MANAGING PROJECTS WITH STOCHASTIC DURATIONS

by

Karolina J. Glowacka

BS Business Administration, University of Tennessee at Chattanooga, 2000

Submitted to the Graduate Faculty of

Joseph M. Katz Graduate School of Business in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2008

UNIVERSITY OF PITTSBURGH

KATZ GRADUATE SCHOOL OF BUSINESS

This dissertation was presented

by

Karolina J. Glowacka

It was defended on

May 6, 2008

and approved by

G.G. Hegde, Associate Professor of Business Administration, KGSB

Timothy Lowe, Professor of Operations Management, Tippie College of Business, University of Iowa

Prakash Mirchandani, Professor of Business Administration, KGSB

Jennifer Shang, Associate Professor of Business Administration, KGSB

Dissertation Advisor: Richard E. Wendell, Professor of Business Administration, KGSB

MANAGING PROJECTS WITH STOCHASTIC DURATIONS

Karolina Glowacka, PhD

University of Pittsburgh, 2008

This research addresses analyzing and responding to uncertainty in projects with stochastic task durations. First we examine the effect of contractor flexibility (or agility) on project completion times. We find that this impact can be significant depending on the size and the structure of the project network.

Next we study a stochastic time-cost trade-off problem with penalties for exceeding a project deadline. In considering this problem, we take a contingency approach to decision making where crashing decisions are made dynamically throughout project execution. For serial projects we develop a dynamic programming algorithm as well as a variety of heuristic methods. Extending and modifying these methods for general projects allows us to deal with more complex network structures. Specifically, we propose hybrid dynamic programming/linear programming algorithms and simulation-based algorithms. We perform computational studies to assess the performance of each method and to compare the contingency approach with a static approach (where all crashing decisions are made before the project start time).

Finally, we study the case with penalties, incentives, and overhead costs. We find that when the project cost function is not convex, the dynamic programming solution may become non-monotonic, which requires further modification of the methods. We show that the performance of our algorithms does not deteriorate with inclusion of additional parameters. In fact, the gaps

between the case with perfect information and the methods presented herein seem to be smaller than in the penalty only case.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

This dissertation would not have been possible without the help and support of many people, both on the academic field and in my personal life. I would like to thank my committee members for their patience, understanding, and support. I extend special thanks to my advisor, Dr. Richard Wendell, and Dr. Timothy Lowe who provided me with many ideas that became a part of this dissertation. I also would like to thank Dr. Jerry May for helping me clarify my thoughts on many occasions. My deepest gratitude goes to Carrie and the rest of the doctoral office staff for keeping me sane all these years.

I am also very grateful to my family for their moral support, Gabe for getting me drunk and helping me temporarily forget about a pile of work waiting to be done, Ray for his guidance, Robo for keeping me entertained, and Booby Monkey for being Booby Monkey. Last but not least, thanks to the CBA students who made teaching enjoyable and put up with my insane schedule and my sometimes insane lectures.

1.0 CHAPTER ONE: INTRODUCTION

1.1 OVERVIEW

A simple definition of a project is a temporary and unique endeavor consisting of tasks that, when completed, yields a deliverable of some sort, whether it is a finished information system, a building, a bridge, or a dissertation. A project has a definite beginning and a definite end, but in order to get to this "end" successfully we need to manage the project effectively. The Project Management Institute (PMI) defines project management as "the application of knowledge, skills, tools and techniques to a broad range of activities in order to meet the requirements of a particular project." So how do we measure project success? While there are many metrics by which to measure project success, such as the quality of the work or customer satisfaction; the scope this dissertation will focus on two factors that we can clearly measure: time and cost.

In order to estimate a project's completion time, we need to develop a schedule. A project schedule is a plan of what needs to be done in order to complete all the work. Project managers define tasks to be completed and precedence relationships or dependencies (which tasks follow other tasks). We can also visualize a project as a network diagram that indicates what tasks need to be done (represented by nodes) and in what order. Directed arcs among project nodes correspond to precedence relationships. The length of the project is defined as the length of the longest path in the network from start to finish; also known as the critical path. In this

dissertation we will use an activity on node (AON) network diagram to represent various projects. In an AON diagram nodes correspond to project activities and arcs represent precedence relationships.

It is common knowledge that projects are often late. There are multiple reasons for this phenomenon and many of those have their roots in uncertainty. Uncertainty in projects can be present in many forms: uncertainty in what resources will be available, uncertainty in when they will be available, uncertainty in the exact scope of the work, uncertainty in the precedence relationships, and uncertainty in the duration of the tasks. Nevertheless, project managers are usually forced to make estimates regarding project completion times and final costs before any uncertainty is resolved. These estimates often become a basis for bidding on project contracts – setting project target date (planned project completion date), start times for project tasks, as well as relevant charges such as contractor revenues, penalties for exceeding the target date, overhead costs; but also incentives for completing the project early. Therefore late projects can lead to higher costs and thus decreased profits for both contractors and clients.

In this dissertation we focus on analyzing uncertainty in task durations and project crashing in the presence of this uncertainty. Project crashing refers to shortening the duration of the project by allocating additional resources (money) to some of the tasks. This leads to a time-cost trade-off – by spending more resources on the project now, we can avoid or lessen the consequences of completing the project late. This can result in cost savings because we can reduce the total amount of penalties (or avoid them altogether) and even take advantage of incentives if there are provisions for such.

1.2 PROBLEM STATEMENT

The first problem considered in this thesis relates to the flexibility of contractors and its effects on project duration. We refer to this flexibility as forward agility and define it as the ability to start a task before its scheduled start time if its predecessors finish sooner than planned. If an activity can take advantage of the early completion of its predecessors, we call it forward agile. Otherwise, the activity is considered not forward agile, which is equivalent to imposing a "start-no-earlier-than" constraint on that task.

The second problem can be described as a stochastic time-cost trade-off where we have options to speed up (crash) some or all of the activities for a given cost, which we refer to as the crash cost. The problem can be stated as follows: Given a set of activities, their precedence relationships, probability distributions of their durations, crash costs and maximum speed-up associated with each task, project target date, per period penalty for exceeding the target date, as well as any other relevant costs (incentives and/or overhead), find a crashing policy that will minimize total project cost. In this thesis we assume that the duration of each task follows a triangular probability distribution with parameters given by the optimistic (O), the most likely (ML), and the pessimistic (P) durations. Traditionally, project managers have used PERT (Program Evaluation and Review Technique) to deal with such a problem. PERT was developed between 1956 and 1958 for the U.S. Navy's Special Project Office (Klastorin, 2004). Despite its wide adoption in industry, PERT suffers from two critical shortcomings: (1) it replaces uncertainty with point estimates (averages) for task durations and (2) it does not allow for dynamic decisions. In this dissertation we will describe methods that address these two issues resulting in better crashing policies. Furthermore, we consider two types of uncertainty in task durations: internal uncertainty and external uncertainty (Elmaghraby, 2005). Internal uncertainty

stems from the difficulty in estimating work content of the tasks, that is, we do not exactly know how much effort each activity will require. External uncertainty is related to external events such as weather changes, catastrophic events (earthquakes, tornadoes), etc. As we will show in later chapters, external uncertainty often results in linear crash costs while internal uncertainty results in nonlinear crash costs.

1.3 PROJECT REPRESENTATION

We visualize projects using a standard activity-on-node (AON) representation. That is, a project network is a direct acyclic graph with nodes representing activities and arcs corresponding to precedence relationships among project tasks, with two dummy activities denoting the start and the finish of the project respectively. Furthermore, in this research we classify project network topology (or structure) using two measures – order strength (OS) and serial-parallel (SP) index.

The idea of order strength was developed by Mastor (1970) and further described by Demeulemeester et al. (2003). The OS uses the concept of transitive and non-transitive arcs. We call an arc between two nodes, *i* and *j*, transitive if there exist *i*, *j* and *k* such that there is a path from *i* to *k* and a path from *k* to *j*. In other words, transitive arcs represent redundant precedence relationships. The OS is defined as the total number of precedence relationships in the network, including transitive but excluding dummy relationships, divided by the theoretical maximum number of precedence relationships. This latter number is equal to $n(n-1)/2$ where n equals the number of activities in the project excluding the two dummy nodes. Therefore, it measures the "density" of the project with respect to precedence relationships. On the other hand, the serial-parallel index (Tavares et al, 1999) measures the "length" of the longest path in the project and is

equal to (m-1)/(n-1) where m is the length of the longest path in terms of the number of activities and n is the total number of activities. Unlike the OS, the SP index does not indicate how many precedence relationships exist in among the activities. We provide a more thorough discussion about these measures in Chapter 2 (SP index) and Chapter 4 (OS).

In this thesis we often consider two special project network structures – a completely serial project, where there exists only one path and a completely parallel project where the number of paths is equal to the number of activities and no precedence relationships exist except for the dummy ones.

1.4 PURPOSE, SCOPE, AND OBJECTIVES OF THE RESEARCH

There are two goals of this research. First, we analyze the effects of forward agility on the project duration and the probability of a late completion. Surprisingly, this topic has been neglected in prior research on project management. We analyze its impact and show that the lack of forward agility can significantly affect the probability of the project finishing late. This dissertation also addresses the subject of conditional decision making in projects with stochastic task durations. There exists a moderate amount of work that deals with the development of static crashing policies for projects with uncertain durations; however, the literature on dynamic policies is scant. One reason for this is that it is difficult to develop exact dynamic algorithms for such a problem – the sheer number of possible scenarios makes the solution space so large that it prohibits enumeration of all solutions. There is, however, one class of problems for which we can calculate optimal dynamic policies – serial projects. For serial project networks we can also develop efficient heuristics, which we will show get very close to the optimum. We then extend

those heuristics to the general network (i.e., not serial) case and test them on a variety of project network topologies. We also calculate the value of perfect information for both serial and general projects and show that the gap between the cost with perfect information and the cost of optimal crashing policy with uncertainty for serial networks is comparable to the gap between perfect information cost and best performing heuristics for general networks.

A contribution of this research is to develop efficient algorithms for dynamic crashing policies in projects with stochastic task durations. In addition, we will show how our methods extend to cases with nonlinear crash costs as well as cases when there are incentives (negative costs) for completing the project early. For all computational tests we used a 3.2 GHz Intel Pentium 4 PC with 1GB RAM, running Windows XP operating system. For the methods requiring an LP/IP solver we used the Common Optimization Interface for Operations Research open source solvers (Lougee-Heimer, 2003). All algorithms presented herein were implemented in C++ using Microsoft Visual Studio .NET development environment.

1.5 CONTRIBUTIONS OF THIS RESEARCH

In this thesis, we consider a previously neglected issue of contractor flexibility, which we refer to as the forward agility, and its impact on project delays. We show how this effect varies with respect to the project structure and the due date. Our results can serve as a guideline for project managers negotiating contracts with subcontractors and setting project target completion dates. In addition, we show that forward agility can have an impact even in projects with multiple parallel paths but this effect depends on the project due date. We also study how to make dynamic crashing decisions in the presence of uncertain task durations and derive additional

insights by studying serial projects. We examine the applicability of dynamic programming for serial project networks and develop other efficient algorithms for more general networks. These methods can help managers make speed-up decisions that depend on the state of the project. We compare such dynamic policies with static speed-up decisions. Algorithms that use simple rules of thumb as well as more sophisticated methods that combine dynamic programming with linear programming are discussed. In the end, we find that simple methods usually work well and that one should avoid focusing too much on the critical path in making speed-up decisions. Finally, we extend our algorithms to projects with incentives, penalties, and overhead cost and discuss additional difficulties in solving such problems, such as having non-convex cost functions or non-monotonic dynamic programming solutions.

1.6 OUTLINE OF THE DISSERTATION

In Chapter 2 we consider effects of the forward agility of contractors on project completion time. Chapter 3 looks at a special case of serial projects with linear crash costs and a penalty for exceeding the target date. We discuss why serial projects are important and present a network generator for creating random serial test problems with certain characteristics. Four algorithms are discussed, including a dynamic programming method yielding optimal policies, and results on random test problems are presented. Chapter 4 considers extensions from serial projects to a more general case. We present another network generator for creating test problems with different topologies (from a completely parallel project to a serial project) and introduce five robust heuristics for solving these problems. We present and compare the results as well as discuss advantages and disadvantages of each method. Chapter 5 looks at extensions to the

problem with penalties by introducing incentives and overhead costs. We show that the project cost function may become non-convex in which case we have to make major changes to the algorithms. Nevertheless, our methods are capable of handling this case as well for both serial and general networks.

2.0 CHAPTER TWO: IMPACT OF AGILITY

2.1 INTRODUCTION

Projects are naturally prone to late completions in that a delay in one activity can delay the start of its successors – especially on a critical path. We refer to this ability to start tasks later than planned as backward agility. Unfortunately, projects are often not forward agile. That is, if a predecessor takes less time than estimated, then its successor often starts at its normally scheduled time (failing to capture the benefits of starting early). This phenomenon was noted by Goldratt (1997) in his book Critical Chain (see page 112): "A delay in one step is passed, in full, to the next step. An advance in one step is usually wasted." While we could find no study addressing this phenomenon in practice, a recent survey by Assaf and Hehhi (2006) examining the causes for delays in large construction projects identifies "inflexibility of contractors" as a major factor. Surprisingly, from our review of the literature, forward agility has not been studied. Thus this chapter will consider the impact of agility in projects.

The chapter proceeds as follows. Section 2.2 gives the basic definitions and assumptions. Section 2.3 considers serial projects and analyzes the impact of agility in terms of project size (number of tasks). The impact of agility in projects with parallel paths is considered in section 2.4 by examining a special class of projects, called strongly parallel projects. Section 2.5 considers agility in general projects using a serial-parallel index to characterize project structure.

Finally, section 2.6 concludes with a summary of the results on the impact of agility in projects. It also gives some observations on identifying and characterizing tasks having a high agility impact in a project.

2.2 DEFINITIONS AND ASSUMPTIONS

If all activities in a project had deterministic durations, we could easily calculate the project completion date using the Critical Path Method (CPM) analysis. Even though in reality there are uncertainties with respect to activity durations, project managers often reduce a stochastic project to its deterministic counterpart using best-guess estimates. Often those estimates are the expected or most likely durations of the activities. The project completion date and schedule are also calculated using the CPM, which gives planned start times for all activities as well as planned project completion date, which we refer to as Target.

However, if activity durations are probabilistic, each activity can take less or more time than estimated. When an activity finishes before the planned start time of its successor, the successor may or may not be able to take advantage of this opportunity by starting early. If an activity can reschedule its start time to take advantage of an earlier finish of its predecessor(s), then we say that it is forward agile. In this research we study the impact of such agility. Specifically, we examine two cases – projects in which all activities are forward agile (100% forward agility) and projects in which no tasks are forward agile (0% forward agility); and we consider the impact of agility on projects with various network topologies. The effects of network topology on project delays were first discussed by Schonberger (1981). The most notable finding was that the difference between the true and estimated durations varies, based on

the project network structure and individual activity duration variance – the "fatter" or more parallel the project, the later the project completion and the higher the variance.

In order to focus attention solely on the effects of agility under different network topologies, we assume independent and identical probability distributions of the durations of all activities. Furthermore, for simplicity in the computational comparisons, we assume that this distribution is triangular. Below is a summary of the notation used herein.

- *Target* – project target date set using the CPM method.
- *n* – total number of activities in a project
- *sigma* – standard deviation of the project assuming 100% forward agility. Computed using simulation.
- $\sigma^2_{activity}$ – variance of activity duration
- *c* – fraction of the standard deviation of the project above *Target*.
- *O* – optimistic activity duration.
- *ML* – most likely activity duration.
- *P* – pessimistic activity duration.

2.3 THE SERIAL PROJECT CASE

We first examine projects with one path (serial projects). The impact of the agility on durations of serial projects is important to study because the effect of having multiple paths is removed. Therefore, we can observe pure effects of agility, not confounded by other factors. In practice, serial projects effectively exist when there is one dominant path. Interestingly, some applications in scheduling trains, busses, and planes naturally correspond to serial projects.

For simplicity, in most cases we assume that the duration of each activity is given by the triangular distribution with *O*=5, *ML*=10, and *P*=15. However, when we examine the impact of agility with respect to the standard deviation of activity durations, we still use symmetric triangular distributions (where *P-ML* = *ML-O*) but we vary the distribution span, or the difference between the pessimistic and optimistic values, between 0 and 10. The Target of each project is set using the CPM and assuming expected activity durations. In addition, when examining varying distribution spans we set the project size to 20 activities. Figure 2-1 shows probability density functions of the five different triangular distributions examined herein (described by their *O*, *ML*, and *P* parameters) as well as variances of activity durations corresponding to these spans.

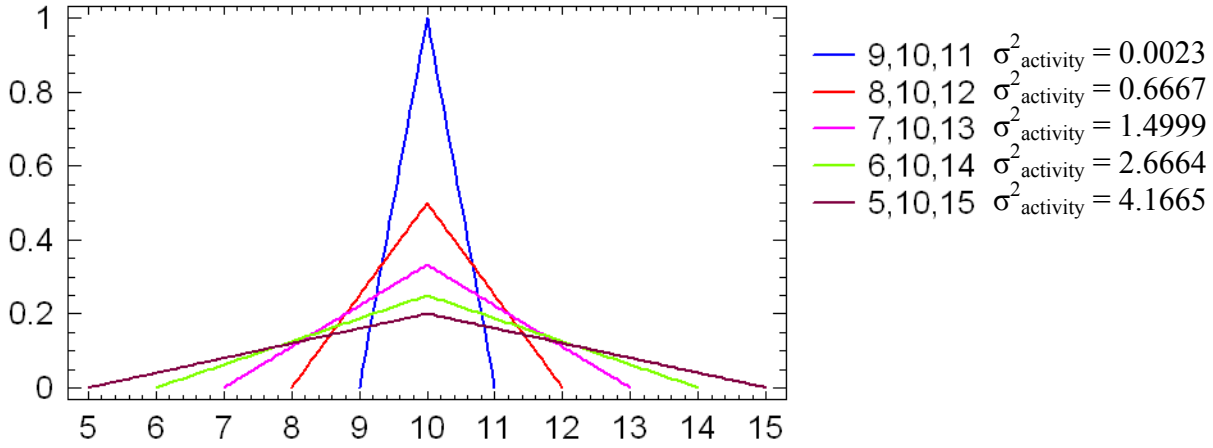


Figure 2-1: Various symmetric triangular distributions

We calculate the average measures presented herein (mean duration and probabilities) by simulating each project *N* times where *N* is equal to the number of activities in that project multiplied by a constant (100).

First we examine the impact of agility on the likelihood of finishing the project before some date specified with respect to project size. Since we know that, even with full agility and only one path, using the CPM method will, on average, underestimate project duration 50% of

the time (assuming symmetric distributions), we examine probabilities of completing the project before *Target*+*c*×*sigma* where *c* can take on values 0, 0.25, 0.5, 0.75, 1, 1.5, 2, 2.5, and 3. The results are presented in Figures 2-2 through 2-10. In addition, we also present the 95% confidence intervals for the differences between probabilities using a method described by Marascuilo and McSweeney (1977). It is important to note that the lack of monotonicity in some of the curves presented in this section and section 2.4 is related to the number of simulation replications performed. With a larger number of replications these curves become monotonic.

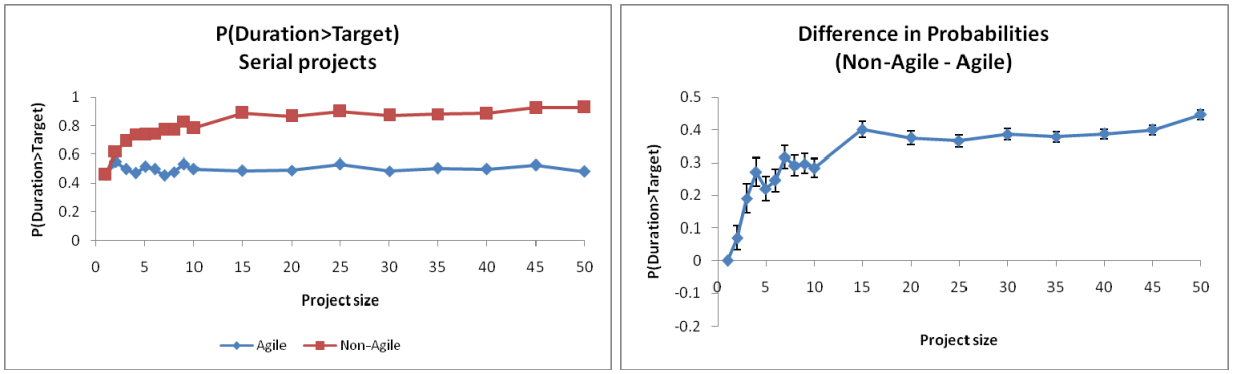


Figure 2-2: P(Duration>Target) by size - serial projects

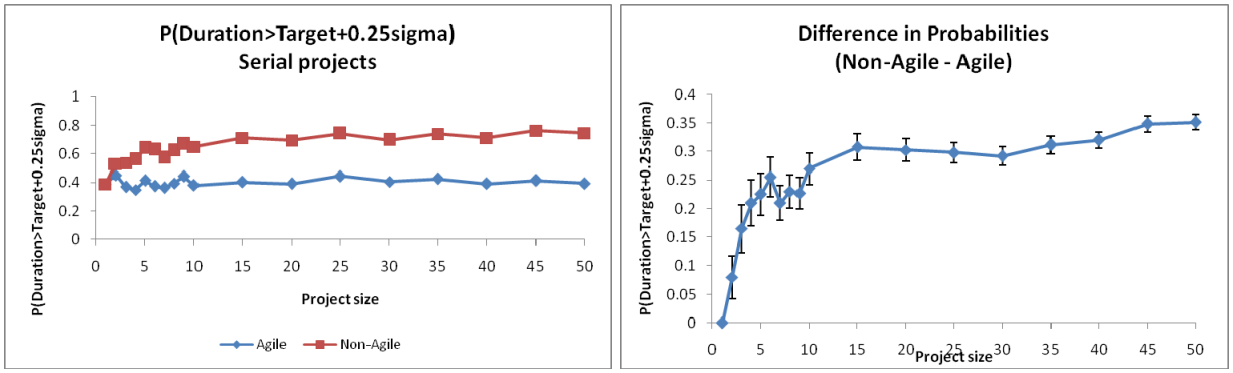


Figure 2-3: P(Duration>Target+0.25sigma) by size - serial projects

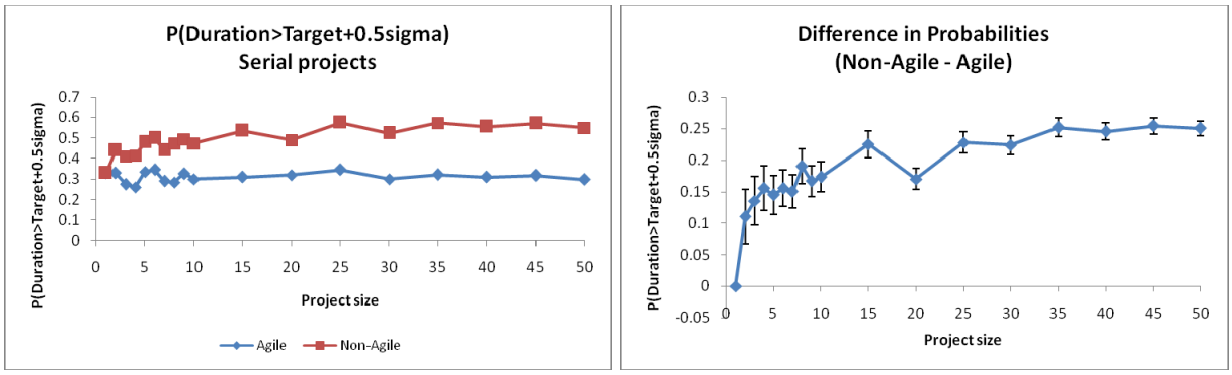


Figure 2-4: P(Duration>Target+0.5sigma) by size - serial projects

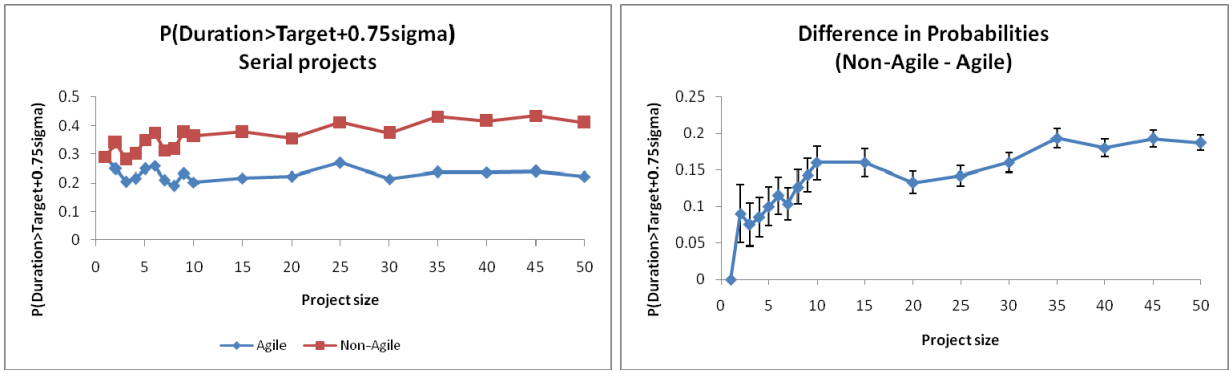


Figure 2-5: P(Duration>Target+0.75sigma) by size - serial projects

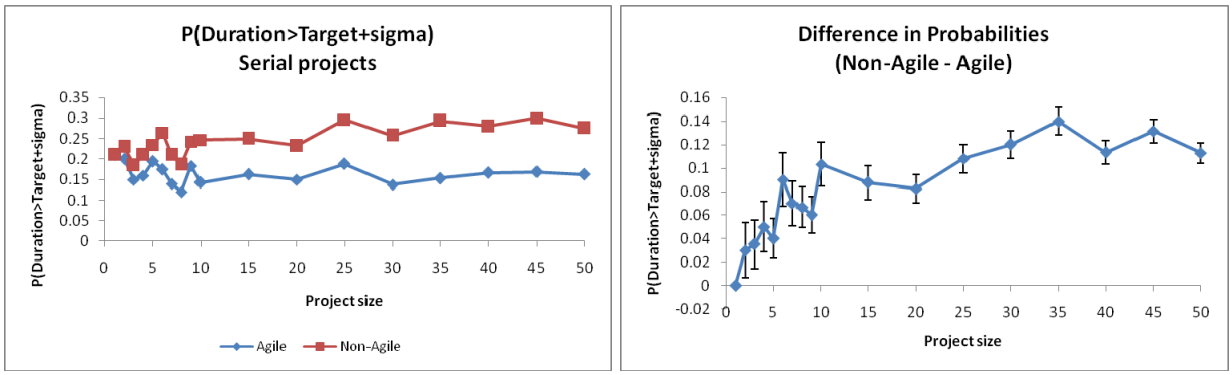


Figure 2-6: P(Duration>Target+sigma) by size - serial projects

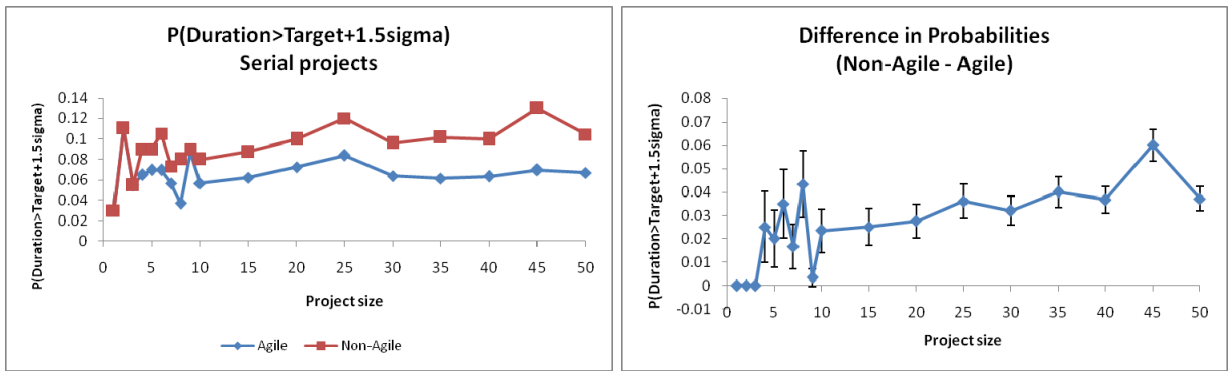


Figure 2-7: P(Duration>Target+1.5sigma) by size - serial projects

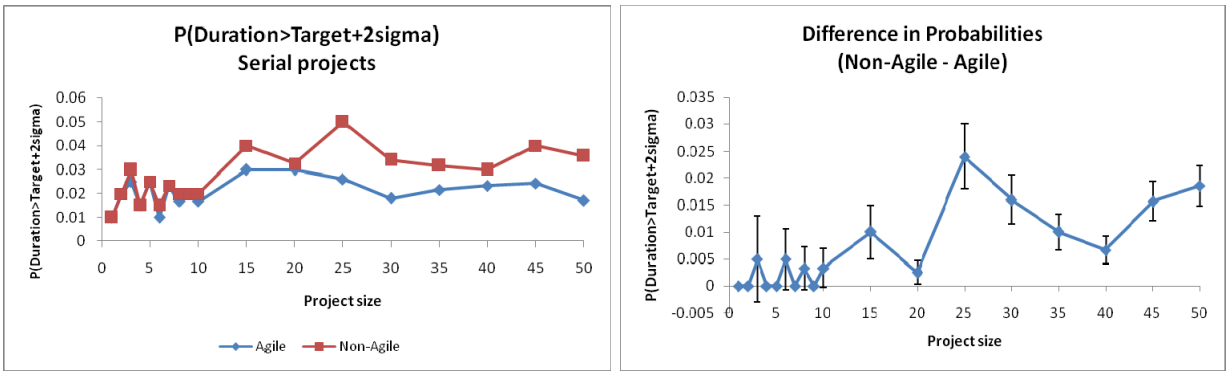


Figure 2-8: P(Duration>Target+2sigma) by size - serial projects

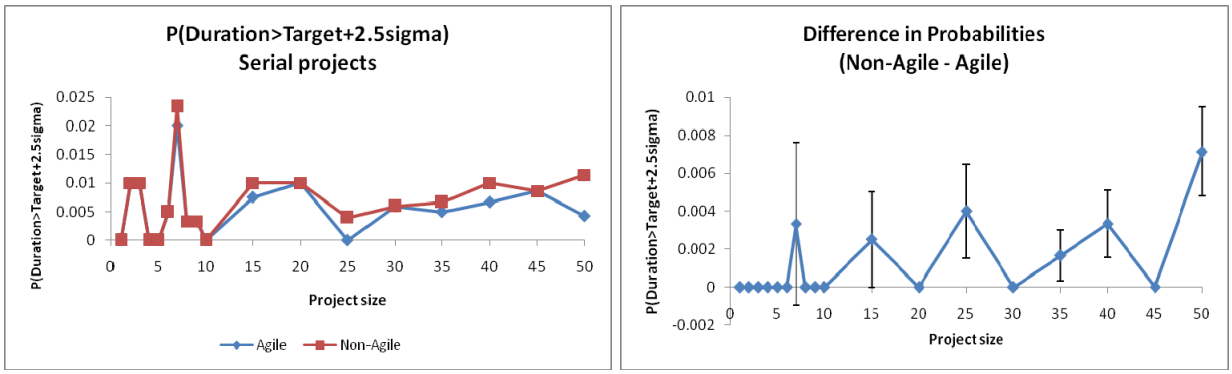


Figure 2-9: P(Duration>Target+2.5sigma) by size - serial projects

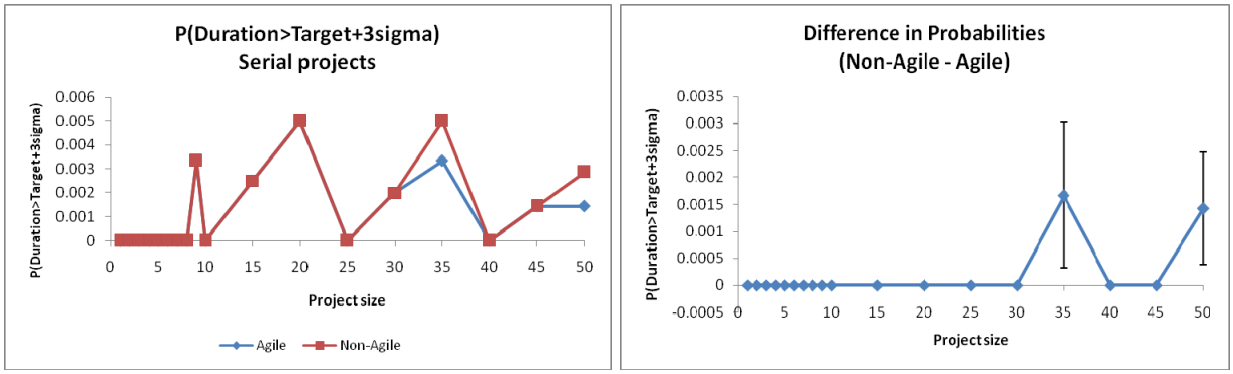


Figure 2-10: P(Duration>Target+3sigma) by size - serial projects

The probability that a fully forward agile project will exceed a specified due date (derived based on the number of activities) remains the same regardless of the project size. In fact, that probability can be easily calculated for agile projects, for instance, when probability distributions of activity durations are Tri(5,10,15), P(Duration>*Target*) = 0.5, P(Duration>*Target*+0.5*sigma*) = 0.3167, and P(Duration>*Target*+*sigma*) = 0.175 for projects with one activity. This is

calculated by taking the integral of the triangular probability density fucntion: $\int_x^{\infty} \frac{2(P-x)}{(P-O)(M-O)} dx$. In projects with a larger number of activities, we can approximate the distribution of the project duration as normal and calculate the probabilities as P(Duration>*Target*) = 0.5, P(Duration>*Target*+0.5*sigma*) = 0.3085, and P(Duration>*Target*+*sigma*) = 0.1587 from the standard normal table. In the non-agile case however, the number of activities (*n*) clearly makes a difference, which is not surprising given our analysis for the mean. The P(Duration>*Target*) in the non-agile case seems to approach 1 as the number of activities increases (therefore the difference between the non-agile and agile probabilities approach 0.5). Similar patterns are observable for different values of *c*. It is interesting to note that, even when we increase the due date by a full standard deviation, the impact of the lack of forward agility is still significant at a 95% confidence level. However, as *c* increases, the impact of agility becomes less visible.

Next we look at the impact of agility on the mean duration of a project. Figure 2-11 presents the average project duration for the agile and non-agile cases as well as the difference between the two with respect to the project size. Figure 2-12 shows the same measure with respect to the distribution span.

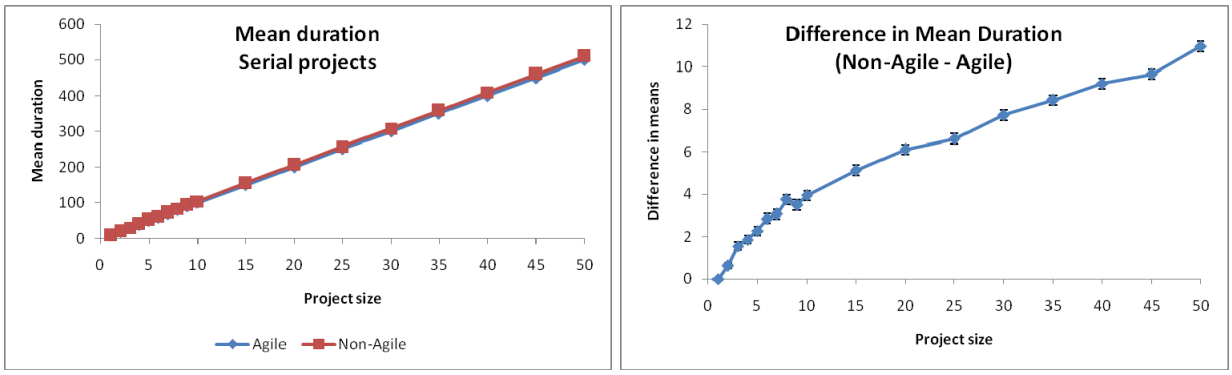


Figure 2-11: Mean comparison by size- serial projects

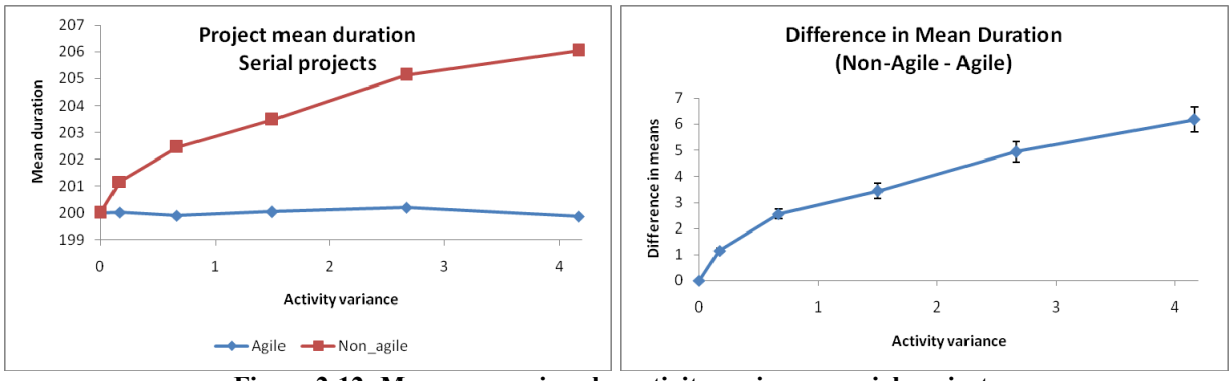


Figure 2-12: Mean comparison by activity variance - serial projects

Not surprisingly, as the number of activities in the project increases, so does the effect of the agility on the average duration. When there is only one activity in the project, the actual start time is always the same as the planned start time of that activity. The greater the difference between the minimum possible start time of the activity, the greater the impact of the lack of forward agility. For the i^{th} activity, the minimum possible start time is $\sum_{l=1}^{i-1} O_l$ and the planned start time is $\sum_{l=1}^{i-1} ML_l$; therefore, the greater the number of terms in the summation or the greater the difference between the most likely and the optimistic durations, the more severe the consequences of not having forward agility. Note that, even though the absolute difference in the mean durations with respect to size increases with the number of activities, the average percentage difference seems to decrease slightly. In projects studied (Figure 2-11), for two or three activities, that difference is about 5% and decreases to about 2% for projects with 50 tasks. However, the percentage difference (as well as the absolute difference) with respect to the distribution span increases with the span increase.

2.4 STRONGLY PARALLEL PROJECTS

We now turn our attention to projects with multiple paths. In addition to the notation specified in Section 2.2, we define p as the number of parallel paths in the project and a as the number of activities on each path. Specifically, consider a project having p paths with $p \geq 2$. We say that such a project is strongly parallel if each activity (excluding the dummy start and finish activities) is on exactly one path and if each path contains the same number of activities. Figure 2-13 illustrates a project with $p = 4$ and $a = 3$. In this section we examine the impact of project structure on agility by considering strongly parallel projects. Specifically, we examine the agility impact in terms of the number of paths p as well as the number of activities on each path. In a subsequent section we consider more general network structures.

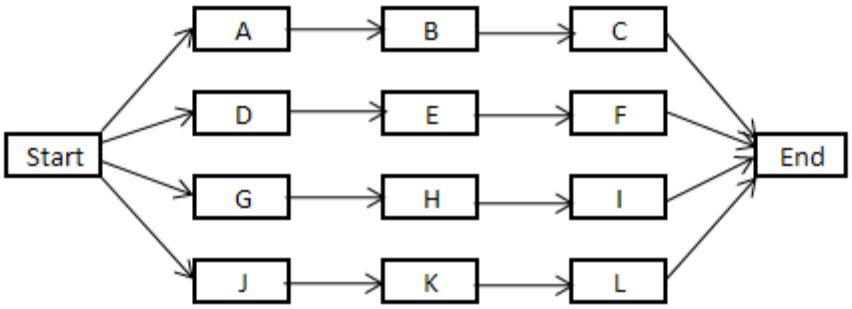


Figure 2-13: Strongly parallel project with p=4 and a=3

As before we perform computational tests assuming, for simplicity, that the duration of each activity is given by a triangular distribution (with O=5, ML=10, P=15, with an exception of examining the impact of standard deviations of the activity durations).

2.4.1 Projects with two strongly parallel paths

We first consider projects with $p = 2$ and varying values of a. We perform similar analyses as in the serial case, that is, we examine differences in probabilities of completing the project before some specified due date as well as differences in mean project durations.

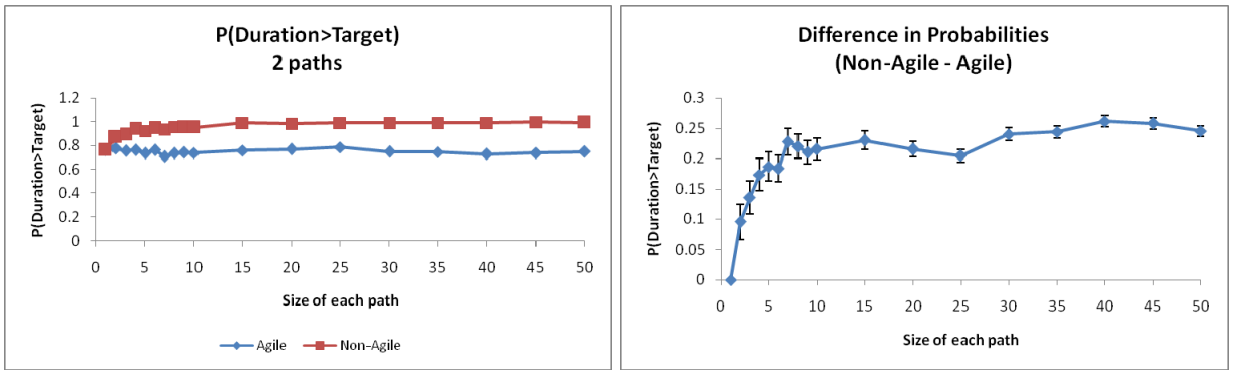


Figure 2-14: P(Duration>Target) by size - 2 paths

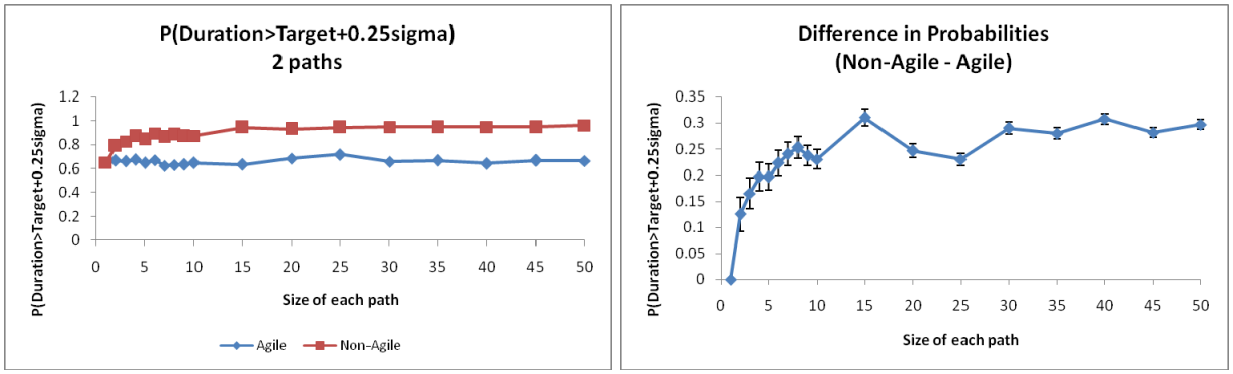


Figure 2-15: P(Duration>Target+0.25sigma) by size - 2 paths

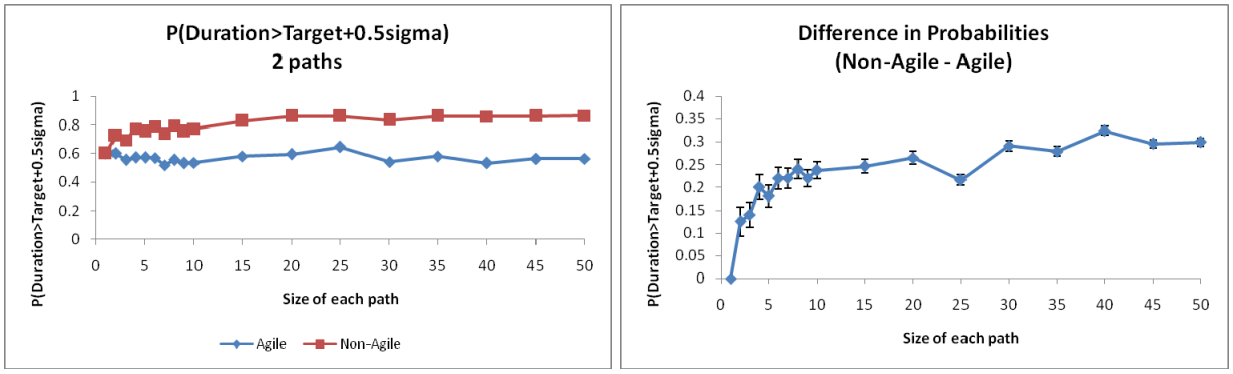


Figure 2-16: P(Duration>Target+0.5sigma) by size - 2 paths

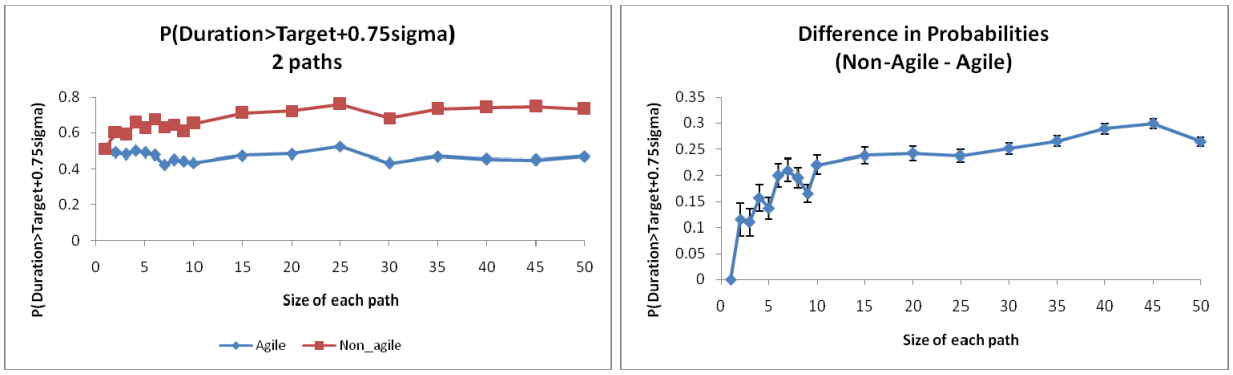


Figure 2-17: P(Duration>Target+0.75sigma) by size - 2 paths

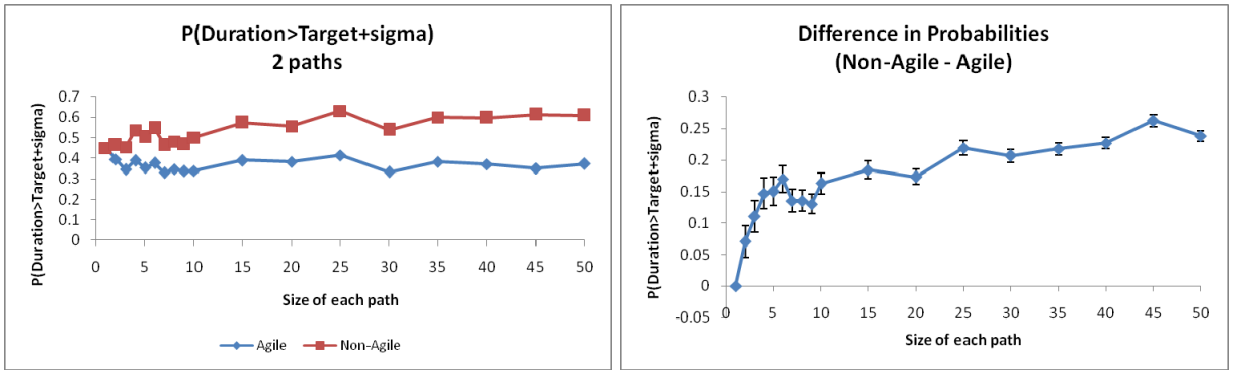


Figure 2-18: P(Duration>Target+sigma) by size - 2 paths

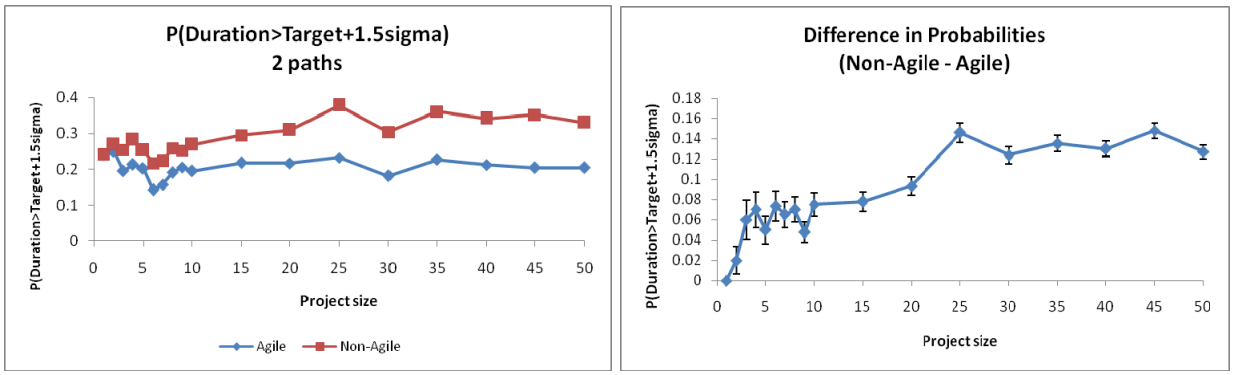


Figure 2-19: P(Duration>Target+1.5sigma) by size - 2 paths

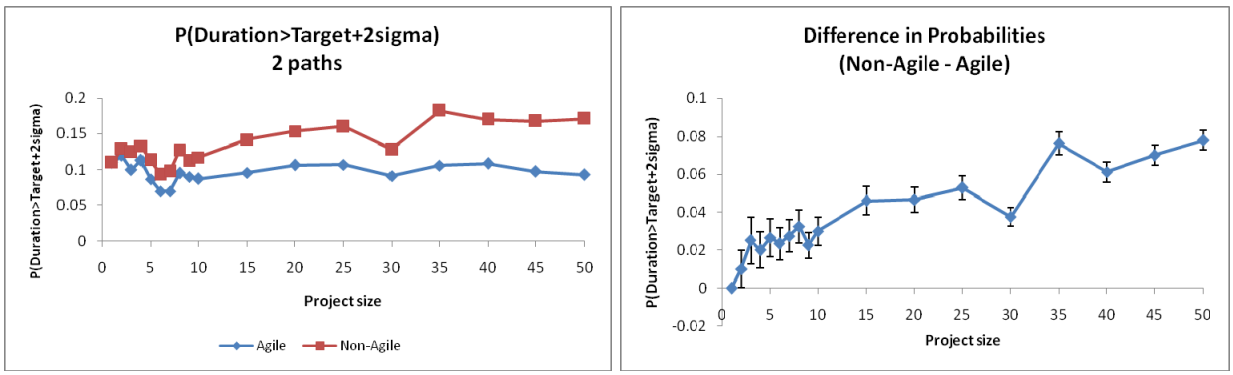


Figure 2-20: P(Duration>Target+2sigma) by size - 2 paths

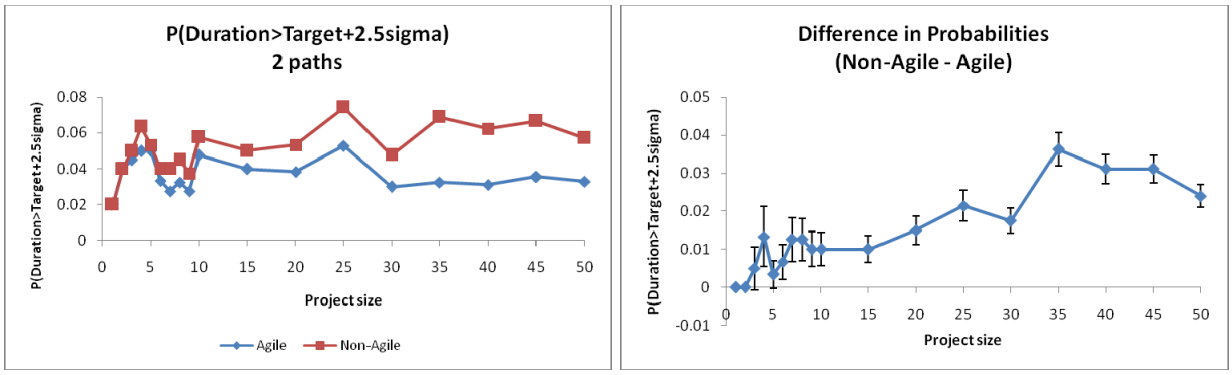


Figure 2-21: P(Duration>Target+2.5sigma) by size - 2 paths

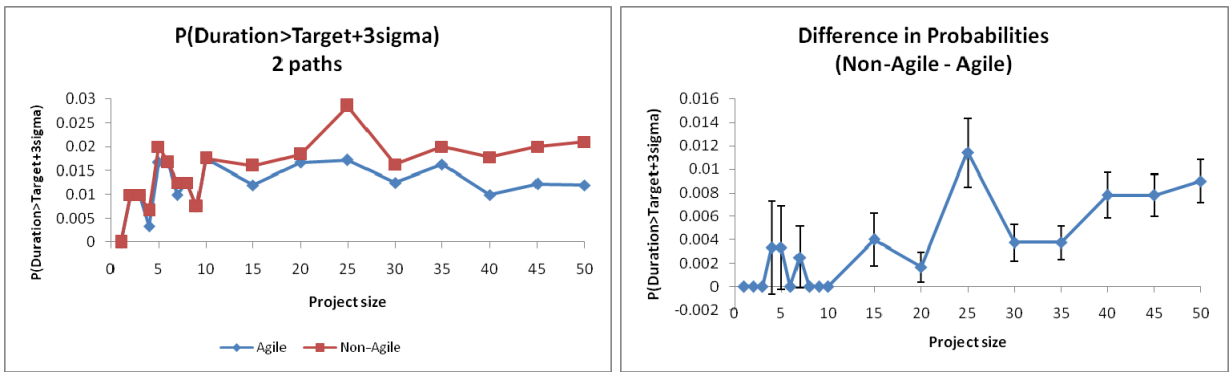


Figure 2-22: P(Duration>Target+3sigma) by size - 2 paths

As before, the probability of completing the project on time remains constant for agile projects regardless of the number of activities on each path. We can show that these probabilities are equal to 1 – P(All paths complete before due date). Therefore, for two paths P(Duration>*Target*) = $1\text{-}0.5^2 = 0.75$, P(Duration>*Target*+0.5*sigma*) = $1\text{-}(1\text{-}0.3085)^2 = 0.5219$, and P(Duration>*Target*+*sigma*) = $1\text{-}(1\text{-}0.1587)^2 = 0.2921$, calculated similarly as in section 2.3. The differences in the probabilities between the agile and the non-agile cases seem to increase for small project sizes and then level off for a greater number of activities. This is caused by the fact that the agile case probabilities remain constant while the non-agile probabilities increase until they reach their upper bound of one. At that point the differences stay at the same level.

