

A METHOD FOR LOCATING AND SIZING PROJECT TIME BUFFERS

Robert L. Bregman, University of Houston, C. T. Bauer College of Business, 334 Melcher Hall,
Houston, TX 77204-6021, RBregman@uh.edu, 713-743-4722

ABSTRACT

An iterative, matrix-analytic heuristic is proposed for dynamically allocating additional resources to tasks in a project network at various points in the project's evolution so as to provide time buffers in order to maximize the probability of timely project completion. At each iteration of the heuristic additional resources are allocated to uncompleted tasks based on a simulation-based estimate of the largest potential increase in the probability of timely completion per unit of resource invested. Two example problems are used to illustrate the operation of the heuristic and a large-scale simulation experiment is performed to evaluate the performance of the heuristic.

INTRODUCTION

In today's fast-paced and increasingly high-tech business environment project managers often must complete large-scale projects comprised of tasks with uncertain durations before predetermined dates to meet product or service implementation schedules. Unfortunately, projects comprised of numerous uncertain tasks often have little chance of being completed before fixed due dates because the start of each task is limited by the joint probability that all preceding tasks have been completed. Ultimately, these numerous joint probabilities interact so that the overall joint probability of completing all tasks before a project due date will approach zero. As a result, project managers need to take a proactive approach to increase the probability of a successful scheduling outcome. This usually involves creating time buffers at key points in a project schedule. For example, feeding time buffers might be used to decouple tasks on critical paths from tasks on alternate paths so that critical path tasks are not delayed by late noncritical tasks. Alternately, a project time buffer might be placed at the end of a project to increase the joint probability that all paths are completed before the project due date. And finally, subpath buffers may be used to decouple noncritical paths. Unfortunately, the large number of potential buffer locations, the complex interactions of uncertain task durations, and the variability of the cost of creating buffers combine to make the determination of the best locations a very difficult problem to solve. Furthermore, when tasks durations are uncertain the locations and sizes of time buffers can only be projected. The actual locations and sizes of the time buffers will be based on probabilistic outcomes. The purpose of this research is to introduce a method that can be used to locate and size these probabilistic time buffers to help managers control their schedule risk in a cost-effective manner when managing these types of projects.

Time buffers in projects can be created or increased by pushing forward the project start time, pushing back the project completion time, pushing forward the start of non-critical network paths, or reducing individual task durations. This research addresses the situation where projects are initiated immediately and due dates are predefined and fixed. In addition, it is assumed that tasks are initiated immediately upon the completion of predecessor tasks. In this situation the project schedule includes pre-existing time buffers that may or may not be sufficient to allow the project to be successfully completed before the due date under probabilistic conditions. The

location and size of these time buffers can then only be changed by reducing individual task durations. Hence, the time buffer location and sizing problem becomes an exercise in determining how to allocate additional resources in order to reduce individual task times to improve the probability of meeting a project due date.

In practice, projects are reevaluated over time. At each evaluation a plan may need to be developed for improving the probability of completing the remainder of the project before the due date. In the context of this research, project time buffers represent probabilistic plans that may or may not come to fruition. The portion of a plan implemented before a reevaluation may or may not actually result in time buffers (and may or may not actually improve the probability of meeting the project due date) depending on the actual probabilistic outcomes of tasks. Similarly, the unimplemented portion of a plan may remain the same or require revision based on probabilistic outcomes since the last evaluation and new information about future tasks. Although it is common for projects to be comprised of tasks with uncertain durations, there currently are no tools available for positioning (and repositioning) time buffers in the resultant probabilistic project networks in order to improve due date performance. This void provided the motivation for the development of the new approach introduced in this research.

