Abstract


The problem we investigate deals with the optimal assignment of resources to the activities of a stochastic project network. We seek to minimize the expected cost of the project, which we take as the sum of resource utilization costs and lateness costs, if the project is completed after a specified due date. These costs are both functions of the resource allocations to the activities with opposite responses to changes in allocation. We assume that the work content required by the activities follows an exponential distribution. An immediate result of this assumption is that the duration of the activities also follows an exponential distribution based on the degree of resource allocation. We construct a continuous time Markov chain (CTMC) model for the activity network and use the Phase-Type (PH-type) distribution to evaluate the project completion time.

Absence of an analytically tractable means of evaluating the sensitivity of the project cost to variation in the resource allocation to an individual activity led us to develop a derivative descent algorithm for the optimization of the expected cost of the project. We approximate the value of the derivative at a particular allocation by evaluating the differential cost of a slightly modified allocation. These quasi-derivatives led to the selection of an activity to which we optimize resource allocation. We use Fibonacci search over the interval of permissible allocations to the activity to seek the minimum expected cost. This iterative process of activity selection followed by changing the resource allocation is repeated until the expected cost is not significantly decreased. Finally, through extensive experimentation with a variety of projects of different structure and size, we show that this algorithm yields promising results in terms of both computation time and accuracy.
An Algorithm for Determining the Optimal Resource Allocation in Stochastic Activity Networks

by

Adam J. Rudolph

A thesis submitted to the Graduate Faculty of North Carolina State University
In partial fulfillment of the Requirements for the Degree of Master of Science

Operations Research

Raleigh, North Carolina
May 2008

APPROVED BY:

Dr. Subhashis Ghosal

Dr. Julie Ivy

Dr. Salah E. Elmaghraby
Chair of Advisory Committee
Biography

Adam Rudolph was born on December 5 of 1983 in Louisville, Kentucky, however most of his childhood was spent in Lafayette, Indiana. He enrolled at Purdue University in West Lafayette, Indiana in August of 2002 where he studied Industrial Engineering and was introduced to the field of Operations Research. While at Purdue, Adam had the opportunity to pursue internships with H.B. Maynard and Co., Caterpillar Inc., and GE Healthcare. He graduated from Purdue in May of 2006.

Adam moved to Raleigh, North Carolina and enrolled at North Carolina State University in August of 2006 to pursue a graduate education in Operations Research. He will graduate with a Master of Science in May of 2008.
Acknowledgements

I would like to express my thanks to Dr. Salah E. Elmaghraby, Dr. Julie Ivy, and Dr. Subhashis Ghosal for their hard work and dedication as my advisory committee. Through their time and help they have assisted me greatly in the completion of my thesis. I would also like to thank our department secretary Barbara Walls for her assistance with the many administrative details throughout my time at NCSU.

I must also thank Dr. Demeulemeester of Katholieke Universiteit Leuven for his provision of the RanGen and RanGen2 network generators.

Personally, I must thank my family, Ed and Sheila Rudolph and Sara and Jerod Patterson for their prayers and support throughout my formal education, graduate and otherwise.

There are so many others I could mention for countless simple acts of support that have made the completion of this graduate degree possible. Thank you to my many teachers, friends, and family who have been there for me throughout my life and especially thanks to God through whom all things, including graduate degrees, are possible.
# Table of Contents

List of Tables ........................................................................................................................................................................ vi

List of Figures ........................................................................................................................................................................... vii

Introduction ............................................................................................................................................................................. 1

Literature Review .................................................................................................................................................................... 3

  2.1 Project Scheduling and PERT networks .......................................................................................................................... 3
  2.2 Continuous Time Markov Chains ..................................................................................................................................... 7
  2.3 Relevant Recent Studies: .................................................................................................................................................... 8

Problem Statement .................................................................................................................................................................. 10

  3.1 Introductory Assumptions ................................................................................................................................................. 10
  3.2 Continuous Time Markov Chains .................................................................................................................................. 14
  3.3 Phase-Type Distributions .................................................................................................................................................. 18
  3.4 The Cost Function ............................................................................................................................................................ 21

Description of Algorithm .......................................................................................................................................................... 22

  4.1 Cost Computation ............................................................................................................................................................ 22
  4.2 Selecting Candidate Activities ........................................................................................................................................ 25
  4.3 Improving Computational Efficiency ................................................................................................................................... 27
  4.4 Optimizing The Allocation to a Single Variable .................................................................................................................. 28
  4.5 Stopping Criteria ............................................................................................................................................................... 31
  4.6 An Illustrative Example ...................................................................................................................................................... 32

Results ...................................................................................................................................................................................... 35

  5.1 Computational Results ....................................................................................................................................................... 35
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2</td>
<td>Network Structure Considerations</td>
<td>39</td>
</tr>
<tr>
<td>Conclusions and Future Research</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>6.1</td>
<td>Conclusions</td>
<td>44</td>
</tr>
<tr>
<td>6.2</td>
<td>Directions for Future Research</td>
<td>46</td>
</tr>
<tr>
<td>Bibliography</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Appendix A: Test Network Generation</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Appendix B: MATLAB Scripts</td>
<td>59</td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

Table 1: State Space of the CTMC.................................................................16
Table 2: Example Derivative Approximation........................................33
Table 3: Fibonacci Search Procedure Example .................................34
Table 4: Test Results .............................................................................36
Table 5: Example Network Test Results ..............................................43
Table 6: State Space of the AoN Example Network .........................57
Table 7: Example Network Generator Output .....................................58
List of Figures

Figure 1: An Example 3-Activity Network .................................................................10
Figure 2: The State Space of the Project of Figure 1: ..................................................17
Figure 3: Solution Time vs Number of States in CTMC...........................................38
Figure 4: Example Network 2 ....................................................................................41
Figure 5: Example Network 2 with Reduced Complexity .........................................41
Figure 6: Example Network 2 with Further Complexity Reduction..........................42
Figure 7: Series-Parallel Network .............................................................................42
Figure 8: Example Project in AoN Representation....................................................56
Chapter 1

Introduction

In many modern business, engineering, and industrial environments, reliable project scheduling tools are invaluable. Predictions of project completion dates and project costs strongly impact many business decisions. In turn, these decisions can impact the course of the coming weeks, and often years, for modern companies. As a result, it is important to have accurate predictions of project costs and completion times for management to make the best, most informed decisions.

Much of the literature pertaining to project scheduling problems ignores the simple fact that activity durations within a project framework are not known with certainty. This fact can strongly impact the validity of project planning tools in real life applications. Further, many project scheduling problems deal with ordering the various tasks of the project in such a way to meet some predefined objective. In reality, often a project manager has several resources at his or her disposal that can be used to complete a multitude of project tasks that can be undertaken at a particular point in time. Keeping this in mind, we seek to develop a model and create an algorithm that will find the optimal way to assign resources to the various stochastic activities of a project to minimize an economic objective function.

In the next chapter we discuss the literature that is available pertaining to project scheduling. We review several papers regarding classical project scheduling problems and Markov Chain representations of projects. We then discuss in more depth a few papers that are closely related to the research at hand. In Chapter 3 we formally state the problem under investigation. We discuss key assumptions and important preliminaries to our research and
arrive at a formal statement of our economic objective function. Chapter 4 contains a complete description of the optimizing algorithm. We pay special attention to a few simplifying assumptions that lighten the computational burden required and include illustrative examples to aid comprehension. In Chapter 5 we present the key results of our research as obtained from testing our algorithm on a set of test networks and discuss the computational burdens imposed by certain network structures. We conclude by discussing several possible extensions to the work presented in this thesis. In Appendix A, we briefly describe the methodology employed in the generation of test networks. Finally, in Appendix B we list the scripts used in MATLAB to optimize resource allocation to the activities of these test project networks. These scripts are detailed at length to assist in verifying our results and to provide a basis for extension to future research in any of the directions mentioned in the Conclusion.
Chapter 2

Literature Review

This review is categorized into three sections. First we briefly survey the literature within the realm of project scheduling and activity networks in both deterministic and stochastic settings. Second, we summarize the relevant concepts in continuous time Markov chains, Markov PERT Networks and the advantages afforded by their application. Finally, we discuss several recent research projects that have studied problems similar to our own.

2.1 Project Scheduling and PERT networks

A great deal of research has been done pertaining to project scheduling since the field’s inception. Classically, CPM (“critical path method”) and PERT (“program evaluation and review technique”) models have been the industry standard for some time both for their effectiveness and for their simplicity. In the traditional sense, however, neither model takes resource constraints into account as they opt to deal more with precedence relationships and time considerations. More recently, much effort has been spent researching the Resource Constrained Project Scheduling Problem (RCPSP) which we look into briefly before discussing the stochastic case.

The objective of the RCPSP studies is to schedule activities in such a manner to meet resource availability and precedence constraints in order to optimize an objective function typically related to the project completion time. The RCPSP can be subdivided into
unimodal and multimodal cases. The unimodal case involves preset durations and resource requirements for each activity, whereas the multimodal case allows activities to be processed in a number of different “modes” requiring different resource levels and completion times. Resource constraints can be classified as renewable or nonrenewable. Renewable resource constraints put limits on the amount of a particular resource that can be utilized at a particular time in the project. Non-renewable resource constraints limit the amount of a particular resource that can be utilized in total throughout the project’s life. Chapters 7 through 9 of Demeulemeester and Herroelen [2002] provide an excellent review of RCPSP problems and related optimization techniques.

The multimodal RCPSP is closely related to the problem at hand. These problems are often referred to as Time-Cost Tradeoff problems (TCTP). At its core, the TCTP assumes that the duration of an activity is a function of the resources assigned to its completion. Further, both rely heavily on the concept of “work content”. Normally, greater resource allocation will demand a greater expense, thus we have a tradeoff between the time to complete a project and its cost (a function of the resources assigned). When the task duration is a linear function of the assigned resources we have the Linear Time Cost Tradeoff (LTCT) problem. Notably, Fulkerson [1961] studied this problem under the objective of minimizing project cost subject to completion before a due date. He then solved the problem through the clever use of primal-dual concepts in linear programming and developed an approach to compute the complete cost curve of the project as a function of the project’s due dates.

If the resource’s allocation is limited to distinct values we have the Discrete Time-Cost Tradeoff (DTCT) problem. Hindelang and Muth [1979] developed a dynamic programming (DP) algorithm to solve this problem in pseudo-polynomial time when the
corresponding AoA project graph is series-parallel. In the case where the graph is not series-parallel De et al [1997] provide an update to the DP algorithm of Hindelang and Muth. In the same article, they show that the DTCT problem is, in general, strongly NP-Hard by reduction from 3SAT.

Deckro et al [1994] discuss the nonlinear case of the TCTP. They deviate from the traditional approach to the problem which would involve piecewise linearization and suggest a true nonlinear program. Presenting a quadratic program they suggest solution using commercially available software. The authors also investigate some extensions to the model including a goal-programming formulation.

When elements of the project are not known deterministically, analysis of the project network becomes much more difficult. The dearth of literature on the topic is perhaps good evidence of this fact. Even so, several studies have been performed both in the single and multiple mode fields.

Golenko-Ginzburg and Gonik [1997] investigate the RCPSP when faced with stochastic activity times. They use Beta, Normal, and Uniform distributions for activity times and determine a heuristically achieved best solution to the problem of minimizing the project's expected completion time. They schedule start times for tasks and proceed heuristically. The authors face the problem of deciding between “entering variables” (the activities to be initiated at a particular state of the project) by first using simulation to determine the probability that the activity is on the critical path and then solve a 0-1 Knapsack Problem.

In another paper, Stork [2001] takes an in-depth look at the RCPSP in the stochastic sense. He investigates the implications of AND/OR constraints to the problem. He also
briefly discusses the Linear Time-Cost Tradeoff problem and shows it is NP-hard when AND/OR constraints are present. He further investigates branch-and-bound, forbidden set, and priority based procedures for optimization in their relationship to the “robustness” of a solution.

Valls et al. [1998] study the RCPSP with some activities facing stochastic interruptions and processing times. They deal with the concept of Stochastic Programming versus Robust Optimization and state the burdens to computation imposed by multiple stages Stochastic Programming. Optimizing the weighted tardiness of either each activity or the total project, the authors propose and validate a meta-heuristic solution that blends Tabu Search and Scatter Search techniques.

Wollmer [1985] studies a version of the LTCT problem with an additional stochastic parameter. He deals with the problem where activity durations have a component that is a linear function of the allocation plus a random element. The author first seeks to find the minimum required investment subject to an expected project completion time and then solves the dual problem of finding the minimum expected completion time subject to a budget constraint. The author utilizes a cutting plane technique for his solutions.

Gutjahr et al. [2000] consider scheduling problems similar to our own. However, in their approach activities can have one of two possible distributions depending on whether or not a project is crashed. In this vein, our problem may be considered one in which projects can be "crashed" to varying degrees (i.e. depending on the allocation of resource to task). They are also under the discrete time assumptions but depending on the degree of discretization this assumption is overcome. Further, they define an “HS-Branch-and-Bound” algorithm that solves deterministic sub-problems using heuristic methods, and they have had
success using this approach as opposed to Total Enumeration, or pure Stochastic Branch-and-Bound. They suggest using different heuristics in different applications.