Examining the difference in expected project durations (Figures 2-23 and 2-24), we notice similar patterns to those we observed in the serial case; however, they are not as pronounced, that is, the differences between the agile and non-agile cases are slightly smaller.

This is due to the fact that, in addition of having effects of the agility, we also experience impact of multiple paths in the project. This is even more visible when we examine projects with five strongly parallel paths in Section 2.4.2.

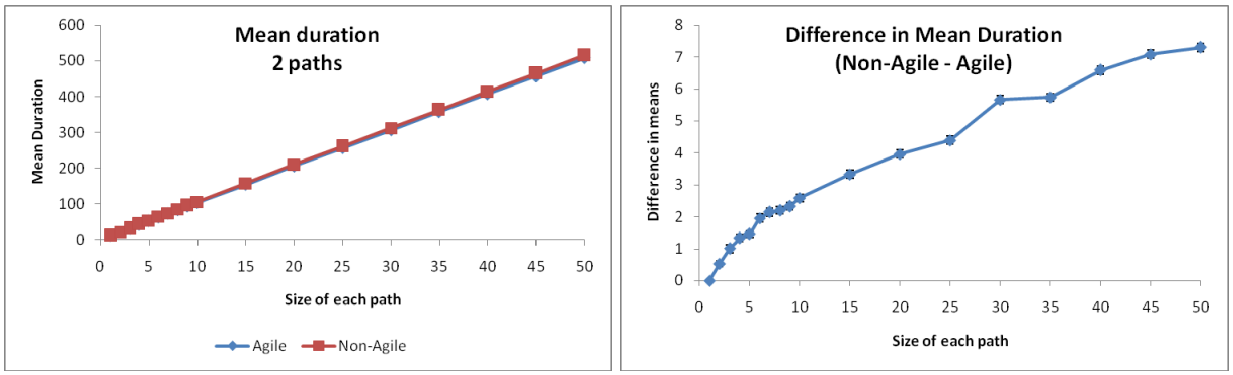


Figure 2-23: Mean comparison by size - 2 paths

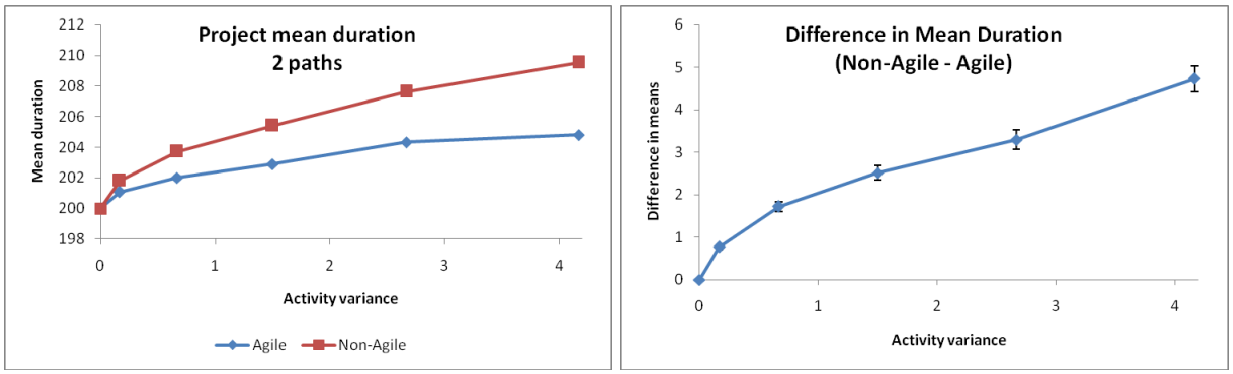


Figure 2-24: Mean comparison by activity variance - 2 paths

2.4.2 Projects with five strongly parallel paths

In order to examine the impact of agility when the number of parallel paths increases, we also looked at the case with five strongly parallel paths. The results are presented in the next set of figures (2-25 through 2-35).

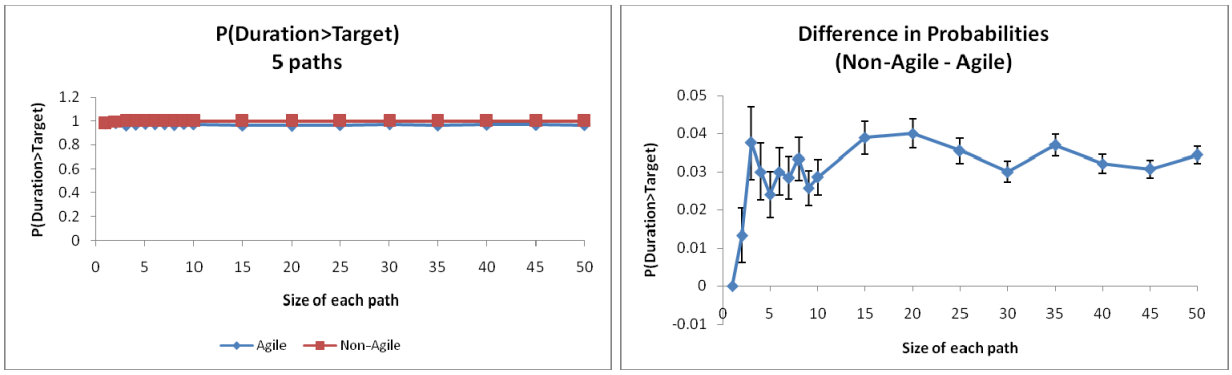


Figure 2-25: P(Duration>Target) by size - 5 paths

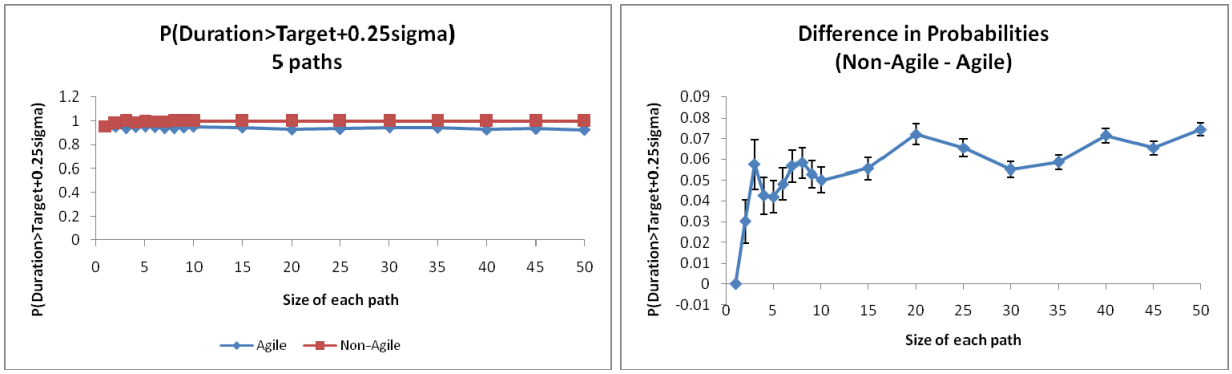


Figure 2-26: P(Duration>Target+0.25sigma) by size - 5 paths

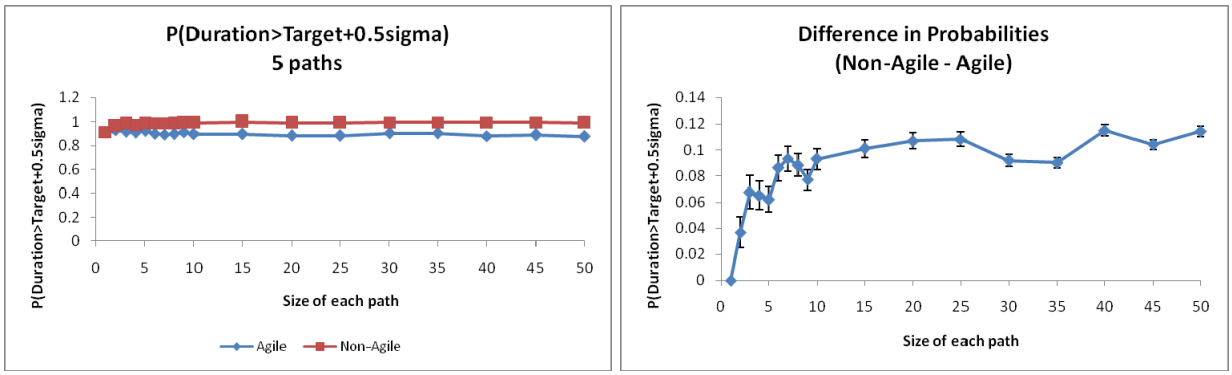


Figure 2-27: P(Duration>Target+0.5sigma) by size - 5 paths

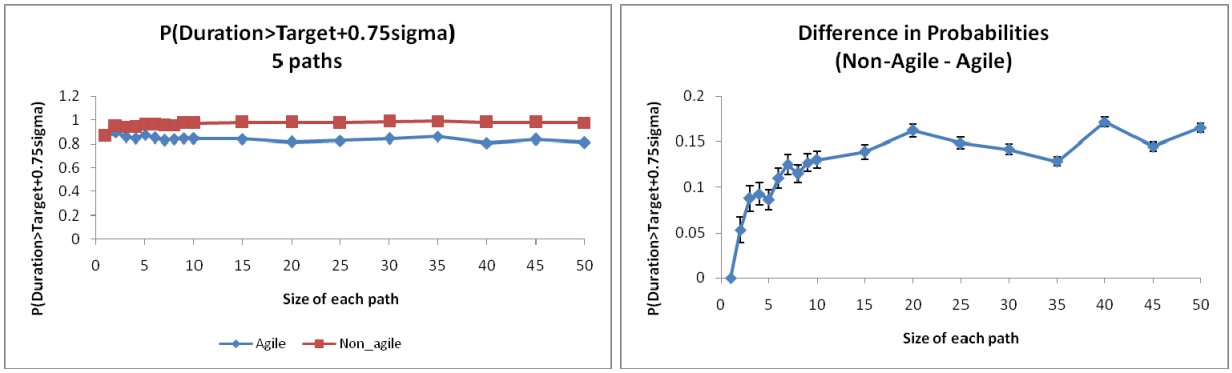


Figure 2-28: P(Duration>Target+0.75sigma) by size - 5 paths

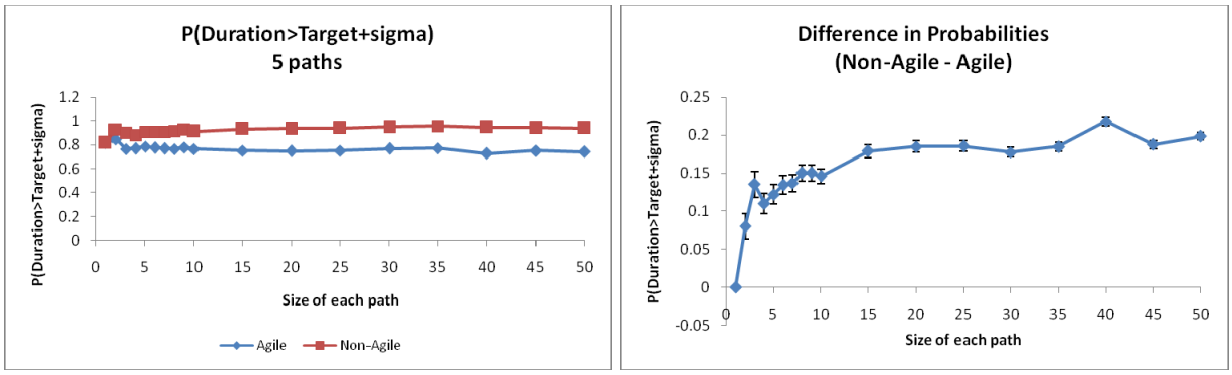


Figure 2-29: P(Duration>Target+sigma) by size - 5 paths

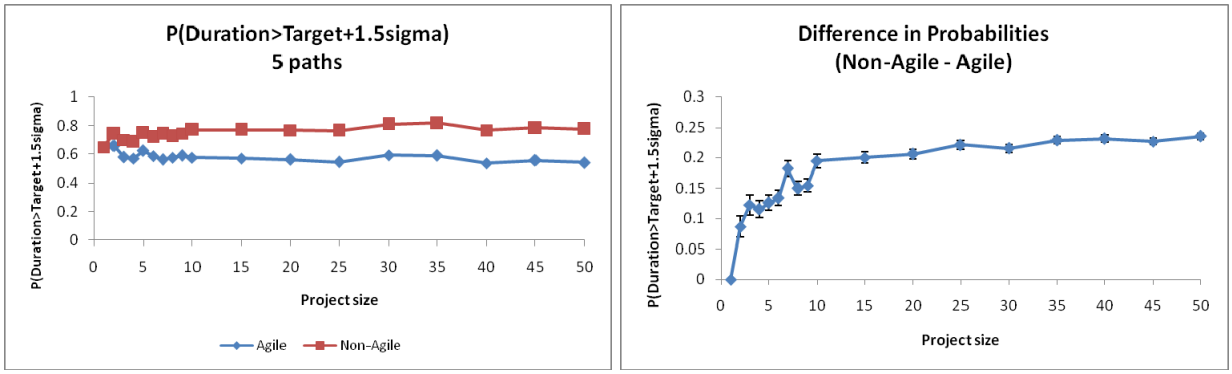


Figure 2-30: P(Duration>Target+1.5sigma) by size - 5 paths

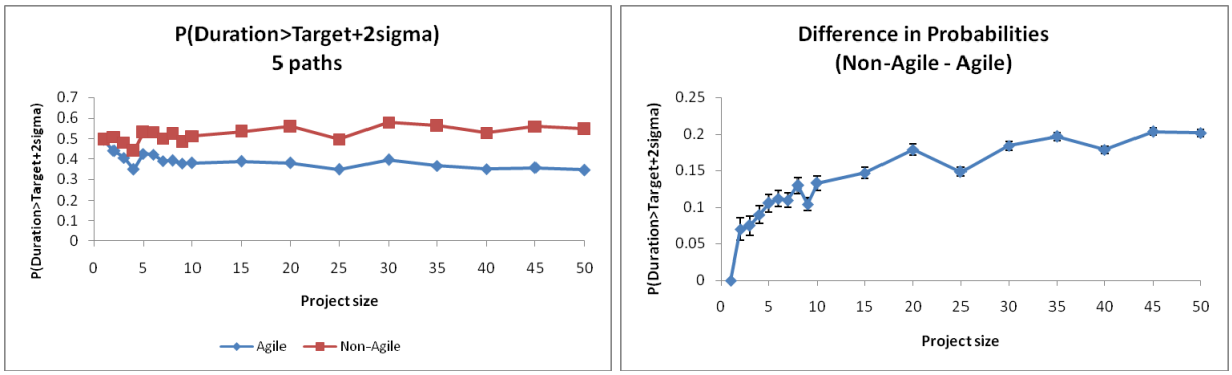


Figure 2-31: P(Duration>Target+2sigma) by size - 5 paths

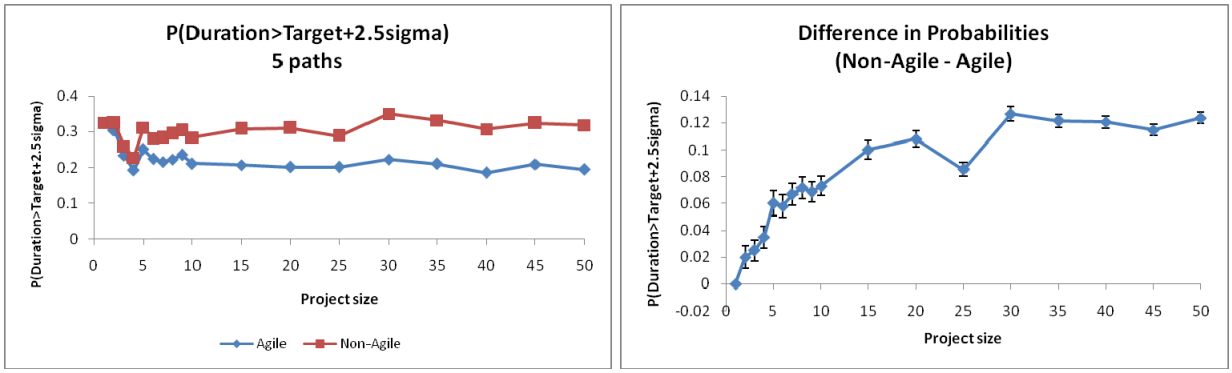


Figure 2-32: P(Duration>Target+2.5sigma) by size - 5 paths

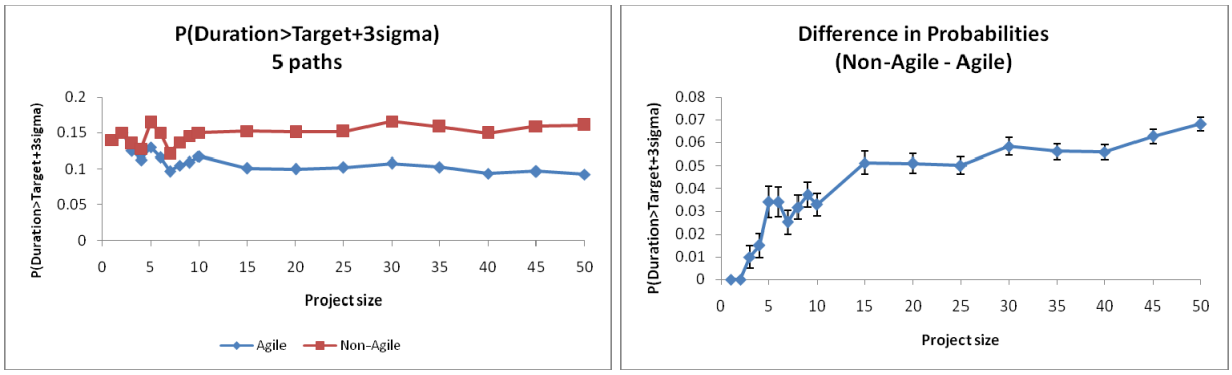


Figure 2-33: P(Duration>Target+3sigma) by size - 5 paths

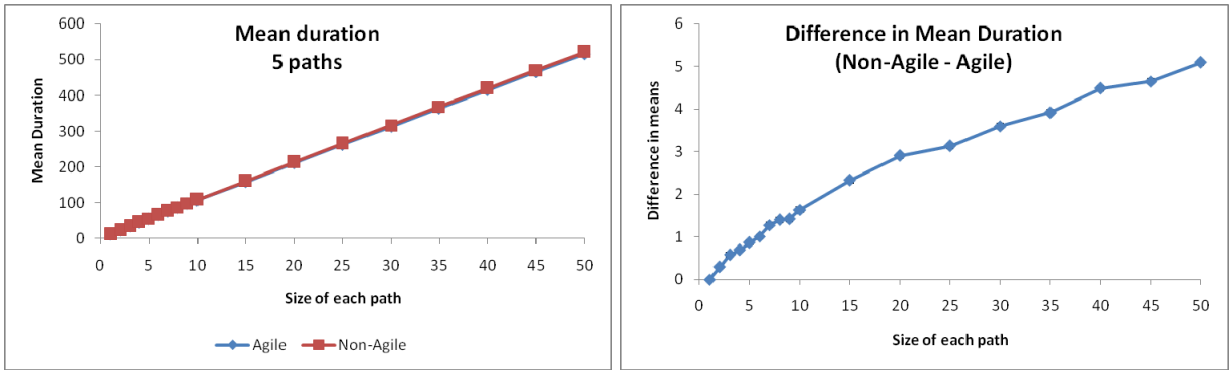


Figure 2-34: Mean comparison by size - 5 paths

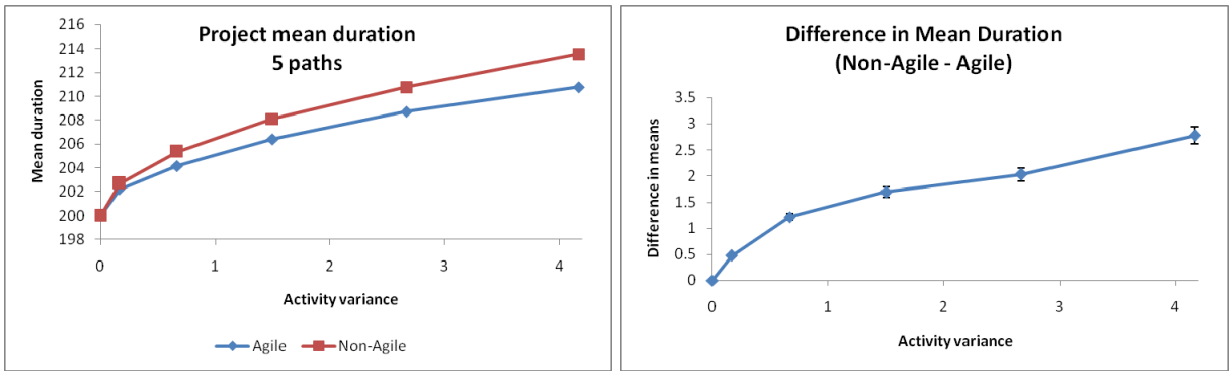


Figure 2-35: Mean comparison by activity variance - 5 paths

There are several notable results here. First, the differences between the agile and the non-agile cases are smaller as compared to the serial case or the case with two strongly parallel paths for small values of *c* (up to 0.75). This again is due to the structure of the project network – the "fatter" the project, the greater the delay so we do not detect such severe impact of the agility as before. This is particularly visible when examining the probabilities of finishing the project before a specified due date. As the number of parallel paths increases, this probability

approaches 1 even for the agile case. Therefore, the differences between the agile and non-agile projects are small. As before we can calculate the exact probabilities for the agile case; specifically, we note that those probabilities are P(Duration>*Target*) = $1-0.5^p$, P(Duration>*Target*+0.5*sigma*) = $1-(1-0.3085)^p$, and P(Duration>*Target*+*sigma*) = $1-(1-0.1587)^p$, where *p* is the number of strongly parallel paths.

However, we also notice that in projects with 5 paths the differences in probabilities between the non-agile and agile projects initially increase as *c* increases but then start to fall. In projects with two strongly parallel paths we also observe an initial increase with *c* although the magnitude of this increase is smaller. This is in contrast to the serial projects where the differences in probabilities decrease monotonically as *c* increases. This phenomenon is illustrated in Figures 2-36 and 2-37 which show differences in probabilities with respect to *c* for projects with 20 and 50 activities per path respectively.

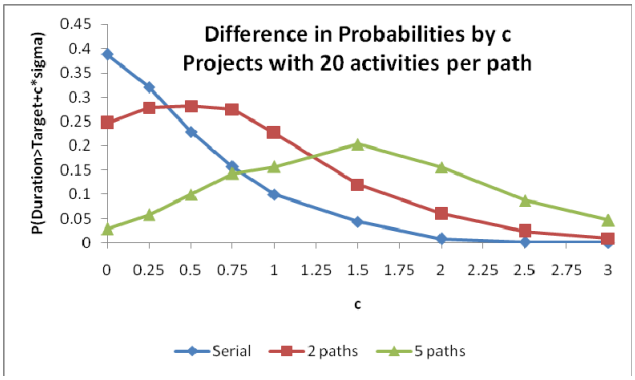


Figure 2-36: Differences in probabilities by c -- projects with 20 activities per path

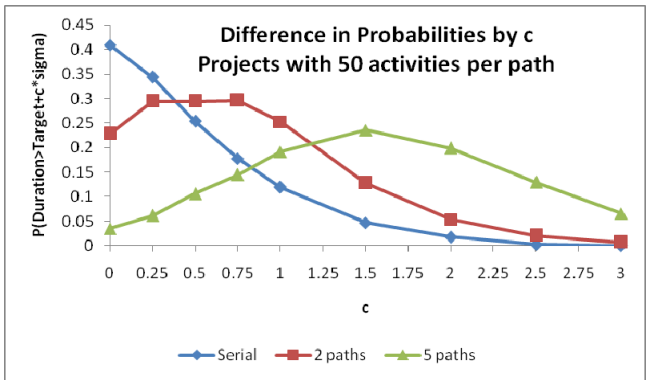


Figure 2-37: Differences in probabilities by c -- projects with 50 activities per path

This result suggests that even in projects with many parallel paths, forward agility can have an effect depending on the due date. In the case of 5 strongly parallel paths, the initial probability ($c = 0$) of exceeding the Target is so large that the effect of agility is overpowered by the effect of multiple parallel paths. However, as the due date increases, the effect of agility becomes more apparent until the due date gets so large that having agility does not make any significant difference.

2.5 SERIAL-PARALLEL INDEX

In section 2.3 and 2.4 we examined projects with special and very well defined structures. However, in practice few project networks will have such well behaved topology; therefore herein we consider more general structures. We generated a large number of project networks based on a serial-parallel (SP) index described in Tavares et al. (1999). An extension of this work was presented in Tavares et al (2004) where the authors introduced the surrogate indicator which served as a predictor of the impact factor. The surrogate index was built using a regression model for which independent variables included the morphological indicators of the project network and randomness of the activity durations. The results showed that the serial-parallel index is the most important of the morphological factors in predicting the distribution of project duration; therefore it is a measure we use to classify structures of general project networks.

The SP index of a project is equal to $(m\text{-}1)/(n\text{-}1)$ where m is the number of sequential stages in the network (or the length of the longest path in terms of the number of activities) and n is the total number of activities. Therefore, a completely parallel project, i.e., each activity is in a path by itself, has a SP index of $(1\text{-}1)/(n\text{-}1) = 0$ and a completely serial project has an SP index of

(n-1)/(n-1) = 1. Essentially, the SP index measures the length of the longest path, in terms of the number of activities, with respect to the total number of activities in the project. Unfortunately, it does not indicate how many parallel paths or how many precedence relationships exist in the project. Figure 2-38 shows a number of different topologies (non-exhaustive) that can be achieved for a 6 activity project with the SP index of 0.4 and the relationship of the SP index to the order strength.

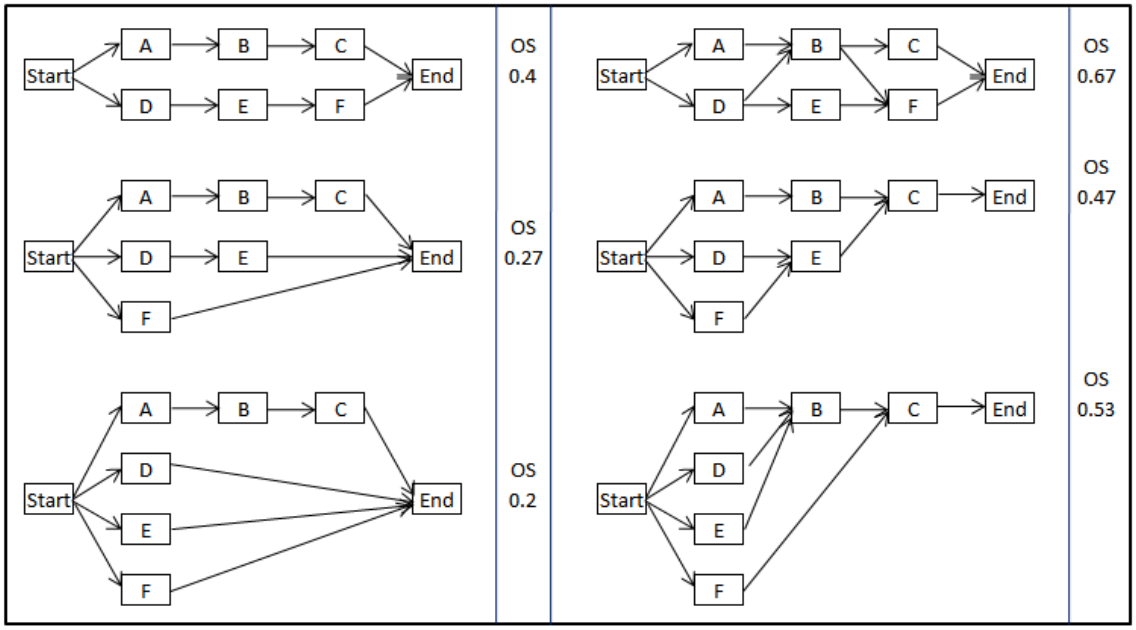


Figure 2-38: Examples of projects with SP index = 0.4

Nevertheless, from our computational experiments, the SP index was a better predictor of project delay than the order strength and it is the measure of topology we use throughout this section.

We generated random problem instances using RanGen2 (Vanhoucke et al. 2004) varying the SP index (between 0 and 1 in 0.1 increments) and the number of activities in the project (between 5 and 50 in increments of 5). Since for a given SP index value, except 0 and 1, there are multiple possible structures, we generated 20 problem instances for each set of parameters, for example, we created 20 projects with 30 activities and an SP index of 0.5. As before, we set

the Target date equal to the longest path where activity durations are set to their expected values. Each project was then simulated *N* times with and without agility. The results are presented in Figures 2-39 through 2-48 and show the average probabilities of the 20 instances for each set of parameters. The x-axis represents the SP index, the y-axis shows the probability that the project will exceed its Target date, and the lines show the number of activities in the project.

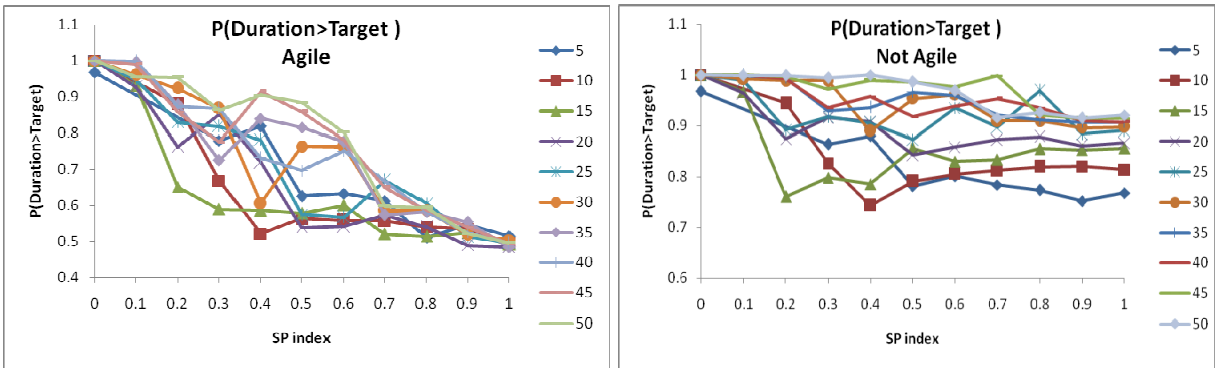


Figure 2-39: P(Duration>Target) by SP index

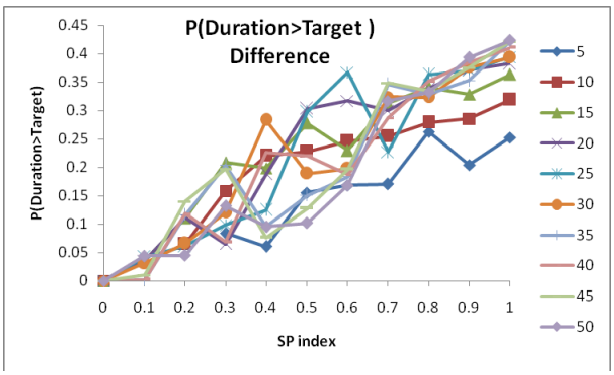


Figure 2-40: P(Duration>Target) - difference by SP index

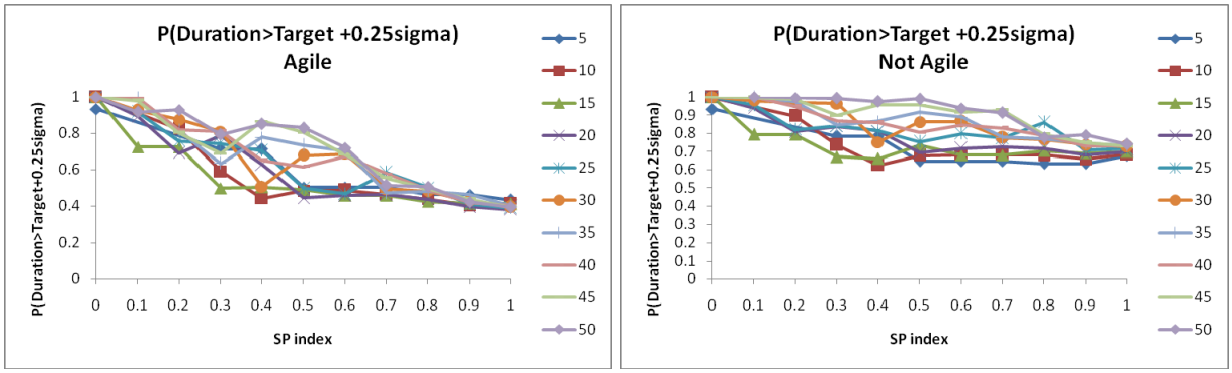


Figure 2-41: P(Duration>Target+0.25sigma) by SP index

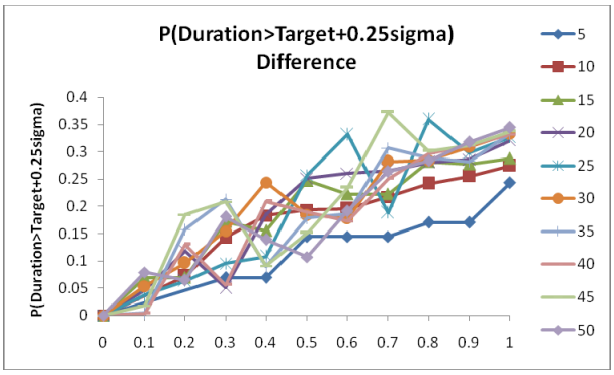


Figure 2-42: P(Duration>Target+0.25sigma) - difference by SP index

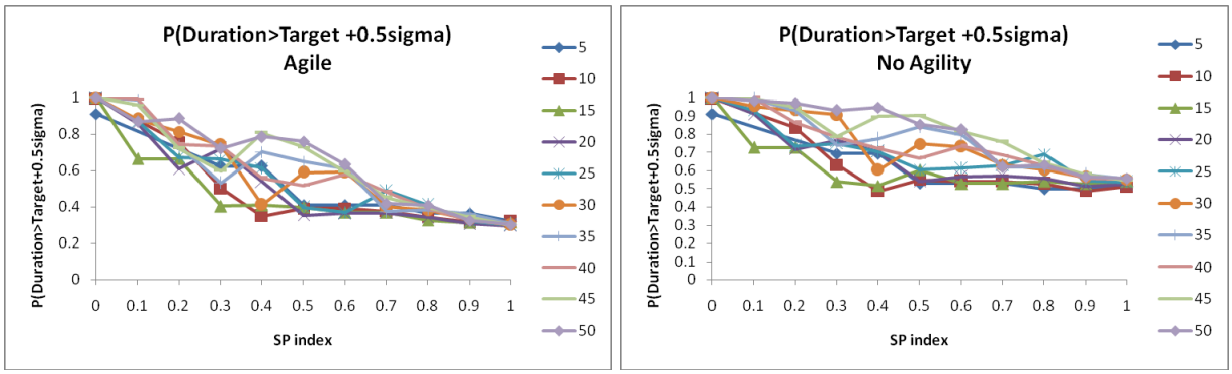


Figure 2-43: P(Duration>Target+0.5sigma) by SP index

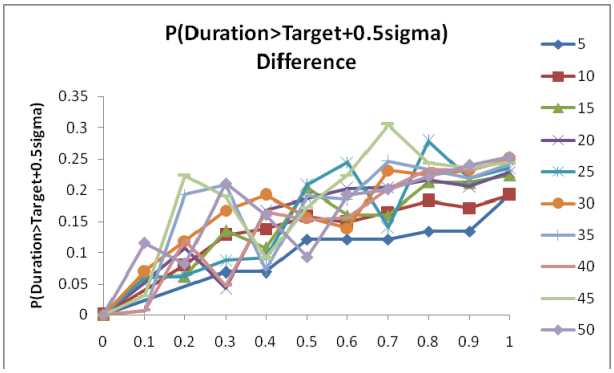


Figure 2-44: P(Duration>Target+0.5sigma) - difference by SP index

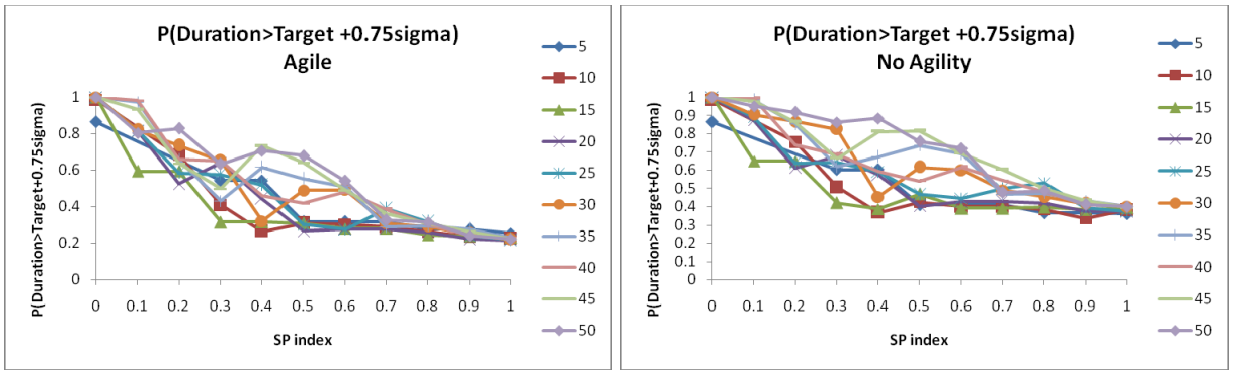


Figure 2-45: P(Duration>Target+0.75sigma) by SP index

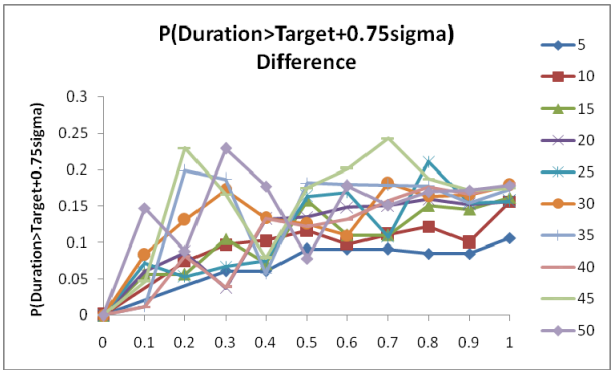


Figure 2-46: P(Duration>Target+0.75sigma) - difference by SP index

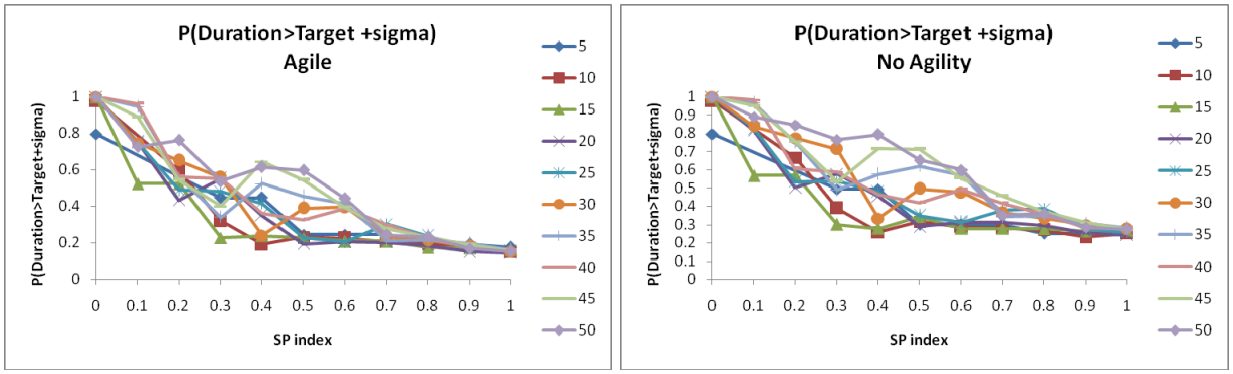


Figure 2-47: P(Duration>Target+sigma) by SP index

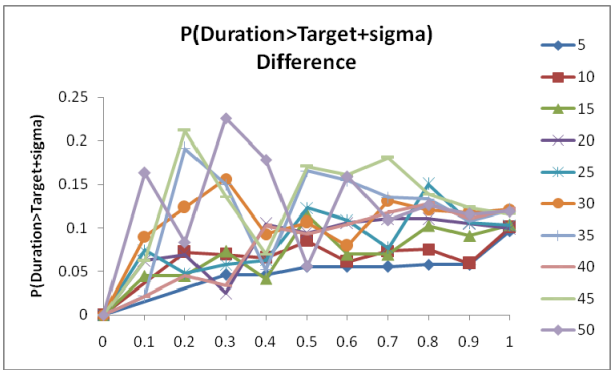


Figure 2-48: P(Duration>Target+sigma) - difference by SP index

In the agile case, as the SP index increases (thus, the number of activities on the longest path increases) and the projects become more serial the probability of delays decreases for all values of *c*, which is consistent with our intuition. Examining the difference in probabilities, the effect of the lack of agility is 0 for completely parallel projects (SP index = 0) and increases as the SP index increases. In addition, the difference between the non-agile and the agile case is larger for projects with larger number of activities. This pattern is especially visible in case of *c*=0 or P(duration>Target) and for completely serial projects (SP index = 1), which confirms the results obtained in section 2.3.

The differences between the agile and the non-agile projects seem to diminish with increasing due dates. This suggests, not surprisingly, that as we increase the due date for the project, agility becomes less significant. In fact, the lack of agility would cease to have any effect if the due date is sufficiently large.

2.6 CONCLUSIONS AND FUTURE WORK

In this chapter we looked at the impact of agility on project delays based on the topology of the project network. We examined serial projects, projects with a special (strongly parallel) topological structure, as well as general projects categorized by the serial-parallel index. We concluded that agility can have a significant effect on the duration of a project; however this effect varies based on the topology of the project network as well as the due date of the project. The more parallel the project, the smaller the impact of the lack of agility. On the other hand, the more serial the project, the greater is the effect of agility. In addition, we notice that for projects with multiple parallel paths, the impact of agility gradually increases with the increase in the due

date until it reaches some maximum after which it starts to fall. This is in contrast to serial projects where the impact of forward agility decreases monotonically with the increase in the due date.

One of the future research avenues to pursue is to examine when having agility of an activity is most valuable. This is an important problem to consider from the project's owner's perspective, especially when negotiating with contractors. Our intuition is that agility should be more valuable in the later tasks as well as in projects with a higher duration variance. Future research will address measuring the value of agility and characterizing structures where agility of activities is most significant.

3.0 CHAPTER THREE: SERIAL PROJECTS

3.1 INTRODUCTION

A serial project consists of a series of tasks that have to be performed sequentially. Serial projects are of interest for two main reasons: (1) a reasonable number of real projects can be modeled as serial networks either because of the nature of the work to be executed, or the presence of one dominant path, and (2) results obtained for serial projects can help us understand the nature and complexity of a more general problem.

In this chapter we consider a stochastic time-cost trade-off problem. Given a set of activities, probability distributions of task durations, a target date for project completion and the penalty for exceeding that target, as well as crashing options for some or all of the activities, our goal is to find the best crashing policy that will minimize total expected project cost. Herein we focus on uncertainty in task durations and examine both linear and nonlinear crashing.

