An Integer Programming Method for CPM Time-Cost Analysis

A. P. Chassiakos, C. I. Samaras, D. D. Theodorakopoulos[1]

Abstract: Time and cost to complete a project is an important tradeoff problem in project planning and control. Existing methods have not provided an accepted solution in terms of both accuracy and efficiency. In an attempt to improve the solution process, a method is presented for developing optimal project time-cost curves based on CPM analysis. Using activity succession information, project paths are developed and duration is calculated. Following that, duration is reduced in an optimal way employing integer programming. Two alternative formulations are proposed which lead to corresponding algorithms, a progressive duration reduction and a direct reduction to the desired level. The first approach leads to suboptimal results but requires less computational effort than the second which, though, finds the optimal solution. The method has been successfully tested on a number of cases and results are presented to illustrate its application and demonstrate its merits.

keyword: critical path method (CPM), integer programming, project planning, time-cost tradeoff

1 Introduction

In project planning, a tradeoff problem between project duration and cost is encountered. In particular, project cost depends, among other factors, on the required (or desired) time to complete the project. Considering that an activity can generally be completed in a number of alternative ways, each of which is associated with particular duration and cost values, the objective is to select the appropriate execution option so that the project is completed by a desired deadline and in an optimal way, i.e., with the minimum cost. The analysis is repeated for any feasible project length and an optimal time-cost curve is developed for the project which can also be used for optimizing project duration.

The time-cost tradeoff problem has extensively been studied since 1960s and has been recognized as a particularly difficult combinatorial problem. Several solution schemes have been proposed, none of which was wholly satisfactory. They include linear, integer, or dynamic programming and other (heuristic) methods. Some of these methods provide exact solutions but with considerable computational effort. Other methods use simplified formulations to reduce resource needs but provide approximate solutions. A third category includes methods for project network decomposition to reduce time requirements.

The problem was initially modeled as either a linear or an integer programming one depending on whether a continuous tradeoff between activity duration and cost exists or only certain combinations are possible. Linear programming formulations have been proposed by Fulkerson (1961) and Perera (1980) assuming a continuous time-cost function which, however, may not be applicable to real life projects. Mixed-integer linear models have been developed to reflect the case where only a finite number of time-cost combinations are available [Shtub, Bard, and Globerson (1994)]. A number of researchers [e.g., Meyer and Shaffer (1965), Crowston (1970)] have proposed similar models with the primary focus on simplifying the problem by alternative formulations for the precedence constraints. Liu, Burns, and Feng (1995) used a combination of linear and integer programming to establish the lower bound of a project time-cost relationship. In general, application of mathematical programming techniques is computationally inefficient in large projects as they require solving a linear (integer) programming problem with numerous variables and constraints. On the other hand, simplified methods, although less complex, do not always reach the optimal solution.

Dynamic programming formulations have provided another modeling approach. More specifically, Robinson (1975) has presented a conceptual framework (but not an algorithm) based on dynamic programming for problem solution. Hindelang and Muth (1979) and, later, De, Dunne, Ghosh, and Wells (1995) proposed dynamic programming algorithms for the discrete time-cost case. Dynamic programming methods are computationally more efficient than linear or integer programming methods and, thus, more suitable for large projects. However, these methods may not obtain the optimal solution in certain cases. The need for an expert analyzer to set up an application in practice is an additional limitation of these methods.

Besides mathematical programming techniques, a number of methods or algorithms are reported in the literature. Feng, Liu, and Burns (1997) proposed using genetic algorithms to address time-cost optimization. The application of this algorithm showed that at least 95% of the optimal solutions were found. Siemens (1971) and Goyal (1975) proposed simple, heuristic methods to solve the problem. These methods are useful only for small networks and cannot guarantee finding the optimal solution. In addition, they require increased ef-

[1] Department of Civil Engineering, University of Patras, GR-26500 Patras, Greece.

fort to set up a large scale problem and computational time to solve it as they need to examine all project activities and paths. In a different direction, Panagiotakopoulos (1977) focuses primarily on problem simplification. In particular, a method was developed to decompose, where possible, a project network into non-overlapping segments and construct a time-cost function for each segment. Following network decomposition, subproblems can be solved employing any of the existing techniques. A similar problem reduction approach has been used in De, Dunne, Ghosh, and Wells (1995).

In summary, the main shortcomings of methods used for project time-cost analysis are that they result in large problems which require excessive effort to solve, formulations are complex and time-consuming to apply, and application is prone to errors. In addition, some methods make certain assumptions on activity time-cost form which limit their applicability.

