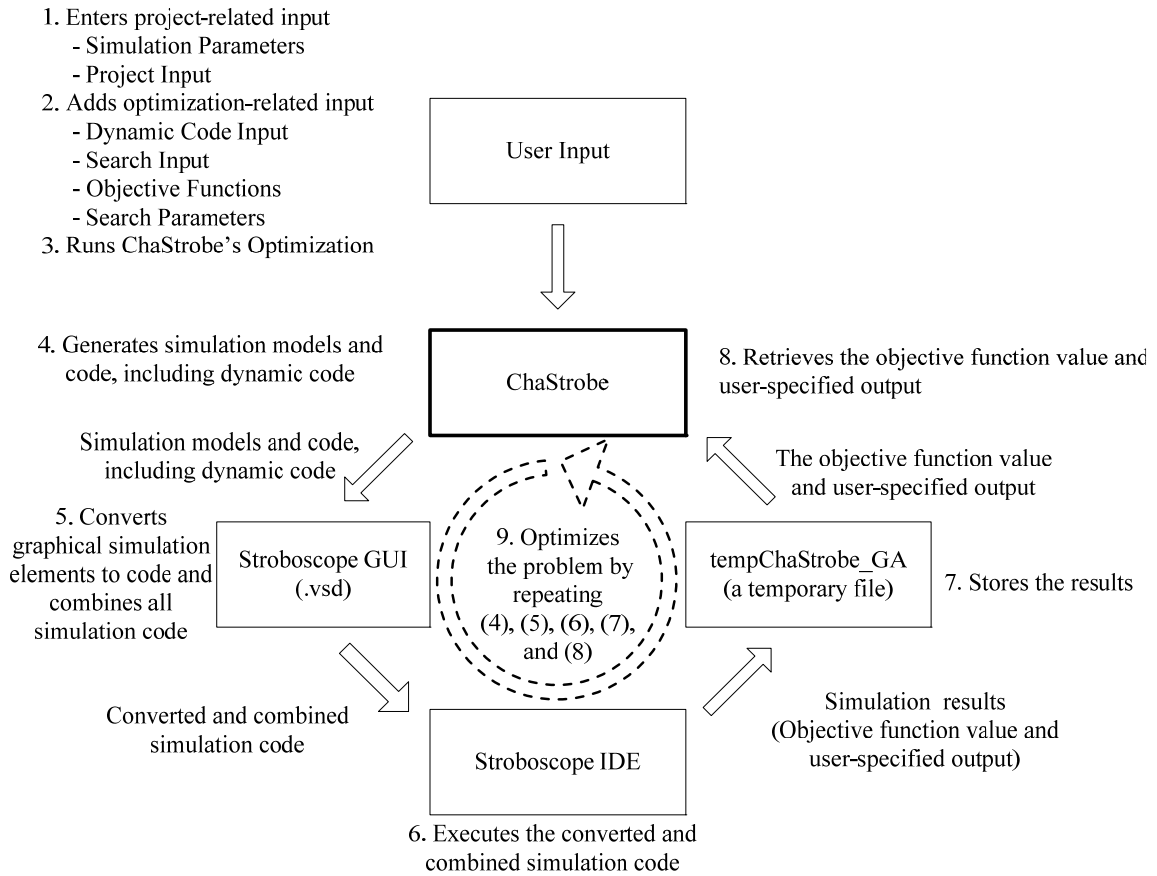


Figure 9.1 Nine steps of optimization process in ChaStrobe

In Step 4, ChaStrobe generates simulation model from Project Input and Simulation Parameters given by the users. The simulation generation for optimizing a problem is performed in the same as for simply solving a problem with two additional crucial parts, updating values of decision variables and including dynamic code. Prior to the simulation generation in optimization, ChaStrobe updates the values in the decision

variable cells on the Search Input sheet based on a selected search method. Then, cells on the Dynamic Code Input sheet referencing the decision variables cells will be updated automatically to derive new updated dynamic simulation code. After the simulation code on the Dynamic Code Input sheet is updated, ChaStrobe will include this code, called “dynamic code,” into various positions in the main simulation code. The positions in which the dynamic code is added are based on a user-specified dynamic code index (discussed in Section 9.2.2).

Steps 5 to 6 for optimizing a problem are performed in the same way as in simply solving the problem, as discussed in Chapter 8.

In Step 7, the resulting objective function value from the simulation must be stored in tempChaStrobe_GA.txt, when using GA. Figure 9.6 shows how to store the objective function value by using Stroboscope PRINT statement. Users must ensure that the objective function value is stored and is the first number in tempChaStrobe_GA.txt.

Note that tempChaStrobe_GA.txt is only a temporary storage for the results from simulation. After ChaStrobe retrieves the data (the objective function values and other user-specified outputs) in the text file, at the end of simulation for each alternative, and stores them in ChaStrobe, it will delete the data in the text file.

In Step 8, at the end of simulation for each alternative, ChaStrobe automatically retrieves the simulation results of the objective value from tempChaStrobe_GA.txt, and stores the results on the Search Output sheet. ChaStrobe uses the first number in the text file as the objective function value for GA.

In Step 9, ChaStrobe changes the values of the decision variables in Row 2 on the Search Input sheet (these cells are called “decision variable cells”) in order to modify the

Project Input, which in turn will change the simulation code and model when created in Step 4. These decision variable cells in Row 2 on the Search Input sheet can be referenced by various cells on other sheets in the Project Input, such as cells on the Precedence Input and Resource Utilization sheets. Changing values in decision variable cells on the Search Input sheet will automatically change values in those referencing cells on other input sheets. After the values in decision variable cells are changed and the cells referencing to them are updated, ChaStrobe repeats Steps 4 to 9. These six steps are repeated many times depending on search methods. For the exhaustive search, ChaStrobe repeats these steps for the number of all possible alternatives, a combination of decision variables given by the user on the Search Input sheet. For GA search, ChaStrobe repeats the steps for a number of times equal to the product of the number of generations and the number of populations.

9.2 Optimization Input

As shown in Figure 9.2, optimizing simulation model in Stroboscope GUI requires ChaStrobe to change the values in the decision variable cells (Step1) in order to change Project Input (Step 2) based on the inputs specified by users on the Search Input sheet and to create a new simulation model (Step 3). Then, the new simulation model is executed to derive the objective function value. For GA, the objective function value is used to evaluate the inputs specifying the simulation code and model modification.

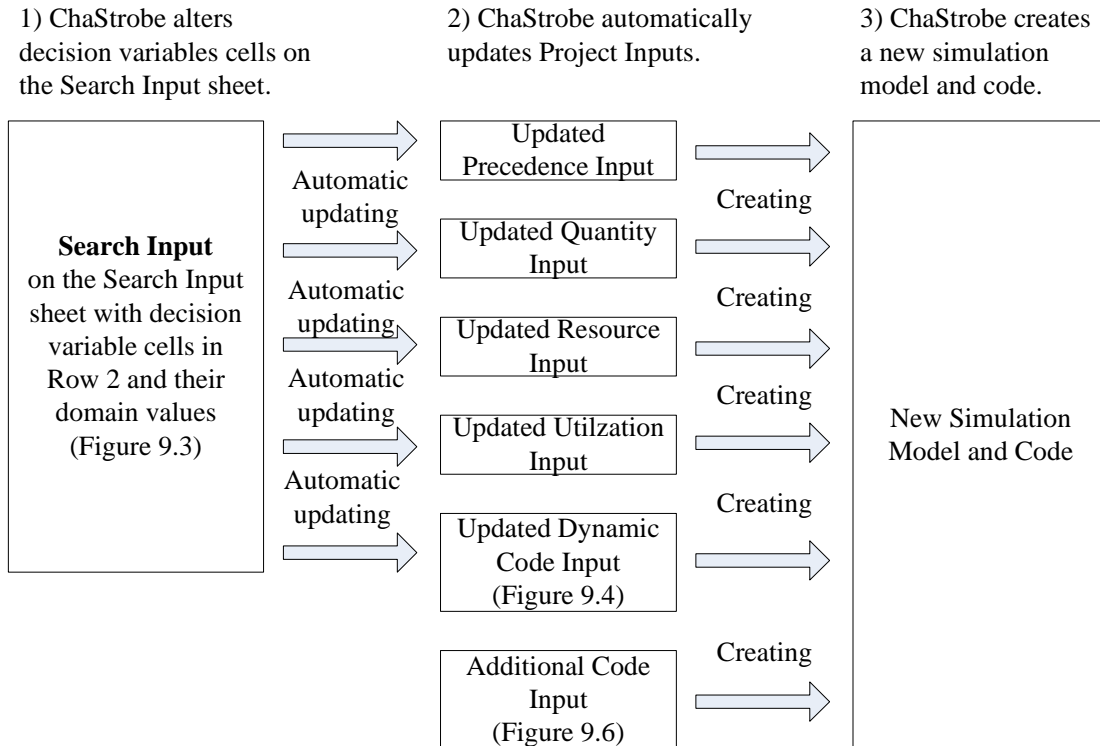


Figure 9.2 Modifying inputs, updating inputs, and creating simulation code and model

9.2.1 Search Inputs

Search Inputs are the inputs, given by users before optimization, specifying the current values of decision variables and their domain values. The names of decision variables are defined by the users in Row 1 on the Search Input sheet, as shown in Row 1 in Figure 9.3. These names specified by users are used only as a reference for the users on the Search Input sheet; they are not used for any other purposes. Accordingly, the names of decision variables in Row 1 on the Search Input sheet can be omitted.

The current values of decision variables are placed in the decision variable cells in Row 2 on the Search Input sheet, as shown in Row 2 in Figure 9.3. During optimization, ChaStrobe changes the values of the decision variables in decision variable cells in order to change Project Input, thus resulting in a different simulation model.

Figure 9.3 is an example of optimizing a repetitive project scheduling problem by considering the confidence levels for resources. Domain values of the decision variables (e.g., confidence levels) range from 0 to 1 with a step of 0.1, as shown in Rows 4 to 14 in Figure 9.3.

	A	B	C	D	E	F
1	B_Conf Level	C_Conf Level	D_Conf Level	E_Conf Level	F_Conf Level	G_Conf Level
2	0.2	0.3	0.4	0.5	0.6	0.7
3						
4	0	0	0	0	0	0
5	0.1	0.1	0.1	0.1	0.1	0.1
6	0.2	0.2	0.2	0.2	0.2	0.2
7	0.3	0.3	0.3	0.3	0.3	0.3
8	0.4	0.4	0.4	0.4	0.4	0.4
9	0.5	0.5	0.5	0.5	0.5	0.5
10	0.6	0.6	0.6	0.6	0.6	0.6
11	0.7	0.7	0.7	0.7	0.7	0.7
12	0.8	0.8	0.8	0.8	0.8	0.8
13	0.9	0.9	0.9	0.9	0.9	0.9
14	1	1	1	1	1	1
15						
16						
17						

Figure 9.3 The Search Input sheet and the current decision variable cells in Row 2

To collect decision variables and their domain values on the Search Input, ChaStrobe determines 1) the number of decision variables and 2) the number of domain values for each decision variable. First, ChaStrobe determines the number of decision variables by counting non-blank cells in Row 2 on the Search Input sheet, starting from Column A. ChaStrobe will stop counting when it reaches a blank cell in Row 2 (e.g., Cell G2 in Figure 9.3). For the example in Figure 9.3, there are six decision variables (non-blank cells in Row 2 from Columns A to F). ChaStrobe uses the column index to indicate each decision variable. Therefore, the first decision variable is specified in Column A, and the second variable is in Column B, and so forth.

Second, ChaStrobe determines the number of domain values for each decision variable (one decision variable has one column) by counting non-blank cells in the row

direction, starting from Row 4. For the example in Figure 9.3, decision variable index 1, as shown in Column A, “B_Conf Level”, has 11 domain values. Each domain value is indexed by the row order in which it is placed. For example, the indexes of domain values in Column A for 0, 0.1, 0.2 and 0.3 are 1, 2, 3, and 4, respectively. After the number of decision variables and the number of domain values for each decision variables are determined, ChaStrobe creates a two-dimensional array to store them. For example in Figure 9.3, there are 6 decision variables with 11 domain values for each of them. Note that each decision variable can have a different number of domain values. See Figure 9.17 for example.

The currently selected values for the decision variables are in decision variable cells, which are cells in Row 2 on the Search Input sheet, as shown in Figure 9.3. The values of these decision variable cells are referenced by other cells on other input sheets in ChaStrobe, which are used to create a new simulation model. During optimization, ChaStrobe alters the values of the decision variable cells using the domain values given by the users on the Search Input sheet. The means of selecting new values of each decision variable depends on the given domain values on the Search Input sheet and the chosen search method. Since the values in decision variable cells are referenced by other cells on Project Input, changing the values in decision variable cells (e.g., Figure 9.3 in Row 2) will change the input for the project used to create the simulation code and model (e.g., Figure 9.4 in Column D). Referencing cells within the spreadsheets in ChaStrobe is done in the same way as in Excel spreadsheet, because the spreadsheets in ChaStrobe are actually Excel components.

In the beginning, users can specify any initial values for the decision variable cells so that they can see and verify that the value of the cells on other input sheets that reference back to these decision variable cells is correct. For example, Cell D2 on the Dynamic Code Input sheet in Figure 9.4 is referenced to Cell A2 on the Search Input sheet in Figure 9.3.

9.2.2 Dynamic Code Input

Dynamic code is simulation code whose part of the code changes according to the currently selected values for the decision variables. Users create dynamic code on the Dynamic Code Input sheet. For the changing parts of the dynamic code, users reference the decision variable cells on the Search Input sheet showing the currently selected values for the decision variables. In Figure 9.4, for example, the changing part of the code in Row 2 is the confidence level for ResB (Column D). An Excel formula must be entered in Cell D2 on the Dynamic Code Input sheet that references Cell A2 on the Search Input sheet (Figure 9.3). For this example, similar formulas, referencing decision variable cells on the Search Input sheet, need to be entered in all cells in Column D on the Dynamic Code Input sheet to obtain the currently selected values for the SaveValues specified in Column C on the Dynamic Code Input sheet. Another example of dynamic code is shown in Figures 9.16 and 9.17.

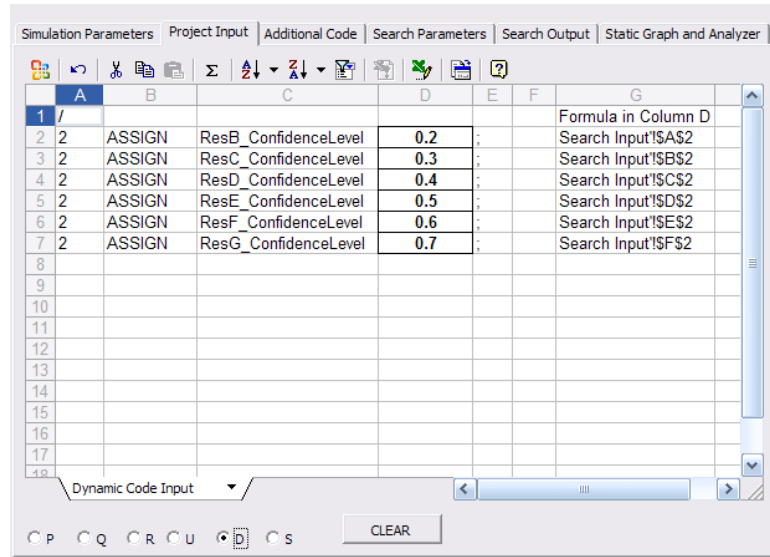


Figure 9.4 Dynamic code input with cells in Column D reference to the decision variable Cells on the Search Input sheet, shown in Figure 9.3

To establish dynamic code, ChaStrobe collects simulation code in the column direction starting from Column B until reaching a blank cell. The code collected from cells in the same row is separated by a space. ChaStrobe considers each row on the Dynamic Code Input sheet as one line of simulation code. For example, ChaStrobe collects code in Cells B2, C2, D2, E2, and stops, since Cell F2 is a blank cell. Accordingly, the dynamic code derived from Row 2 in Figure 9.4 is

“ASSIGN ResB_ConfidenceLevel 0.2;”

Figure 9.5 displays code positions in the combined code between the main code and dynamic code. On the left-hand side of Figure 9.5, the main code is presented. The main code is the code created by Stroboscope GUI, consisting of: 1) the simulation code in Model Parameters, Programming Objects, and Control Statements, and 2) the code converted from graphical simulation elements such as Queues and Combis in Stroboscope GUI.

To determine the position where each line of the dynamic code will be inserted into the main simulation code, users must specify a dynamic code index (1 to 6) in Column A for each row on the Dynamic Code Input sheet. For example, dynamic code in Row 2 in Figure 9.4 is given a dynamic code index of 2, meaning that it will be inserted into the main code after the code for Model Parameter in the main code, as shown in Figure 9.5.

Besides specifying an index for each dynamic code on the Dynamic Code Input sheet, users must ensure that the order in which simulation code will be presented to Stroboscope is valid. Since Stroboscope reads and interprets simulation code sequentially line by line, names of model elements (e.g., Queues and Combis), resources, and variables must be declared before they can be referred elsewhere in the simulation code.

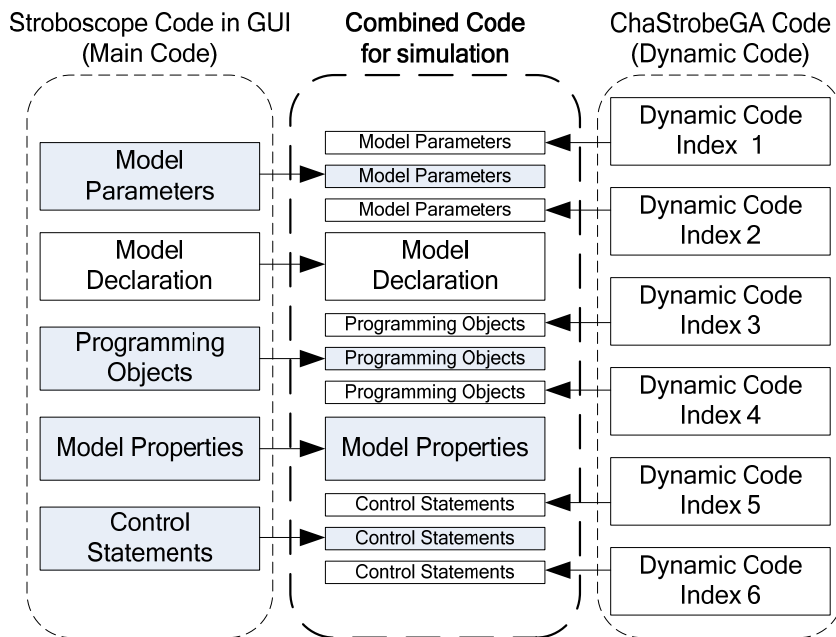


Figure 9.5 Dynamic code indexes and dynamic code positions in the main code

Four parts of the main code, generated by ChaStrobe through the Stroboscope GUI, are the simulation code for Model Parameters, Programming Objects, Model

Properties, and Control Statements. They are stored in the simulation in the Stroboscope GUI in the form of simulation code. These four parts of the main code are shown in the gray rectangles on the left side in Figure 9.5. On the other hand, the main code for Model Declaration, generated by the Stroboscope GUI, is stored in the simulation in the form of graphical simulation elements (e.g., Queues, Combis, and Links). Before executing a simulation model in optimization, ChaStrobe creates the four part of the main code, updates the dynamic code, and inserts the dynamic code into the four part of the main code. Then, upon the execution, Stroboscope GUI creates the simulation code for the graphical simulation elements (converting them to code) and combines all the code for simulation execution in Stroboscope GUI.

Note that the main code generated by ChaStrobe is derived from the inputs on the four main project input sheets: Precedence Input, Quantity Input, Resource Input, and Utilization Input, as discussed in Chapter 8.

9.2.3 Additional Consistent Code

Additional consistent code is the code that does not change throughout the optimization and is not subject to changes in the decision variables. Users enter the additional consistent code in the Additional Code tab, as shown in Figure 9.6. On the Additional Code tab, there are five sub-tabs indicating five different positions where the additional consistent code will be placed in the main code after the part specified by the name of the sub-tabs.

Prior to dynamic code insertion and simulation execution, ChaStrobe collects the additional consistent code and inserts it into the main code. For example, in Figure 9.6, additional consistent code on the sub-tab of Control Statements in the Additional Code

alter decision variables in the simulation, which in turn modify the simulation code and/or model. The modified simulation, then, is executed to derive an objective function value for optimization purposes, as discussed so far.

The exhaustive search is for testing all the possibilities of combinations of decision variables. It is recommended when the number of the possibilities is small. Details of the exhaustive search are given in Section 9.5.1. To start the exhaustive search, users click on the “Run Exhaustive Search” button on the tab of Search Parameters, as shown in Figure 9.7.

The genetic algorithm (GA) is a search technique based on the mechanism of natural selection. GA uses the objective function value from simulation results (stored in tempChaStrobe_GA.txt) to evaluate the selection rate of each combination of decision variables. To use the genetic algorithm, users enter GA parameters, shown in Figure 9.7, and click the “Run GA Search” button. Details of GA and GA parameters are discussed in Section 9.5.2.

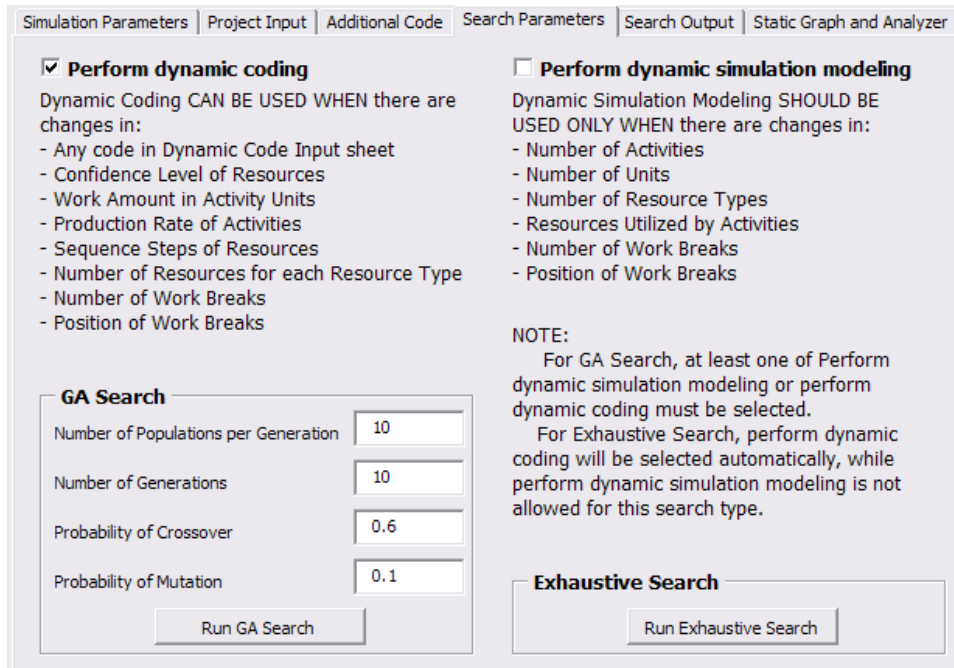


Figure 9.7 Search Parameters with two main search methods, Exhaustive Search and Genetic Algorithm

In Figure 9.7, there are two options for simulation manipulation: 1) Perform dynamic coding and 2) Perform dynamic simulation modeling. The option of dynamic coding is used when optimization requires changes in simulation code, whereas the option of dynamic modeling is used when optimization requires changes in simulation model (graphical simulation elements and the code in them). Details of Dynamic Coding and Modeling are discussed in Section 9.3.

9.3 Simulation Code and Model Manipulation

ChaStrobe offers three levels of simulation model and code manipulation in order to optimize problems. These three levels are used to answer the need in modifying either simulation code and/or simulation model. After the specification, users can properly enter Project Input and select necessary options of dynamic coding and modeling.

First, Parameter manipulation is for altering values of variables in simulation code in order to optimize problems. Variables in Stroboscope's language that can be altered must be stored in SaveValues. Examples of decision variables in the SQS-AL simulation code are: 1) confidence levels, 2) resources' sequence steps, and 3) the number of resources for each type.

To employ parameter manipulation, users must create dynamic code for the parameters on the Dynamic Code Input sheet. The part of the dynamic code that changes according to the values of the decision variables must reference the decision variable cells on the Search Input sheet. Then, users must select the option of dynamic coding before initiating a search method, either the exhaustive search or the genetic algorithm (GA) in Figure 9.7. An example of optimization using GA is shown in Section 9.6.

Most simulation software with optimization offers this capability of parameter manipulation. It is a simple means of optimizing problems by altering certain parameters in the simulation model. However, parameter manipulation limits users to optimizing a problem by changing the value of decision variables, which may not be sufficient to optimize problems with various scenarios. Accordingly, ChaStrobe offers simulation code manipulation, an extended version of the parameter manipulation that allows additional code such as conditional statements to be added to or altered in the simulation, in addition to variables.

Secondly, code manipulation involves altering simulation code upon decision variables in order to optimize problems. It is used to add additional code into the main code or change part of the additional code. Simulation code manipulation is a superset of parameter manipulation. Examples of simulation code manipulation in the SQS-AL

simulation code are: adding additional precedence constraints, specifying working sequence for resources, and changing objective function based on different scenarios. Simulation code manipulation is very useful and not time consuming. To employ simulation code manipulation, users must select the option of dynamic coding, shown in Figure 9.7. An example of optimizing a problem requiring dynamic coding is demonstrated in Section 9.6.

However, parameter and code manipulation are strictly for simulation code modification. They cannot modify a simulation model in order to optimize problems.

Thirdly, model manipulation involves changes in the four parts of the main code generated by ChaStrobe in order to optimize the problem. It is used when there are many simulation model alternatives that require adding, removing, or changing graphical simulation elements (such as Combis and Queues) or modifying the main code generated by ChaStrobe (such as adding simulation code for work breaks). Examples of simulation model manipulation are adding a repetitive activity (adding a work flow sub-network), removing a resource (removing a resource flow sub-network), changing resource utilization (changing links between work flow and resource flow sub-networks), and applying work breaks (adding or removing the simulation code for work breaks). These examples require both simulation code and model modification. Accordingly, ChaStrobe offers an automation of simulation code and model manipulation used to serve such need in optimizing repetitive project scheduling problems.