We make several assumptions regarding the problem: (1) activity durations follow triangular distributions (This assumption however, is for convenience only. The methods we present in this chapter do not require that this assumption is satisfied); (2) crashing an activity reduces its duration by a discrete value and durations of all tasks are also discrete; (3) there are

no lead times and all activities start as soon as possible; (4) we incur a constant penalty for each time period the project is past due; (5) there are no incentives for finishing early; (6) activities cannot be performed in parallel; and (7) there are no resource constraints. Example 3.1 illustrates a possible project with linear crash costs meeting all of those criteria:

Consider a serial project with three tasks as follows:

Table 3-1: Illustrative Example 3.1 – Serial project with linear crashing

Task	Predecessor	Days duration			Crash Cost	Crash days
		Optimistic	Most Likely	Pessimistic	$ per day	Maximum
A	None	2	3	4	15	1
B	A	3	5	8	20	2
C	B	4	8	12	18	2

Suppose that we have a target of 16 days for finishing the project and a penalty cost of $100 per day for each day that the target date is exceeded.

The AON network representation of this project is shown in Figure 3-1 where *O* = optimistic duration, *ML* = most likely duration and *P* = pessimistic duration:

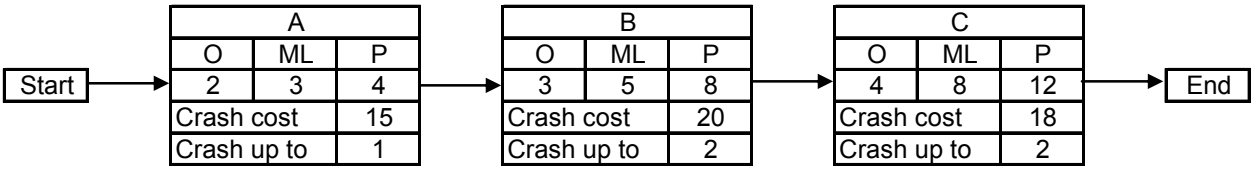


Figure 3-1: Illustrative Example #3.1 -- serial project with linear crashing

When uncertainty is related to the work content of a task (internal), crash costs and times are no longer linear. Example 3.2 shows a case with internal uncertainty. Note that in deriving crash cost we use an artificial number which we refer to as a "regular cost" per day. This value is not included in determining the total project cost but is needed to calculate crash costs for each option. In addition, the number of days an activity will be crashed by depends both on the option chosen as well as on the normal duration of the activity.

Table 3-2: Illustrative Example 3.2 – Serial project with nonlinear crashing

Task	Predecessor	Days duration			Regular cost
		Optimistic	Most Likely	Pessimistic	$ per day
A	None	2	3	4	15
B	A	3	5	8	20
C	B	4	8	12	18

Table 3-3 presents some crashing options for Example 3-2.

Table 3-3: Crashing options (nonlinear case)

A	Option 0 (0% reduction)		Option 1 (25% reduction)		Option 2 (50% reduction)	
Duration	days crashed	Crash Cost	days crashed	Crash Cost	days crashed	Crash Cost
2	0	$ -	1	$ 6.00	1	$ 10.00
3	0	$ -	1	$ 9.00	2	$ 15.00
4	0	$ -	1	$ 12.00	2	$ 20.00

B	Option 0 (0% reduction)		Option 1 (25% reduction)		Option 2 (50% reduction)	
Duration	days crashed	Crash Cost	days crashed	Crash Cost	days crashed	Crash Cost
3	0	$ -	1	$ 24.00	2	$ 40.00
4	0	$ -	1	$ 32.00	2	$ 53.33
5	0	$ -	1	$ 40.00	3	$ 66.67
6	0	$ -	2	$ 48.00	3	$ 80.00
7	0	$ -	2	$ 56.00	4	$ 93.33
8	0	$ -	2	$ 64.00	4	$ 106.67

C	Option 0 (0% reduction)		Option 1 (25% reduction)		Option 2 (50% reduction)	
Duration	days crashed	Crash Cost	days crashed	Crash Cost	days crashed	Crash Cost
4	0	$ -	1	$ 28.80	2	$ 48.00
5	0	$ -	1	$ 36.00	3	$ 60.00
6	0	$ -	2	$ 43.20	3	$ 72.00
7	0	$ -	2	$ 50.40	4	$ 84.00
8	0	$ -	2	$ 57.60	4	$ 96.00
9	0	$ -	2	$ 64.80	5	$ 108.00
10	0	$ -	3	$ 72.00	5	$ 120.00
11	0	$ -	3	$ 79.20	6	$ 132.00
12	0	$ -	3	$ 86.40	6	$ 144.00

Unlike in the linear case, we do not have a constant crash cost per day or a constant number of days we can shorten each activity. Instead we have options to shorten duration of an activity by a certain percentage. In the example above, there are three options available for each task – no crashing (or 0% reduction), 25% reduction, and 50% reduction. The number of days each activity is shortened by depends on the normal duration realized and is simply equal to that duration multiplied by the percentage reduction and rounded to the nearest integer. The crash costs are calculated according to the following formula:

$\frac{Regular\ cost \times Normal\ duration \times Reduction}{1 + Reduction}$ where *Reduction* is the percentage reduction expressed in decimals and $Normal\ duration \times Reduction$ is simply the number of days we would crash the activity by, given the reduction and the normal duration.

To illustrate the process of calculating crash durations and crash costs, consider activity A in Table 3-2 and 3-3. Possible durations are 2, 3, or 4 days and the regular cost is $15 per day. Consider option 2 (50% reduction). First we calculate the crash duration (Table 3-4).

Table 3-4: Days crashed – calculations (nonlinear case)

Normal duration	Days Crashed
2	$2 \times 0.5 = 1$
3	$3 \times 0.5 = 1.5 \approx 2$
4	$4 \times 0.5 = 2$

The crash cost calculations are presented in Table 3-5 below.

Table 3-5: Crash cost calculations (nonlinear case)

Normal duration	Crash cost
2	$\frac{15 \times 2 \times 0.5}{1.5} = \$10$
3	$\frac{15 \times 3 \times 0.5}{1.5} = \$15$
4	$\frac{15 \times 4 \times 0.5}{1.5} = \$20$

The difficulty in the problem considered in this chapter comes from uncertain task durations. When task durations are deterministic, the problem becomes trivial – we simply crash the cheapest activities until we achieve duration equal to the Target date. However, in the presence of stochastic durations the solution is not so obvious. There are at least two approaches that we can take: making all decisions at once without revising them at a later time vs. making decisions dependent on the state of the project. These can be described respectively as containment and contingency strategies.

Since the contingency approach provides us with managerial flexibility (Elmaghraby, 2005) and thus results in better solutions, in this chapter we focus on contingency planning. The rest of this chapter is organized as follows: in section 3.2 we describe previous related research, section 3.3 states the research problem, section 3.4 describes algorithms for the linear case. In 3.5 we present computational results for the linear case, section 3.6 discusses algorithms and computational results for the nonlinear case, and we present our conclusions in section 3.7.

3.2 PREVIOUS RESEARCH

A solution to the serial network problem using decision theory was previously proposed by Lowe and Wendell (2002). They considered a serial project with binomial distributions of the task durations, that is, each task could have either short or long duration. In addition, crashing options existed for each activity. Crashing a task would change the distribution of its duration and would incur a one-time cost; however, the type of the distribution would remain the same. Moreover, the project under consideration would have a predetermined target date. Exceeding the target date would result in a per day penalty. The objective was to devise a policy that would lead to the completion of the project with minimum cost. Two types of problems were investigated – one without lead times for speed-up decisions and one where the lead times existed. The proposed solution method involved constructing a decision tree in order to find a policy that would yield the minimum expected cost. However, while the method works well for small problem instances, the size and the complexity of the decision tree grows very quickly as the number of activities increases. Moreover, the proposed method is impractical for problems with multiple decision options or with tasks with more complex probability distributions.

In this chapter, we investigate other approaches for dealing with the serial network problem, such as dynamic programming and various heuristic methods. Dynamic programming (DP) is a technique developed by Bellman (1957) for solving sequential, multi-stage decision problems by decomposing them into subproblems which are easier to solve. DP can be applied to deterministic as well as to stochastic problems where either the payoff or the next state are uncertain, that is, we can only describe them as some probability function. DP can solve our problem with multiple decisions and multiple discrete events for each task quite efficiently. It can be viewed as a classic use of DP in a stochastic environment where the next state is uncertain as described in Bellman (1958). In the absence of decision lead times, the stages and states of the problem are very well defined. A decision regarding a particular activity takes place only right before that activity begins and each state is fully defined by two variables – time elapsed since the beginning of the project and the current task.

3.3 STATEMENT OF THE RESEARCH PROBLEM

The research problem considered in this chapter is a stochastic time-cost trade-off problem for serial projects where uncertainty comes only from activity durations and can be either external or internal. Given the set of activities A={1,…,*n*} where *n* is the total number of activities excluding dummy start and end tasks, a serial topology of the project, discrete triangular probability distributions of task durations where O_i = optimistic duration of activity *i*, ML_i = most likely duration of activity *i*, and P_i = pessimistic duration of activity *i*, target date (*Target*), and a per day penalty for exceeding the target date (*Penalty*), find a crashing policy that will minimize total expected project cost. Total expected cost includes total crash cost plus total expected

penalty. It is important to note that approaches presented in this chapter are applicable to any probability distributions; however, for convenience, we limit our discussion to triangular distributions. In the linear case, we also know the maximum number of time periods to crash each activity (d_i) and per day crash cost for each activity (c_i) that remains constant regardless of the number of days we crash that activity. In the nonlinear case, we consider percentage reductions in the duration of each activity. Therefore, the number of days we end up crashing each activity as well as the total crash cost depend on this activity's normal duration as shown in Tables 3-4 and 3-5.

Furthermore, we assume that the duration of a task is discrete and becomes known only after that task is fully completed. Crashing is also discrete (integer); that is, we can reduce the duration of an activity by a multiple of a full time period only. Additionally, we make crashing decisions dynamically throughout the execution of the project; a new decision making stage occurs before the start of each activity. Because we are looking at serial projects only, at each decision stage we have full information about durations of all preceding activities.

There is one important difference between linear and non-linear cases. Since we are looking at serial projects, crashing any activity by one day in the linear case simply shifts the probability distribution of project duration without changing its shape (only the mean changes). In the non-linear case however, the shape of the distribution changes due to the calculation of days crashed as illustrated in Table 3-4. Figures 3-2 and 3-3 show the probability distributions of projects in Examples 3.1 and 3.2 respectively, before and after crashing activity B to its maximum.

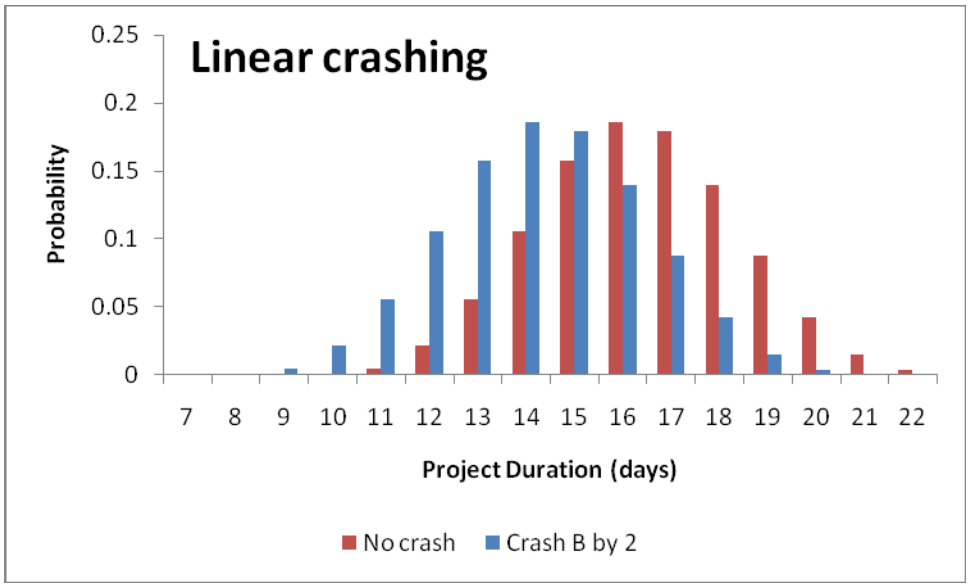


Figure 3-2: Distribution of project duration (linear case)

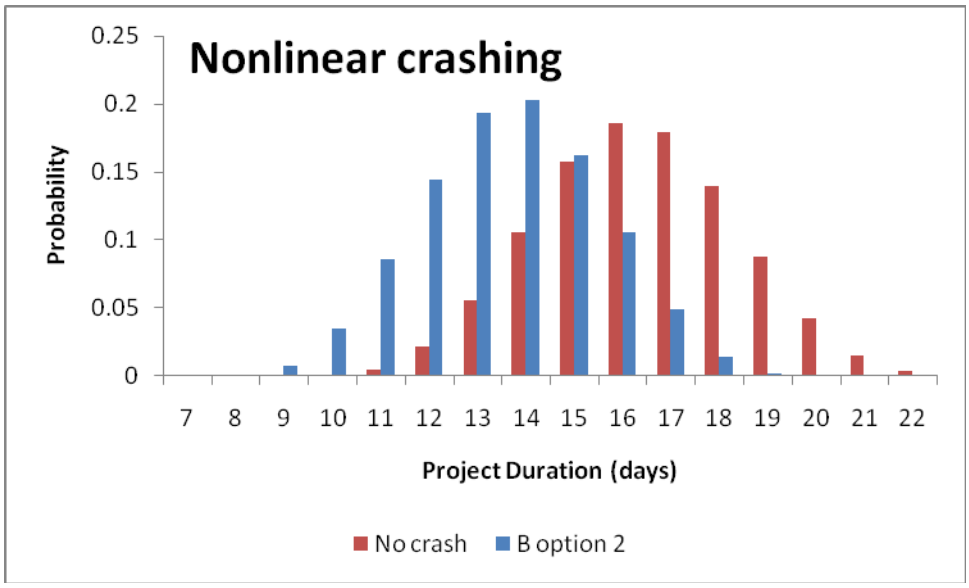


Figure 3-3: Distribution of project duration (nonlinear case)

3.4 ALGORITHMS – LINEAR CASE

In this chapter we present a number of algorithms. We start our discussion with the exact method – dynamic programming, which guarantees optimal solutions. Next we discuss several heuristic

methods and compare their performance to the optimum as well as to the case with perfect information. Those heuristics include the Biggest Bang with simulation, the Biggest Bang with normal approximation, and the Simple Minded method. To illustrate how each of these methods work, we use our project from Example 3.1.

3.4.1 Dynamic Programming

In DP, a problem is divided into stages and each stage has a number of states associated with it. In deterministic DP, a decision in one stage transforms the problem into a state in the next stage. In stochastic DP with uncertain states, such a transformation occurs after both a decision and a realization of a random variable associated with the current state. Based on the principle of optimality (Bellman, 1958), the optimal decision for each stage/state combination does not depend on previous states or decisions. In addition, the terminal state has to be solvable by itself and there must exist a recursive relationship that identifies the optimal decision for stage i, given that stage i+1 has already been solved (we solve the problem in reverse order, so that decisions at stage i+1 are determined before stage i is considered).

We can easily show that the problem considered in this section satisfies all conditions for stochastic dynamic programming. The stages in the problem correspond to sequential activities in the project. Each stage has a number of states, or in our case, activity starting times, and the decisions are whether to crash an activity and by how much. The payoff function is associated with decisions (crash costs) and final completion time of the project (penalty). The stochastic nature of the problem stems from uncertain activity durations, or in the DP framework, uncertain states. Probability distributions of task durations can be modeled as transition probabilities from one stage to a state in the next stage. Those probabilities do not depend on actions (decisions)

chosen. The terminal stage (or the dummy end activity) is solvable by itself since we can easily calculate the value of the payoff function at that point – it is simply a function of the project completion time (or the start time of the dummy activity). Finally, at each stage we should be able to identify the optimal decision, given a state that the project is in, by calculating the "cost-to-go function". Cost-to-go is simply the value of the payoff function which includes only costs and expected costs incurred from the current stage to the terminal stage. We will use backward recursion for our DP models presented in this and in future chapters.

To illustrate how DP finds the optimal crashing policy consider the project in Example 3.1. Since DP makes decisions at discrete points in time, we need to convert continuous probability distributions of task durations into their discrete counterparts. The new, discrete probability distributions are shown in table below:

Table 3-6: Discrete probability distributions

Activity	Duration	Probability
A	2	0.1250
	3	0.7500
	4	0.1250
B	3	0.0250
	4	0.2000
	5	0.3583
	6	0.2667
	7	0.1333
	8	0.0167
C	4	0.0078
	5	0.0625
	6	0.1250
	7	0.1875
	8	0.2344
	9	0.1875
	10	0.1250
	11	0.0625
	12	0.0078

Next we need to identify all possible states for each stage or equivalently, all possible start times for each activity. Activity A can only start at time 0 since it does not make sense to postpone the start time of the project. Start time of activity B obviously depends on the duration

of A as well as the crashing decision we made for A. The minimum start time for B is day 1 – the minimum duration of A (2 days) minus A's maximum possible crashing (1 day). The maximum start for B is equal to the maximum duration of A with no crashing or day 4. Therefore, the range of start times (states) for activity B is 1 through 4. We perform similar analysis for activity C. The minimum start time of C is equal to the minimum start time of B plus minimum duration of B minus B's maximum crashing or 1+3-2 = 2. The maximum start time of C is equal to the maximum start time of B plus maximum duration of B, or 4+8 = 12. Finally, for the dummy End node, the range of start times is 2+4-2 = 4 through 12+12 = 24.

We are now ready to start the recursive procedure. We start with the terminal stage (or the dummy End node). For each possible state we calculate the payoff function. The Target date is 16, therefore, we incur penalty only if the duration of the project exceeds 16 days. The cost-to-go function is therefore

$$Cost^*(END, t_{END}) = \max\{0, (t_{END} - Target)\} \times Penalty$$

where:

*Cost** – optimal cost to go,

i – stage of the problem (denotes the activity about to start),

t_i – start time of activity *i*,

Target – the target date for the project,

Penalty – penalty per day for exceeding the *Target.*

In other words, the cost-to-go at stage *END* is 0 if $t_{END} \leq Target$ and $(t_{END} - Target) \times Penalty$ if $t_{END} > Target$. Now we move on to the previous stage, that is, activity C. Again we need to calculate the cost to go for each stage and each decision. Because C has uncertain durations, we

need to use transition probabilities in our calculations. At $t_C = 2$, and no crashing, the cost-to-go is calculated as follows:

$$Cost(C, t_C = 2, z_C = 0) = \sum_k p_k^C (Cost*(END, t_C + k - 0)) + c_C \times 0$$

where:

z_i – number of time periods to crash activity i (in this case, $z_C = 0$)
c_i – cost of crashing activity i by one day (in this case, $c_C = 18$)
p_k^i – probability that normal duration of activity i will be k days

We need to calculate the cost-to-go for each possible decision and pick a crashing decision that minimizes expected cost-to-go (repeat for each state). Therefore, the optimal cost-to-go for activity C is expressed as:

$$Cost^*(C, t_C) = \min_{z_C} \{\sum_k p_k^C (Cost*(END, t_C + k - z_C)) + c_C \times z_C\}$$

When we perform calculations for all possible start times of activity C, we get the following:

Table 3-7: Expected cost-to-go for activity C

	Crash by			Optimal
Start time	0	1	2	Cost-to-go
2	0	18	36	0
3	0	18	36	0
4	0	18	36	0
5	0.78125	18	36	0.78125
6	7.8125	18.78125	36	7.8125
7	27.34375	25.8125	36.78125	25.8125
8	65.625	45.34375	43.8125	43.8125
9	127.3438	83.625	63.34375	63.34375
10	207.8125	145.3438	101.625	101.625
11	300.7813	225.8125	163.3438	163.34375
12	400	318.7813	243.8125	243.8125

Similarly, for activity B, the optimal cost to go is:

$$Cost^*(B, t_B) = \min_{z_B} \{\sum_k p_k^B (Cost^*(C, t_B + k - z_B)) + c_B \times z_B\}$$

and the exact expected costs are as follows:

Table 3-8: Expected cost-to-go for activity B

	Crash by			Optimal
Start time	0	1	2	Cost-to-go
1	16.73645	26.53515	41.68021	16.73645
2	32.65442	36.73645	46.53515	32.65442
3	54.22133	52.65442	56.73645	52.65442
4	85.04866	74.22133	72.65442	72.65442

Finally, for activity A we get the following:

$$Cost^*(A,t_A) = \min_{z_A}\{\sum_k p_k^A (Cost^*(B,t_A + k - z_A)) + c_A \times z_A\}$$

Table 3-9: Expected cost-to-go for activity A

	Crash by		Optimal
Start time	0	1	Cost-to-go
0	52.65442	48.16467	48.1646699

Therefore, the optimal crashing policy for this project is to crash A by 1 day, crash B by 1 day if B starts at time 3, or crash B by 2 days if B starts at time > 3, and crash C by 1 day if C starts at time 7 or crash C by 2 days if it starts at time > 7.

The most general DP formulation, for an arbitrary number of activities is presented in Exhibit 3.1.

State representation:

$x = (i, t_i)$ where

i ={1,..,*END*} – current activity
N = number of non-dummy activities
t_i – starting time of activity *i*

z_i – number of time periods to crash activity *i*

c_i – cost of crashing activity *i*

Cost*(x) – the optimal cost-to-go for state *x*

p_k^i -- probability that normal duration of activity *i* will be *k* days

Target – target date for project completion

Penalty – cost per day for exceeding *Target*

For the terminal dummy activity (END):
For all t_{END} compute:

$$Cost^*(END, t_{END}) = \max\{(t_{END} - Target), 0\} \times Penalty$$

For all activities i, where $i \le N$:
For all t_i do:

$$Cost^*(i, t_i) = \min_{z_i}\{\sum_k p_k^i (Cost^*(i+1, t_i + k - z_i)) + c_i \times z_i\}$$

Exhibit 3-1: Dynamic programming formulation

Dynamic programming provides an elegant formulation and an optimal solution method for serial projects with linear crashing. However, one of the reasons for considering serial projects is to develop methods that can be later modified to solve more general cases (i.e., non-serial projects). We also know that more general cases can become too complex for dynamic programming to handle; therefore we consider several heuristic methods in this chapter. To maintain consistency with DP method, we also use discrete probability distributions of task durations (therefore, making all durations integer).

3.4.2 Biggest Bang

Biggest Bang (BB) is an iterative algorithm that, at each stage, calculates a probability that the project will finish late, and uses that probability to make crashing decisions. Although exact probability calculations are possible (a probability distribution of project duration is the sum of individual activity durations), the computational overhead required is too expensive to utilize this approach on large problems. Instead, we estimate the probability of completing the project late by either using simulation or normal approximation. In this section, we first give the general description of the algorithm without regard as to how the probabilities were obtained. We later show the implementation of the algorithm with simulation and with normal approximation.

At each stage we calculate the Biggest Bang index (BBI) for each activity, which is simply equal to the expected cost savings from crashing an activity by one time period. BBI is calculated as follows:

$$BBI_i = P(\textit{project duration} > \textit{Target}) \times \textit{Penalty} - c_i$$

Since the BB indices give us cost savings from crashing only one activity at a time by only one time period at a time, we repeat these calculations multiple times at each stage. We stop when we either crashed all activities to maximum or when all BB indices are negative. It is important to note that BB is a dynamic algorithm; therefore, we only crash the activity that is starting immediately and repeat the procedure at each stage of the project.

3.4.2.1 Simulation

As we mentioned before, one of the two efficient ways for obtaining probability distribution of project duration is simulation. We simulate the project N times and record the

resulting probability distribution. Using our project from Example 3.1, we obtain the following before the project begins:

Table 3-10: BB indices (simulation) -- decision stage 1

	Iteration 1	Iteration 2	Iteration 3
Activity	Prob = .4667	Prob = .2867	Prob = .1467
A	31.67	13.67	-0.33
B	26.67	8.67	-5.33
C	28.67	10.67	-3.33

At the first iteration (first column), we assume that nothing has been crashed. Simulating the project results in a probability of 0.4667 that the project will be late. The values in the table represent the Big Bang indices for each activity respectively. For example, we calculate the BB index for activity A as $BBI_A = 0.4667 \times 100 - 15 = 31.67$. We want to select activities with large BB indices for crashing; therefore, at iteration 1 we reduce the duration of activity A. We continue the process and at the second iteration activity A again has the highest index; however, we already reduced A by its allowed maximum so we select activity C (with the second largest BBI). Finally, at iteration 3, all BBIs are negative and we terminate the procedure. The final crashing policy before the project begins is to crash A by 1 day and postpone all other decisions until we have more information.

Assume that the normal duration for activity A turned out to be 4 days so the actual duration of A was 4-1 = 3 (we crashed A by 1 day). The start time for activity B is 3 and we enter the new decision stage of the problem. We simulate the project (setting actual duration of A to 3) and we get the following indices:

Table 3-11: BB (simulation) -- decision stage 2

	Iteration 1	Iteration 2	Iteration 3
Activity	Prob = .455	Prob = .25	Prob = .11
A	N/A	N/A	N/A
B	25.5	5	-9
C	27.5	7	-7

At this stage the policy is to reduce C by 2 days; however, we are going to postpone (and possibly revise) this decision until we know the true duration of activity B.

If the normal (and actual) duration of B turned out to be 5 days, we would start the final decision stage of the project at time 8. We obtain values shown in Table 3.12, We therefore crash C by one day and terminate the procedure.

Table 3-12: BB (simulation) -- decision stage 3

	Iteration 1	Iteration 2
Activity	Prob = .38	Prob = .17
A	N/A	N/A
B	N/A	N/A
C	20	-1

3.4.2.2 Normal approximation

Another way to estimate the distribution of project duration is to use normal approximation. We can apply the central limit theorem if the number of activities in a project is large enough. Of course, the project from Example 3.1 that we use to illustrate the algorithms has only three activities so the normal approximation in this case may be questionable; however, we assume that it is sufficient for demonstration purposes.

Using PERT methodology, we calculate the mean duration of the project by summing the expected durations of individual activities. The standard deviation of the project is calculated as

a square root of the sum of activity variances. The mean and the variance of a triangular distribution are calculated as follows:

$$\text{Mean} = \frac{O + ML + P}{3}$$

$$\text{Variance} = \frac{O^2 + ML^2 + P^2 - O(ML) - O(P) - ML(P)}{18}$$

Table 3.13 shows these values for all three activities:

Table 3-13: Mean and variance of activities

Task	Expected Duration	Variance
A	3	0.17
B	5.33	1.06
C	8	2.67

We therefore, approximate the distribution of the project duration as $\sim N(\mu = 16.33, \sigma = 1.97)$. At this point we can use the standard normal table to find the probability that the project will be late, which is equal to 0.5671. Tables 3-14 through 3-16 show BB calculated indices and activities selected for reduction for all three decision stages assuming normal durations of activities A and B are the same as in the case with simulation (4 and 5 respectively):

Table 3-14: BB (normal) -- decision stage 1

	Iteration 1	Iteration 2	Iteration 3
Activity	Prob = .5671	Prob = .3677	Prob = .199
A	41.71	21.77	4.90
B	36.71	16.77	-0.10
C	38.71	18.77	1.90

Table 3-15: BB (normal) -- decision stage 2

	Iteration 1	Iteration 2	Iteration 3
Activity	Prob = .5686	Prob = .3648	Prob = .1938
A	N/A	N/A	N/A
B	36.86	16.48	-0.62
C	38.86	18.48	1.38

Table 3-16: BB (normal) -- decision stage 3

	Iteration 1	Iteration 2	Iteration 3
Activity	Prob = .5	Prob = .2701	Prob = .1103
A	N/A	N/A	N/A
B	N/A	N/A	N/A
C	32.00	9.01	-6.97

At the beginning of the project, we would crash A by 1 day (as in the simulated case) and postpone all other decisions. There are no differences between the two algorithms at stage 2 either; however, at stage 3, BB with normal approximation suggests to crash activity C by 2 days as opposed to 1 day in the simulated case.

3.4.3 Simple-Minded Method

The last algorithm considered in this chapter is the Simple-Minded (SM) method. It is a greedy procedure that iteratively crashes the cheapest activity until expected project duration is less than or equal to *Target*. As with the BB, we perform the SM at each stage of the project, the main difference is that the SM only looks at one point estimate (the mean) of the project duration and does not take into account the size of *Penalty*. We developed this heuristic because (1) this is the simplest procedure that many project managers might be tempted to use, (2) it has certain similarities to the PERT method (crashing cheapest activities until we reach a certain expected

project duration), and (3) we expect it to perform worse than other methods, thus giving us the worst performance benchmark.

The procedure consists of the following: at each decision stage of the project (1) calculate project expected duration, (2) as long as the expected duration is greater than *Target*, crash the cheapest activities. Stop when expected duration ≤ *Target*. To illustrate how the method works, we again use the project from Example 3.1.

At the beginning, the expected duration of the project is equal to 16.33 (3+5.33+8), which is greater than the target of 16 days. Therefore we crash the cheapest activity by one day because the difference between the expected duration and Target is less than or equal to 1. Therefore, we crash A, which is the cheapest activity and terminate the procedure at this decision stage. If we again assume the same activity durations as in the description of previous algorithms, at the next decision stage (after A finishes) the expected duration is again 16.33 (the true duration of A after crashing is 4-1=3, the expected duration of B is 5.33 and the expected duration of C is 8). Using the same steps, we select C for reduction by 1 day but do not act on that decision yet. After we know the duration of activity B, we will have an opportunity to revise the decision about activity C. If B's normal duration turns out to be 5 (as before), the expected duration at stage 3 is 16 and since it is equal to *Target*, we do not crash C.

3.5 COMPUTATIONAL TESTS – LINEAR CASE

We constructed several sets of test problems in order to evaluate the effectiveness of each of the presented algorithm. We looked at networks with different sizes (number of activities), different distribution spans (average difference between pessimistic and optimistic durations), and

different magnitude of crash costs. In order to generate distributions, crash costs, as well as *Target* and *Penalty*, we followed the procedure presented in Gutjahr et al. (1998). We set the target date equal to the expected duration of the project (calculated by summing the expected durations of all activities) and the penalty for exceeding the target to a constant $100 per day. Next, we simulate the project, assuming no crashing, *N* times where *N* is equal to the number of activities in the project multiplied by a constant (20 in our case). The procedure consists of the following steps:

(1) Determine the size of the project. We tested problems with 5, 10, 25, 50, and 75 activities.

(2) Set distribution span. We looked at average distribution spans of 8, 12, 16, 20, and 24 days.

(3) For each activity generate optimistic, most likely, and pessimistic times. *ML* and *P* times are derived from a geometric distribution with the mean equal to the span. Optimistic duration is also derived from a geometric distribution; however, we decided to fix the mean at 8 regardless of the span. We wanted to modify only the width of the project duration distribution without shifting it to the right at the same time.

(4) Determine the maximum number of days by which each activity can be crashed (d_i) – generated from a discrete uniform distribution from [0, O_i-1] interval.

(5) Generate crash cost per day for each activity

   a. Estimate expected penalty cost E(Penalty) of the project with no compression using Monte Carlo simulation.

b. Assign a fraction of E(Penalty) as the total cost for maximum crashing (*TCMC*) of all activities (total cost if we crash all activities to maximum). We refer to this fraction as a *cost structure* and tested values of 0.5 and 1.

c. Calculate an adjustment value for each activity (A_i)

$$A_i = \frac{d_i}{\max_k \{d_k\}} \quad \forall k \in \{1, ..., n\}$$

d. For each activity, generate a random number ~U[0, 1) and multiply it by A_i. We refer to this value as a cost distribution index (*CDI*).

e. Calculate a normalized cost distribution index (*NCDI*) for each activity:

$$NCDI_i = \frac{CDI_i}{\sum_{k=1}^{n} CDI_k}$$

f. Calculate cost of maximum crashing for each activity:

$$\text{cost of maximum crashing}_i = TCMC \times NCDI_i$$

g. Calculate crash cost per day for each activity:

$$c_i = \frac{\text{cost of maximum crashing}_i}{d_i}$$

3.5.1 Results

Since we have three parameters that vary in our test problems (size, span, and cost structure), we present the results in three sections. The first problem set consists of project with the average activity distribution span of 16 days, and the cost structure of 1. Next we discuss results for the set with varying spans, fixing size at 25 activities and cost structure at 1. Finally, the last set consists of projects with varying cost structures but with constant size (25 activities) and average span (16 days). In addition to the four algorithms discussed in in this chapter, for each problem

instance we calculate the expected cost assuming we have full knowledge of activity durations that will be realized in the future. That is, we make all activity durations known to the algorithm a priori, thus reducing the problem to its deterministic counterpart. We refer to the method as the "Perfect Information."

Set 1: Cost structure = 1, Span = 16, varying sizes

Tables 3-17 and 3-18 show average expected costs and average running times by project size and heuristic. The same information is also presented in Figures 3-4 and 3-5.

Table 3-17: Expected cost by project size

Size	DP	BB Simulated	BB Normal	SM	Perfect Information
5	156.79	158.75	160.99	197.42	112.19
10	116.13	121.06	122.82	162.22	56.13
25	155.02	164.81	165.46	329.54	84.08
50	47.30	60.30	62.02	172.34	25.65
75	59.18	75.03	76.75	271.32	30.70

*Averaged over 20 instances of each problem type

Table 3-18: Average running time (in seconds) by project size

Size	DP	BB Simulated	BB Normal	SM
5	0.00200	0.00088	0.00002	0.00001
10	0.01744	0.00843	0.00004	0.00002
25	0.34711	0.27725	0.00038	0.00021
50	3.12492	3.28732	0.00218	0.00117
75	11.18753	15.01350	0.00611	0.00332

*Averaged over 20 instances of each problem type

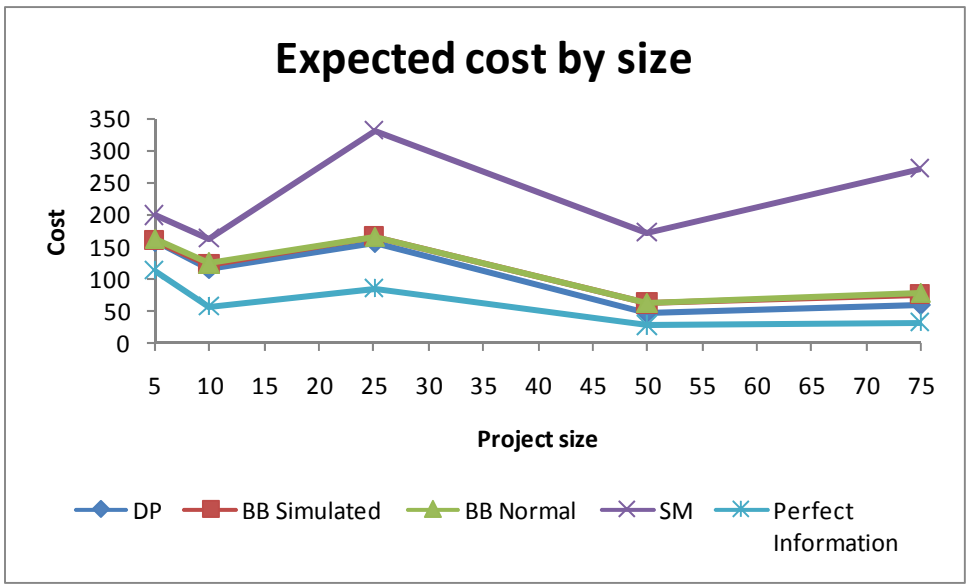


Figure 3-4: Average cost by size

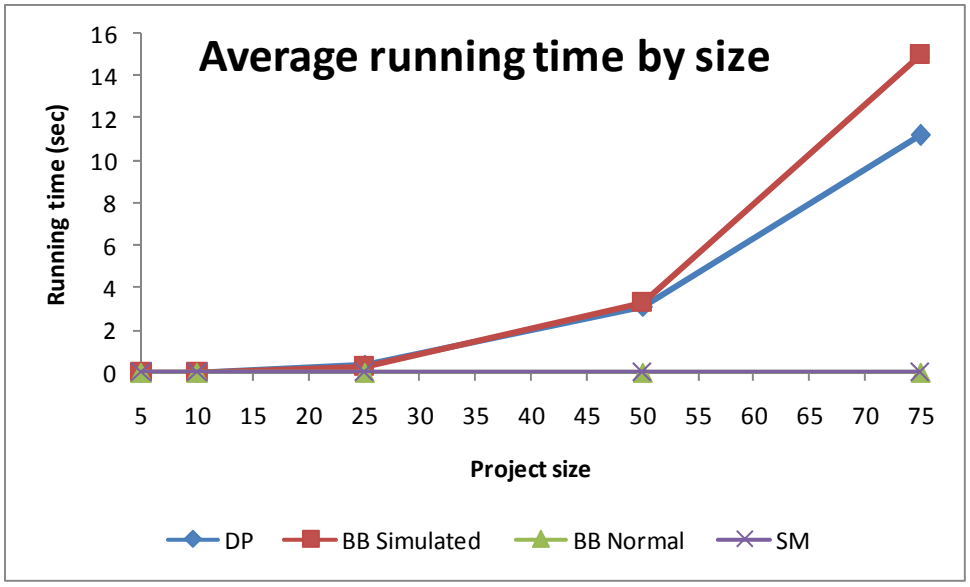


Figure 3-5: Average running time by size

Set 2: Cost structure = 1, Size = 25, varying spans

The next group of tables and figures shows expected costs and average running time by the average project span and heuristic.

Table 3-19: Expected cost by project span

Span	DP	BB Simulated	BB Normal	SM	Perfect Information
8	35.85	40.72	41.40	101.43	18.99
12	46.86	52.42	53.14	135.82	20.12
16	155.02	164.81	165.46	329.54	84.08
20	189.21	203.07	204.39	380.46	106.25
24	220.66	231.99	233.65	586.57	108.31

*Averaged over 20 instances of each problem type

Table 3-20: Average running time (in seconds) by project span

Span	DP	BB Simulated	BB Normal	SM
8	0.04250	0.15261	0.00020	0.00011
12	0.14356	0.21554	0.00028	0.00016
16	0.34711	0.27725	0.00038	0.00021
20	0.86207	0.35410	0.00053	0.00030
24	1.99832	0.47142	0.00064	0.00031

*Averaged over 20 instances of each problem type

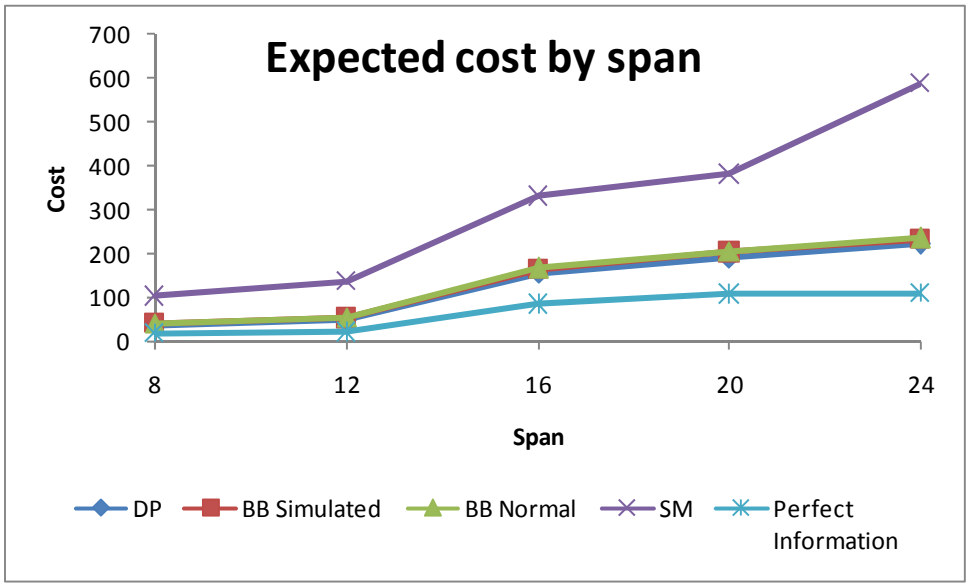


Figure 3-6: Expected cost by project span (linear case)

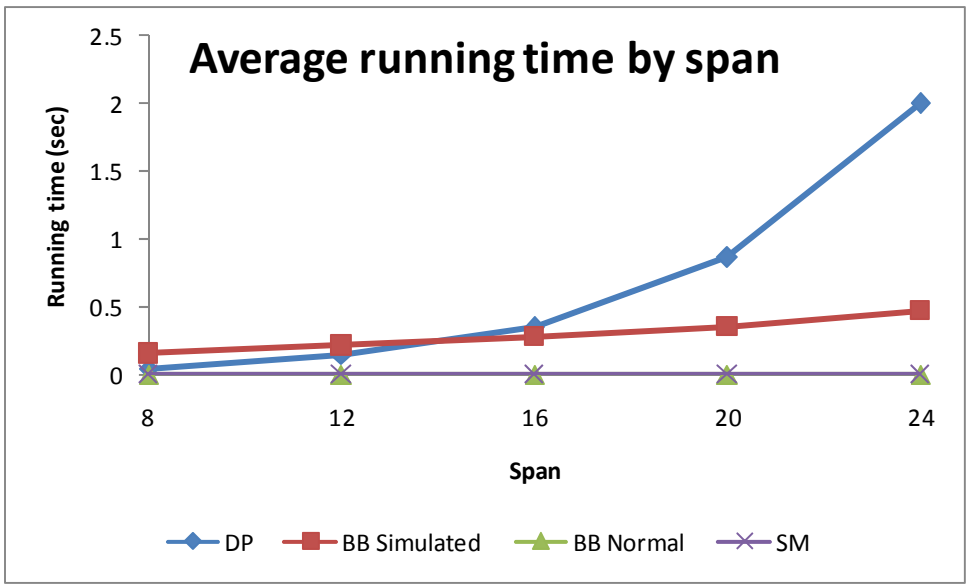


Figure 3-7: Average running time by project span (linear case)

Set 3: Size = 25, span = 16, varying cost structures

Finally, we present the results for the final set where we vary only the cost structure, keeping size fixed at 25 activities and average span at 16 days.

Table 3-21: Expected cost by project cost structure

Cost Structure	DP	BB Simulated	BB Normal	SM	Perfect Information
0.5	92.70	95.40	95.50	246.84	44.57
1	155.02	164.81	165.46	329.54	84.08

Table 3-22: Average running times by project cost structure

Cost Structure	DP	BB Simulated	BB Normal	SM
0.5	0.35338	0.33142	0.00048	0.00020
1	0.34711	0.27725	0.00038	0.00021

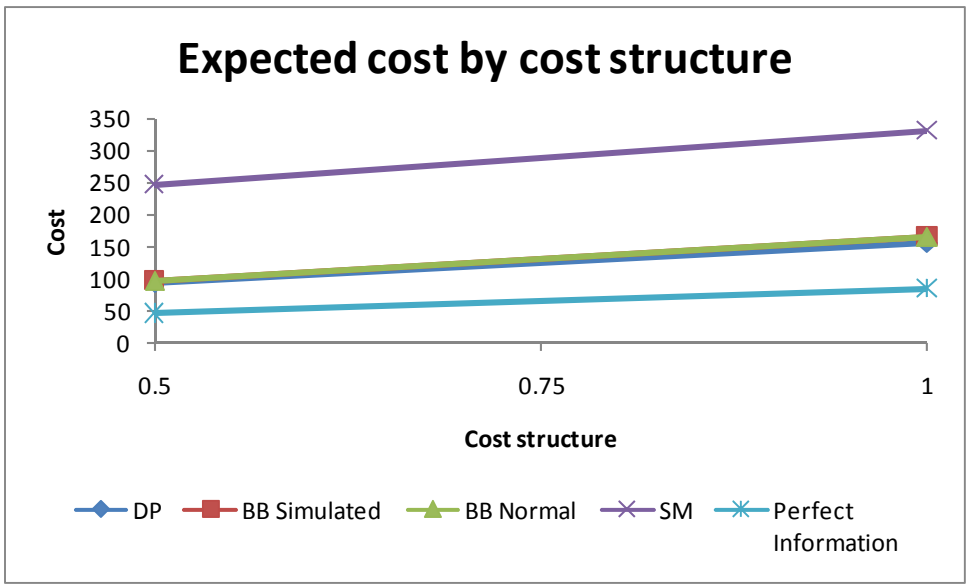


Figure 3-8: Expected cost by project cost structure (linear case)

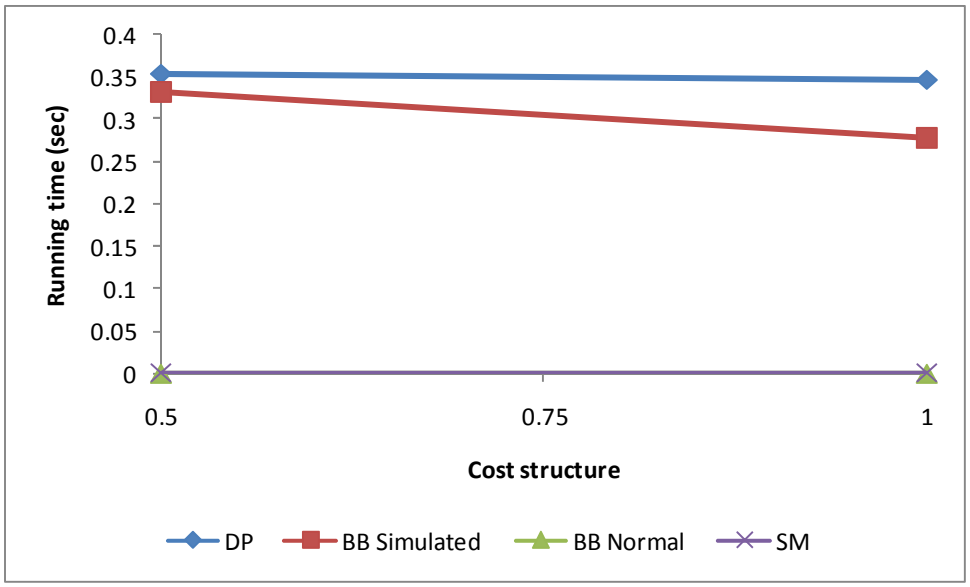


Figure 3-9: Average running times by project cost structure (linear case)

The relative performance of the methods is the same across all problem sets. Not surprisingly, the DP algorithm is superior; however, the expected costs of the Biggest Bang (both simulated and with normal approximation) are very close to those obtained by DP. As expected, the SM is by far the worst performer. It is due to the fact that this method does not take into

account the magnitude of *Penalty*. It would provide the same crashing policy whether the penalty for exceeding *Target* is $1 or $1 million.

In the first problem set, it appears that, as the number of activities increases, the cost gap between DP and BB heuristics increases as well. For 5 activity projects the gap is around 1.25% whereas for projects with 75 activities, that gap is 26.8%. Similarly, the gap between DP and cost with perfect information increases as the projects get larger (from 39.8% for 5 activities to 92.75% for projects with 75 activities), which is consistent with our intuition – the larger the number of decision stages in a problem, the higher the value of information. The trend in average running times is also what we would expect. As the number of activities increases, the times required to perform each algorithm also increase but they do it at different rates. The running times for the simulation based BB and the DP increase faster than those of the BB with normal approximation or the SM because the complexities of both the DP and the Simulated BB much more strongly depend on the size of the project. However, even at 75 activities, the running times for BB Simulated and DP are 15 and 11 seconds respectively, which suggests that these methods are viable even for large projects. If we only require good solutions (as opposed to the optimal) we can also use the BB with normal approximation and, that way, achieve significant time savings while sacrificing very little in terms of cost performance (since it is comparable to the BB with simulation).

In problem set 2, we looked at differences in methods by the average distribution span. An interesting point to note here is that, as the spans get larger, the gap between DP and BB methods gets smaller. For projects with the average span of activity distributions, the gap is around 13.6% and goes down to 5.14% for the average span of 24. There is no obvious pattern in the variation of the gap between DP and the case with perfect information. The gap varies from

78.08% for the span of 20 to 132.87% for the span of 12 but the variations seem to be random rather than attributable to any other cause. The average running times tend to increase as the span increases, especially for DP. Spans seem to have a much smaller effect on the running times of the other methods. This is due to the fact that DP explicitly looks at all possible start times and all possible durations of each activity, and those two values are directly tied to the distribution span. The same is not true for the remaining methods. The number of simulation runs in BB Simulated depends only on the number of activities so the rest of the methods are affected only in the calculations of the mean and the variance of each activity.

Finally, in problem set 3, we examine varying cost structures. We only looked at two values of cost structure because, when performing numerical experiments we noticed that projects with cost structure < 0.5 were not very realistic as the crash costs were so low that an algorithm which always crashes all activities to their maxima would actually have a reasonable performance. An opposite is true for cost structures > 1 – the crash costs are large enough that an algorithm that never crashes anything would perform well. Not surprisingly, larger crash costs result in a higher total expected cost; however, running times for the cost structure of 1 seem to be lower than those for cost structure of 0.5, especially for the BB heuristics. Our intuition here is that larger crash costs results in crashing by fewer time periods therefore, each decision stage of the BB methods will terminate sooner.