This paper presents a method for developing optimal project time-cost curves. The proposed method results, in many cases, in smaller problem size compared to that of existing methods reducing, thus, resource requirements. In addition, the method leads to an elementary problem formulation which allows easy application. According to this method, the project is initially represented in terms of its paths and activities. Following that, integer programming is employed to determine the best combination of activity durations which results in a desired project length. Starting from the normal duration, the aim is to reduce this value by a certain amount with the lowest cost increase. Two alternative formulations are proposed leading to corresponding algorithms. The first is based on a progressive duration reduction which gives an approximate solution but has increased computational efficiency. The second attempts a single-step optimization, obtains the exact solution but requires increased effort. An example to illustrate the algorithm application is presented, the results are discussed, and the efficiency is compared to that of a previous method.

2 Problem description

Project planning and control has been based on project network representation and critical path method (CPM) analysis. For a given project, a network is drawn to depict the logical sequence of operations (activities). Any sequence of activities connecting the start and end points of the project determines a path. The sum of the estimated length of time required to carry out each activity in a path determines path duration. Among all paths, the one (or more) with the largest duration is known as the critical path and its duration determines the project length.

The duration considered for each activity corresponds to the value associated with the minimum cost. However, this (normal) duration often leads to unacceptable project lengths in view of external time constraints for project completion. In such cases, shortening project length is considered. In general, project acceleration (alternatively known as project crashing)

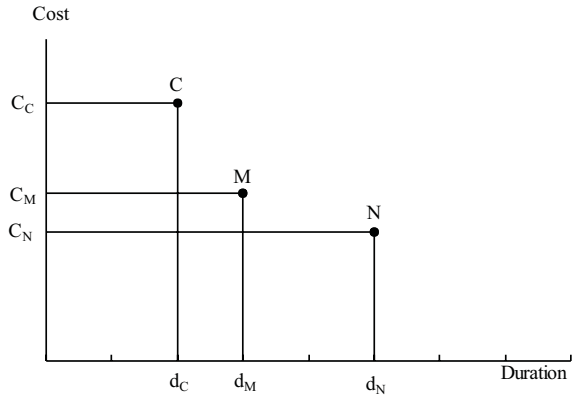

Figure 1 : A typical activity time - cost relationship

is achieved by reducing the duration of appropriate activities through the use of additional resources, overtime work, shift allowance, or by selecting different technologies to perform an activity. This is accomplished at the expense of an extra crashing cost. The objective of the analysis is then to obtain the desirable project duration at the lowest cost increase, i.e., to decide which activities to expedite and by what amount towards this objective. The analysis is generally extended to any possible project length leading to the development of a time-cost tradeoff curve which can also be used for optimizing project duration. In small projects crashing can be done manually, however, in projects with a large number of activities, computer algorithms may be the only alternative to problem solution.

An important factor in the analysis is the form of the time-cost relationship which is assumed for project activities. This factor considerably affects the problem formulation and the extend to which a method can realistically be applied. An activity generally presents a number of alternative time-cost combinations corresponding to various options to carry out the work. The time required to complete an activity under normal working conditions is known as the normal duration and corresponds to the minimum activity cost. Activity execution in a shorter than the normal time typically results in increased cost. The minimum feasible duration of an activity is known as the crash time and the corresponding cost as the crash cost.

Activity time-cost relationships may take a variety of forms. The most common relationship is a convex one which appears when operating limitations prevent an increasing resource application from showing a proportional return. Another type is a linear relationship where cost increases linearly as duration decreases. A relationship presenting a step-like cost increase at a specific duration may appear if shortening the activity duration below this point requires an incremental resource charge.

A time-cost curve may be continuous or discrete. A continuous relationship represents an activity that can be completed at

any time-cost combination along the curve. Continuous curves are usually approximated by piece-wise linear functions to establish cost slopes and simplify calculations. A discrete time-cost relationship is more appropriate than a continuous one to model actual engineering projects. An activity can be completed at distinct values of duration and each one is associated with a cost value (Figure 1). Point N represents normal execution, C execution in the shortest possible duration and M an intermediate time alternative. Feasible durations for an activity are not necessarily uniformly distributed within the range between normal and crash time (e.g., $d_N - d_M \neq d_M - d_C$). Activities which can be executed only in a specific way present a single time-cost operating point and do not have an effect on project crashing.