When a cell on the four main input sheets (Precedence Input, Quantity Input, Resource Input, and Utilization Input sheets) references a cell on the Search Input sheet or Dynamic Code Input sheet, simulation model manipulation is required. Changes in the

values of the decision variables that change the inputs on the four main input sheets require a re-construction of the simulation including the simulation code and graphical simulation elements created by ChaStrobe. If the changes can be accomplished without referencing cells the four main input sheets to cells on the Search Input sheets or the Dynamic Code Input sheet, simulation code manipulation should instead be used. When simulation model manipulation is required, users must select the option of Dynamic Modeling (the checkbox of “Perform dynamic simulation modeling” in Figure 9.7. An example of optimizing a problem requiring dynamic coding is demonstrated in Section 9.6.

9.4 Search Output from ChaStrobe’s Optimization

Figure 9.8 displays an intermediate result from optimization on the Search Output sheet, which is terminated by the user before the simulation for the alternative shown in Row 14 has been completed. Each row on the Search Output sheet represents one alternative for the simulation model created by different values of decision variables. The outputs on the Search Output sheet can be categorized into three sets: 1) the values of decision variables (Columns A to F), 2) the resulting objective function values (Column H), and 3) the user-specified additional outputs (Columns L to O).

Firstly, the selected values of decision variables are placed in cells starting from Column A in the same order that users setup the decision variable cells on the Search Input sheet. For example, in Figure 9.8, the value in Cell A3 is the confidence level for ResB, Cell B3 is the confidence level for ResC, and so forth. Secondly, the resulting objective function value from the simulation is placed two cells away (Column H in Figure 9.8) from the last cell for the decision variables (Column F). Thirdly, user-

specified additional outputs are placed four cells way (Column L) from the objective function value cell (Column F). The data in Columns I and J in Figure 9.8 are calculated and shown only when users use the genetic algorithm (GA) to optimize problems.

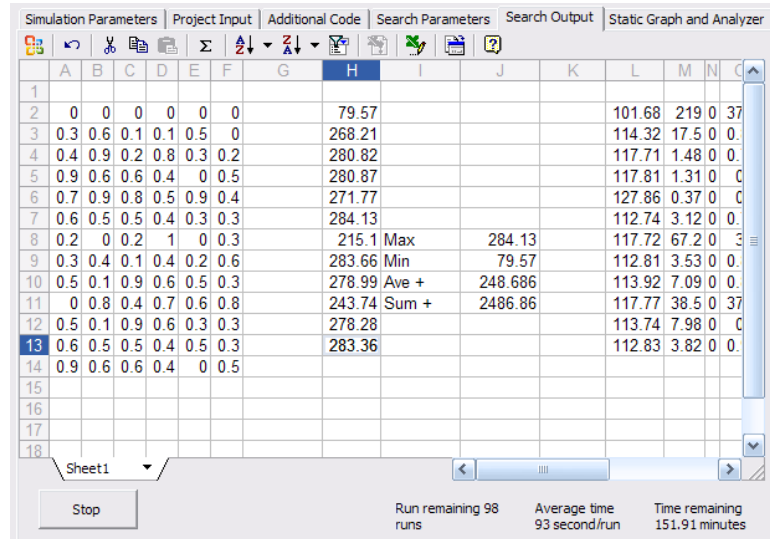


Figure 9.8 Output from optimization using the genetic algorithm

9.5 Search Methods in ChaStrobe

ChaStrobe offers two search methods to optimize problems modeled by ChaStrobe in the Stroboscope GUI: the exhaustive search and the genetic algorithm (GA).

9.5.1 The Exhaustive Search

The exhaustive search is a procedure of testing all possible combinations of decision variables in order to find the best solution. For the exhaustive search, ChaStrobe creates all possible combinations, generates the simulation from the combinations, and executes the simulation. After all the possibilities are tested, then the best combination can be identified. The exhaustive search is suggested only when the number of

possibilities is small or there is no time constraint. Otherwise, it may be inefficient or impossible to test all the possibilities. It also provides a baseline for evaluating the effectiveness of GA, which will be discussed next.

9.5.2 The Genetic Algorithm

The genetic algorithm (GA) is a search algorithm based on the mechanism of natural selection and genetics that healthy creatures are most likely to survive from one generation to another, while unhealthy creatures will become extinct. Birthing a new generation, the survivors mate and produce offspring whose chromosomes are inherited from their parents. Occasionally, the offspring may genetically mutate (Goldenberg 2004).

Imitating the natural mechanism, the genetic algorithm weighs the selection rate of combinations (creatures) of decision variables (chromosomes) based on their resulting objective function value (healthy or unhealthy); a combination is a vector of decision variable values, where as its resulting objective function value is the degree of fitness. Then, the selected combinations (survivors) are paired, and exchange their variables (mate) in order to derive a new combination (offspring). In a new generation, new combinations with mixed decision variables (inherited) from their predecessors (parents) are likely to improve the selection rate, in other words, the objective function value. In addition, decision variables of the new combinations are occasionally altered (mutated) to prevent a local optimum.

As stated above, GA in ChaStroke is composed of six main processes: Initialization, Simulation, Evaluation, Reproduction, Crossover, and Mutation. Figure 9.9 depicts GA's six main processes for the example shown in Section 9.2.

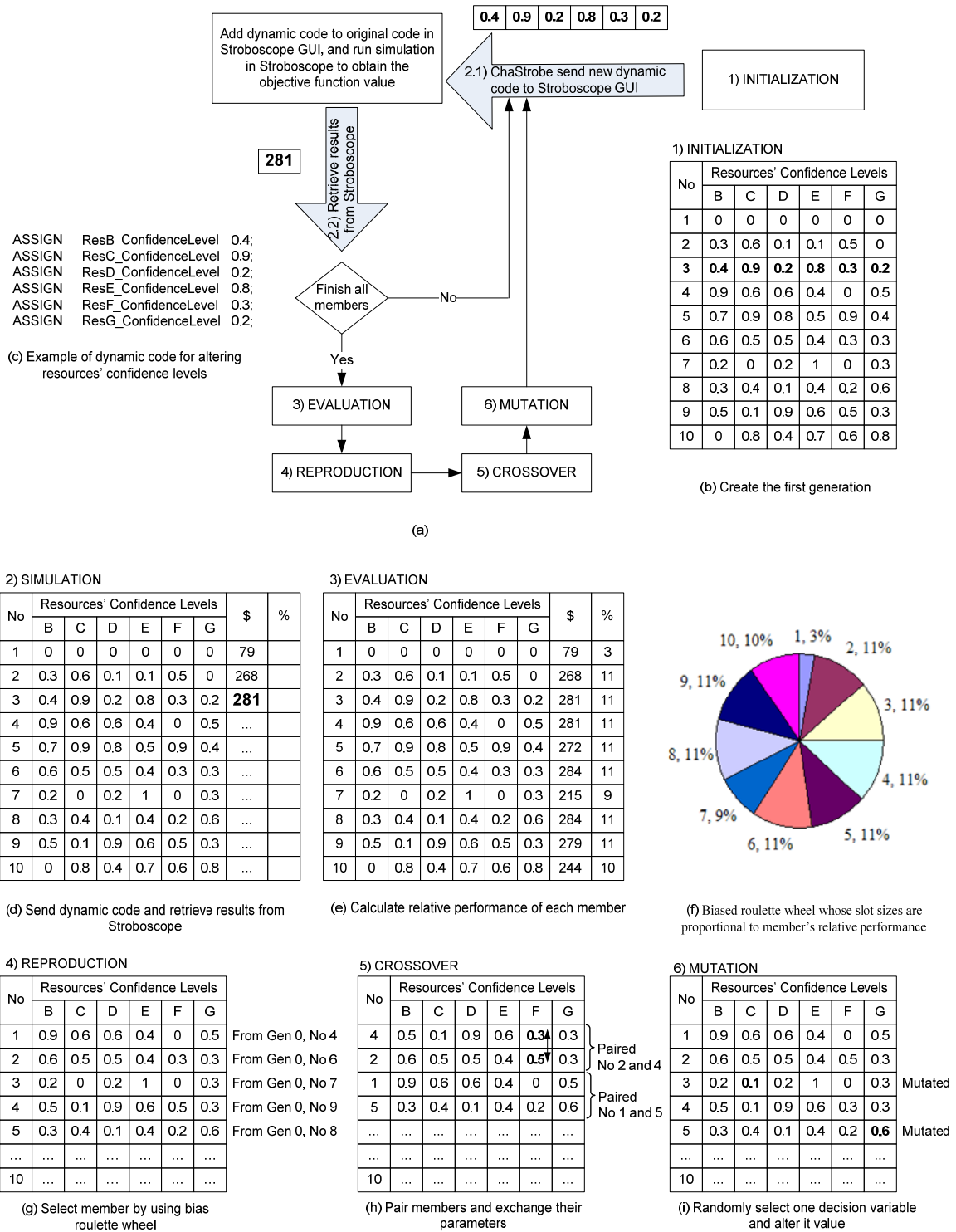


Figure 9.9 The Genetic Algorithm in ChaStrobe

Initialization is the process of creating the initial generation that does not have a preceding generation. To create the initial generation, ChaStrobe uses the uniform distribution to randomly select values for each of the decision variable. As shown in Figure 9.9.a, the combinations for the decision variables for the initial generation are those assigned by the user on the Search Input sheet in Figure 9.3.

Execution is the process of creating and executing the simulation. The execution process for GA involves 3 steps:

Execution.1) Create the simulation for the currently selected values of decision variables, which is the simulation for the current combination. Figure 9.9 shows that the current combination is No 4 in the initial generation. Note that, if users only select dynamic coding, as shown in Figure 9.7, ChaStrobe will create one version of original main code that will be used without changing (being consistent) throughout the GA process. Then, the dynamic code is created based on the currently selected values of decision variables for the current combination and added to the original main code.

Execution.2) Executing the simulation. After the simulation for the current combination is created, ChaStrobe automatically submits it to Stroboscope for simulation execution.

Execution.3) Retrieving simulation results. After the simulation execution, ChaStrobe retrieves the simulation results by reading the data stored in an automatically created text file called tempChaStrobe_GA.txt.

These three steps are repeated until all combinations for the current generation are executed. Then, ChaStrobeGA will start the GA evaluation process.

Evaluation is the process of calculating the chance of survival of each combination based on each combination's *relative performance* within the same generation. The relative performance of a combination of decision variables is the ratio between its resulting objective function value and the sum of all objective function values for all combinations in the same generation. The greater this ratio for a particular combination is, the better chance the combination has to survive and be chosen for the next generation. Figure 9.9.e shows the relative performance of all combinations in percent based on their objective function value. For example in Figure 9.9.e, Combination 1 has only a 3% chance to be chosen for the next generation, while Combination 2 has an 11% chance. Hence, combinations' relative performance reflects the probability of that combination being chosen for the reproduction process.

Reproduction is the process of selecting combinations for a new generation by using a biased roulette wheel whose slot size is proportional to combinations' relative performance. Figure 9.9.f depicts the biased roulette wheel for the initial generation. To produce the next generation, the wheel is spun 10 times to get 10 combinations for the new generation. Figure 9.9.g shows examples of five selected combinations from the previous generation to the next generation. After reproduction finishes, the crossover process starts.

Crossover is the process of exchanging values of decision variables between combinations. Crossover consists of two steps.

Crossover.1) Pairing two combinations. Two combinations are randomly selected and paired. Figure 9.9.h shows the reproduced Combinations 2 and 4 are paired, as well as for Combinations 1 and 5.

Crossover.2) Then, GA determines whether to exchange decision variables between combinations by using the crossover probability given by users in Figure 9.7. The decision is made by using a uniform distribution, randomly creating a number between 0 and 1. If the random number is less than the crossover probability, GA will, then, unbiased and randomly select a position of decision variable, and exchanges the decision variables in the position between the two paired combinations Figure 9.9.h shows Combinations 2 and 4 exchange a decision variable in Position 5, while Combinations 1 and 5 do not.

These two steps of crossover are repeated until each combination has its pair, and has been determined whether to exchange its decision variables with its paired combination. After the crossover operation finishes, the mutation operation begins.

Mutation is the process of altering decision variables in order to prevent premature loss of important notions (Goldberg 2004). Mutation ensures that results from GA are not limited by a local optimum. To decide whether to mutate a combination, GA randomly generates a number from 0 to 1 for the combination. Then, GA compares the random number to the user-specified mutation probability (see Figure 9.7). If the random number is less than the mutation probability, the combination will be mutated. To select a decision variable (position) for mutation (assigning a new value for the decision variable), ChaStrobe uses the uniform distribution. Figure 9.9.i shows that Combinations 3 and 5 are mutated in the 2nd and 6th positions, respectively.

These processes of simulation execution, evaluation, reproduction, crossover, and mutation are repeated until the total number of generations given by the users has been

reached. Then, the results from all generations are arranged in the same table and sorted by the objective function values, which is project profit in this case, to derive the optimum solution according to GA.

9.6 Example 9.1 Optimizing a Repetitive Project

The example repetitive project with resource-sharing activities and work breaks from Chapter 8 is used to demonstrate how to optimize the problem using ChaStrobe. Figure 9.10 is a precedence diagram with resource nodes for the example. For this example, GA is used to optimize the problem with parameter, code, and model manipulations as follows:

- Decision variables using “parameter” manipulation are the confidence levels for resources. (See Figure 9.16)
- Decision variables using “code” manipulation are additional semaphores for Activities N, Y, and U. (See Figure 9.16)
- One decision variable requiring “model” manipulation is whether to schedule ResMN’s work break position at the end of M5. (See Figure 9.14)

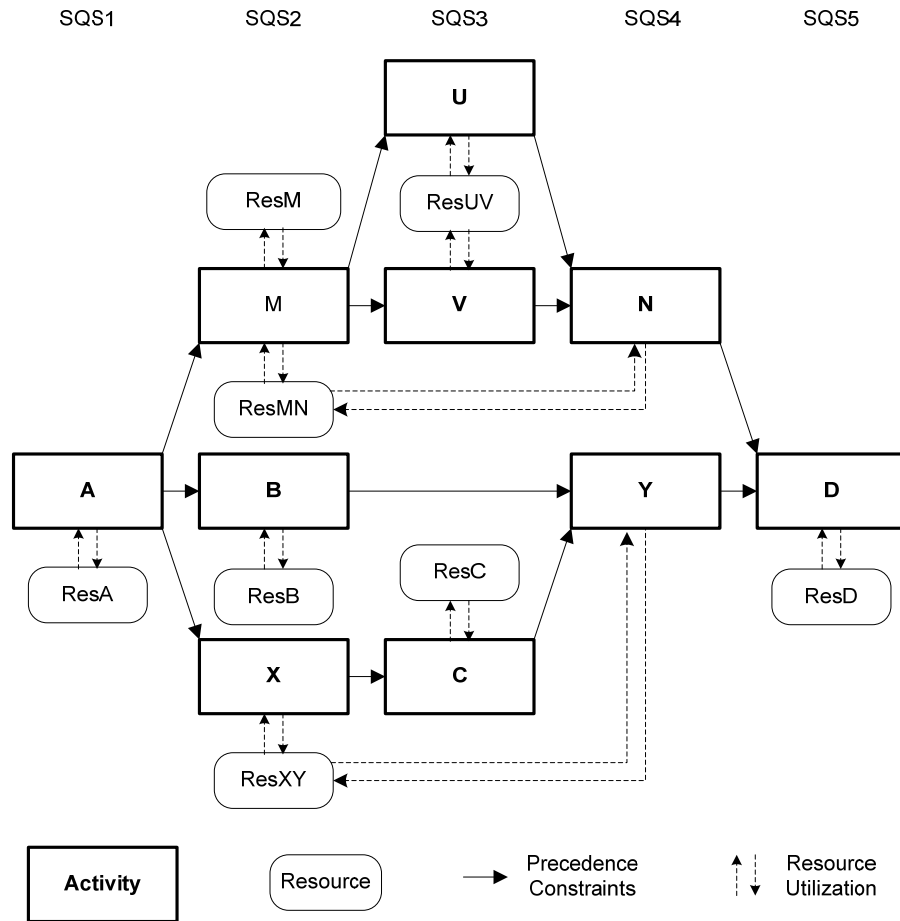


Figure 9.10 Precedence diagram with resource nodes for Example 9.1

Figure 9.11 is the Simulation Input for the example. Notice that the number of replications is set at 100. Since many alternatives of simulation models will be tested, the number of replications should be low, yet still appropriate. Otherwise, it will be too time-consuming. A low number of replications can be used to scope down the possible decision variables and estimate a probable optimum solution in the initial test.

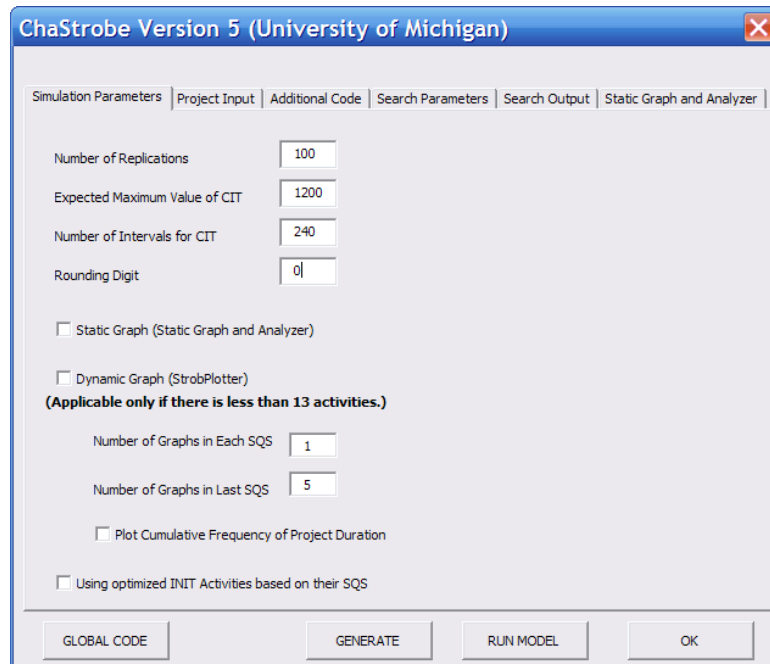


Figure 9.11 Simulation Parameters for Example 9.1

Figure 9.12 and Figure 9.13 display the Precedence Input sheet and the Quantity Input sheet for the example, respectively. The inputs for precedence constraints, work amounts, and production rates shown in the figures remain the same throughout the optimization processes.

	A	B	C	D	E	F	G	H	I	J	K	L
1	ACTIVITY	Pred1	Pred2									
2	A											
3	M	A										
4	B	A										
5	X	A										
6	U	M										
7	V	M										
8	C	X										
9	N	U	V									
10	Y	B	C									
11	D	N	Y									
12												
13												
14												
15												

Figure 9.12 Precedence Input for Example 9.1

	A	B	C	D	E	F	G
1	ACTIVITY	1/PRODUCTIVITY	1	2	3	4	5
2	A	Normal[1,0.1]	40	45	40	40	45
3	M	Normal[1,0.1]	15	15	10	10	10
4	B	Normal[1,0.1]	50	40	50	50	40
5	X	Normal[1,0.1]	20	30	25	20	20
6	U	Normal[1,0.1]	15	20	15	25	20
7	V	Normal[1,0.1]	40	40	45	45	40
8	C	Normal[1,0.1]	15	15	15	15	15
9	N	Normal[1,0.1]	20	25	30	20	25
10	Y	Normal[1,0.1]	20	20	20	20	20
11	D	Normal[1,0.1]	45	35	40	40	30
12							
13							
14							
15							

Figure 9.13 Quantity Input for Example 9.1

Figure 9.14 presents the Resource Input sheet for Example 9.1. On the Resource Input sheet, Cells H7 and I7 reference Cell G2 on the Search Input sheet (Figure 9.17) with different conditional statements, as shown in Rows 11 and 12 in the figure. Cells H7 and I7 specify a work break in Resource ResMN. Cell G2 (on the Search Input sheet in Figure 9.17) is a decision variable with domain values of $\{0, 1\}$ determining whether to include a work break at the end of M5. If Cell G2 on the Search Input Sheet is set to 0, Cell H7 and Cell I7 on the Resource Input sheet will be “N-5” and empty, respectively. This means there is no work break in ResMN’s schedule (Cell G2=0). Without a work break, ResMN will stay on the site until it completes both Activities M and N. On the other hand, if Cell G2 is set to 1, Cell H7 will be “M-5” and Cell I7 will be “N-5”, meaning ResMN will take a break after completing M5 and then return later to finish Activity N.

Changing the value in the decision variable cell in Cell G2 on the Search Input sheet will change the inputs in Cells H7 and I7 on the Resource Input sheet and will change how ChaStrobe creates simulation code, stored within graphical simulation

elements. Since cells (Cells H7 and I7 on the Resource Input sheet for this example) in one of the four main input sheets (Precedence Input, Quantity Input, Resource Input, and Utilization Input sheets) reference to cells on the Search Input sheet, this optimization requires model manipulation. Therefore, users must inform ChaStrobe to perform Dynamic Modeling by checking the option of “Perform dynamic simulation modeling” on the “Search Parameters” tab, as shown in Figure 9.21.

	A	B	C	H	I	J
1	RESOURCE	Confidence Level	INIT	Break 1	Break 2	
2	ResA	0.8	1			
3	ResB	0.8	1	B-5		
4	ResC	0.8	1	C-5		
5	ResD	0.8	1	D-5		
6	ResM	0.8	1	M-5		
7	ResMN	0.8	1	N-5		
8	ResUV	0.8	1	V-5		
9	ResXY	0.8	1	Y-5		
10						
11				Formula in Cell H7: =IF('Search Input!\$G\$2=0,"N-5","M-5")		
12				Formula in Cell I7: =IF('Search Input!\$G\$2=0,"", "N-5")		
13						
14						
15						

Figure 9.14 Resource Input with cells referencing to decision variable cells for Example 9.1

Note that inputs in Figure 9.14 remain the same throughout the optimization process except for the inputs in Cells H7 and I7, which reference Cell G2 on the Search Input sheet. Solid cell borders indicate the varying cells.

Figure 9.15 displays the Utilization Input sheet for the example. The inputs on the Utilization Input sheet remain the same throughout the optimization processes. In this project, there are six resource-sharing activities. Activities M and N require the same Resource ResMN; X and Y require ResXY; and U and V require ResUV.

It is important to mention that Activity M utilizes two types of resources: one dedicated resource, ResM, and one shared resource, ResMN. Since ResM serves only

Activity M in SQS2, ResM's crew lead time ($CLT1_M$) will be determined in processing SQS2. On the other hand, ResMN's crew lead time could be determined in either processing SQS2 or SQS4 depending on whether a work break at the end of M5 is scheduled. When there is no work break in ResMN's schedule, there will be only one crew lead time for ResMN, $CLT1_{MN}$. When there is one work break, there will be two crew lead times for ResMN, $CLT1_{MN}$ and $CLT2_{MN}$. Without a work break, $CLT1_{MN}$ will, by default, be determined at the end of processing SQS4, which is Activity N's SQS. With a work break at M5, $CLT1_{MN}$ will be determined when processing SQS2, which is Activity M's SQS, and $CLT2_{MN}$ will be determined when processing SQS4, which is Activity N's SQS.

	A	B	C	D	E	F	G	H	I	J
1	Activity	Resource 1	Resource 2							
2	A	ResA								
3	M	ResM	ResMN							
4	B	ResB								
5	X	ResXY								
6	U	ResUV								
7	V	ResUV								
8	C	ResC								
9	N	ResMN								
10	Y	ResXY								
11	D	ResD								
12										
13										
14										
15										

Figure 9.15 Utilization Input for Example 9.1

Figure 9.16 presents the dynamic code on the Dynamic Code Input sheet corresponding to the decision variable cells on the Search Input sheet, shown in Figure 9.17. Cells D2 to D8, D10 to D11, and B13 to B15 on the Dynamic Code Input sheet reference the decision variable cells in Row 2 on the Search Input sheet and are automatically updated to reflect the current values of decision variables.

	A	B	C	D	E	F	G	H	
1	/	Confidence Levels							
2	2	ASSIGN	ResB_ConfidenceLevel	0.8	:		Formula in Cell D2:	=Search Input!\$A\$2	
3	2	ASSIGN	ResMN_ConfidenceLevel	0.8	:		Formula in Cell D3:	=Search Input!\$D\$2	
4	2	ASSIGN	ResM_ConfidenceLevel	0.8	:		Formula in Cell D4:	=Search Input!\$D\$2	
5	2	ASSIGN	ResXY_ConfidenceLevel	0.2	:		Formula in Cell D5:	=Search Input!\$F\$2	
6	2	ASSIGN	ResUV_ConfidenceLevel	0.8	:		Formula in Cell D6:	=Search Input!\$E\$2	
7	2	ASSIGN	ResC_ConfidenceLevel	0.4	:		Formula in Cell D7:	=Search Input!\$B\$2	
8	2	ASSIGN	ResD_ConfidenceLevel	0.4	:		Formula in Cell D8:	=Search Input!\$C\$2	
9	/	Processing SQS							
10	2	ASSIGN	ResXY_CLT1_SQS	2	:		Formula in Cell D10:	=Search Input!\$H\$2	
11	2	ASSIGN	ResUV_CLT1_SQS	4	:		Formula in Cell D11:	=Search Input!\$I\$2	
12	/	Fix the work orders of ResMN							
13	4	/Do not fix ResMN's work orders							
14	4	/Do not fix ResXY's work orders							
15	4	/Do not fix ResUV's work orders							
16	/								
17	/	Formula in Cell B13:	=IF(Search Input!\$J\$2=0,"/Do not fix ResMN's work orders", "ONENTRY ResMN_Idle ASSIGN N_Perform_Additional_Semaphore M_Complete.CurCount==5?1:0;")						
18	/	Formula in Cell B14:	=IF(Search Input!\$K\$2=0,"/Do not fix ResXY's work orders", "ONENTRY ResXY_Idle ASSIGN Y_Perform_Additional_Semaphore X_Complete.CurCount==5?1:0;")						
19	/	Formula in Cell B15:	=IF(Search Input!\$L\$2=0,"/Do not fix ResUV's work orders", "ONENTRY ResUV_Idle ASSIGN U_Perform_Additional_Semaphore V_Complete.CurCount==5?1:0;")						
20									

Figure 9.16 Dynamic Code Input for the initial decision variables in Figure 9.17

	A	B	C	D	E	F	G	H	I	J	K	L
1	ResB	ResC	ResD	ResM, ResMN	ResUV	ResXY	Beak at M-5	ResXY SQS	ResUV SQS	Fix ResMN Work Order	Fix ResXY Work Order	Fix ResUV Work Order
2	0.8	0.4	0.4	0.8	0.8	0.2	0	2	4	0	0	0
3												
4	0	0	0	0	0	0	0	2	3	0	0	0
5	0.2	0.2	0.2	0.2	0.2	0.2	1	3	4	1	1	1
6	0.4	0.4	0.4	0.4	0.4	0.4		4				
7	0.6	0.6	0.6	0.6	0.6	0.6						
8	0.8	0.8	0.8	0.8	0.8	0.8						
9	1	1	1	1	1	1						
10												

Figure 9.17 Initial decision variables for Dynamic Code in Figure 9.16

Showing an updated version for the dynamic code in Figure 9.16, Figure 9.18 presents the Dynamic Code Input sheet corresponding to the decision variable cells in Figure 9.19. Notice that many decision variables have changed from Figure 9.16 to Figure 9.18. In Figure 9.16 and Figure 9.18, the dynamic code in Rows 1 to 11 involves parameter manipulation, while the code in Rows 13 to 15 involves simulation code manipulation.