Because the BB methods perform so well as compared to the DP, we investigated whether there are any special cases for which the BB will give optimal solutions. One that comes immediately to mind is having decreasing crash costs – that is, the crash costs for early activities in the project are high and get lower the further an activity is in the project network. Our intuition was that it is generally preferable to make crashing decisions later in the project because

more information is available. Since the BB is a greedy method where it picks the cheapest activity to crash, having decreasing crash costs would force it to postpone all decisions as late as possible. To illustrate consider Example 3-3 (Table 3-23):

Table 3-23: Serial project -- decreasing crash costs

	O	ML	P	Days to Crash	Crash Cost/day
A	2	3	6	1	$34.00
B	3	4	9	2	$27.00
C	1	3	4	0	--

Assume the target date is 10 days and the penalty for exceeding that date is $100/day. The only decisions need to be made about activities A and B since we cannot crash C at all, so in a sense this reduces to a project with two activities. The dynamic programming solution to this problem is presented in Table 3-24.

Table 3-24: DP solution to Example 3-1

Activity	Start time	Crash by
A	0	1
B	1	0
	2	1
	3+	2
C	N/A	N/A

When we perform the BB method (using exact probability calculations for illustrative purposes), we get the probability distribution of project duration as shown in Table 3-25. The probability of the project exceeding the target equals 0.7322 and is calculated by summing up the probabilities of all durations greater than *Target*.

Table 3-25: Probability distribution of project duration

Nothing Crashed	
Duration	Probability
6	0.000109
7	0.002329
8	0.019293
9	0.078125
10	0.167947
11	0.217838
12	0.206163
13	0.154167
14	0.093251
15	0.043186
16	0.014301
17	0.002951
18	0.000326
19	1.4E-05
P(late)	0.7322

The BBIs are as follows: $BBI_A = 39.22$, $BBI_B = 46.22$, $BBI_C = N/A$, therefore we reduce B by one day and recalculate the BBIs. At the next step, the BBI indices are: $BBI_A = 17.44$, $BBI_B = 24.44$, $BBI_C = N/A$ and again we reduce B by another day. Repeating the procedure, we get the new probability of exceeding the Target of 0.3082. Notice that B has now been reduced to its maximum so we only need to calculate the BB index for A, which is $BBI_A = -3.18$. Since the index of the only activity eligible for crashing is negative, we terminate the procedure. The final policy before the project starts is to not crash A and reduce B by 2 days (again, the decision about B will be postponed until A finishes). This is clearly a different solution than the DP, which required us to crash A by 1 day at the beginning of the project.

We also want to examine the differences between a static (containment) and a dynamic (contingency) algorithm. In this work we present dynamic methods for crashing stochastic projects; however, most of the prior literature discusses static algorithms, therefore we want to assess how much improvement we can gain by making contingent decisions. Figure 3-10 shows dynamic vs. static performance by the size of a project for the BB with simulation. Results are shown for cost structure of 1 and span of 16.

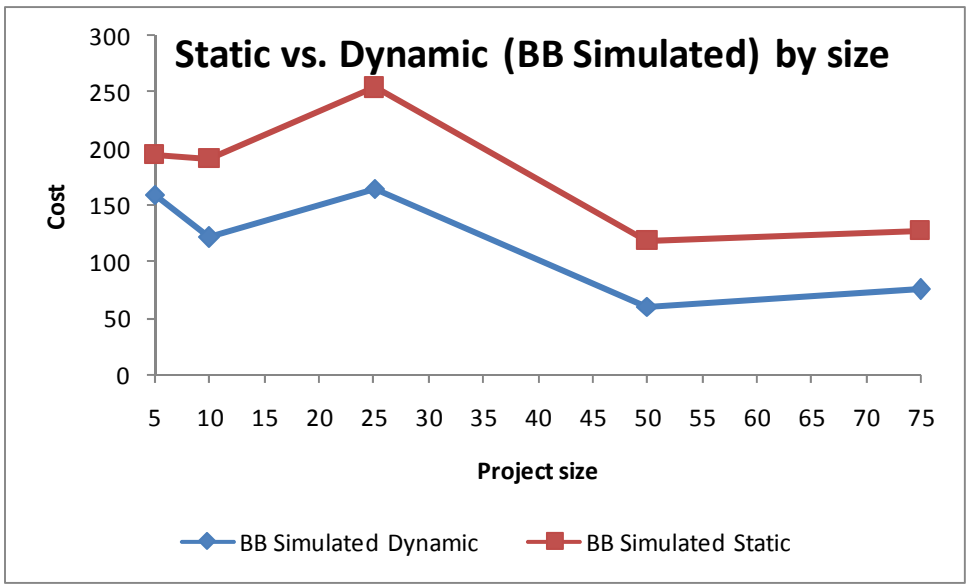


Figure 3-10: Static vs. Dynamic (BB Simulated) by size (linear case)

We would expect the difference between static and dynamic versions to get larger as the project size increases. For projects with one activity only, the results of static and dynamic versions should be identical and the more activities there are in the project (more decision stages) the more opportunities we have to revise our decisions in the dynamic version. Therefore, we would expect the gap between the static and dynamic versions to increase with project size. Nonetheless, the absolute difference between static and dynamic versions in the BB case seems to be relatively constant. However, when we calculate percentage differences, the pattern does emerge although it is not very strong. The percent difference ranges from 22.7 to 69.19.

The next figure (3-11) presents static vs. dynamic versions by average distribution span for the BB with simulation.

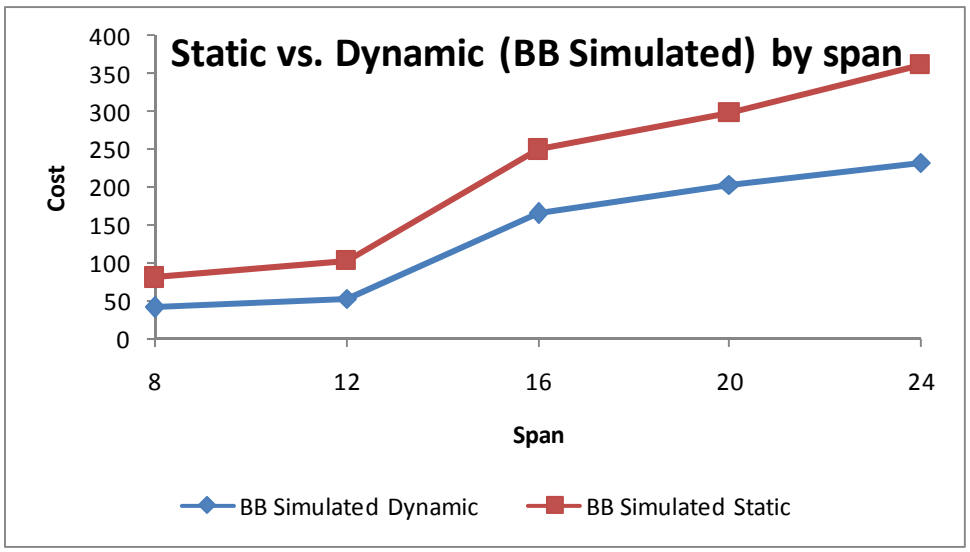


Figure 3-11: Static vs. Dynamic (BB Simulated) by span (linear case)

By increasing distribution spans, we also increase the variance of the project; therefore, we expect to see the gap grow as the span increases, which is the case here.

3.6 ALGORITHMS & COMPUTATIONAL TESTS – NONLINEAR CASE

In addition to looking at external uncertainty or the linear case, we also considered the case with internal uncertainty (nonlinear). Here, the algorithms presented for the linear case need to be slightly modified.

3.6.1 Dynamic Programming

Recall the DP formulation from section 3.4.1. For the last (dummy) activity we calculated the cost to go as:

$$Cost^{*}(END, t_{END}) = \max\{0, (t_{END} - Target)\} \times Penalty$$

where:

*Cost** – optimal cost to go,

i – stage of the problem (denotes the activity about to start),

t_i – start time of activity *i*,

Target – the target date for the project,

Penalty – penalty per day for exceeding the *Target.*

The formulation for the terminal activity is still applicable to the nonlinear case. The differences are in the calculations for the remaining activities. Recall that for activity C we had the following:

$$Cost^*(C, t_C) = \min_{z_C}\{\sum_k p_k^C (Cost*(END, t_C + k - z_C)) + c_C \times z_C\}$$

where:

z_i – number of time periods to crash activity *i*

c_i – cost of crashing activity *i* by one day

p_k^i – probability that normal duration of activity *i* will be *k* days

However, now both z_C and c_C depend on the option chosen and the normal duration of the activity. We define $z_{l,k}^C$ as the number of days we will shorten activity C using crash option *l* if C's normal duration turns out to be *k*, and $c_{l,k}^C$ as the crash cost of shortening activity C by $z_{l,k}^C$. For instance, using Example 3.2, we get $z_{1,7}^C = 2$, $c_{1,7}^C = \$50.40$, $z_{2,10}^C = 5$, $c_{2,10}^C = \$120.00$. The cost to go for activity C is therefore calculated as:

$$Cost^*(C, t_C) = \min_l\{\sum_k p_k^C (Cost^*(END, t_C + k - z_{l,k}^C) + c_{l,k}^C \times z_{l,k}^C)\}$$

The general formulation for any non-terminal activity is

$$Cost^*(i, t_i) = \min_l\{\sum_k p_k^i (Cost^*(i+1, t_i + k - z_{l,k}^i) + c_{l,k}^i \times z_{l,k}^i)\}$$

3.6.2 Biggest Bang

For the Biggest Bang algorithm, we only considered the simulated case. This is because in the nonlinear crashing, the probability distribution of project duration changes shape and not simply shifts as illustrated in Figure 3-3. In such circumstances, assuming that the project duration will follow normal distribution may be unreasonable.

Recall that in the linear case we simulated the project to find the probability that the project's duration will exceed the target date. Then, the BB indices were simply calculated as $BBI_i = P(project\ duration > Target) \times Penalty - c_i$ where c_i is the crash cost of activity i. However, this formulation assumes that we always crash the project in one time period increments and that the per day crash cost for each activity is constant. Those assumptions are now violated in the nonlinear case. The only way to get reasonably good estimates of cost savings from crashing a particular activity is to simulate the project with that option chosen. For instance, using Example 3-2, the BB steps are as follows:

(1) Simulate the project with no crashing. Record the total expected cost, denoted $Cost_{NC}$.

(2) Simulate the project assuming Option 1 for activity A. Record the total expected cost ($Cost_1^A$)

(3) Simulate the project assuming Option 2 for activity A. Record the total expected cost ($Cost_2^A$).

(4) Continue in this manner until all options for all activities are explored.

(5) Choose the activity/option combination that resulted in the lowest expected cost.

(6) Repeat steps (2) through (4) assuming the option from step (5) is also executed.

(7) Repeat steps (5) and (6) until we have options chosen for all activities.

(8) Act on the solution that yields the lowest expected cost. Again, we only execute decisions for those activities that are starting immediately, postponing other decisions until we have to make them.

3.6.3 Simple Minded method

As before, the SM uses the expected project duration to make decisions. It iteratively crashes the cheapest activity until expected project duration is less than or equal to *Target*. The only difference is in identifying the cheapest activity to crash. Since the crash costs vary based on the normal duration as well as the option chosen, the SM calculates the **expected** crash cost **per day**. Using Example 3-2, we calculate the expected crash costs/day for activity A are as follows:

(1) Discretize the probability distribution of A's duration. we get P(duration=2) = 0.125, P(duration=3) = 0.75, and P(duration=4)=0.125.

(2) Calculate expected crash cost per day for Option 1:

Table 3-26: Activity A, Option 1

		Option 1 (25% reduction)		
Duration	Probability	days crashed	Crash Cost	Crash Cost per day
2	0.125	1	6	6
3	0.75	1	9	9
4	0.125	1	12	12

E(crash cost/day) for option 1 = $0.125 \times 6 + 0.75 \times 9 + 0.125 \times 12 = 9$

(3) Calculate expected crash cost per day for Option 2:

Table 3-27: Activity A, Option 2

		Option 2 (50% reduction)		
Duration	Probability	days crashed	Crash Cost	Crash Cost per day
2	0.125	1	10	10
3	0.75	2	15	7.5
4	0.125	2	20	10

$$\text{E(crash cost/day) for option 2} = 0.125\times 10 + 0.75\times 7.5 + 0.125\times 10 = 8.125$$

We perform similar calculations for all other activities in the project. The rest of the algorithm proceeds in the same manner as in the linear case (see Section 3.4.3).

3.6.4 Generating problem instances

Recall the problem generator described in Section 3.5. Herein we follow the same steps with an exception of step 4 ("Determine the maximum number of days by which each activity can be crashed *(d_i)*"), step 5c ("Calculate adjustment value for each activity"), and step 5f ("Calculate crash cost per day for each activity") which are not needed as well as step 5d ("For each activity, generate a random number ~U[0, 1) and multiply it by A_i. We refer to this value as a cost distribution index (*CDI*)") and step 5e ("Calculate cost of maximum crashing for each activity") which are slightly modified. The complete procedure is outlined below:

(1) Determine the size of the project. We tested problems with 5, 10, 25, 50, and 75 activities.

(2) Set distribution span. For simplicity and convenience, we only looked at a span of 16.

(3) For each activity generate optimistic, most likely, and pessimistic times. *ML* and *P* times are derived from a geometric distribution with the mean equal to the span. Optimistic

duration is also derived from a geometric distribution; however, we decided to fix the mean at 8 regardless of the span. We wanted to modify only the width of the project duration distribution without shifting it to the right at the same time.

(4) Generate crash cost per day for each activity

a. Estimate expected penalty cost E(Penalty) of the project with no compression using Monte Carlo simulation.

b. Assign a fraction of E(Penalty) as the total cost for maximum crashing (*TCMC*) of all activities (total cost if we crash all activities to maximum). We refer to this fraction as a *cost structure* and, unlike in the linear case we the cost structure of 1 only.

c. For each activity, generate a random number ~U[0, 1). We refer to this value as a cost distribution index (*CDI*).

d. Calculate a normalized cost distribution index (*NCDI*) for each activity:

$$NCDI_i = \frac{CDI_i}{\sum_{k=1}^{n} CDI_k}$$

e. Calculate regular cost for each activity:

$$\text{regular cost}_i = TCMC \times NCDI_i$$

f. Calculate crash options for each activity. In this research we investigated six duration reduction options: 0%, 10%, 20%, 30%, 40%, and 50%. The crash durations and crash costs are calculated as described in Section 3.1.

3.6.5 Computational results

For convenience, in case of nonlinear crashing we only examined problems with the span of 16 and the cost structure of 1. Table 3-28 and Figure 3-12 show the expected project cost by size. The average running times are presented in Table 3-29 and Figure 3-13.

Table 3-28: Expected cost (nonlinear case)

Size	DP	BB Simulated	SM
5	9.55	43.12	223.82
10	9.00	67.89	454.32
25	5.73	85.16	564.64
50	1.56	47.92	1053.97
75	6.46	30.17	1477.92

Table 3-29: Average running time (nonlinear case)

Size	DP	BB Simulated	SM
5	0.3514	0.1474	0.0043
10	3.8471	1.2990	0.0771
25	22.0861	6.6343	0.2216
50	100.5802	22.8508	1.0065
75	173.9868	38.2013	3.4804

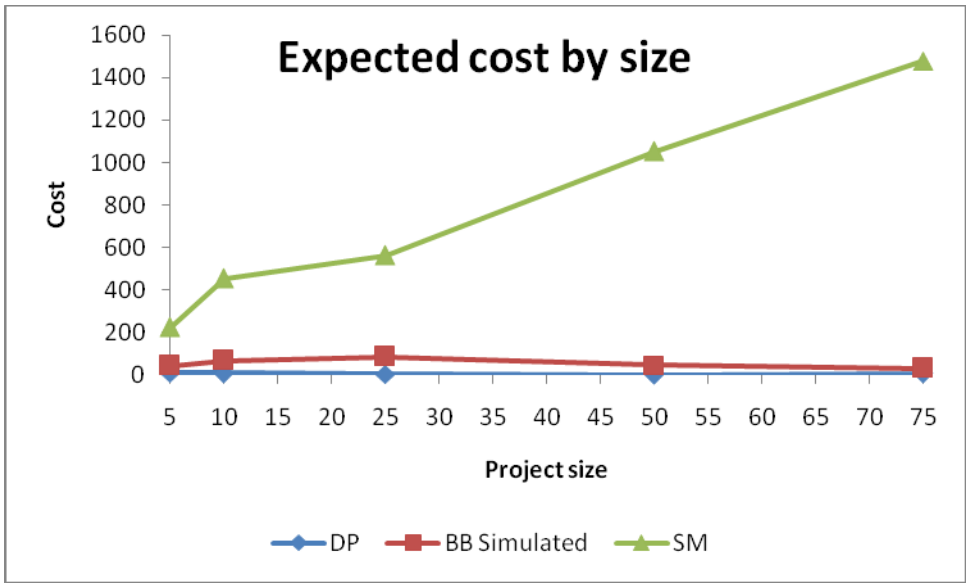


Figure 3-12: Expected cost by project size (nonlinear case)

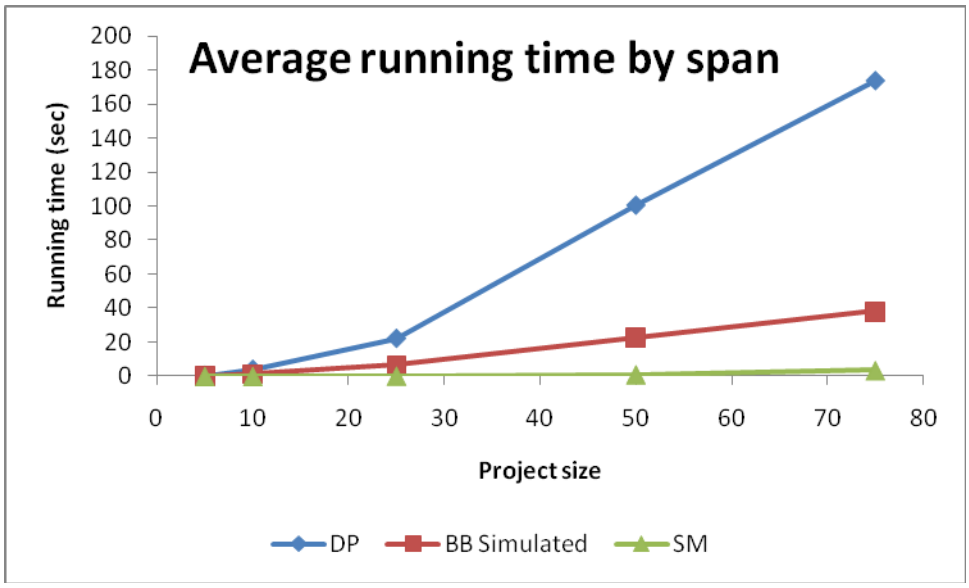


Figure 3-13: Average running times in seconds (nonlinear case)

As in the linear case, the SM method performs poorly; however, the differences between the SM and the optimal solutions (DP) are even more pronounced. The BB algorithm appears to be doing well but gaps are again higher than in the linear case. Not surprisingly, the running times of all the methods are much higher than in the linear case. This and the fact that the

expected costs of the heuristics are, on average, further away from the optimal solution, suggests that problems with nonlinear crashing are harder to solve.

3.7 CHAPTER SUMMARY

In this chapter we studied a special case of project networks where all activities need to be performed sequentially (in a series). We developed three methods finding a crashing policy when task durations are stochastic. We considered both linear as well as nonlinear crash costs.

We showed that the dynamic programming can solve large problem instances to optimality; therefore it should be the method of choice when dealing with serial projects or projects with one dominant path. In addition, we showed that a greedy, simple minded method performs very poorly and should be avoided whenever possible. Finally, we also showed that the Biggest Bang method, which calculates expected savings from each decision, performs very well when crash costs are linear and gives reasonable solutions for projects with nonlinear crash costs. In the linear case we presented two methods of obtaining expected savings for the BB: (1) a simulation method and (2) normal approximation. Even though both methods perform well, we feel that the BB with simulation is a better choice. When the number of activities in the project is small, the normal approximation may not be reasonable. Furthermore, the BB approximates combined probability distributions of those activities that are not yet completed. Therefore, the further we are in the project execution, the less reasonable the normality assumption.

4.0 CHAPTER FOUR: GENERAL PROJECTS

4.1 INTRODUCTION

Effective project management requires making decisions under conditions of high uncertainty. Project managers are under significant pressure to complete projects on time and within budget, otherwise they may face large penalties for late completion. While there can be many aspects of such uncertainty, herein we focus on the uncertainty of task durations. In contrast to serial projects considered in Chapter 3, we now consider general projects - that is, projects with multiple paths. While some general projects can be reduced to a serial project due to the presence of one dominant path (as discussed in Chapter 3), it is often not possible to do so. Thus, a general project is a common situation in practice. Herein, we focus on the stochastic time-cost trade-off problem for a general project – similar to what we did for serial projects in Chapter 3.

As in Chapter 3, we again have a given target date, a per period penalty for exceeding that target date, linear crash costs, maximum allowed compression, and a given probability distribution of duration for each activity. In this research we assume that duration of each activity follows a triangular distribution; however, methods derived herein are applicable to any probability distribution. Further, we are assuming that crashing decisions require no lead time and all activities start as soon as possible. In addition we have no resource constraints. Below is a simple example of such a problem.

Example 4.1

Consider a project with five tasks as follows:

Table 4-1: Illustrative Example #1

Task	Predecessor	Days Duration Optimistic	Most Likely	Pessimistic	Crash cost $ per day	Crash days Maximum
A	none	2	3	4	15	1
B	none	3	5	8	20	2
C	B	2	3	5	18	1
D	C	2	3	6	22	2
E	A, B	4	8	12	17	2

Suppose that we have a target of 12 days for finishing the project and a penalty cost of $100 per day for each day that the target date is exceeded. A network representation for this problem is given below.

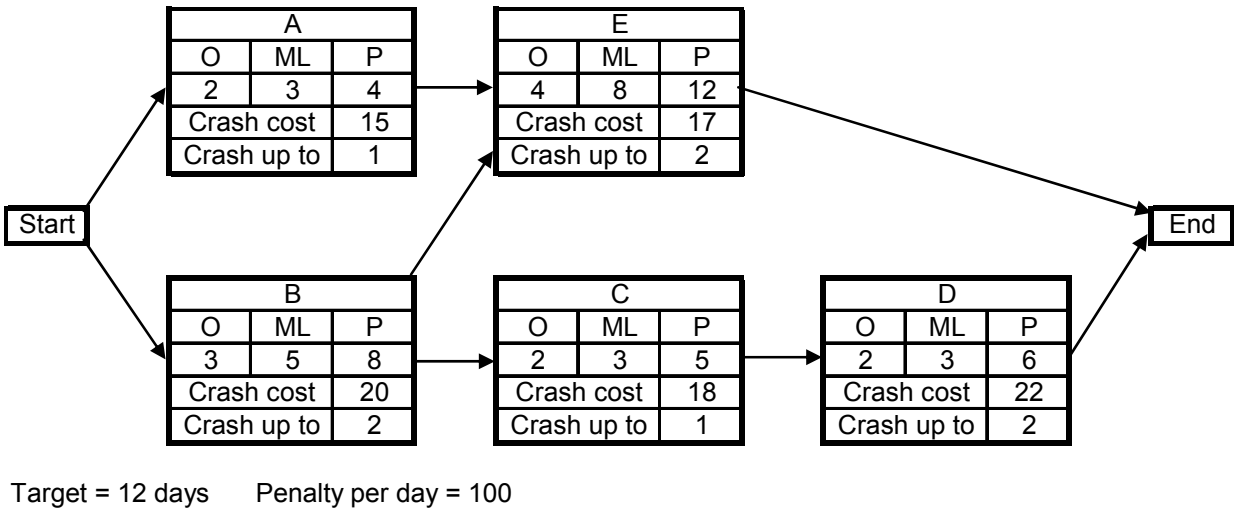


Figure 4-1: Illustrative Example #4.1

It is important to note that a decision on crashing an activity in such problems depends not only on its cost, the penalty cost and the elapsed time of the project, but it also depends on the status of activities on other paths in the network. As an illustration, consider activity D in Example 1. Suppose that 12 days have already elapsed in the project. If all other activities except D are finished, then we would clearly want to speed up D. If, however, activity E had just started then it is not obvious whether or not we should speed up D. Herein we consider such a contingency approach to making crashing decisions (we refer to these as conditional decisions) in which the status of all other activities at the time of a decision is considered. This is in contrast to deriving one static crashing policy at the beginning of the project.

As we have shown in Chapter 3, dynamic programming (Bellman, 1957) is a viable solution method for deriving optimal decisions in serial projects. The reason for this is that in the serial case the state of a project can be fully described by current time t and the activity being considered for crashing. Unfortunately, for a general project the state is not that simple to describe. In addition to knowing the current time t and the activity under consideration for crashing, we also need to know what is happening with the other paths in the project. Thus, for a general project, dynamic programming quickly becomes intractable as the number of activities and as the complexity of the network increase. As noted by Elmaghraby (2005), "any approach that aspires to confront uncertainty [in general project planning] head on is computationally overwhelming."

Since we know that getting an exact (optimal) solution to the stochastic time-cost trade-off problem for general projects is impossible in most cases, herein we propose and evaluate a variety of heuristic procedures for obtaining good solutions. These include the Bang for the Buck, the Biggest Bang, Basic Linear Programming, Linear Programming with Dynamic

Programming, and Protect the Critical Path. The rest of this chapter is organized as follows. Section 4.2 reviews previous research on this problem, section 4.3 describes the algorithms developed herein, section 4.4 outlines a procedure for randomly generating projects to use in evaluating the heuristics, section 4.5 presents our findings, and finally section 4.6 reviews our conclusions.

4.2 PREVIOUS RESEARCH

PERT is one well known method for dealing with uncertainty in projects. This approach, however, has a lot of shortcomings which are described in more detail in Klastorin (2004) and Johnson and Schou (1990). A significant problem with PERT is that it assumes one unique critical path. However, in reality, other paths may become critical since the duration of each task is stochastic. Van Slyke (1963) introduced Monte Carlo simulation method for estimating the probability that an activity will be on the critical path – a so called criticality index. Several improvements to the original method of Van Slyke have been proposed by Herbert (1979), Kulkarni and Adlakha (1986), and Ioannou and Martinez (1998). These methods, however, do not address problems in which we have options of speeding up some or all of the activities. Nonetheless, simulation can be useful in evaluating different crashing policies. Johnson and Schou (1990) applied simulation to test three crashing rules and found one of those (Rule 3), which is dependent on the crash cost and on the criticality of a task, to be most appropriate under most general conditions since it is a combination of the other two rules. While, Johnson and Schou considered projects with stochastic durations and linear crashing, they did not consider any penalty for project completion beyond the target date. Haga and O'Keefe (2001) and Haga

(1998) investigated another crashing method in which they tested each activity in the network that has not yet been maximally compressed to determine which crashing decision would reduce the mean expected total cost (including crash cost and penalty for exceeding the target date) the most.

Gutjahr, Strauss, and Toth (2000) showed that the deterministic discrete time-cost problem (that is, a project with known task durations, speed-up options, and a penalty function for exceeding the target date) is NP-hard. It follows, of course that, the stochastic discrete time-cost problem is also NP-hard since it is a generalization of its deterministic counterpart. The authors also proposed a hybrid algorithm based on simulated annealing and importance sampling (a rare event simulation procedure) to solve the stochastic problem. Simulated annealing was used to generate a new policy, which was then evaluated by simulating the project and estimating the value of the objective function (crash cost plus penalty cost for exceeding the target date). Gutjahr, Strauss, and Wagner (2000) developed a stochastic branch-and-bound algorithm for crashing tasks in projects, which uses importance sampling as a method for estimating the value of the objective function.

Mitchell (2005) examined a slightly different problem where the source of uncertainty is not linked to task durations but rather to an occurrence of some disruptive event. If a disruptive event does occur, all the work on the project has to stop. The objective is to develop a crashing policy that would minimize total expected cost (crash cost plus overhead cost). The time before the disruptive event as well as the duration of the disruptive event are random variables with known probability distributions. The author investigated cases with and without a due date – in the case where the due date is specified, a penalty cost is incurred if the total project duration exceeds that date.

The above mentioned papers present viable methods for selecting speed-up options in general, stochastic networks; however they do not consider conditional decisions, that is decisions dependent on the current state of a project. One attempt to incorporate contingency planning into crashing policy selection was made by Wollmer (1985) who investigated the use of stochastic programming to solve a stochastic time-cost trade-off problem with discrete probability distributions of task durations. Another one is a lesser known influence diagram approach by Jenzarli (1995); however, beyond the work of Wollmer and Jenzarli there are not any published efforts that address conditional decisions in such problems. This research aims to fill this gap by providing a contingency planning approach to speeding-up projects under uncertainty.

4.3 STATEMENT OF THE RESEARCH PROBLEM

The research problem considered in this chapter is a stochastic time-cost trade-off problem for general networks where uncertainty comes only from activity durations. Given the set of activities A={1,…,*n*} where *n* is the total number of activities excluding dummy start and end tasks, a set of precedence relationships Π (where Π_i is a set of immediate predecessors of activity *i*), discrete triangular probability distributions of task durations where O_i = optimistic duration of activity *i*, ML_i = most likely duration of activity *i*, and P_i = pessimistic duration of activity *i*, maximum number of time periods to crash each activity (d_i), per day crash cost for each activity (c_i), a project target date (*Target*), and a per day penalty for exceeding the target date (*Penalty*); find a crashing policy that minimizes total **expected** project cost. Approaches discussed in this

chapter are applicable to any probability distributions; however, for convenience, we limit our discussion to triangular distributions.

Furthermore, we assume that duration of a task is discrete and becomes known only after that task is fully completed. Crashing is also discrete (integer), that is, we can reduce the duration of an activity by a multiple of a full time period only. As in the serial case, we make crashing decisions dynamically throughout the execution of the project; a new decision-making stage occurs before the start of each activity. However, unlike the serial case, we may not have full information about previously started activities. It is likely that, at the time of a crashing decision, there will be some activities that are already completed (we have full information about their durations), some activities that have not yet started (we have no information beyond their probability distributions), and some tasks in progress (we have partial information about their durations). Therefore, we need to update the probability distributions of in-progress activities accordingly. Consider a project with 3 activities (Example 4.2) as illustrated in Figure 4-2:

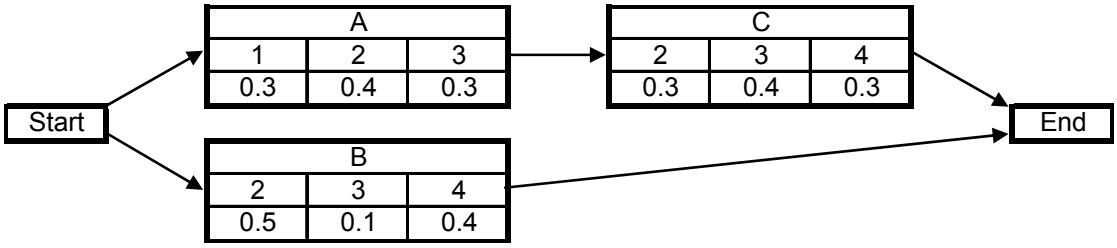


Figure 4-2: Illustrative Example #4.2

Activity A has the following possible durations: 1, 2, or 3 with probabilities 0.3, 0.4, and 0.3 respectively. Crashing information is omitted because it is not relevant to this problem. Activities A and B start at the same time (t=0). Assume that two days have elapsed and suppose that activity A is finished while B is still in progress. If we wanted to make crashing decisions about C at this point, we need to be able to calculate the expected duration of B. Originally, that

value was $2(0.5)+3(0.1)+4(0.4)=2.9\approx 3$; however, now we know that B is going to take *more* than 2 days. Therefore, using Bayesian probability update, the probability distribution of activity B becomes 3 with probability of $0.1+\frac{0.1}{0.1+0.4}(0.5)=0.2$ and 4 with the probability of $0.4+\frac{0.4}{0.1+0.4}(0.5)=0.8$. The new expected duration is therefore $3(0.2)+4(0.8)=3.8\approx 4$.

It is also important to note that we use the expected cost criterion to evaluate algorithms presented herein. This criterion is most commonly used when evaluating uncertainty and considers the average behavior of a method, or average case. Other criteria (not discussed here) one could use include optimizing best case, worst case, or variance.

4.4 ALGORITHMS

In this section, we present a number of algorithms: Biggest Bang, Bang for the Buck, Basic LP, Linear Programming with Dynamic Programming, and Protect the Critical Path. These algorithms can be divided into three groups: methods that: (1) focus on uncertainty, (2) focus on path interdependencies, and (3) hybrid methods that focus on both uncertainty and path interdependencies. Bang for the Buck (BFB) and the Biggest Bang (BB), known already from the serial project chapter, are in group (1). The Basic LP (LP) algorithm is in group (2), and finally, Linear Programming with Dynamic Programming (LPDP) and Protect the Critical Path (PCP) are in group (3). To illustrate how each method works, we will use the simple project presented in Figure 4.1.

4.4.1 Biggest Bang

The Biggest Bang (BB) utilizes a concept of a criticality index of a task, which is simply the probability that the task is critical. Since our objective is to minimize total expected cost (including any penalty cost), we are more interested in a probability that crashing an activity will result in a lower penalty cost. Therefore, we developed a concept of the Penalty Target Criticality Index (PTCI), which we calculate using simulation. At each simulation replication, an activity is considered Penalty Target Critical if and only if such an activity lies on the longest path in the project network and the duration of the longest path is strictly greater than Target date. The PTCI of an activity is therefore equal to the fraction of the time the activity was Penalty Target Critical. The BB index (BBI) for each activity can be calculated as:

$$BBI_i = PTCI_i \times Penalty - c_i$$

As time progresses in a project, at the beginning of each task we must make a speedup decision. For a given task, the decision on how much, if at all, to speed it up depends not only on its cost but also on the expected benefits (expected savings in penalty costs). Also, the decision must be weighed against the costs and possibilities of speeding up subsequent tasks in the project. The BB method uses a greedy approach to make such a decision, that is, at each decision stage it iteratively chooses to crash the activity which provides the highest expected savings. The decision stage occurs whenever an activity (or activities) is about to start.

The BB algorithm consists of the following steps:

(1) At decision stage *i*, simulate the project *N* times. At each simulation replication identify those activities which are Penalty Target Critical.

(2) For each activity calculate the Penalty Target Criticality Index, namely the number of times the activity was Penalty Target Critical divided by *N*.

(3) Calculate the BB index for each activity.

(4) Select the activity with the highest BB index and reduce it by one time period (since BBI calculates expected savings from crashing an activity, we want to reduce tasks with high BB indices).

(5) Repeat steps (1) through (4) until we either reduced all activities to their maximum or until the BB indices of those activities still eligible for crashing are negative.

(6) Make crashing decisions, according to the policy derived, for those activities that are starting immediately. Postpone all other decisions until later.

(7) Repeat steps (1) through (6) at decision stage *i*+1 until we reach the end of the project.

To illustrate how the BB algorithm works, consider the project from Figure 4.1. After simulating the project *N* times, we obtain the following BB indices:

Table 4-2: BB indices -- iteration 1

Activity	*PTCI*	*BBI*
A	0.017	-13.300
B	0.752	55.200
C	0.242	6.200
D	0.242	2.200
E	0.594	42.400

Because activity B has the highest BB index, we would select it for reduction by one day and recalculate BB indices. The next 3 iterations are presented in Table 4.3.

Table 4-3: BB indices -- iterations 2-4

	Iteration 2		*Iteration 3*		*Iteration 4*	
Activity	***PTCI***	***BBI***	***PTCI***	***BBI***	***PTCI***	***BBI***
A	0.059	-9.100	0.119	-3.100	0.045	-10.500
B	0.535	33.500	0.313	11.300	0.202	0.200
C	0.155	-2.500	0.065	-11.500	0.082	-9.800
D	0.155	-6.500	0.065	-15.500	0.082	-13.800
E	0.437	26.700	0.319	14.900	0.161	-0.900

At iteration 2, activity B again had the highest BB index so we reduce it by another day. Notice that task B has now reached its maximum compression. At iteration 3, we select E for reduction by one day since it has the highest BB index. At iteration 4, all activities eligible for crashing have negative expected savings therefore we terminate the procedure. The final *static* policy before the project starts would be to crash B by 2 days and E by 1 day. However, since BB is a dynamic algorithm, we only need to crash those activities that are starting immediately, that is activity B. We will postpone decisions about tasks C, D, and E until later. The decision at the start of the project is therefore to crash B by 2 days and to not crash A. We will repeat this procedure once we are able to start additional activities (in this case only after B ends).

Observe that the BB algorithm handles uncertainty by using criticality indices; however, it does not really consider path interdependencies. We do not know what impact crashing an activity will have on other activities until we recalculate the PTCIs. It is a myopic method in a sense that, in determining the best policy, it greedily picks an activity with the highest index to crash.

4.4.2 Bang for the Buck

Bang for the Buck (BFB) is a slight modification of the BB algorithm. Instead of calculating expected savings, we calculate expected savings per dollar spent. The BFB also uses the concept of the Penalty Target Criticality Indices and the BFB index (BFBI) for each activity can be calculated as:

$$BFBI_i = \frac{PTCI_i \times Penalty - c_i}{c_i} = \frac{PTCI_i}{c_i} \times Penalty - 1$$

BFBI gives the expected cost savings per dollar spent. Therefore, the larger the expected savings, the higher the BFBI; the larger the crash cost, the lower the BFBI. Therefore, to maximize impact per dollar spent we want to crash an activity with the highest BFBI.

Using the project in Figure 4.1 we get the following indices:

Table 4-4: BFB indices -- iteration 1

Activity	*PTCI*	*BFBI*
A	0.018	-0.880
B	0.768	2.840
C	0.236	0.311
D	0.236	0.073
E	0.616	2.624

Since activity B has the highest BFB index, we would reduce it by one time period (a day in this case). Because PTCIs might have changed due to shortening activity B by one day, we need to recalculate the BFBIs. The next 3 iterations are presented in Table 4.5.

Table 4-5: BFB indices -- iterations 2-4

	Iteration 2		Iteration 3		Iteration 4	
Activity	*PTCI*	*BFBI*	*PTCI*	*BFBI*	*PTCI*	*BFBI*
A	0.065	-0.567	0.133	-0.113	0.053	-0.647
B	0.538	1.690	0.296	0.480	0.193	-0.035
C	0.161	-0.106	0.065	-0.639	0.067	-0.628
D	0.161	-0.268	0.065	-0.705	0.067	-0.695
E	0.441	1.594	0.335	0.971	0.165	-0.029

The final *static* policy before the project starts would be to crash B by 2 days and E by 1 day. However, like the BB, the BFB is a dynamic algorithms and we disregard any decisions that do not have to be made right away. Thus, we crash B by 2 days and hold off on making any further decisions until more information is available. In the example shown, the starting policies for BBF and BB are the same. The differences among the PTC indices in Tables 4-2, 4-3 and 4-4, 4-5 arise from the fact that we use simulation to *approximate* the true probability of realizing cost savings due to lower penalties. However, the BB and BFB algorithms can, of course, yield different decisions. Consider a simple project in Figure 4-3. Assume that activity A has a Penalty Target Criticality Index of 0.4 and activity B has an index of 0.6. The per day crash costs are $10 and $20 for activities A and B respectively.

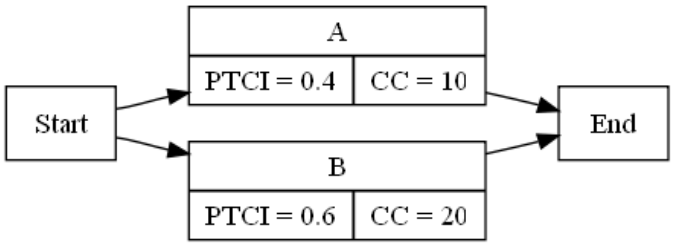


Figure 4-3: Illustrative Example #4.3

Assuming a per day penalty of $100, we get the following indices:

$$BBI_A = 0.4 \times 100 - 10 = 30 \qquad BFBI_A = \frac{0.4 \times 100 - 10}{10} = 3$$

$$BBI_B = 0.6 \times 100 - 20 = 40 \qquad BFBI_B = \frac{0.6 \times 100 - 20}{20} = 2$$

Therefore, using the BB algorithm, we would crash activity B (index of 40), while using the BFB algorithm results in crashing activity A (index = 3).

One of the possible future directions for this work is to include budget constraints. Since the BFB effectively considers the expected cost savings per dollar spent, it may be more beneficial for such problems. The BFB is also a modification of (and indeed an extension of) Rule 3 due to Johnson and Schou (1990). Recall that, Johnson and Schou's method was applicable to a stochastic time-cost trade-off problem with linear crash costs but no target date and no penalty for exceeding that date. We generalized this rule so that it can also be applied to our problem. Johnson and Schou's Rule 3 crashes an activity with the lowest expected marginal crash cost per time period.

$$\text{Marginal crash cost per time period} = \frac{Cost_{Crash} - Cost_{Normal}}{CI\left\{E[dur_{Normal}] - E[dur_{Crash}]\right\}}$$

where: $Cost_{Normal}$ = total (expected) cost of performing a task under its normal duration
$Cost_{Crash}$ = total (expected) cost of performing a task under its crash duration
CI = criticality index of a task
$E[dur_{Normal}]$ = expected normal task duration
$E[dur_{Crash}]$ = expected crash task duration

In our case, $Cost_{Crash} - Cost_{Normal}$ reduces to the crash cost per time period (c_i) since we have linear crash costs, and $E[dur_{Normal}] - E[dur_{Crash}]$ reduces to one since we consider crashing a task by one time period at a time. Thus, we want to crash an activity with the lowest marginal crash cost per time period ($\frac{c_i}{CI_i}$), which is equivalent to crashing an activity with the largest $\frac{CI_i}{c_i}$.

Recall that we that BFB indices are calculated as $BFBI_i = \frac{PTCI_i}{c_i} \times Penalty - 1$. If we did not have a penalty or a target date, the two methods for choosing which activity to crash would be

equivalent; however, the algorithms themselves would still be different. Unlike Johnson and Schou method, which is a static policy (all decisions made at the beginning of a project), both BFB and BB are dynamic, that is, they adapt their policies to the current state of the project.

Like the BB algorithm, the BFB does not consider path interdependencies. Similar to the BB, the BFB picks an activity with the highest index to crash in order to determine the best policy and does not consider the effects this action may have on other paths. Even in a deterministic case, crashing the cheapest activity on the critical path sequentially may not lead to the best solution. In the deterministic case, the best approach is to use optimization to find the best combination of activities to crash, which is a method used in the remaining algorithms presented in this chapter.

4.4.3 Basic LP

The basic LP method tries to address the main shortcoming of the BFB and BB algorithms, that is, not considering path interdependencies. Our formulation follows the standard LP approach for crashing AON networks, where expected durations of activities not completed are used instead of their deterministic values. The objective function is to minimize total cost, that is, total crash cost plus any penalty cost, subject to precedence relationships constraints (a task cannot start until all its predecessors are finished), maximum crash limit constraints, and a project finish time constraint, or *Target*.

Variables:

t_i = starting time of activity i

z_i = number of time periods to crash activity i

θ = number of time periods the project duration exceeds Target date (project lateness)

Constants:

c_i = cost to crash activity i by one time period

d_i = upper limit on the number of time periods we are allowed to crash activity i

$E[dur_i]$ = expected duration of activity i (for triangular distribution, $\frac{O_i + ML_i + P_i}{3}$)

Π_i = set of immediate predecessors of activity i

Penalty = cost per day for exceeding target date

Target = target date for project completion

Formulation:

$$\min \sum_i^N c_i z_i + Penalty \times \theta$$

$$s.t.\ \ t_i - t_k + z_k \geq E[dur_k] \quad where\ k \in \Pi_i \quad \forall k, \forall i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$t_{END} - \theta \leq Target$$

$$z_i, t_i, \theta \geq 0$$

Using the example project from Figure 4.1, we get the following formulation at time $t = 0$.

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 100\theta$$

$$s.t.\ \ t_C - t_B + z_B \geq 5^{(*)} \qquad z_A \leq 1$$

$$t_E - t_B + z_B \geq 5^{(*)} \qquad z_B \leq 2$$

$$t_E - t_A + z_A \geq 3 \qquad z_C \leq 1$$

$$t_D - t_C + z_C \geq 3^{(*)} \qquad z_D \leq 2$$

$$t_{END} - t_E + z_E \geq 8 \qquad z_E \leq 2$$

$$t_{END} - t_D + z_D \geq 4^{(*)}$$

$$t_{END} - \theta \leq 12$$

$^{(*)}$ Rounded to the nearest integer

The solution to this LP is to reduce task E by 1 day at a total cost of 17. Since this is a dynamic algorithm, and at time $t = 0$ we cannot start E, the decision is to crash nothing and start activities A and B. We resolve this LP substituting known durations for $E[dur_i]$, for all activities already completed or recalculating $E[dur_i]$ for those activities in progress, and setting maximum crash

limit constraints to equalities for activities already started or completed (so at the next stage, after activity B finishes, we would set $z_A = 0$ and $z_B = 0$).

Even though the LP algorithm takes into considerations path interdependencies, it does not handle uncertainty very well since it replaces uncertainty with expected values (much like PERT). The following approach exploits the advantages of BFB/BB (uncertainty) and LP (path interdependencies) into a single algorithm.

4.4.4 Linear Programming with Dynamic Programming

Linear Programming with Dynamic Programming (LPDP) method uses the uncertainty handling ability of the dynamic programming algorithm while at the same time considering path dependencies through the use of LP. The motivation for using the DP algorithm came from the serial project case. We know that for serial projects or for projects with one dominant path, the DP algorithm guarantees optimal solutions. Exploiting this property, the LPDP algorithm performs DP on the PERT critical path and uses LP for the non-critical activities.

State representation:

$x = (i, t_i)$ where

i ={1,..,*END*} – current activity
N = number of non-dummy activities
t_i – starting time of activity *i*

z_i – number of time periods to crash activity *i*

c_i – cost of crashing activity *i*

Cost*(x) – the optimal cost-to-go for state *x*

p_k^i -- probability that normal duration of activity *i* will be *k* days

Target – target date for project completion

Penalty – cost per day for exceeding *Target*

For the terminal dummy activity (END):
For all t_{END} compute:

$$Cost^*(END, t_{END}) = \max\{(t_{END} - Target), 0\} \times Penalty$$

For all activities i, where $i \le N$:
For all t_i do:

$$Cost^*(i, t_i) = \min_{z_i}\{\sum_k p_k^i (Cost^*(i+1, t_i + k - z_i)) + c_i \times z_i\}$$

Exhibit 4-1: Dynamic Programming formulation for serial projects

The LPDP algorithm consists of the following steps: (1) find a PERT critical path – this is done by calculating the longest path using expected durations for all activities, (2) perform DP on the PERT critical path using formulation from Chapter 3 (see Exhibit 4-1), (3) formulate and solve a mixed integer program to combine DP with LP (using expected durations), (4) from the solution consider only those activities that are starting immediately and postpone all other decisions, (5) repeat the procedure at each stage of the project (when we can start another activity) substituting known durations for the expected.

We need to modify the basic LP model to account for the DP solution on the PERT critical path. To do this we introduce a new set of variables (z_{ij}) and a corresponding set of parameters (p_{ij}). Parameters p_{ij} correspond to the j^{th} level of the DP solution for activity *i* and we refer to them as crossover points that indicate the last start time before the number of days to

crash an activity increases. Variables z_{ij} are set to 1 if the start time of an activity is less than or equal to the corresponding crossover point.

Variables:

t_i = starting time of activity i

z_i = number of time periods to crash activity i

θ = number of time periods the project duration exceeds Target date (project lateness)

z_{ij} = binary variables to look up DP policy

Constants:

c_i = cost to crash activity i by one time period

d_i = upper limit on the number of time periods we are allowed to crash activity i

$E[dur_i]$ = expected duration of activity i (for triangular distribution, $\frac{O_i + ML_i + P_i}{3}$)

Π_i = set of immediate predecessors of activity i

Penalty = cost per day for exceeding target date

Target = target date for project completion

p_{ij} = crossover point j for activity i (for DP policy lookup) – defined as the last start time before the number of days to crash an activity increases.

Formulation:

$$\min \sum_{i}^{N} c_i z_i + Penalty \times \theta$$

$$s.t. \quad t_i - t_k + z_k \geq E[dur_k] \quad where\, k \in \Pi_i \quad \forall k, \forall i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$z_i = \sum_j z_{ij} \qquad \forall i \text{ not yet started}$$

$$t_i - p_{ij} \leq \mathrm{M} z_{ij} \qquad \forall j, \forall i$$

$$t_i - (p_{ij} + 1) z_{ij} \geq 0 \qquad \forall j, \forall i$$

$$t_{END} - \theta \leq Target$$

$$z_i, t_i, z_{ij}, \theta \geq 0$$

Using our sample project from Figure 4.1, we get the following steps:

(1) Find a PERT critical path

In order to find PERT critical path, we need to calculate expected durations for all task: $E[dur_A] = 3$, $E[dur_B] = 5.33 \approx 5$, $E[dur_C] = 3.33 \approx 3$, $E[dur_D] = 3.67 \approx 4$, $E[dur_E] = 8$. The critical path is Start→B→E→End as illustrated in Figure 4-4.