Many existing approaches assume a convex time-cost curve and has based development on this requirement. Although usually realistic, the assumption does not always hold. In the present analysis, any decreasing time-cost function is applicable. Outliers of such a function i.e., points with higher cost at higher duration than others can be eliminated in advance either manually or employing a simple algorithm as in Liu et.al. (1995).

3 The proposed method

The proposed method is based on developing a matrix with all project paths. To facilitate development, an appropriate tabular representation of the project network is employed. Based on the logical sequence in which activities are executed, a triangular matrix is formed where activities (rows) and their successors (columns) are inputted in a zero-one format. In particular, denoting activity number by j and activity code by $A(j)$, the matrix values are given as:

$$s(j_1, j_2) = \begin{cases} 1 & \text{if activity } A(j_2) \text{ is a successor of} \\ & A(j_1),\ j_2 > j_1,\ j_1 = 0, 1, 2, \ldots, n \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where n is the number of activities. Activities are numbered in such a way that, if $j > k$, $A(j)$ cannot precede $A(k)$. An initial zero-duration, zero-cost (dummy) activity $A(0)$ and a similar terminal activity $A(n+1)$ are introduced to assist the development process. The number of activities that follow activity j is denoted by $w(j)$ and equals the sum of values in the j-th row of the matrix, i.e.,

$$w(j) = \sum_{k=1}^{n+1} s(j,k), \quad j = 0, 1, 2, \ldots, n. \quad (2)$$

Based on activity successor information, the project matrix is developed. Each row corresponds to a path while each column represents an activity. Unit values are assigned to the appropriate matrix cell if an activity belongs to a specific path; zeros otherwise. The project matrix is developed in steps. Initially, a number of matrix rows (paths) equal to $w(0)$ are generated. Each row has a unit value in the cell corresponding to an activity that follows $A(0)$. Activity $A(1)$ is considered next. From the row that includes activity $A(1)$, a number of $w(1)$ paths (rows) are generated, one for each immediate successor of $A(1)$, and unit values are inserted in the appropriate cells. For activity $A(j)$, in general, all paths (matrix rows) that lead to this activity are considered. In each path, a number of $w(j)$ new paths are generated and successor activities are indicated with ones. The process finishes with the last project activity $A(n)$ which has no successor but the dummy activity $A(n+1)$. After developing the project matrix, dummy activities are not further needed and the corresponding columns are eliminated from the matrix.

The duration of each path is calculated as the sum of durations of the activities included in the path, i.e., the duration $D(i)$ of path i is given by

$$D(i) = \sum_{j=1}^{n} p(i,j)\, d(j), \quad i = 1, 2, \ldots, n_p, \quad (3)$$

where $d(j)$ is the duration of activity j, $p(i,j)$ is the (i,j)-th element of the project matrix (equals 1 if activity j belongs to path i; 0 otherwise), n is the number of project activities, and n_p is the number of paths. The array of project durations is attached to the matrix for easy reference. The project matrix is sorted in descending order of path duration and is augmented with information regarding activity durations and crashing costs. The longest path (or paths) of the network represents the critical path and its length determines the project duration. Since paths have been sorted, the project length is determined by $D(1)$.

The second part of the method constitutes the optimization process. More specifically, starting from the normal project length, the objective is to expedite the project in an optimal way. Two alternative formulations are used leading to corresponding algorithms (both algorithms include a module for project matrix development as described previously). Algorithm #1 employs a repetitive process to progressively shorten the project length by one time unit at each stage. In this respect, only critical paths need to be considered for the optimization. Further, to reduce project length by one time unit, it is sufficient that one critical activity in each path is crashed by at least the same amount. Among all alternative combinations of critical activities, the one resulting in the lowest total crashing cost is sought.

An integer programming (IP) formulation is employed to find the optimal crashing alternative. A zero-one variable $x(j)$ is defined for each activity to indicate whether this activity will be selected for crashing or not. The integer program is written as

$$\text{minimize} \quad \sum_{j=1}^{n} c(j)x(j) \quad (4)$$

subject to

$$\sum_{j=1}^{n} p(i,j)\,x(j) \geq 1, \quad i = 1,2,\ldots,n_{cp} \tag{5}$$

and

$$x(j) = 0 \quad \text{or} \quad 1, \tag{6}$$

where $c(j)$ is the required additional cost for reducing the duration of activity j to its next feasible value (referred below as crashing cost) and n_{cp} is the number of critical paths. The objective function (4) represents the cumulative crashing cost over all activities while the set of constraints (5) ensure that at least one activity in every critical path is selected for crashing. Activities that cannot be crashed may be assigned a high value of cost so that they will not be selected by the model. An alternative and computationally more efficient procedure is to exclude from the analysis activities that are or become non-compressible in the process. In this case, although some additional complexity is introduced to the algorithm, the computational time savings may be considerable.