2.2 Continuous Time Markov Chains

In this thesis, we assume that the work content of any task is an exponentially distributed random variable. As a direct result of this assumption, activity durations are also exponential random variables since the duration is derived from the work content via the relation \( Y = W/x \), in which \( W \) is the (random) work content and \( x \) is the resource allocation. Further, when we assume independence of activity times with respect to one another we have a Markov PERT Network (MPN). Under this assumption the analysis of project networks is greatly aided by the use of Continuous Time Markov Chain (CTMC) theory. A good introduction to Continuous Time Markov Chains is provided in Ross [2002].

The term Markov PERT Network originated with Kulkarni and Adlakha [1986] and refers to stochastic activity networks where activity times follow an exponential distribution. Kulkarni and Adlakha [1986] describe Markov PERT Networks in great detail. Further, they provide a method for analysis involving uniformly directed cutsets (UDC’s) that allows one to transform the PERT network into a Continuous Time Markov Chain. An exposition of this process will be provided with appropriate definitions in section 3.2 below. The authors then develop recursive formulas for obtaining the moments of the completion time of a MPN. Though the use of these formulas is beneficial in certain applications (such as in bounding the completion time of the project), we instead favor the use of the Phase-type (PH-type) distribution concept in our analysis, as we are concerned not just with the moments of the
project completion time but also in its distribution based on the project parameters. We will develop the key concepts regarding this distribution in section 3.3.

2.3 Relevant Recent Studies:

A few studies have been conducted recently that are in a similar vein to the research considered here. However, they differ from our study in the objective, the optimization methodology, or the constraining set.

Tereso, Araújo, and Elmaghraby [2004] investigate optimal resource allocation under stochastic conditions. They optimize the same objective with which we are faced and relax the assumption that work content follows an exponential distribution. They solve the problem using a Dynamic Programming algorithm that is extremely taxing computationally. Their approach optimizes the allocation to resources along a single path through the network from start to finish while holding the allocations to other activities constant. Their solution necessitates enumerating the optimal path for all possible values of allocations to other variables. In conclusion they suggest treating only exponential distributions which gives rise to the research presented in this thesis.

Morgan [2006] derives a fast method for optimizing resource allocation in a single stage stochastic activity network. He justifies optimizing a single stage by suggesting that a project manager’s strategy should change once the state of the project changes as evidenced by the completion of an activity. Morgan uses Sample Path Optimization and appeals to Geometric Programming to solve the single stage problem. Repeatedly applying the single stage procedure he derives a methodology for solving the multiple stage project. Morgan states that his procedure could be applied to any distribution of work content provided it is
amenable to random sampling. Morgan’s approach also allows for absolute bounds on the resources to be applied.

Ramachandra [2006] studies the problem of optimal resource allocation in a Markov PERT Network with respect to an economic objective. His objective function is comprised of the expected cost of the resources and the cost of expected tardiness. Though the cost of expected tardiness is a major simplifying assumption when compared to the expected cost of tardiness, the author goes on to show that his approach to solution delivers an excellent approximation of the optimal solution. The author first describes a “Policy Iteration-like” approach to solution and then shows a Simulation-cum-Optimization approach to the problem. He uses Monte Carlo and Latin Hypercube Sampling approaches and shows the latter to have advantages with respect to variance minimization. The author further admits that any aggregate bounds on resource allocation would need to be applied in the search technique.

Finally, Azaron et al. [2007] use control theory to study resource allocation in Markov PERT Networks. The authors explain that the resource allocation costs and project tardiness costs are in conflict and therefore decide to model the problem as a multiple objective stochastic program. Ultimately the authors describe Goal Attainment and Goal Programming methodologies to solve the problem and provide computational results for a few examples. The authors state that their solution allows for activity times to be taken from any distribution as a generalized Erlang distribution. They also arrive at the conclusion that an optimal solution to the control problem cannot be found and so they resort to the discretization of time, modeling the problem with difference (rather than differential) equations and use non-linear programming to solve the problem.
Chapter 3

Problem Statement

3.1 Introductory Assumptions

We begin with a set of \( n \) activities that define the project. The project can be expressed as a graph \( G = (N, A) \) where either the nodes or the arcs signify the activities in a project. For this problem statement, we will consider the arcs to be representative of the activities and the nodes to be representative of “events” in the progression of the project (the activity on arc (AoA) mode of representation). The work on a task cannot commence until all its predecessors are completed. This is symbolized in the nodes of the project graph, wherein the arcs into a node are predecessors to the arcs out of the node. Further, we consider the project completed only when all tasks therein are completed. Figure 1, below, provides a simple example. Activity 2 cannot begin until activity 1 is completed. Also, we consider the project completed only when all three activities have been completed; i.e. Node 3 is realized.

Figure 1: An Example 3-Activity Network
We are given for each activity $j$ its work content, $W_j$, representing the amount of work (effort, energy) to be done on the activity. We assume that $W_j$ is known in probabilistic terms (non-deterministic). It is important to note that we consider the source of uncertainty to be internal to the project. External uncertainty, such as that due to machine failure or weather, is beyond the scope of this research. In general, work content is measured in some convenient unit depending on the activity and the resource; such as “worker-hours” or “computer-seconds”, etc. Consequently, only when the resource level is specified can we estimate the time to complete the activity. This relationship depends on three entities: work content, resource level, and activity duration. If a simple hyperbolic relationship is assumed then the activity duration is equal to the work content divided by the resource level, as we shall show in equation (2). For instance, if the activity requires 36 worker-hours and 3 workers are assigned to it, it shall consume 12 hours to accomplish. However, if 4 workers are assigned to it, it shall require only 9 hours, etc. We are also given a due date for the project, $T_s$, which is specified by the project “owner” and is known with certainty.

We let $x_j$ represent the allocation of the resource to activity $j$. It is important to note that the value of $x_j$ is our decision variable while $W_j$ is a random variable; thus we seek to find the best values of the $x_j$ variables that optimize the objective function. We consider only one resource of unlimited availability, although in application the particular resource may vary from activity to activity within a project; i.e. one activity may require electricians as a resource while another may require bricklayers but no activity requires more than one (critical) resource at a time. This could be accommodated in the proposed model by replicating the activity arc in parallel for each resource required for the represented activity with the same source and terminal nodes.
Costs incurred in the execution of the project are of two forms: resource costs and lateness costs. We assume a constant marginal cost for the resource, $c_R$, and also assume that this cost is constant across all activities. The latter assumption is made for ease of computation although it could be relaxed easily in the algorithm detailed in the next section. We are also given a cost of lateness for the project, $c_L$ per unit time of delay. In our application, we normalize these costs by dividing through by $c_R$. Again, this assumption serves to simplify the computations; in any application more realistic costs would be known and could be used (such as piece-wise linear and convex costs). If we let $Y$ denote the time of project completion, a random variable, the cost of lateness is defined as $c_L(\max\{0, Y - T_s\})$. Finally, we seek to find an allocation vector, $X$, minimizing the sum of the expected value of these two costs.

The first major assumption we make is the distribution function of each activity’s work content, $W_j$. Because of its tractability and ease of use, we assume that $W_j$ follows an exponential distribution with parameter $\lambda_j$:

$$W_j \sim \exp(\lambda_j), \forall j \in A$$ (1)

As we will show, this assumption will greatly decrease our computational burden because of the exponential distribution’s memory-less property. Further, we assume that the resource allocation, $x_j$, is bounded from below by $l_j$ and from above by $u_j$:

$$0 < l_j \leq x_j \leq u_j < \infty, \forall j \in A$$

Note that this assumption does not conflict with the earlier assumption of unlimited resource availability – it only constrains the resource allocation to any individual activity. We assume the total resource availability is abundant enough to accommodate all activities running in
parallel at a given time. Given our assumptions of exponential work content and bounded resource allocations we let \( y_j \) denote the duration of activity \( j \):

\[
y_j = \frac{w_j}{x_j}
\]  

As \( w_j \) is exponentially distributed and \( x_j \) is a decision variable, it can be easily demonstrated that \( y_j \) also follows an exponential distribution with parameter \( x_j/\lambda_j \). Note that this definition necessitates our bounding of \( x_j \); without bounds on \( x_j \), the duration of activity \( j \) will not necessarily take on a meaningful value. While the assumption that the duration of an activity takes on an exponential distribution may seem unreasonable, such a choice can be justified on three grounds.

First, the memory-less property of the exponential distribution aids greatly in computation and, as we will show in section 3.2, lends well to the use of continuous time Markov chain (CTMC) theory for its analysis. Second, and perhaps more importantly, the exponential distribution is the limit of the class of distributions known as “new better than used in expectation” (NBUE) distributions, to which all proposed distributions of activity durations known to us belong. That is to say, if a task has not completed after \( t \) time units, the expectation that the task will take \( 1/x_{j}/\lambda_j \) additional units provides an upper bound on the expected time the task should take in actuality. Thus this assumption ensures that we provide an upper bound on the expected project duration and cost. Lastly, using the exponential distribution and modeling the project as a CTMC can be considered the fundamental “building block” which serves other potential work content distributions. This is due to the fact that any continuous distribution can be approximated to any desired degree of accuracy by the generalized Erlang distribution (GED). The GED is a “phase distribution” in which
each “phase” is exponentially distributed – and we are back to the CTMC model based on these exponential distributions.

Finally, we assume that the cost of resource allocation to activity $j$ is quadratic in the allocation over the duration of the activity. This reflects the added cost of employing additional resources. It also could be considered representative of the law of diminishing marginal returns. It also has the added advantage of resulting in a linear function of the allocation $x_j$. Thus the cost of resource allocation to activity $j$ is defined as:

$$C_j = c \alpha_j Y_j = c \alpha_j W_j.$$  \hspace{1cm} (3)

Therefore, the expected cost due to resource allocation to all the project activities is given by:

$$E[C_R] = c_a \sum_{a \in A \setminus \lambda_a} \left( \frac{x_a}{\lambda_a} \right).$$  \hspace{1cm} (4)

In order to properly evaluate the portion of the project cost due to lateness we must first establish the distribution of the project completion time. To do so, we will first apply the principles established by Kulkarni and Adlakha [1986] to develop a continuous time Markov chain representative of the project network. Next we present the known results from the Phase-Type Distributions in order to evaluate the density function of the project distribution.

### 3.2 Continuous Time Markov Chains

Kulkarni and Adlakha [1986] developed a methodology for transforming a project network with exponentially distributed activity completion times into a CTMC. We will now describe their notation and methodology.
We begin with a graph $G = (N, A)$ representing the project in the AoA mode of representation as previously stated. Without loss of generality, we assume that the project begins at time zero and will end at a randomly distributed later time, $\Upsilon$, once all of the activities have been completed. At a particular time, $t$, between 0 and $\Upsilon$, each activity in $A$ can be in one of three states:

**Active:** An activity is *active* if it is in process at time $t$.

**Dormant:** An activity is *dormant* if it is completed but at least one other activity ending at the same node is still *active* at time $t$.

**Idle:** An activity is *idle* if it is neither *active* nor *dormant*. This includes all activities that are yet-to-be-initiated and those that are completed along with all other activities ending at the same node.

Clearly, a project is completed when it has no *active* or *dormant* activities. Since each activity is in one of these three states Kulkarni and Adlakha [1986] define the following sets for any time $t \geq 0$:

$$Y(t) = \{a \in A: \text{a is active at time } t\},$$

$$Z(t) = \{a \in A: \text{a is dormant at time } t\},$$

and the state of the project is given by:

$$X(t) = (Y(t), Z(t)).$$

The set of idle activities is the complement of the set $X(t)$ in $A$. Before proceeding further we must define a *cutset*. Given a project graph $G = (N, A)$ in AoA representation, let $B$ be a proper, nonempty subset of the set of nodes, $N$, and $\overline{B}$ be its complement in $N$. Further, we define $(B, \overline{B}) = \{j \in A: i \in B, i' \in \overline{B}\}$ where $i$ and $i'$ denote the start and end nodes of activity $j$. The set of edges $(B, \overline{B})$ is defined as an *(s-t)* cut or *cutset* if $s \in B$ and $t \in \overline{B}$, where $s$
denotes the start node of a project (node 1) and \( n \) denotes the terminal node of the project, \( n = |N| \). Further, the cutset is a uniformly directed cutset (UDC) if no arcs extend from \( \overline{B} \) into \( B \), symbolically, \( (\overline{B}, B) = \emptyset \). The project in Figure 1 has 2 UDC’s, both of which are labeled.