RESEARCH MOTIVATION

The idea of inserting additional time buffers into project schedules to improve due date performance was first proposed as part of the critical chain scheduling and buffer management (CC/BM) approach to project scheduling introduced by Goldratt [1] and later described by Newbold [2] and Leach [3]. As noted by Herroelen and Leus [4], the time buffer management portion of the approach involves using a project buffer at the end of the schedule and feeding buffers between the critical chain and other noncritical tasks. Unfortunately, the CC/BM approach assumes that a well-defined sequence of tasks can be identified for buffering and the static buffers can then be evaluated over time. However, probabilistic project networks are not likely to have well-defined longest paths and buffer locations and sizes may shift dramatically as probabilistic outcomes become known. Predetermining and fixing buffers will often lead to incorrectly located buffers over time and delay the completion of the overall project as alternate paths turn out to have longer durations than the buffered path(s). Oddly, by predetermining buffer locations the CC/BM approach ignores the probabilistic conditions that create the need for the buffers. In addition, the CC/BM method requires that buffers be created by pushing forward the project start time or pushing back the project completion time. While this may be possible for many less-important projects, it is impractical for scheduling high-pressure projects that often require the expansion of buffers through task time reduction plans.

Of course, over the years many researchers have looked at the problem of determining the best set of tasks to reduce in order to shorten the expected duration of a project. The original work on the critical path method by Kelley and Walker [5], Kelley [6], and Fulkerson [7] evaluated the time-cost tradeoff problem under the assumptions of deterministic conditions and a continuous and convex total cost curve that can be accurately approximated as piece-wise linear. About the same time that the critical path method was being developed, Malcolm et al. [8], Miller [9], Pocock [10], Kahn [11], and others were describing details of the program evaluation and review technique, which allows probabilistic task durations based on a simplified version of the beta distribution. Although the program evaluation and review technique captures probabilistic task

times, like the critical path method it focuses entirely on critical path tasks which represent only a small subset of the overall probabilistic problem. The probabilistic network scenarios covered by this research are ignored by the program evaluation and review technique. This limitation can be overcome by using simulation methods per Bildson and Gillespie [12], Van Slyke [13], MacCrimmon and Ryavec [14], Klingel [15], Schonberger [16], and others. However, general simulation techniques are too inefficient to model and evaluate the large number of potential task time reductions in large-scale projects.

In addition to the prior cited weaknesses of the critical path method, the program evaluation and review technique, and simulation; all prior research in project task time reduction has been predicated on reducing the longest network path(s) to reduce the duration of the project as opposed to locating and sizing time buffers to improve due date performance. Although reduction of the longest path through a network will increase the project buffer, there is no methodology available for evaluating and selecting the combination of task time reductions that will create the best set of probabilistic network buffers. While the CC/BM approach directly addresses the buffer sizing problem, the method uses an overly simplistic static network with predefined buffer locations and does not provide a method to increase buffers through task time reductions, which makes it an impractical alternative for solving common time-limited probabilistic project network problems. The purpose of the new approach for locating and sizing project time buffers introduced in this research is to fill this methodological void.

A METHOD FOR CREATING PROBABILISTIC TIME BUFFERS

In order to provide the highest possible probability of completion before a project due date, a time buffering approach using matrix-based simulations at each project evaluation was developed. The matrix design overcomes the previously noted inefficiencies of general simulation by allowing numerous simulations to be conducted simultaneously. A new expediting option is considered for adding to the selected set (or plan) at each of the j iterations of the heuristic process, which is initiated at each of the i project evaluations (or reviews). The definitions (and dimensions) of the vectors, matrices, and hypermatrices developed and used by the method are as follows:

- \mathbf{G}_i 0-1 definitional matrix relating the q options to the p tasks at evaluation i (q,p)
- \mathbf{T}_i vector of times remaining until the due date for each of the m paths at evaluation i (m)
- \mathbf{E}_i diagonal matrix denoting the known time reductions for each of the q options at evaluation i (q,q)
- \mathbf{C}_i vector of costs associated with implementing each of the q options at evaluation i (q)
- $\hat{\mathbf{A}}_i$ matrix of n time estimates for each of the p tasks at evaluation i (n,p)
- \mathbf{O}_i 0-1 definitional matrix relating the p tasks to the m paths at evaluation i (p,m)
- \mathbf{a} vector of minimum duration estimates for the p tasks (p)
- $\mathbf{\beta}$ vector of maximum duration estimates for the p tasks (p)
- Min** vector of minimum possible path durations (m)
- Max** vector of maximum possible path durations (m)
- \mathbf{X}_{ij} vector of units of the q options available at evaluation i , iteration j (q)
- $\hat{\mathbf{D}}_{ij}$ matrix of n duration estimates for each of the m paths at evaluation i , iteration j (n,m)
- \mathbf{R}_i matrix denoting the reductions of the m paths by the q options at evaluation i (m,q)
- \mathbf{F}_{ij} hypermatrix of n replications of the reductions of the m paths for each of the q options at evaluation i , iteration j (n,m,q)
- $\hat{\mathbf{B}}_{ij}$ hypermatrix of q replications of $\hat{\mathbf{D}}_{ij}$ at evaluation i , iteration j (n,m,q)