At the end of each phase, the project length has been reduced by at least one time unit. The project matrix is then updated, the duration of paths which include crashed activities is accordingly decreased, new critical paths are identified, activity and path durations are calculated and updated activity crashing costs are inserted. The process is then repeated by running the updated IP model to find the appropriate combination of activities to expedite in the next phase.

An advantage of the progressive duration reduction approach is that analysis is confined to critical paths reducing, therefore, the computational effort. However, this method imposes a restriction on activity selection as crashing decision at a particular stage is dependent on decisions at previous stages. In other words, accuracy limitations may result from the inherent assumption that the least total crashing cost for the N-stage process will be obtained by adding together the least crashing costs in each of the N stages. Such assumption does not always hold and, as a result, an optimal solution is not guaranteed. Such inefficiency can be removed if crashing is performed directly from the normal to the target project duration.

The alternative formulation that is proposed here under Algorithm #2 requires examining all project paths with longer duration than the desired one (leading, thus, to an increased problem size compared to Algorithm #1). A zero-one variable $y(j,k)$ is defined for each activity to indicate whether activity j will be crashed by k time steps in order to achieve the desirable project duration. The integer program is written as:

$$\text{minimize} \sum_{j=1}^{n} \sum_{k=1}^{K(j)} c(j,k)\,y(j,k) \tag{7}$$

subject to

$$\sum_{j=1}^{n} \sum_{k=1}^{K(j)} p(i,j)\Delta d(j,k)\,y(j,k) \geq D(i) - D_d, \quad i = 1,2,\ldots,n_{cp}, \tag{8}$$

$$\sum_{k=1}^{K(j)} y(j,k) \leq 1, \quad j = 1,2,\ldots,n, \tag{9}$$

and

$$y(i,k) = 0 \quad \text{or} \quad 1 \tag{10}$$

where $K(j)$ is the number of crashing steps for activity j, $c(j,k)$ is the cost for crashing activity j for the first k steps, $\Delta d(j,k)$ is the duration reduction if activity j is crashed by the first k steps, $D(i)$ is the duration of path i, D_d is the desired (target) project duration, and n_{cp} is the number of "critical" paths, i.e., those with $D(i) > D_d$. The objective function (7) represents the total crashing cost. The set of constraints (8) ensure that each path is adequately reduced in length to conform to the desired project duration. Finally, the set of constraints (9) prevent activity crashing phases from being double-counted. To find the optimal crashing strategy at various project durations, the IP model attains the same form except that additional constraints (8) may be required according to "critical" paths and the constants in the right-hand side of constraints (8) are modified in accordance with each particular duration.

Before proceeding to algorithm application, some issues concerning the expected performance of the algorithms are discussed. The problem size is an important factor when evaluating algorithm efficiency. De. et.al. (1995) have found that the computational complexity of an exact solution approach would be exponential in the worst-case (i.e., the solution time would grow as an exponential function of the problem size). In fact, any solution to this problem presents rapidly increased resource requirements with the problem size and the proposed algorithm cannot escape of such rule. Assessing the size for the proposed formulation indicates that the number of zero-one variables included in the first model is at most equal to the number of project activities while the number of constraints equals the number of critical paths. The second model requires an increased number of zero-one variables which equals the number of possible crashing steps of all activities. The number of constraints is at most equal to the number of paths plus the number of activities that have three or more alternative time-cost options (i.e., more than one crashing step). As shown in Section 4, the proposed method is more efficient than previous methods in certain cases. This improvement mainly appears in small or moderate size project networks but efficiency is reduced in larger networks. The limitation results from the need to enumerate all network paths.