	A	B	C	D	E	F	G	H
1	/	Confidence Levels						
2	2	ASSIGN	ResB_ConfidenceLevel	0	:		Formula in Cell D2:	=Search Input!\$A\$2
3	2	ASSIGN	ResMN_ConfidenceLevel	0.6	:		Formula in Cell D3:	=Search Input!\$D\$2
4	2	ASSIGN	ResM_ConfidenceLevel	0.6	:		Formula in Cell D4:	=Search Input!\$D\$2
5	2	ASSIGN	ResXY_ConfidenceLevel	1	:		Formula in Cell D5:	=Search Input!\$F\$2
6	2	ASSIGN	ResUV_ConfidenceLevel	0.8	:		Formula in Cell D6:	=Search Input!\$E\$2
7	2	ASSIGN	ResC_ConfidenceLevel	0.2	:		Formula in Cell D7:	=Search Input!\$B\$2
8	2	ASSIGN	ResD_ConfidenceLevel	0.4	:		Formula in Cell D8:	=Search Input!\$C\$2
9	/	Processing SQS						
10	2	ASSIGN	ResXY_CLT1_SQS	3	:		Formula in Cell D10:	=Search Input!\$H\$2
11	2	ASSIGN	ResUV_CLT1_SQS	4	:		Formula in Cell D11:	=Search Input!\$I\$2
12	/	Fix the work orders of ResMN						
13	4	ONENTRY ResMN_Idle ASSIGN N_Perform_Additional_Semaphore M_Complete.CurCount==5?1:0;						
14	4	/Do not fix ResXY's work orders						
15	4	ONENTRY ResUV_Idle ASSIGN U_Perform_Additional_Semaphore V_Complete.CurCount==5?1:0;						
16	/							
17	/	Formula in Cell B13:	=IF(Search Input!\$J\$2=0,"/Do not fix ResMN's work orders", "ONENTRY ResMN_Idle ASSIGN N_Perform_Additional_Semaphore M_Complete.CurCount==5?1:0;")					
18	/	Formula in Cell B14:	=IF(Search Input!\$K\$2=0,"/Do not fix ResXY's work orders", "ONENTRY ResXY_Idle ASSIGN Y_Perform_Additional_Semaphore X_Complete.CurCount==5?1:0;")					
19	/	Formula in Cell B15:	=IF(Search Input!\$L\$2=0,"/Do not fix ResUV's work orders", "ONENTRY ResUV_Idle ASSIGN U_Perform_Additional_Semaphore V_Complete.CurCount==5?1:0;")					

Figure 9.18 Dynamic Code for the initial decision variables in Figure 9.19

	A	B	C	D	E	F	G	H	I	J	K	L
1	ResB	ResC	ResD	ResM, ResMN	ResUV	ResXY	Beak at M-5	ResXY SQS	ResUV SQS	Fix ResMN Work Order	Fix ResXY Work Order	Fix ResUV Work Order
2	0	0.2	0.4	0.6	0.8	1	1	3	4	1	0	1
3												
4	0	0	0	0	0	0	0	2	3	0	0	0
5	0.2	0.2	0.2	0.2	0.2	0.2	1	3	4	1	1	1
6	0.4	0.4	0.4	0.4	0.4	0.4		4				
7	0.6	0.6	0.6	0.6	0.6	0.6						
8	0.8	0.8	0.8	0.8	0.8	0.8						
9	1	1	1	1	1	1						
10												

Figure 9.19 Initial decision variables for Dynamic Code in Figure 9.18

Figure 9.20 presents the objective function for this example, encoded in the sub-tab Control Statements in the Additional Code tab. The objective function, measured in days, is 1200 minus expected project duration and expected sum of idle time in resources from the simulation. The expected project duration is presented in the code in Figure 9.20 by `bcltProjectDurationSQS6 BinCollector`, whereas the expected sum of idle time in resources is presented by `bcltProjectIdleTimeSQS6 BinCollector`. Note that the number of 1200 in the formula is arbitrarily selected by the user to ensure that the objective function

is always positive. The greater the resulting objective function value is, resulting from a particular set of values of the decision variables, the better the selected values for decision variables are. For the purpose of demonstration, the objective function used herein is very simple and easy to understand; it is stored in a variable called “ObjFunc,” shown in Figure 9.20.

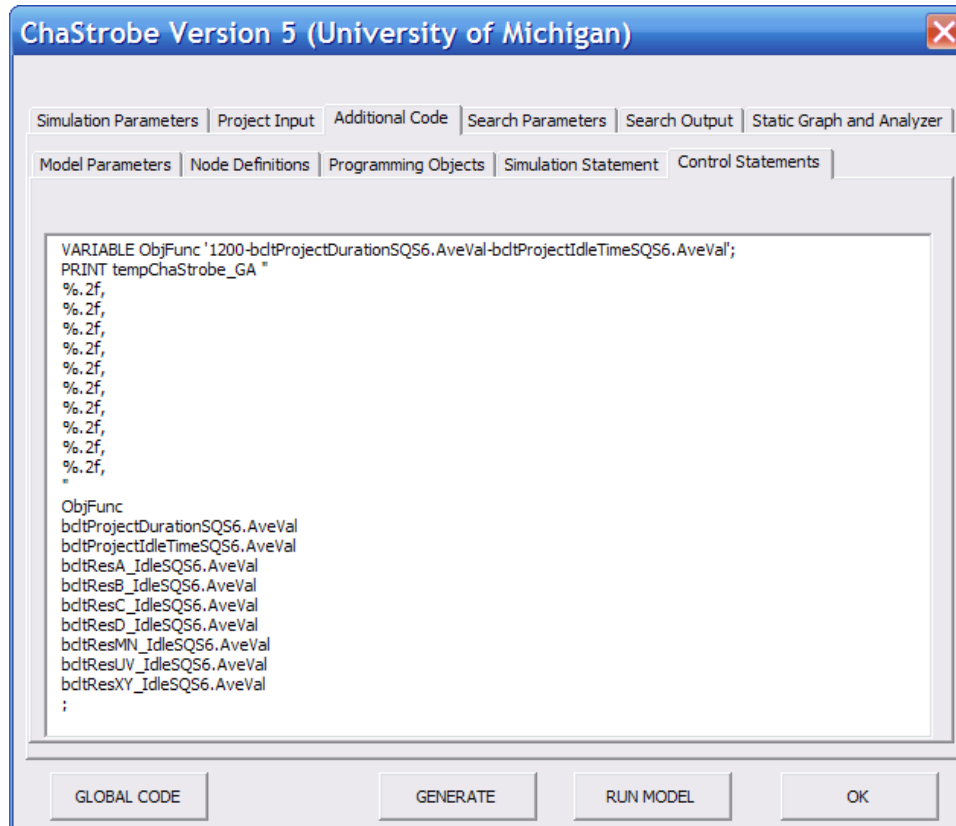


Figure 9.20 Objective function and user-specified additional output for Example 9.1

The value of the ObjFunc variable in Figure 9.20, calculating the objective function value, will be stored (printed) in tempChaStrobe_GA.txt at the end of the simulation run for each set of decision variables. As stated earlier, ChaStrobe always treats the first number in tempChaStrobe_GA.txt as an objective function value, which is very important for the calculation and optimization in GA. The rest of the numbers are

considered user-specified additional output, collected for the purposes of presentation only.

Figure 9.21 illustrates Search Parameters for Example 9.1. As shown in the figure, both options of Dynamic Coding and Dynamic Simulation Modeling are checked to modify simulation code and model. For this example, the genetic algorithm is used since the total number of the possible alternatives is tremendously large, 4,478,976 (from $6^6 \times 2^5 \times 3$). The number of populations and generations are set to 100 and 40, respectively. The probabilities of crossover and mutation are set to 0.6 and 0.1, respectively.

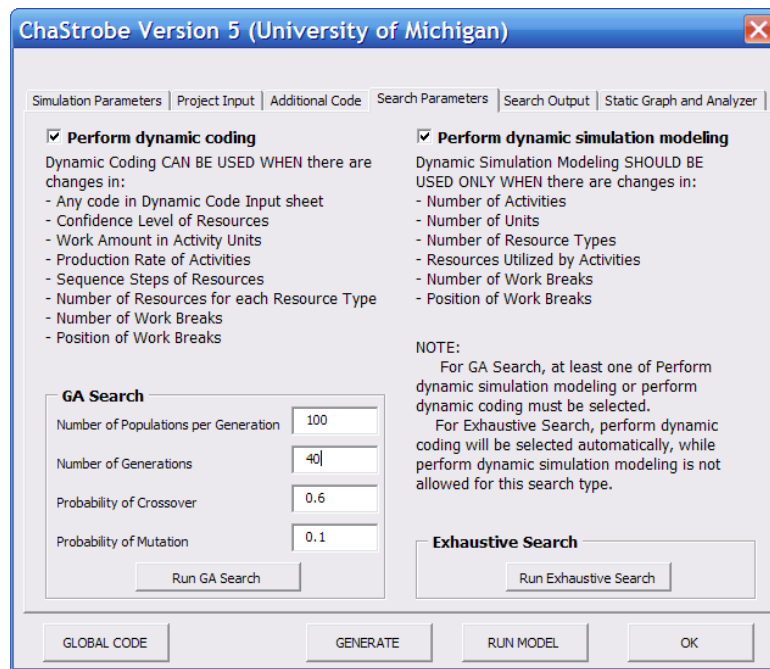


Figure 9.21 Search Parameters for Example 9.1

Figure 9.22 displays the results from the GA optimization. The data is sorted by the objective function values in Column M from maximum to minimum. As shown in the figure, there are three solutions returning the objective function value of 590 days. According to the formula for the objective function ($1200 - \text{bcltProjectDurationSQS6} - \text{bcltProjectIdleTimeSQS6}$), greater objective function value means lower project duration

and idle time in the project. The confidence levels from these three solutions are close, but not exactly the same. The results in Figure 9.22 show that the work break at the end of M5 (Column G) and the specified working sequence for ResMN (Column J) are mandatory in achieving an optimum result for this project. Optimum project duration should be about 600 days (Column N), and total idle time should be within a range of 5 to 30 days. Although not shown, it should be noted that the objective function values ranged from 102 to 590 days; project duration from 419 to 826; total idle time from 3 to 499 days.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	ResB	ResC	ResD	ResM, ResMN	ResUV	ResXY	Beak at M-5	ResXY SQS	ResUV SQS	Fix ResMN Work Order	Fix ResXY Work Order	Fix ResUV Work Order	Objective Value	Project Duration	Total Idle Time
1															
2	0.8	0.8	1	0.2	0.2	0.4	1	4	3	1	0	0	590.33	590.16	19.51
3	0.6	0.4	0.8	0.4	0.2	0.8	1	2	4	1	0	0	589.59	596.54	13.87
4	0.8	0.8	0.8	0.2	0.2	0.4	1	4	3	1	0	0	589.58	585.6	24.82
5	0.8	0.8	0.8	0.4	0.4	0.2	1	2	3	1	0	0	589.46	592.07	18.47
6	0.2	1	0.2	0.2	0	0.8	1	4	3	1	1	0	589.28	591.42	19.3
7	0.6	0.8	0.8	0.2	0.6	0.8	1	2	4	1	0	0	589.03	593.97	17
8	0.6	0.6	0.8	0.2	0.2	0.2	1	2	4	1	0	0	588.88	590.99	20.13
9	0.6	0.8	0.8	0.2	0	0.2	1	4	3	1	0	0	588.58	583.63	27.79
10	0.6	0.8	0.8	0.2	0.2	0.2	1	2	4	1	0	0	588.23	585.99	25.78
11	0.8	0.8	1	0.2	0.2	0.4	1	4	4	1	0	0	587.97	591.69	20.34
12	0.6	0.8	0.8	0.6	0.2	0.6	1	4	3	1	0	0	587.87	606.52	5.61

Figure 9.22 GA Results with three best solutions providing an objective value of 590

9.7 Summary

This chapter presents optimization in ChaStrobe. The optimization components of ChaStrobe are explained in detail with examples, focusing on optimization-related inputs. These inputs are Search Input, Dynamic Code Input, Additional Code, and Search Parameters. Search Input collects the decision variables and their domain

values for the optimization. ChaStrobe's Search Input allows users to optimize problems with any number of decision variables and their domain values.

After the input on the Search Input sheet is established, it is used as a reference for Dynamic Code Input. Dynamic Code Input is the simulation code that changes to reflect the current values of the decision variables, which are altered by ChaStrobe's search methods in order to optimize problems. The code or part of the code in Dynamic Code Input references the decision variable cells (the current decision variables) on the Search Input sheet. The code on the Dynamic Code Input sheet is updated automatically every time ChaStrobe selects new values for decision variables. After the update, ChaStrobe creates simulation based on Simulation Parameters, Project Input (including Dynamic Code Input), and Additional Code.

Additional Code tab stores the simulation code that remains the same throughout the process of optimization; the code in Additional Code tab is called "additional consistent code." Normally, user-specified objective function is encoded in the Additional Code tab. Nevertheless, if the equation for the objective function is subjected to change based on the current values of the decision variables. The equation must be encoded in Dynamic Code Input and references to the decision variable cells on the Search Input sheet. The objective function specified by users to optimize problems can be:

- Maximizing project profit (e.g., contract prize minus direct cost minus indirect cost including cost associated with idle time, penalty cost from delaying the project, etc.)

- Minimizing project duration and idle time (e.g., 1200 minus project duration minus the sum of idle time)
- Maximizing resource utilization (e.g., project duration minus the sum of idle time)

To optimize problems with the user-specified objective function, ChaStrobe is able to modify simulation code and model in three different levels. The three levels of simulation code and model manipulation are:

- 1) Parameter manipulation, altering values of variables in simulation code, such as changing resources' confidence levels
- 2) Simulation code manipulation, adding and altering simulation code, such as stipulating additional precedence constraints
- 3) Simulation model manipulation, modeling different scenarios to optimize the problem, such as modeling a shared resource with two or more dedicated resources

Employing the simulation model and code manipulation to modify the simulation, examples of decision variables and their suggested manipulation types are:

- Activities' work amounts, using parameter manipulation
- Resources' confidence levels, using parameter manipulation
- The number of resources for each type, using parameter manipulation
- Resources' processing sequence steps, using parameter manipulation
- Additional constraints, using code manipulation
- Resources' working sequence, using code manipulation
- Whether to use a work break, using model manipulation

- The number of work breaks, using model manipulation
- The positions of work breaks, using model manipulation
- Whether to use dedicated resources or shared resources, using model manipulation

ChaStrobe offers two search methods to find an optimum schedule for repetitive projects. Two search methods are:

- 1) The Exhaustive Search, which tests all possible combinations of decision variables in order to find the best solution.
- 2) The Genetic Algorithm, which imitates the mechanism of natural selection and selectively tests combinations of decision variables based on their objective function values.

With the three types of modification and two search methods, users can use the optimization in ChaStrobe to optimize repetitive projects effectively and intelligently. An example of a repetitive project with resource-sharing activities (Example 9.1) is demonstrated how to setup optimization inputs in ChaStrobe and optimize the project. In the example, there are 12 decision variables with a different number of domain values, resulting in 4,478,976 possibilities. These decision variables are resources' confidence levels, position and the number of work breaks, resources' sequence steps, and additional constraints of working sequences for the resource-sharing activities. Due to the large number of alternatives, GA is employed to optimize this example. The objective function is maximizing the value of 1200 days minus project duration and idle time. Data from optimization shows that the objective values could range from 102 to 590 days; project duration from 419 to 826; and idle time from 3 to 499 days. In the end, there are three

optimum solutions providing the objective function value of 590 days, compared to 564 days from a non-optimized solution, shown in Example 8.4.

As presented in this chapter, the concepts of the sequence step algorithm, the idea of using confidence levels and work breaks to relax the continuity constraints, and the suggestion of scheduling resource-sharing activities can be cooperatively employed to optimize repetitive projects. The means of optimization in ChaStrobe and the results attests the effectiveness, usability, flexibility, and extensibility of the ChaStrobe application.

CHAPTER 10

CONCLUSIONS AND RECOMMENDATIONS

10.1 Summary

This dissertation presents the Sequence Step Algorithm (SQS-AL), a simulation-based scheduling algorithm for repetitive construction projects with deterministic and/or probabilistic activity durations. SQS-AL is capable of scheduling repetitive projects under variability and uncertainty while maintaining continuous resource utilization. To solve scheduling problems of repetitive projects, SQS-AL consists of two nested loops: 1) replication loop to collect crew idle time (inner loop) and 2) sequence step loop to assign crew lead time for each resource serving an activity in the current sequence step (outer loop). Using the concepts of SQS-AL, risk analysis can be performed, and project duration and project cost can be optimized.

The development and testing of SQS-AL was performed using the Stroboscope, discrete-event simulation system. Designed in the system, two simulation model templates are proposed: 1) Work Flow Template and 2) Resource Flow Template. These templates simplify and standardize the development of simulation model for repetitive projects within Stroboscope, which otherwise could be complicated and involve many simulation elements representing activities and resources. Using these templates,

simulation models are easy to create and understand, systematically organized, and effective. The work flow template models repetitive work and the flow of work in activities, whereas the resource flow template models dynamic resource utilization and allocation. Activities' precedence constraints are modeled in the work flow template using semaphore, a conditional statement in Stroboscope. Resource availability constraints are modeled in the resource flow template and through links to the work flow template. Resource continuity constraints are modeled in and satisfied by SQS-AL and the data collected from the resource flow templates.

The proposed SQS-AL and the two simulation model templates have been implemented in Visio on top of the Stroboscope Graphical User Interface (Stroboscope GUI) to form a Visio add-on called "ChaStrobe." ChaStrobe has been programmed in Visual Basic for Applications (VBA) within Visio. ChaStrobe creates simulation model and code, for a repetitive project, conforming to SQS-AL. The process of creating the simulation and solving the problem in ChaStrobe takes only minutes depending on the number of repetitive activities and sequence steps. Without ChaStrobe, this process could be lengthy and laborious. Moreover, after the graphical simulation elements and code are created, users can customize them within the Stroboscope GUI to fit additional requirements prior to submitting the model to the Stroboscope Simulation Engine. Currently, the integration of SQS-AL, ChaStrobe, and Stroboscope is the most powerful tool for scheduling repetitive projects under uncertainty. The usability, flexibility, and extensibility of the system establish a vigorous foundation for continuous development in the scheduling field for both repetitive and non-repetitive projects with deterministic and probabilistic activity durations.

SQS-AL minimizes idle time and promotes continuous resource utilization by controlling the arrival dates of resources to the site. As expected, project duration is sometimes extended due to the imposed resource continuity constraints. SQS-AL ensures that project duration is minimized for a given probability of achieving resource continuity. Clearly, there is a tradeoff between imposing and relaxing the continuity constraints which must be carefully examined and balanced. To mitigate the prolonged project duration while maintaining continuity constraints, a lower confidence level of resources can be employed. With a lower confidence level, SQS-AL will relax the continuity constraints, resulting in shorter project duration, but greater expected idle time.

The prudent selection of confidence levels is effective enough to optimize a problem. However, there is a need to employ another form of relaxation of continuity constraints, which is the introduction of deliberate “work breaks” to mitigate the prolonged project duration resulting from achieving resource continuity. A work break is the deliberate interruption of the work of a resource during which the resource is directed to leave the site and for how long. Thus, work breaks are essentially predetermined idle time in activity schedule, but not in a resource schedule, since the resource is unemployed and not getting paid. Applying a work break in a resource’s schedule splits the resource’s continuity constraints; as a result, it allows the sub-activities served by the resource to start sooner. In turn, the prolonged project duration from SQS-AL should become shorter. Nevertheless, to effectively reduce the project duration by using work breaks, the activity served by the resource whose continuity is spitted must have the following characteristics:

- The activity must be on the controlling sequence.

- The activity must have a converging relationship with its direct predecessor on the same controlling sequence.
- The activity must have a diverging relationship with its direct successor on the same controlling sequence.

The joint use of proper confidence levels and work breaks immensely improve resource utilization and project duration. To improve project cost, users can set both confidence levels and work breaks as decision variables and establish an objective function in terms of project cost in ChaStrobe's optimization to optimize the problem.

For optimization, ChaStrobe offers three levels of simulation model and code modification: 1) parameters, 2) simulation code, and 3) simulation model. Using these three levels of simulation modification, ChaStrobe automatically modifies the simulation (parameters, code, and model) and employs either the exhaustive search or the genetic algorithm to optimize a problem.

10.2 Contributions

The development of SQS-AL and ChaStrobe has allowed for the first time to solve the problems of scheduling repetitive projects with probabilistic activity durations. Both SQS-AL and ChaStrobe have broadened the field of repetitive project scheduling in the construction industry today in several ways:

- SQS-AL is a generalized algorithm that can be implemented in most simulation applications or even Excel spreadsheet to solve either deterministic or probabilistic scheduling problems of repetitive projects. Since SQS-AL is a generalized algorithm, it is applicable to most construction projects.

Moreover, SQS-AL does not rely on particular tools, as other techniques limit themselves to, for example, linear or dynamic programming.

- SQS-AL is a simple and effective algorithm requiring two nested loops, replication loop and sequence step loop, and the data collection of resources' idle time in order to calculate resources' arrival dates. (The data collection of idle time is required only when solving probabilistic problems.)
- SQS-AL can maintain continuous resource utilization and control idle time via resources' confidence levels. As a result, idle time in resources and interruptions in activities are minimized and managed.
- SQS-AL's concept of confidence levels allows a repetitive project scheduling problem to be optimized by altering confidence levels for each resource. The results from the alteration can be analyzed further to evaluate the benefit of satisfying resource continuity constraints at a different confidence level for each resource.
- SQS-AL's proposed work flow template can represent the realistic nature of repetitive construction activities having variability in production rates and varying work amounts in each unit.
- SQS-AL's proposed resource flow template can represent the realistic and dynamic nature of resources in repetitive activities by modeling a separated resource flow sub-network for each resource, from the work flow sub-network. The separated resource flow sub-network allows the model to realistically simulate repetitive tasks and dynamically serve more than one activity.

- The concept of introducing deliberate work breaks to shorten the prolonged project duration (due to resource continuity constraints) while maintaining continuous resource utilization and controlling idle time and interruptions is original. The determination of which activities to test for work breaks using SQS-AL's crew lead time and RSM's controlling sequence is also new.
- SQS-AL and the two proposed simulation model templates can model and solve a scheduling problem of repetitive projects having activities sharing resource(s). Significant insight has been provided into the problem of scheduling resource-scheduling activities to allow for proper exploration and analysis in practice.

For the first time, the problem of scheduling repetitive construction projects is addressed and analyzed at this extremely complex level. Variability in activity durations, differences in work amounts of activities in each unit, maintenance of continuous resource utilization, application of work breaks, and existence of resource-sharing activities contribute to the high complexity. For the construction industry, SQS-AL and ChaStrobe are now available to solve a very complicated scheduling problem. Using SQS-AL and ChaStrobe, activities are effectively scheduled with less interruption, and resources are efficiently utilized with less idle time. Risk analysis and optimization can be performed to maximize project profitability.

10.3 Recommendations

The development of this research investigate has been comprehensive. Even so, the impact of weather and other global factors on the repetitive projects is one of the

subjects that needs to be studied in order to more realistically model repetitive activities and their resources, and more effectively schedule repetitive projects, even further.

To maintain continuous resource utilization, repetitive activities are delayed. Delaying activities, resulting in longer project duration, does not change activities' durations. In this investigation, the activity durations are assumed to be given and remain constant. This is a general assumption made by most, if not all, repetitive project scheduling techniques, including SQS-AL. However, delaying a weather-sensitive activity from one season to another could result in a longer or shorter activity duration. Decreasing and increasing activity durations results in new idle times. Therefore, the re-calculation of resources' arrival dates is required. This means an additional loop for re-calculating resources' arrival dates is necessary due to the new activities' durations, effected by the weather.

To account for the impact of weather on repetitive activities' durations or production rates, an additional weather loop should be implemented between the replication loop and sequence step loop. After the certain number of simulation replications is executed and resources in that sequence step are scheduled, activity durations must be updated. If there is a slight change or no change in activity duration, there is no need to re-calculate the idle time for the resources, which have just been delayed. On the other hand, if activity duration changes, idle time of the resource must be recollected from the simulation to check whether it is in the acceptable range of user-specified confidence levels (upper bound and lower bound). Thus, the replication loop for this particular sequence step must be repeated more than once to re-determine the new crew idle time.

Scheduling repetitive projects with weather-sensitive activities is an interesting and challenging topic, especially when activities' durations are probabilistic. As discussed earlier, the suggested idea of updating idle time and re-determining resources' arrival dates can be implemented in SQS-AL by adding weather-impact loops between replication loop and sequence step loop. Considering the impact of weather on activity durations makes the derived schedule more realistic and accurate. Moreover, it could reveal a hidden benefit or detriment of scheduling repetitive activities differently, especially in a region where severe weather is common.