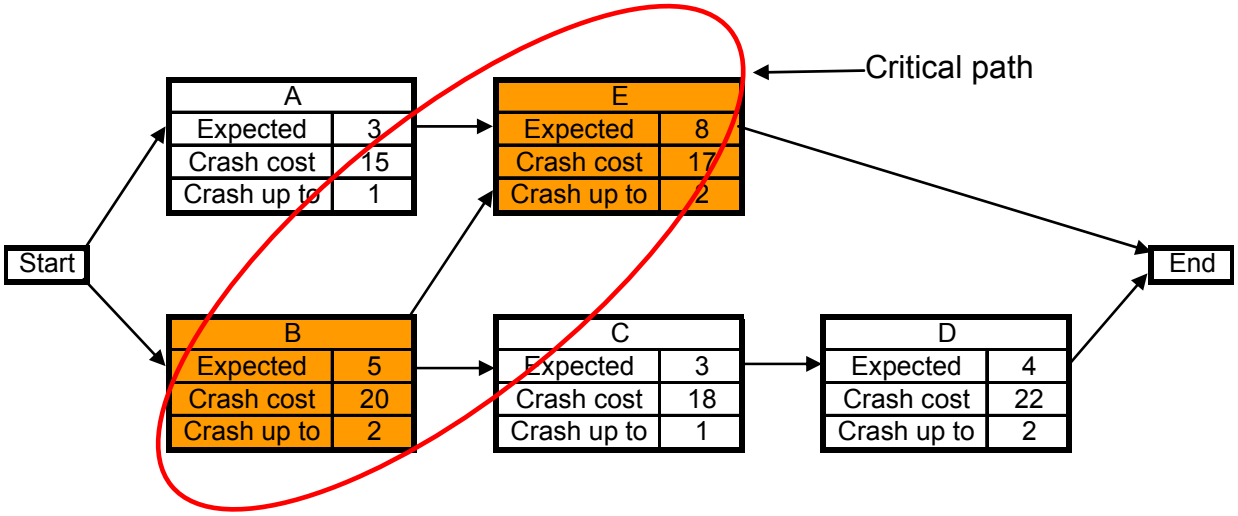


Figure 4-4: Simple project -- PERT critical path

(2) Perform DP on the critical path

The DP policy for path Start→B→E→End assuming Target date of 12 is presented in Table 4-6 below.

Table 4-6: DP policy for PERT critical path

Task	ti	Crash by
B	0	2
E	1	0
E	2	0
E	3	1
E	4+	2

The crossover points are as follows: $p_{B1} = -2$, $p_{B2} = -1$, $p_{E1} = 2$, $p_{E2} = 3$.

(3) Formulate and solve MIP

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 100\theta$$

$$\begin{aligned}
s.t.\ \ & t_C - t_B + z_B \geq 5^{(*)} && z_A \leq 1 \\
& t_E - t_B + z_B \geq 5^{(*)} && z_B \leq 2 \\
& t_E - t_A + z_A \geq 3 && z_C \leq 1 \\
& t_D - t_C + z_C \geq 3^{(*)} && z_D \leq 2 \\
& t_{END} - t_E + z_E \geq 8 && z_E \leq 2 \\
& t_{END} - t_D + z_D \geq 4^{(*)} \\
& t_{END} - \theta \leq 12
\end{aligned}$$

(*) Rounded to the nearest integer

$$\begin{aligned}
& t_B - (-2) \leq 1{,}000 z_{B1} && t_B - (-2+1) z_{B1} \geq 0 \\
& t_B - (-1) \leq 1{,}000 z_{B2} && t_B - (-1+1) z_{B2} \geq 0 \\
& t_E - 2 \leq 1{,}000 z_{E1} && t_E - (2+1) z_{E1} \geq 0 \\
& t_E - 3 \leq 1{,}000 z_{E2} && t_E - (3+1) z_{E2} \geq 0 \\
& z_B = z_{B1} + z_{B2} \\
& z_E = z_{E1} + z_{E2} \\
& z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary
\end{aligned}$$

Constraint $t_E - 2 \leq 1{,}000 z_{E1}$ forces $z_{E1} = 1$ when $t_E \geq 3$ ($3 - 2 \leq 1{,}000 z_{E1} \Rightarrow 1 \leq 1{,}000 z_{E1}$) and constraint $t_E - (2+1) z_{E1} \geq 0$ forces $z_{E1} = 0$ when $t_E < 3$ (i.e., if $t_E = 2$, we have $2 - 3 z_{E1} \geq 0$). Similarly, constraint $t_E - 3 \leq 1{,}000 z_{E2}$ forces z_{E2}=1 when $t_E \geq 4$ ($4 - 3 \leq 1{,}000 z_{E2} \Rightarrow 1 \leq 1{,}000 z_{E2}$) and constraint $t_E - (3+1) z_{E2} \geq 0$ forces $z_{E2} = 0$ when $t_E < 4$ (i.e., if $t_E = 3$, we get $3 - 4 z_{E1} \geq 0$).

When we solve the MIP, we get the following solution:

t_A	t_B	t_C	t_D	t_E	t_{END}	θ	z_{B1}	z_{B2}	z_{E1}	z_{E2}	z_A	z_B	z_C	z_D	z_E
0	0	3	6	3	12	0	1	1	1	0	0	2	0	0	1

(4) Consider solution for the activities starting immediately

We would crash B by 2 days, not crash A, and postpone all other decisions until future stages.

(5) Repeat at each stage of the project

We would repeat this procedure, recalculating the critical path whenever we need to make a new decision, substituting known durations for the expected, and setting crash limit constraints to equalities for those activities we already made decisions about.

4.4.5 Protect the Critical Path

In the LPDP algorithm, the PERT critical path can change often during the course of the project. This may not be desirable from a managerial point of view; project managers often prefer to focus on one path that will remain critical throughout the execution of the project. This was the motivation for designing the Protect the Critical Path (PCP) algorithm. At the beginning of a project, the PCP finds a PERT critical path like the LPDP; however unlike the LPDP, it tries to prevent that path from becoming non-critical. In finding solutions, the PCP also combines dynamic programming with linear programming but the implementation and the formulation differ from those developed for the LPDP. In addition, the PCP uses a concept of buffers motivated by the work of Goldratt (1997). Goldratt's idea was to build in buffers into the project to protect the critical chain (critical path in our case since we have no resource constraints) from any delays. We use buffers in a similar context to absorb any delays caused by those non-critical activities that link to the critical path.

The PCP consists of the following steps, some of which are the same as in the LPDP: (1) find a PERT critical path – this is done by calculating the longest path using expected durations for all activities, (2) perform DP on the PERT critical path using formulation from Chapter 3, (3) formulate and a solve mixed integer program that combines DP with LP to find critical path buffers, using optimistic durations for critical activities and pessimistic durations for non-critical

tasks effectively forcing the algorithm to try to prevent the critical path from changing at all cost, (4) formulate and solve an MIP (combining DP with LP) to find crashing values (using optimistic and pessimistic durations accordingly), (5) from the solution consider only those activities that are starting immediately and postpone all other decisions, (5) repeat the procedure at each stage of the project (when we can start another activity) substituting known durations for the optimistic/pessimistic durations.

Variables:

t_i = starting time of activity i

z_i = number of time periods to crash activity i

θ = number of time periods the project duration exceeds Target date (project lateness)

z_{ij} = binary variables to look up DP policy

y_i = buffer (delay on the critical path) for activity i

Constants:

c_i = cost to crash activity i by one time period

d_i = upper limit on the number of time periods we are allowed to crash activity i

$$dur_i = \begin{cases} \text{optimistic duration of activity } i \text{ if } i \text{ is critical} \\ \text{pessimistic duration of activity } i \text{ if } i \text{ is non-critical} \end{cases}$$

Π_i = set of immediate predecessors of activity i

$Penalty$ = cost per day for exceeding target date

$Target$ = target date for project completion

p_{ij} = crossover point j for activity i (for DP policy lookup)

Formulation:

Model 1: find the minimum buffers (delay on the critical path) – find y_i

$$\min \sum_i y_i$$

$$\begin{aligned}
s.t.\quad & t_i + y_i - t_k - y_k + z_k \geq dur_k \quad where\ k \in \Pi_i \quad \forall\, k, \forall i \\
& t_i - t_k + z_k = dur_k \qquad \forall\, k, i \in \text{critical path} \quad where\ k \in \Pi_i \\
& z_i \leq d_i \qquad \forall i \text{ not yet started} \\
& z_i = \sum_j z_{ij} \qquad \forall i \text{ not yet started} \\
& t_i + y_i - p_{ij} \leq \mathrm{M} z_{ij} \qquad \forall j, \forall i \\
& t_i + y_i - (p_{ij} + 1) z_{ij} \geq 0 \qquad \forall j, \forall i \\
& t_{END} + y_{END} - \theta \leq Target \\
& y_i \geq 0 \quad \forall i \in \text{ critical path} \\
& y_i = 0 \quad \forall i \notin \text{ critical path} \\
& z_i, t_i, z_{ij}, \theta \geq 0
\end{aligned}$$

Constraints $t_i - t_k + z_k = dur_k$ set t_i variables for all $i \in$ critical path to minimum starting times in a serial project corresponding to the critical path.

Model 2: minimize total project cost given the minimum buffers (fixing y_i variables)

$$\min \sum_i^N c_i z_i + Penalty \times \theta$$

$$\begin{aligned}
s.t.\quad & t_i + y_i - t_k - y_k + z_k \geq dur_k \quad where\ k \in \Pi_i \quad \forall\, k, \forall i \\
& t_i - t_k + z_k = dur_k \qquad where\ k \in \Pi_i \quad \forall\, k, \forall i \in \text{critical path} \\
& z_i \leq d_i \qquad \forall i \text{ not yet started} \\
& z_i = \sum_j z_{ij} \qquad \forall i \text{ not yet started} \\
& t_i + y_i - p_{ij} \leq 1{,}000 z_{ij} \qquad \forall j, \forall i \\
& t_i + y_i - (p_{ij} + 1) z_{ij} \geq 0 \qquad \forall j, \forall i \\
& t_{END} + y_{END} - \theta \leq Target \\
& y_i = \text{ values from model 1} \quad \forall i \in \text{ critical path} \\
& y_i = 0 \quad \forall i \notin \text{ critical path} \\
& z_i, t_i, z_{ij}, \theta \geq 0
\end{aligned}$$

The objective of the PCP algorithm is to, first, prevent the critical path from changing and second, to minimize total project cost. In steps (3) and (4) above we are using optimistic durations for critical tasks and pessimistic durations for non-critical tasks to achieve critical path protection. This can be illustrated more clearly by working through the example shown in Figure 4.1.

(1) Find PERT critical path

In order to find PERT critical path, we need to calculate expected durations for all task: $E[dur_A]=3$, $E[dur_B]=5.33\approx 5$, $E[dur_C]=3.33\approx 3$, $E[dur_D]=3.67\approx 4$, $E[dur_E]=8$. The critical path is Start→B→E→End as illustrated in Figure 4-5.

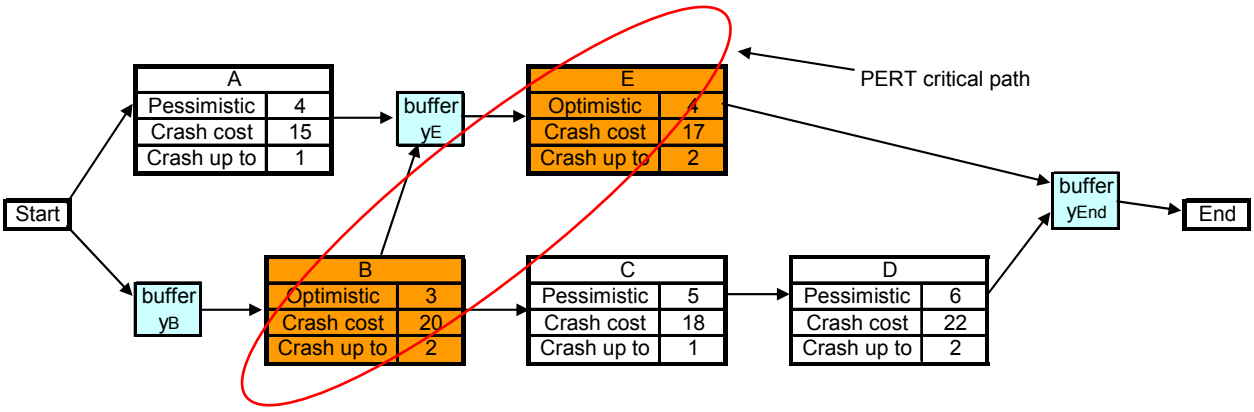


Figure 4-5: Project network diagram with buffers

(2) Perform DP on the critical path

Since, at the beginning of the project, the critical path is the same as in the LPDP algorithm, the DP policy for path Start→B→E→End is also the same (see Table 4.5). The crossover points are also $p_{B1}=$ -2, $p_{B2}=$ -1, $p_{E1}=2$, $p_{E2}=3$.

(3) Formulate and solve MIP to find buffers

$$\min\ y_A + y_B + y_C + y_D + y_E + y_{END}$$

$$
\begin{aligned}
s.t.\ \ & t_C + y_C - t_B - y_B + z_B \geq 3^{(*)} && z_A \leq 1 \\
& t_E + y_E - t_B - y_B + z_B \geq 3^{(*)} && z_B \leq 2 \\
& t_E + y_E - t_A - y_A + z_A \geq 4^{(**)} && z_C \leq 1 \\
& t_D + y_D - t_C - y_C + z_C \geq 5^{(**)} && z_D \leq 2 \\
& t_{END} + y_{END} - t_E - y_E + z_E \geq 4^{(*)} && z_E \leq 2 \\
& t_{END} + y_{END} - t_D - y_D + z_D \geq 6^{(**)} \\
& t_E - t_B + z_B = 3^{(*)} \\
& t_{END} - t_E + z_E = 4^{(*)} \\
& t_{END} + y_{END} - \theta \leq 12
\end{aligned}
$$

(*) Using optimistic duration

(**) Using pessimistic duration

$$
\begin{aligned}
& t_B + y_B - (-2) \leq 1{,}000 z_{B1} && t_B + y_B - (-2+1) z_{B1} \geq 0 \\
& t_B + y_B - (-1) \leq 1{,}000 z_{B2} && t_B + y_B - (-1+1) z_{B2} \geq 0 \\
& t_E + y_E - 2 \leq 1{,}000 z_{E1} && t_E + y_E - (2+1) z_{E1} \geq 0 \\
& t_E + y_E - 3 \leq 1{,}000 z_{E2} && t_E + y_E - (3+1) z_{E2} \geq 0 \\
& z_B = z_{B1} + z_{B2} \\
& z_E = z_{E1} + z_{E2} \\
& z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary \\
& y_B, y_E, y_{END} \geq 0 \\
& y_A, y_C, y_D = 0
\end{aligned}
$$

The solution is $y_A = 0, y_B = 0, y_C = 0, y_D = 0, y_E = 2, y_{END} = 8$

(4) Formulate and solve MIP to find crashing values

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 100\theta$$

$$
\begin{aligned}
s.t.\ \ & t_C + y_C - t_B - y_B + z_B \geq 3^{(*)} && z_A \leq 1 \\
& t_E + y_E - t_B - y_B + z_B \geq 3^{(*)} && z_B \leq 2 \\
& t_E + y_E - t_A - y_A + z_A \geq 4^{(**)} && z_C \leq 1 \\
& t_D + y_D - t_C - y_C + z_C \geq 5^{(**)} && z_D \leq 2 \\
& t_{END} + y_{END} - t_E - y_E + z_E \geq 4^{(*)} && z_E \leq 2 \\
& t_{END} + y_{END} - t_D - y_D + z_D \geq 6^{(**)} \\
& t_E - t_B + z_B = 3^{(*)} \\
& t_{END} - t_E + z_E = 4^{(*)} \\
& t_{END} + y_{END} - \theta \leq 12
\end{aligned}
$$

(*) Using optimistic duration

(**) Using pessimistic duration

$$
\begin{aligned}
& t_B + y_B - (-2) \leq 1{,}000 z_{B1} && t_B + y_B - (-2+1) z_{B1} \geq 0 \\
& t_B + y_B - (-1) \leq 1{,}000 z_{B2} && t_B + y_B - (-1+1) z_{B2} \geq 0 \\
& t_E + y_E - 2 \leq 1{,}000 z_{E1} && t_E + y_E - (2+1) z_{E1} \geq 0 \\
& t_E + y_E - 3 \leq 1{,}000 z_{E2} && t_E + y_E - (3+1) z_{E2} \geq 0 \\
& z_B = z_{B1} + z_{B2} \\
& z_E = z_{E1} + z_{E2} \\
& z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary \\
& y_B, y_E, y_{END} = \text{values from model 1} \\
& y_A, y_C, y_D = 0
\end{aligned}
$$

t_A	t_B	t_C	t_D	t_E	t_{END}	y_E	y_{END}	θ	z_{B1}	z_{B2}	z_{E1}	z_{E2}	z_A	z_B	z_C	z_D	z_E
0	0	1	6	1	4	2	8	0	1	1	1	0	1	2	0	0	1

(5) Consider solution for the activities starting immediately

We would crash B by 2 days, crash A by 1 day, and postpone all other decisions until future stages. Notice the difference between the LPDP and the PCP crashing policies. The PCP crashes activity A by 1 day (to shorten non-critical (from PERT's perspective) path giving the current critical path a greater chance of staying critical).

(6) Repeat at each stage of the project

We would repeat this procedure, recalculating the critical path whenever we need to make a new decision, substituting known durations for the optimistic/pessimistic (updating probability distributions if necessary), and setting crash limit constraints to equalities for those activities we already made decisions about.

4.5 COMPUTATIONAL TESTS

In order to evaluate the effectiveness of each of the presented algorithms, we constructed a set of test problems. To differentiate among multiple network topologies (such as parallel network, serial network, and everything in between) we used the concept of order strength (Mastor, 1970, Demeulemeester et al., 2003) as a measure of network topology. This is in contrast to Chapter 2 in which we used a serial-parallel (SP) index to differentiate among various topologies. Recall that, order strength (OS) can be defined as the total number of precedence relationships in the network (including transitive but excluding dummy relationships) divided by the theoretical maximum number of precedence relationships or $n(n-1)/2$ where n equals the number of activities in the project excluding dummy nodes (denoting a beginning and an end of the project). On the other hand, the serial-parallel index measures the length of the longest path and does not indicate how many precedence relationships exist in a project. Based on a preliminary analysis and due to the fact that performance of some of the methods discussed herein depends on the existence of a dominant path in the project, we decided to use the order strength instead of the SP index.

Figure 4.6 shows network topologies with a varying OS (the dashed arcs indicate implied (transitive) precedence relationships). Notice that a completely parallel network has an order strength of 0 whereas a completely serial network has an OS of 1.

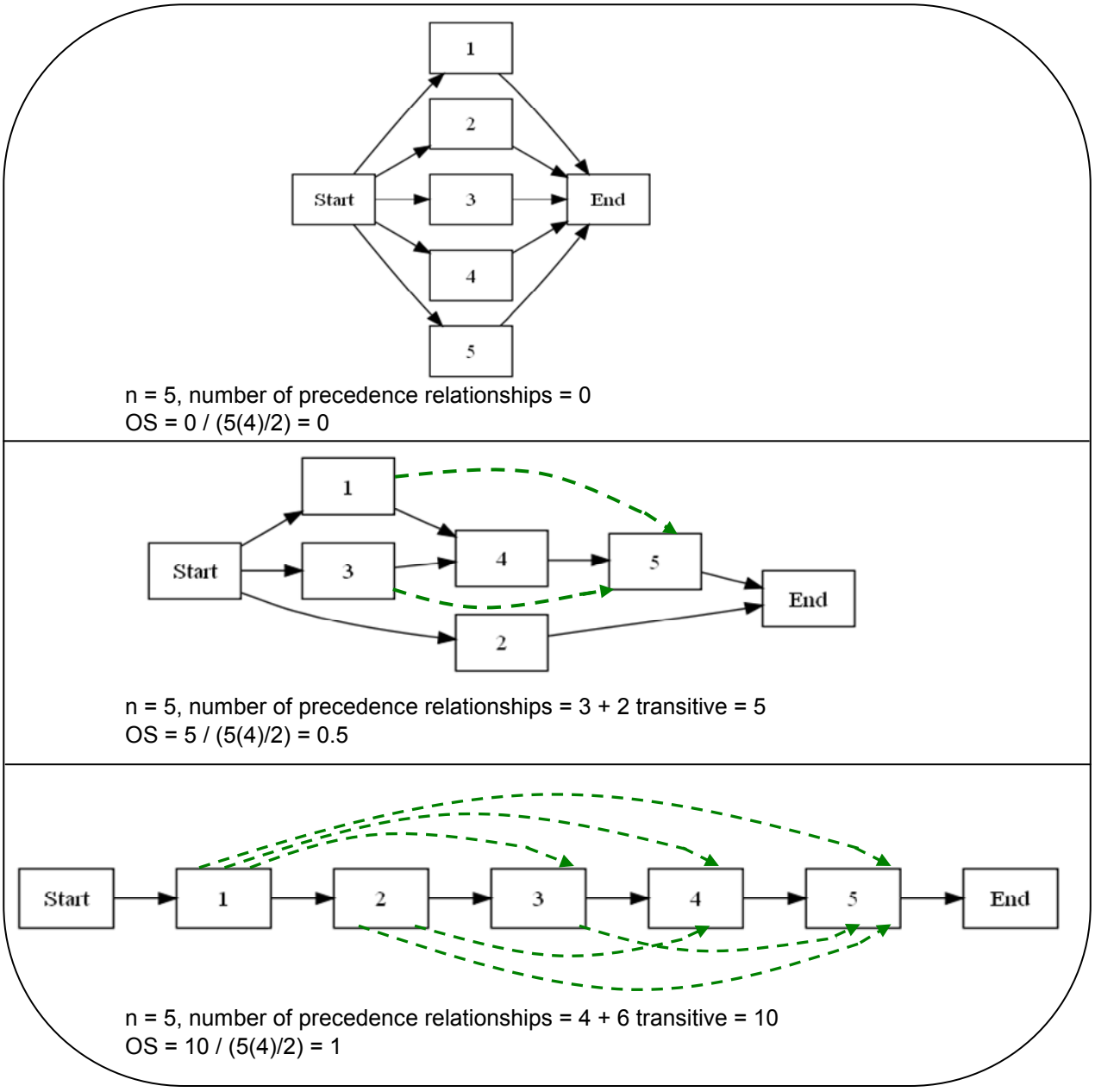


Figure 4-6: Project networks with various order strengths

Following Demeulemeester et al. (2003), we represent project networks using an upper triangular node incidence matrix. To differentiate between regular and transitive arcs, we denote each regular arc as a "1", and each transitive arc as "-1". Matrix representations of the three projects in Figure 4.6 are as follows:

OS = 0

	2	3	4	5
1	0	0	0	0
2	--	0	0	0
3	--	--	0	0
4	--	--	--	0

OS = 0.5

	2	3	4	5
1	0	0	1	-1
2	--	0	0	0
3	--	--	1	-1
4	--	--	--	1

OS = 1

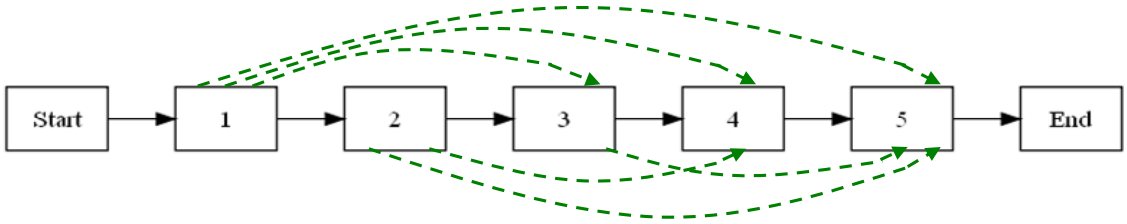
	2	3	4	5
1	1	-1	-1	-1
2	--	1	-1	-1
3	--	--	1	-1
4	--	--	--	1

4.5.1 Generating project topology

In order to generate project networks with varying OS we use the *RanGen* procedure described in Demeulemeester et al. (2003). Starting with a serial network (OS = 1), we randomly remove existing arcs from the network until we get the particular OS we are interested in.

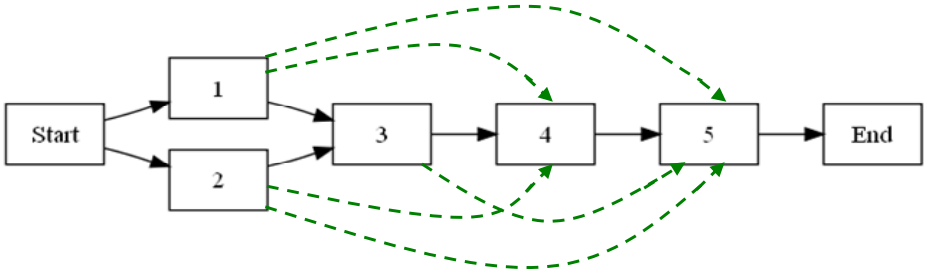
To illustrate the procedure we generate a 5-activity network with an OS = 0.4:

(1) Start with a completely connected, serial, network (OS = 1).

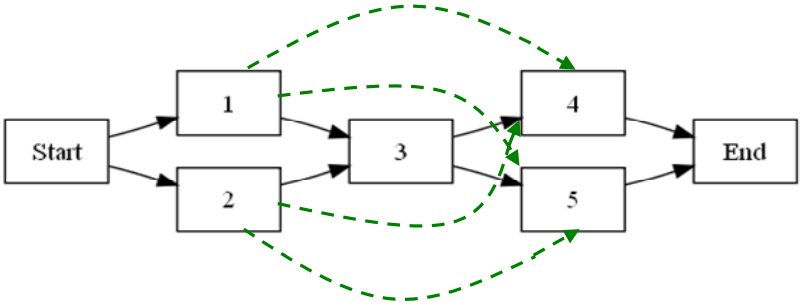


(2) Randomly select a non-transitive arc for removal – e.g., select (1,2). When we remove arc (1,2), node 1 loses connection to activities 3, 4, and 5. In order to "restore this connection", we convert one of the previously transitive arcs into a non-transitive (or regular) arc. The choice of an arc is made based on how many of the "lost" connections it "restores". There is always one unique transitive arc (unless no connections have been lost) that will re-establish

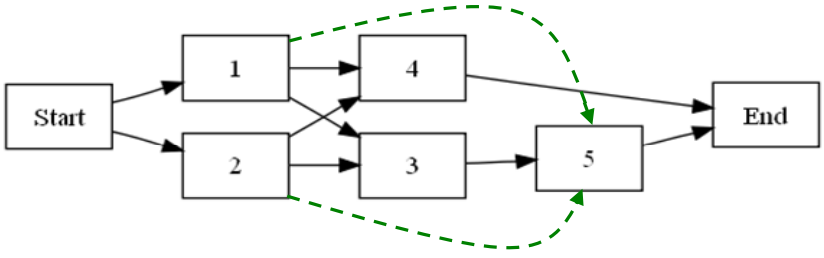
all of the necessary connections. In our case, adding arc (1,3) into the project restores connection of activity 1 to 3, 4, and 5. After this step, we get the following network with an OS = 9 / (5(4)/2) = 0.9



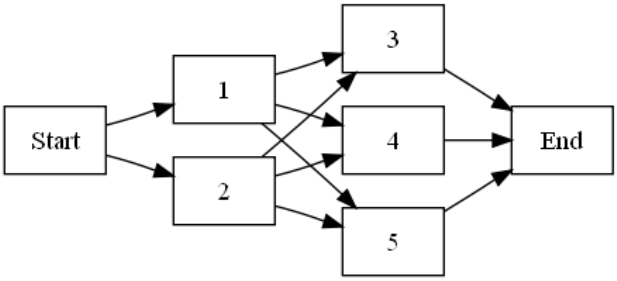
(3) Since current OS > desired OS, we continue with arc removal. Randomly select another non transitive arc – e.g., (4,5):



(4) OS = 0.8 → continue. Remove (3,4):

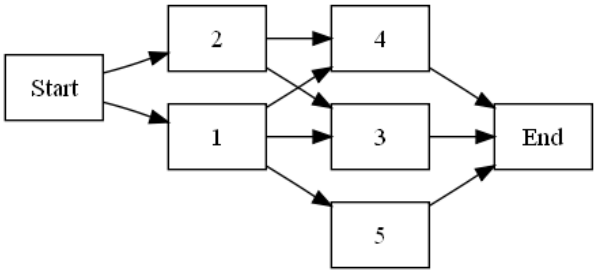


(5) OS = 0.7 → continue. Remove (3,5):

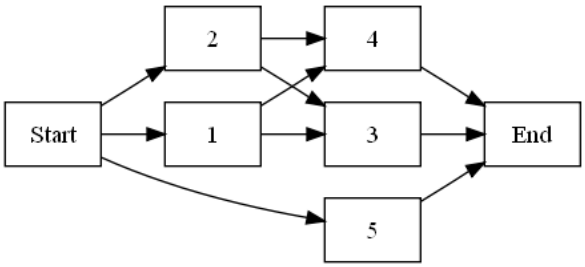


At this point we have no more transitive arcs.

(6) OS = 0.6 → continue. Remove (2,5):



(7) OS = 0.5 → Continue. Remove (1,5):



At this point OS = 0.4, which is the desired OS therefore we terminate the procedure.

We can also represent the above transformations using matrix notation:

OS = 1

	2	3	4	5
1	**1**	-1	-1	-1
2	--	1	-1	-1
3	--	--	1	-1
4	--	--	--	1

OS = 0.9

	2	3	4	5
1	*0*	*1*	-1	-1
2	--	1	-1	-1
3	--	--	1	-1
4	--	--	--	**1**

OS = 0.8

	2	3	4	5
1	0	1	-1	-1
2	--	1	-1	-1
3	--	--	**1**	*1*
4	--	--	--	*0*

OS = 0.7

	2	3	4	5
1	0	1	*1*	-1
2	--	1	*1*	-1
3	--	--	*0*	**1**
4	--	--	--	0

OS = 0.6

	2	3	4	5
1	0	1	1	*1*
2	--	1	1	***1***
3	--	--	0	*0*
4	--	--	--	0

OS = 0.5

	2	3	4	5
1	0	1	1	**1**
2	--	1	1	*0*
3	--	--	0	0
4	--	--	--	0

OS = 0.4

	2	3	4	5
1	0	1	1	*0*
2	--	1	1	0
3	--	--	0	0
4	--	--	--	0

In the matrices above, 1's represent real arcs (immediate precedence relationships) and -1's represent transitive arcs. Bold numbers denote arcs selected for removal, and numbers in italics indicate a previously transitive arc that is added to the project network.

Of course, there is a variety of topologies we can get with 5 nodes and an OS of 0.4. Figure 4-7 shows 4 different network structures, all with OS = 0.4.

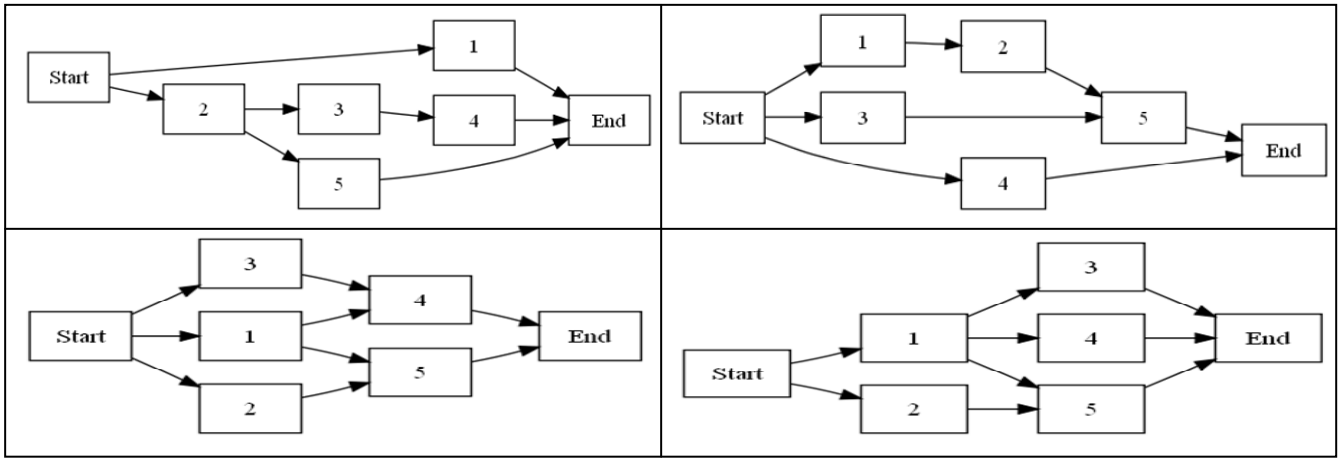


Figure 4-7: Project networks with OS = 0.4

4.5.2 Generating Target, Penalty, and Crash Costs

Once the project network topology is set, we can generate cost information. As in the serial project case, we follow the procedure based on Gutjahr et al. (1998). We set the target date equal to the expected duration of the PERT critical path and the penalty for exceeding the target to a constant $100 per day. Next, we simulate the project assuming no crashing is used N times where N is equal to the number of activities in the project multiplies by a constant (20 in our case). The procedure consists of the following steps:

(1) Determine the size of the project. We tested problems with 5, 10, 25, and 50 activities.

(2) Set the distribution span. Since in the serial case we did not see any patterns in performance of the algorithms with respect to different distribution spans, herein we only examine problems with an average distribution span of 16 days.

(3) For each activity generate an optimistic time, a most likely time, and a pessimistic time according to the span.

(4) Calculate the Target date completion of the project.

(5) Determine the maximum number of days by which each activity can be crashed (d_i) – generated from a discrete uniform distribution from [0, O_i-1] interval.

(6) Generate crash cost per day for each activity

   a. Estimate expected penalty cost E(Penalty) of the project with no compression using Monte Carlo simulation.

   b. Assign a fraction of E(Penalty) as the total cost for maximum crashing (*TCMC*) of all activities (total cost if we crash all activities to maximum). We refer to this fraction as a *cost structure* and tested values of 0.5 and 1.

   c. Calculate adjustment value for each activity (A_i)

$$A_i = \frac{d_i}{\max_k \{d_k\}} \quad \forall k \in \{1, ..., n\}$$

d. For each activity, generate a random number ~U[0, 1) and multiply it by A_i. We refer to this value as a cost distribution index (*CDI*).

e. Calculate a normalized cost distribution index (*NCDI*) for each activity:

$$NCDI_i = \frac{CDI_i}{\sum_{k=1}^{n} CDI_k}$$

f. Calculate cost of maximum crashing for each activity:

$$\text{cost of maximum crashing}_i = TCMC \times NCDI_i$$

g. Calculate crash cost per day for each activity:

$$c_i = \frac{\text{cost of maximum crashing}_i}{d_i}$$

4.5.3 Results

We define a problem type as a specific class of problems (for example, projects with a particular OS, size, and cost structure) and a problem instance as an individual occurrence of a problem type. The results are presented in the following manner. First we discuss problem types with cost structure of 1 with different project sizes. For cost structure of 0.5, we limit our discussion to problem types with 25 activities for convenience. Finally, we illustrate differences between dynamic (contingent decisions) vs. static (all decisions made at the beginning of the project) algorithms.

Since we do not have optimal solutions for general projects, in addition to the algorithms discussed in previous sections, for each problem instance we calculate the expected cost assuming we have full knowledge of activity durations that will be realized in the future. That is,

we make all simulated activity durations known to the algorithm a priori, thus reducing the problem to its deterministic counterpart, and solve using linear programming. We refer to the method as "Perfect Information."

Cost structure = 1, Size =5

Tables 4-7 and 4-8 show average expected costs and average running times by order strength and heuristic for five activity projects. The same information is also presented in Figures 4-8 and 4-9.

Table 4-7: Expected cost -- 5 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	160.48	160.47	202.26	160.37	184.17	117.42
0.1	148.65	148.82	194.90	152.45	168.39	108.04
0.2	168.61	168.97	206.56	179.59	236.42	112.28
0.3	192.45	192.65	228.58	192.55	264.01	148.49
0.4	173.25	173.76	221.33	174.80	248.52	125.15
0.5	190.19	190.03	235.82	190.29	207.66	141.30
0.6	161.39	161.38	202.59	160.55	175.80	113.85
0.7	197.89	197.81	238.93	197.96	244.49	149.75
0.8	193.78	193.08	265.17	195.65	243.15	131.71
0.9	124.08	124.09	155.42	123.84	131.67	77.82
1	169.33	169.32	200.98	166.10	166.10	117.70

*Averaged over 20 instances of each problem type

Table 4-8: Average running time (sec) -- 5 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.0005	0.0005	0.0016	0.0065	0.0160
0.1	0.0008	0.0007	0.0024	0.0110	0.0239
0.2	0.0007	0.0008	0.0021	0.0127	0.0290
0.3	0.0008	0.0008	0.0029	0.0147	0.0324
0.4	0.0010	0.0010	0.0028	0.0155	0.0336
0.5	0.0007	0.0007	0.0022	0.0118	0.0245
0.6	0.0010	0.0009	0.0023	0.0166	0.0316
0.7	0.0010	0.0011	0.0022	0.0192	0.0374
0.8	0.0011	0.0011	0.0018	0.0178	0.0333
0.9	0.0013	0.0014	0.0023	0.0284	0.0407
1	0.0016	0.0017	0.0042	0.0593	0.0667

*Averaged over 20 instances of each problem type

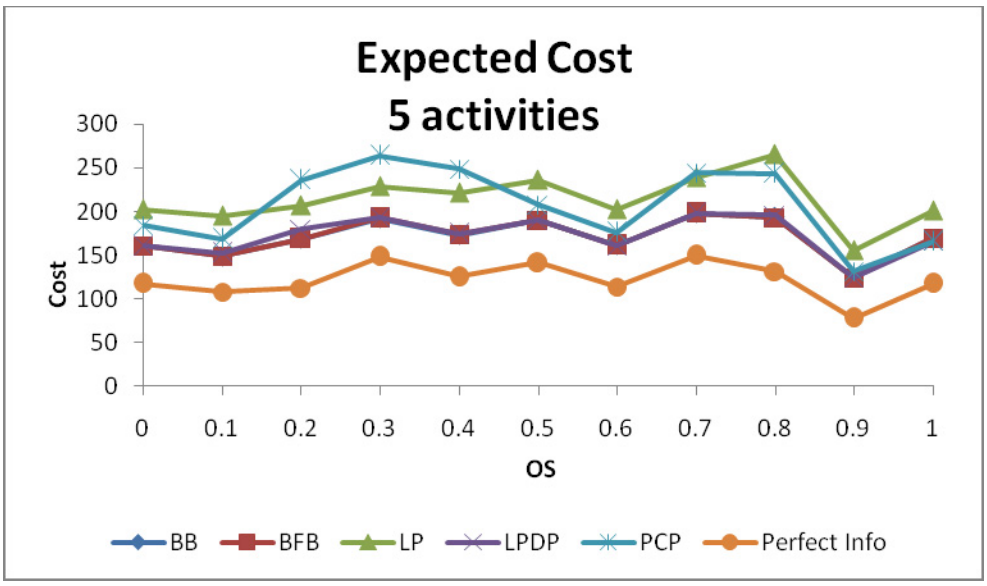


Figure 4-8: Expected cost -- 5 activity projects

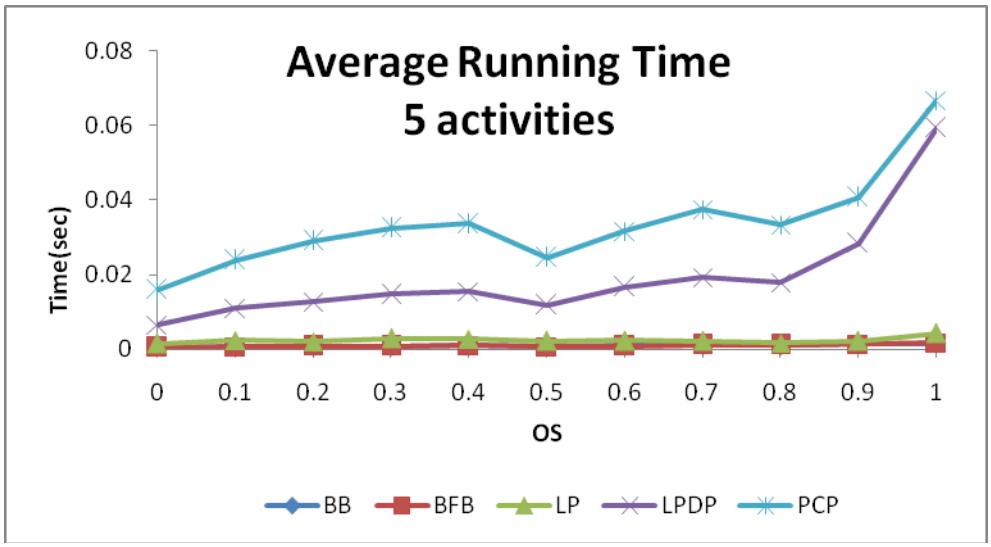


Figure 4-9: Average running time (sec) -- 5 activity projects

In addition we applied Bonferroni correction (Bonferroni, 1936) to a multiple comparison procedure in order to measure differences among expected costs (means) of the heuristics for each order strength. Figure 4-10 shows 95% Bonferroni intervals with homogenous groups circled.

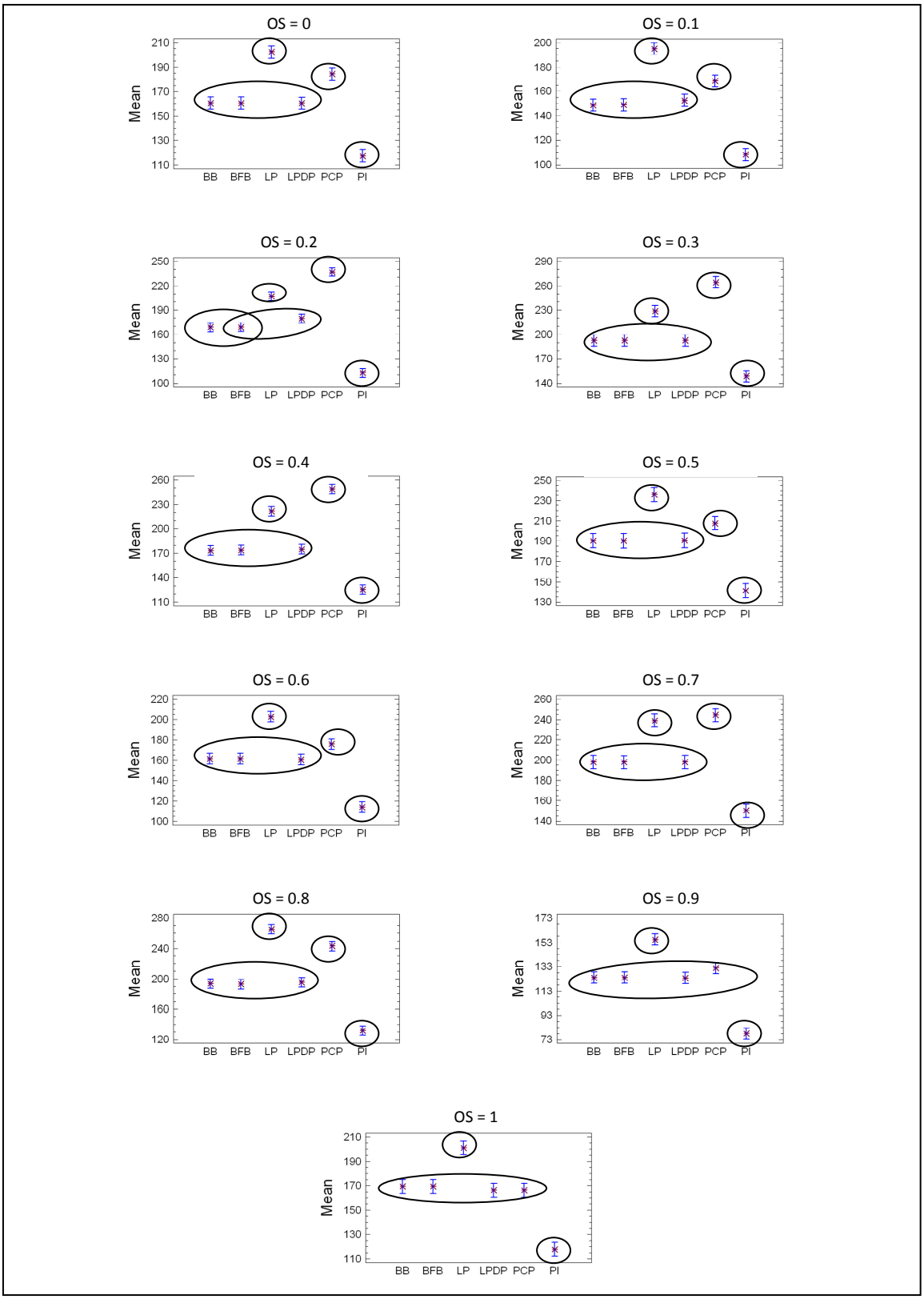


Figure 4-10: 5 activities -- 95% Bonferroni intervals

Cost structure = 1, Size =10

Tables 4-9 and 4-10 show average expected costs and average running times by order strength and heuristic for ten activity projects. The same information is also presented in Figures 4-11 and 4-12. Figure 4-13 shows 95% Bonferroni confidence intervals.