- $\hat{\mathbf{W}}_{ij}$ hypermatrix representing the n duration estimates for the p paths after applying each of the q options at evaluation i , iteration j (n, m, q)
- \mathbf{S}_{ij} vector of estimated benefits of using each of the q options at evaluation i , iteration j (q)
- \mathbf{Z}_{ij} vector of estimated additional benefit per cost ratios of using each of the q options at evaluation i , iteration j (q)
- \mathbf{H}_{ij} vector of cumulative costs when adding each of the q options to the solution at evaluation i , iteration j (q)
- $\underline{\mathbf{F}}_{ij}$ alternate version of the \mathbf{F}_{ij} hypermatrix for the \bar{q} options in the solution at evaluation i , iteration j (n, m, \bar{q})
- $\hat{\mathbf{B}}_{ij}$ alternate hypermatrix of \bar{q} replications of $\hat{\mathbf{D}}_{ij}$ at evaluation i , iteration j (n, m, \bar{q})
- $\underline{\mathbf{W}}_{ij}$ alternate hypermatrix representing the n duration estimates for the p paths after removing each of the \bar{q} options from the plan at evaluation i , iteration j (n, m, \bar{q})
- $\underline{\mathbf{H}}_{ij}$ alternate version of the \mathbf{H}_{ij} vector for the \bar{q} options in the solution at evaluation i , iteration j (\bar{q})
- $\underline{\mathbf{S}}_{ij}$ alternate version of the \mathbf{S}_{ij} vector for the \bar{q} options in the solution at evaluation i , iteration j (\bar{q})
- $\underline{\mathbf{Z}}_{ij}$ alternate version of the \mathbf{Z}_{ij} vector for the \bar{q} options in the solution at evaluation i , iteration j (\bar{q})

In order to reduce notational complexity the matrix dimensions m , p , q , and \bar{q} are not shown with subscripts, but note that these dimensions may vary at the i evaluations as tasks (and expediting options applied to those tasks) are consumed and at the j iterations as matrices are trimmed (for computational efficiency).

The project data are used to create an initial matrix of 0-1 variables denoting the linkage of the q task time reduction options to the p tasks (\mathbf{G}_0), a vector of times remaining until the due date (\mathbf{T}_0) for the m paths to allow comparisons based on completed tasks at each evaluation, and both a diagonal time reduction matrix (\mathbf{E}_0) and cost vector (\mathbf{C}_0) for the q task time reduction options. Then a matrix of n time estimates for each of the p tasks ($\hat{\mathbf{A}}_0$) is generated and a matrix of p tasks by m paths of 0-1 variables denoting a task on a path (\mathbf{O}_0) is defined. In practice, the time estimates for the $\hat{\mathbf{A}}_0$ matrix would be randomly sampled from task distributions defined by field estimates with upper and lower limits. The method can accommodate any preferred task distribution method. In this research the individual task distributions were randomly sampled from sets of predefined uniform distributions per a task variability experimental factor setting. The time required to randomly sample the distributions and populate the $\hat{\mathbf{A}}_0$ matrix is trivial and expands linearly with the number of project tasks. The \mathbf{O}_0 matrix is developed from the results of a single-pass algorithm for enumerating the paths in a project network. The algorithm, which is similar to the longest path algorithm, requires a maximum of p evaluations and storage of the m resultant paths. Very simply, the algorithm involves the addition of a task at each node to all partial paths assembled at prior nodes. The performance of the algorithm is consistent with the polynomial time complexity function reported by Garey and Johnson [17] for application of the longest path algorithm to project networks. The number of paths is then reduced by an efficient matrix-based version of the original Van Slyke [13] paths reduction procedure using the following calculated vectors of path duration limits:

$$\mathbf{Min} = \boldsymbol{\alpha} \mathbf{O}_0 \tag{1}$$

$$\mathbf{Max} = \boldsymbol{\beta} \mathbf{O}_0 \tag{2}$$

Paths corresponding to a \mathbf{Max} vector value less than the maximum value in the \mathbf{Min} vector can be deleted. The premise behind the reductions is that paths where the sum of the task duration upper limits are less than the sum of the task duration lower limits of another path can never be

longest and therefore need not be evaluated. Finally, a three-step heuristic selection process is used to develop the buffering plan at each of the i evaluations.

The purpose of the first step of the heuristic is to generate some of the initial matrices and determine an initial solution. The first step begins with the recording of an initial maximum crash vector (\mathbf{X}_{i0}) denoting the remaining number of reductions available per option and the calculation of an initial matrix of n path time estimates by m paths ($\hat{\mathbf{D}}_{i0}$) as follows:

$$\hat{\mathbf{D}}_{i0} = \hat{\mathbf{A}}_i \mathbf{O}_i \quad (3)$$

Since each row of the $\hat{\mathbf{D}}_{i0}$ matrix represents the durations of all paths remaining after the Van Slyke reduction for one of the n sets of randomly-generated task duration estimates, each row with all column values less than or equal to the respective times in the \mathbf{T}_i vector signifies a successful outcome to the project. The proportion of these successes over the n sets of estimates equals the initial solution (λ_0), which is the probability of completing the project with only time buffers from pre-existing conditions. Then a matrix of potential path time reductions (\mathbf{R}_i) is determined as follows:

$$\mathbf{R}_i = (\mathbf{E}_i \mathbf{G}_i \mathbf{O}_i)' \quad (4)$$

Equation (4) essentially associates each of the expediting time reductions in the \mathbf{E}_i matrix to the tasks through the \mathbf{G}_i matrix and the paths through the \mathbf{O}_i matrix. Duplicating the resultant \mathbf{R}_i matrix n times in the first dimension generates an initial hypermatrix \mathbf{F}_{i0} of n replications by m paths by q task time reduction options. This hypermatrix can be thought of as a set of q reduction matrices in the same form as the original paths durations matrix ($\hat{\mathbf{D}}_{i0}$).

The second step is a greedy construction heuristic that adds task time reduction options to the plan one at a time based on the ratio of the projected benefit (increased probability of project completion) divided by the cost of the options in the plan. The second step begins with a duplication of the $\hat{\mathbf{D}}_{ij}$ matrix q times in the third dimension to generate a hypermatrix $\hat{\mathbf{B}}_{ij}$ of n replications by m paths by q task time reduction options at each of the j iterations. Then a work hypermatrix $\hat{\mathbf{W}}_{ij}$ can be defined as follows:

$$\hat{\mathbf{W}}_{ij} = \hat{\mathbf{B}}_{ij} - \mathbf{F}_{ij} \quad (5)$$

The work hypermatrix is comprised of q 2-dimensional matrices of n replications by m paths, each representing the projected path durations when implementing a task time reduction option. Alternately, the work matrix can be thought of as a set of identical $\hat{\mathbf{D}}_{ij}$ paths durations matrices each reduced by one of the q task time reduction options. As before, each row with all time estimates less than or equal to the respective times in the \mathbf{T}_i vector represents a successful project outcome. The proportion of the successes over the n replications for each of the q options, which denotes the results from the simultaneous simulations less λ_0 , is recorded in an \mathbf{S}_{ij} vector. An initial solution cost vector (\mathbf{H}_{0j}) is set equal to the \mathbf{C}_i vector and these results are used to determine the following performance vector (\mathbf{Z}_{ij}) of benefit-cost ratios.