APPENDICES

APPENDIX A

DETERMINATION OF THE CONTROLLING SEQUENCE

This section demonstrates how to determine the controlling sequence and controlling sequence activities in repetitive projects under variability and uncertainty. According to the Sequence Step Algorithm (SQS-AL), activities are postponed from their early start date in order to satisfy continuity constraints. Under variability and uncertainty, the postponement period for each activity is not determined by a fixed sum of its idle time, but by the cumulative distribution of the sum of idle time and a user-specified confidence level for the activity. Accounting for idle time, variability, and uncertainty, the postponement period incurs free floats between activities on control points in the project. Consequently, a control point between two directly dependent activities is not exactly vertical from the predecessor's finish date to the successor's start date; there is usually a free float between the activities. Therefore, there is a need for modifying the determination of control points and controlling sequences suggested by Harris and Ioannou (1998).

A control point between two repetitive activities is the precedence constraint that determines the start date of a succeeding repetitive activity under resource continuity constraints. In deterministic scheduling problems of repetitive projects, the predecessor's finish date on a control point is exactly (vertically) the same as the successor's start date

on the same control point; there is no free float between two activities on a control point. Thus, schedulers can easily identify the control point between the two activities. After all the control points between activities in the project are identified, schedulers simply navigate through the control points from the last activity's finish date to the first activity's start date to determine the controlling sequence for the project. This straightforward process of identifying control points and navigating through the points in order to determine the controlling sequence is applicable only to repetitive projects with deterministic activity durations.

In probabilistic scheduling problems of repetitive projects, however, identifying control points is not as simple as in deterministic problems because floats between activities with probabilistic durations on the controlling sequence usually exist and vary. Under variability and uncertainty, activity durations vary, resulting in different idle times from one simulation replication to another. Therefore, activities are scheduled beyond their start dates in the RSM schedule to account for such variability and uncertainty. For example, SQS-AL schedules activities by their crew lead times, which are determined from collected crew idle times and user-specified confidence levels. The determined crew lead times from SQS-AL result in near-continuous resource utilization and provide sufficient floats between activities to prevent interruptions. These crew lead times result in floats at a control point between two activities.

A.1 Determination of Control Points and the Controlling Sequence

The determination of control points and the controlling sequence(s) addressed in this section is applicable to both deterministic and probabilistic problems. An example of a repetitive project consisting of 4 units requiring 7 activities in Figure A.1 is used to

present the means of determining control points and the controlling sequences. Two tables are given, one for deterministic activity durations in Table A.1 and another one for probabilistic activity durations in Table A.2.

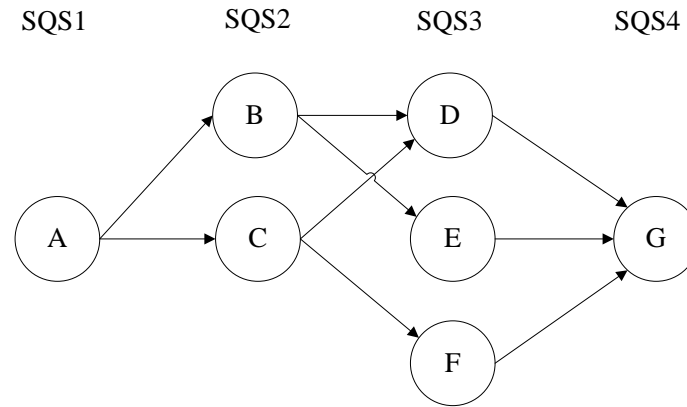


Figure A.1 Single unit precedence diagram for Examples A.1 and A.2

To determine control points and the controlling sequence in repetitive projects, the following steps are required:

Step 1: Schedule activities using RSM for deterministic problems or SQS-AL for probabilistic problems to obtain start and finish dates of activities in each unit. For probabilistic problems, the state and finish dates from each replication from processing the final sequence step are collected.

Step 2: Determine all possible paths in the project. For example in Figure A.1, there are 4 possible paths: A-B-D-G, A-B-E-G, A-C-D-G, and A-C-F-G. Each path is called an “Activity Path.”

Step 3: Identify the minimum lags between any two directly dependent activities and the repetitive units with the minimum lags. The unit(s) resulting in the minimum lag between two activities is the control point(s) between the activities. The total number of the minimum lags that must be determined in a

repetitive project is equal to the number of precedence constraints in a single unit precedence diagram. In Figure A.1, there are 9 precedence constraints (links); as a result, 9 minimum lags (one each for any two directly dependent activities) must be determined: A-B, A-C, B-D, B-E, C-D, C-F, D-G, E-G, and F-G. Moreover, the repetitive units in which the minimum lags locate must also be identified.

Step 4: Calculate the sum of the determined minimum lags for each activity path. This sum of minimum lags for each activity path is called “*Path Idle Time*” (PIT). For example, in Figure A.2.b, PITs for Activity Paths A-B-D-G, A-B-E-G, A-C-D-G, and A-C-F-G are 2, 8, 0, 0 days, respectively.

Step 5: Determine the activity path(s) with the minimum PIT and identify the units with the minimum lags on the path(s). The activity path with the minimum PIT is called the “*Controlling Activity Path*” for the project. In Example A.1, shown in Figure A.2, there are two controlling activity paths: A-C-D-G and A-C-F-G with zero-day PIT.

Step 6: Navigate through the minimum-lag units (control points) on the controlling activity path from project finish to start. The activities on this controlling activity path connected by the control points are controlling sequence activities. In Example A.1, controlling sequence activities are A1 to A4, B2 to B3, D1 to D4, F1 to F2, and G2 to G4.

For deterministic problems, Steps 1 to 6 are performed only once to determine control points and the controlling sequences for the project, since activity durations are deterministic. On the other hand, for probabilistic problems, Steps 2 to 6 must be

repeated for each replication to determine the probability that each activity may be on the controlling sequence, given that the start and finish dates from each replication are already collected from Step 1 using SQS-AL. Therefore, additional steps for probabilistic problems are:

Step 7: Repeat Steps 2 to 6 for the number of user-specified replications and record the controlling sequence activities for each replication from processing the finalized sequence step.

Step 8: Calculate the number of times an activity is a controlling sequence activities. This is an estimate of the probability of being on the controlling sequence.

The next two sections demonstrate how to determine control points and controlling sequences for deterministic problems and probabilistic problems.

A.2 Example A.1 Repetitive project with deterministic activity durations

Table A.1 displays constant production rate and work amounts for activities in each unit. As a result of using a constant production rate for each activity, activity durations are deterministic. The duration for each activity in each unit is the ratio between the work amount in a particular unit and its production rate.

Activity	Resource Production Rate	Quantity of Work in Repetitive Units			
		1	2	3	4
A	10	100	250	150	200
B	20	150	100	200	150
C	15	200	150	50	200
D	15	150	200	100	150
E	25	100	150	50	100
F	15	150	250	50	100
G	20	50	200	50	150

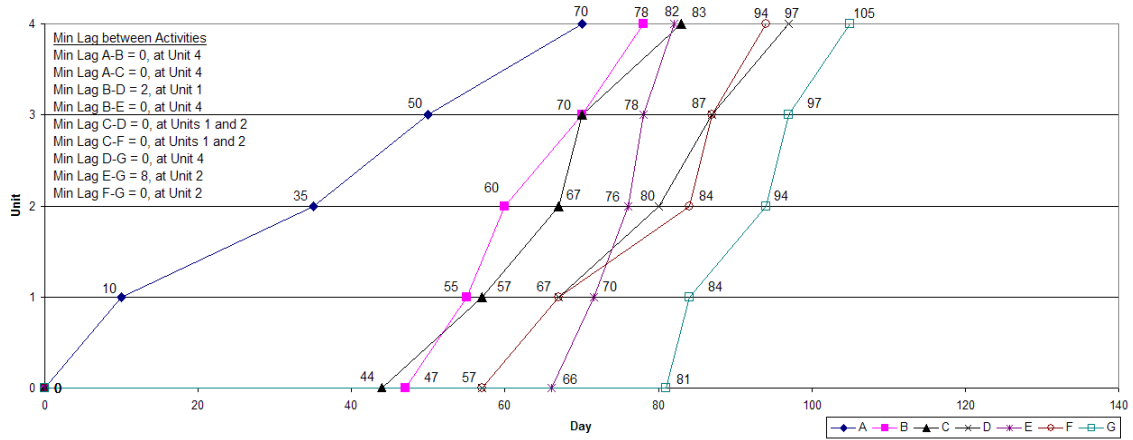
Table A.1 Resource production rate and quantity of work for Example A.1

After the final schedule is derived, the minimum lags between two directly dependent activities are determined, as shown in Figure A.2.a in the top-left corner. The lag between activities is the free float derived from the difference between the successor's start date (e.g., Activity B1's start date is 44) and the predecessor's finish date (e.g., Activity A1's finish date is 10) in the same unit. Therefore, the lag between Activities A1 and B1 is 34 days. The minimum lag between Activities A and B in Figure A.2 is zero in Unit 5 (B5's start date is 70 days, and A5's finish date is 70 days.)

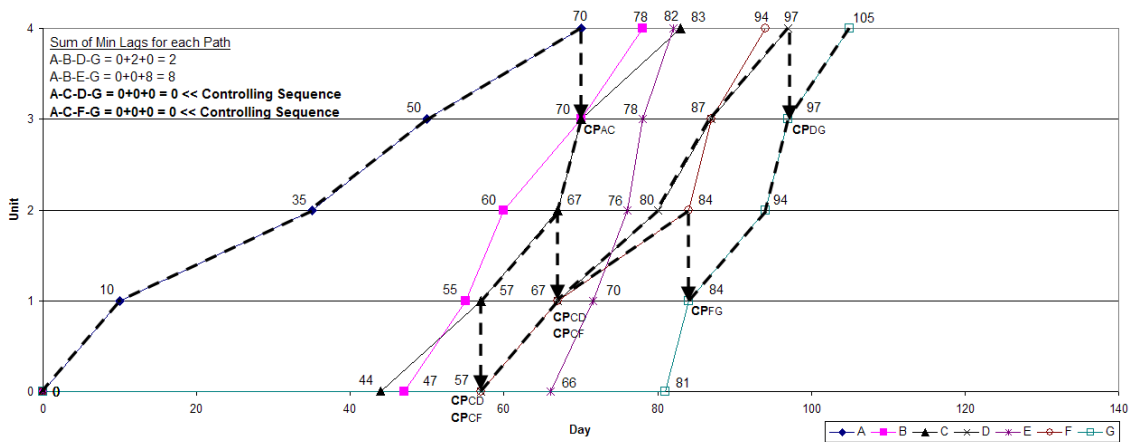
Note that it is possible for two directly dependent activities to have more than one unit with the minimum lag. This is valid for both deterministic and probabilistic problems. For example, in Figure A.2.a, Activities C and D have two units providing the minimum lag of zero, which are Units 1 and 2.

After the minimum lags are determined, path idle times (PIT) for each activity path are calculated as shown in Figure A.2.b in the top-left corner. Activity paths with the minimum PIT, the controlling activity paths, in Figure A.2.b are A-C-D-G and A-C-F-G, giving the minimum lag of zero. Therefore, there are two controlling sequences in this project.

It is important to note that one controlling activity path may consist of one or more controlling sequences. Moreover, the minimum PIT for deterministic and probabilistic problems can be zero or non-zero, depending on scheduling methods. If there is no additional buffer assigned between activities, the minimum PIT for deterministic problems is zero, as shown in Figure A.2 using RSM to schedule the project. Nevertheless, the minimum PIT for deterministic problems is a constant, while it is a variable for probabilistic problems.



(a) Final schedule for Example A.1 with deterministic durations



(b) Controlling sequences for Example A.1 with deterministic durations

Figure A.2 Control points and controlling sequences for Example A.1 with deterministic activity durations

The next step is to determine the controlling sequence activities on the identified controlling activity path. To do so, schedulers navigate through activities on the controlling activity path via the specified minimum lags from project finish to project start. For the example in Figure A.2.b, activities on the Controlling Activity Path A-C-D-G are Activities A1 to A4, C3 to C2, D1 to D4, and G4. The bold dashed lines in Figure A.2.b indicate the controlling sequence activities.

A.3 Example A.2 Repetitive project with probabilistic activity durations

Table A.2 displays activity productivities and work amounts in each unit using a normal distribution. As a result of using a normal distribution for productivity for each activity, activity durations are probabilistic. The duration for each activity in each unit is the ratio between the work amount in a particular unit and its probabilistic productivity, which is randomly generated according to a given distribution.

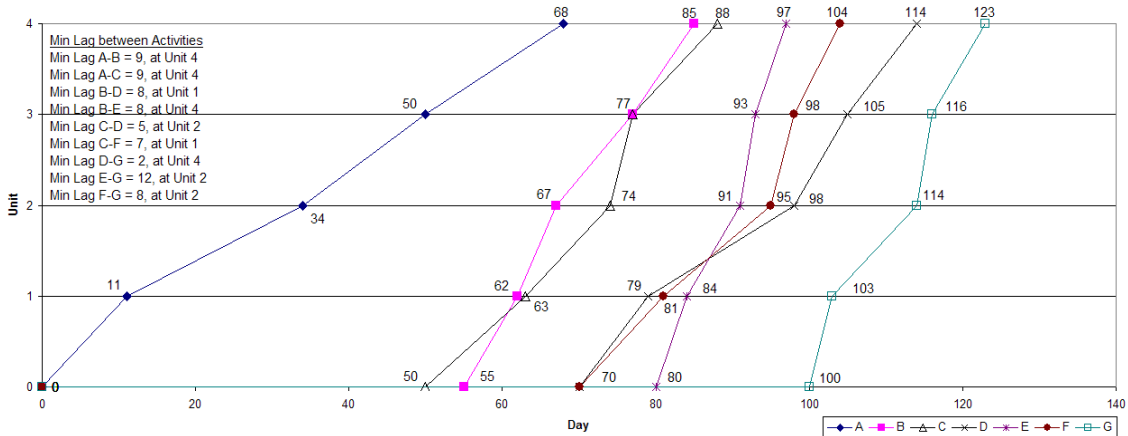
Activity	Resource Production Rate	Quantity of Work in Repetitive Units			
		1	2	3	4
A	Normal[10,1]	100	250	150	200
B	Normal[20,2]	150	100	200	150
C	Normal[15,1.5]	200	150	50	200
D	Normal[15,1.5]	150	200	100	150
E	Normal[25,2.5]	100	150	50	100
F	Normal[15,1.5]	150	250	50	100
G	Normal[20,2.0]	50	200	50	150

Table A.2 Probabilistic productivity and work amounts for Example A.2

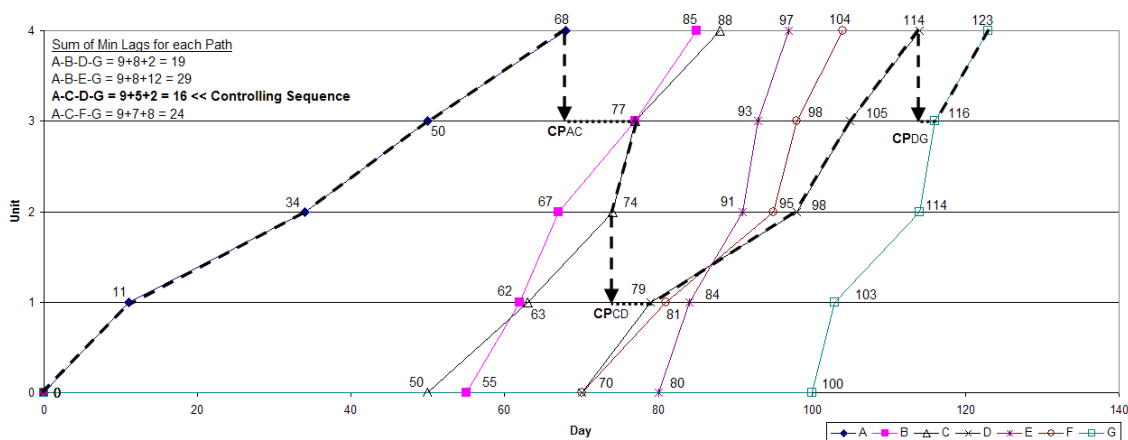
Assuming that SQS-AL has been performed and that the CLTs for all resources have been determined, Figures A.3 to A.7 are five schedules resulting from 5 replications. To determine the controlling sequences for Example A.2 with probabilistic durations, the controlling sequence activities for each of these replications must be determined, as shown in the previous section. Then, the probability of activities being on the controlling sequence is the ratio between the number of times an activity is on a controlling sequence and the number of replications.

Figures A.3 to A.7 are five production diagrams (schedules) derived from SQS-AL using an 80% confidence level at the end of SQS-AL. As shown in the figures, activities have been postponed from their earliest start date under continuity constraints

due to the cumulative distribution of idle time and the user-specified confidence level of 80%. Figure A.3 is the production diagram from the first replication.



(a) Final schedule for Example A.2 from the 1st replication



(b) Controlling sequence in Example A.2 from the 1st replication

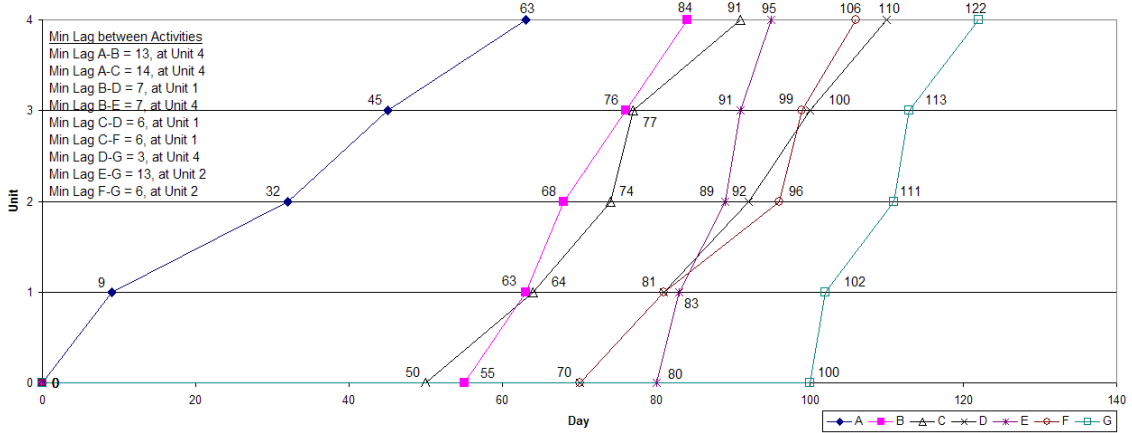
Figure A.3 Control points and controlling sequence for Example A.2 Replication 1 with probabilistic activity durations

As shown in the top-left corner of Figure A.3.a, the minimum lags between any two directly dependent activities are greater than zero; as a result, the minimum PIT is greater than zero. For the first replication, the activity path with the minimum PIT, the controlling activity path, is A-C-D-G, as show in Figure A.3.b, and activities on the controlling sequence are A1 to A4, C3, D2 to D4, and G5 with PIT of 16 days.

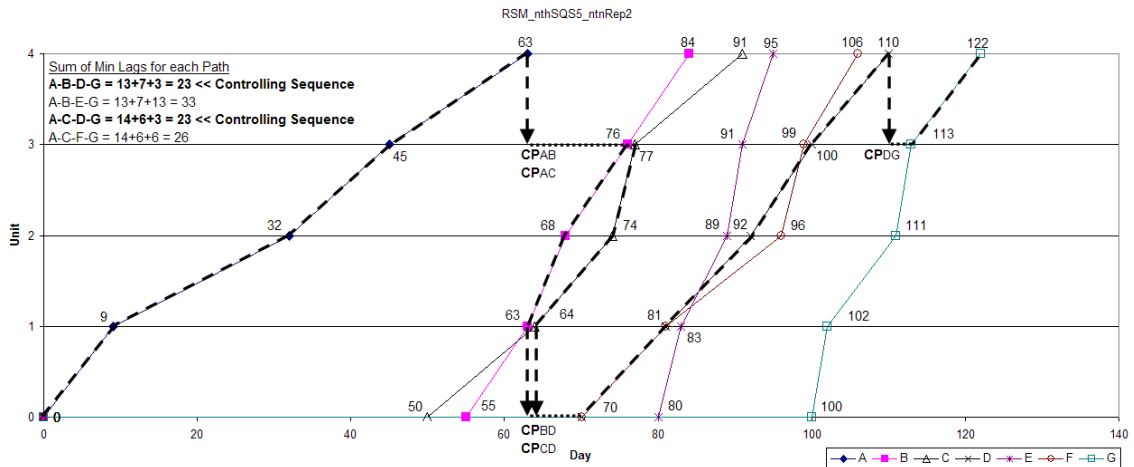
As can be seen, the controlling sequence in the first replication (Figure A.3) differs from the controlling sequence in the deterministic case (Figure A.2) and from the controlling sequence in the second replication (Figure A.4). Therefore, identifying the individual activity on the controlling sequence(s) for each replication is necessary, before calculating the probability of activities being on controlling sequences for the project. The probability of activities being on the controlling sequence(s) is the ratio between the number of times an activity is on a controlling sequence and the total number of replications.

Figure A.4 is the production diagram from the second replication, using SQS-AL with an 80% confidence level. Figure A.4.b shows there are two activity paths resulting in a minimum PIT of 23 days, A-B-D-G and A-C-D-G. For this replication, the two controlling activity path result in two controlling sequences: 1) A1 to A4, B3 to B2, D1 to D4, and G4, and 2) A1 to A4, C3 to C2, D1 to D4 and G4.

It is interesting to note that changes in different activities on the controlling sequences might have a different impact on the controlling sequences for the project. For example, in Figure A.4, Activities A1 to A4, D1 to D4, and G4 are on both controlling sequences, while B2 to B3 and C2 to C3 are on different controlling sequences. Therefore, changes in Activities A (A1 to A4), D (D1 to D4), and G4 may not result in a different controlling sequence. However, changes in Activities C2, C3, B3, and B4 are likely to change the controlling sequences for this replication, shown in Figure A.4.



(a) Final schedule for Example A.2 from the 2nd replication

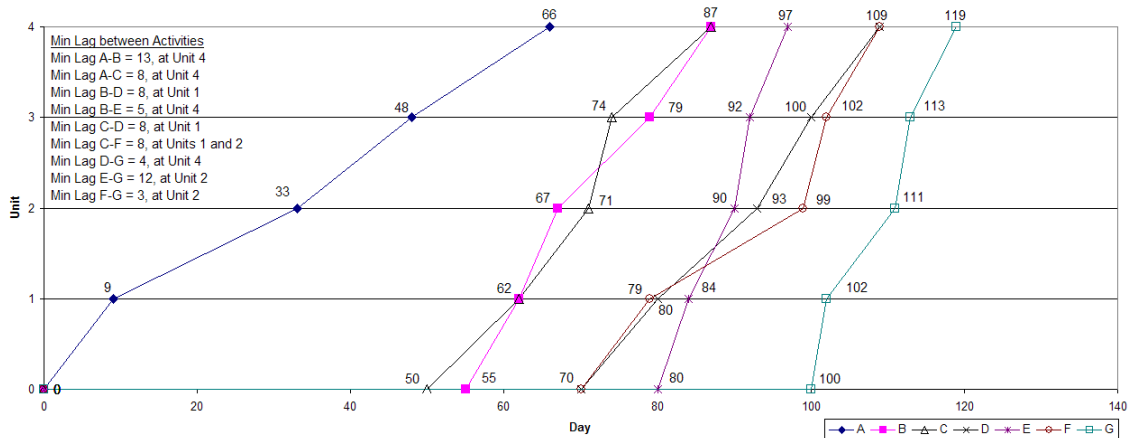


(b) Controlling sequences in Example A.2 from the 2nd replication

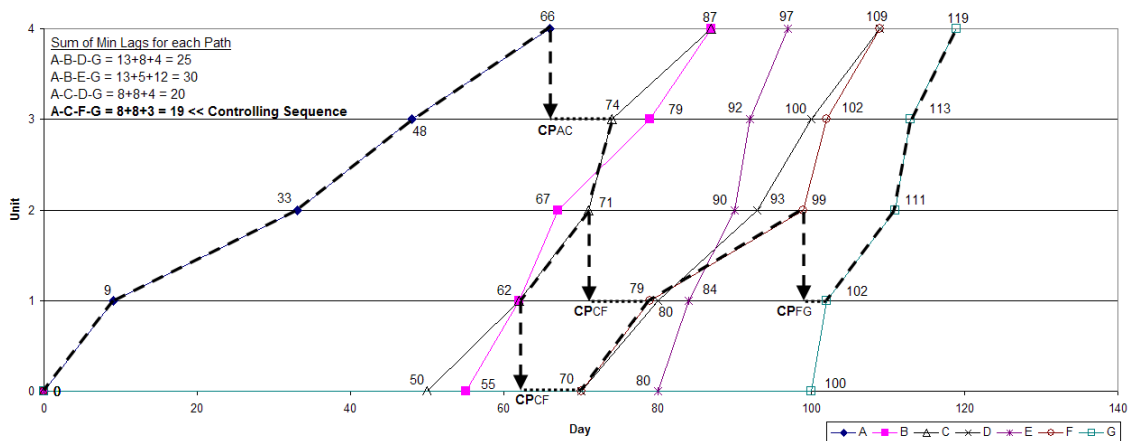
Figure A.4 Control points and controlling sequences for Example A.2 Replication 2 with probabilistic activity durations

Figure A.5 is the production diagram from the third replication. Figure A.5.b shows there is one activity path with the minimum PIT of 19 days, A-C-F-G. Nevertheless, since there are two positions providing the minimum lag between Activities C and F, which are in Units 1 and 2 (Figure A.5.a), there are two controlling sequences (Figure A.5.b) within Activity Path A-C-F-G, which are 1) Activities A1 to A4, C3, F2, and G2 to G4, and 2) Activities A1 to A4, C3 to C2, F1 to F2, and G2 to G4. Therefore,

changes in Activities B2 and F1 are likely to have more impact on the controlling sequences than other activities on the controlling sequences.