As stated by Kulkarni and Adlakha [1986], \( \{X(t), t \geq 0\} \) is a representation of the state of the project which forms a continuous time Markov chain (CTMC) with a single absorbing state when mutual independence of the activities is assumed. This stems from the fact that \( X(t) \) forms a 2-partition of a UDC for all times \( t \). If we define \( S \) as the set of all admissible 2-partitions of UDCs in \( G \), then clearly \( X(t) \in S \). When \( t \geq Y \), the sets \( Y(t) \) and \( Z(t) \) are both empty, thus \( X(t) = (\emptyset, \emptyset) \) and \( \overline{S} = S \cup (\emptyset, \emptyset) \) forms the state space for the CTMC. Generally, \( \overline{S} \) is the set of all possible combinations of active and dormant activities of the project that can occur according to the precedence constraints, extended by the empty set, \((\emptyset, \emptyset)\). For the project in Figure 1, \( \overline{S} \) contains 6 states, which are enumerated in Table 1. A graphical representation of the CTMC is given in Figure 2:

**Table 1: State Space of the CTMC**

<table>
<thead>
<tr>
<th>State</th>
<th>Activity 1</th>
<th>Activity 2</th>
<th>Activity 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Active</td>
<td>Idle</td>
<td>active</td>
</tr>
<tr>
<td>1</td>
<td>Idle</td>
<td>Active</td>
<td>active</td>
</tr>
<tr>
<td>2</td>
<td>Active</td>
<td>Idle</td>
<td>dormant</td>
</tr>
<tr>
<td>3</td>
<td>Idle</td>
<td>Active</td>
<td>dormant</td>
</tr>
<tr>
<td>4</td>
<td>Idle</td>
<td>Dormant</td>
<td>active</td>
</tr>
<tr>
<td>5</td>
<td>Idle</td>
<td>Idle</td>
<td>idle</td>
</tr>
</tbody>
</table>
A few observations made by Kulkarni and Adlakha [1986] are worthy of note due to their importance and relevance to our research.

- Transitions in the CTMC correspond to the completion of activities in the project and their associated rates correspond to the parameters of the completion time distribution. This directly implies that in our research, the transition rates within the CTMC are random variables of rates \( \{x_j \lambda_j\} \).

- The state \((\varnothing, \varnothing)\) corresponds to the completion of the project. With no activities left to initiate or complete, no transitions are possible from this state and so \((\varnothing, \varnothing)\) is the single absorbing state of the CTMC.

- Since each activity is completed exactly once, each state of the CTMC may be visited only once. Thus, the states of \( S \) are transient and the states of \( \bar{S} \) can be numbered so that the rate transition matrix is upper-triangular: \( |S| = m \) and \( |\bar{S}| = m + 1 \).
• The number of states in \( \{X(t), t \geq 0\} \) is finite for projects with a finite number of activities. Therefore, \( \{X(t), t \geq 0\} \) will be completed in finite time with probability 1.

• Finally, if \(|A| = a\), the number of states in the CTMC follows:

\[
a \leq |\bar{S}| \leq 2^a
\]

with the lower bound occurring when all activities of the project are in series and the upper bound occurring when all activities are in parallel.

3.3 Phase-Type Distributions

In order to establish the expected cost of lateness, we must first describe the distribution of time until the project’s completion. Employing the CTMC interpretation of the model previously described the project can be considered completed when the Markov process reaches its absorbing state. When an absorbing CTMC has a single absorbing state the distribution of time until absorption can be described by the phase-type (PH-type) distribution. The phase-type distribution gets its name from the path taken by the project from initiation to completion, which can be viewed as a series of stages, or “phases,” each having an exponential distribution.

The parameters of the phase-type distribution are a probability vector, \((\alpha, \alpha_{m+1})\), and the CTMC’s rate generator matrix:

\[
Q = \begin{bmatrix}
T_{(m \times m)} & T^0_{(m \times 1)} \\
\theta_{(1 \times m)} & 0_{(1 \times 1)}
\end{bmatrix}.
\]
Here, the $T$ matrix is representative of the transient states of the matrix, with $T_{ii} < 0$ for $1 \leq i \leq m$ and $T_{ij} \geq 0$ for $i \neq j$. Additionally:

$$T^0 + Te = \theta,$$

(7)

where $e$ represents a column vector of dimension $(1 \times m)$ and $\theta$ is a vector of zeros of equivalent dimension; i.e. the rows of the matrix $Q$ sum to 0. As an example, the $Q$ matrix for the project in Figure 1:

$$Q = \begin{bmatrix}
-0.27 & 0.20 & 0.07 & 0 & 0 & 0 \\
0 & -0.17 & 0 & 0.07 & 0.10 & 0 \\
0 & 0 & -0.2 & 0.20 & 0 & 0 \\
0 & 0 & 0 & -0.1 & 0 & 0.10 \\
0 & 0 & 0 & 0 & -0.07 & 0.07 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

The vector $\alpha$ is an initial probability vector, also called the “counting probability” of $Q$. This vector represents the probability that the Markov process begins in a given state. As such, we have:

$$\alpha \cdot e + \alpha_{m+1} = 1.$$

(8)

For our purposes, we assume the project begins with no activities completed; i.e. the project is in the first state with probability 1. Thus we have $\alpha = [1 \ 0_{(1 \times m)}]$.

In general, the PH-type distribution, hereby referred to as $F(\cdot)$, can be represented by $(\alpha, T)$. A necessary and sufficient condition for the states of the CTMC represented by $T$ to be transient is that the matrix $T$ is nonsingular. As a result, $T^k \to 0$ as $k \to \infty$. When given an initial probability vector $(\alpha, \alpha_{m+1})$, the cumulative distribution function of the time to absorption in state $m+1$ is given by:

$$F(t) = 1 - \alpha \cdot e^{T^t} \cdot e \text{ for } t \geq 0.$$

(9)
The evaluation of $e^{T_i}$ has been studied in depth. Traditionally, it is given by the Taylor series:

$$e^{T_i} = \sum_{i=0}^{\infty} \frac{1}{i!}(T_i)^i,$$

however, it can be calculated in a myriad of other ways as demonstrated by Moler and van Loan [2003].

A few observations about the PH-type distribution are now necessary for our research:

- The function $F(\cdot)$ has a jump of height $\alpha_{m+1}$ at $t = 0$. This is equivalent to the probability that the process begins in the absorbing state. For our research, this is somewhat irrelevant because we assume the project starts with no activities completed. In practice, this assumption could be relaxed in the interest of managerial flexibility; i.e. after a certain amount of time, the manager could re-optimize his or her resource allocation according to the progress of the project at that time.

- The probability density function (pdf) $f(\cdot)$ is given by:

$$f(t) = \frac{d}{dt} F(t) = \alpha \cdot e^{T_i} \cdot T^0.$$  

(11)

- The expected value and variance of the PH-type distribution, respectively, are given by:

$$-\alpha \cdot T^1 \cdot e$$

and

$$2\alpha \cdot T^2 \cdot e$$

(12)

and, in general, the non-central moments $\phi_i$ of $F(\cdot)$ are given by:

$$\phi_i = (-1)^i \cdot i! \cdot (\alpha \cdot T^i \cdot e),$$

for $i \geq 0$.  

(13)
3.4 The Cost Function

The PH-type distribution can be used to formally state the cost function over which we seek to optimize. As of the moment, we have in hand the expected cost of resource allocation, given by (4) above; it remains to derive the portion pertaining to the expected cost of lateness.

First, as previously stated, the cost of lateness is defined by \( C_L(Y) = c_L(\max\{0, Y - T_s\}) \). With the distribution of \( Y \) following the PH-type distribution we must use integration to calculate the expected cost of lateness:

\[
\mathbb{E}[C_L] = \int_0^\infty C_L(t) \cdot f(t) \, dt
= \int_0^\infty C_L(t) \cdot \alpha \cdot e^{T_\alpha} \cdot T^0 \, dt
= \int_{T_\alpha}^\infty (t-T_\alpha) \cdot \alpha \cdot e^{T_\alpha} \cdot T^0 \, dt
\]

It is important here to remember that the entries of the matrix \( T \), as well as \( T^0 \), are determined by the completion times of the activities in \( A \). Under our assumptions these completion times are functions of the resource allocations to the activities in the project. Thus, the expected cost of lateness is also a function of the vector \( X \). In total, we seek to find the optimal resource allocation \( X \) to minimize the expected project cost:

\[
\mathbb{E}[C(X)] = \sum_{a \in A} \left( \frac{X_a}{2} \right) + c_L \cdot \int_{T_\alpha}^\infty (t-T_\alpha) \cdot \alpha \cdot e^{T_\alpha} \cdot T^0 \, dt
\]

subject to the bounds on resource allocation as well as the precedence constraints expressed as nodes in \( G \). Here, the value \( c_R \) has dropped out of the equation as it has been set equal to 1 and normalized with respect to \( c_L \).
Chapter 4

Description of Algorithm

In order to solve the problem at hand, we focus our research on a modified derivative descent algorithm. We first compute an approximation to the partial derivative of the expected cost with respect to the allocation to each of the activities and then choose to modify the allocation to the activity that has the greatest negative derivative. Next we use line search techniques to find the minimum expected cost allocation to the chosen activity while holding the other allocations constant. The algorithm iterates using these two steps until a specified stopping criterion is met. In order to execute any of these steps, we first must be able to compute the expected cost at a particular allocation.

4.1 Cost Computation

As previously described, the expected cost of the project (given in (15) and repeated here for convenience) contains two terms:

\[
\mathcal{E}[C(X)] = \sum_{a \in A} \left( \frac{X_a}{\lambda_a} \right) + c_i \cdot \int_{T_i}^{\infty} \alpha \cdot e^{T_i} \cdot T^0 dt
\]

The first term relates exclusively to the level of resource allocation. Since the work content distribution parameter, \( \lambda_a \), is given for each activity the expected cost of resource allocation can be computed directly. The second term, which relates to the lateness of the project with respect to a given due date, \( T_i \), cannot be computed directly for the following reason. Due to the complexity of the integrand and the need to evaluate \( e^{T_i} \), we must settle upon a
summation to approximate the expected cost of lateness. Applying a summation requires assuming time takes on discrete values. Azaron et al. [2007] arrive at the same conclusion through the use of control theory. They take the approach of dividing the “time interval” (that is, the interval of time to project completion) into $K$ portions of length $\Delta t$ and solve a system of difference equations. However, we choose to depart from this approach since the “time interval” is not known. For example, a trivial project involving only a single activity will have a significantly shorter “time interval” when allocating 3 units of the resource than when allocating just 1 unit. Instead, we set the length of $\Delta t$ and take the sum of lateness costs until the probability that the project takes longer than $m \cdot \Delta t$ time units is sufficiently small, say $p_{\text{lim}}$. Since the expectation of cost involves computing the probability that the project completes within a certain time interval, this method of calculation requires no additional computational steps.

We, therefore, take a midpoint Riemann sum of the product to approximate the value of the integral. For suitably small values of $\Delta t$ this method provides an excellent approximation. The downside, however, is that while decreasing $\Delta t$ to increase the precision of the approximation we also increase the computational burden. We settle this tradeoff by letting $\Delta t$ equal a fraction of the standard deviation of the project completion time provided it is greater than a lower bound, $I_{\Delta t}$, as this helps to normalize the computational time for each instance in which the expected project cost is computed.

The computational burden required by the cost calculation derives from the calculation of the probability that a project completes in a specified window of time. Recall from (9) that $F(\cdot)$ requires $e^{Tr}$ to be computed. Since the $T$ matrix is not changed during the
calculation of the expected cost, much can be gained if we let \( R = e^T \). Equation (9) then becomes:

\[
F(t) = 1 - \alpha \cdot R^t \cdot e \quad \text{for } t \geq 0. 
\] (16)

This computation requires computing \( R^t \) rather than \( e^{Tt} \).

The algorithm for computing the cost then proceeds as follows:

Step 1: Initialization: Let \( t = T_s \), let \( \Delta t = \max \{ z \cdot \sigma_{tn}, I_{st} \} \) where \( t_n \) denotes the project completion time and \( z \) is a predefined fraction, let \( C_P = 0 \), let \( p_0 \) denote the probability that the project completes before time \( T_s \), i.e. let \( p_0 = F(T_s) \), let \( R_{mat} = R^t \), and let \( U = R^\Delta t \).

Step 2: Calculate the probability, \( p \), that \( t_n \) falls in the interval of \([t, t + \Delta t] \):

\[
p_1 = F(t + \Delta t) = 1 - \alpha \cdot R_{mat} \cdot U \cdot e
\]

\[
p = p_1 - p_0
\] (17) (18)

Step 3: Let \( C_P = C_P + c_L \cdot p \cdot ((2t + \Delta t)/2 - T_s) \). If \( p_1 \) exceeds \( 1 - p_{lim} \) then go to Step 4.

Otherwise let \( p_0 = p_1 \), let \( t = t + \Delta t \), let \( R_{mat} = R_{mat} \cdot U \), and repeat from Step 2.

Step 4: Add the expected cost of resource usage to the expected lateness cost, \( C_P \), using (4) and stop. \( C_P \) now equals the expected total project cost.

In the algorithm, we use \( U \) and \( R_{mat} \) in order to eliminate the need to calculate the power of an exponential in each step. Instead, we calculate these quantities once (from knowledge of the \( W_j \)’s and the resource allocation) and can then rely on matrix multiplication, which is significantly less burdensome computationally than matrix exponentiation.

With an effective way to calculate the total expected project cost in hand, we can now describe the optimizing procedure we employ.
4.2 Selecting Candidate Activities

In general, our algorithm optimizes the project cost by changing the allocation to one activity at-a-time. We now need to explain the procedure to select activities that are the best candidates for optimization. The best candidates in our procedure are those that could lead to the greatest decrease in expected project cost.