Another concern in project time-cost analysis is that methods are often complex, require significant time and effort to set up

Table 1 : Activity duration and cost data for example project: (a) raw data, (b) summary data

(a) raw data

Activity	Normal duration	Normal cost	Duration	Cost	Duration	Cost
A	6	68.0	5	78.0		
B	7	65.0	6	71.0	5	75.0
C	10	72.0	8	77.0	6	83.0
D	3	80.0				
E	9	102.0	7	114.0		
F	6	54.0	5	62.0		
G	8	85.0	7	92.0	6	101.0
H	5	40.0	4	48.0		
I	4	56.0	3	69.0		

(b) summary data

(Reduced duration, crashing cost)								
A	B	C	D	E	F	G	H	I
(1, 10.0)	(1, 6.0)	(2, 5.0)		(2, 12.0)	(1, 8.0)	(1, 7.0)	(1, 8.0)	(1, 13.0)
	(1, 4.0)	(2, 6.0)				(1, 9.0)		

a problem and are prone to errors. With the proposed method, matrix development is done automatically based on the primary project data. Project representation through the project matrix allows setting up the integer program easily as indicated in the application example in the following section. Further, the project matrix provides a simple and intuitive view of the project which, unlike previous methods, facilitates problem solution by hand computations in small networks. The simplicity of representation may also have an effect on future developments, in particular project decomposition to subprojects to increase efficiency or extension to other types of activity relationships, e.g., start-to-start.

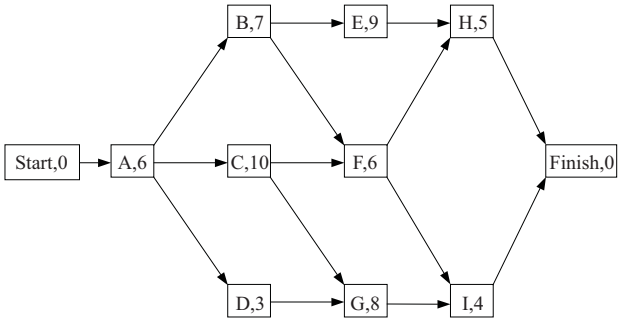

Figure 2 : Activity-on-node network for example project

4 An application example

The optimization algorithms have been applied to a number of test cases. Results of a preliminary evaluation indicate that the second approach leads to the best solution. The step-by-step duration reduction method usually finds the optimal solution. There are cases, however, that not all points on the time-cost curve are optimal. Such an example has been structured and presented below to illustrate the algorithm application.

The activity-on-node project network for the application example is depicted in Figure 2. In this representation, activities correspond to nodes while arrows indicate precedence relationships. Activity codes and durations are shown in the boxes. Alternative time-cost options for each activity are presented in Table 1a. Durations are assumed to be given in weeks and costs in some appropriate unit. The project consists of nine activities, eight of which can be crashed. In particular, activities B, C, and G present three alternative time-cost combinations, A, E, F, H, and I two alternatives while D can be executed in a specific way. Crashing is done at one-week intervals except for activities C and E that duration is reduced by two week intervals. Because a comparative type of analysis is performed, activity cost for normal duration is irrelevant. The useful data for the analysis are extracted and summarized in Table 1b. Time-cost functions are convex for any activity except B which presents a non-convex decreasing form.

Table 2 presents the activity successor matrix. The process for generating the project matrix is illustrated in Table 3 (some intermediate steps are not presented). Project paths, activity and path durations for normal activity execution, and activity crashing costs are presented in Table 4. Non-compressible activity D has been assigned a crashing cost of 100.0 and appears last in the project matrix. Paths have been sorted in decreasing order of their length. The normal project duration is 28 weeks as determined by the critical path A-C-G-I. To reduce the project length to 27 weeks, the critical path should be shortened. The IP model is written as (non-compressible activity D

Table 2 : Activity successor matrix

	Start	A	B	C	D	E	F	G	H	I	Finish
Start		1									
A			1	1	1						
B						1	1				
C							1	1			
D								1			
E									1		
F									1	1	
G										1	
H											1
I											1
Finish											

Table 3 : Illustration of project matrix development process

Path No	↓ Path \ Activity →	A	B	C	D	E	F	G	H	I
1	Start - A	1								
1	Start - A - B	1	1							
2	Start - A - C	1		1						
3	Start - A - D	1			1					
1	Start - A - B - E	1	1			1				
2	Start - A - B- F	1	1				1			
3	Start - A - C	1		1						
4	Start - A - D	1			1					
1	Start - A - B - E	1	1			1				
2	Start - A - B- F	1	1				1			
3	Start - A - C - F	1		1			1			
4	Start - A - C - G	1		1				1		
5	Start - A - D	1			1					
1	Start-A-B-E-H-Finish	1	1			1			1	
2	Start-A-B-F-H-Finish	1	1				1		1	
3	Start-A-B-F-I-Finish	1	1				1			1
4	Start-A-C-F-H-Finish	1		1			1		1	
5	Start-A-C-F-I-Finish	1		1			1			1
6	Start-A-C-G-I-Finish	1		1				1		1
7	Start-A-D-G-I-Finish	1			1			1		1