(a) Final schedule for Example A.2 from the 3rd replication

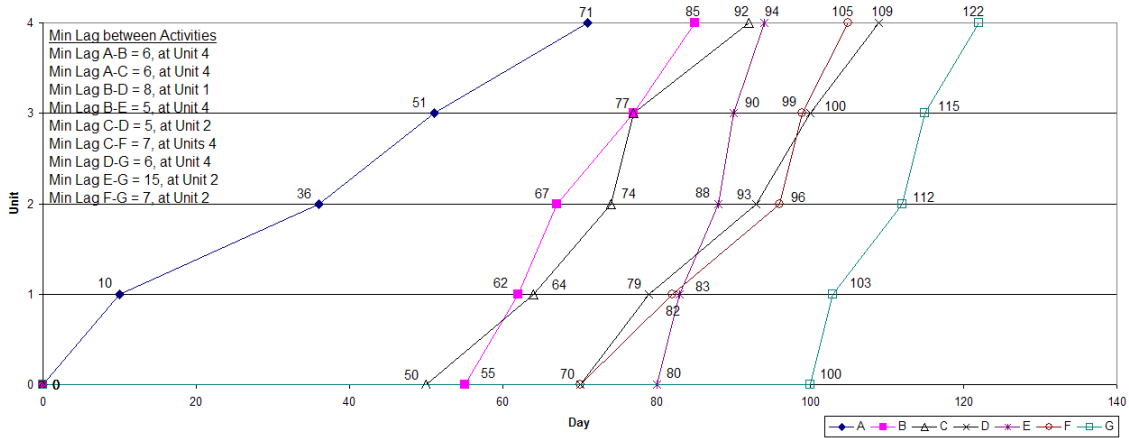


(b) Two controlling sequences in Example A.2 from the 3rd replication

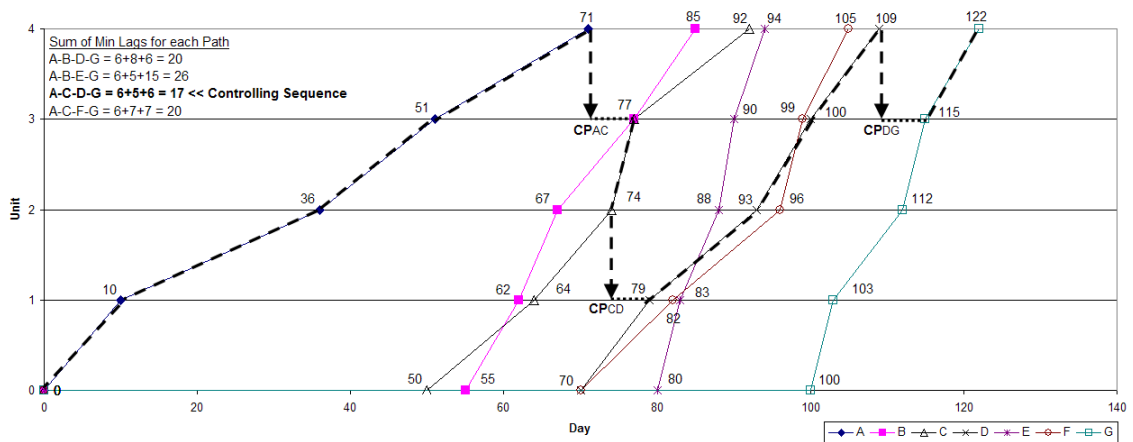
Figure A.5 Control points and controlling sequences for Example A.2 Replication 3 with probabilistic activity durations

Figure A.6 is the production diagram from the fourth replication. The controlling sequence from the fourth replication is the same as from the first replication; however, the minimum PITs from the two replications are different. The same controlling sequence from different replications may have a different sum of minimum lags (minimum PIT). Therefore, collecting the statistical data of PIT for each controlling activity path from different replications reveals the criticality and floats of activity paths. Moreover, the

statistical data of minimum PIT, such as its average value, shows the number of days additionally included in the project to prevent the idle time stemming from variability and uncertainty.



(a) A final schedule for Example A.2 from the 4th replication

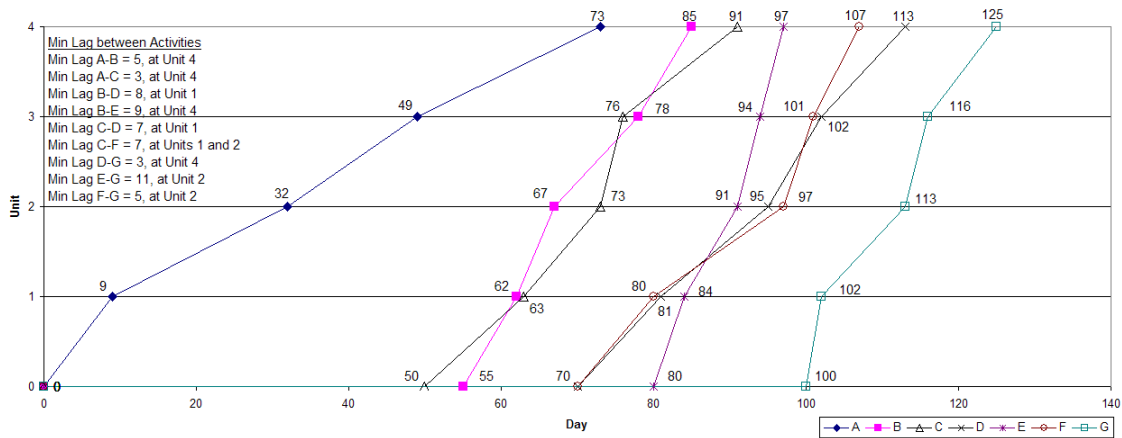


(b) The controlling sequence in Example A.2 from the 4th replication

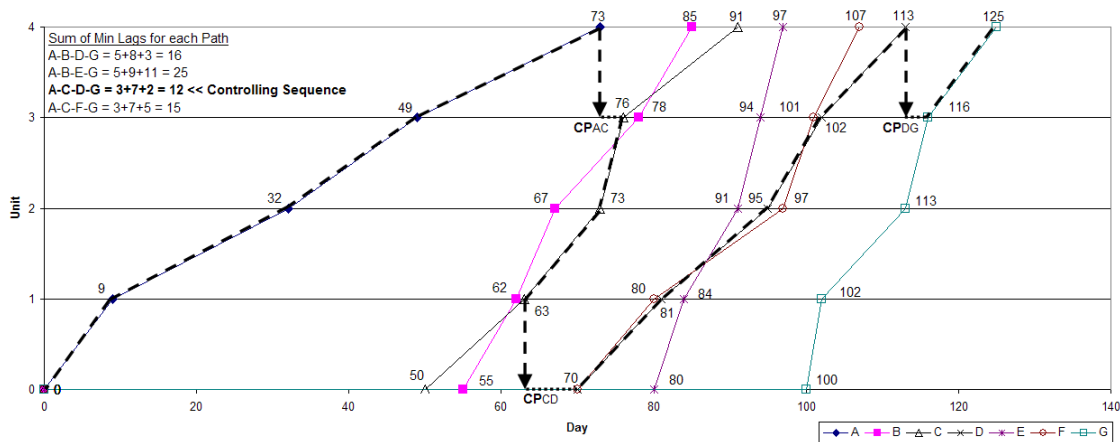
Figure A.6 Control points and controlling sequences for Example A.2 Replication 4 with probabilistic activity durations

Figure A.7 is the production diagram from the fifth replication. For this replication, the minimum PIT is only 12 days, without any interruption within each activity between its units; there is not unit idle time (UIT) in this replication. As shown in Figure A.7.a, the duration of Activity A in the 5th replication is relatively longer than in

any other previous replications. However, the time buffers (floats) between Activities A and B, and between A and C are able to prevent an interruption in Activities B4 and C4.



(a) Final schedule for Example A.2 from the 5th replication



(b) Controlling sequence for Example A.2 from the 5th replication

Figure A.7 Control points and controlling sequences for Example A.2 Replication 5 with probabilistic activity durations

From five replications, Tables A.3 to A.5 are constructed to display the statistical data of activities, activity paths, and also controlling sequences. Table A.3 shows the probability that an activity is on a controlling sequence. The “Y” in Table A.3 indicates that an activity is on a controlling sequence in the particular replication. Column “Hit” is the number of times an activity is on a controlling sequence, whereas Column “%Hit”

presents the probability of the activity being on a controlling sequence, based on the five replications (Figures A.3 to A.7).

Activity	On a Controlling Sequence in Replication (Y=Yes)					Hit	% Hit
	1	2	3	4	5		
A1	Y	Y	Y	Y	Y	5	100
A2	Y	Y	Y	Y	Y	5	100
A3	Y	Y	Y	Y	Y	5	100
A4	Y	Y	Y	Y	Y	5	100
B1						0	0
B2		Y				1	20
B3		Y				1	20
B4						0	0
C1						0	0
C2		Y	Y		Y	3	60
C3	Y	Y	Y	Y	Y	5	100
C4						0	0
D1		Y			Y	2	40
D2	Y	Y		Y	Y	4	80
D3	Y	Y		Y	Y	4	80
D4	Y	Y		Y	Y	4	80
E1						0	0
E2						0	0
E3						0	0
E4						0	0
F1			Y			1	20
F2			Y			1	20
F3						0	0
F4						0	0
G1						0	0
G2			Y			1	20
G3			Y			1	20
G4	Y	Y	Y	Y	Y	5	100

Table A.3 Probability of activities being on controlling sequences

For example, in Table A.3, Activity A has a 100% chance of being on a controlling sequence, while Activity E has a 0% chance. It is common that sub-activities (e.g., C1, C2, C3, and C4) of an activity (e.g., C) may have different probabilities of being on a controlling sequence. For example, Activities C1 and C4 have a 0% chance of

being on a controlling sequence, while Activities C2 and C3 have a 60% and 100% chance, respectively. Table A.3 also shows Activities A1, A2, A3, A4, C2, C3, D2, D3, D4, and G4 having greater than 50% probability of being on controlling sequences greater than 50%. Schedulers should pay attention to these activities since they influence project duration under the resource continuity constraints.

Tables A.4 and A.5 present statistical data of activity paths from five replications. Table A.5 shows the probability that an activity path is the controlling activity path for that particular replication. As shown in the table, Activity Path A-C-D-G has an 80% probability of being the controlling activity path for the project, based on the five replications. On the other hand, Activity Path A-B-E-G has a 0% chance.

Activity Path	On a Controlling Sequence in Replication (Y=Yes)					Hit	%Hit
	1	2	3	4	5		
A-B-D-G		Y				1	20
A-B-E-G							0
A-C-D-G	Y	Y		Y	Y	4	80
A-C-F-G			Y			1	20

Table A.4 Probability of activity path being the controlling activity path

Table A.5 presents statistical data of PIT, average (Avg), standard deviation (Std), maximum, and minimum values of each activity path. The bold numbers indicate activity paths providing the minimum PIT for a corresponding replication. Activity Path A-B-E-G, having an 80% probability of being a controlling sequence, has an average PIT value of 17.6 days. Approximately 18 days of buffers (floats) are included between activities in order to prevent interruptions under variability and uncertainty. If RSM was used to schedule this project with perfect hindsight, these 18 days would become zero without interruptions or idle time, because activity durations would have been determined and known before scheduling. This means that the difference in project duration between

assuming perfect information (idealistic or deterministic cases) and imperfect information (realistic or probabilistic cases) is 18 days. As the effectiveness in construction operation and project control increases, the project should be scheduled toward the idealistic cases (perfect information) from the realistic cases by thoughtfully lowering variability in activity durations.

In addition, in Table A.5, it is interesting to note that the maximum value of the PIT for Activity Path A-C-D-G is lower than the minimum value of the PIT for A-B-E-G. Accordingly, A-B-E-G has the least impact on the project duration. Nevertheless, it is important to realize activities on the same controlling activity path also have a different impact on the project duration.

Activity Path	PIT in each Replication (day)					Avg.	Std.	Max	Min
	1	2	3	4	5				
A-B-D-G	19	23	25	20	16	20.6	3.51	25	16
A-B-E-G	29	33	30	26	25	28.6	3.21	33	25
A-C-D-G	16	23	20	17	12	17.6	4.15	23	12
A-C-F-G	24	26	19	20	15	20.8	4.32	26	15

Table A.5 PIT of each activity path with its statistical data

A.4 Computerized Procedure of Determining Controlling Activity

As shown in the previous section, identifying the controlling sequence(s) after the controlling activity path(s) and its PIT is specified is relatively simple. Yet, it is tedious and cumbersome to manually calculate and determine the controlling sequence, especially when the number of units is large and activity durations are stochastic, resulting in different possible controlling sequences for each simulation replication. Thus, the process of determining controlling sequences should be computerized to reduce human effort and time. The following steps present a programmatic means of identifying activities on controlling sequences.

Step 1) Determine all activity paths in the project.

Step 2) Create a one-dimensional array for each activity path with a size twice as large as the total number of units (1 x 8 for Example A.2). Each member of the array represents possible control points on the activity start and finish dates, as shown in Table A.6.

Step 3) Assign a value of 1 to start date of the first activity in the first unit (e.g., SD1 of A) and the finish date of the last activity (e.g., FD4 of G).

Step 4) Determine the minimum lag between activities and calculate PIT for each activity path.

Step 5) Identify the controlling activity path, which is the activity path with the minimum PIT. For example, the activity path providing the minimum PIT in Figure A.7.b is A-C-D-G.

Step 6) Assign the value of 1 for the predecessors and the successors having minimum lags between activities on the controlling activity path in Step 5. As shown in Table A.6.a, each member in the array with value of 1 indicates a control point on the controlling activity path. The arrow indicates the predecessor and the successor of the control point. For example, since the minimum lag between A and C is 13 days from A4's finish date (FD4 of A) to C4's start date (SD4 of C), as indicated in Table A.6.a, FD4 of A and SD4 of C are the predecessor and the successor of the control point (i.e., CP_{AC}), respectively.

Step 7) Connect the navigating path in each activity by assigning a value of 1 from the first dimension to the last dimension having the value of 1 in the same

activity (in the same row in Table A.6.a). Table A.6.b shows the updated array of activities after connecting the navigating path.

	SD1	FD1	SD2	FD2	SD3	FD3	SD4	FD4
A	1*							1
B								
C		1					1	
D	1							1
E								
F								
G							1	1*

(a) Array indicating control points on the controlling activity path (minimum PIT)

	SD1	FD1	SD2	FD2	SD3	FD3	SD4	FD4
A	1*	1	1	1	1	1	1	1
B								
C		1	1	1	1	1	1	
D	1	1	1	1	1	1	1	1
E								
F								
G							1	1*

(b) Array indicating controlling sequence activities

Table A.6 Arrays determining activities on controlling sequences

Step 8) Determine whether an activity is on the controlling sequence by checking the values in the established arrays. If the values in the updated array for both SD and FD in the same unit of an activity are 1, the activity is a controlling sequence activity (being on the controlling sequence). For example, in Table A.6.b, since the values in the updated members for SD2 and FD2 of Activity C are 1, Activity C2 is a controlling sequence activity. In contrast, because the values of SD1 and FD1 for Activity C are 1 and 0 respectively, Activity C1 is not a controlling sequence activity (not being on the controlling sequence). After finishing Step 8, activities on the controlling sequence are A1 to A4, C2 to C3, D1 to D4, and G4.

APPENDIX B

EXTENSION OF SIMULATION MODEL TEMPLATES

This section examines variations of simulation model templates discussed in Chapter 5, Simulation Model Templates. Suggestions and examples of how to modify the work flow and resource flow templates to model a more complicated repetitive activity are given. Two modifications of simulation model templates shown in Appendix B are:

- Modeling different types of precedence relationships
- Modeling repetitive activities at an operational level

One of the main concerns in modifying the simulation model templates is how to keep the simulation model simple, organized, and functional. The modified templates should not change the functionality for each part of the suggested templates. For example, any modification relevant to how an activity is performed or simulated should be made in or replace the Act_Perform Combi in the work flow template. Another example is that any modification relevant to resources should be made in the resource flow template. It is recommended to study Chapter 5 carefully before modifying the simulation model generated by ChaStrobe.

B.1 Modeling Different Types of Precedence Constraints

In scheduling, there are four types of precedence relationships, which are Finish-To-Start (FTS), Start-To-Start (STS), Start-To-Finish (STF), and Finish-To-Finish (FTF).

In many cases, a composed set of FTS relationships, activities, and lead times can be used to model STS, STF, and FTF relationships. When possible and practical, FTS relationships should be preferred due to the simplicity. Modeling STF and FTF relationships in simulation should always be avoided, since the start dates of the successors of STF and FTF relationships are unconstrained. Instead of modeling STF and FTF, additional activities and FTS relationships should be used.

To model different types of precedence relationships, three main components are necessarily added in the work flow template:

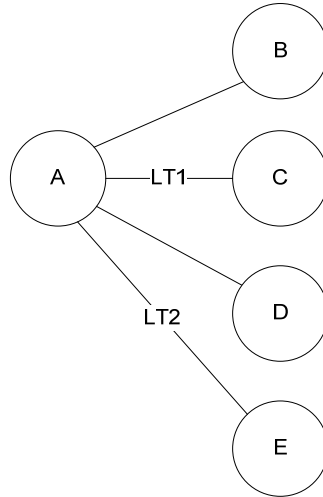
- Triggering event
- Lead time duration
- Tracking mechanism

Firstly, since discrete-event simulation advances the simulation clock from one event to the next closest event, the simulation model must ensure that events related to the precedence constraints, such as the start date of a successor, are registered in the future event list by creating a triggering event. In the original work flow template with FTS and zero lead time, the triggering event for a successor's start date is the completion of its predecessor (the end of an instance in ACT_Perform Combi). This is straightforward. However, when different types of precedence relationships are modeled, additional simulation elements, such as Combis and Queues, are required to trigger the events. For example, FTS relationships with lead time can be modeled by adding a Normal (a simulation model element in Stroboscope) to simulate the lead time. This is to ensure that the simulation clock will advance to the completion of the lead-time activity, and then the successor may start.

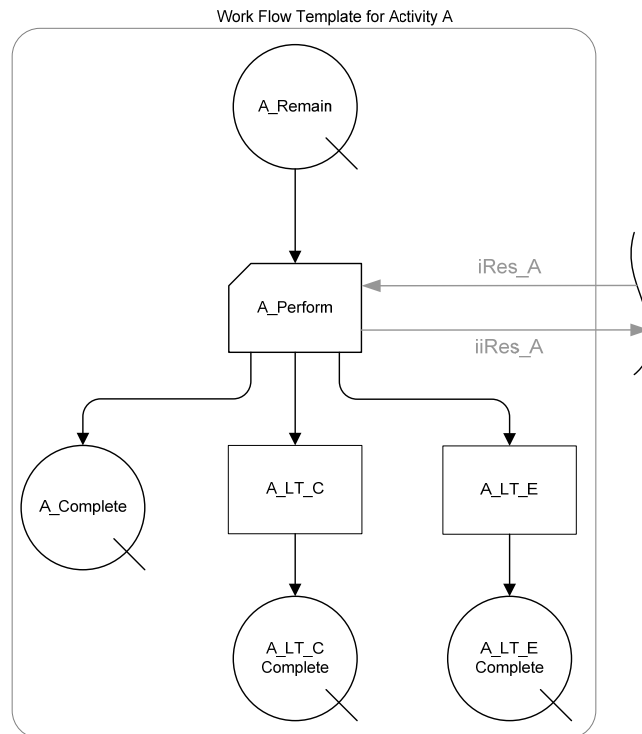
Secondly, lead time duration is recommended to be modeled as an activity with the duration of the lead time. At the completion of a lead-time activity, which is registered in the future event list of the simulation (FEL), the scanning process of discrete-event simulation will determine which activities can start, including the successors of the lead-time activity.

Thirdly, it is necessary to separate the number of completed works in predecessor activities with a lead time from the number of completed works without a lead time. For example, Activity A has two successors, B and C. Activities A and B have a FTS relationship without lead time, while Activities A and C have an FTS relationship with lead time. Therefore, the completed work in A should be tracked using two different sets: completed work after A is completed and completed work after the lead time.

Figure B.1 is an example of modeling FTS relationships with lead time. Figure B.1.a is a precedence diagram showing Activity A has four successors, B, C, D, and E. The precedence relationships between A and B and between A and D are FTS without lead time, while the precedence relationships between A and C and between A and E are FTS with different lead times. Figure B.1.b shows a work flow sub-network for Activity A having two different lead times for its successors. As shown in the figure, The A_LT_C and A_LT_E Normals are used to model the FTS lead times between Activities A and C, and between Activities A and E. The completions of A_LT_C and A_LT_E Normals are the triggering events, ensuring the simulation clock will advance to the events. The durations of A_LT_C and A_LT_E Normals simulate the duration of the lead times. Since lead times between Activities A and C and between Activities A and E are different, the two lead times are modeled separately using two Normals.



(a) Precedence diagram with activities with FTS with different lead



(b) A work flow sub-network with FTS with different lead

Figure B.1 Modeling Finish-To-Start relationships with different lead times

In Figure B.1.b, the number of Resource rq_RES in $A_LT_C_Complete$ and $A_LT_E_Complete$ Queues is used to determine whether successors C and E can start, by coding the semaphores of Activities C and E in $C_Perform$ and $E_Perform$ Combis,

respectively. For more detail about the semaphores of ACT_Perform, see Section 5.2.1, Work Flow Template.

Figure B.2 is an example of modeling FTS relationships without lead time and STS relationships with lead time. Figure B.2.a is a precedence diagram showing Activity A has four successors, B, C, D, and E, with different precedence relationships. The precedence relationships between A and B and between A and D are FTS without lead time, while the precedence relationships between A and C and between A and E are STS with different lead times. Figure B.1.b shows a work flow sub-network for Activity A having two different lead times for its successors. As shown in the figure, A_Trigger Combi is added in the work flow template as a trigger for the start of Activity A; the duration of A_Trigger Combi is zero. After the completion of A_Trigger (duration of zero), three activities start (i.e., A_Perform, A_LT_C, and A_LT_E). Normals A_LT_C and A_LT_E are used to model the STS lead times between Activities A and C, and between Activities A and E. The completion of A_LT_C and A_LT_E Normals is also a triggering event, ensuring that simulation clock will advance to the events. The durations of A_LT_C and A_LT_E Normals simulate the duration of the lead times. Since lead times between Activities A and C and between Activities A and E are different, the two lead times are modeled separately using two Normals.

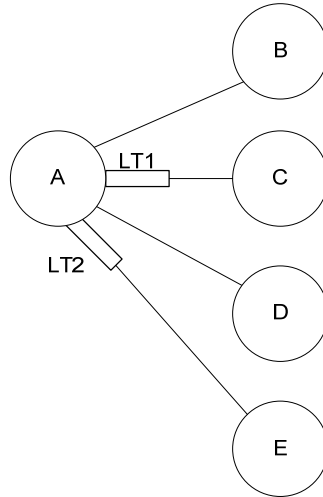
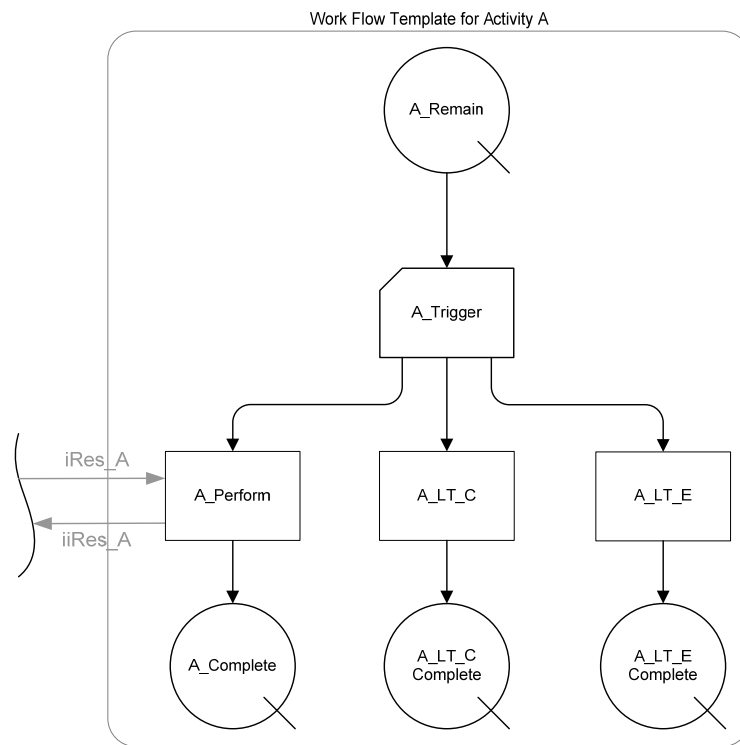


Figure B.2.a Precedence diagram with activities with FTS and STS with different lead times



(b) A work flow sub-network with FTS and STS with different lead times

Figure B.2 Modeling Start-To-Start relationships with different lead times

B.2 Modeling Operational Level of Activities

Although the Sequence Step Algorithm (SQS-AL) and the simulation model templates are designed to solve and model repetitive projects at an activity level, it is possible to modify the templates to model the project at an operational level. As shown Figure B.3, there are two types of resources here, activity resources and operational resources. The activity resource is modeled by using the resource flow template, while the operational resource is not. Only queue and resource are specified for the use of the operational resource, and their data are not taken into account by SQS-AL.

The operational activities and operational resources can be modeled in the work flow template by adding queues and combis, as shown in Figure B.3. Operations 1 and 2, modeled by Operation 1 Combi and Operation 2 Normal in Activity A requires operational resource residing in the “operational resource” Queue to perform the operation. After Activity A starts, which is determined by precedence constraints and resource available constraints of Resource X in the resource flow sub-network, Operation 1 draws an operational resource from the queue. Then, both work and operational resource will be passed to Operation 2 to complete this unit. After Operation 2 completes, the operational resource returns to its queue, whereas completed work is sent to “finish perform” Normal and “work done” Queue.