The two terms of the project cost behave diametrically opposite in their response to changing resource allocations. The project resource cost increases linearly with respect to an increase in resource allocation to any activity. On the other hand, increased resource allocation to the activities tends to shorten their expected duration and so the expected lateness cost would decrease with respect to such a change in allocation. A decrease in overall cost, therefore, could be obtained from either a decrease or an increase in resource allocation.

An important observation is that the expected project cost is a convex function with respect to the allocation to a single activity. This fact is a result of the two convex components of cost. The expected cost of lateness decreases convexly due to the exponential distributions on which it is based. Further, since the resource costs are linear (hence convex) in the resource allocation, the sum of the two costs is convex. Thus, repeatedly optimizing allocations to single activities one-at-a-time will descend monotonically to reach the optimal solution.

Since no analytical expression is known for the partial derivative of the cost function with respect to the allocation to any activity $a_j$, we must rely on approximations to proceed. Such an approximation would enable us to select the activities to which a change in allocation could cause the greatest change in expected cost. If we compute the cost
associated with a small change in allocation, \( \delta \), to a particular activity we can approximate
the derivative of the cost function by first taking the difference between this cost and the cost
of the initial resource allocation and then dividing by the magnitude of \( \delta \). Since a decrease in
cost can occur via an increase or a decrease in allocation, and thanks to the convexity of the
cost function, the best candidate allocations are those with the steepest derivatives causing a
decrease in cost. Further, if \( \delta \) is constant across all activities, we can simply find the change
in allocation reflecting the greatest decrease in cost, as we are not concerned with the actual
value of the derivatives, but rather their magnitudes with respect to one another.

Letting \( X_{+a} = X + \delta \cdot e_a \), where \( e_a \) is a vector of dimension \(|A|\) with zeros everywhere
except in position \( a \), denote the allocation \( X \) with the value of the \( a \)th allocation increased by
the amount \( \delta \), further we allow \( X_{-a} = X - \delta \cdot e_a \) to denote the allocation with the \( a \)th allocation
decreased. We calculate \( E[C(X_{+a})] \) and \( E[C(X_{-a})] \) for each activity. With \( E[C(X)] \) in hand,
we find \( E[C(X_{+a})] - E[C(X)] \) and \( E[C(X_{-a})] - E[C(X)] \) for each activity, \( a \), and use this as a
surrogate for the value of the derivatives of the cost function with respect to the allocation to \( a \). We select the activity corresponding to the greatest decrease in cost as the candidate
activity. If the decrease in cost comes from \( X_{+a} \), the allocation to activity \( a \) must be increased.
Likewise, if this comes from \( X_{-a} \) we must decrease the allocation to activity \( a \). Since we are
dealing with minimization of the cost function, for certain network structures both increasing
and decreasing the allocation can cause an increase in cost. Thus it is important to check
both an increase and decrease in allocation rather than assuming that an increase in cost in
one direction implies a decrease in cost in the opposite.
4.3 Improving Computational Efficiency

As with computing the cost function, approximating its derivative can be computationally burdensome, considering that we must calculate $E[C(X_{na})]$ and $E[C(X_{\lambda a})]$ for each activity. We let $Q(X_{na})$ represent the matrix $Q$ with the allocation to activity $a$ increased by the value $\delta$. Note that increasing the allocation to a single activity may correspond to increasing the value of several entries in the original $Q$ matrix, as well as the decreasing their corresponding elements along the main diagonal. Changing the allocation to activity $i$ by $\delta$ implies that we must change the values in the $Q$ matrix by $\delta \lambda_i$. This change can be viewed as matrix addition. These facts are best illustrated with an example from the project in Figure 1, where we change the allocation to activity 1 ($\lambda_i = 0.2$) with $\delta = 0.1$:

$$Q(X_{+i}) = \begin{bmatrix}-0.27 - 0.02 & 0.20 + 0.02 & 0.07 & 0 & 0 & 0 \\ 0 & -0.17 & 0 & 0.07 & 0.10 & 0 \\ 0 & 0 & -0.20 - 0.02 & 0.20 + 0.02 & 0 & 0 \\ 0 & 0 & 0 & -0.1 & 0 & 0.10 \\ 0 & 0 & 0 & 0 & -0.07 & 0.07 \\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

$$Q(X_{+i}) = Q + \begin{bmatrix}-0.02 & 0.02 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.02 & 0.02 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

$$Q(X_{+i}) = Q + \Phi_{+1}$$

where, $\Phi_{+a}$ represents the matrix corresponding to the increase in allocation to activity $a$.

Note that the matrix $\Phi_{+a}$ does not depend on the current allocation to the other activities.
Recall from elementary matrix algebra that $e^{A+B} = e^A \cdot e^B$ when $A$ and $B$ are commutative: $A \cdot B = B \cdot A$. Unfortunately for our purposes, $Q \cdot \Phi_a \neq \Phi_a \cdot Q$. However, the sparse nature of the $\Phi_a$ matrix implies that these two terms are approximately equal. Further, $e^{\Phi_a}$ is nearly equal to the identity matrix, $I$. With these facts in mind, we can use $e^{Q+\Phi_a} = e^Q \cdot e^{\Phi_a}$ or rather the transient matrices $e^{T+\phi_a} = e^T \cdot e^{\phi_a}$, where $\phi_a$ denotes the transient portion of $\Phi_a$, for use as an approximation in the density function of the completion time distribution. This approximation provides a good indicator of the derivative of the cost function with respect to direction (as in whether increasing the allocation leads to an increase or a decrease in cost) as well as the magnitude of the change in cost. We will validate this approximation in the next section by comparing the expected costs returned by the algorithm when using the approximation and the exact determinations of the cost derivatives. Note that this analysis can be applied to the computation of $E[C(X_a)]$.

The key advantage this approximation provides is that it eliminates the need to calculate the matrix exponential for the evaluation of the derivative at each iteration. Since $\Phi_a$ does not depend on the current allocation, the $e^{\phi_a}$ matrices can be computed once at the beginning of the algorithm and stored for use throughout the optimization process. Since $e^T$ is needed for calculating the cost at each iteration this matrix is in hand as well. Thus, $e^T \cdot e^{\phi_a}$ is computed and passed to the cost calculation algorithm as the matrix, $R$.

### 4.4 Optimizing The Allocation to a Single Variable

With a candidate variable in hand, we now seek to optimize the allocation of resources to that activity while leaving the other allocations unchanged. As previously
stated, the expected project cost is a convex function with respect to the allocation to a single activity. Any convex optimization procedure could therefore be applicable here. However, due to the difficulty involved in finding exact analytical expressions for the partial derivatives of the cost function, we opt to use Fibonacci search as our method of determining the optimal allocation to the selected activity. Note that this “optimal” allocation is “locally optimal” in a sense as it depends upon the allocation to the other activities.

At its core, Fibonacci search, often referred to as “golden mean” search, finds the optimal point in a range of feasible values by repetitively shrinking the range, stopping when the range is sufficiently small to suggest a single optimal point. Fibonacci search makes use of the so-called “golden ratio,” which Wilde [1964] shows is the most computationally efficient method of finding the optimal value of a variable when searching along a line. Given a lower bound, \( l \), and an upper bound, \( u \), on the range, two new points are calculated within the range: \( m_l \) and \( m_u \) where \( m_l < m_u \). We define \( r \) as the inverse of the “golden ratio” and use \( r \) to give values for \( m_l \) and \( m_u \):

\[
\begin{align*}
\frac{2}{1 + \sqrt{5}} & \approx 0.618 \\
m_l = l \cdot r + u \cdot (1 - r) \\
m_u = l \cdot (1 - r) + u \cdot r.
\end{align*}
\]

For our purposes, these points represent different resource allocations. If activity \( j \) was selected as the candidate activity and its cost decreases with increased allocation we let \( l \) be the current allocation and let \( u \) be \( u_j \) for activity \( j \) while keeping the allocations to the other activities unchanged. If its cost decreases with decreased allocation, we define the upper bound as the current allocation and let \( l \) be the lower bound on the allocation to activity \( j \).

For example, suppose the current allocation in the project given in Figure 1 is \( X = (2, 2, 2) \)
with lower and upper bounds defined at 1 and 3 respectively. If activity 1 is selected as the candidate activity and its cost decreases with increased allocation we let \( l = X \) (the current allocation) and \( u = (3, 2, 2) \). Conversely, if the cost increases with increased allocation, let \( l = (1, 2, 2) \) and \( u = X \). Values for \( m_l \) and \( m_u \) are then defined by (20) and (21) respectively.

With these four values in hand \((l, u, m_l, \text{and } m_u)\), we recursively redefine the bounds on the range to hone our search to the minimum expected cost allocation. Optimization proceeds by first calculating the expected costs of allocations \( m_l \) and \( m_u \). The convexity of the expected cost function allows us to conclude that the minimum falls between the greater of the two values and the opposite absolute bound; i.e. if \( E[C(m_l)] < E[C(m_u)] \) then the optimal value must lie between \( l \) and \( m_u \), otherwise \( E[C(m_l)] > E[C(m_u)] \) and the optimal value must lie between \( m_l \) and \( u \). If the two expected costs are equal, either bound can be used. Using these values we redefine the bounds on optimization using the absolute bound nearer \( m_l \) or \( m_u \) corresponding the lower cost and redefine the opposite bound as \( m_l \) or \( m_u \) corresponding the greater expected cost. The two points within the range are then redefined; however, the use of the “golden ratio” permits calculation of only one of these points. The \( m_l \) or \( m_u \) corresponding to the lower expected cost will take the value of the opposite midpoint in the next iteration, as the use of equation (20) or (21) will yield this value.

Further, only one new expected cost must be calculated in each iteration. As an example, suppose \( E[C(m_l)] < E[C(m_u)] \). In the next iteration, \( l \) will remain unchanged, \( u \) will take the previous value of \( m_u \), \( m_u \) will take the previous value of \( m_l \), and a new \( m_l \) can be calculated using (20). These concepts are illustrated numerically in Section 4.6 below.

In each iteration, we shrink the search range by the ratio \( r \) as defined in (19), approximately 61.8%. The optimizing procedure concludes when an appropriately small
interval exists between \( l \) and \( u \); i.e. \( u - l < \varepsilon_d \) for some small \( \varepsilon_d \). When this is reached, any allocation within the interval \([l, u]\) could be taken as the optimal solution, however, we elect to use the midpoint of \( m_l \) and \( m_u \). In practice, any value for \( r \), given \( 0.5 < r < 1 \) could be used, however, any \( r \) other than that specified in (19) will require computing new values for both \( m_l \) and \( m_u \) in each iteration.

### 4.5 Stopping Criteria

The optimizing algorithm repeats the derivative approximation and Fibonacci search steps, finding a resource allocation with a lower expected cost in each iteration. The algorithm stops when the expected cost in a given iteration improved the expected cost by only a small amount. If \( X_k \) represents the best resource allocation after iteration \( k \), the algorithm terminates when \( E[C(X_{k-1})] - E[C(X_k)] < \varepsilon_p \).

Another potential stopping criterion would be when the approximated derivatives are all sufficiently close to 0. This criterion, however, would require a more exact calculation of derivatives, as currently we are concerned only with finding the derivative causing the greatest decrease in cost. Further, this criterion requires all derivatives to be less than the pre-defined \( \varepsilon_p \). Assume, without loss of generality, that the last iteration increased the allocation to activity \( a \) and caused a decrease in expected cost less than \( \varepsilon_p \). This implies that \( E[C(X_{n_a})] - E[C(X_{k-1})] \) (the approximated derivative in the previous iteration) had the minimum value across all activities. If the difference in allocation after the Fibonacci search procedure is less than \( \delta \), for a sufficiently small \( \delta \), the current allocation is effectively unchanged, therefore, further iteration will not improve the solution. Further, if the new
allocation to activity $a$, after the Fibonacci search procedure, differs from the previous allocation by more than $\delta$, then $E[C(X_{k-1})] - E[C(X_{a\alpha})] < E[C(X_{k-1})] - E[C(X_2)]$ implying that all derivatives are less than $\epsilon_p$. In other words, if we move in the direction of the steepest derivative and end with an effectively different allocation yet the expected cost decreased by less than $\epsilon_p$, then $E[C(X_0)]$ is less than $E[C(X_{a\alpha})]$ from the previous iteration. This implies that all the approximate derivatives at the previous allocation were less than $\epsilon_p$, since the steepest derivative did not cause a change of more than $\epsilon_p$. Convexity of the cost function implies that this will hold true at the new allocation as well.