is not included in the model):

minimize

$$10x_A + 6x_B + 5x_C + 12x_E + 8x_F + 7x_G + 8x_H + 13x_I \quad (11)$$

subject to

$$x_A + x_C + x_G + x_I \geq 1 \quad (12)$$

$$x_A, x_B, x_C, x_E, x_F, x_G, x_H, x_I = 0, 1. \quad (13)$$

The solution (which can be derived by a commercial LP/IP computer program as, for instance, LINDO) is:

$$x_C = 1,\ x_A = x_B = x_E = x_F = x_G = x_H = x_I = 0. \quad (14)$$

The result indicates that activity C should be crashed to a duration of eight weeks at a cost of 5.0 units.

The project matrix is updated with new durations and costs as illustrated in Table 5 which corresponds to a 27-week project duration. Critical and non-critical paths that contain activity C have been shortened by two weeks. Path durations have been recalculated indicating that another path (A-B-E-H) has become critical. The cost for further crashing activity C from 8 to 6 weeks is 6.0 units. Solving the new IP problem indicates that activity B should be selected for crashing leading to a 26-week project duration. The updated project matrix is shown in Table 6. Following the same procedure, the project matrix for a 21-week duration is developed (Table 7). The project cannot be crashed any further since there is no feasible solution to the corresponding IP problem (i.e., no activity in path A-B-E-H can be shortened). Thus, the 21-week duration represents the minimum project length.

Table 4 : Project matrix for normal activity duration

	Activity j	A	B	C	E	F	G	H	I	D	Path
	Duration d(j)	6	7	10	9	6	8	5	4	3	duration
	Crashing cost c(j)	10.0	6.0	5.0	12.0	8.0	7.0	8.0	13.0	100.0	d(i)
	A-C-G-I	1		1			1		1		28
P	A-B-E-H	1	1		1			1			27
a	A-C-F-H	1		1		1		1			27
t	A-C-F-I	1		1		1			1		26
h	A-B-F-H	1	1			1		1			24
s	A-B-F-I	1	1			1			1		23
	A-D-G-I	1					1		1	1	21
	Cumulative cost			5.0							

Table 5 : Project matrix after the first crashing phase

	Activity j	A	B	C	E	F	G	H	I	D	Path
	Duration d(j)	6	7	8	9	6	8	5	4	3	duration
	Crashing cost c(j)	10.0	6.0	6.0	12.0	8.0	7.0	8.0	13.0	100.0	d(i)
	A-B-E-H	1	1		1			1			27
P	A-C-G-I	1		1			1		1		26
a	A-C-F-H	1		1		1		1			25
t	A-C-F-I	1		1		1			1		24
h	A-B-F-H	1	1			1		1			24
s	A-B-F-I	1	1			1			1		23
	A-D-G-I	1					1		1	1	21
	Cumulative cost		6.0	5.0							

Table 6 : Project matrix after the second crashing phase

	Activity j	A	B	C	E	F	G	H	I	D	Path
	Duration d(j)	6	6	8	9	6	8	5	4	3	duration
	Crashing cost c(j)	10.0	4.0	6.0	12.0	8.0	7.0	8.0	13.0	100.0	d(i)
	A-B-E-H	1	1		1			1			26
P	A-C-G-I	1		1			1		1		26
a	A-C-F-H	1		1		1		1			25
t	A-C-F-I	1		1		1			1		24
h	A-B-F-H	1	1			1		1			23
s	A-B-F-I	1	1			1			1		22
	A-D-G-I	1					1		1	1	21
	Cumulative cost		10.0	11.0							

Table 7 : Project matrix for minimum project duration

	Activity j	F	I	A	B	C	D	E	G	H	Path
	Duration d(j)	6	4	5	5	6	3	7	6	4	duration
	Crashing cost c(j)	8.0	13.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	d(i)
	A-C-G-I		1	1		1			1		21
P	A-B-E-H			1	1			1		1	21
a	A-C-F-H	1		1		1				1	21
t	A-C-F-I	1	1	1		1					21
h	A-B-F-H	1		1	1					1	20
s	A-B-F-I	1	1	1	1						20
	A-D-G-I		1	1			1		1		18
	Cumulative cost			10.0	10.0	11.0		12.0	16.0	8.0	