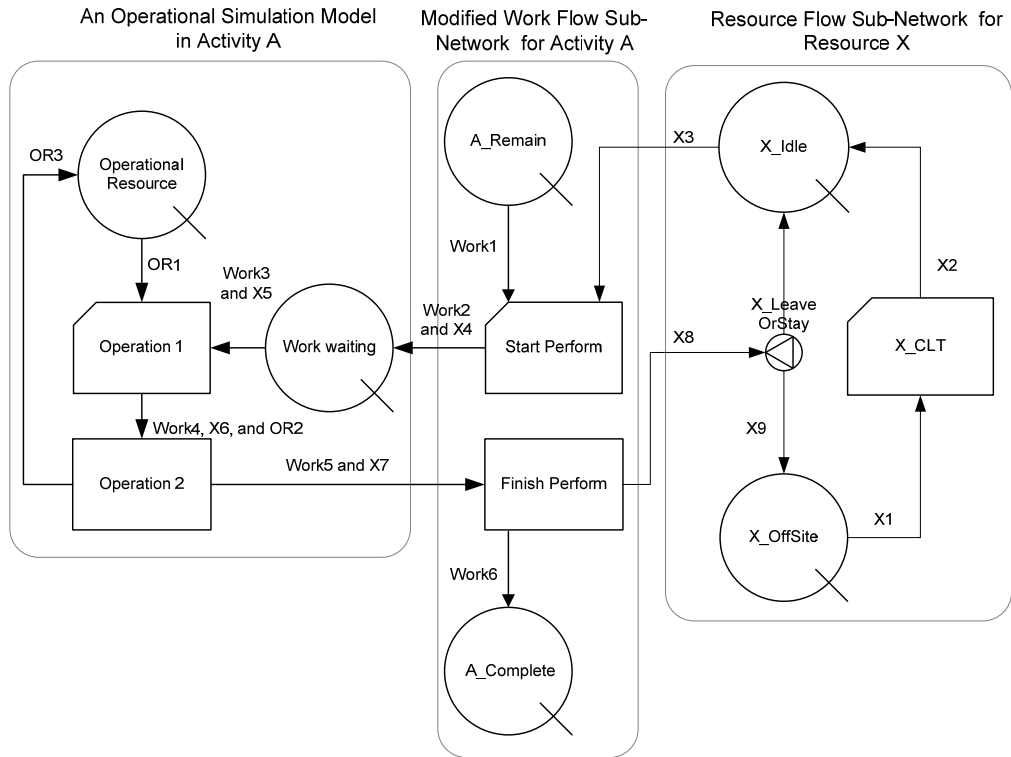


Figure B.3 Modeling an activity at an operational level

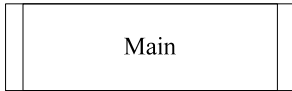
In certain circumstances, one might want to model an operational resource that serves several operations belong to different activities. This can be done in traditional simulation modeling way. However, sequence of activities may change and possibly complicate the schedule. Imagine that an operational resource is utilized among several operations within two different activities, and it causes both activities to start and stop due to the lack of the operation resource. Accordingly, the idle times of the main resources for both activities are disguised by the fact that activities have started, yet not finished. Remember, sub-operation and operational resources in this discussion are relatively insignificant compared to activity resource. If the operational resources are significant or desired to model, one solution to this is modeling the sub-operation by

using work flow template and modeling the operational resource by using resource flow template. Although this is viable, it is recommended to keep the simulation model simple.

APPENDIX C

FLOW CHART FOR THE CHASTROBE APPLICATION

Appendix C presents the flow charts of ChaStrobe. These flow charts provide information of how ChaStrobe collects data and create simulation. Explanations, references, and simulation code examples are given to clarify each process in the flow charts.

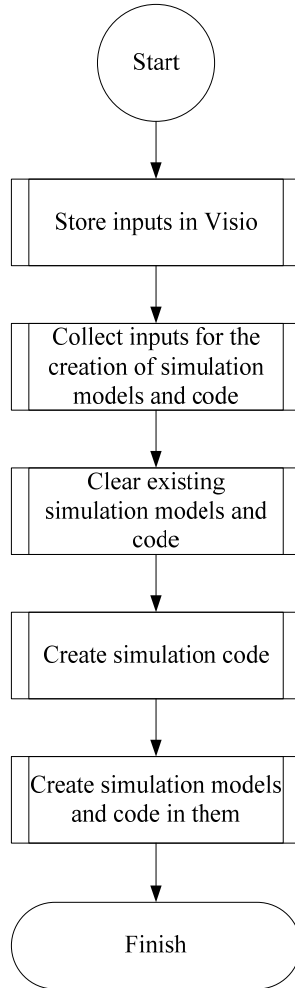


See hidden sheets in ChaStrobe Visio file by right click on the drawing and select Show/Hide Page Tab

See Section 8.1.1 for explanations of how ChaStrobe collects inputs

See Sections 5.3.1 (MP), 5.3.2 (PO), and 5.3.4 (CS) for simulation code and explanations

See Section 5.3.3 (CME) for simulation code and explanations



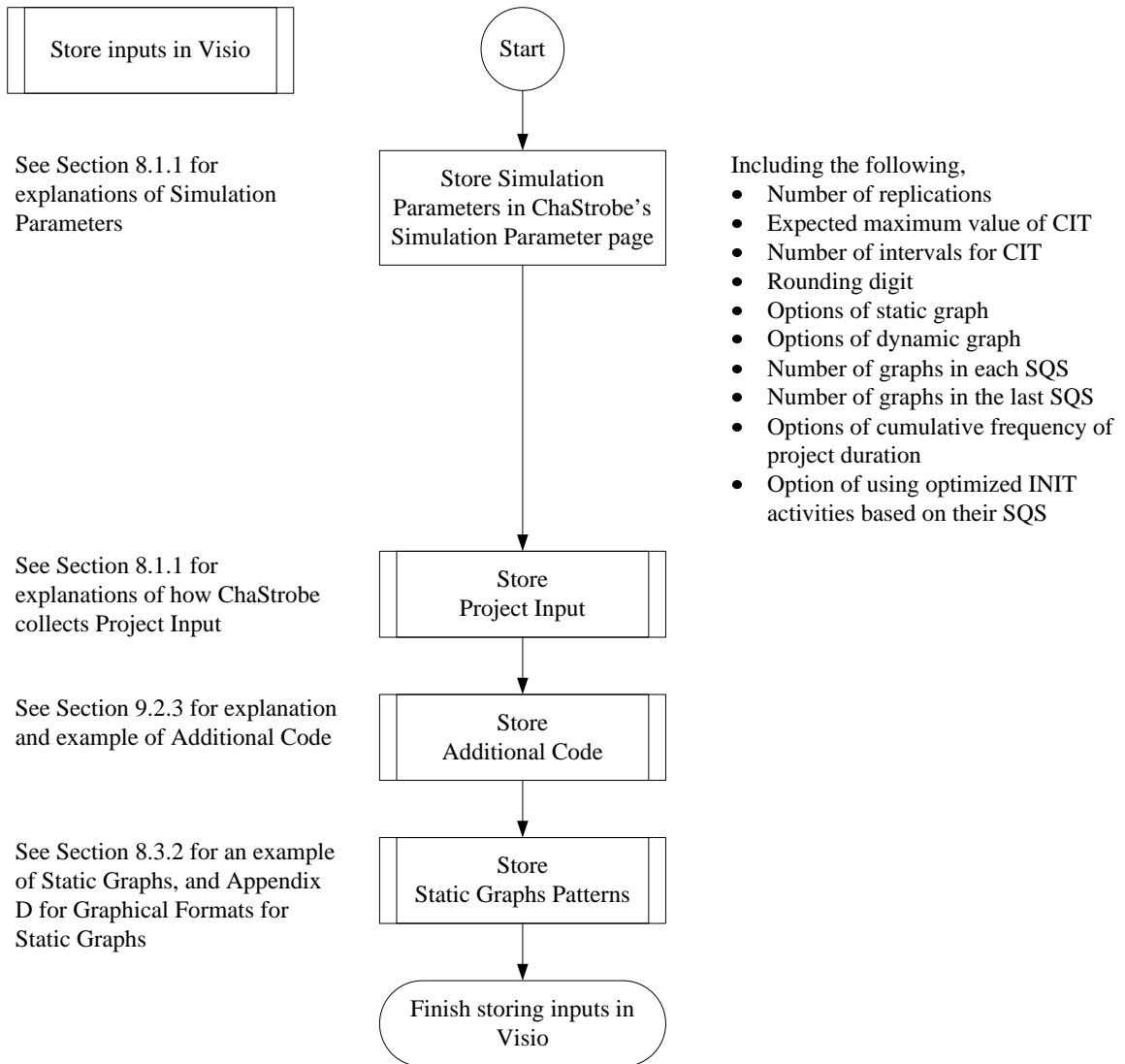
Data are saved in ChaStrobe Visio file

Data are collected for the purposes of simulation models and code generation

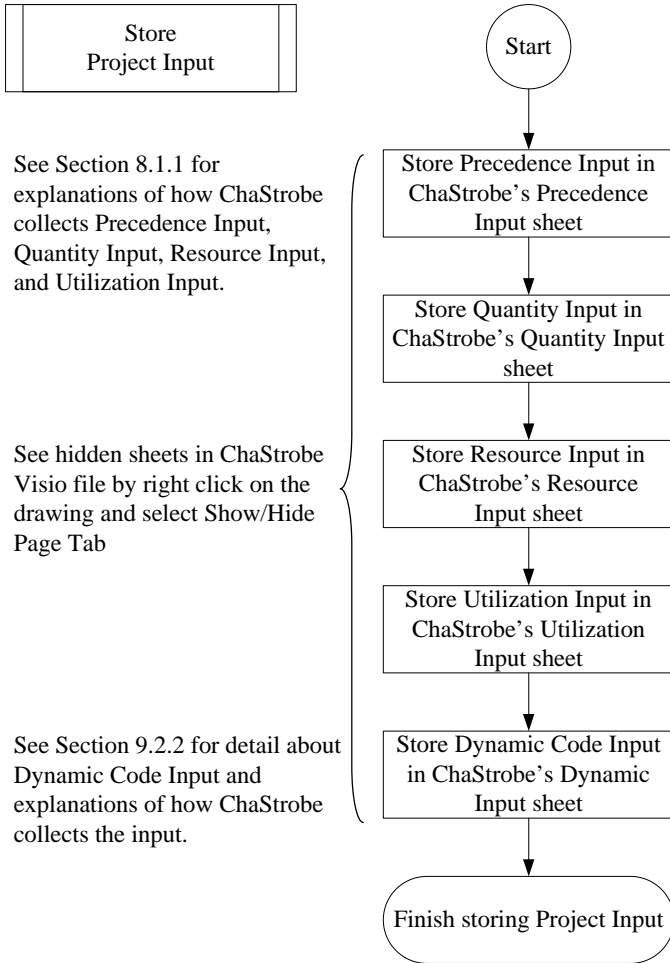
Including simulation code for Model Parameters, Programming Objects, Control Statements, Additional Code, and Dynamic Code

Including work flow sub-networks and resource flow sub-networks and their simulation code

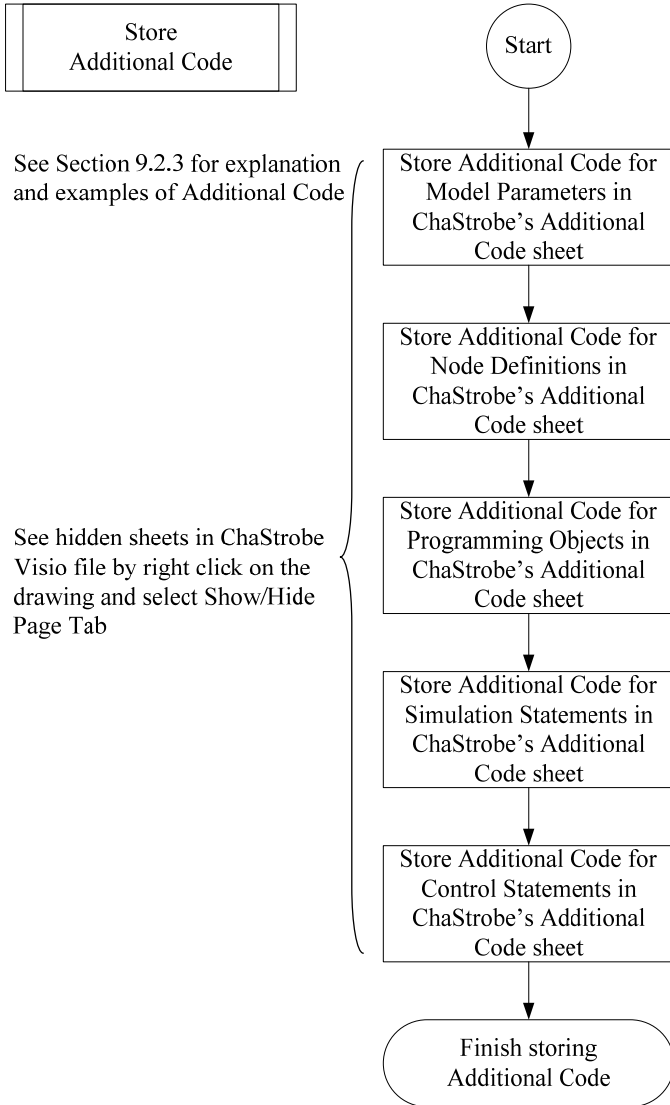
Flowchart 1 Main procedure



Flowchart 2 Storing inputs in Visio



Flowchart 3 Storing Project Input

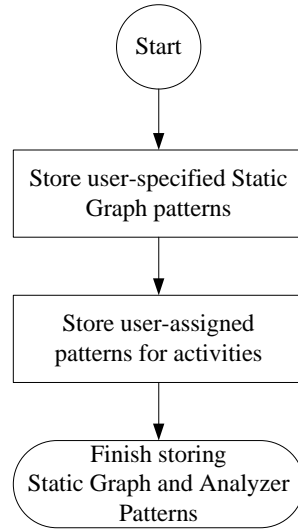


Flowchart 4 Storing Additional Code

Store
Static Graph and
Analyzer Patterns

See hidden sheets in ChaStrobe
Visio file by right click on the
drawing and select Show/Hide
Page Tab

See Section 8.3.2 for an example
of Static Graphs, and Appendix
D for Graphical Formats for
Static Graphs

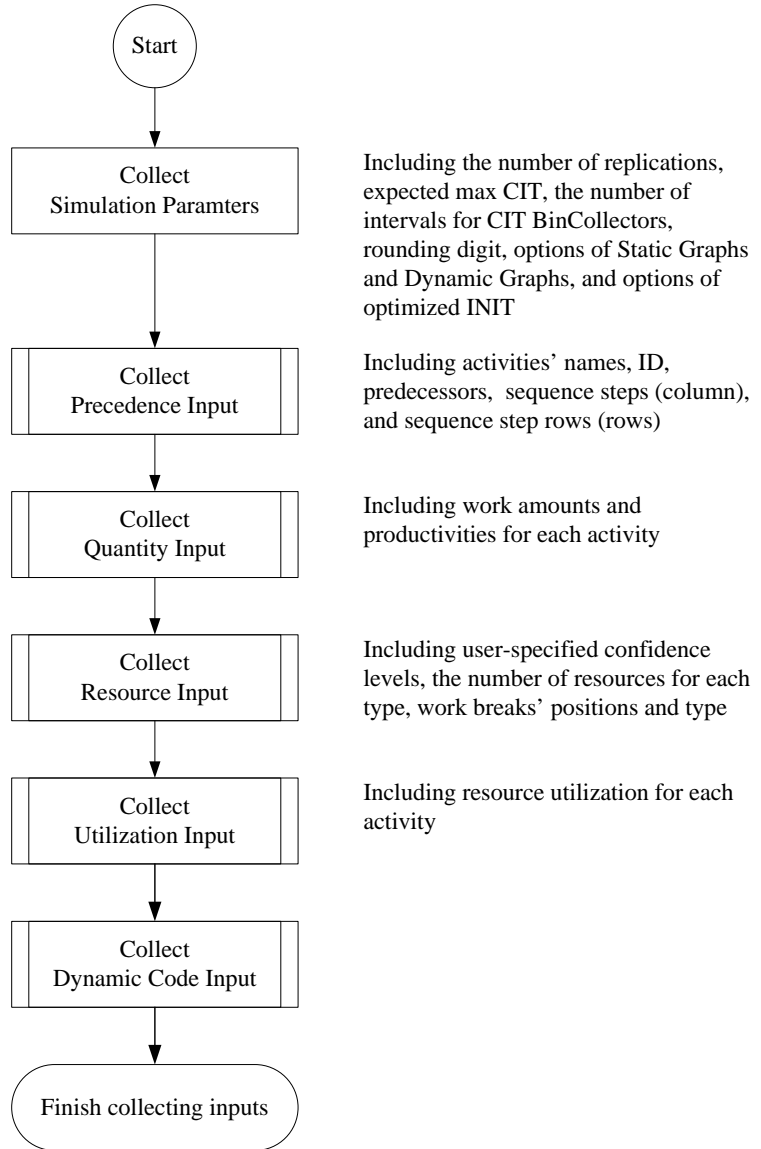


Flowchart 5 Storing Static Graph and Analyzer Patterns

Collect inputs for the creation of simulation models and code

See Section 8.1.1 for explanations of how ChaStrobe collects Simulation Parameters, Precedence Input, Quantity Input, Resource Input, and Utilization Input.

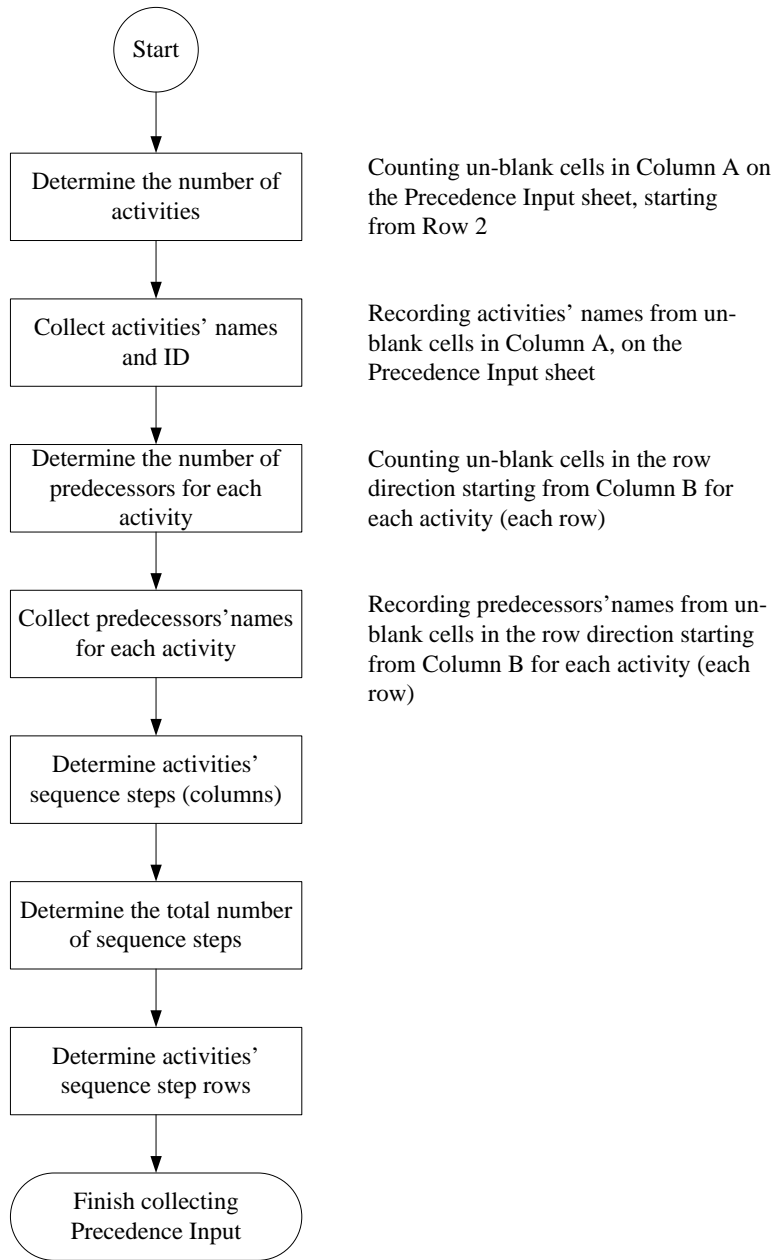
See Section 9.2.2 for detail about Dynamic Code Input and explanations of how ChaStrobe collects the input.



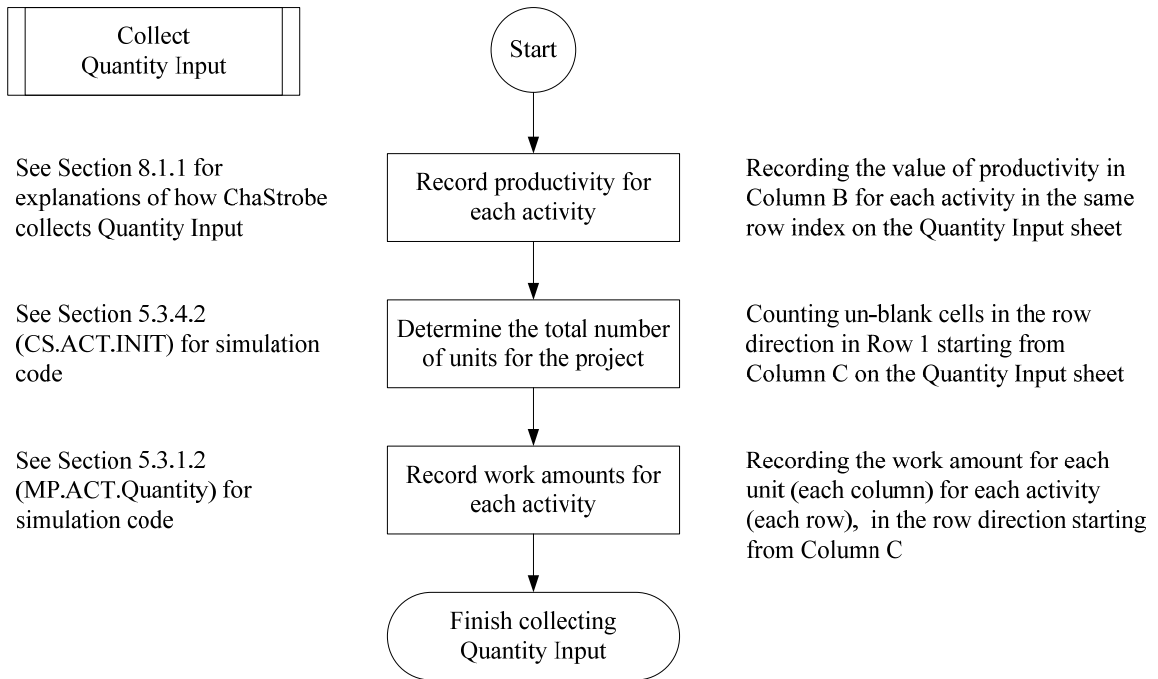
Flowchart 6 Collecting inputs for the creation of simulation model and code

Collect
Precedence Input

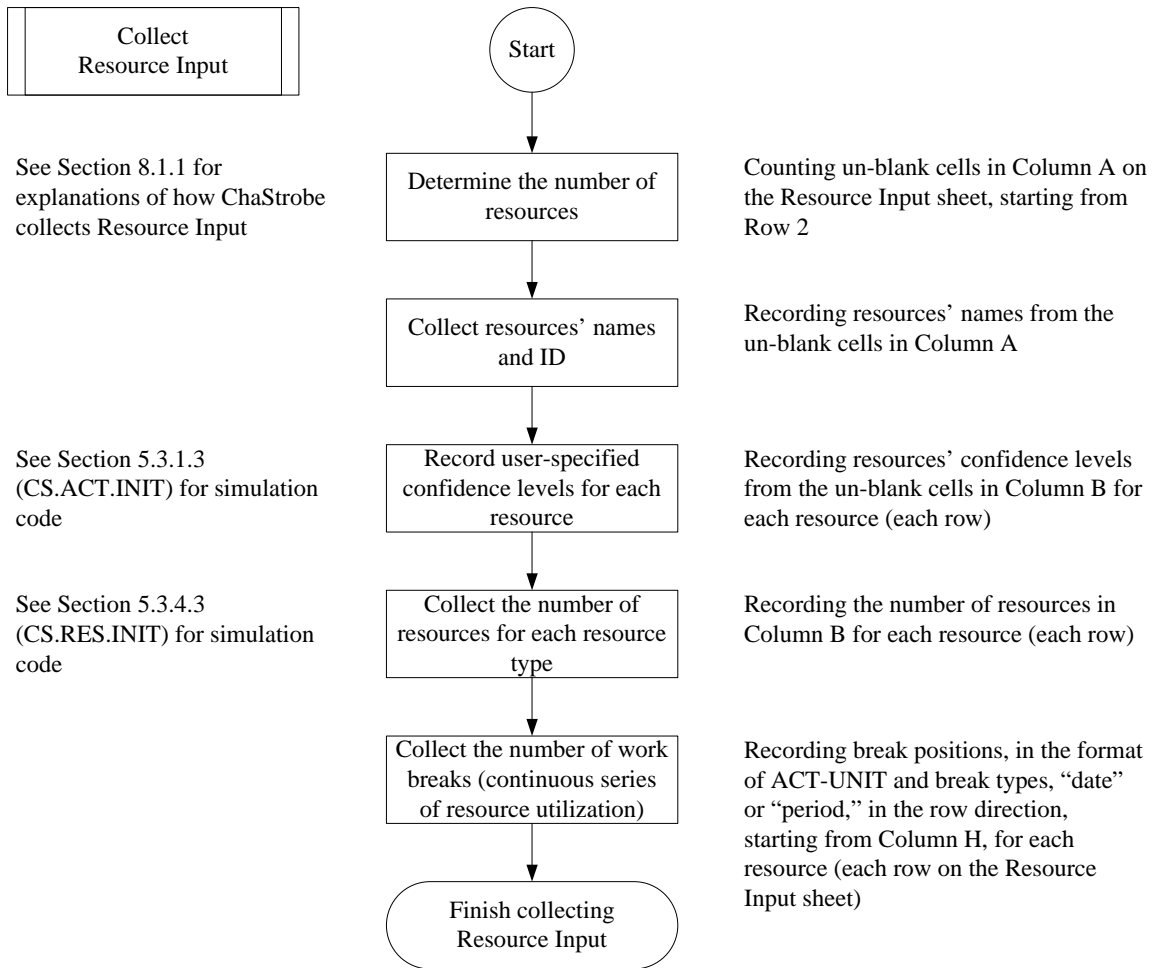
See Section 8.1.1 for explanations of how ChaStrobe collects Precedence Input