Table 4-9: Expected cost -- 10 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	166.64	166.56	266.15	187.39	175.62	123.67
0.1	120.82	120.94	185.23	123.63	139.62	87.22
0.2	214.13	214.12	281.17	216.26	278.92	153.91
0.3	163.62	163.49	259.24	184.42	213.34	94.49
0.4	268.55	268.73	373.35	290.36	369.69	191.13
0.5	287.47	286.49	342.64	288.66	297.38	233.52
0.6	192.37	192.30	282.00	193.37	290.84	107.73
0.7	252.46	252.53	331.68	255.09	412.96	180.34
0.8	165.04	164.29	269.53	166.73	299.21	88.11
0.9	149.74	149.94	197.37	147.48	157.59	86.49
1	274.92	274.71	320.56	268.92	268.92	143.15

*Averaged over 20 instances of each problem type

Table 4-10: Average running time (sec) -- 10 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.0030	0.0031	0.0012	0.0078	0.0182
0.1	0.0039	0.0040	0.0021	0.0155	0.0341
0.2	0.0065	0.0065	0.0033	0.0300	0.0557
0.3	0.0092	0.0090	0.0033	0.0323	0.0640
0.4	0.0070	0.0068	0.0035	0.0330	0.0619
0.5	0.0078	0.0080	0.0035	0.0604	0.0840
0.6	0.0096	0.0094	0.0040	0.0527	0.0923
0.7	0.0113	0.0111	0.0045	0.0932	0.1247
0.8	0.0129	0.0129	0.0046	0.1197	0.1440
0.9	0.0145	0.0144	0.0052	0.2232	0.1997
1	0.0179	0.0179	0.0064	0.5501	0.4274

*Averaged over 20 instances of each problem type

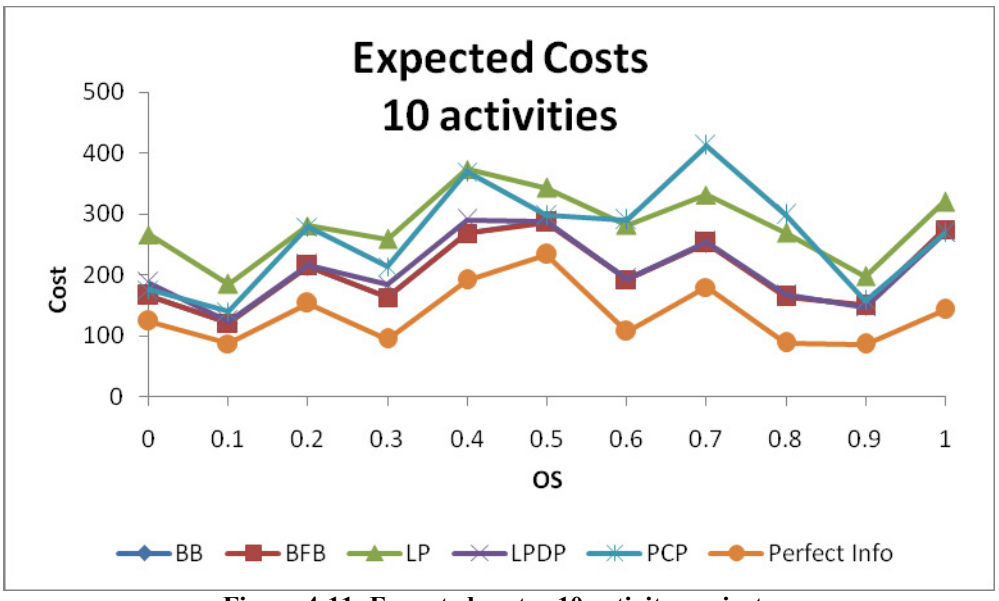


Figure 4-11: Expected cost -- 10 activity projects

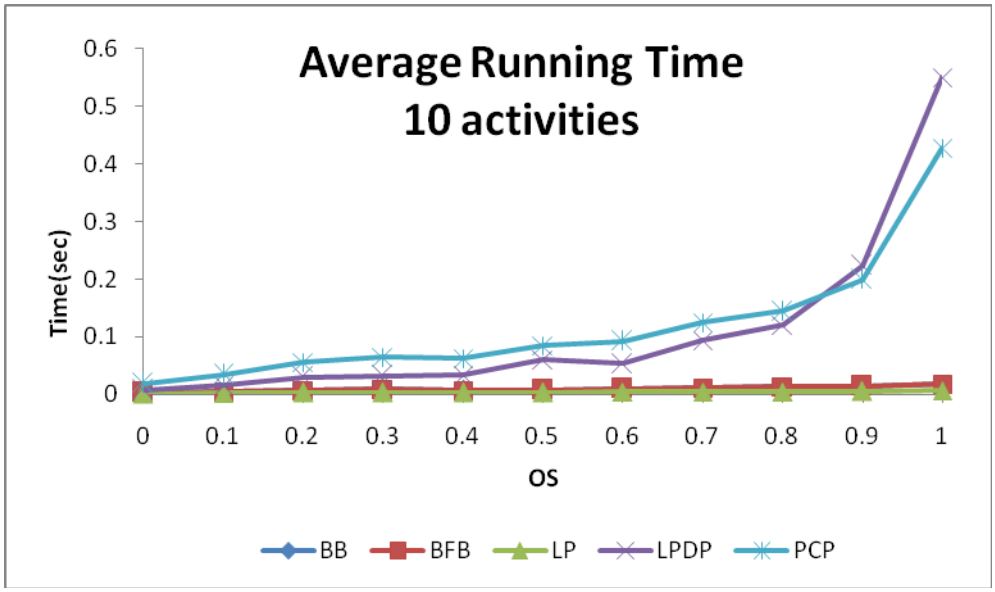


Figure 4-12: Average running time -- 10 activity projects

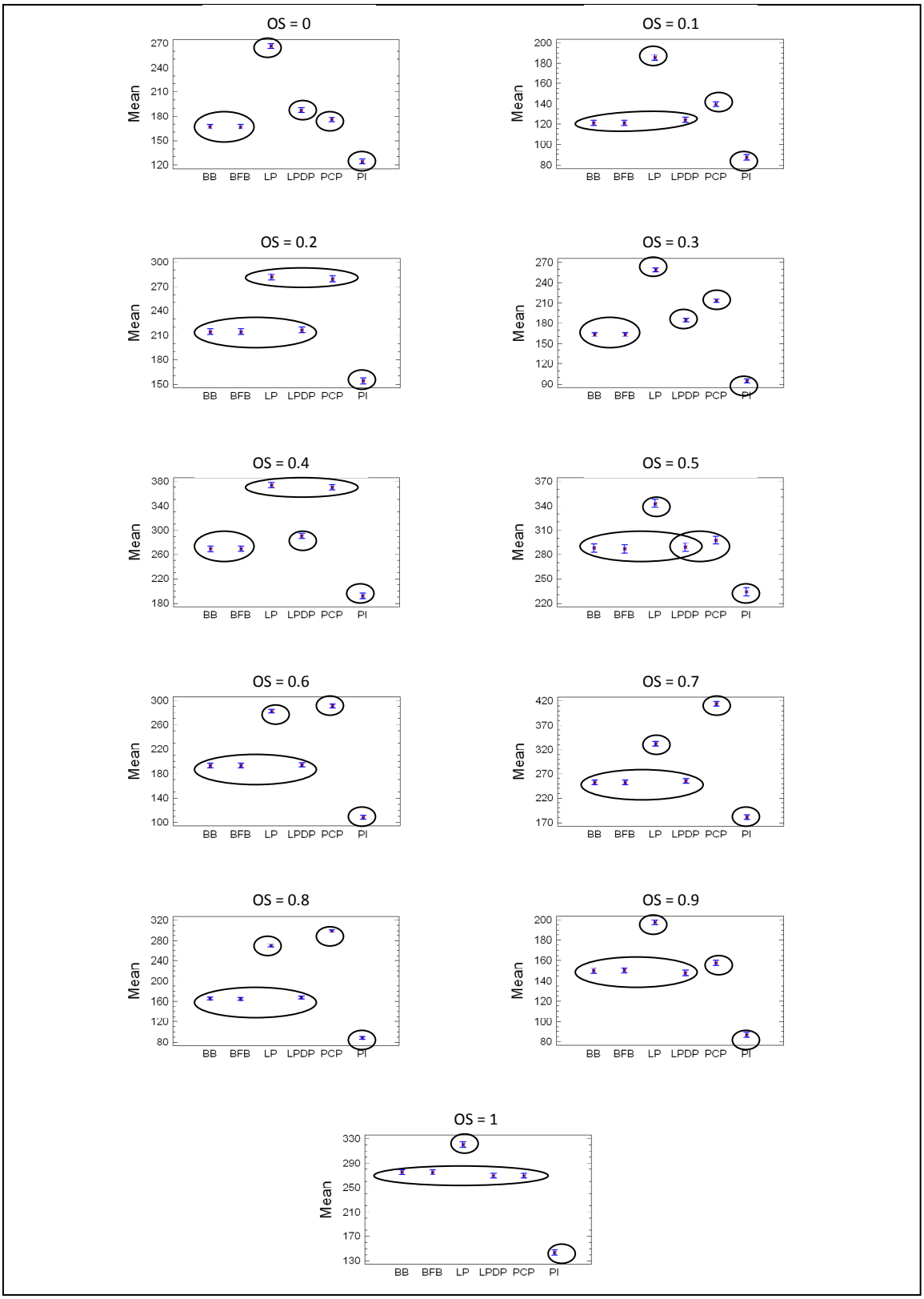


Figure 4-13: 10 activities -- 95% Bonferroni intervals

Cost structure = 1, Size = 25

Tables 4.11 and 4.12 show average expected costs and average running times by order strength and heuristic for projects with twenty five activities. The same information is also presented in Figures 4-14 and 4-15. Figure 4-16 shows 95% Bonferroni confidence intervals.

Table 4-11: Expected cost -- 25 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	194.58	194.58	323.41	242.01	205.20	170.99
0.1	161.03	161.16	338.94	195.36	198.49	113.07
0.2	188.40	188.50	354.20	203.61	242.19	132.61
0.3	248.78	248.61	409.81	271.35	341.09	169.08
0.4	252.06	252.21	392.36	266.86	357.84	174.02
0.5	260.11	259.55	429.44	289.23	439.95	167.89
0.6	341.06	341.01	510.90	365.55	561.97	216.04
0.7	275.53	275.12	435.12	288.35	514.26	165.76
0.8	174.81	174.09	362.87	207.70	480.14	83.79
0.9	221.41	220.36	422.47	241.31	499.59	95.93
1	472.00	472.02	614.57	468.35	468.35	266.03

*Averaged over 20 instances of each problem type

Table 4-12: Average running time -- 25 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.0213	0.0215	0.0024	0.0122	0.0310
0.1	0.0883	0.0889	0.0096	0.1033	0.2174
0.2	0.0976	0.0979	0.0131	0.2008	0.4071
0.3	0.1212	0.1214	0.0147	0.2335	0.4838
0.4	0.1580	0.1584	0.0210	0.3664	0.6444
0.5	0.2187	0.2210	0.0226	0.4039	0.8403
0.6	0.2244	0.2256	0.0250	1.0496	1.4710
0.7	0.2966	0.2990	0.0300	1.8514	2.0412
0.8	0.2792	0.2870	0.0295	2.2224	2.6154
0.9	0.3053	0.3110	0.0317	4.4180	4.5353
1	0.4882	0.4889	0.0562	23.9949	13.8437

*Averaged over 20 instances of each problem type

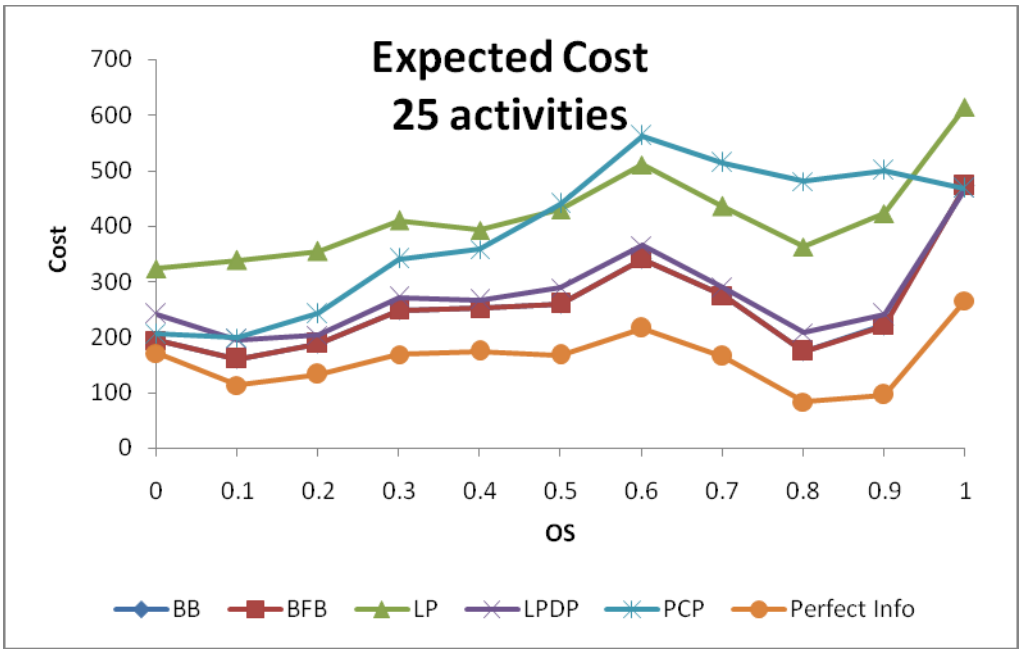


Figure 4-14: Expected cost -- 25 activity projects

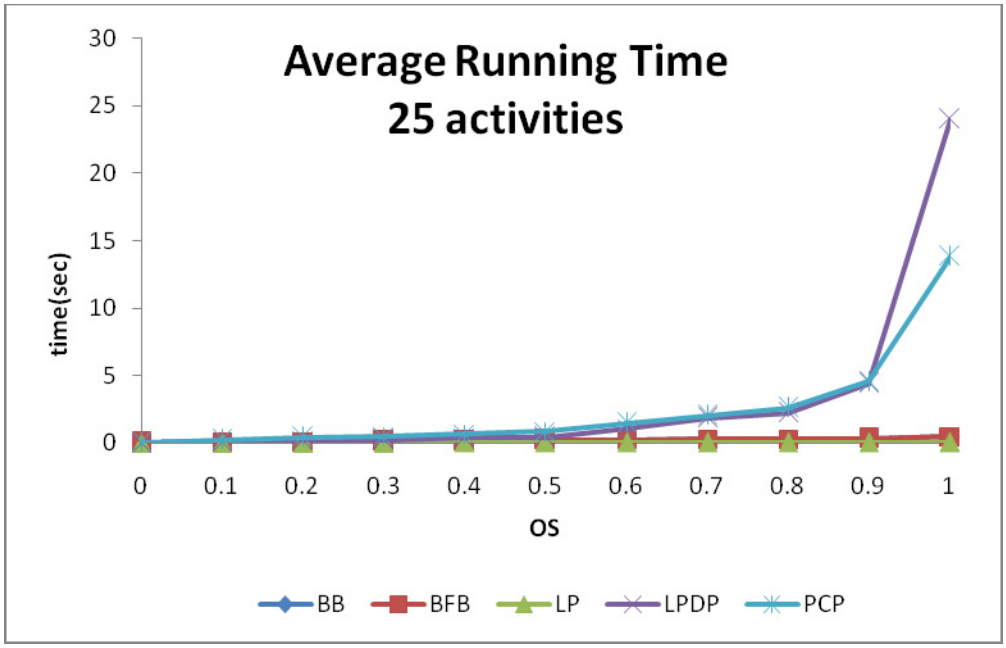


Figure 4-15: Average running time -- 25 activity projects

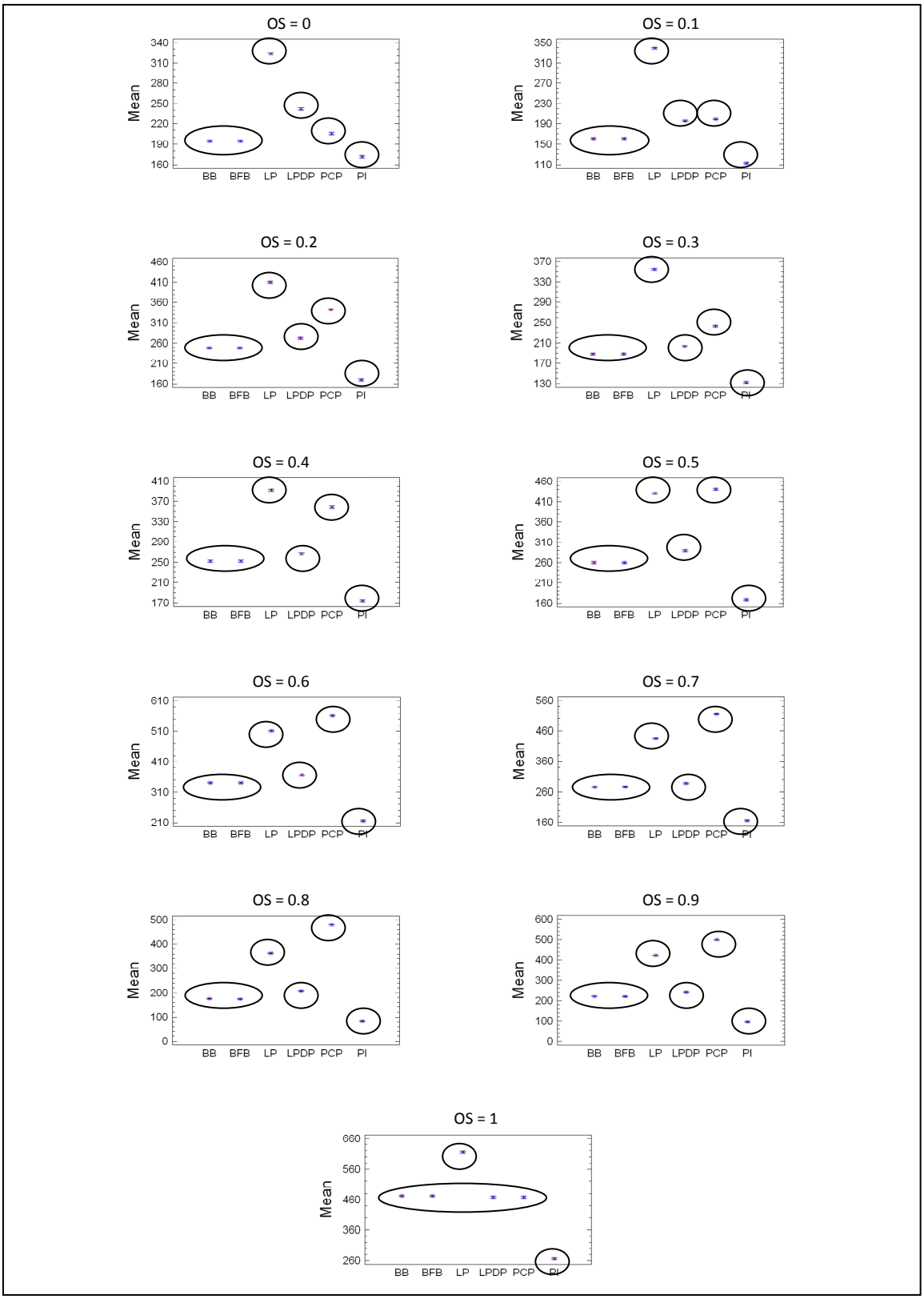


Figure 4-16: 25 activities -- 95% Bonferroni intervals

The cost performance of the algorithms is similar across all problem sizes. Biggest Bang and Bang for the Buck lead the way followed closely by LPDP. Basic LP and PCP are the worst performers. We can observe some variations in expected costs as the order strength changes. However, except for cases where the OS is equal to 0 or 1, we have no reason to believe that they reflect anything other than random causes. Since for OS = 1 (serial project), dynamic programming gives optimal solutions, our intuition was that DP based algorithms, that is, the LPDP and the PCP, would perform better as the order strength increases. This is, however, not the case with the PCP. Especially for larger projects, the PCP performs poorly even at order strength of 0.9. The PCP attempts to protect the critical path by crashing, possibly unnecessarily, non-critical activities. The more non-critical activities there are in a project, the poorer the PCP will perform. When we generated 25 activity project networks with OS = 0.9, we noticed that, on average, there were twelve non-critical activities (vs. one and three for 5-activity and 10-activity projects). Thus, for larger projects, the PCP performs poorly even at large order strengths.

Notice that, as the number of activities in a project increases, the Bonferroni confidence intervals shrink. This is because the number of simulation replication depends on the number of activities, that is the larger the project, the larger the sample size we obtain. Since confidence intervals are closely related to the measure of standard error, larger sample sizes result in smaller confidence intervals. Regardless of the project size, we notice that there are never any significant cost differences between the BFB and the BB. The LPDP belongs to the same homogenous group as the BFB and the BB in the majority of cases with 5 or 10 activities. Finally, for the OS of 1, there are no significant differences at the 95% confidence level between LPDP, PCP, BFB, and BB regardless of the project size.

Running times for all algorithms go up as the order strength and size increase; however, they do it at different rates. As the order strength increases, we have more decision stages in the project because the average length of a path increases as well. This is true for all the methods presented herein. However, running times for LPDP and PCP go up much more rapidly. This is due to the increased number of integer variables. As the order strength increases, the length of the PERT critical path increases as well, thus we need more DP lookup binary variables. The LPDP average running time of one simulation of one problem instance was about 24 seconds. Since we simulate each problem instance 20*n times (in this case, 20*25 = 500), and we have 20 instances of each problem type, the length of time to run the LPDP on a problem set with 25 activities and OS=1 was approximately 240,000 seconds or 66 hours and 40 minutes. We attempted to test the LPDP and the PCP on problem instances with 50 activities; however, the computational overhead was prohibitive. Therefore, we only tested the BB and the BFB on 50-activity instances.

Cost structure = 1, Size = 50

Tables 4-13 and 4-14 show average expected costs and average running times by order strength and heuristic (BB, BFB, and Perfect Information only) for projects with fifty activities. The same information is also presented in Figures 4-17 and 4-18.

Table 4-13: Expected cost -- 50 activity projects

OS	BB	BFB	Perfect Information
0	181.93	181.86	158.00
0.1	127.06	126.97	85.05
0.2	94.94	94.71	62.69
0.3	101.27	100.69	67.91
0.4	185.72	185.69	135.92
0.5	123.40	123.06	65.57
0.6	149.15	149.34	78.56
0.7	131.23	130.40	59.78
0.8	181.71	181.63	84.45
0.9	239.19	237.69	127.16
1	142.25	142.23	55.53

*Averaged over 20 instances of each problem type

Table 4-14: Average running time -- 50 activity projects

OS	BB	BFB
0	0.1889	0.1485
0.1	1.3107	1.1337
0.2	1.4366	1.2577
0.3	1.7456	1.5710
0.4	2.4611	2.1672
0.5	2.4788	2.1999
0.6	2.7310	2.4163
0.7	2.9478	2.6016
0.8	3.6488	3.1778
0.9	4.7484	4.1358
1	7.9854	6.3895

*Averaged over 20 instances of each problem type

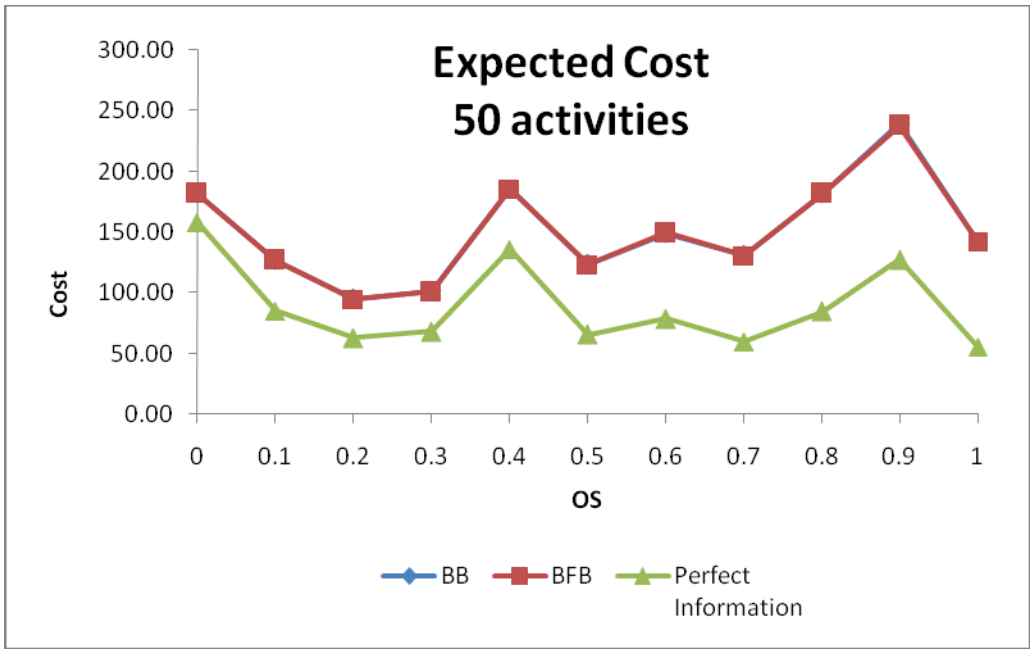


Figure 4-17: Expected cost -- 50 activity projects

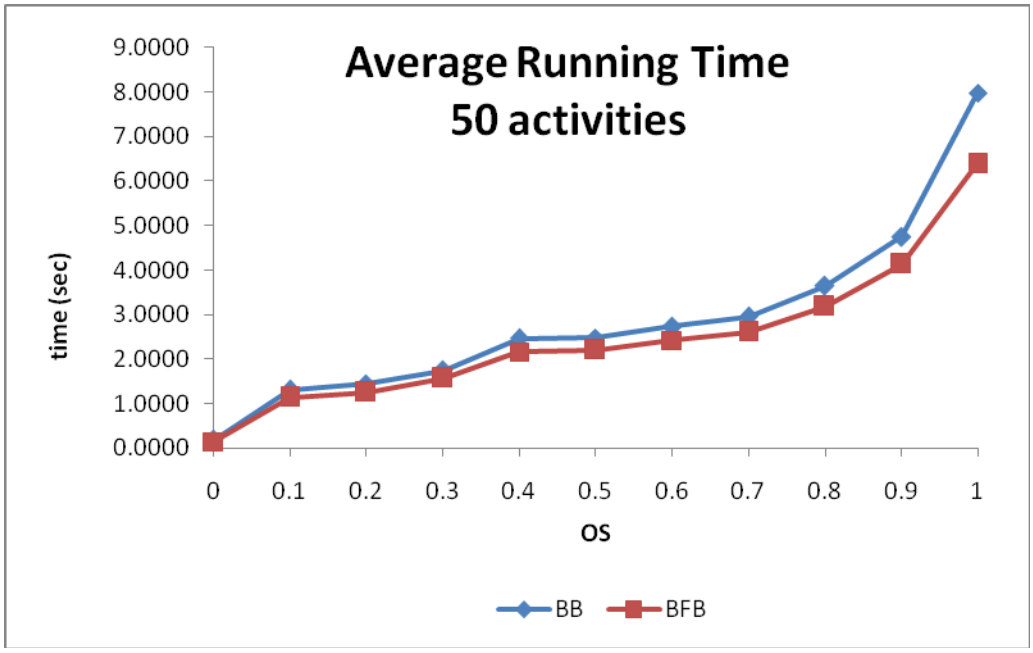


Figure 4-18: Average running time -- 50 activity projects

As before we can observe some variations in the cost values but there are no clear patterns. The average running times went up drastically; for order strength of 1, running time of

BB is slightly below 8 seconds compared to approximately 0.5 seconds for the same order strength with 25 activities. However, the running times of the BB and the BFB on problems with 50 activities are still lower than the running times of the LPDP and the PCP on 25-activity projects.

As we noticed earlier in this section, the PCP was not performing well due to the fact that the method, on average, compresses the project more by crashing non-critical activities in hopes of preserving the current critical path. Therefore, we conjecture that, if the crash costs are lower, the relative performance of the PCP should improve. We generated test problems with 25 activities and the cost structure of 0.5. Because now we are using half of the expected penalty with no crashing to generate crash costs, on average they should be 50% lower than those generated when cost structure was equal to 1. The results are presented below.

Cost structure = 0.5, Size = 25

Table 4-15 and Figure 4-19 show average expected costs by order strength and heuristic.

Table 4-15: Expected cost -- 25 activities with 0.5 cost structure

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	173.65	173.51	303.20	222.24	194.52	154.77
0.1	242.40	242.48	403.68	285.69	262.08	210.56
0.2	86.66	86.63	266.83	126.73	126.30	64.87
0.3	148.64	148.22	370.76	201.20	186.35	100.24
0.4	70.97	70.70	266.16	89.16	124.95	41.50
0.5	98.60	97.81	284.10	122.76	165.17	59.63
0.6	154.56	154.08	401.62	192.34	239.19	91.79
0.7	190.73	190.60	372.05	204.57	303.68	128.32
0.8	105.82	105.23	334.34	141.84	281.81	38.78
0.9	117.77	117.27	311.52	122.88	265.16	43.29
1	172.01	171.77	295.18	163.28	163.28	73.06

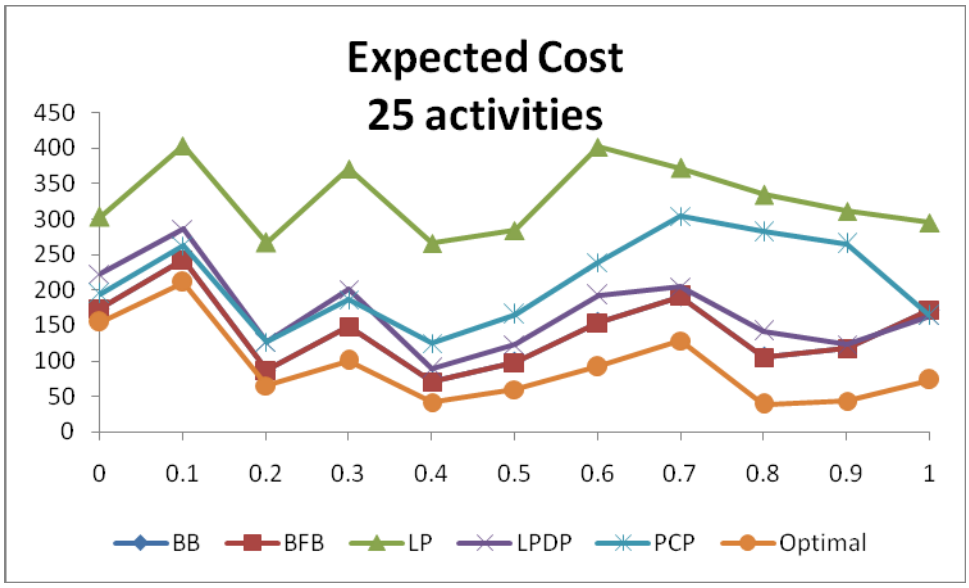


Figure 4-19: Expected cost -- 25 activities with 0.5 cost structure

As we can see, the relative performance of the PCP improved; however, the BB, the BFB, and the LPDP are still preferable if total expected cost is the criterion we use for evaluating these methods.

4.5.4 Impact of the order strength

As in the serial network case, we examine relative cost improvement of dynamic vs. static algorithms. Figures 4-20 and 4-21 present these results for the BB and the LPDP respectively. We tested problem instances with 25 activities and cost structure on 1. The results show expected costs averaged over 20 instances of each problem type.

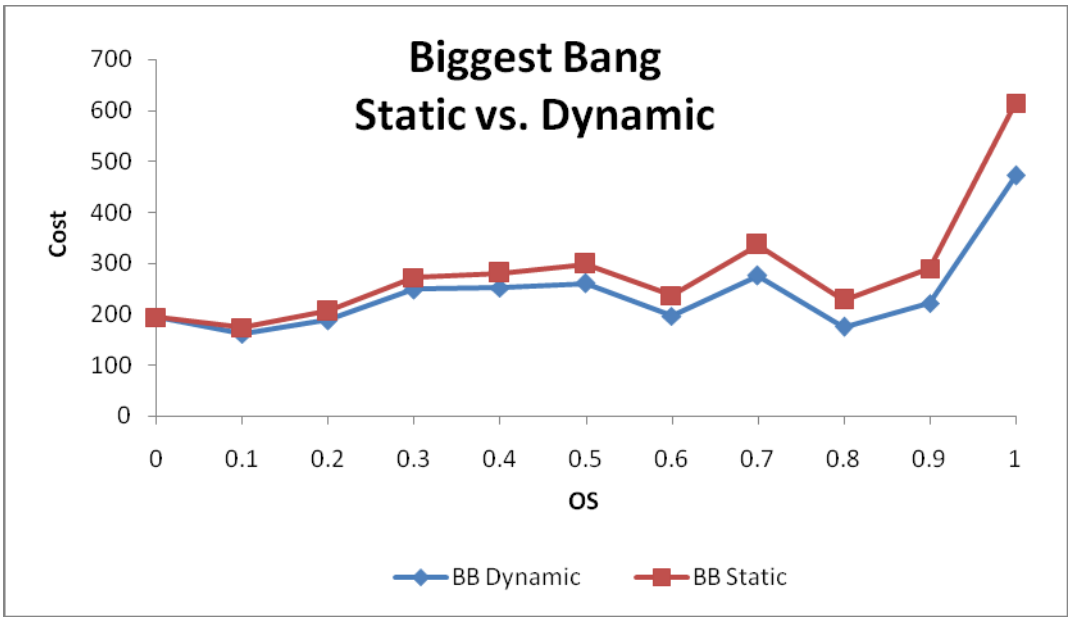


Figure 4-20: Biggest Bang -- static vs. dynamic

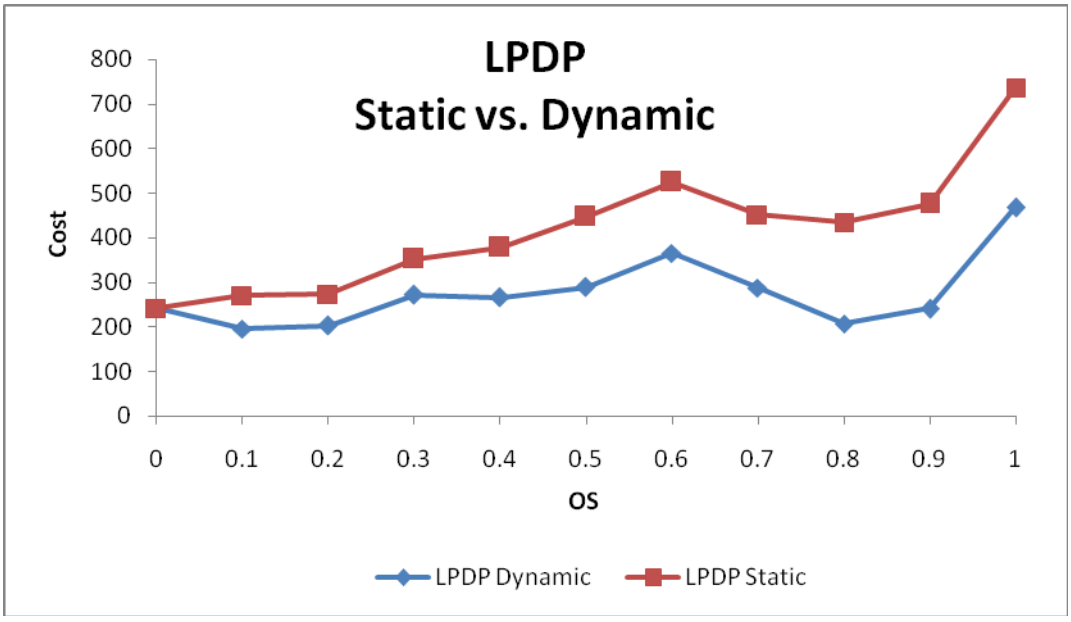


Figure 4-21: LPDP -- static vs. dynamic

As the order strength increases, so does the performance gap between static and dynamic versions of the algorithm. Even though we presented only results for the BB and the LPDP, this pattern is visible in all other algorithms. When OS = 0, that is, we have a completely parallel

project, the performance of a dynamic algorithm is the same as the performance of a static algorithm because we have only one decision stage (beginning of the project). As the order strength increases, we have more opportunities in the dynamic version of an algorithm to revise our decisions, thus, the larger the number of decision stages, the larger the performance gap between static and dynamic algorithms. For the order strength of 1, the relative improvement of a dynamic algorithm over a static one is around 23% for the BB and over 36% for the LPDP.

In addition, we notice that, as the OS increases, so does the gap between expected cost with perfect information and expected costs of our heuristics. This suggests, not surprisingly, that, as the projects become more serial (have more decision stages), the value of information increases. Table 4-16 shows the BB-Perfect Information gap by project size for OS of 0 and OS of 1 (for projects with cost structure = 1).

Table 4-16: BB/Perfect Info gaps

	Gap		
Size	OS=0	OS=1	Difference
5	36.66%	43.87%	7.20%
10	34.75%	92.05%	57.30%
25	13.80%	77.42%	63.63%
50	15.15%	156.17%	141.02%

Notice that the difference between the gap for OS of 1 and the gap for OS of 0 increases as the project size increases, which is consistent with our intuition – since for serial projects (OS=1), the number of decision stages is equal to the number of activities, thus the larger the serial project, the greater should be the value of information.

4.6 CHAPTER SUMMARY

In this chapter, we study an important problem faced by a large number of project managers: in presence of inevitable deadlines (and penalties for exceeding these deadlines) and given options to speed up some or all of project activities, which activities should we speed up and when should we make such speed-up decisions to minimize total project cost. We developed five algorithms to help managers make such decisions and analyzed their performance. The Biggest Bang and the Bang for the Buck use simulation and therefore handle uncertainty well; however, they are also myopic procedures in a sense that they do not explicitly take into consideration task interdependencies. The LP based methods were designed to deal with this shortcoming of the BB and the BFB. The Basic LP, Linear Programming with Dynamic Programming, and Protect the Critical Path, all take into consideration task interdependencies but they vary in their ability to handle uncertainty with Basic LP lagging behind. The LPDP and the PCP have an advantage of using dynamic programming on the PERT critical path thus, they should perform well for "almost" serial projects. The PCP method, while not having a stellar performance, can be attractive to those project managers who want to maximize the probability that current critical activities remain critical and therefore avoid constant reassignment of resources. In addition, the PCP should be adequate when crash costs are low compared to the penalty of exceeding the target date.

We also showed that using a dynamic algorithm, that is, making decisions contingent on a current state of the project, is preferable to making all decisions before any work on the project begins as it can lead to significant cost savings. Finally, we showed that simple rules of thumb, such as calculating expected savings from crashing an activity (the BB and the BFB) give good

results in a relatively short amount of time. They are also simple to understand and can be implemented without relying on any external packages, such as LP solvers.

5.0 CHAPTER FIVE: PROJECTS WITH INCENTIVES & OVERHEAD

5.1 INTRODUCTION

In Chapters 3 and 4 we presented a stochastic time-cost trade-off problem with a penalty, or disincentive, for exceeding the target date. However, in many real world applications there are other costs associated with executing a project, such as an overhead cost, incurred throughout the duration of the project, as well as incentives (can be regarded as negative costs) for completing the project early. As before, the only uncertainty is in task durations and the goal is to find the best crashing policy at the minimum expected cost. In this chapter we assume that crash costs are linear and we discuss both serial and general projects.

Incentive/disincentive (I/D) contracts are especially common in construction industry (Herten and Peeters, 1986) and are used primarily to achieve risk sharing or risk transfer. The main role of incentives is to motivate the contractor to embrace the client's project objectives (Bower et al. 2002). To illustrate this, consider a project with a target date of 100 days and a penalty of $100 per day for exceeding that date. The contractor's cost function is shown in Figure 5-1.

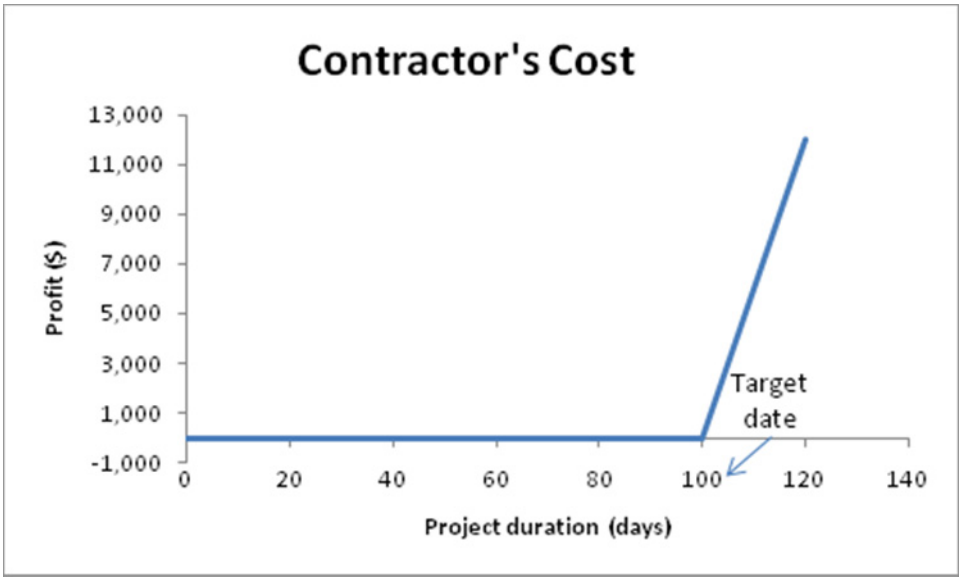


Figure 5-1: Contractor's cost (penalty only)

In this case, the contractor bears all the risk of the schedule overrun but receives no benefit if he completes the project early. Therefore, the optimal behavior from the contractor's perspective is to complete the project right on the Target date. However, it has been suggested by Herten and Peeters (1986) that incentives provided to contractors can reduce cost overruns.

There are various types of incentive contracts that can be used. According to Herten and Peeters (1986) the following types of contracts are available:

(1) Cost incentives – a target cost is agreed upon. Any overruns or underruns are shared according to some ratio. Both fixed price incentive (FPI) and cost plus incentive fee (CPIF) contracts fall under this category. The main difference between the two is that CPIF contract set the minimum fee the contractor will be paid regardless of the final project cost.

(2) Schedule incentives – a premium is paid to the contractor if the project is completed before the target date. If the project is completed after that date, the contractor incurs a penalty.

(3) Performance incentives – based either on improvement of target performances, such as quality, or on client evaluation. These types of incentives are more subjective.

In this research we look at schedule incentives but treat them as cost incentives. That is, we assume that the total indirect cost of the project is influenced only by the project duration. Any additional cost incurred (such as cost of speeding up tasks) is a responsibility of the contractor and, as such, is not included in target cost calculations. To illustrate this assume that a project has a target duration of 100 days. If indirect (overhead) costs are $100 per day, the target cost is then $10,000. If the project is completed before the target date, we realize cost savings proportional to the difference between the target and the actual completion date (same is true when actual duration exceeds *Target* with negative cost savings).

Next we need to agree on the cost sharing ratios. In this chapter we consider multiple levels of cost sharing, that is, the cost share ratio can change according to the project duration. Consider a project with one level of cost sharing. Assume the project target duration of 100 days and an overhead cost of $100/day. The contractor always pays a fixed percentage of the cost regardless of project duration. The contractor's cost function with the contractor : employer share ratio of 5 : 95 is presented in Figure 5-2. In this case, the contractor is responsible for 5% of the cost (or $5) on a daily basis, while the client absorbs the remaining 95%.

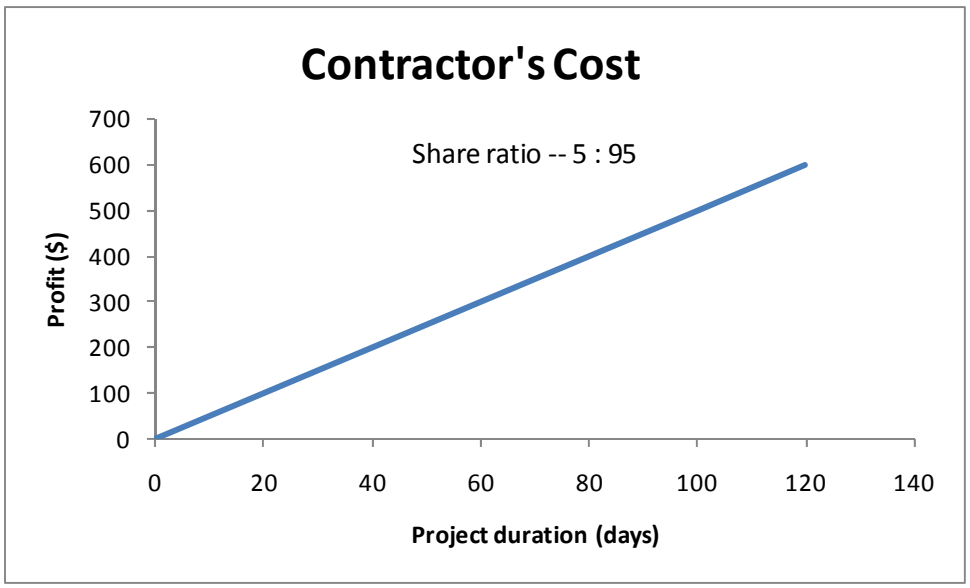


Figure 5-2: Constant share ratio

In the Penalty only case, we have two levels of cost sharing, where the contractor participates in the cost over-run sharing (but not cost under-run). Such a case is presented in Figure 5-3. The contractor pays a fixed portion of the cost (in this case, 5%) until the project duration reaches the Target date. Beyond the Target, the contractor is responsible for a higher percentage (100% in Figure 5-3) of the cost.

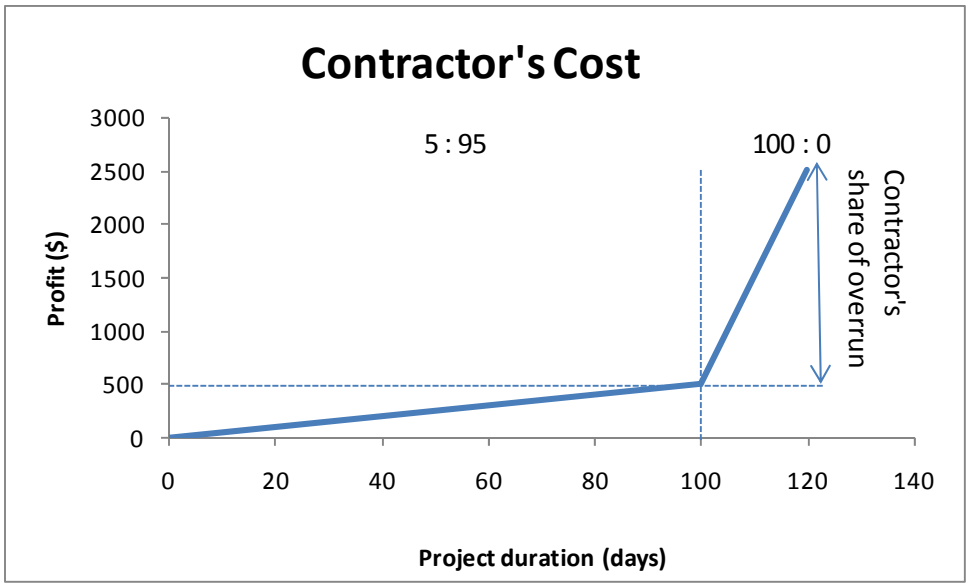


Figure 5-3: Two levels of cost sharing (penalty only)

Contracts with incentives and with no penalties can also have different levels of cost sharing. Consider the case where the contractor pays the fixed 5% of the overhead cost but he can also realize cost savings (incentives) if the project is completed before the due date. Figure 5-4 illustrates a contract where the contractor is eligible to receive 25% of cost under-run.

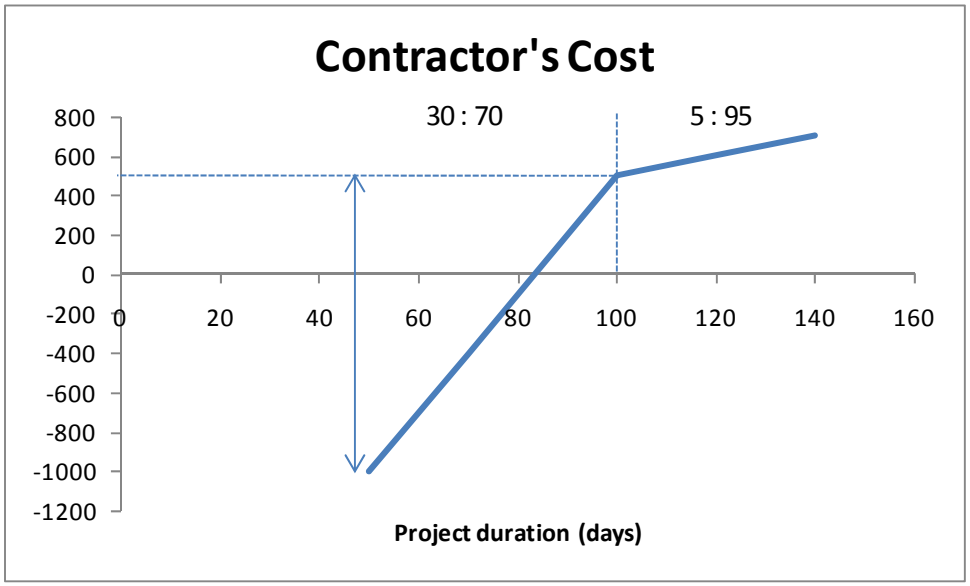


Figure 5-4: Two levels of cost sharing (incentives only)

Finally, there can also be contracts with both incentives and penalties as shown in Figure 5-5. The contractor incurs a constant 5% of the overhead until the project reaches the Target date – beyond that point he is responsible for the entire cost over-run. He can also realize incentives of 25% of cost under-run if the project completes in less than 70 days.

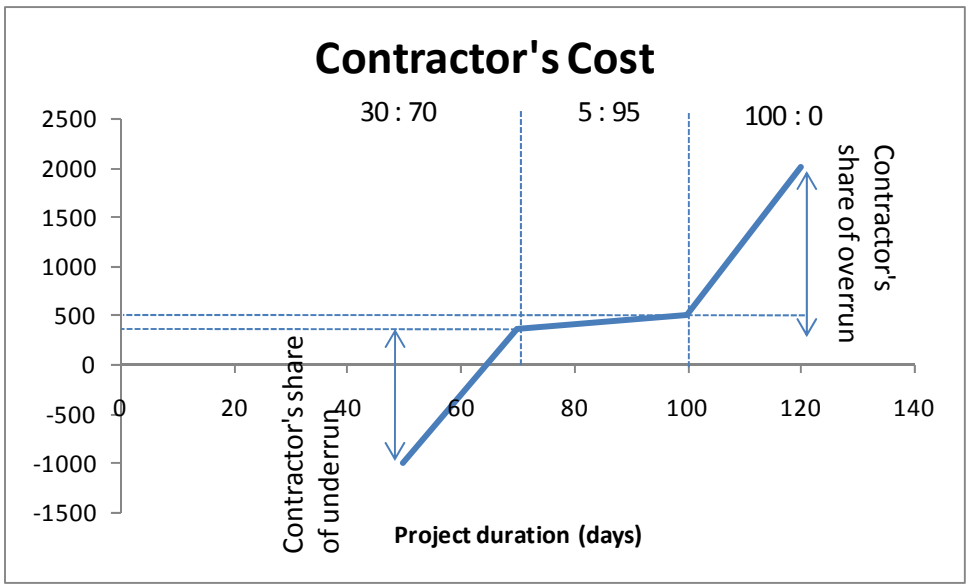


Figure 5-5: Three levels of cost sharing -- penalties and incentives

All the cost functions presented above are piecewise linear. Notice that the cost function is convex in case of penalty-only contracts and it is concave in the case of incentive-only contracts. However, if we consider both penalties and incentives, the cost function may not be well behaved as illustrated in Figure 5-5. Of course, the shape of the cost function depends on the cost share ratios chosen or agreed upon. Additionally, in this research we limit our discussion to at most three cost share levels (and refer to this as the basic incentive-penalty problem) although one could analyze a problem with more levels using the methods presented herein.

The rest of this chapter is organized as follows: in section 5.2 we discuss previous research on using incentives in projects, section 5.3 states the research problem; we present our algorithms in section 5.4 and the computational results in section 5.5. We discuss possible extensions to more general piecewise cost curves in section 5.6. Finally, we close the chapter with our conclusions in section 5.7.

5.2 PREVIOUS RESEARCH

A moderate amount of work has been done on analyzing incentive contracts in the construction industry; however, only a few studies exist that examine incentive/disincentive contracts in the project management framework. Ward and Chapman (1994) describe different types of contracts: (1) fixed price contract where the contractor bears all the risk associated with uncertain costs, (2) cost plus fixed fee contract where the client carries all the risk, and (3) incentive contracts with risks shared between the client and the contractor, which is the case considered in this Chapter. The authors talk about selecting the cost sharing rate and conclude that this choice should depend on the nature of the risk and whether it is controllable by the client or the contractor, or neither. According to the authors, incentive contracts are preferable because they provide a way to achieve risk sharing; however they are also difficult to negotiate, especially when project costs are uncertain. In addition, different payment arrangements should be adopted in different situations. The difficulty in setting the cost sharing rate was also discussed by Al-Harbi (1998). The author suggests the use of the utility theory and negotiations to derive the rate the contractor and the client can agree upon.