Table 8 : Optimal crashing strategy: (a) approximate method, (b) exact method

(a) approximate method

Project duration	A	B1	B2	C1	C2	E	F	G1	G2	H	I	Cumulative cost
27				•								5.0
26		•		•								11.0
25		•	•	•	•							21.0
24	•	•	•	•	•							31.0
23	•	•	•	•	•					•		39.0
22	•	•	•	•	•	•		•		•		58.0
21	•	•	•	•	•	•		•	•	•		67.0

(b) exact method

Project duration	A	B1	B2	C1	C2	E	F	G1	G2	H	I	Cumulative cost
27				•								5.0
26		•		•								11.0
25		•	•	•	•							21.0
24		•		•	•	•						29.0
23	•	•		•	•	•						39.0
22	•	•	•	•	•	•		•				50.0
21	•	•	•	•	•	•		•	•	•		67.0

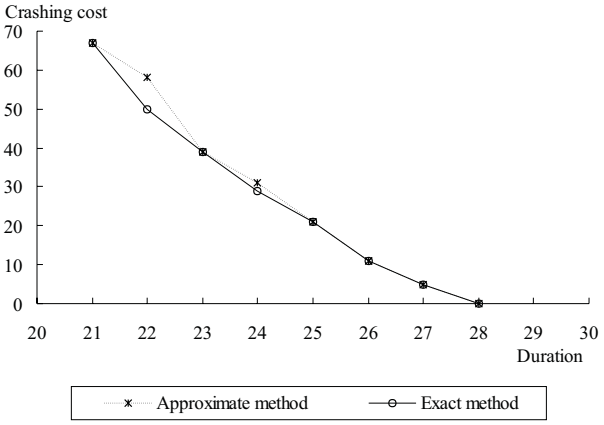

Figure 3 : Crashing cost as a function of project duration

Referring to the alternative direct reduction approach, the project matrix for normal execution is reconsidered (Table 4). For all target durations, the integer program presents a common structure except that as the desired project duration reduces, additional path constraints are inserted and the constants in the right-hand side of path constraints successively increase. The integer program is presented here for the indicative case of 23-week target duration. In the following relationships, y_{B1} refers to the first crashing step of activity B while y_{B12} to both steps together (similar notation applies to other activities). The problem is written as:

minimize

$$10y_{A1}+6y_{B1}+10y_{B12}+5y_{C1}+11y_{C12}+12y_{E1}+8y_{F1} + 7y_{G1}+16y_{G12}+8y_{H1}+13y_{I1} \quad (15)$$

subject to

$$y_{A1}+2y_{C1}+4y_{C12}+y_{G1}+2y_{G12}+y_{I1} \geq 5 \quad (16)$$

$$y_{A1}+y_{B1}+2y_{B12}+2y_{E1}+y_{H1} \geq 4 \quad (17)$$

$$y_{A1}+2y_{C1}+4y_{C12}+y_{F1}+y_{H1} \geq 4 \quad (18)$$

$$y_{A1}+2y_{C1}+4y_{C12}+y_{F1}+y_{I1} \geq 3 \quad (19)$$

$$y_{A1}+y_{B1}+2y_{B12}+y_{F1}+y_{H1} \geq 1 \quad (20)$$

$$y_{B1}+y_{B12} \leq 1 \quad (21)$$

$$y_{C1}+y_{C12} \leq 1 \quad (22)$$

$$y_{G1}+y_{G12} \leq 1 \quad (23)$$

$$y_{A1},\ y_{B1},\ y_{B12},\ y_{C1},\ y_{C12},\ y_{E1},\ y_{F1},\ y_{G1},\ y_{G12},\ y_{H1},\ y_{I1} = 0, 1. \quad (24)$$

The solution to this problem is obtained as:

$$y_{A1}=y_{B1}=y_{C12}=y_{E1}=1, \quad (25)$$

$$y_{B12}=y_{C1}=y_{F1}=y_{G1}=y_{G12}=y_{H1}=y_{I1}=0, \quad (26)$$

at a total cost of 39.0 units.

Table 8 summarizes the output of the alternative formulations within the range of feasible durations. In terms of notation, B1 and B2 refer to the first and second crash of activity B respectively. In the step-by-step duration reduction, any decision at a particular stage is dependent on decisions at previous

Table 9 : Problem size assessment

Algorithm	Normal variables	Zero-one variables	Constraints
#1 (1st step)	-	8	1
#1 (last step)	-	2	4
#2 ($D_d = 23$)	-	11	8
Liu, et.al. (1995)	9	20	42

stages. The consequence of this dependency may be illustrated with reference to activity B. In particular, following this approach project crashing to duration of 24 weeks includes B2 because this alternative was chosen at a previous stage (Table 8a). However, the best solution includes B2 for a 25-week duration but not for the 24-week one as indicated by the direct duration reduction method (Table 8b). The project time-cost curves derived by the proposed algorithms are graphically illustrated in Figure 3. Selected activities and, therefore, cost values are identical at most duration levels. Cost differences appear at durations of 22 and 24 weeks.