### 4.6 An Illustrative Example

Consider the project represented by the graph in Figure 1. We assume an initial allocation of $X_0 = (1, 1, 1)$ with lower and upper bounds on allocation defined at 1 and 3 respectively. In our calculations we let $\delta = 0.05$, $c_L = 3$, $T_s = 8$, and $\epsilon_p = .005$. First we compute the expected cost of the initial allocation: $E[C(X_0)] = 69.8278$. Next, we approximate the derivative of the expected cost with respect to the allocations to activities 1, 2, and 3:
Table 2: Example Derivative Approximation

<table>
<thead>
<tr>
<th>Activity</th>
<th>$X_a$</th>
<th>$E[C(X_a)]$</th>
<th>$E[C(X_a)] - E[C(X)]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, increase</td>
<td>(1.05, 1, 1)</td>
<td>69.3474</td>
<td>-0.5006</td>
</tr>
<tr>
<td>1, decrease</td>
<td>(0.95, 1, 1)</td>
<td>70.4152</td>
<td>0.5874</td>
</tr>
<tr>
<td>2, increase</td>
<td>(1, 1.05, 1)</td>
<td>69.5934</td>
<td>-0.2513</td>
</tr>
<tr>
<td>2, decrease</td>
<td>(1, 0.95, 1)</td>
<td>70.1057</td>
<td>0.2779</td>
</tr>
<tr>
<td>3, increase</td>
<td>(1, 1, 1.05)</td>
<td>69.1142</td>
<td>-0.7306</td>
</tr>
<tr>
<td>3, decrease</td>
<td>(1, 1, 0.95)</td>
<td>70.7115</td>
<td>0.8837</td>
</tr>
</tbody>
</table>

Since activity 3, yields the approximate derivative of greatest decrease in cost, it becomes the candidate activity. Thus, we take the bounds $l = (1, 1, 1)$ and $u = (1, 1, 3)$ and we begin the Fibonacci search procedure with $e_d = .01$. Since $u_3 - x_3 = 2$, we know the Fibonacci search procedure will take 12 iterations, as $2 \cdot r^{10} < .01$. The following table details the optimization procedure:
Table 3: Fibonacci Search Procedure Example

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$L_3$</th>
<th>$U_3$</th>
<th>$m_{l3}$</th>
<th>$C(m_l)$</th>
<th>$m_{u3}$</th>
<th>$C(m_u)$</th>
<th>$u_3 - l_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1.7639</td>
<td>69.325</td>
<td>2.2361</td>
<td>73.7746</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2.2361</td>
<td>1.4721</td>
<td>67.7198</td>
<td>1.7639</td>
<td>69.325</td>
<td>1.2361</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.7639</td>
<td>1.2918</td>
<td>67.5562</td>
<td>1.4721</td>
<td>67.7198</td>
<td>0.7639</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.4721</td>
<td>1.1803</td>
<td>67.9483</td>
<td>1.2918</td>
<td>67.5562</td>
<td>0.4721</td>
</tr>
<tr>
<td>4</td>
<td>1.1803</td>
<td>1.4721</td>
<td>1.2918</td>
<td>67.5562</td>
<td>1.3607</td>
<td>67.5179</td>
<td>0.2918</td>
</tr>
<tr>
<td>5</td>
<td>1.2918</td>
<td>1.4721</td>
<td>1.3607</td>
<td>67.5179</td>
<td>1.4033</td>
<td>67.5592</td>
<td>0.1803</td>
</tr>
<tr>
<td>6</td>
<td>1.2918</td>
<td>1.4033</td>
<td>1.3344</td>
<td>67.5189</td>
<td>1.3607</td>
<td>67.5179</td>
<td>0.1115</td>
</tr>
<tr>
<td>7</td>
<td>1.3344</td>
<td>1.4033</td>
<td>1.3607</td>
<td>67.5179</td>
<td>1.3769</td>
<td>67.5318</td>
<td>0.0689</td>
</tr>
<tr>
<td>8</td>
<td>1.3344</td>
<td>1.3769</td>
<td>1.3506</td>
<td>67.5126</td>
<td>1.3607</td>
<td>67.5179</td>
<td>0.0426</td>
</tr>
<tr>
<td>9</td>
<td>1.3344</td>
<td>1.3607</td>
<td>1.3444</td>
<td>67.5107</td>
<td>1.3506</td>
<td>67.5126</td>
<td>0.0263</td>
</tr>
<tr>
<td>10</td>
<td>1.3344</td>
<td>1.3506</td>
<td>1.3406</td>
<td>67.51</td>
<td>1.3444</td>
<td>67.5107</td>
<td>0.0163</td>
</tr>
<tr>
<td>11</td>
<td>1.3344</td>
<td>1.3444</td>
<td>1.3382</td>
<td>67.5188</td>
<td>1.3406</td>
<td>67.51</td>
<td>0.0101</td>
</tr>
<tr>
<td>12</td>
<td>1.3382</td>
<td>1.3444</td>
<td>1.3406</td>
<td>67.51</td>
<td>1.342</td>
<td>67.5102</td>
<td>0.0062</td>
</tr>
</tbody>
</table>

Thus the new allocation at the end of the first iteration is $X_1 = (1, 1, 1.3413)$, where 1.3413 is the midpoint between $m_{l3}$ and $m_{u3}$ in the final iteration. The algorithm continues from this new allocation. After 5 total iterations the algorithm stops at an optimal solution of $X_5 = (1.4284, 1.3150, 1.4171)$ at a cost of 64.3559. Ramachandra [2006] solved the same example problem, though his cost function accounted for lateness in terms of the cost of expected lateness and he limited his resource allocations to 0.25 increments. His procedure resulted in a solution of $X^* = (1.5, 1.5, 1.5)$ at an expected cost of 62.38. Taking into account the fundamental differences in the cost functions and resource allocations, these solutions are more or less equivalent.
Chapter 5

Results

5.1 Computational Results

The algorithm was tested on a set of 33 networks representing a variety of project
network structures. Project networks were generated randomly using RanGen2, developed
by Vanhoucke et al [2008]. Details regarding RanGen2 and its use can be found in Appendix
A. The RanGen2 generator allows the user to select networks by specifying values of six
parameters. The values for these parameters were chosen to represent a wide between
variety of network structures and properties, such as the number of activities, the degree of
parallelism, the length of the longest path in the graph; etc. Work content distribution
parameters were generated randomly in MATLAB. These parameters were sampled from a
uniform distribution between 0.1 and 2.0. The lower bound on resource allocation was taken
as 0.1 and the upper bound was 3.0 units for all activities. Relative lateness costs, $c_L$, were
fixed at 3 for all examples and $e_p$ was fixed at 0.005.

Testing was conducted using the optimization algorithm initiated from 5 input
allocations (i.e. for each network the optimization procedure was repeated 5 times, each with
a different initial resource allocation). These allocations included the allocation of 1 resource
unit, 2 resource units, and a randomly selected whole number of resource units between 1
and 3 to each activity. The optimization algorithm was then run to completion for each of the
5 initial allocations. Details regarding the test cases and the test results can be found in Table
4. Several additional networks were generated, however, we were unable to achieve the
optimum allocation therein because the size of the state spaces in their CTMC representations were prohibitively large for the computer at our disposal.

Table 4: Test Results

<table>
<thead>
<tr>
<th>Network</th>
<th>Indicators as per Vanhoucke et al</th>
<th>Number of States</th>
<th>Min Sol'n Time, sec</th>
<th>Average Sol'n Time, sec</th>
<th>Range of Sol'n, $</th>
<th>Maximum % Deviation From Minimum</th>
<th>Exact % Deviation From Approx.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8 0.25 0.16 1</td>
<td>74</td>
<td>15.049</td>
<td>18.878</td>
<td>0.006</td>
<td>0.00987</td>
<td>-9.929E-05</td>
</tr>
<tr>
<td>2</td>
<td>8 0.25 0.83 1</td>
<td>34</td>
<td>2.350</td>
<td>3.239</td>
<td>0.017</td>
<td>0.000232</td>
<td>-0.0001041</td>
</tr>
<tr>
<td>3</td>
<td>8 0.25 0.16 1</td>
<td>160</td>
<td>111.359</td>
<td>165.440</td>
<td>0.004</td>
<td>0.000269</td>
<td>-0.0003656</td>
</tr>
<tr>
<td>4</td>
<td>8 0.25 0.83 1</td>
<td>130</td>
<td>61.390</td>
<td>74.742</td>
<td>0.016</td>
<td>0.002571</td>
<td>-0.0025246</td>
</tr>
<tr>
<td>5</td>
<td>8 0.25 0.33 1</td>
<td>39</td>
<td>2.440</td>
<td>2.828</td>
<td>0.003</td>
<td>0.000620</td>
<td>-0.0005444</td>
</tr>
<tr>
<td>6</td>
<td>8 0.75 0.8 1</td>
<td>11</td>
<td>0.144</td>
<td>0.296</td>
<td>0.015</td>
<td>0.001165</td>
<td>-0.0004186</td>
</tr>
<tr>
<td>7</td>
<td>8 0.75 0.8 0.5</td>
<td>12</td>
<td>0.513</td>
<td>0.570</td>
<td>0.434</td>
<td>0.002897</td>
<td>-0.003021</td>
</tr>
<tr>
<td>8</td>
<td>8 0.75 0.8 0.5</td>
<td>17</td>
<td>0.151</td>
<td>0.354</td>
<td>0.097</td>
<td>0.008333</td>
<td>0.0006324</td>
</tr>
<tr>
<td>9</td>
<td>8 0.75 0.99 0</td>
<td>25</td>
<td>0.708</td>
<td>1.224</td>
<td>0.075</td>
<td>0.001804</td>
<td>0.0019552</td>
</tr>
<tr>
<td>10</td>
<td>8 0.75 0.99 0.25</td>
<td>23</td>
<td>0.614</td>
<td>0.782</td>
<td>2.575</td>
<td>0.024365</td>
<td>0.0009171</td>
</tr>
<tr>
<td>11</td>
<td>12 0.25 0.5 0.8</td>
<td>77</td>
<td>24.228</td>
<td>29.261</td>
<td>0.009</td>
<td>0.000925</td>
<td>0.0008868</td>
</tr>
<tr>
<td>12</td>
<td>12 0.25 0.16 1</td>
<td>228</td>
<td>681.933</td>
<td>1056.203</td>
<td>0.033</td>
<td>0.000799</td>
<td>0.0208778</td>
</tr>
<tr>
<td>15</td>
<td>12 0.25 0.33 0.21 0.66</td>
<td>246</td>
<td>1123.007</td>
<td>1787.993</td>
<td>0.077</td>
<td>0.000999</td>
<td>0.0224454</td>
</tr>
<tr>
<td>16</td>
<td>12 0.75 0.75 1</td>
<td>16</td>
<td>0.707</td>
<td>0.828</td>
<td>0.059</td>
<td>0.005738</td>
<td>-0.0008925</td>
</tr>
<tr>
<td>17</td>
<td>12 0.75 0.75 0.5</td>
<td>33</td>
<td>1.797</td>
<td>2.441</td>
<td>0.012</td>
<td>0.001089</td>
<td>0.0033800</td>
</tr>
<tr>
<td>18</td>
<td>12 0.75 0.75 0.5 0.61</td>
<td>20</td>
<td>0.846</td>
<td>1.062</td>
<td>0.033</td>
<td>0.001005</td>
<td>0.0025955</td>
</tr>
<tr>
<td>19</td>
<td>12 0.75 0.99 0</td>
<td>80</td>
<td>16.706</td>
<td>24.237</td>
<td>0.024</td>
<td>0.001839</td>
<td>0.0034233</td>
</tr>
<tr>
<td>20</td>
<td>12 0.75 0.99 0.28</td>
<td>58</td>
<td>9.773</td>
<td>13.206</td>
<td>1.804</td>
<td>0.011680</td>
<td>0.0031863</td>
</tr>
<tr>
<td>22</td>
<td>16 0.25 0.22 1</td>
<td>93</td>
<td>49.454</td>
<td>70.324</td>
<td>0.013</td>
<td>0.000779</td>
<td>0.0001813</td>
</tr>
<tr>
<td>26</td>
<td>16 0.75 0.9 0 1</td>
<td>196</td>
<td>491.106</td>
<td>918.612</td>
<td>1.450</td>
<td>0.015716</td>
<td>0.0164392</td>
</tr>
<tr>
<td>27</td>
<td>16 0.75 0.72 0</td>
<td>126</td>
<td>39.913</td>
<td>105.635</td>
<td>0.032</td>
<td>0.001849</td>
<td>-0.0010142</td>
</tr>
<tr>
<td>28</td>
<td>16 0.75 0.72 0.5</td>
<td>128</td>
<td>113.700</td>
<td>173.073</td>
<td>0.306</td>
<td>0.007955</td>
<td>-0.0070740</td>
</tr>
<tr>
<td>29</td>
<td>16 0.75 0.72 0.5 0.5</td>
<td>58</td>
<td>5.387</td>
<td>14.399</td>
<td>0.325</td>
<td>0.016273</td>
<td>-0.0015032</td>
</tr>
<tr>
<td>30</td>
<td>16 0.75 0.81 0.5 0.92</td>
<td>26</td>
<td>1.269</td>
<td>2.020</td>
<td>1.368</td>
<td>0.043330</td>
<td>-0.0034718</td>
</tr>
<tr>
<td>33</td>
<td>20 0.25 0.08 0.82 1</td>
<td>96</td>
<td>94.483</td>
<td>137.139</td>
<td>0.030</td>
<td>0.000601</td>
<td>0.0218544</td>
</tr>
<tr>
<td>36</td>
<td>20 0.75 0.71 0.5 0.71</td>
<td>234</td>
<td>1711.437</td>
<td>2199.084</td>
<td>0.376</td>
<td>0.0043725</td>
<td>0.0093561</td>
</tr>
<tr>
<td>37</td>
<td>20 0.75 0.71 0.71</td>
<td>77</td>
<td>36.775</td>
<td>48.890</td>
<td>0.047</td>
<td>0.003105</td>
<td>0.0040716</td>
</tr>
<tr>
<td>38</td>
<td>20 0.75 0.71 0.42 0.9</td>
<td>35</td>
<td>3.802</td>
<td>6.165</td>
<td>0.092</td>
<td>0.001536</td>
<td>0.0017733</td>
</tr>
<tr>
<td>39</td>
<td>20 0.75 0.78 1</td>
<td>28</td>
<td>1.928</td>
<td>3.354</td>
<td>0.281</td>
<td>0.013102</td>
<td>0.0023925</td>
</tr>
<tr>
<td>40</td>
<td>20 0.75 0.78 0.62 1</td>
<td>31</td>
<td>2.702</td>
<td>4.973</td>
<td>0.057</td>
<td>0.001242</td>
<td>0.0072959</td>
</tr>
<tr>
<td>AKS I</td>
<td>4 NA NA NA NA</td>
<td>7</td>
<td>0.119</td>
<td>0.273</td>
<td>0.007</td>
<td>0.002311</td>
<td>0.0002230</td>
</tr>
<tr>
<td>AKS II</td>
<td>6 NA NA NA NA</td>
<td>17</td>
<td>0.360</td>
<td>0.450</td>
<td>0.034</td>
<td>0.003725</td>
<td>-0.000464</td>
</tr>
<tr>
<td>AKS III</td>
<td>10 NA NA NA NA</td>
<td>25</td>
<td>1.669</td>
<td>1.869</td>
<td>0.040</td>
<td>0.002898</td>
<td>-0.001299</td>
</tr>
</tbody>
</table>
The indicators used by Vanhoucke et al. [2008] can briefly be described as follows:

- **I1** – The number of activities in the network
- **I2** – A series vs. parallel indicator. Low values for I2 indicate the graph has a high degree of parallelism and high values indicate that the graph follows a more series representation. \( I2 = 1 \) indicates that all activities occur in series.
- **I3** – Indicator describing the spread of activities over the progressive levels of the graph
- **I4** – Indicator representing the presence of short arcs.
- **I5** – Indicator representing the presence of long arcs.

Further details on these indicators, including equations for their calculation is provided in Appendix A. Vanhoucke et al. [2008] also specify an I6 as an indicator representing the topological float of the activities, however, after specifying only these five indicators we were often left with only one randomly generated network.

These results show that this algorithm is relatively efficient as many of the networks were solved a few seconds on average. No network averaged longer than 37 minutes to solve. The computed optimal costs across all 5 input allocations differed by less than 5% in all of the test networks and less than 1% in 82% of the test networks. Note that decreasing \( \Delta t \), thus making the integral portion of the cost function more accurate, could increase the precision of the returned solutions while increasing the computation time, however the obtained level of accuracy was satisfactory for our purposes.

One important observation that immediately can be seen is the correlation between the number of states in the CTMC and the computation time required for optimization. Clearly, as the number of states in the CTMC grows, the computational requirement of the
algorithm increases exponentially as demonstrated in Figure 3 below. This is due to the many matrix operations required by the algorithm, especially exponentiation. Though many of these exponentiations have been simplified or approximated throughout the algorithm, it is impossible to remove them completely and retain the CTMC method of analysis. The remaining steps involving computing $e^T$ are concentrated, primarily, in the Fibonacci Search Procedure, wherein it is not advantageous to make simplifying assumptions, as we did in the candidate activity selection step. The accuracy required for optimizing resource allocation to a single variable demands precise calculation of $e^T$.

![Figure 3: Solution Time vs Number of States in CTMC](image)

The last column of Table 4 measures the percentage difference between the minimum solution determined by our optimization algorithm and the solution determined using the exact evaluation of the matrix exponential in the derivative approximation step. As can be seen, the absolute difference from the exact optimum never exceeds 2.24% in all instances, and is less than 0.5% in 39 of our 43 experiments. This illustrates the validity of the approximation used in this step of the optimization procedure (described in Section 4.2).
Another important observation is that while the exact derivative procedure often arrives at a better solution than the approximation, equally often the approximation yields a lower optimal cost. Such a result is possible because when we deal with approximations to the derivatives and different initial allocations, activities may be selected in differing orders. Defining our stopping criteria as we have allows iteration to conclude in a small range of allocations, thus the exact procedure may conclude in this range at a greater optimal cost than our approximation. This further illustrates that the exact procedure does not provide any real advantage in terms of accuracy to the approximation pursued in our algorithm.

5.2 Network Structure Considerations

As expected, the most important determining factor in the computational time required by the algorithm is the number of states in the CTMC representation of the project. Larger CTMCs have larger generator matrices and thus require more computational time to multiply and especially more time to exponentiate these matrices. Furthermore, the storage requirement becomes especially burdensome when the state space grows into the thousands. This scenario is possible for projects with as few as 10 activities depending on the structure of the network (recall that the state space grows exponentially in the number of activities for highly parallel networks).

A few observations can be made regarding the projects in our sample that shall help determine how many states will exist in the CTMC. First, independent parallel paths are multiplicative in nature when producing states in the CTMC, implying the aforementioned exponential growth. As an example, suppose a project network has 10 states in its CTMC representation. Adding a single activity in parallel with the existing activities will double the
number of states in the CTMC representation, as now we will have the 10 original states with the additional activity *active* and 10 states with the additional activity *dormant*. Additional activities in series are additive in nature when producing states in the CTMC. Consider as before, a project network with 10 states in its CTMC representation. A single activity in series that must be completed before (or after) all other activities adds only 1 state to the CTMC as this activity will be *idle* in the 10 original states. These facts are further illustrated in the networks presented in Figure 5. In Figure 5a, activities 1 and 3 are in series where in 5b these two activities are in parallel. The network of Figure 5b yields two more states in its CTMC than that of Figure 5a.

Another observation is that projects with a greater number of progressive levels tend to have fewer states in their CTMC representations than other projects with the same number of activities. The progressive level of an activity is the largest number of tasks along a path from the activity to the start node. The large number of progressive levels means that a long path exists within the network, leaving fewer activities to lie on parallel paths.

An additional observation is that often the complexity index, CI, of a project, defined by Bein, Kamburowski, and Stallmann, [1992], is less indicative of the difficulty of a project than in other project scheduling problem scenarios (such as in estimating the completion time). This stems from the fact that non-zero complexity indices are often derived from arcs extending between multiple, otherwise distinct, parallel paths. These “cross arcs” often dramatically reduce the number of states in the CTMC representation. Additionally, the optimizing algorithm could be considered robust in that it does not rely heavily on specific precedence constraints in its iteration, further reinforcing this idea.
Some of these ideas can be best reinforced with an example. Consider the networks in Figures 4 through 7 below. Figure 4 provides an example of a network with 11 activities and CI = 3. This example has 44 states in its representative CTMC:

![Network Diagram](image)

**Figure 4: Example Network 2**

Figure 5 provides 2 examples of networks obtained from the example in Figure 4, both with 11 activities and equivalent complexity indices. These graphs have 42 and 44 states in their CTMCs respectively. Note that the graph of Figure 5a has an additional progressive level that is lacking in Figure 5b and both graphs have CI = 2.

![Network Diagram](image)

(a)

![Network Diagram](image)

(b)

**Figure 5: Example Network 2 with Reduced Complexity**

Figure 6 provides another example with 11 activities obtained from the example in Figure 4, again with a reduced complexity index. This graph has 28 states in its CTMC and CI = 1:
Figure 6: Example Network 2 with Further Complexity Reduction

Figure 7 provides a network with 11 activities that is series parallel; i.e. CI = 0:

Figure 7: Series-Parallel Network

Figure 7 is an excellent example of how series-parallel graphs often have more states in their corresponding CTMCs than graphs with greater complexity indices. Figure 7 has 120 states in its corresponding CTMC. It is also important to note that the complexity reductions obtained through the reductions between Figures 4, 5, and 6 lead to reductions in the number of states in the CTMC, however reducing the graph to a completely series-parallel graph produces more states than the original graph in Figure 4.

These five project networks, represented graphically in Figures 4, 5a, 5b, 6, and 7, were tested using the optimization algorithm in the same manner as the randomly generated networks. Test results are given in Table 5. Project parameters are as follows:

- $\lambda_j = [0.6885, 1.1724, 1.3330, 0.4007, 0.2630, 1.3469, 0.3725, 1.6764, 1.2735, 1.7496, 1.5418]$,  
- $T_s = 15$,  
- $C_L = 3$.  

42
Note that these tests further verify the statements made in the previous section regarding the number of states in the CTMC being indicative of the solution time. Further, this information confirms that a non-zero complexity index can dramatically reduce the number of states in the CTMC. Here, the percentage deviation from the minimum value is less than 1% regardless of the input allocation. Further, the percentage deviation of the solutions obtained using the exact computation of the matrix exponential in the candidate activity selection step from the minimum obtained through the use of our algorithm is less than 1% in each of these 5 cases as well.
Chapter 6

Conclusions and Future Research

6.1 Conclusions

The objective of this research was to develop and test an algorithm to solve the problem of optimal resource allocation in an activity network with exponential work content distributions. Previous papers have been concerned with optimizing either multiple objectives, as in Azaron et al. [2007] or an economic objective evaluating the cost of resources and the cost of expected lateness, as in Ramachandra [2006]. Our research instead dealt with the expected cost of a project as the sum of resource costs and the expected cost of lateness.

In Chapter 2 we explored much of the research that has been previously performed in relation to our own. We show that the deterministic Resource Constrained Project Scheduling Problem and the Time Cost Tradeoff Problem have been studied in depth but many of the discoveries made have not been carried over to the stochastic case. The research that has been conducted pertaining to Stochastic project scheduling problems is sparse at best, however, several papers have been published recently indicating that perhaps new interest is arising in this area.

In Chapter 3 we discuss several preliminaries demanded by our research. We validate our assumption that work content follows an exponential distribution by showing that such an assumption can provide an upper bound on the expected project cost and lay an important foundation for treating other probability distributions of the work content. With this
assumption in hand we show that the problem can be modeled as a CTMC, thus enabling us to apply PH-Type distribution functions to the project completion time. These functions allow us to formally state the cost function as the sum of resource costs and lateness costs.

We turn our research to the development of an optimizing algorithm in Chapter 4. With the cost function in hand, we show an efficient method for the computation of the expected project cost. We then use this computation in our approximation of the derivatives of the cost function with respect to resource allocation to a single activity. Further, we show that these derivatives can be approximated through matrix multiplication, which greatly decreases computation time while causing no effective change in the output of the optimization algorithm. The derivative approximation yields the selection of a candidate activity to which we will optimize the resource allocation through the use of Fibonacci search. Finally, we discuss a few stopping conditions that may be adopted for termination and validate our own choice where iteration stops when the expected cost decreases by only a small amount in its last iteration.

We have performed copious tests of the algorithm using randomly generated project networks and discussed several key results. We show that for the sample projects we tested, the optimal resource allocation could be determined from any initial allocation through the use of our algorithm. Tests showed that the worst expected cost returned by an input solution differed from the exact minimum expected cost by less than 5% in all of the tests conducted and less than 1% in the vast majority of the tests. Further, the precision of these figures can be increased by careful selection of some project parameters. We show that the computation time required by the optimization algorithm is strongly determined by the number of states in the CTMC representation of the project. As a result, we discuss a few of the determining
factors to the number of states in the CTMC. Notably, the “parallelism” of the project network (i.e. the degree to which different activities can be conducted in parallel) has a large effect on the number of states in the CTMC. Finally, we illustrate these results through several examples.

6.2 Directions for Future Research

A direct extension to this research would be to consider more general functional relationships between the $W$, $x$ and $Y$ parameters. For example, one might investigate the use of $Y = W/x^\alpha$ where $\alpha$ represents the potential decrease in the “efficiency” in resource utilization due to the interference. Researchers could also incorporate various levels of $\alpha$ for different project activities. In our research the hyperbolic relationship ($Y = W/x$) meshed well with the quadratic assumption regarding project cost by creating linear resource costs. This quadratic assumption could be relaxed as well by using a factor $\beta$, transforming equation (3) to:

$$C_j = c_\alpha x_j^\beta Y_j = c_\alpha x_j^{\beta-\alpha} W_j$$

Considering other versions of tardiness costs yields a variety of possible research extensions. For simplicity we have assumed that the cost of tardiness is a linear function of the completion time and due date, but a more general form of tardiness cost could be incorporated. Analysis could be applied to various forms of tardiness costs such as quadratic or piecewise linear to test the sensitivity of the optimal solution to various cost functions. Further, the due date itself could be taken as a decision variable as in much of the project
scheduling research. Such a change would increase the complexity of the problem greatly as a change in due date can cause extensive changes in the allocation of resources to activities.

One possible extension to the research presented here is to relax the assumption that the work content follows an exponential distribution. This assumption predicates our ability to model a project as a CTMC. The question then arises: would this approach fail if one departs from this assumption? From a conceptual point of view the answer is “No”.