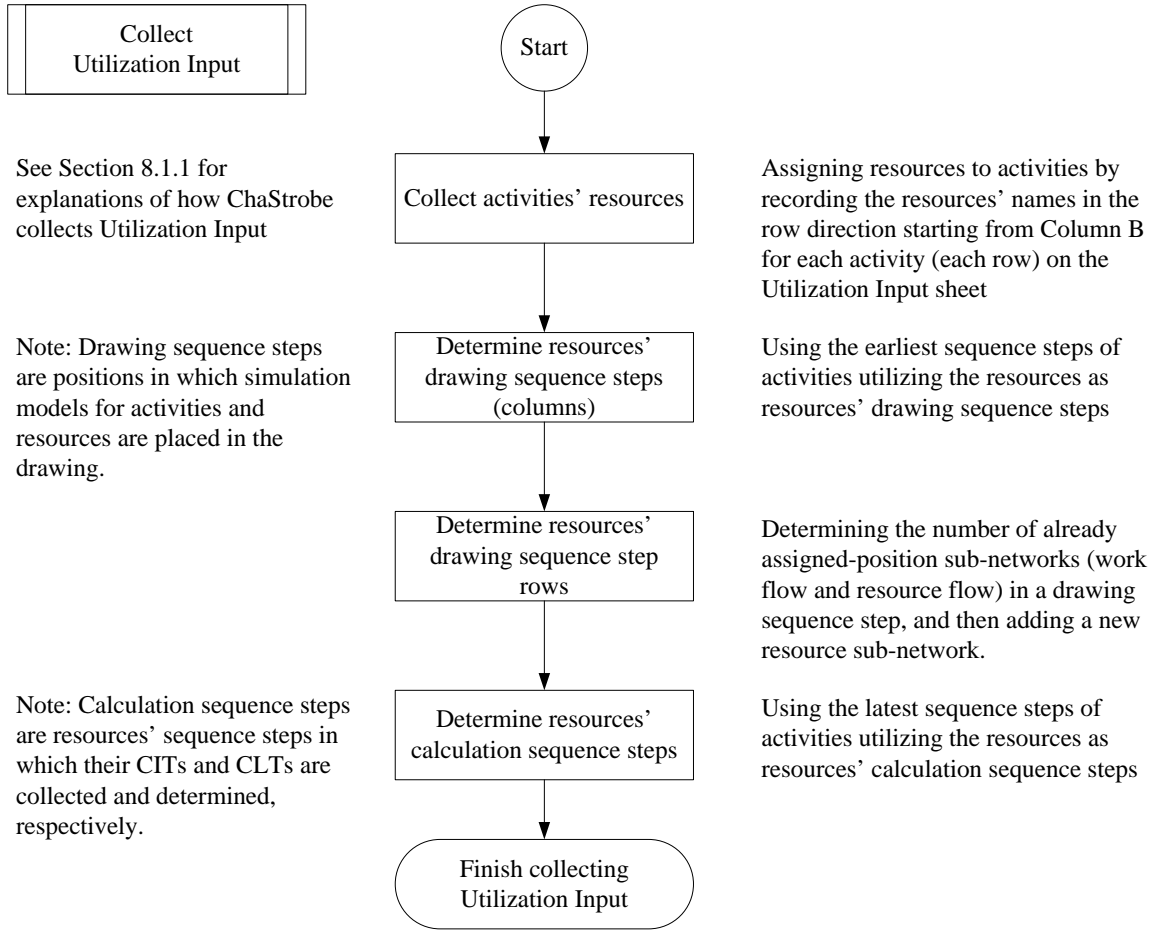
Flowchart 7 Collecting Precedence Input



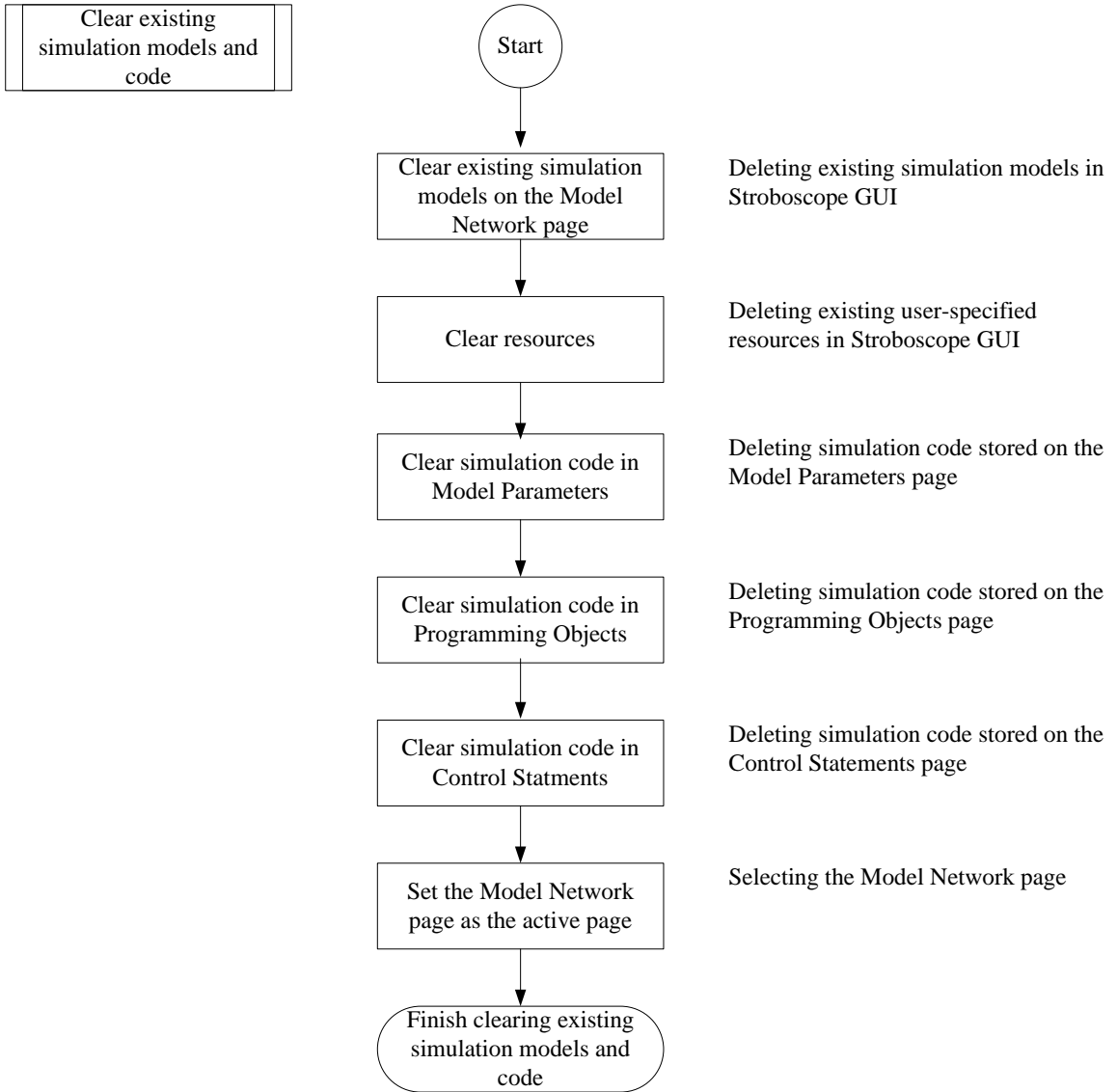
Flowchart 8 Collecting Quantity Input



Flowchart 9 Collecting Resource Input



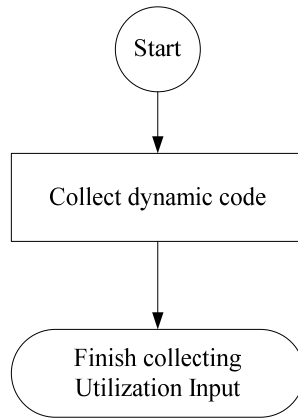
Flowchart 10 Collecting Utilization Input



Flowchart 11 Clearing existing simulation model and code

Collect
Dynamic Code Input

See Section 9.2.2 for detail about Dynamic Code Input and explanations of how ChaStrobe collects the input.



On the Dynamic Code Input sheet, recording Dynamic Code index in Column A and appending dynamic code in the row direction (separated by a single space), starting from Column B, for each row.

Note: Each row on the Dynamic Code Input sheet is one line of simulation code.

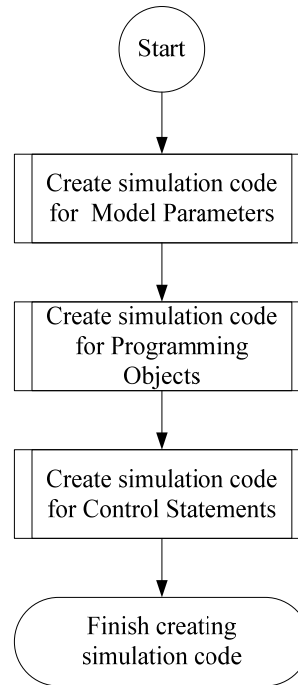
Flowchart 12 Collecting Dynamic Code Input

Create simulation code

See Section 5.3.1 (MP) for simulation code and explanation

See Section 5.3.2 (PO) for simulation code and explanation

See Section 5.3.4 (CS) for simulation code and explanation

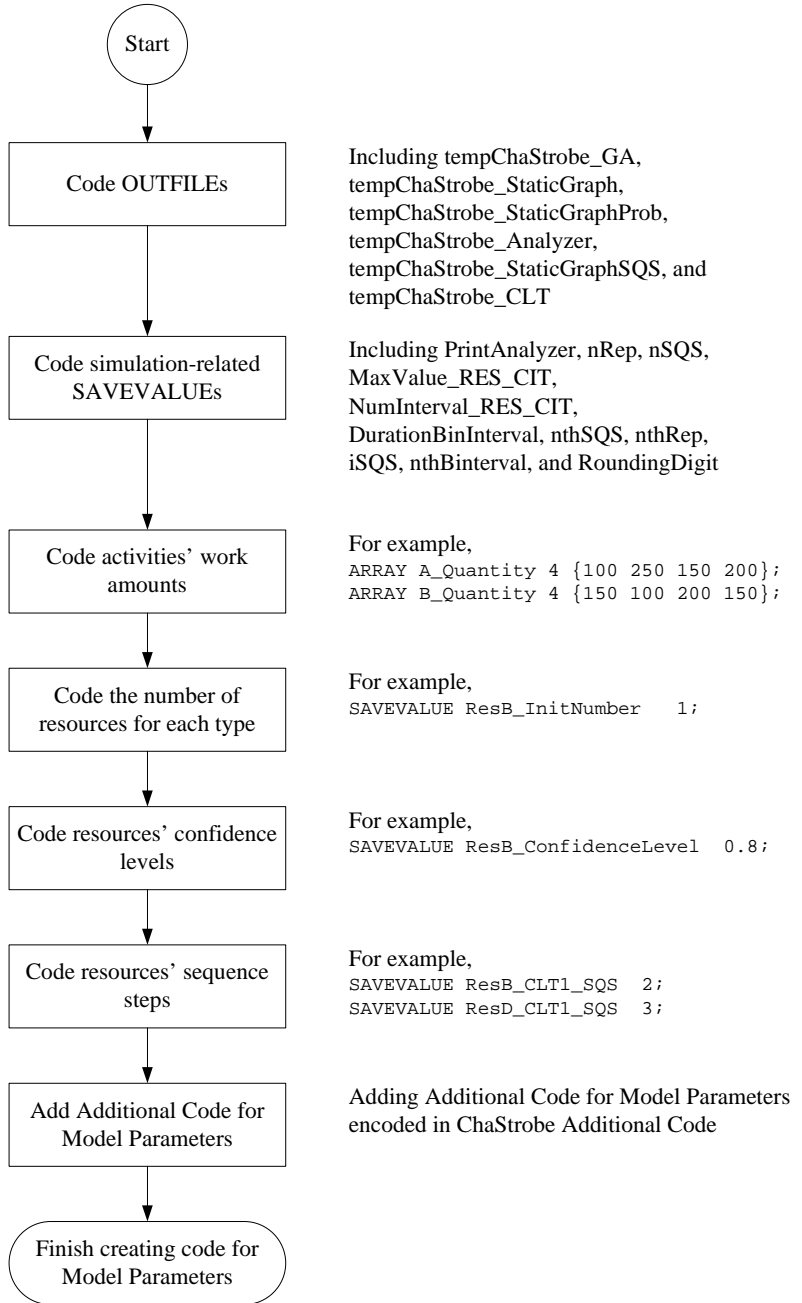


Note: Simulation code for Model Parameters, Programming Objects, and Control Statements can be viewed in Stroboscope Global Code form after created.

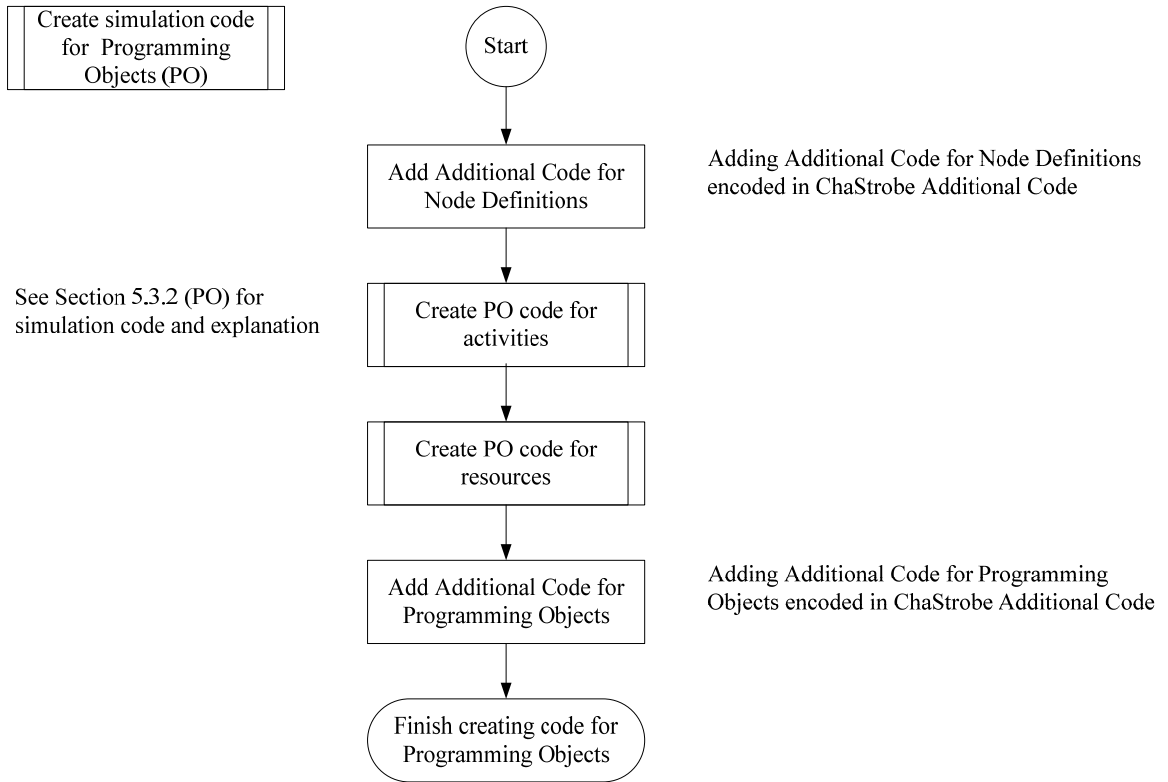
Flowchart 13 Creating simulation code

Create simulation code for Model Parameters

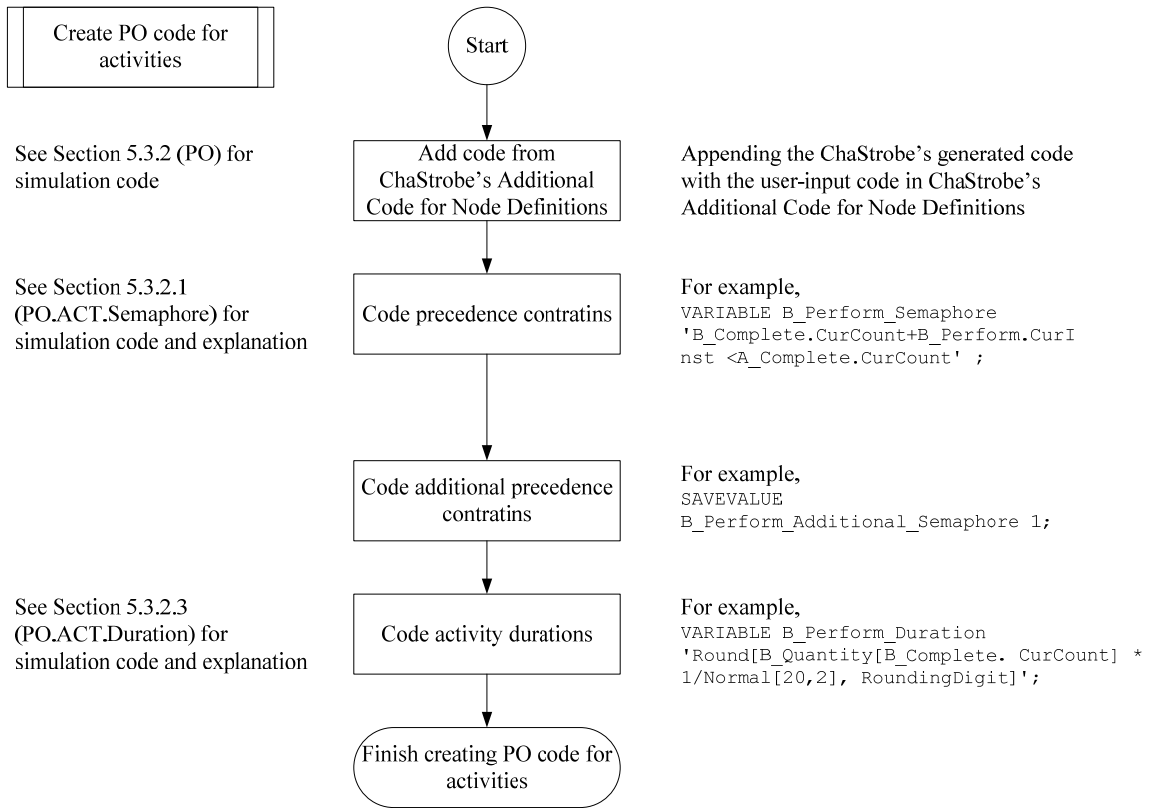
See Section 8.3 for details about output from ChaStrobe



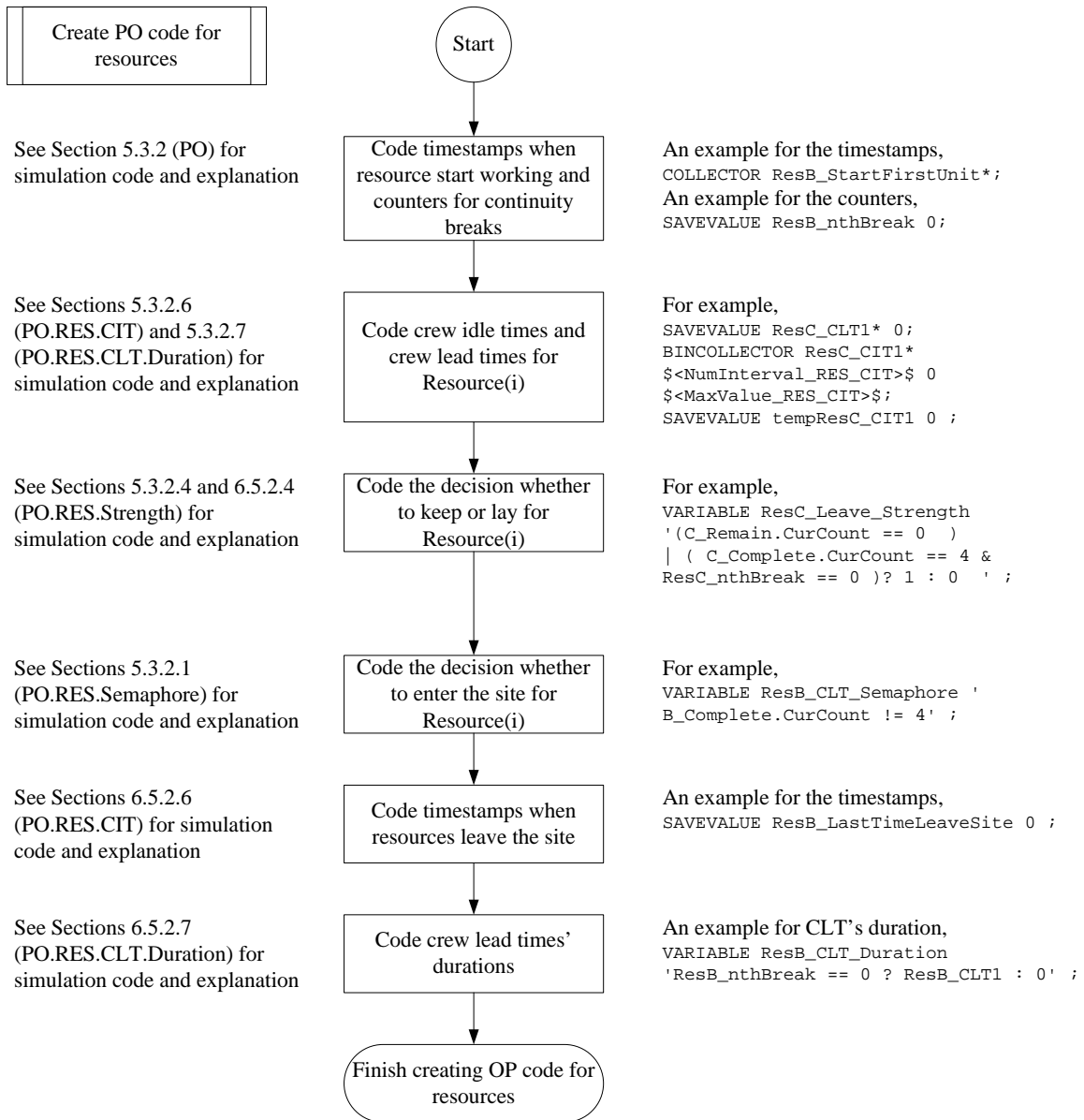
Flowchart 14 Creating code for Model Parameters



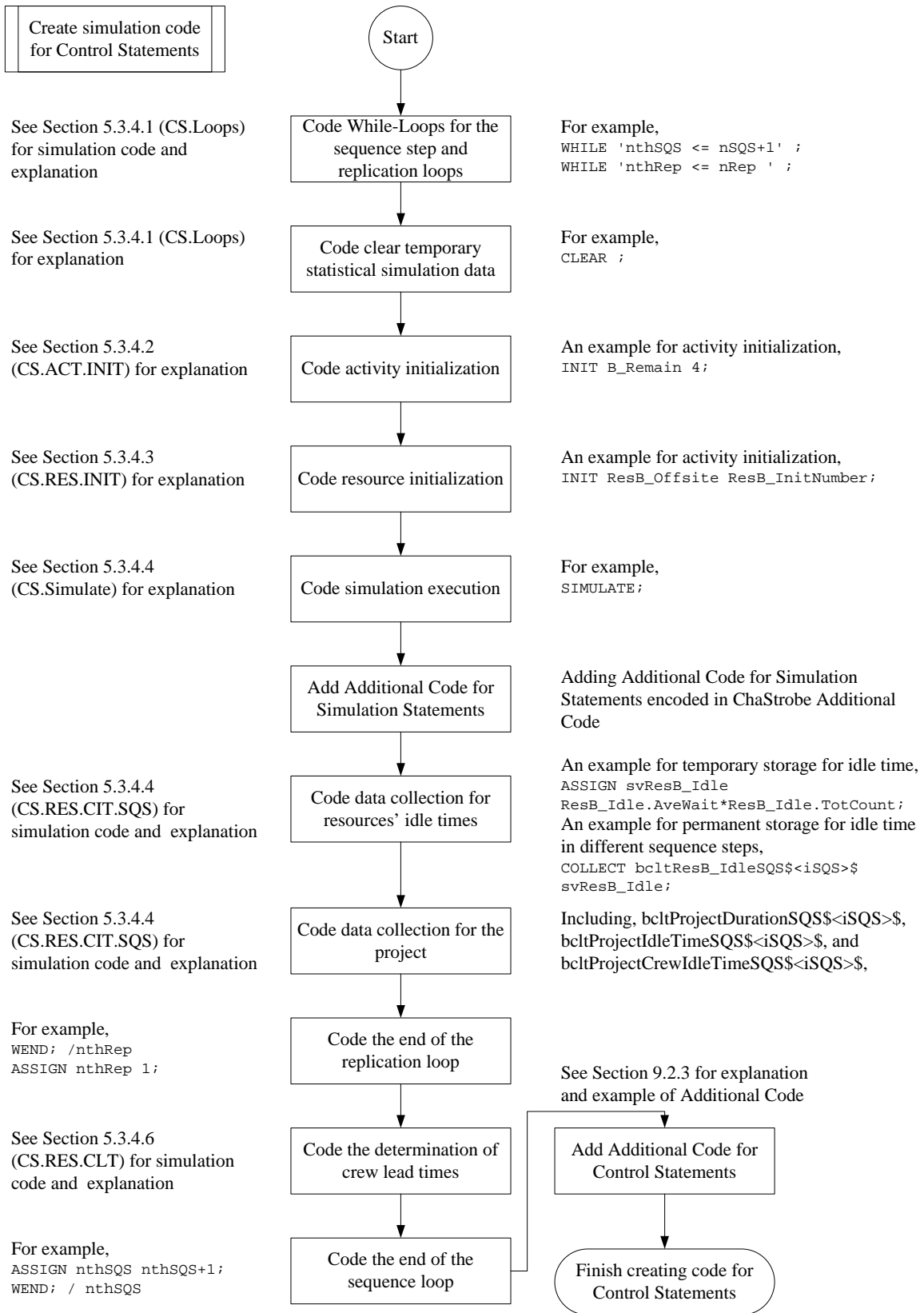
Flowchart 15 Creating code for Programming Objects