To obtain an efficiency indication in terms of resource requirements, the size of the integer programs are assessed and compared to a previous method with reference to the application example. A recently presented method that also employs integer programming is examined (Liu, Burns, and Feng 1995). Table 9 summarizes the results of the comparison. The proposed algorithms present a varying problem size depending on the crashing stage. For this reason, two cases are shown for Algorithm #1, one at the first and another for the last (infeasible) project crashing stage. For Algorithm #2, a typical case associated with a desired project duration of 23 weeks is included. The integer program in Liu et.al. (1995) has a constant size in all cases. The assessment results, although in favor of the proposed algorithms, should be considered as indicative and cannot be generalized for any network type and size. This is because it is difficult to establish a robust method to compare requirements among different formulations and, thus, comparisons can be only made with regard to particular applications.

5 Conclusions

The time-cost tradeoff problem which is encountered in project planning has been recognized as a particularly difficult combinatorial problem. Many different methods employing mathematical programming, heuristics, or genetic algorithm applications have been proposed, none of which is completely satisfactory considering accuracy, efficiency, and applicability. In an attempt to improve efficiency and applicability, a method for assessing minimum cost options for executing a project at various project lengths is presented. The project is described through a matrix where all paths are tabulated with respect to activities employing a zero-one representation. Given the time-cost relationships for project activities, integer programming is employed to choose among all feasible activity durations those which limit project duration at a desired level with the lowest possible cost. Two alternative formulations are proposed (and incorporated in corresponding algorithms) to reduce project duration from its normal to the desired level in an optimal way. The first performs a step-by-step project duration reduction. In each step, only critical paths are considered which are compressed by one time unit. This approach accelerates the solution process but may lead to sub-optimal results. The second formulation performs a direct reduction to the desired duration level, leads to the optimal solution but requires solving a larger integer program than before. Any discrete time-cost relationship for project activities can be modeled and this makes the method valuable for real life project applications. The proposed approach is intuitive and can facilitate future extensions to other activity relationships besides the default finish-to-start which has been assumed in this analysis.

References

Crowston, W. B. (1970): Decision cpm: Network reduction and solution. *Operational Research Quarterly*, vol. 21, pp. 435–452.

De, P.; Dunne, E.; Ghosh, J.; Wells, C. (1995): The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research*, vol. 81, pp. 225–238.

Feng, C.; Liu, L.; Burns, S. A. (1997): Using genetic algorithms to solve construction time-cost trade-off problems. *ASCE Journal of Computing in Civil Engineering*, vol. 11, no. 3, pp. 184–189.

Fulkerson, D. R. (1961): A network flow computation for project cost curves. *Management Science*, vol. 7, pp. 167–178.

Goyal, S. (1975): A note on a simple cpm time-cost tradeoff algorithm. *Management Science*, vol. 21, pp. 718–722.

Hindelang, T. J.; Muth, J. F. (1979): A dynamic programming algorithm for decision cpm networks. *Operations Research*, vol. 27, pp. 225–241.

Liu, L.; Burns, S. A.; Feng, C. (1995): Construction time-cost trade-off analysis using lp/ip hybrid method. *ASCE Journal of Construction Engineering and Management*, vol. 121, no. 4, pp. 446–454.

Meyer, W. L.; Shaffer, L. R. (1965): Extending cpm for multiflow project time-cost curves. *ASCE Journal of the Construction Division*, vol. 91, pp. 45–67.

Panagiotakopoulos, D. (1977): Cost-time model for large cpm project networks. *ASCE Journal of the Construction Division*, vol. 103, pp. 201–211.

Perera, S. (1980): Linear programming solution to network compression. *ASCE Journal of the Construction Division*, vol. 106, pp. 315–326.

Robinson, D. R. (1975): A dynamic programming solution to cost-time tradeoff for cpm. *Management Science*, vol. 22, pp. 158–166.

Shtub, A.; Bard, J.; Globerson, S. (1994): *Project Management: Engineering, Technology, and Implementation*. Prentice Hall International Editions.

Siemens, N. (1971): A simple cpm time-cost tradeoff algorithm. *Management Science*, vol. 17, pp. 354–363.