Jaraiedi et al. (1995) provide general guidelines for the use of incentives in highway construction projects. The main use of incentives is to reduce the overall project completion time. They advocate the use of incentives when the construction work has an adverse impact on local businesses, emergency services, safety of road users, or traffic. The authors also suggest that accelerated work schedules should be used to determine the amount of time that can be saved by using incentive contracts. Finally, incentive amounts should be set in a way that makes economic sense for the contractor to expedite the work. In another study examining highway construction projects, Shr and Chen (2003) looked at a deterministic time-cost trade-off problem

in presence of incentives and penalties. The research considered utilizing incentives to cover the cost of schedule compression. The authors did not look at crashing specific activities but assumed a constant cost (per time period) for project speed-up. They concluded that the possibility of receiving incentive payments should be taken into consideration as early as the bidding stage.

Bubshait (2003) studied perceptions of the clients and contractors about incentive/disincentive contracting. Some of the more notable findings were (1) most contractors are unaware of the reasons to include incentives in a contract whereas most clients indicated that the main reason for incentives is to achieve a rapid return on investment (due to shorter project durations), (2) most contractors add manpower to speed-up projects in order to receive bonus payments, (3) the amount of incentives and penalties should depend on the criticality of the project and the risk sharing, and (4) no consensus as to whether the amount of the incentive should be equal to, lower, or higher than the amount of the penalty.

Broome and Perry (2002) give several examples of cost-plus-incentive-fee contracts (or what they call target cost contracts) and examine how sharing ratios are set in the construction industry. Finally, Herten and Peeters (1986) give a background and history of incentive contracting. They also observe that, the main industries that utilize incentives include military, space programs, construction, power plants, commercial airlines, and software development.

5.3 STATEMENT OF THE RESEARCH PROBLEM

The research problem considered in this chapter is a stochastic time-cost trade-off problem with penalties and incentives for general projects with stochastic activity durations. The goal is to find

a crashing policy that will minimize total expected project cost. Recall the following project parameters described in Chapters 3 and 4: a set of activities A={1,…,*n*} (where *n* is the total number of activities), a set of precedence relationships Π (where Π_i is a set of immediate predecessors of activity *i*), discrete triangular probability distributions of task durations where O_i = optimistic duration of activity *i*, ML_i = most likely duration of activity *i*, and P_i = pessimistic duration of activity *i*, maximum number of time periods to crash each activity (d_i), and per day crash cost for each activity (c_i). Furthermore, we need to specify additional parameters. In this chapter we consider the basic problem only (with three cost share ratios), which we denote r_1, r_2, and r_3, (see Figure 5-6). More specifically, r_1 denotes cost under-run share ratio (or incentives), r_2 denotes and the portion of the overhead cost incurred by the contractor, and r_3 specifies cost over-run share ratio (or penalties). In general we set the ratios so that $r_3 \geq r_1 \geq r_2$, however different structures are possible. Additionally in practice problems with a larger number of cost share ratios are possible. We provide some examples of these and discuss extensions of the methods presented herein in Section 5.6.

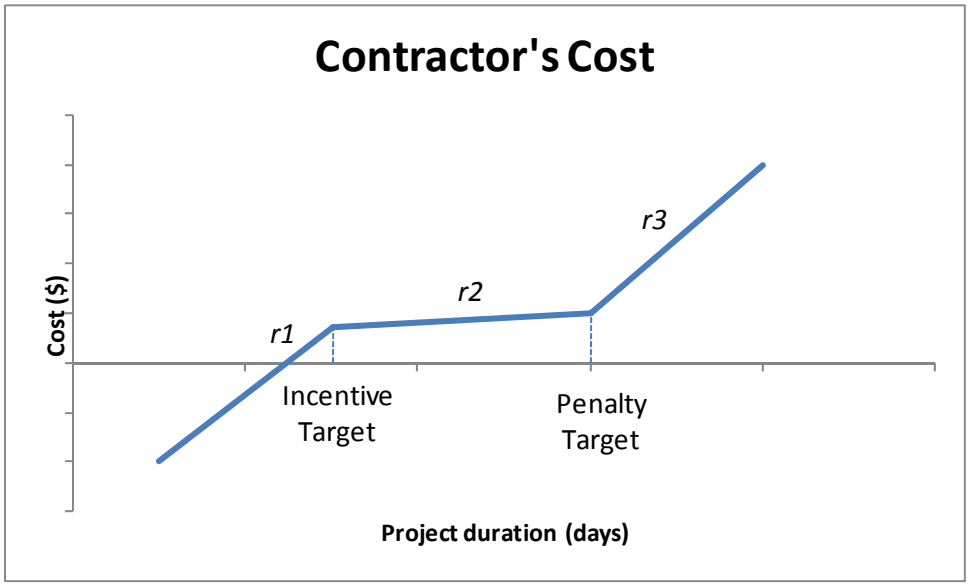


Figure 5-6: Contractor's additional cost

We also set a project penalty target date (*Penalty Target*), incentive target date (*Incentive Target*) and a per day indirect cost to the client (*Indirect*). As before, we assume that duration of a task is discrete and becomes known with certainty only after that task is fully completed (however, when the task is in progress we can update the probabilities of its duration according to the procedure presented in chapter 4 – see Figure 4-2). Crashing decisions are made dynamically throughout the execution of the project and are discrete (we can reduce the duration of an activity by a multiple of a full time period only). Figure 5-7 shows a project from Example 4.1 modified to include incentives.

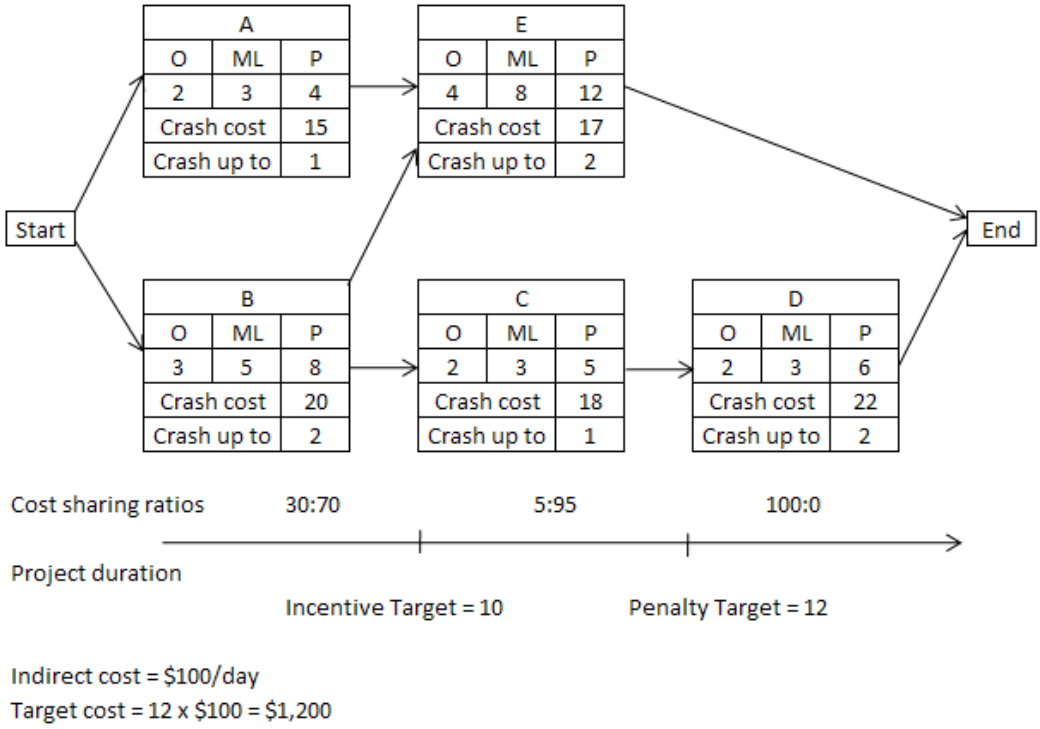


Figure 5-7: Illustrative Example 5.1

Notice that this setup results in an incentive of $25 per day if the project is completed in less than 10 days, overhead cost of $5/day incurred through the duration of the project, and an additional penalty of $95/day if the duration of the project exceeds 12 days. Therefore, the

contractor incurs an overhead cost (*Overhead*) of $5 per day throughout the execution of the project, a penalty cost (*Penalty*) of $95 for every day the duration of the project exceeds 12, and can receive incentive (*Incentive*) of $25 per day if the final duration if the project is less than 10. In other words, speeding up the project by 1 day results in $100 ($95 + $5) savings if the duration of the project is greater than 12, $5 savings if the duration of the project is between 10 and 12 days, and $30 ($25+$5) savings if the duration of the project is less than 10 days.

An important thing to note is that the problem with incentives and penalties reduces to the penalty only case (described in chapters 3 and 4) if r_1 and r_2 are both 0.

5.4 ALGORITHMS

In this section we extend the algorithms for general projects, described in Chapter 4, to the case with incentives and penalties. Because two of the algorithms – the LPDP and the PCP – depend on dynamic programming formulation, we also present the DP algorithm for a serial case with incentives to illustrate some of the challenges brought on by the non-convex cost function. For simplicity, we assume that the problem is of a basic incentive-penalty type, that is we have three cost share ratios as illustrated in Figure 5-6.

5.4.1 Dynamic Programming (serial case)

Consider a simple serial project presented in Figure 5-8. Assume the *Penalty Target* of 128 days, *Incentive Target* of 115 days, *Penalty* of $95, *Overhead* of $5, and *Incentive* of $25.

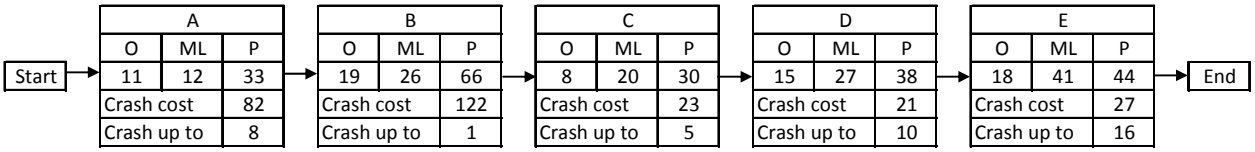


Figure 5-8: Illustrative Example 5.2 -- serial project

We have to modify slightly our standard dynamic programming formulation (with backward recursion). The difference is in the calculation of the cost for the last stage – in addition to accounting for the penalty cost, we also need to account for the overhead cost and the incentives. Therefore, the cost for the last stage becomes:

$$Cost^*(END, t_{END}) = t_{END} \times Overhead + \max\{(t_{END} - Penalty\ Target), 0\} \times Penalty - \max\{(Incentive\ Target - t_{END}), 0\} \times Incentive$$

The complete DP formulation is presented in Exhibit 5-1.

State representation:

$x = (i, t_i)$ where

i = {1,..,*END*} – current activity
n = number of non-dummy activities
t_i – starting time of activity *i*

z_i – number of time periods to crash activity *i*

c_i – cost of crashing activity *i*

Cost*(x) – the optimal cost-to-go for state *x*

p_k^i -- probability that normal duration of activity *i* will be *k* days

Penalty Target – target date for project completion – if duration exceeds *Penalty Target*, a penalty cost is incurred

Incentive Target – target date for qualifying for incentives

Penalty – cost per day for exceeding *Penalty Target* date

Incentive – incentive per day for completing the project before *Incentive Target*

For the terminal dummy activity (*END*):
For all t_{END} compute:

$$Cost^*(END, t_{END}) = t_{END} \times Overhead + \max\{(t_{END} - Penalty\ Target), 0\} \times Penalty - \max\{(IncentiveTarget - t_{END}), 0\} \times Incentive$$

For all activities i, where $i \le n$:
For all t_i do:

$$Cost^*(i, t_i) = \min_{z_i}\{\sum_k p_k^i (Cost^*(i+1, t_i + k - z_i)) + c_i \times z_i\}$$

Exhibit 5-1: Dynamic Programming formulation with overhead and incentives

Next we find a crashing policy for every possible state of the project at each stage. The final policy is presented in Table 5-1. Notice the lack of monotonicity in the dynamic programming policy. For example, we would crash activity E by 16 days if this activity starts between day 29 and day 79; however, if the start time of activity E is between day 80 and 88, we do not crash it at all. Then, as the start time increases, we start crashing E again.

Table 5-1: DP Policy (incentive, overhead, and penalty)

Activity	Start time	Crash by
E	29-79	16
E	80-88	0
E	89	1
E	90	2
E	91	3
E	92	4
E	93	5
E	94	6
E	95	7
E	96	8
E	97	9
E	98	10
E	99	11
E	100	12
E	101	13
E	102	14
E	103	15
E	104+	16

Activity	Start time	Crash by
D	24-63	10
D	64	5
D	65	6
D	66	7
D	67	8
D	68	9
D	69+	10
C	21-48	5
C	49-52	0
C	53	1
C	54	2
C	55	3
C	56	4
C	57+	5
B	3	0
A	0	0

This situation occurs because there is a part of the cost function where the benefit of shortening project duration is smaller than the cost of crashing an activity (between *Incentive Target* and *Penalty Target*). Therefore, the DP solution for the case with overhead cost and incentives may be non-monotonic. Figure 5-9 compares the DP solution to the penalty only case with the solution to a problem with incentives and overhead.

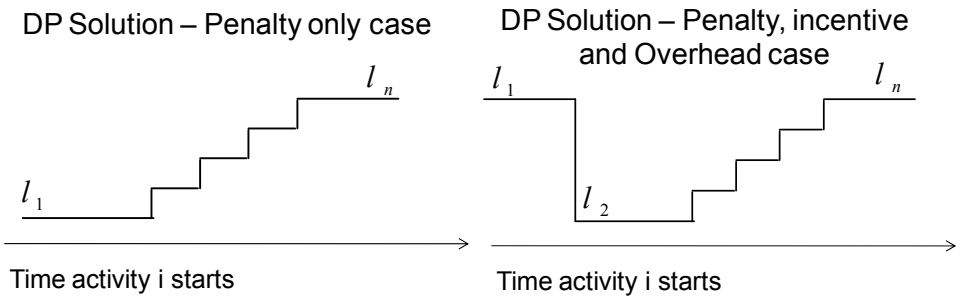


Figure 5-9: DP solutions – comparison

We conjecture the following:

(1) In the penalty only case, the dynamic programming solution is always monotonic:

$l_k \le l_{k+1} \quad \forall k$

(2) In the case with penalties, incentives and overhead:

a. $l_1 =$ max. crash for activity *i*

b. $l_2 \le l_1$

c. $l_k \le l_{k+1} \quad \forall k \ge 2$

5.4.2 Biggest Bang

As before, the Biggest Bang (BB) utilizes a concept of the Penalty Target Criticality Index (PTCI), which is a probability that crashing an activity will result in a lower penalty cost; however, since we also need to account for incentives, we introduce the Incentive Target Criticality Index (ITCI). The ITCI is a probability that crashing an activity will result in higher total incentive earned. Finally, to account for the *Overhead*, we calculate simple criticality index (CI), which is a probability that crashing an activity will result in lower overall overhead cost (or equivalently, it is a probability that an activity is critical). The BB index (BBI) for each activity can be calculated as:

$$BBI_i = PTCI_i \times Penalty + ITCI_i \times Incentive + CI_i \times Overhead - c_i$$

The decision on how much to speed up each task depends, as before, on the expected benefits (savings in penalty cost plus savings in overhead cost plus incentives earned) and on the cost of crashing this task as well as on possibilities of speeding up subsequent tasks in the project.

Since BBI is a greedy approach, we want to crash an activity that would provide the highest expected savings, or equivalently the activity with the highest BB index. Using the project from Figure 5-7 we get the following:

Table 5-2: BB indices – iteration 1

Activity	*CI*	*PTCI*	*ITCI*	*BBI*
A	0.03	0	0	-14.85
B	1	0.77	0.03	58.9
C	0.41	0.28	0.03	11.4
D	0.41	0.28	0.03	7.4
E	0.75	0.6	0.01	44

Because activity B has the highest BB index, we would reduce it by one day and recalculate BB indices. The next 5 iterations are presented in Table 5-3.

Table 5-3: BB indices – iterations 2-6

	Iteration 2				Iteration 3				Iteration 4				Iteration 5				Iteration 6			
Activity	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*
A	0.14	0.06	0.01	-8.35	0.33	0.05	0.09	-6.35	0.35	0.04	0.13	-6.2	0.27	0.02	0.21	-6.5	0.49	0.01	0.41	-1.35
B	0.97	0.56	0.02	38.6	0.81	0.32	0.16	18.5	0.89	0.23	0.21	11.6	0.93	0.13	0.43	7.75	0.82	0.07	0.51	3.5
C	0.48	0.22	0.02	5.8	0.4	0.09	0.14	-3.95	0.55	0.08	0.17	-3.4	0.77	0.08	0.41	3.7	0.54	0.04	0.39	-1.75
D	0.48	0.22	0.02	1.8	0.4	0.09	0.14	-7.95	0.55	0.08	0.17	-7.4	0.77	0.08	0.41	-0.3	0.54	0.04	0.39	-5.75
E	0.72	0.44	0.01	28.7	0.73	0.29	0.11	17	0.59	0.18	0.14	6.55	0.44	0.05	0.23	-4.3	0.7	0.04	0.47	2.05

At iteration 2, activity B again had the highest BB index so we reduce it by another day. Notice that task B has now reached its maximum compression. At iteration 3 we reduce E by one day since it has the highest BB index. At iteration 4 we again reduce E by 1 day (E has now

reached its maximum). At iteration 5 activity C, which was previously not an attractive choice to crash becomes a viable option. Finally, at iteration 6, all activities eligible for crashing have negative expected savings therefore we terminate the procedure. The final *static* policy before the project starts would be to crash B by 2 days, C by 1 day, and E by 2 days. However, since BB is a dynamic algorithm, we only need to crash those activities that are starting immediately, that is activity B. We will postpone decisions about tasks C, D, and E until later. The decision at the start of the project is therefore to crash B by 2 days and to not crash A. We will repeat this procedure once we are able to start additional activities (in this case only after B ends).

Notice the trade-off between the PTCIs and the ITCIs. As the project duration becomes shorter, the ITCIs increase whereas the PTCIs decrease. Therefore, at the beginning of the procedure, the PTCIs contribute more to the BBIs than the ITCIs (and the trend reverses later in the procedure).

5.4.3 Bang for the Buck

Recall that, the Bang for the Buck (BFB) is a slight modification of the BB algorithm. Instead of calculating expected savings, we calculate expected savings per dollar spent. The BFB also uses the concept of the Penalty Target Criticality Indices and the Incentive Target Criticality Indices. The BFB index (BFBI) for each activity can be calculated as:

$$BFBI_i = \frac{PTCI_i \times Penalty + ITCI_i \times Incentive + CI_i \times Overhead - c_i}{c_i}$$

The BFBI gives the expected cost savings per dollar spent, therefore, the larger the expected savings, the higher the BFBI and the larger the Crash Cost, the lower the BFBI. Therefore, we want to crash an activity with the highest BFBI.

Using the project in Figure 5-7 we get the following indices:

Table 5-4: BFB indices -- iteration 1

Activity	*CI*	*PTCI*	*ITCI*	*BBI*
A	0.03	0	0	-0.99
B	1	0.77	0.03	2.95
C	0.41	0.28	0.03	0.63
D	0.41	0.28	0.03	0.34
E	0.75	0.6	0.01	2.59

Since activity B has the highest BFB index, we would reduce it by one day. Because the criticality indices might have changed due to shortening activity B by one day, we need to recalculate the BFBIs. The next 5 iterations are presented in Table 5-5.

Table 5-5: BFB indices -- iterations 2-6

	Iteration 2				Iteration 3				Iteration 4				Iteration 5				Iteration 6			
Activity	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*	*CI*	*PTCI*	*ITCI*	*BBI*
A	0.18	0.06	0.02	-0.53	0.11	0.01	0.03	-0.85	0.39	0.09	0.13	-0.08	0.24	0.01	0.16	-0.59	0.35	0	0.26	-0.45
B	0.97	0.5	0.09	1.73	1	0.49	0.13	1.74	0.79	0.22	0.15	0.43	0.87	0.1	0.37	0.16	0.88	0.06	0.51	0.14
C	0.35	0.09	0.07	-0.33	0.69	0.27	0.12	0.78	0.48	0.06	0.12	-0.38	0.76	0.08	0.35	0.12	0.58	0	0.41	-0.27
D	0.35	0.09	0.07	-0.45	0.69	0.27	0.12	0.46	0.48	0.06	0.12	-0.5	0.76	0.08	0.35	-0.08	0.58	0	0.41	-0.4
E	0.77	0.47	0.04	1.91	0.53	0.27	0.05	0.74	0.64	0.22	0.15	0.64	0.39	0.03	0.19	-0.44	0.65	0.06	0.34	0.03

The final policy is therefore to crash B by 2 days and postpone decisions about C, D, and E until more information is available. Recall, that even though the BB and BFB gave the same starting policies, it is possible for the algorithms to suggest different policies.

5.4.4 Basic LP

The basic LP method again follows the standard LP approach for crashing activity-on-node networks, where expected durations of activities not completed are used instead of their deterministic values. The objective function is to minimize total cost, that is, total crash cost plus any penalty cost, plus overhead cost minus incentives, subject to precedence relationships

constraints (a task cannot start until all its predecessors are finished), maximum crash limit constraints, and a project finish time constraint. The formulation however, requires a substantial modification because of the introduction of the incentive term. Recall the LP formulation from Chapter 4:

Variables:

t_i = starting time of activity i

z_i = number of time periods to crash activity i

θ = number of time periods the project duration exceeds Target date (project lateness)

Constants:

c_i = cost to crash activity i by one time period

d_i = upper limit on the number of time periods we are allowed to crash activity i

$E[dur_i]$ = expected duration of activity i (for triangular distribution, $\frac{O_i + ML_i + P_i}{3}$)

Π_i = set of immediate predecessors of activity i

$Penalty$ = cost per day for exceeding target date

$Target$ = target date for project completion

Formulation:

$$\min \sum_i^N c_i z_i + Penalty \times \theta$$

$$s.t.\ \ t_i - t_k + z_k \geq E[dur_k] \quad where\ k \in \Pi_i \quad \forall k, \forall i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$t_{END} - \theta \leq Target$$

$$z_i, t_i, \theta \geq 0$$

First, we need to modify the objective function to reflect additional costs and incentives. We need to introduce two new variables:

θ_O = number of time periods the project duration exceeds the Penalty Target

θ_U = number of time periods the project duration is less than the Incentive Target

The objective function then becomes

$$\min \sum_{i}^{N} c_i z_i + Penalty \times \theta_O - Incentive \times \theta_u + Overhead \times t_{END}$$

The precedence relationship constraints remain the same but we need to bound θ_O and θ_U. The constraint for θ_O is straight forward. Since large values of θ_O increase the value of the objective function, constraint $t_{END} - \theta_O \le Penalty\ Target$ and the minimization function will ensure the bound on θ_O.We can write the constraint for θ_U as $t_{END} + \theta_u - s = Incentive\ Target$, where *s* is the slack associated with θ_U constraint. However, there are two problems with this constraint: (1) we do not have an upper bound on θ_u and (2) we do not upper bound on s. We know that

$$t_{END} \ge Incentive\ Target \Rightarrow \theta_u = 0,\ s \ge 0 \text{ and } t_{END} < Incentive\ Target \Rightarrow \theta_u > 0,\ s = 0$$

We need to introduce a new binary variable (let us call it b).

$\theta_u \le \mathrm{M}b$ → this constraint specifies that $\theta_u > 0$ if and only if b = 1, otherwise, $\theta_u = 0$

We now have to make sure that if $t_{END} - Incentive\ Target \ge 0$ (or $Incentive\ Target - t_{END} \le 0$), b = 0.

$Incentive\ Target - t_{END} + \mathrm{M} - (\mathrm{M}+1)b \ge 0$ so if

$Incentive\ Target - t_{END} \le 0 \Rightarrow b = 0$ and

$Incentive\ Target - T_{END} > 0 \Rightarrow b = 0\ or\ b = 1$

Therefore, constraints $Incentive\ Target - t_{END} + \mathrm{M} - (\mathrm{M}+1)b \ge 0$ and $\theta_u \le \mathrm{M}b$ ensure that $\theta_u = 0$ if $t_{END} - Incentive\ Target \ge 0$

There is still a problem with the upper bound for s. If we do not bound s and if $t_{END} <$ *Incentive Traget*, the model would maximize θ_u setting s so that constraint $t_{END} + \theta_u - s = Incentive\ Target$ is still satisfied.

So we have to ensure that:

- If $\theta_u > 0 \Rightarrow s = 0$
- If $\theta_u = 0 \Rightarrow s \geq 0$

Recall the b variable. The above is equivalent to

- If $b = 1 \Rightarrow s = 0$
- If $b = 0 \Rightarrow s \geq 0$

Now we can write a simple constraint: $s \leq \mathrm{M}(1-b)$.

The complete formulation is as follows:

$$\min \sum_{i}^{N} c_i z_i + Penalty \times \theta_O - Incentive \times \theta_u + Overhead \times t_{END}$$

$$s.t. \quad t_i - t_k + z_k \geq E[dur_k] \quad where\ k \in \Pi_i \quad \forall\, k, \forall i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$t_{END} - \theta_O \leq Penalty\ Target$$

$$t_{END} + \theta_u - s = Incentive\ Target$$

$$Incentive\ Target - t_{END} + \mathrm{M} - (\mathrm{M}+1)b \geq 0$$

$$\theta_u \leq \mathrm{M}b$$

$$s \leq \mathrm{M}(1-b)$$

$$z_i, t_i, \theta_O, \theta_u, s \geq 0$$

$$b = binary$$

Using the example project from Figure 5-7, we get the following formulation at time $t = 0$.

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 95\theta_O - 30\theta_U + 5t_{END}$$

$$\begin{aligned}
s.t.\quad & t_C - t_B + z_B \geq 5^{(*)} && z_A \leq 1 \\
& t_E - t_B + z_B \geq 5^{(*)} && z_B \leq 2 \\
& t_E - t_A + z_A \geq 3 && z_C \leq 1 \\
& t_D - t_C + z_C \geq 3^{(*)} && z_D \leq 2 \\
& t_{END} - t_E + z_E \geq 8 && z_E \leq 2 \\
& t_{END} - t_D + z_D \geq 4^{(*)} \\
& t_{END} - \theta_O \leq 12 \\
& t_{END} + \theta_u - s = 10 \\
& 10 - t_{END} + 1{,}000 - 1001b \geq 0 \\
& \theta_u \leq 1{,}000b \\
& s \leq 1{,}000(1-b)
\end{aligned}$$

(*) Rounded to the nearest integer

The solution to this LP is to crash task E by 1 day at a total cost of 17. Since this is a dynamic algorithm, and at time $t = 0$ we cannot start E, the decision is to crash nothing and start activities A and B. We resolve this LP substituting known durations for $E[dur_i]$, for all activities already completed or recalculating $E[dur_i]$ for those activities in progress, and setting maximum crash limit constraints to equalities for activities already started or completed (so at the next stage, after activity B finishes, we would set $z_A = 0$ and $z_B = 0$). Notice that at the beginning of this project, the LP algorithm does not take advantage of incentives. It is because the algorithm replaces uncertainty with point estimates. Using expected durations, path B-E has a length of 13 and can be shortened by at most 4 days (2 days off of B and 2 off of E). In order to get the benefit of incentives, the duration of the project needs to be at most 9 days, which we can achieve by shortening path B-E by 4 days (the maximum allowed) but that would come at a cost of 2x\$20 + 2x\$17 = \$74. The benefit of doing that is a reduction in overhead cost by 4x\$5 plus one day of

incentive payment of $25 for the total of $45 – clearly not a cost effective action from the algorithm's point of view.

5.4.5 Linear Programming with Dynamic Programming

Recall that the Linear Programming with Dynamic Programming (LPDP) method uses the uncertainty handling ability of the dynamic programming algorithm while at the same time considering path dependencies through the use of LP.

The LPDP algorithm consists of the following steps: (1) find a PERT critical path – this is done by calculating the longest path using expected durations for all activities, (2) perform DP on the PERT critical path using the modified formulation from Section 5.4.1, (3) formulate and solve a mixed integer program (from section 5.4.4) to combine DP with LP (using expected durations), (4) from the solution consider only those activities that are starting immediately and postpone all other decisions, (5) repeat the procedure at each stage of the project (when we can start another activity) substituting known durations for the expected.

Variables:

t_i = starting time of activity i
z_i = number of time periods to crash activity i
θ_O = number of time periods the project duration exceeds the Penalty Target
θ_U = number of time periods the project duration is less than the Incentive Target
z_{ij} = binary variables to look up DP policy
s = slack variable in the θ_U constraint
b = binary variable for bounding θ_U and s

Constants:

c_i = cost to crash activity i by one time period
d_i = upper limit on the number of time periods we are allowed to crash activity i
$E[dur_i]$ = expected duration of activity i (for triangular distribution, $\frac{O_i + ML_i + P_i}{3}$)

Π_i = set of immediate predecessors of activity *i*

Penalty Target = target date for project completion – if duration exceeds *PenaltyTarget*, a penalty cost is incurred

Incentive Target = target date for qualifying for incentives

Penalty = cost per day for exceeding *PenaltyTarget*

Incentive = incentive per day for completing the project before *IncentiveTarget*

p_{ij} = crossover point *j* for activity *i* (for DP policy lookup) – defined as the last start time before the number of days to crash an activity increases.

Formulation:

$$\min \sum_i^N c_i z_i + Penalty \times \theta_O - Incentive \times \theta_u + Overhead \times t_{END}$$

$$s.t.\quad t_i - t_k + z_k \geq E[dur_k] \quad where\, k \in \Pi_i \quad \forall k, \forall i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$z_i = \sum_j z_{ij} \qquad \forall i \text{ not yet started}$$

$$t_i - p_{ij} \leq \mathrm{M} z_{ij} \qquad \forall j, \forall i$$

$$t_i - (p_{ij} + 1) z_{ij} \geq 0 \qquad \forall j, \forall i$$

$$t_{END} - \theta_O \leq Penalty\ Target$$

$$t_{END} + \theta_u - s = Incentive\ Target$$

$$Incentive\ Target - t_{END} + \mathrm{M} - (\mathrm{M}+1) b \geq 0$$

$$\theta_u \leq \mathrm{M} b$$

$$s \leq \mathrm{M}(1-b)$$

$$z_i, z_{i,j}, t_i, \theta_O, \theta_u, s \geq 0$$

$$b = binary$$

Using our sample project from Figure 5-7, we get the following steps:

(1) Find a PERT critical path

In order to find PERT critical path, we need to calculate expected durations for all task: $E[dur_A] = 3$, $E[dur_B] = 5.33 \approx 5$, $E[dur_C] = 3.33 \approx 3$, $E[dur_D] = 3.67 \approx 4$, $E[dur_E] = 8$. The critical path is Start→B→E→End as illustrated in Figure 5-10.

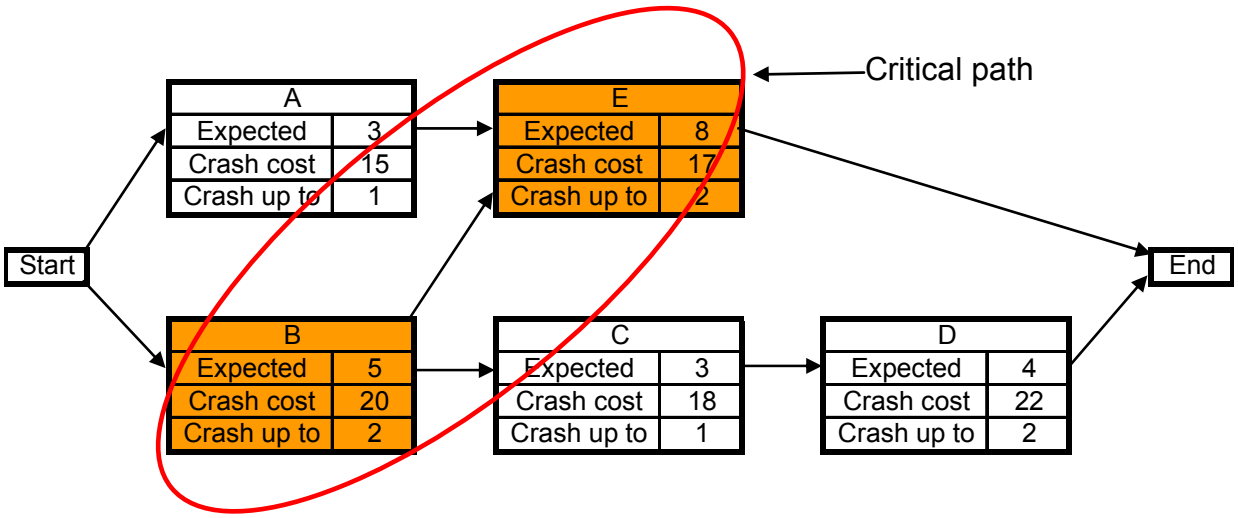


Figure 5-10: Simple project -- PERT critical path

(2) Perform DP on the critical path

The DP policy for path Start→B→E→End assuming *Penalty Target* date of 12 and *Incentive Target* of 10 is presented in Table 5-6 below.

Table 5-6: DP policy for PERT critical path

Task	ti	Crash by
B	0	2
E	1+	2

The crossover points are as follows: $p_{B1} = -2,\ p_{B2} = -1,\ p_{E1} = -1,\ p_{E2} = 0.$

(3) Formulate and solve MIP

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 95\theta_O - 30\theta_U + 5t_{END}$$

$$
\begin{aligned}
s.t.\quad & t_C - t_B + z_B \geq 5^{(*)} && z_A \leq 1 \\
& t_E - t_B + z_B \geq 5^{(*)} && z_B \leq 2 \\
& t_E - t_A + z_A \geq 3 && z_C \leq 1 \\
& t_D - t_C + z_C \geq 3^{(*)} && z_D \leq 2 \\
& t_{END} - t_E + z_E \geq 8 && z_E \leq 2 \\
& t_{END} - t_D + z_D \geq 4^{(*)} \\
& t_{END} - \theta_O \leq 12 \\
& t_{END} + \theta_u - s = 10 \\
& 10 - t_{END} + 1{,}000 - 1001b \geq 0 \\
& \theta_u \leq 1{,}000b \\
& s \leq 1{,}000(1-b)
\end{aligned}
$$

(*) Rounded to the nearest integer

$$
\begin{aligned}
& t_B - (-2) \leq 1{,}000 z_{B1} && t_B - (-2+1) z_{B1} \geq 0 \\
& t_B - (-1) \leq 1{,}000 z_{B2} && t_B - (-1+1) z_{B2} \geq 0 \\
& t_E - (-1) \leq 1{,}000 z_{E1} && t_E - (-1+1) z_{E1} \geq 0 \\
& t_E - 0 \leq 1{,}000 z_{E2} && t_E - (0+1) z_{E2} \geq 0 \\
& z_B = z_{B1} + z_{B2} \\
& z_E = z_{E1} + z_{E2} \\
& z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary
\end{aligned}
$$

When we solve the MIP, we get the following solution:

t_A	t_B	t_C	t_D	t_E	t_{END}	θo	θu	z_{B1}	z_{B2}	z_{E1}	z_{E2}	z_A	z_B	z_C	z_D	z_E
0	0	3	5	3	9	0	1	1	1	1	1	0	2	1	0	2

(4) Consider solution for the activities starting immediately

We would crash B by 2 days, not crash A, and postpone all other decisions until future stages.

(5) Repeat at each stage of the project

We would repeat this procedure, recalculating the critical path whenever we need to make a new decision, substituting known durations for the expected, and setting crash limit constraints to equalities for those activities we already made decisions about.

5.4.6 Protect the Critical Path

Recall that the PCP algorithm tries to prevent the original PERT cirtical path from becoming non-critical. Like the LPDP, it combines dynamic programming with linear (integer) programming. In addition, the PCP is a two step procedure: (1) find buffers to absorb any delays caused by the non-critical activities and (2) given those buffers find the minimum cost crashing policy.

The steps in the PCP algorithm are as follows: (1) find a PERT critical path – this is done by calculating the longest path using expected durations for all activities, (2) perform DP on the PERT critical path using formulation from section 5.4.1, (3) formulate and a solve mixed integer program that combines DP with LP to find critical path buffers, using optimistic durations for critical activities and pessimistic durations for non-critical tasks, (4) formulate and solve an MIP (combining DP with LP) to find crashing values (using optimistic and pessimistic durations accordingly), (5) from the solution consider only those activities that are starting immediately and postpone all other decisions, (5) repeat the procedure at each stage of the project (when we can start another activity) substituting known durations for the optimistic/pessimistic durations.

Variables:

t_i = starting time of activity i
z_i = number of time periods to crash activity i
θ_O = number of time periods the project duration exceeds the Penalty Target
θ_U = number of time periods the project duration is less than the Incentive Target
z_{ij} = binary variables to look up DP policy
s = slack variable in the θ_U constraint
b = binary variable for bounding θ_U and s
y_i = buffer (delay on the critical path) for activity i

Constants:

c_i = cost to crash activity i by one time period

d_i = upper limit on the number of time periods we are allowed to crash activity i

$dur_i = \begin{cases} \text{optimistic duration of activity } i \text{ if } i \text{ is critical} \\ \text{pessimistic duration of activity } i \text{ if } i \text{ is non-critical} \end{cases}$

Π_i = set of immediate predecessors of activity i

Penalty Target = target date for project completion – if duration exceeds *PenaltyTarget*, a penalty cost is incurred

Incentive Target = target date for qualifying for incentives

Penalty = cost per day for exceeding *PenaltyTarget*

Incentive = incentive per day for completing the project before *IncentiveTarget*

p_{ij} = crossover point j for activity i (for DP policy lookup)

Formulation:

Model 1: find the minimum buffers (delay on the critical path) – find y_i

$$\min \sum_i y_i$$

$$s.t.\ \ t_i + y_i - t_k - y_k + z_k \geq dur_k \quad where\ k \in \Pi_i \quad \forall k, \forall i$$

$$t_i - t_k + z_k = dur_k \qquad \forall k, i \in \text{critical path} \quad where\ k \in \Pi_i$$

$$z_i \leq d_i \qquad \forall i \text{ not yet started}$$

$$z_i = \sum_j z_{ij} \qquad \forall i \text{ not yet started}$$

$$t_i + y_i - p_{ij} \leq \mathrm{M} z_{ij} \qquad \forall j, \forall i$$

$$t_i + y_i - (p_{ij} + 1) z_{ij} \geq 0 \qquad \forall j, \forall i$$

$$t_{END} + y_{END} - \theta_O \leq PenaltyTarget$$

$$t_{END} + y_{END} + \theta_u - s = IncentiveTarget$$

$$IncentiveTarget - t_{END} - y_{END} + \mathrm{M} - (\mathrm{M}+1) b \geq 0$$

$$\theta_u \leq \mathrm{M} b$$

$$s \leq \mathrm{M}(1 - b)$$

$$y_i \geq 0 \quad \forall i \in \text{ critical path}$$

$$y_i = 0 \quad \forall i \notin \text{ critical path}$$

$$z_i, z_{i,j}, t_i, \theta_O, \theta_u, s \geq 0$$

$$b = binary$$

Constraints $t_i - t_k + z_k = dur_k$ set t_i variables for all $i \in$ critical path to minimum starting times in a serial project corresponding to the critical path.

Model 2: minimize total project cost given the minimum buffers (fixing y_i variables)

$$\min \sum_{i}^{N} c_i z_i + Penalty \times \theta_O - Incentive \times \theta_u + Overhead \times (t_{END} + y_{END})$$

$$s.t.\quad t_i + y_i - t_k - y_k + z_k \geq dur_k \quad where\ k \in \Pi_i \quad \forall\, k, \forall i$$

$$t_i - t_k + z_k = dur_k \quad where\ k \in \Pi_i \quad \forall\, k, \forall i \in \text{critical path}$$

$$z_i \leq d_i \quad \forall i \text{ not yet started}$$

$$z_i = \sum_j z_{ij} \quad \forall i \text{ not yet started}$$

$$t_i + y_i - p_{ij} \leq \mathrm{M} z_{ij} \quad \forall j, \forall i$$

$$t_i + y_i - (p_{ij} + 1) z_{ij} \geq 0 \quad \forall j, \forall i$$

$$t_{END} + y_{END} - \theta_O \leq Penalty\ Target$$

$$t_{END} + y_{END} + \theta_u - s = Incentive\ Target$$

$$IncentiveTarget - t_{END} - y_{END} + \mathrm{M} - (\mathrm{M} + 1) b \geq 0$$

$$\theta_u \leq \mathrm{M} b$$

$$s \leq \mathrm{M}(1 - b)$$

$$y_i = \text{ values from model 1} \quad \forall i \in \text{ critical path}$$

$$y_i = 0 \quad \forall i \notin \text{ critical path}$$

$$z_i, z_{i,j}, t_i, \theta_O, \theta_u, s \geq 0$$

$$b = binary$$

Recall that the objective of the PCP algorithm is to first prevent the critical path from changing and second, to minimize total project cost. In steps (3) and (4) above we are using optimistic durations for critical tasks and pessimistic durations for non-critical tasks to achieve critical path protection. Using example from shown in Figure 5-7 we get:

(1) Find PERT critical path

In order to find PERT critical path, we need to calculate expected durations for all task: $E[dur_A] = 3$, $E[dur_B] = 5.33 \approx 5$, $E[dur_C] = 3.33 \approx 3$, $E[dur_D] = 3.67 \approx 4$, $E[dur_E] = 8$. The critical path is Start→B→E→End as illustrated in Figure 5-11.

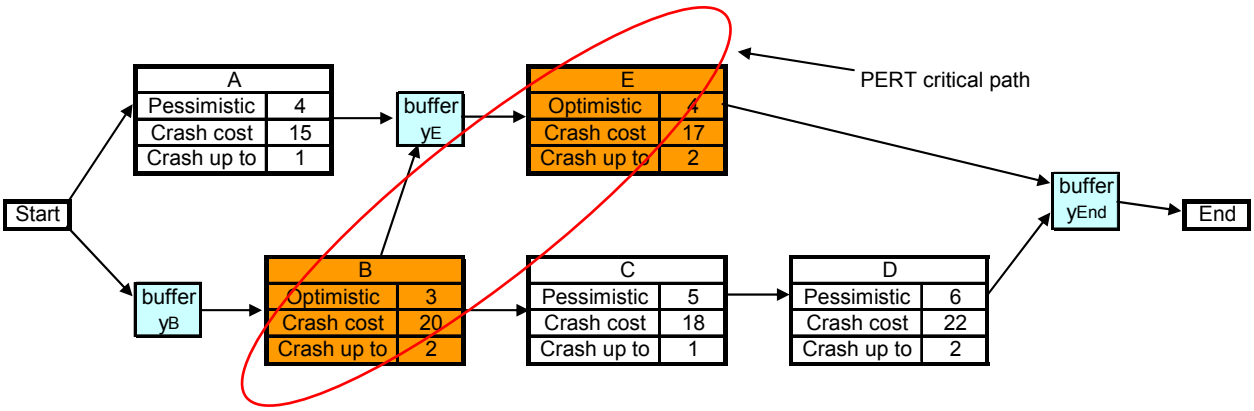


Figure 5-11: Project network diagram with buffers

(2) Perform DP on the critical path

Since, at the beginning of the project, the critical path is the same as in the LPDP algorithm, the DP policy for path Start→B→E→End is also the same (see Table 4.5). The crossover points are also $p_{B1} = -2,\ p_{B2} = -1,\ p_{E1} = -1,\ p_{E2} = 0.$

(3) Formulate and solve MIP to find buffers

$$\min\ y_A + y_B + y_C + y_D + y_E + y_{END}$$

$$\begin{aligned}
s.t.\ \ & t_C + y_C - t_B - y_B + z_B \geq 3^{(*)} && z_A \leq 1\\
& t_E + y_E - t_B - y_B + z_B \geq 3^{(*)} && z_B \leq 2\\
& t_E + y_E - t_A - y_A + z_A \geq 4^{(**)} && z_C \leq 1\\
& t_D + y_D - t_C - y_C + z_C \geq 5^{(**)} && z_D \leq 2\\
& t_{END} + y_{END} - t_E - y_E + z_E \geq 4^{(*)} && z_E \leq 2\\
& t_{END} + y_{END} - t_D - y_D + z_D \geq 6^{(**)}\\
& t_E - t_B + z_B = 3^{(*)}\\
& t_{END} - t_E + z_E = 4^{(*)}\\
& t_{END} + y_{END} - \theta_O \leq 12\\
& t_{END} + y_{END} + \theta_u - s = 10\\
& 10 - t_{END} - y_{END} + 1{,}000 - 1001b \geq 0\\
& \theta_u \leq 1{,}000b\\
& s \leq 1{,}000(1-b)
\end{aligned}$$

(*) Using optimistic duration

(**) Using pessimistic duration

$$t_B + y_B - (-2) \le 1{,}000 z_{B1} \qquad t_B + y_B - (-2+1) z_{B1} \ge 0$$
$$t_B + y_B - (-1) \le 1{,}000 z_{B2} \qquad t_B + y_B - (-1+1) z_{B2} \ge 0$$
$$t_E + y_E - 2 \le 1{,}000 z_{E1} \qquad t_E + y_E - (2+1) z_{E1} \ge 0$$
$$t_E + y_E - 3 \le 1{,}000 z_{E2} \qquad t_E + y_E - (3+1) z_{E2} \ge 0$$
$$z_B = z_{B1} + z_{B2}$$
$$z_E = z_{E1} + z_{E2}$$
$$z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary$$
$$y_B, y_E, y_{END} \ge 0$$
$$y_A, y_C, y_D = 0$$

The solution is $y_A = 0, y_B = 0, y_C = 0, y_D = 0, y_E = 2, y_{END} = 7$

(4) Formulate and solve MIP to find crashing values

$$\min 15c_A + 20c_B + 18c_C + 22c_D + 17c_E + 95\theta_O - 30\theta_U + 5t_{END}$$

s.t.
$$t_C + y_C - t_B - y_B + z_B \ge 3^{(*)} \qquad z_A \le 1$$
$$t_E + y_E - t_B - y_B + z_B \ge 3^{(*)} \qquad z_B \le 2$$
$$t_E + y_E - t_A - y_A + z_A \ge 4^{(**)} \qquad z_C \le 1$$
$$t_D + y_D - t_C - y_C + z_C \ge 5^{(**)} \qquad z_D \le 2$$
$$t_{END} + y_{END} - t_E - y_E + z_E \ge 4^{(*)} \qquad z_E \le 2$$
$$t_{END} + y_{END} - t_D - y_D + z_D \ge 6^{(**)}$$
$$t_E - t_B + z_B = 3^{(*)}$$
$$t_{END} - t_E + z_E = 4^{(*)}$$
$$t_{END} + y_{END} - \theta_O \le 12$$
$$t_{END} + y_{END} + \theta_u - s = 10$$
$$10 - t_{END} - y_{END} + 1{,}000 - 1001b \ge 0$$
$$\theta_u \le 1{,}000b$$
$$s \le 1{,}000(1-b)$$

(*) Using optimistic duration

(**) Using pessimistic duration

$$t_B + y_B - (-2) \le 1{,}000 z_{B1} \qquad t_B + y_B - (-2+1) z_{B1} \ge 0$$
$$t_B + y_B - (-1) \le 1{,}000 z_{B2} \qquad t_B + y_B - (-1+1) z_{B2} \ge 0$$
$$t_E + y_E - 2 \le 1{,}000 z_{E1} \qquad t_E + y_E - (2+1) z_{E1} \ge 0$$
$$t_E + y_E - 3 \le 1{,}000 z_{E2} \qquad t_E + y_E - (3+1) z_{E2} \ge 0$$

$$z_B = z_{B1} + z_{B2}$$
$$z_E = z_{E1} + z_{E2}$$
$$z_{B1}, z_{B2}, z_{E1}, z_{E2} = binary$$
$$y_E = 2$$
$$y_{END} = 7$$
$$y_A, y_C, y_D, y_B = 0$$

t_A	t_B	t_C	t_D	t_E	t_{END}	θo	θu	z_{B1}	z_{B2}	z_{E1}	z_{E2}	z_A	z_B	z_C	z_D	z_E
0	0	1	5	1	3	0	0	1	1	1	1	1	2	1	1	2

(5) Consider solution for the activities starting immediately

We would crash B by 2 days, crash A by 1 day, and postpone all other decisions until future stages. Notice the difference between the LPDP and the PCP crashing policies. The PCP crashes activity A by 1 day to shorten non-critical (from PERT's perspective) path giving the current critical path a greater chance of staying critical.

(6) Repeat at each stage of the project

We would repeat this procedure, recalculating the critical path whenever we need to make a new decision, substituting known durations for the optimistic/pessimistic (updating probability distributions if necessary), and setting crash limit constraints to equalities for those activities we already made decisions about.

5.5 COMPUTATIONAL TESTS

5.5.1 Generating Problem Instances

We used the same method for generating project topologies as in Chapter 4 based on Demeulemeester et al. (2003). The procedure for generating relevant costs, presented in Chapters

3 and 4, was modified slightly to account for the presence of incentives. As before, we set the *Penalty Target* equal to the expected duration of the PERT critical path. The *Incentive Target* is set to 90% of the *Penalty Target* (rounded to the nearest integer). The values of *Incentive*, *Overhead*, and *Penalty* parameters depend on the cost sharing ratio, which we set to r_1 = 30:70, r_2 = 5:95, and r_3 = 100:0 as well as on the value of the indirect cost (set to \$100 per day). This is equivalent to a project with *Penalty* = \$95, *Overhead* = \$5, and *Incentive* = \$25. It is important to note that different ratios or different *Incentive Target* could be used and, in fact, we ran preliminary experiments testing various values of those parameters. We found that the values we chose produced the most reasonable problem instances, that is, we rarely encountered projects with obvious solutions (such as always crash everything or never crash anything). Next, we simulate the project assuming no crashing is used N times where N is equal to the number of activities in the project multiplies by a constant (20 in our case). The procedure consists of the following steps:

(1) Determine the size of the project. We tested problems with 5, 10, 25, and 50 activities.