Fortunately, this model remains valid due to the fact that almost any continuous distribution function that may be proposed, such as the beta or gamma distribution, can be approximated to an arbitrarily close degree by the generalized Erlang distribution. Since the latter is a “phase-type distribution” of a number of “stages”, wherein stage $k$ is exponentially distributed with parameter $\lambda_k$, we return to the exponential distribution paradigm. In this model, however, several activities in series now stand for the original, single activity. Note that such a change would cause a dramatic increase in the state space of the CTMC and the computing time.

It should be mentioned that this fact was noted by Azaron et al [2007], who used it in an illustrative example of 10 activities with one activity possessing a generalized Erlang distribution of only two stages. As they noted relative to the CTMC model,

“The limitation of the proposed model is that the state space can grow exponentially with the network size. As the worst case example, for a complete transformed network with $n$ nodes and $n(n - 1)/2$ activities, the size of the state space is given by [Kulkarni and Adlahka [1986]]:

$$N(n) = U_n - U_{n-1}$$

where
in practice, the number of activities in an activity network (AN) is generally much less than \(n(n - 1)/2\). It should also be noted that for large networks, any alternate method of producing reasonably accurate answers will be prohibitively expensive.”

To these remarks it should be added that substituting several activities in series (representing the “stages” of the generalized Erlang distribution) would increase the size of the state space – which is already large enough – by at least an order of magnitude, thus rendering it computationally onerous except for small ANs. (This explains our caveat above concerning the “conceptual” extension of our CTMC model to non-exponential work content. It also explains why Azaron et al [2007], limited their example to a single activity with a generalized Erlang distribution of only two stages.)

Another possible extension to this research would be the application of this optimization approach to other stochastic problems that can be modeled using CTMCs. Though the cost would be evaluated by a different function, the optimizing procedure could perform in the same way, provided the cost function remains convex. Thus the algorithm could be modified to optimize continuous time Markov Decision processes where the transition rates are continuous functions of bounded input parameters. A few such problems may be machinery maintenance modeling where failure times are random, Automatic Guided Vehicle (AGV) routing, and emergency services planning. These scenarios have already been modeled as discrete time Markov Chains. Without much further effort, continuous models could be developed which could be solved with this algorithm.

\[
U_n = \sum_{k=0}^{n} 2^{k(n-k)} .
\]
A few research extensions pertain to the resource requirements. One key limitation to this model is the single resource requirement. Many projects have activities requiring multiple different types of resources, e.g. bulldozers and bulldozer drivers. One alteration to the research in this paper would be to extend the resource requirements to multiple resource allocations. Problems of this type are often difficult, even in the deterministic case, but there are certainly many advantages to be gained by successful solution of such a problem.

Additionally, we assume no aggregate limit on resource availability. Adding an aggregate resource constraint would certainly be beneficial in real scenarios, however, further research is required for optimization of problems of this type as it makes the problem much more complex. In this scenario we must consider the points in time in which the resource allocation to a particular activity may be changed. For example, if one resource is assigned to a task due to the aggregate resource constraint, when another activity is completed additional resources become available and may be assigned to the first task. This idea suggests a single stage model in which assignment is “reoptimized” after any activity completes.

Yet another avenue of future research may be to consider Generalized Activity Networks (GANs). The research conducted in this paper deals with a predetermined set of activities that must be completed with specified precedence constraints. Often, a project has certain decision points in which a planner may have a few choices of activities that can occur, often with different costs and/or rewards. For example, a home could be built with a brick or composite siding exterior, each offering different activity times, costs, and value added to the home. In some scenarios, this “choice” of activities is specified by a probability distribution. Researchers would need to consider both the choice of activities as well as the
resource allocation to the chosen activities in this subject area. Certainly, there are many opportunities for further research here.
Bibliography


Appendix
Appendix A: Test Network Generation

All of our test networks were generated using RanGen2, developed by Vanhoucke et al [2008]. The RanGen2 generator takes the number of activities, \( I_1 \), and a series-parallel indicator, \( I_2 \), as input to generate test networks, where:

\[
I_2 = \begin{cases} 
1 & \text{if } n = 1 \\
\frac{m-1}{n-1} & \text{if } n > 1 
\end{cases}
\]  

where \( n \) is the number of activities and \( m \) is the number of precedence levels. This series-parallel indicator measures how close the graph holds to a series or a parallel representation, not if the graph is series-parallel. The RanGen2 generator operates by generating a network in the AoN mode of representation with the pre-specified number of nodes (which represent activities in RanGen2) and enough precedence relationships (arcs) to ensure that the value of the series-parallel indicator is at least as great as the user-specified input value. Arcs are then removed one by one until the input value for the series-parallel indicator is reached.

In each generation run, RanGen2 creates multiple networks based on the set of arcs removed in the last step of generation. Several other parameters are calculated for each network. RanGen2 users can then select random networks from this set based on the other parameters at the values of their choosing. These parameters are summarized as follows:

- \( I_3 \) – Measures the distribution of the activities over the progressive levels of the network:

\[
I_3 = \begin{cases} 
0 & \text{if } m \in \{1,n\} \\
\frac{\sum_{a=1}^{m} w_a - \eta_a}{\sum_{a=1}^{m-1} \frac{w_a - \eta_a}{m-1}} & \text{if } m \not\in \{1,n\} 
\end{cases}
\]  

where \( w_a \) is the width of progressive level \( a \); i.e. the number of activities at this progressive level, and \( \bar{w} \) is the average width of a progressive level. The progressive
(regressive) level of an activity can be thought of as the length of the longest path from the start (end) node to the activity in question plus 1.

- **I4** – Measures the presence of short arcs based on the progressive levels of the endpoints of each arc in the network:

\[
I_4 = \begin{cases} 
1 & \text{if } D = n - w_1 \\
\frac{n - n + w_1}{D - n + w_1} & \text{if } D > n - w_1 
\end{cases} 
\]  

(25)

where \( D \) is the maximum possible number of arcs of length 1 (short arcs) in a network, \( n' \) is the number of short arcs in the network, and \( w_1 \) is the number of activities in the first progressive level; i.e., the number of activities with no predecessors.

- **I5** – Measures the presence of long arcs based on the progressive levels of the endpoints of each arc in the network:

\[
I_5 = \begin{cases} 
1 & \text{if } |A| = n - w_1 \\
\frac{\sum_{i=2}^{m-1} n_i - n + w_1}{|A| - n + w_1} & \text{if } |A| = n - w_1 
\end{cases} 
\]  

(26)

where \( |A| \) is the number of arcs in the network.

- **I6** – Measures the topological float of each activity based on the progressive and regressive levels of each activity:

\[
I_6 = \begin{cases} 
0 & \text{if } m \in \{1, n\} \\
\sum_{i=1}^{m} \frac{(RL_i - PL_i)}{(m-1)(n-m)} & \text{if } m \not\in \{1, n\}
\end{cases} 
\]  

(27)

Where \( RL_i \) and \( PL_i \) denote the regressive and progressive levels, respectively, of activity \( i \) in the given network.

The projects generated by RanGen2 are created in the AoN (Activity on Node) mode of representation. In the AoN mode, nodes represent activities and arcs represent precedence...
constraints. Since the AoA mode of representation often necessitates the creation of “dummy activities” with no work content to properly identify all precedence constraints, the AoN mode is a convenient alternative. The AoN mode of representation, however, often includes dummy start and finish activities to give the project a definite starting and finishing point. As an example, the AoN representation of the project from Figure 1 is represented in Figure 8:

![Figure 8: Example Project in AoN Representation](image)

where activity 1 represents the dummy start activity and activity 5 represents the dummy finish activity. Note that the indices for activities 1, 2, and 3 in Figure 1 have been changed but their parameters remain the same.

Since the test networks are created in the AoN mode, in order to proceed with the optimization algorithm we must be able to enumerate the states in the CTMC. Since the transitions in the CTMC occur whenever an activity is completed, the CTMC will have a state corresponding to each possible combination of complete and incomplete activities. We can consider active and the subset of idle activities that have not begun processing as “incomplete” and dormant and the subset of idle activities that have completed processing as “complete”. This classification of states corresponds to the set of UDCs in the AoN mode of representation. As an example the states of the project represented in the AoN mode in Figure 8 (AoA mode in Figure 1) are described below:
Table 6: State Space of the AoN Example Network

<table>
<thead>
<tr>
<th>State</th>
<th>Activity 2 (AoN)</th>
<th>Activity 3 (AoN)</th>
<th>Activity 4 (AoN)</th>
<th>UDC (AoN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Incomplete</td>
<td>Incomplete</td>
<td>Incomplete</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Complete</td>
<td>Incomplete</td>
<td>Incomplete</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Incomplete</td>
<td>Incomplete</td>
<td>Complete</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Complete</td>
<td>Incomplete</td>
<td>Complete</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>Complete</td>
<td>Complete</td>
<td>Incomplete</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Complete</td>
<td>Complete</td>
<td>Complete</td>
<td>6</td>
</tr>
</tbody>
</table>

Note that these states correspond to the states we enumerated in Section 3.2. Thus, the states of the CTMC can be found directly from the AoN mode of representation by enumerating the UDCs of the AoN project graph.

Output from the RanGen2 application is not in either of the traditional forms of an adjacency matrix or an arc-node incidence matrix. Instead, RanGen2 creates a matrix with a row for each activity (including a dummy start and dummy finish activity). The first column of the output matrix is the activity number. Next, multiple columns are added representing any resource measures. For our purposes, however, we generated activity specific details, that is, the work content parameters, using Matlab. Finally, there is a column listing the number of arcs emanating from the particular activity followed by the destination nodes of these arcs. As an example, had RanGen2 generated the project given in Figure 3 without any resource measures, its output would be identical to the following table:
Table 7: Example Network Generator Output

<table>
<thead>
<tr>
<th>Activity</th>
<th>Out-degree</th>
<th>Nodes</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is the form of the “A” matrix given as input to the MATLAB optimizing procedure detailed in Appendix B.
Appendix B: MATLAB Scripts

The MATLAB scripts herein are added for reference only. A brief description of each is provided in the header of each script. Briefly, NetSolve.m runs the main iterative procedure of the algorithm. All other scripts are called from this script, either directly or indirectly.

This script operates by transforming the random network generator output into useful form and creating other necessary variables and matrices. BuildRTMAoN.m creates the first Rate Transition Matrix for the optimizing procedure and creates matrices enabling the BuildQ.m function to create subsequent RTMs based on any input allocation. Costsum.m is the function by which the cost is computed for a particular resource allocation. Costsum.m is used in the main iterative procedure as well as the derivative approximation step (modeled with RunDiff.m) and the single variable optimization step (modeled with StepLength.m). Finally, costofvector.m abbreviates the length of other scripts by aggregating the creation of the Q, T, and R matrices and the subsequent call of the Costsum.m script into a single line.

```matlab
% File: NetSolve.m
% Author: Adam Rudolph
% Input: A - Representation of project network
% lambda - Distribution parameter for work content
% TS - Project due date
% delta - Allocation added in approximation of derivatives
% cL - Cost of lateness. Normalized with respect to allocation costs.
% alloci - Input allocation vector (alloci is the current best allocation vector.
% Output: alloci - The optimal allocation vector
% newcost - The cost associated with a solution
% T - A vector stating which state was changed in each iteration (if negative, this indicates the allocation was decreased)
% mean - The mean completion time
% z - The time the function took to optimize the problem.
% Description: This function computes the allocation that minimizes the expected cost of the project.

function [alloci newcost T mean z] = NetSolve(A,lambda,TS,delta,cL,alloci)
tic % Starts timer

% Function BuildRTMAoN takes the AoN representation graph and transforms it into the problem's rate transition matrix (RTM). csedge is a matrix with a row for each possible transition in RTM. It is useful for computing
```

59
% rate transition matrices later.
[csedge lambda RTM] = BuildRTMAoN(A, lambda);

s = size(RTM,1); %Number of states in the Markov Chain representation
n = size(lambda,2); %Number of tasks (Same as number of values for lambda
alpha = [1, zeros(1,s-2)]; %Initialize alpha (the counting probability)
dcost = -1; %dcost is the change in cost for a particular iteration.

% Next five lines calculate the cost of the initial allocation. These five
% lines are aggregated in the file costOfVector.m for brevity later. In
% NetSolve.m they are disaggregated because the Rmat variable is essential
% to the derivative process.
[Qmat] = buildQ(alloci, lambda, csedge, s, n);
[Tmat] = Qmat(1:s-1, 1:s-1);
Rmat = expm(Tmat);
SDev = sqrt(2*alpha*(inv(Tmat))^2*ones(s-1,1));
[oldcost] = Costsum(Rmat, alpha, cL, TS, lambda, alloci, s, n, SDev);

% Initialize vector to store variable changed
T = [];

% The next section creates the delta matrices to use in approximating the
% derivative. Alloc is a square matrix with a row representing a change of
% delta for each task.
alloc = [];
for j = 1:n
    for i = 1:2*n
        if i == j
            alloc(i,j) = delta;
        elseif i == n + j
            alloc(i,j) = -delta;
        else
            alloc(i,j) = 0;
        end
    end
end

% ChApprox is a 3 dimensional matrix with a layer for the exponential
% associated with each changed allocation created above. The derivative
% function pulls a layer for matrix multiplication to approximate the
% derivatives.
ChApprox = [];
for i = 1:2*n
    QmatTemp = buildQ(alloc(i,:), lambda, csedge, s, n);
    TmatTemp = QmatTemp(1:s-1, 1:s-1);
    RmatTemp = expm(TmatTemp);
    ChApprox = cat(3, ChApprox, RmatTemp);
end

% While statement contains the main iterative scheme of the algorithm. It
% runs until the cost decreases by less than .5% of the resource cost.
while dcost < -.005
    % Function RunDiff calculates the task to change and the direction of
    % change. This is the "derivative" step. dir = 1 if the allocation
    % should be increased and -1 if it should be decreased.
    [task dir] = RunDiff(alloci, TS, alpha, lambda, cL, oldcost, Rmat, ChApprox, alloc, s, n, SDev);
    T = [T, task*dir]; % Add the task changed to the matrix T. i.e. add 8 if
    % task 8 is increased and -8 if 8 is decreased. The
    % length of T will correspond to the number of
    % iterations undertaken upon completion.