$$\mathbf{Z}_{ij} = \frac{\mathbf{S}_{ij}}{\mathbf{H}_{ij}} \quad (6)$$

The option with the largest value in the \mathbf{Z}_{ij} vector (z_k) is added to the plan, the 2-dimensional portion of the work hypermatrix representing that option becomes $\hat{\mathbf{D}}_{i,j+1}$, the corresponding value in the \mathbf{S}_{ij} vector (s_k) becomes the next solution (λ_{j+1}), and every value of the solution cost vector (\mathbf{H}_{ij}) is increased by the cost of the selected option (h_k). Then the corresponding value in

the \mathbf{X}_{ij} matrix (x_k) is decreased by one, a copy of the corresponding slice of the \mathbf{F}_{ij} hypermatrix is entered into an alternate time reduction matrix ($\underline{\mathbf{F}}_{ij}$), and the cost of the selected option is set aside for later use in an alternate solution cost vector ($\underline{\mathbf{H}}_{ij}$). The \mathbf{F}_{ij} matrix slice and the \mathbf{H}_{ij} and \mathbf{X}_{ij} vectors are then trimmed for the k index if $x_k = 0$ and all other indices of the \mathbf{H}_{ij} vector that equal or exceed the available budget. This step iterates over j as long as options exist and $z_k > 0$.

After the second step ends, the third step of the heuristic attempts to improve the current plan by dropping and adding task reduction options. The third step begins with the calculation of an alternate work matrix ($\hat{\mathbf{W}}_{ij}$), as follows:

$$\hat{\mathbf{W}}_{ij} = \hat{\mathbf{B}}_{ij} + \underline{\mathbf{F}}_{ij} \quad (7)$$

By adding the task time reductions to the current solution hypermatrix ($\hat{\mathbf{B}}_{ij}$) this alternate work hypermatrix determines the best solution with one option dropped from the plan. Then the resultant benefits and performance vectors are evaluated similarly as to the work hypermatrix in the second step with each of the vectors replaced by an alternate version representing options currently in the solution. The initial alternate solution cost vector ($\underline{\mathbf{H}}_{ij}$) is equal to the sum of the costs of the options in the solution *less* a vector representing the costs of each task time reduction option. The option with the largest value in the $\underline{\mathbf{Z}}_{ij}$ vector is then dropped from the solution. After the third step ends the $\hat{\mathbf{D}}_{ij}$ and \mathbf{F}_{ij} matrices, and \mathbf{H}_{ij} and \mathbf{X}_{ij} vectors are updated; and the second step is repeated to determine which option to add to the solution. Evaluation i terminates when the dropped and added task time reduction options are the same.

Steps 1-3 are completed at each of the i evaluations and the resulting buffer location and expansion plans based on task time reductions are implemented through the next evaluation. Bregman [18] previously showed this heuristic selection process to generate near-optimal solutions on randomly-generated small test problems and efficiently solve the large-scale static task time reduction problems that must be completed at each evaluation. In this study, task time reductions are implemented at the initiation of the respective tasks and actual task durations are recorded at the completion of the tasks. The \mathbf{T}_i vector is updated at each of the i project evaluations for tasks that have been completed since the last evaluation. As a result, the effective times until the due date in the \mathbf{T}_i vector may vary during the project, but will converge at the completion of the project for the longest path(s).

The heuristic and the subsequent experiment were programmed using software freely available on the internet to allow the results to be easily replicated by others. Perl version 5.8.8 [19] was used as the general programming language so that the matrices could be manipulated using the Perl Data Language (PDL) module version 2.4.2 provided by Soeller [20] and the random numbers could be generated using the Hedden [21] automatic self-seeding interface to the Menon-Sen [22] implementation of the Matsumoto and Nishimura [23] Mersenne Twister random number generator. The PDL module is recommended by Christiansen and Torkington [24], is significantly faster than alternative matrix modules [25], and has been compiled for use with newer 64-bit computer processors. The Menon-Sen random number generator implementation performs well [25] on the Marsaglia [26] DIEHARD randomness tests and the Matsumoto and Nishimura Mersenne Twister on which it is based is considered state-of-the-art.

The complete paper (including results and references) is available upon request from the author.