(2) Set distribution span. As in the general case (Chapter 4), we only looked at an average distribution span of 16 days.

(3) For each activity generate optimistic, most likely, and pessimistic times according to the span.

(4) Calculate the *Penalty Target* date.

(5) Calculate the *Incentive Target* date.

(6) Determine the maximum number of days by which each activity can be crashed (d_i) – generated from a discrete uniform distribution from [0, O_i-1] interval.

(7) Generate crash cost per day for each activity

a. Estimate expected total cost E(Cost) of the project with no compression using Monte Carlo simulation.

b. Assign a fraction of E(Cost) as the total cost for maximum crashing (*TCMC*) of all activities (total cost if we crash all activities to maximum). We refer to this fraction as a *cost structure*. In this Chapter we only use cost structure of 1 for convenience.

c. Calculate adjustment value for each activity (A_i)

$$A_i = \frac{d_i}{\max_k \{d_k\}} \quad \forall k \in \{1, ..., n\}$$

d. For each activity, generate a random number ~U[0, 1) and multiply it by A_i. We refer to this value as a cost distribution index (*CDI*).

e. Calculate a normalized cost distribution index (*NCDI*) for each activity:

$$NCDI_i = \frac{CDI_i}{\sum_{k=1}^{n} CDI_k}$$

f. Calculate cost of maximum crashing for each activity:

$$\text{cost of maximum crashing}_i = TCMC \times NCDI_i$$

g. Calculate crash cost per day for each activity:

$$c_i = \frac{\text{cost of maximum crashing}_i}{d_i}$$

5.5.2 Results

The results are presented in the following manner. We first discuss problem types with different project sizes and next we illustrate differences between dynamic (contingent decisions) vs. static (all decisions made at the beginning of the project) algorithms. We also apply the perfect

information algorithm (described in Chapter 4) to all the problem instances for comparison purposes. Recall that this is achieved by making all activity durations known to the algorithm a priori, thus reducing the problem to its deterministic counterpart, and solve using linear programming.

Size =5

Tables 5-7 and 5-8 show average expected costs and average running times by order strength and heuristic for five activity projects. The same information is also presented in Figures 5-12 and 5-13.

Table 5-7: Expected cost -- 5 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	214.67	214.64	246.21	224.31	264.86	172.87
0.1	240.54	240.55	266.51	251.84	308.43	202.39
0.2	343.66	343.73	364.65	346.60	480.30	289.38
0.3	319.62	319.41	349.64	336.94	445.91	252.02
0.4	378.95	378.80	405.73	381.62	543.71	331.92
0.5	222.10	221.95	250.66	228.30	383.85	172.02
0.6	419.65	419.64	450.40	425.58	593.30	370.03
0.7	328.41	328.73	360.08	334.25	508.91	286.84
0.8	318.54	318.62	344.44	327.30	491.43	271.91
0.9	426.34	426.13	458.73	420.49	536.69	372.12
1	500.70	500.74	529.19	498.84	498.84	448.00

*Averaged over 20 instances of each problem type

Table 5-8: Average running time (sec) -- 5 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.00077	0.00075	0.00637	0.01036	0.01970
0.1	0.00103	0.00080	0.00982	0.01812	0.03145
0.2	0.00106	0.00121	0.01412	0.02617	0.04690
0.3	0.00120	0.00116	0.01214	0.02453	0.04430
0.4	0.00089	0.00090	0.01047	0.01695	0.03737
0.5	0.00158	0.00148	0.01176	0.02575	0.05901
0.6	0.00140	0.00146	0.01510	0.03523	0.06436
0.7	0.00190	0.00207	0.01429	0.03289	0.07107
0.8	0.00206	0.00220	0.01496	0.06419	0.10293
0.9	0.00221	0.00224	0.01763	0.08200	0.11282
1	0.00187	0.00189	0.02106	0.07285	0.10080

*Averaged over 20 instances of each problem type

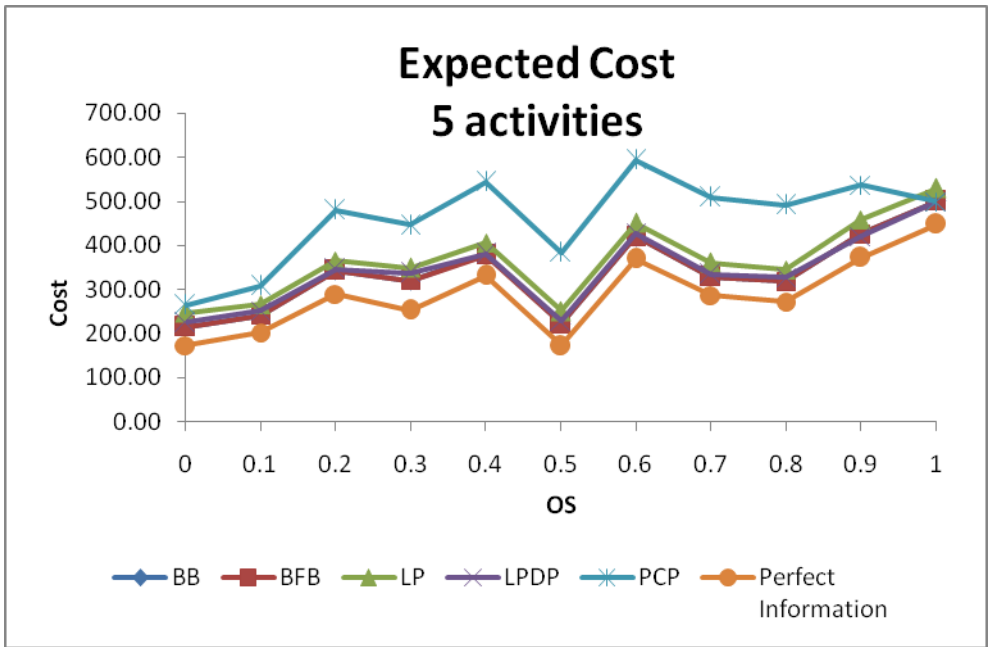


Figure 5-12: Expected cost -- 5 activity projects

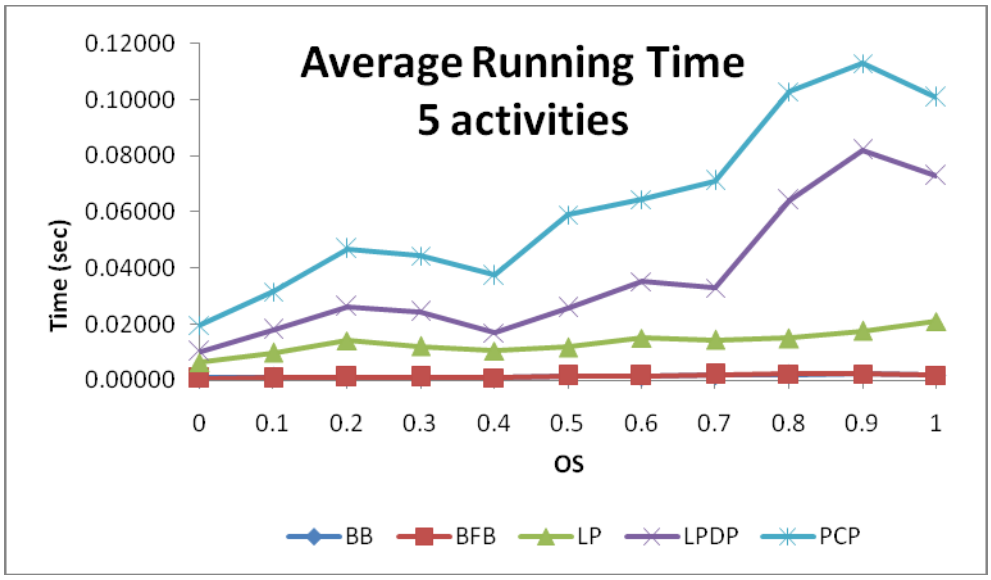


Figure 5-13: Average running time (sec) -- 5 activity projects

Size =10

Tables 5-9 and 5-10 show average expected costs and average running times by order strength and heuristic for ten activity projects. The same information is also presented in Figures 5-14 and 5-15.

Table 5-9: Expected cost -- 10 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	263.25	263.02	301.45	278.44	294.55	208.51
0.1	284.45	284.44	309.34	296.52	378.00	235.92
0.2	367.29	366.74	413.18	388.57	479.40	293.42
0.3	479.61	479.56	515.22	487.85	688.91	417.00
0.4	466.77	466.66	527.54	460.16	684.45	417.31
0.5	463.05	463.37	500.09	477.92	685.92	374.76
0.6	598.53	598.57	668.71	628.95	857.76	502.60
0.7	592.62	592.11	626.90	614.92	821.12	510.57
0.8	842.83	843.10	896.04	863.33	1072.49	737.49
0.9	874.80	874.77	943.77	874.65	1028.46	782.79
1	1068.80	1068.77	1128.12	1062.45	1062.45	955.73

*Averaged over 20 instances of each problem type

Table 5-10: Average running time (sec) -- 10 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.00406	0.00409	0.00701	0.01395	0.02630
0.1	0.00655	0.00654	0.01341	0.03939	0.06659
0.2	0.00960	0.00951	0.01729	0.04760	0.07940
0.3	0.00906	0.00903	0.02045	0.07127	0.12043
0.4	0.00938	0.00929	0.02291	0.10082	0.17146
0.5	0.01584	0.01583	0.02547	0.11137	0.19880
0.6	0.01498	0.01491	0.02539	0.10433	0.17584
0.7	0.01463	0.01456	0.02961	0.16011	0.26544
0.8	0.01603	0.01605	0.03313	0.25964	0.37144
0.9	0.01698	0.01699	0.03665	0.37354	0.45963
1	0.01736	0.01761	0.04541	0.56301	0.68367

*Averaged over 20 instances of each problem type

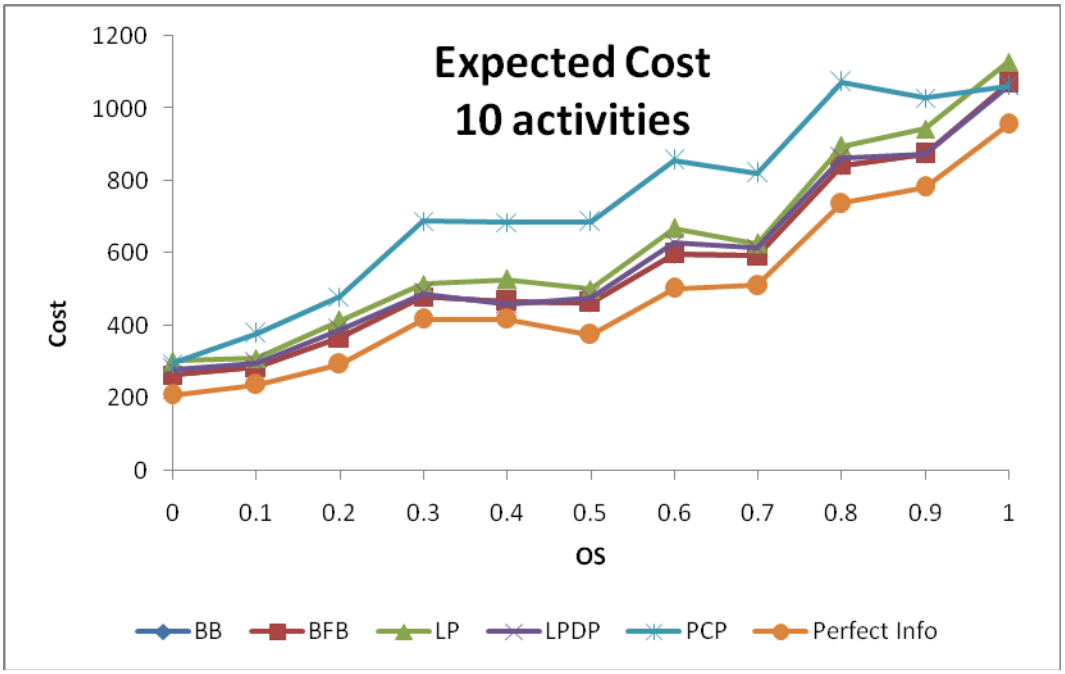


Figure 5-14: Expected cost -- 10 activity projects

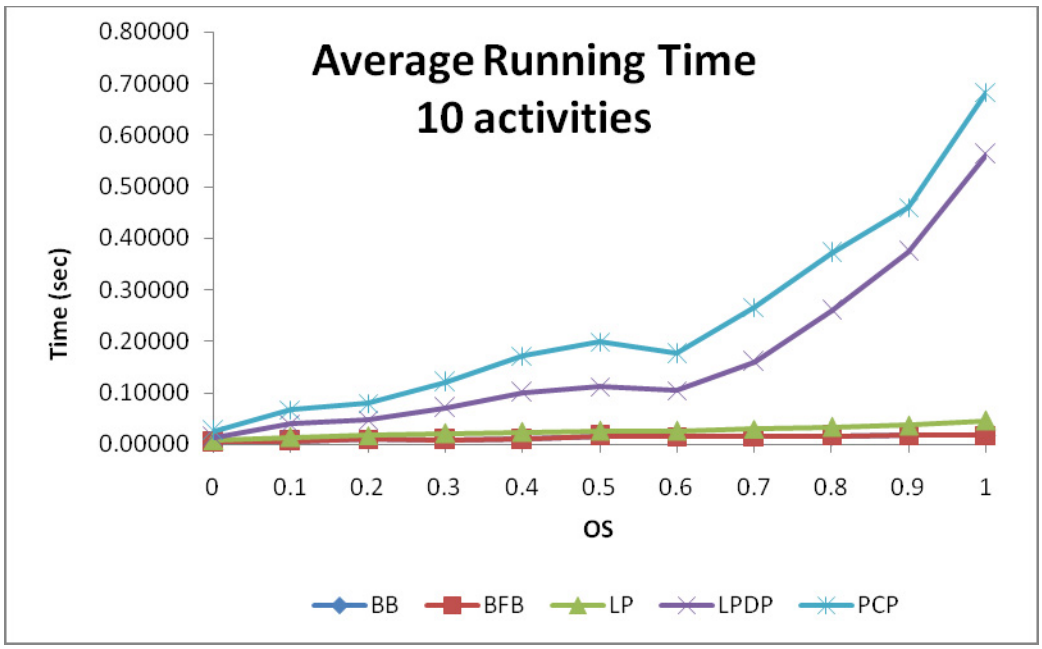


Figure 5-15: Average running time -- 10 activity projects

Size = 25

Tables 5-11 and 5-12 show average expected costs and average running times by order strength and heuristic for twenty five activity projects. The same information is also presented in Figures 5-16 and 5-17.

Table 5-11: Expected cost -- 25 activity projects

OS	BB	BFB	LP	LPDP	PCP	Perfect Information
0	361.46	361.43	418.37	391.42	389.81	306.50
0.1	319.41	319.51	373.83	353.24	385.50	231.15
0.2	463.40	463.53	518.02	485.26	572.91	370.17
0.3	484.29	484.27	541.20	516.40	616.66	382.61
0.4	563.16	563.31	627.15	587.54	755.25	464.11
0.5	562.44	562.62	607.72	592.17	816.76	480.62
0.6	846.22	846.09	926.49	881.80	1155.44	735.23
0.7	859.04	858.52	955.15	904.87	1302.93	720.38
0.8	1098.09	1097.79	1277.49	1165.93	1582.63	934.50
0.9	1463.51	1463.32	1630.15	1493.41	2151.65	1298.39
1	1413.51	1413.63	1505.01	1399.98	1399.98	1251.69

*Averaged over 20 instances of each problem type

Table 5-12: Average running time -- 25 activity projects

OS	BB	BFB	LP	LPDP	PCP
0	0.03983	0.03995	0.01054	0.02974	0.05959
0.1	0.16949	0.17045	0.04169	0.21702	0.46134
0.2	0.20384	0.20501	0.05849	0.28838	0.91822
0.3	0.24439	0.24650	0.06639	0.40159	1.26682
0.4	0.25188	0.25372	0.07715	0.61382	1.79200
0.5	0.22065	0.22211	0.07787	0.76095	2.53029
0.6	0.33951	0.34190	0.09446	1.81201	5.13433
0.7	0.45658	0.45992	0.10749	2.37395	5.39899
0.8	0.33139	0.33417	0.11492	2.71358	6.22986
0.9	0.38611	0.38801	0.12965	10.68737	24.25111
1	0.54925	0.55074	0.17692	76.96781	62.36829

*Averaged over 20 instances of each problem type

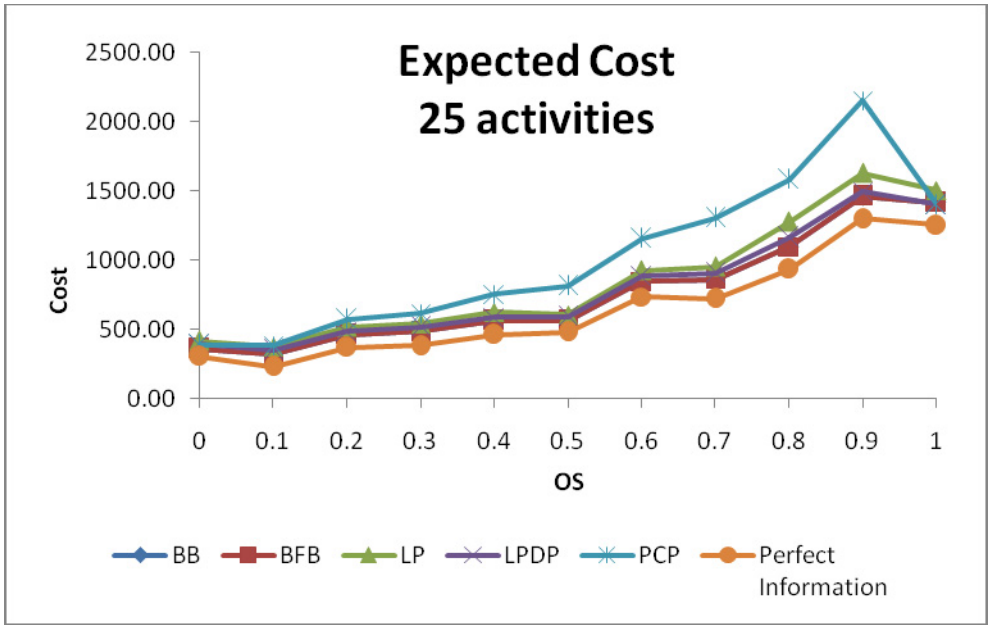


Figure 5-16: Expected cost -- 25 activity projects

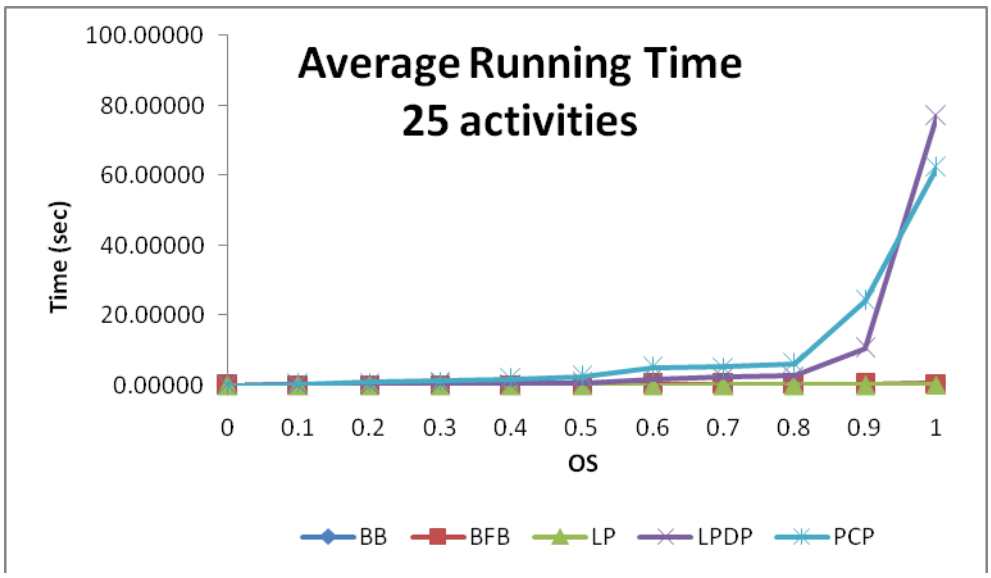


Figure 5-17: Average running time -- 25 activity projects

The relative cost performance of the algorithms is similar to the findings from Chapter 4. Again the Biggest Bang and the Bang for the Buck the best performers; however, the gaps between the algorithms are much smaller than before. We notice that as the order strength increases so does the expected cost for all algorithms. We conjecture that this is due to the way the Incentive Target is generated. In the problem instances tested, the *Incentive Target* was set equal to 90% of the *Penalty Target*. Therefore, the larger *Penalty Target*, the larger the gap between the *Penalty Target* and the *Incentive Target* and the harder it is to reduce the project's duration to the point where incentives can be earned. Since the *Penalty Target* depends on the length of the longest path calculated using expected activity duration estimates, the larger the OS, the more serial the project, and the larger the *Penalty Target*. As before for OS = 1 (serial project), dynamic programming gives optimal solutions, therefore the LPDP and the PCP have the lowest cost.

We also suspected that the performance of the BB and the BFB may suffer because these two are myopic algorithms that look at cost savings from crashing one activity by one time

period at a time. If the difference between the *Penalty Target* and the *Incentive Target* is large, the BB indices (or the BFB indices) may indicate that nothing should be crashed once the maximum duration of the project becomes smaller than the *Penalty Target* (or if the probability of exceeding the *Penalty Target* is sufficiently small). Consider a simple (serial) project in Figure 5-18.

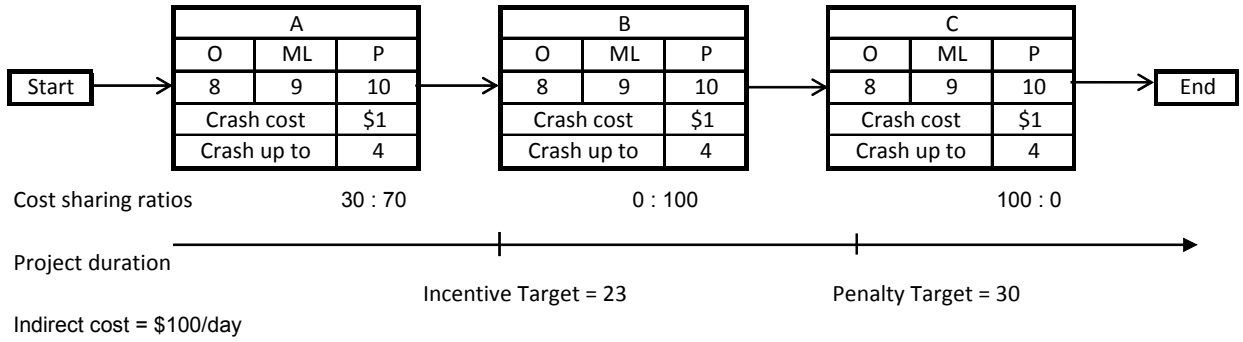


Figure 5-18: BB/BFB illustration (serial project)

The maximum project duration is equal to the sum of the pessimistic times, or 30 days which is less than or equal to the *Penalty Target*. Therefore, none of the activities will ever be Penalty Target Critical and all PTCIs are equal to 0. The minimum project duration is 24 days which is greater than the *Incentive Target*, thus all ITCIs are zero as well. Since this is a serial project, all regular Criticality Indices (CIs) are 1 but, because there is no *Overhead*, the BB indices and the BFB indices are equal to:

$$BBI_{A,B,C} = PTCI \times Penalty + ITCI \times Incentive + CI \times Overhead - c = 0 \times 100 + 0 \times 30 + 1 \times 0 - 1 = -1$$

$$BFBI_{A,B,C} = \frac{PTCI \times Penalty + ITCI \times Incentive + CI \times Overhead - c}{c_i} = \frac{-1}{1} = -1$$

Thus we would never crash any of the activities even though it is obvious that the optimal policy is to crash everything to the maximum. Nevertheless, the BB and the BFB give the best results, which suggest that the project generating procedure was able to avoid creating such problem instances, in the majority of cases.

Running times for all algorithms go up as the order strength and size increase and again, they increase at different rates. The LPDP and the PCP have an even greater number of integer variables than in Chapter 4, which slows down these algorithms even further. The LPDP average running time of one simulation of one problem instance with 25 activities and OS of 1 was about 77 seconds. Since we simulate each problem instance 20*n times (in this case, 20*25 = 500), and we have 20 instances of each problem type, the length of time to run the LPDP on a problem set with 25 activities and OS=1 was approximately 770,000 seconds or 213 hours and 53 minutes. Again, the computational requirements to test the LPDP and the PCP on bigger problem instances were prohibitively large.

5.5.3 Impact of the order strength

As in Chapters 3 and 4, we examine relative cost improvement of dynamic vs. static algorithms. Figures 5-19 and 5-20 present differences in the expected costs of static and dynamic versions of the algorithms for the BB and the LPDP respectively on problem instances with 25 activities. The results show expected costs averaged over 20 instances of each problem type.

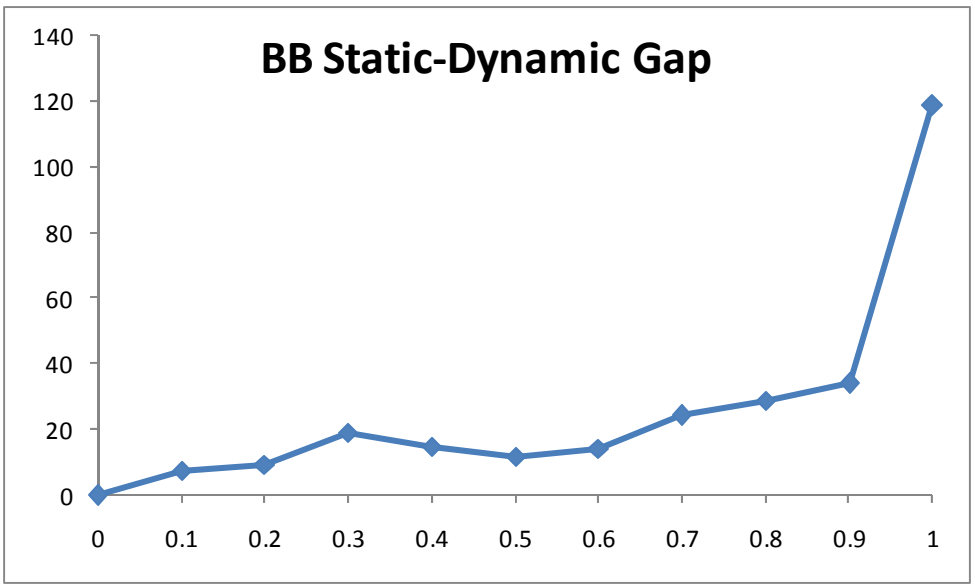


Figure 5-19: Biggest Bang -- static vs. dynamic

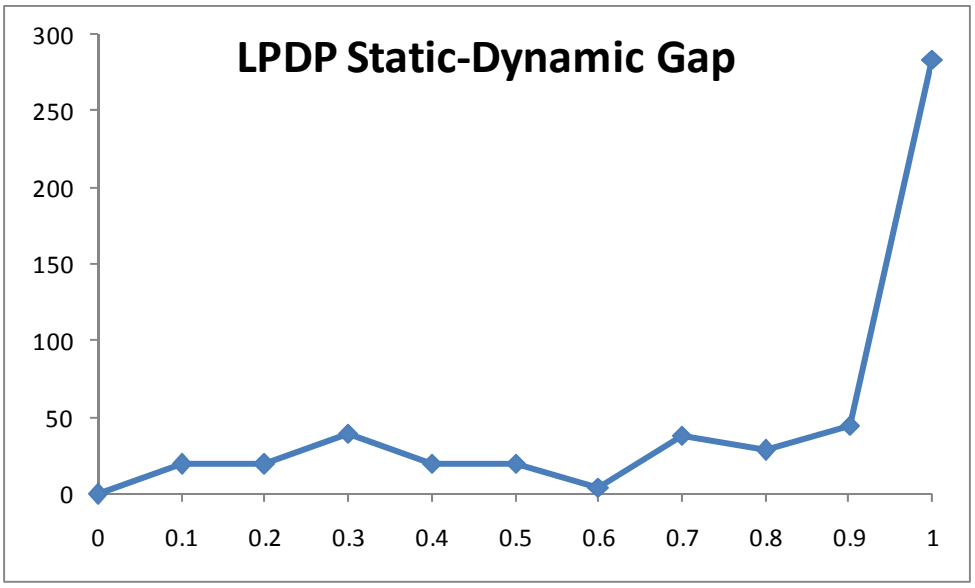


Figure 5-20: LPDP -- static vs. dynamic

The results are not as dramatic as in Chapter 4; however there is a visible direct correlation of the order strength and the cost difference. In a completely parallel project (OS = 0) the performance of a dynamic algorithm should the same as the performance of a static algorithm because we have only one decision stage. As the number of decision stages increases, which

happens when we increase the OS, we have more opportunities in the dynamic version of an algorithm to revise our decisions, therefore the difference in performance of the dynamic and the static algorithms grows. For the order strength of 1, the relative improvement of a dynamic algorithm over a static one is around 8.4% for the BB and 20.3% for the LPDP.

We also examined the gap between the perfect information case and our algorithms. Table 5-13 shows the BB-Perfect Information gap by project size for OS of 0 and OS of 1.

Table 5-13: BB/Perfect Info % gaps

	%Gap		
Size	OS=0	OS=1	Difference
5	24.18%	11.76%	-12.42%
10	26.25%	11.83%	-14.42%
25	17.93%	12.93%	-5.00%

Unlike in Chapter 4, the percentage gap between the BB cost and the Perfect Information case seems to get smaller as projects become more serial. This is a surprising result until we take the change in the cost magnitude into consideration. In calculating the percentage gaps, we divide the absolute difference in costs by the cost of the Perfect Information case. The expected cost with Perfect Information for serial projects (OS=1) is about 4 times as large as for parallel projects, therefore the denominator in the serial case is much larger. Examining absolute differences in costs (presented in Table 5-14) we notice that the absolute gaps increase with the OS, which is the expected results. However, the differences are not as pronounced as in Chapter 4, which may suggest that the problem instances generated were easier to solve.

Table 5-14: BB/Perfect Info absolute gaps

	Absolute Gap		
Size	OS=0	OS=1	Difference
5	41.80	52.70	10.90
10	54.73	113.07	58.33
25	54.96	161.82	106.86

Notice that both the percentage gaps and the absolute gaps increase as the project size increases, which is consistent with our intuition – since for serial projects (OS=1), the number of decision stages is equal to the number of activities, thus the larger the serial project, the greater should be the value of information.

5.6 EXTENSIONS TO MORE COMPLEX COST STRUCTURES

In this chapter we focused on the case with three levels of cost sharing – one level for cost under-run (incentive), one for cost over-run (penalty), and one for constant overhead. However, as we mentioned before, project contracts can specify multiple levels of cost sharing. Figure 5-21 illustrates the case with two penalty levels (over-run only) plus a constant overhead. When the project exceeds the target date of 100 days, the contractor pays a penalty of $30 per day until the duration reaches 120 days. Beyond that, the contractor is responsible for 100% of the overhead cost – therefore this is a case with increasing penalties.

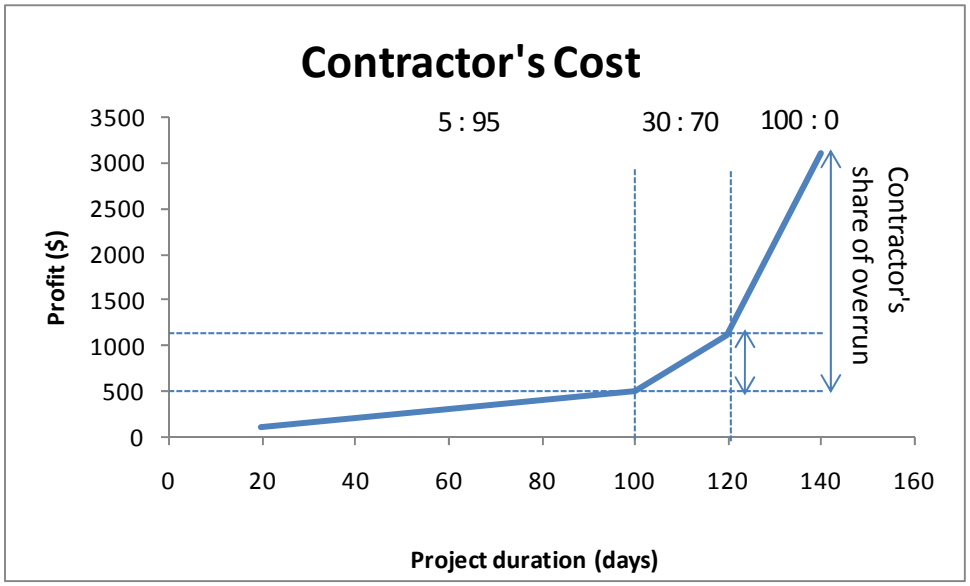


Figure 5-21: Three levels of cost sharing (penalty only)

As in the penalty only case, we can have multiple levels of sharing for the incentives as illustrated in Figure 5-22. In this case, the contractor pays fixed 5% of the cost overhead and receives incentives of 25% of cost under-run if the project is completed in under 70 days and 15% of cost under-run if the project is completed in more than 70 days but before the Target date (thus it is a case with decreasing incentives).

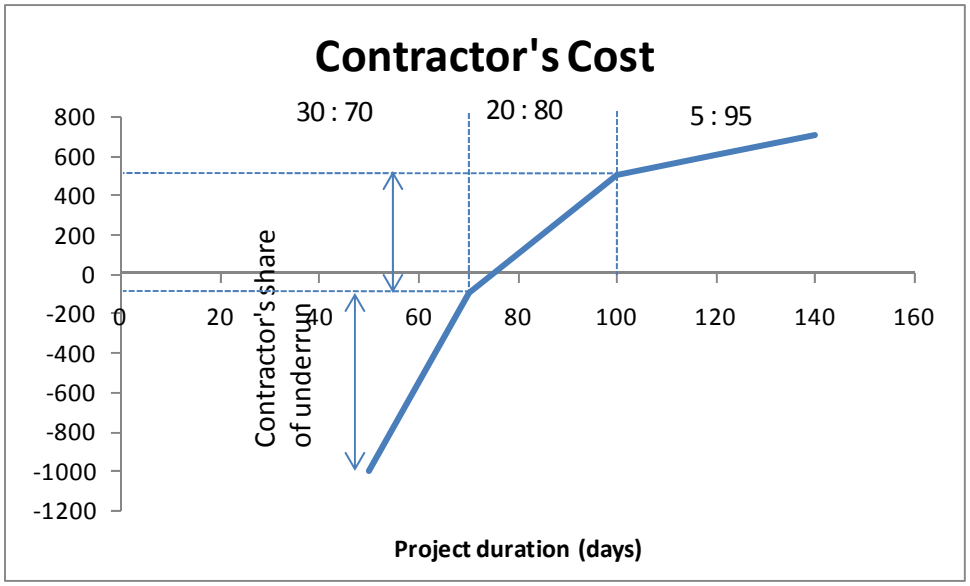


Figure 5-22: Three levels of cost sharing (incentives only)

As before, we can also have a case with both penalties and incentives. Figure 5-23 illustrates the case with five levels of cost sharing: two levels of incentives (decreasing), one level of constant overhead, and two levels of penalties (decreasing). Decreasing penalties can occur in a situation where there is a cap on the amount of penalty that can be imposed on the contractor. The cost function in Figure 5-23 is non-convex with two inflection points – one at 100 and the other at 130.

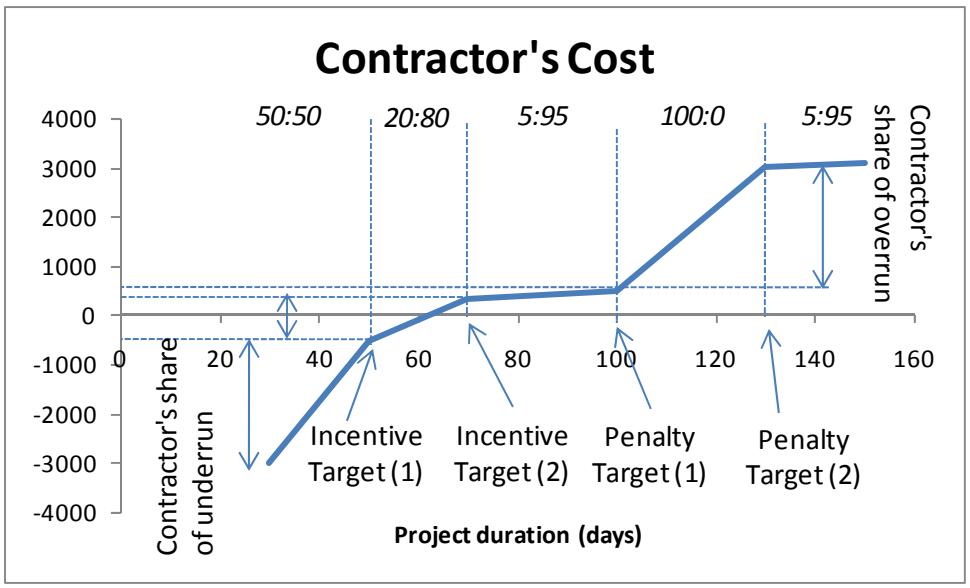


Figure 5-23: Five levels of cost sharing (incentives and penalties)

The question now is whether the algorithms presented in this chapter are applicable, or at least easily extendable to the more general case (such as in Figure 5-23). For the BB and the BFB the extension should be straightforward. Recall that those two methods calculate different criticality indices for each ratio or line segment. We could follow a similar procedure but we would have a larger number of criticality indices, equal to the number of straight line segments in the cost function. However, the other methods, especially those based on the DP, would become more complex. When the cost function has multiple inflection points, the DP solution may have several increasing and decreasing segments as shown in Figure 5-24. This creates the need for an even greater number of binary variables in the LPDP and the PCP formulations.

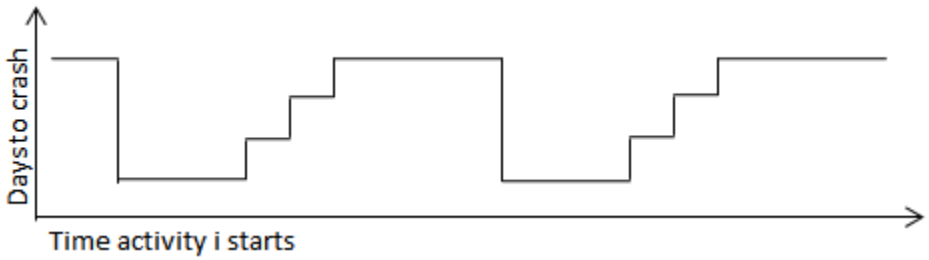


Figure 5-24: DP solution (multiple increasing/decreasing segments)

5.7 CHAPTER SUMMARY

In this chapter we considered a stochastic time-cost trade-off problem for projects with penalties, incentives, and overhead with the goal of minimizing the contractor project cost in presence of cost-plus-incentive-fee contracts. In such contracts, the contractor and the owner of the project share the associated risk according to some agreed upon ratio, which can change according to the project duration. We examined a case with three cost share ratios where a fixed percentage of the project indirect overhead cost is always incurred by the contractor. In addition, the contractor can receive a portion of cost under-runs (if the project duration is shorter than some predetermined date, which we denote as the *Incentive Target*) and is also responsible for a portion of cost over-runs (if the project duration exceeds the *Penalty Target*).

We extended and applied the five algorithms presented in Chapter 4 to the problem discussed herein. We showed that, when the contractor's cost function is non-convex, the dynamic programming solution loses its monotonicity, which forces the inclusion of additional binary variables in the LPDP and the PCP formulation. We were also concerned that the BB and the BFB would not perform well due to the myopic nature of those two methods. In theory it is possible for the BB and the BFB to stop prematurely, that is, if the difference between the *Penalty Target* and the *Incentive Target* is large and if the *Overhead* amount is sufficiently small, the algorithms will conclude that crashing should be terminated when, in reality, we can realize additional cost savings by reducing the project beyond the *Incentive Target*. However, the computational tests performed again pointed to the BB and the BFB as the best methods, which suggests that, using the described project generator, we avoided such problem instances.

In addition, we noticed that the gaps between the expected costs of the algorithms and the case with perfect information are smaller than those observed in Chapter 4. In general, perfect

information is worth more when the project is more serial but this relationship is also not as strong as in Chapter 4. This may suggest that the problem instances we generated herein are, on average, easier to solve than those generated in Chapter 4. This could be due to the fact that even when over-crashing we can still get the benefits of incentives and decreased overhead cost. One of the possible future research avenues is therefore to investigate the impact of the cost share ratio values on the performance of the algorithms.

Furthermore, one could examine whether different cost functions have an effect on how well the methods perform. Recall that in this chapter we used three levels of cost sharing. It would be interesting to study cost functions with a higher number of share levels. If the share ratios are monotonically increasing (as shown in Figure 5-21), the cost function is convex. If the share ratios are monotonically decreasing (Figure 5-22), the cost function is concave. In those instances extending our methods should be relatively straightforward. In addition, when the cost function is non-convex with multiple inflection points the BB and the BFB, which turned out to be reasonably good techniques, should be easily extendable as well. It is however not clear how to modify the LP, the LPDP, or the PCP to deal with more share levels. Our intuition is that the performance of the methods may deteriorate; however, to answer this question with any degree of confidence, more computational experiments are necessary.

BIBLIOGRAPHY

Adlakha, V. G. and V. G. Kulkarni (1989). "A classified Bibliography of Research on Stochastic PERT Networks: 1966-1987." INFOR **27**(3): 272-296.

Al-Harbi, K. and M. Al-Subhi (1998). "Sharing fractions in cost-plus-incentive-fee contracts." International Journal of Project Management **16**(2): 73-80.

Assaf, S. A. and S. Al-Hejji (2006). "Causes of delay in large construction projects." International Journal of Project Management **24**: 349-357.

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

Bellman, R. (1958). "Dynamic Programming and Stochastic Control Processes." Information and Control **1**: 228-239.

Berends, T. C. (2000). "Cost plus incentive fee contracting -- experiences and structuring." International Journal of Project Management **18**: 165-171.

Bonferroni, C. E. (1936). "Teoria statistica delle classi e calcolo delle probabilità." Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commerciali di Firenze **8**: 3-62.

Bower, D., G. Ashby, et al. (2002). "Incentive Mechanisms for Project Success." Journal of Management in Engineering **18**(1): 37-43.

Broome, J. and J. Perry (2002). "How practitioners set share fractions in target cost contracts " International Journal of Project Management **20**: 59-66.

Bubshait, A. (2003). "Incentive/disincentive contracts and its effects on industrial projects." International Journal of Project Management **21**: 63-70.

Buss, A. H. and M. J. Rosenblatt (1997). "Activity Delay in Stochastic Project Networks." Operations Research **42**(1): 126-139.

Demeulemeester, E., B. De Reyck, et al. (1998). "New Computational Results on Discrete Time/Cost trade-Off Problem in Project Networks." Journal of the Operational Research Society **49**(11): 1153-1163.

Demeulemeester, E., B. Dodin, et al. (1993). "A Random Activity Network Generator." Operations Research **41**(5): 972-980.

Demeulemeester, E., M. Vanhoucke, et al. (2003). "RanGen: a random network generator for activity-on-the-node networks." Journal of Scheduling **6**: 17-38.

Dodin, B. (1985a). "Approximating the Distribution Functions in Stochastic Networks." Computers and Operations Research **12**(3): 251-264.

Dodin, B. (1985b). "Bounding the Project Completion Time Distribution in PERT Networks." Operations Research **33**(4): 862-881.

Elmaghraby, S. E. (1967). "On the Expected Duration of PERT Type Networks." Management Science **13**(5): 299-306.

Elmaghraby, S. E. (1977). Activity Networks: Project Planning and Control by Network Models. New York, NY, John Wiley & Sons, Inc.

Elmaghraby, S. E. (2000). "On Criticality and Sensitivity in Activity Networks." European Journal of Operational Research **127**(2): 220-238.

Elmaghraby, S. E. (2005). "On the fallacy of averages in project risk management." European Journal of Operational Research **165**(2): 307-313.

Elmaghraby, S. E. and W. Herroelen (1980). "On the Measurement of Complexity in Activity Networks." European Journal of Operational Research **5**(4): 223-234.

Goldratt, E. M. (1997). Critical Chain: A Business Novel. Great Barrington, MA, North River Press.

Gutjahr, W. J., C. Strauss, et al. (2000). "Crashing of Stochastic Processes by Sampling and Optimisation." Business Process Management Journal **6**(1): 65-83.

Gutjahr, W. J., C. Strauss, et al. (2000). "A Stochastic Branch-and-Bound Approach to Activity Crashing in Project Management." INFORMS Journal on Computing **12**(2): 125-135.

Haga, W. A. (1998). Crashing PERT Networks. Applied Statistics Department. Greeley, CO, University of Northern Colorado. **Doctoral Dissertation**.

Haga, W. A. and T. O'Keefe (2001). Crashing PERT Networks: a Simulation Approach. 4th International Conference of the Academy of Business and Administrative Sciences, Quebec City, Canada.

Herbert, J. E. (1979). Applications of Simulation in Project Management. Winter Simulation Conference, San Diego, CA.

Herroelen, W. and R. Leus (2005). "Project Scheduling under Uncertainty: survey and research potentials." European Journal of Operational Research **165**(2): 289-306.

Herroelen, W., R. Leus, et al. (2002). "Critical Chain Project Scheduling: Do Not Oversimplify." Project Management Journal **33**(4): 48-60.

Herten, H. J. and W. A. R. Peeters (1986). "Incentive contracting as a project management tool." Project Management **4**(1): 34-39.

Howard, R. A. and J. E. Matheson (1981). Influence Diagrams. Readings on the Principles and Applications of Decision Analysis. R. A. Howard and J. E. Matheson. Menlo Park, CA, Strategic Decisions Group. **2:** 719-762.

Ioannou, P. G. and J. C. Martinez (1998). Project Scheduling Using State-Based Probabilistic Decision Networks. Winter Simulation Conference, Washington, DC.

Jaraiedi, M., R. W. Plummer, et al. (1995). "Incentive/Disincentive Guidelines for Highway Construction Contracts." Journal of Construction Engineering and Management **121**: 112-120.

Jenzarli, A. (1995). Modeling Dependence in Project Management, University of Kansas. **Doctoral Dissertation**.

Johnson, G. A. and C. D. Schou (1990). "Expediting Projects in PERT with Stochastic Time Estimates." Project Management Journal **21**(2): 29-32.

Klastorin, T. (2004). Project Management. Tools and Trade-offs. New York, NY, John Wiley & Sons, Inc.

Kulkarni, V. G. and V. G. Adlakha (1986). "Markov and Markov-Regenerative PERT Networks." Operations Research **34**(5): 769-781.

Lougee-Heimer, R. (2003). "The Common Optimization INterface for Operations Research." IBM Journal of Research and Development **47**(1): 57-66.

Lowe, T. J. and R. E. Wendell (2002). Managing Risks in Projects with Decision Technologies. Frontiers of Project Management. D. P. Slevin, D. I. Cleland and J. K. Pinto, Project Management Institute: 331-348.

Marascuilo, L. A. and M. McSweeney (1977). Nonparametric and distribution-free methods for the social sciences. Monterey, CA, Brooks/Cole.

Mastor, A. A. (1970). "An experimental and comparative evaluation of production line balancing techniques." Management Science **16**: 728-746.

Mitchell, G. (2005). Managing Risks in Complex Projects Using Compression Strategies. School of Business Administration, University of Washington. **Doctoral Dissertation**.

Schonberger, R. J. (1981). "Why projects are "always" late: a rationale based on manual simulation of a PERT/CPM network." Interfaces **11**(5): 66-70.

Shachter, R. D. (1986). "Evaluating Influence Diagrams." Operations Research **34**(6): 871-882.

Shachter, R. D. (1988). "Probabilistic Inference and Influence Diagrams." Operations Research **36**(4): 589-604.

Shr, J. F. and W. T. Chen (2003). "A method to determine minimum contract bids for incentive highway projects." International Journal of Project Management **21**: 601-615.

Tavares, L., A. Ferreira, et al. (1999). "The risk of delayof a project in terms of the morphology of its network." European Journal of Operational Research **119**: 510-537.

Tavares, L., A. Ferreira, et al. (2004). "A surrogate indicator of criticality for stochastic project networks." International Transactions in Operational Research **11**: 193-202.

Van Slyke, R. M. (1963). "Monte Carlo Methods and the PERT Problem." Operations Research **11**(5): 839-860.

Vanhoucke, M. (2008). Improved project time performance using activity sensitivity and network topology information. Gent, Belgium, Ghent University.

Vanhoucke, M., J. Coelho, et al. (2008). "An evaluation of the adequacy of project network generators with systematically sampled networks." European Journal of Operational Research **187**: 511-524.

Von Branconi, C. and C. H. Loch (2004). "Contracting for major projects: eight business levers for top management." International Journal of Project Management **22**(2): 119-130.

Ward, S. and C. Chapman (1994). "Choosing contractor payment terms." International Journal of Project Management **12**(4): 216-221.

Wollmer, R. D. (1985). "Critical Path Planning under Uncertainty." Mathematical Programming Study **25**: 164-171.