    % Function Step Length calculates the amount by which to increase or
    % decrease the selected task. Function returns a new allocation
    alloci = StepLength(task, alloci, csedge, lambda, alpha, cL, dir, s, n, SDev);

    % The next four lines compute the cost of the new allocation
    [Qmat] = buildQ(alloci, lambda, csedge, s, n);
    [Tmat] = Qmat(1:s-1, 1:s-1);
    Rmat = expm(Tmat);
    [newcost] = Costsum(Rmat, alpha, cL, TS, lambda, alloci, s, n, SDev);
    dcost = newcost - oldcost; % Compute the change in cost
\begin{verbatim}
SDev = sqrt(2*alpha*(inv(Tmat))^2*ones(s-1,1)); %Find the St Dev of time to completion
oldcost = newcost; % Update oldcost and then iterate if necessary.
end
mean = -1*alpha*inv(Tmat)*ones(s-1,1); %Find the mean time to completion. This is only done for reference
z = toc; %Stop Timer
\end{verbatim}
% File: BuildRTMAoN.m
% Author: Adam Rudolph
% Input: A, lambda
% Output: RTM - The rate transition matrix for the given allocation
% cseedge - Matrix with a row for each transition in RTM.
% lambda - Function truncates lambda by removing parameter for
dummy start and dummy finish nodes.
%
% Description: Function transforms input graph (A) into the initial Rate
% Transition Matrix (RTM). Also constructs matrix cseedge for producing
% appropriate RTMs later. For highly parallel projects, the number of
% states could grow prohibitively large.
%
% 1 2 3 4 5 6 7
%23456789012345678901234567890123456789012345678901234567890123456
function [cseedge lambda RTM] = BuildRTMAoN(A,lambda)

n = size(A,1); %Defines n as the number of activities in the original
%project graph. Includes dummy start and finish
lambda = lambda(2:n-1); %Truncate lambda matrix
endpts = []; %Initialize endpts matrix, counter and taskpreds.
count = 1;
taskpreds = zeros(n);

for i = 1:n
    for j = 1:A(i,2)
        taskpreds(A(i,1),A(i,2+j)) = 1;
        endpts = [endpts;[count A(i,1) A(i,2+j)]]; %number row for each
        count = count+1;
    end
end

%Truncate taskpreds to just the real activities of the project.
taskpreds = taskpreds(2:n-1,2:n-1);

n = n-2; %n is now the number of actual activities in the project.
cseedge = [];

%States matrix represents the state space for the Markov chain
%representation of the project. Rows represent project activities and
%columns represent states. A 1 indicates that for the given state the
%activity is complete and a 0 indicates that the activity is incomplete.
states = zeros(n,1);

%Counters for the number of states and the number of states explored
explored = 0; %Number of states explored
numstates = 1; %Number of states discovered

%While loop terminates when each state has been explored
while explored < numstates
    cst = states(:,explored+1); %Current state is the 1st unexplored state.

    %For each activity, i, we consider if it is possible to transition to a
%new state. Transitions are possible if all of activity i's
%predecessors are complete from the current state.
    for i = 1:n
        % If the activity is complete in the current state, skip
        % exploration for that activity.
        if cst(i) == 1
            predsmet = 0;

            %For loop counts the number of activity i's predecessors that
            %met at the current state.
            for j = 1:n
                if cst(j) >= taskpreds(j,i)
                    predsmet = predsmet + 1;
                end
            end

            % If all predecessors met, transition is possible
            if predsmet == n-2
                states(:,explored+1) = 1;

                %Increment state counter
                explored = explored + 1;

                %Increment state counter
                numstates = numstates + 1;
            end
        end
    end
end
end

\% If all of i's predecessors are met
if predsmet == n
    \% t is an indicator as to whether the state is new or
\% repeated. A state can only be repeated if it has been
\% discovered from a different state. Transitions in RTM
\% still exist if the state has been discovered.
t = 0;

\% Create a column for the state associated with completing
\% activity i.
newstate = cst;
newstate(i) = 1;

\% For each of the states already discovered
for l = 1:numstates
    \% s is a counter for the number of similarities between
\% the new state and a particular previously discovered
\% state
    s = 0;

    \% For loop counts similarities
    for k = 1:n
        if newstate(k) == states(k,l)
            s = s+1;
        end
    end

    \% If the new state is the same as a previously discovered
\% state, end the loop, change t to indicate state has
\% been discovered previously.
    if s == n
        t = t+1;
        repeatstate = l;
    end
end

\% If the new state has not been previously discovered, add it
\% to the states matrix, update the number of states, and add
\% the transition to the csedge matrix.
if t == 0
    states = [states,newstate];
    numstates = numstates+1;
    csedge = [csedge;[explored+1,numstates,lambda(i),i]];
else
    csedge = [csedge;[explored+1,repeatstate,lambda(i),i]];
end
end

\% After all possible activity completions have been investigated, update
\% the number of states explored.
explored = explored+1;
end

\% Build initial RTM matrix
C = size(csedge,1); \% Number of transitions in RTM
s = size(states,2); \% Total number of states
RTM = zeros(s);

\% Add each transition to RTM
for i = 1:C
    RTM(csedge(i,1),csedge(i,2)) = csedge(i,3);
end
%Correct diagonal elements of RTM.
for i = 1:s
    sumup = 0;
    for j = i+1:s
        sumup = sumup + RTM(i,j);
    end
    RTM(i,i) = -1*sumup;
end
function [RTM] = buildQ(alloci, lambda, csedge, s, n)

C = size(csedge,1); %Number of transitions in RTM
RTM = zeros(s); %Initialize RTM

%For each transition, add the rate in RTM
for i = 1:C
    RTM(csedge(i,1),csedge(i,2)) = csedge(i,3)*alloci(csedge(i,4));
end

%For loop corrects values along main diagonal
for i = 1:s
    sumup = 0;
    for j = i+1:s
        sumup = sumup + RTM(i,j);
    end
    RTM(i,i) = -1*sumup;
end
function [cost] = Costsum(R, alpha, cL, TS, lambda, alloc, s, n, SDev)

dt = max(SDev/20,.2); %dt is the width of each rectangle in the Riemann sum. Smaller values of dt will create a better approximation for the integral but will require more computation.

t=TS; %Initialize t (Since the cost if the project completes before the due date is zero we can initialize t to TS to save computation)

dtMat = R^dt; %Matrices stored to eliminate exponential in each step
RootMat = R^t;

%Calculates the probability that the project is finished before time = t.
prob = 1-alpha*RootMat*ones(s-1,1); 

cost = 0; %Initialize cost
% While loop runs until the probability that the project completes by time t is .9997. While loop does not incorporate the cost of the allocation, only the cost of the completion time.
while prob < .9997
    RootMat = RootMat*dtMat; %Update RootMat = R^t+dt
    prob1 = 1-alpha*RootMat*ones(s-1,1); 
    %Calculate the cost associated with a completion time of (2t+dt)/2 (i.e. the midpoint of the interval from t to t+dt)
    cost = cost + cL*((2*t+dt)/2-TS)*(prob1-prob);
    % Set the probability that the task will complete before time t+dt
    prob = prob1;
    t = t+dt; %Increase t by dt
end
%For loop adds the cost associated only with the allocation and not the completion time. Incorporating this cost in the above while loop would assume resources are used for the entire project's duration.
for i = 1:n
    cost = cost + alloc(i)/lambda(i);
end
% File: RunDiff.m
% Author: Adam Rudolph
% Input: alloci, TS, alpha, lambda, cL, s, n
% inicost - The cost associated with the input allocation
% Rmat - The exponentiated portion of the Q matrix.
% ChApprox - The exponentiated change allocations
% alloc - Matrix of change allocations
% Output: task - The task that has the steepest derivative
% dir - The direction to change the chosen task
% Description: This function computes the "derivative" of the cost
% function with respect to the allocation to each task and selects the
% candidate task associated with the greatest magnitude.

function [task dir] = RunDiff(alloci,TS,alpha,lambda,cL,initcost,Rmat,ChApprox,alloc,s,n,SDev)

% Initializes X matrix
X = [];

% For loop calculates the approximate cost associated with a minute change
% in allocation to a particular task. i.e. the derivative with respect to
% each allocation.
for i = 1:2*n
    % Approximate the exponentiated matrix
    R = Rmat*ChApprox(:,:,i);

    %Calculates the expected cost of the given allocation
    [cost] = Costsum(R,alpha,cL,TS,lambda,(alloci+alloc(:,i)),s,n,SDev);

    %Stores the task changed, cost, and the absolute change in cost in X
    %matrix.
    X = [X,[i;cost;cost-initcost]];
end

% Saves the value of the largest decrease in cost and the task the change is
% associated with
[change task] = min(X(3,:));

% If the cost associated with the changed allocation is less than the
% original cost then the allocation should be changed in the positive
% direction, otherwise it should be changed in the negative direction.
if task > n
    task = task - n;
    dir = -1;
else
    dir = 1;
end
function [alloci] = StepLength(task, alloci, csedge, lambda, alpha, cL, TS, s, n, SDev)

% Initialize vectors for upper and lower bounds and midpoints.
A = []; B = []; l = []; m = [];

% Store the given allocation as A
A(1,:) = alloci;

% Store the given allocation as B and change the allocation to the selected task to an acceptable value. If decreasing the allocation to a task, the second line below keeps the value from becoming negative. Upper bounds could be established at this step as well.
B(1,:) = alloci;
B(1,task) = max(B(1,task) + dir*2,.1);

% If the allocation is being decreased this interchanges A and B. Thus A is always the lower bound and B always the upper bound.
if dir == -1
    C = B;
    B = A;
    A = C;
end

% Initialize i and r (R is equal to (sqrt(5)-1)/2. It is associated with the golden ratio. This value of r allows us to calculate only one additional cost at each iteration.
i = 1;
r = 0.618034;

% Initializes the lower and upper midpoints and calculates their costs
l(i,:) = alloci;
m(i,:) = alloci;
l(i,task) = A(i,task)*r + B(i,task)*(1-r);
m(i,task) = A(i,task)*(1-r) + B(i,task)*r;
[costl] = costofvector(l(i,:),csedge,lambda,alpha,cL,TS,s,n,SDev);
[costm] = costofvector(m(i,:),csedge,lambda,alpha,cL,TS,s,n,SDev);

%While loop runs until the upper and lower bounds near eachother while abs(B(i,task)-A(i,task)) > .01;

% Compares costs and reassigns values to the variables.
if costl < costm
    A(i+1,:) = A(i,:);
    B(i+1,:) = m(i,:);
    l(i+1,:) = l(i,:);
    l(i+1,task) = A(i+1,task)*r + B(i+1,task)*(1-r);
    m(i+1,:) = l(i,:);
    costm = costl;
    costl = costofvector(l(i+1,:),csedge,lambda,alpha,cL,TS,s,n,SDev);
else
    A(i+1,:) = l(i,:);
    B(i+1,:) = B(i,:);
    l(i+1,:) = m(i,:);
    m(i+1,:) = m(i,:);
    m(i+1,task) = A(i+1,task)*(1-r) + B(i+1,task)*r;
    costl = costm;
    costm = costofvector(m(i+1,:),csedge,lambda,alpha,cL,TS,s,n,SDev);
end
\[ i = i+1; \]
\[ \text{end} \]
\[ \text{alloc}_i = (m(i,:) + l(i,:))/2; \]
\[ \% \text{Saves the final allocation as the value} \]
\[ \% \text{Of the midpoint between m and l.} \]
% File: costofvector.m
% Author: Adam Rudolph
% Input: alloci, cedge, lambda, alpha, cL, TS, s, n, SDev
% Output: cost - The cost associated with the allocation vector
% Description: This function computes the cost associated with a given
% allocation vector. Aggregation of four steps for the sake of brevity.

function [cost] = costofvector(alloci, cedge, lambda, alpha, cL, TS, s, n, SDev)

% Build the Q matrix for the given allocation
[Qmat] = buildQ(alloci, lambda, cedge, s, n);

% Remove the last row and column
[Tmat] = Qmat(1:s-1, 1:s-1);

% Exponentiate the matrix
Rmat = expm(Tmat);

% Run the cost summation function. The Rmat is computed external to the
% summation so that it is only computed once, cutting down on the
% computational burden
[cost] = Costsum(Rmat, alpha, cL, TS, lambda, alloci, s, n, SDev);