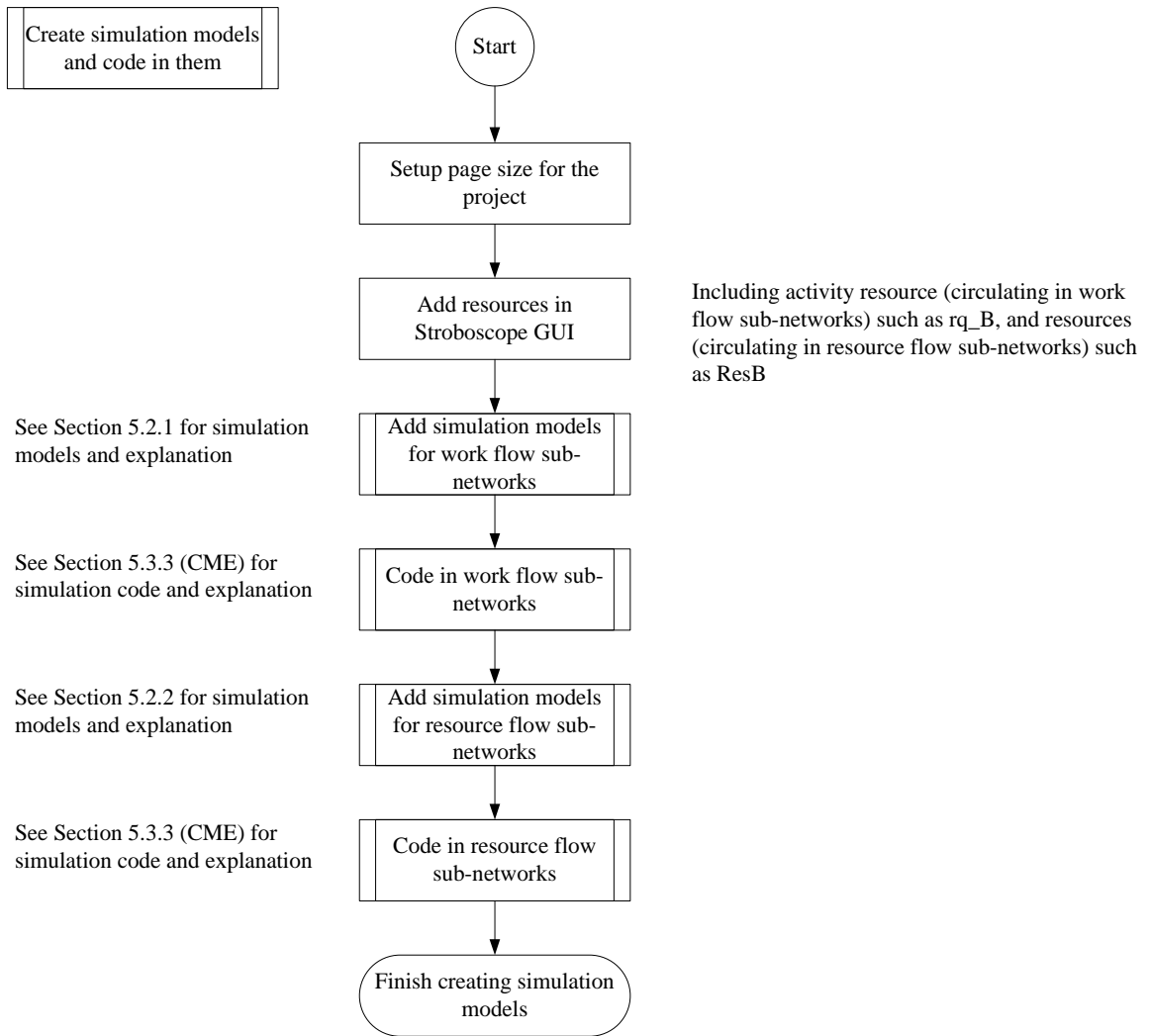
Flowchart 16 Creating PO code for activities



Flowchart 17 Creating PO code for resources



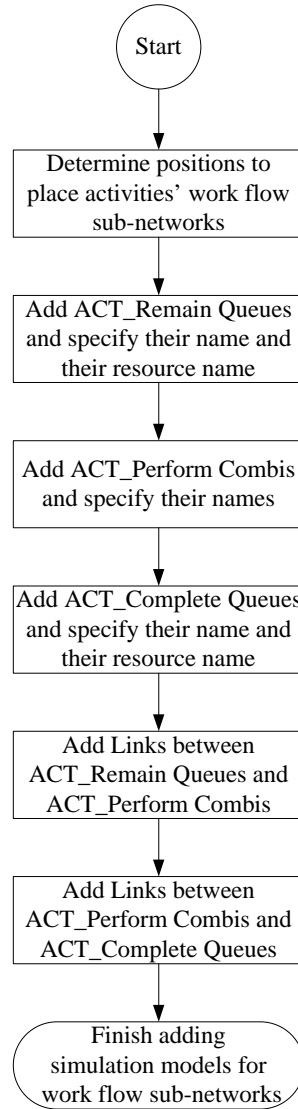
Flowchart 18 Creating code for Control Statements



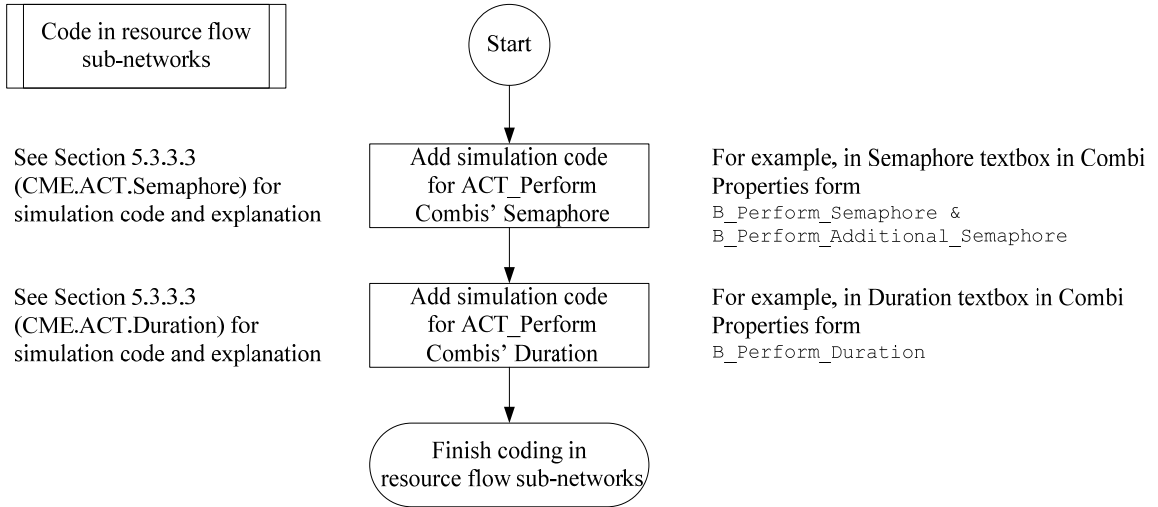
Flowchart 19 Creating simulation model

Add simulation models
for work flow sub-
networks

See Section 5.2.1 for simulation
models and explanation



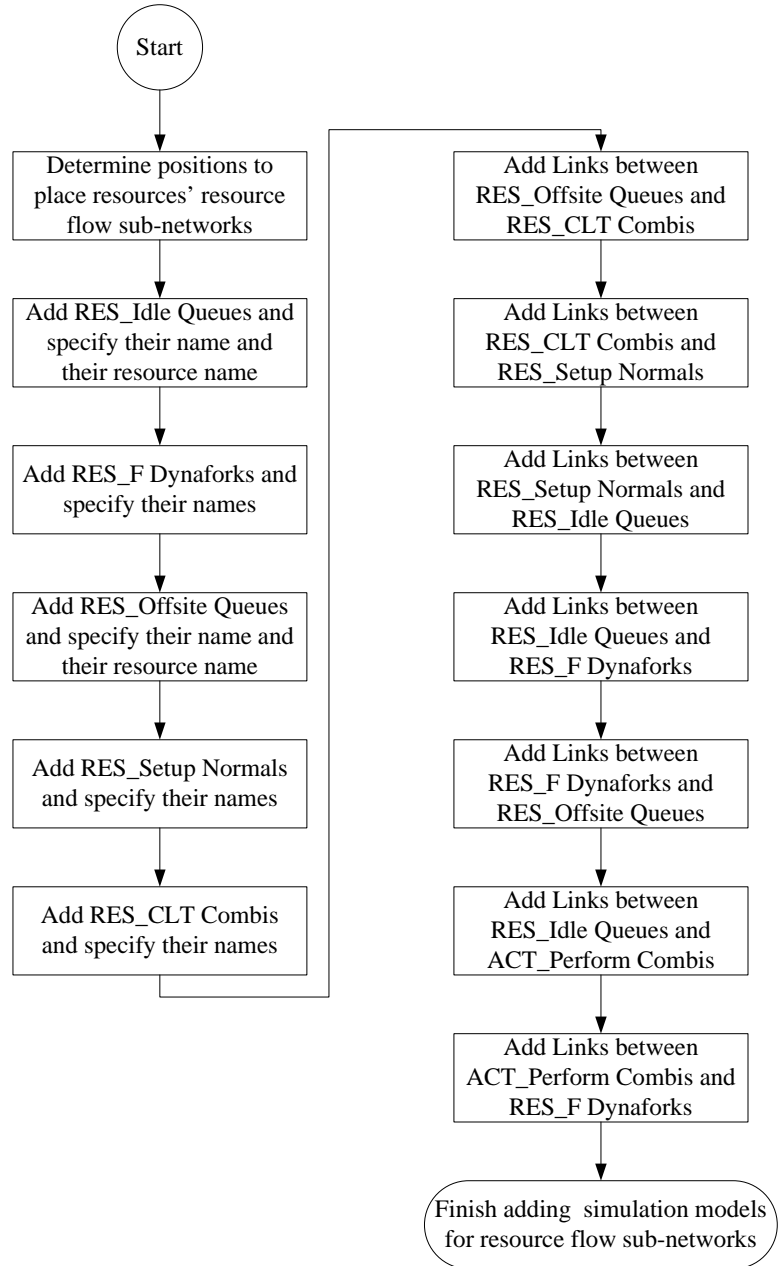
Flowchart 20 Adding simulation model for work flow sub-networks



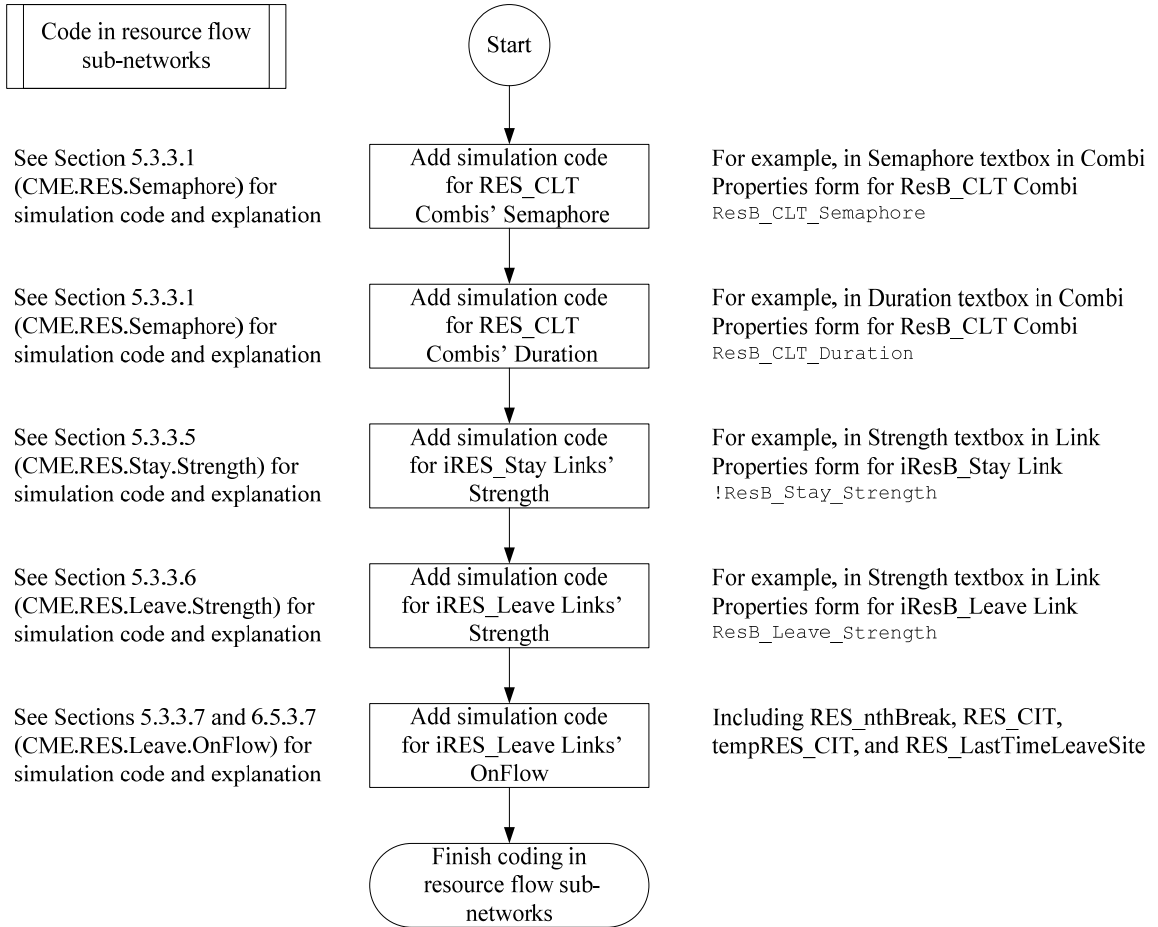
Flowchart 21 Coding in resource flow sub-networks

Add simulation models
for resource flow
sub-networks

See Section 5.2.2 for simulation
models and explanation



Flowchart 22 Adding simulation model for resource flow sub-networks



Flowchart 23 Coding in resource flow sub-networks

APPENDIX D

GRAPHICAL FORMATS FOR STATIC GRAPHS

Appendix D presents a key to numerical indicators for graphical formats for Static Graphs in the ChaStrobe application. As explained in Section 8.3.2, users can create a production diagram using the data recorded in tempChaStrobe_StaticGraph.txt, after ChaStrobe schedules the project. Each line in the created production diagram in Static Graphs is formatted by user-specified patterns. Users first create Static Graph patterns in ChaStrobe's Static Graph and Analyzer tab, as shown in Figure D.1, and then assign a created pattern to each sub-activity, as shown in Figure D.2.

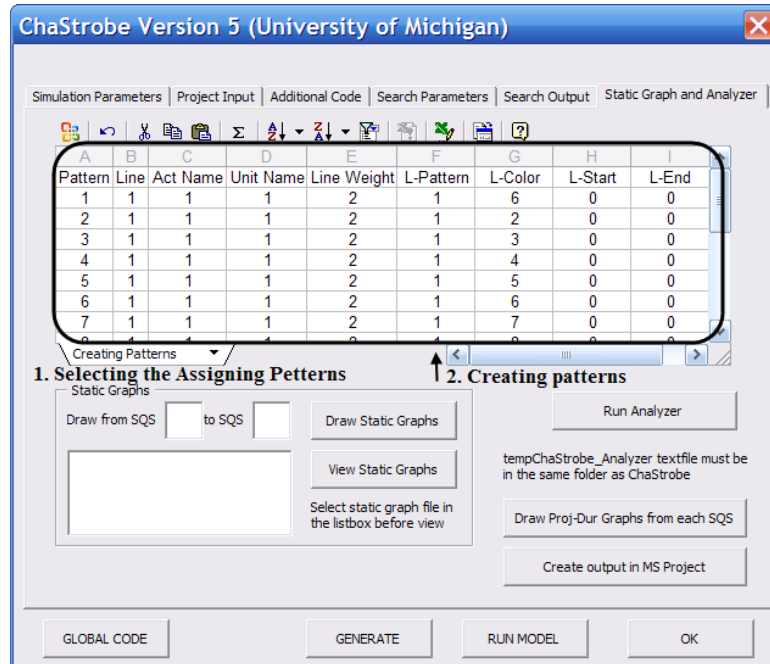


Figure D.1.a Creating line patterns for Static Graphs

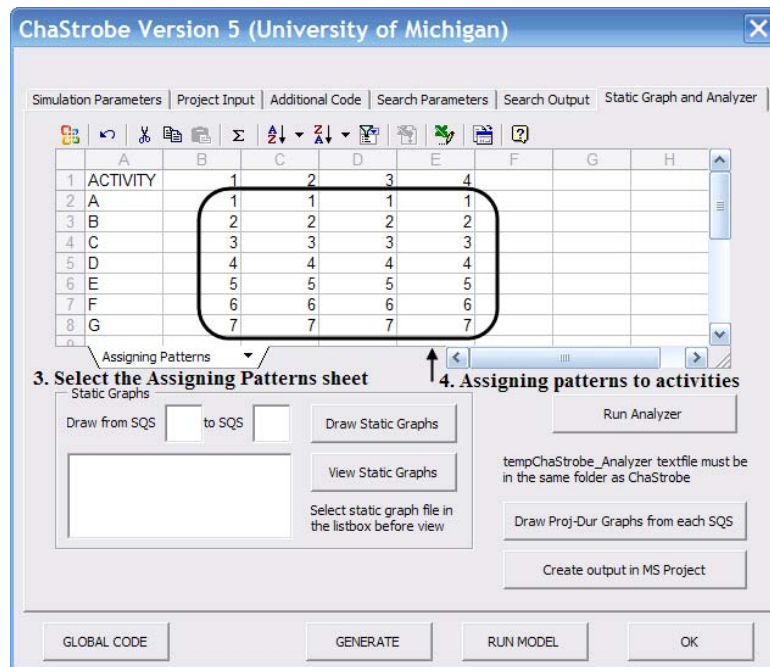


Figure D.1.b Assigning line patterns to sub-activities

Figure D.1 Creating and assigning line patterns to production diagrams in Static Graphs

To create a pattern, users must specify the following parameters in the Creating Patterns sheet:

1. Pattern ID, used to assign a pattern to an activity. Users specify Pattern ID in Column A, as shown in Figure B.1.
2. Line options, 0 and 1, specifying whether to draw a line or not. If this option is set to 1, the line for the corresponding pattern will be drawn. If it is set to 0, the line will not be drawn. Users specify Line options in Column B.
3. Act Name options, 0 and 1, specifying whether to label a line with its activity name. If this option is set to 1, the line for the corresponding pattern will be labeled with its activity name. If it is set to 0, the line will not be labeled. Users specify Act Name options in Column C.
4. Unit Name options, 0 and 1, specifying whether to label a line for an activity with its unit name (e.g., ID1, House 1, or just 1). If this option is set to 1, the line for the corresponding pattern will be labeled with its unit name. If it is set to 0, the line will not be labeled. Users specify Unit Name options in Column D.
5. Line Weight parameters, specifying the weight or thickness of the line. Users specify Line Weight parameters in Column E.
6. L-Pattern parameters (Line Pattern), specifying line patterns, such as dotted lines and dashed lines. Users specify Line Pattern in Column F. Examples of L-Patterns numerical indicators are given in Table D.1.




















Line Pattern ID	Example
0	No Line
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	

Table D.1 Line pattern ID for Static Graphs

7. L-Color parameters (Line Color), specifying line colors, such as red or green.

Users specify Line Pattern in Column G. Examples of L-Color numerical indicators are given in Table D.2:

Line Color ID	Color
0	Black
1	White
2	Red
3	Bright Green
4	Blue
5	Yellow
6	Pink
7	Turquoise
8	Dark Red
9	Green
10	Dark Blue
11	Dark Yellow
12	Violet
13	Teal
14	Gray-25%

Table D.2 Line color IDs for Static Graphs

8. L-Start parameters (Line Start), specifying line starts, such as arrows and circulars. Users specify line starts for each pattern in Column H. Examples of L-Color numerical indicators are given in Table D.3.
9. L-End parameters (Line End), specifying line ends for each pattern, such as arrows or circulars. Users specify Line Pattern in Column I. Examples of L-Color numerical indicators are shown in Table D.3, the same as for L-Start parameters.

After patterns are specified on the Creating Patterns sheet (Figure D.1.a) in the Static Graph and Analyzer tab, users assign a specified pattern to activities on the Assigning Patterns sheet as shown in Figure D.1.b.


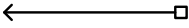
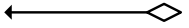

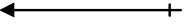
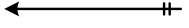
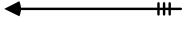
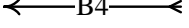

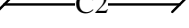

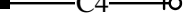

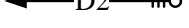
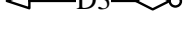

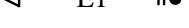
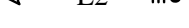
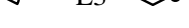
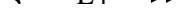
Line Start and End Pattern ID (at the start)	Example	Line Start and End Pattern ID (at the end)
0		20
1		21
2		22
3		23
4		24
5		25
6		26
7		27
8		28
9		29
10		30
11		31
12		32
13		33
14		34
15		35
16		36
17		37
18		38
19		39

Table D.3 Line start and end pattern IDs for Static Graphs

BIBLIOGRAPHY

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.

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

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

Carr, R. I., and Meyer, W. L. (1983), "Planning Construction of Repetitive Building Units", *Journal of the Construction Division*, ASCE, 100(CO3), 403-412.

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

Chehayeb, N. N., and AbouRizk, S. M. (1998), "Simulation-Based Scheduling with Continuous Activity Relationships", *Journal of Construction Engineering and Management*, ASCE, 124(2), 107-115.

Chrzanowski, E. N., and Johnston, D. W. (1987), "Application of Linear Scheduling", *Journal of Construction Engineering and Management*, ASCE, 112(4), 476-491.

Crandall K.C. (1976), "Probabilistic Time Scheduling", *Journal of the Construction Division*, ASCE, 102(CO3), 415-423.

Crandall K.C. (1977), "Analysis of Schedule Simulations", *Journal of the Construction Division*, ASCE, 103(CO3), 387-394.

El-Rayes, K. (2001), "Object-Oriented Model for Repetitive Construction Scheduling", *Journal of Construction Engineering and Management*, ASCE, 127(3), 199-205.

El-Rayes, K. A. (1997), *Optimized Scheduling for Repetitive Construction Projects*, Ph.D. Thesis, Concordia University, Montreal, Quebec, Canada.

- El-Rayes, K., and Moselhi, O. (2001) "Optimizing Resource Utilization for Reptitive Construction Projects", *Journal of Construction Engineering and Management*, ASCE, 127(1), 18-27.
- Handa, V.K., and Barcia, R.M. (1987), "Linear Scheduling Using Optimal Control Theory", *Journal of Construction Engineering and Management*, ASCE, 112(3), 387-393.
- Harris, R.B. (1978). *Precedence and Arrow Networking Techniques for Construction*, John Wiley & Sons. New York, NY.
- Harris, R. B., and Ioannou, P. G. (1998), "Scheduling Projects with Repeating Activities", *Journal of Construction Engineering and Management*, ASCE, 124(4), 269-278.
- Hijazi, A. M. (1989), *Simulation Analysis of Linear Construction Processes*, Ph.D. Thesis, Purdue University, West Lafayette, IN.
- Johnston, D.W. (1981), "Linear Scheduling Method for Highway Construction", *Journal of the Construction Division*, ASCE, 107(CO2), 247-261.
- Law, A.M., and Kelton, D.K. (2000) *Simulation Modeling and Analysis*, International Edition. McGraw-Hill, New York, NY.
- Leu, S., and Hwang, S. (2001), "Optimal Repetitive Scheduling Model with Shareable Resource Constraint", *Journal of Construction Engineering and Management*, ASCE, 127(4), 270-280.
- Lutz, J. D. (1990), *Planning of Linear Construction Projects using Simulation and Line Of Balance*, Ph.D. Thesis, Purdue University, West Lafayette, IN.
- Lutz, J.D., and Halpin, D.W. (1994), "Simulation of Learning Development in Repetitive Construction", *Journal of Construction Engineering and Management*, ASCE, 120(4), 753-773.
- Martinez, J. C. (1996), *STROBOSCOPE: State and Resource Based Simulation of Construction Process*, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.
- Mattila, K. G. and Park, A. (2003), "Comparison of Linear Scheduling Model and Repetitive Scheduling Method", *Journal of Construction Engineering and Management*, ASCE, 129(1), 56-64.
- Miller, R.W., (1963), "Schedule, Cost, and Profit Control with PERT" McGraw-Hill, New York, NY.
- Moselhi, O., El-Rayes, K., (1993) "Scheduling of Repetitive Projects with Cost Optimization", *Journal of Construction Engineering and Management*, ASCE, 119(2), 681-697.

- O'Brien, D.P. (1985), "SIREN: A Repetitive Construction Simulation Model", *Journal of Construction Engineering and Management*, ASCE, 111(3), 308-323.
- O'Brien, J.J. (1975), "VPM Scheduling for High-Rise Building", *Journal of the Construction Division*, ASCE, 101(CO4), 895-905.
- 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.
- Peer, S. (1974), "Network Analysis and Construction Planning", *Journal of the Construction Division*, ASCE, 100(CO3), 203-210.
- Perera, S. (1980), "Linear Programming Solution to Network Compression", *Journal of the Construction Division*, ASCE, 106(CO2), 315-326.
- Perera, S. (1983), "Resource Sharing in Linear Construction", *Journal of Construction Engineering and Management*, ASCE, 109(1), 102-111.
- Reda, R. M., (1990), "RPM: Repetitive Project Modeling", *Journal of Construction Engineering and Management*, ASCE, 116(2), 316-330.
- Russell, A. D., and Wong, C.M., (1993), "New Generation of Planning Structures", *Journal of Construction Engineering and Management*, ASCE, 119(2), 196-214.
- Selinger, S. (1980), "Construction Planning for Linear Projects", *Journal of the Construction Division*, ASCE, 106(CO2), 195-205.
- Senior, B. A. (1993), A Study of The Planning and Integrated Cyclic Analysis of Serial System Operations, Ph.D. Thesis, Purdue University, West Lafayette, IN.
- Senior, B. A. (1995) "Late-Time Computation for Task Chains Using Discrete-Event Simulation", *Journal of Construction Engineering and Management*, ASCE, 121(4), 397-403.
- Stradal, O., and Cacha, J. (1982), "Time Space Scheduling Method", *Journal of the Construction Division*, ASCE, 108(CO3), 445-457.
- 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.
- Thabet, W.Y., and Beliveau, Y.J. (1994), "HVLS: Horizontal and Vertical Logic Scheduling for Multistory Projects", *Journal of Construction Engineering and Management*, ASCE, 120(4), 875-892.

Thabet, W.Y., and Beliveau, Y.J. (1994), "Modeling Work Space to Schedule Repetitive Floors in Multistory Buildings", *Journal of Construction Engineering and Management*, ASCE, 120(1), 62-116.

Tokdemir, O. B. (2003), *ALISS: Advance Linear Scheduling System*, Ph.D. Dissertation, Illinois Institute of Technology, Chicago, IL.

Tovakoli A. (1985), "Productivity Analysis of Construction Operations", *Journal of Construction Engineering and Management*, ASCE, 111(1), 31-39.

Yang, I. (2002), *Repetitive Project Planner Resource-Driven Scheduling for Repetitive Construction Projects*, Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.