ONDERZOEKSRAPPORT NR 9337

# Optimal Procedures for the Discrete Time/Cost Trade-off Problem in Project Networks

by

Erik L. DEMEULEMEESTER

Willy S. HERROELEN

Salah E. ELMAGHRABY

D/1993/2376/37

# OPTIMAL PROCEDURES FOR THE DISCRETE TIME/COST TRADE-OFF PROBLEM IN PROJECT NETWORKS

**Erik L. DEMEULEMEESTER**
**Willy S. HERROELEN**
Katholieke Universiteit Leuven
Department of Applied Economics
Debériotstraat 36, B-3000 Leuven (Belgium)

**Salah E. ELMAGHRABY**
North Carolina State University at Raleigh
Operations Research and Industrial Engineering
P.O. Box 7913
Raleigh
NC 27695-7913
U.S.A.

# OPTIMAL PROCEDURES FOR THE DISCRETE TIME/COST TRADE-OFF PROBLEM IN PROJECT NETWORKS

## ABSTRACT

We describe two algorithms, based on dynamic programming logic, for optimally solving the discrete time/cost trade-off problem (DTCTP) in deterministic activity-on-arc networks of the CPM type, where the duration of each activity is a discrete, nonincreasing function of the amount of a single nonrenewable resource committed to it. The first algorithm is based on a procedure proposed by Bein, Kamburowski and Stallmann for finding the minimal number of reductions necessary to transform a general network to a series-parallel network. The second algorithm minimizes the estimated number of possibilities that need to be considered during the solution procedure. Both procedures have been programmed in C and tested on a large set of representative networks to give a good indication of their performance, and indicate the circumstances in which either algorithm performs best.

**Key words:** resource constrained projects, time/cost trade-off, dynamic programming.

## GLOSSARY OF TERMS AND SYMBOLS

| | |
|---|---|
| AoA: | Activity-on-Arc |
| BaB: | Branch-and-Bound |
| BKS: | Bein, Kamburowski and Stallmann |
| CI: | Complexity Index of the network |
| CPM: | Critical Path Method |
| dag: | directed acyclic graph |
| DP: | Dynamic Programming |
| DTCTP: | Discrete Time/Cost Trade-off Problem |
| index activity: | the activity whose duration is to be fixed at its different modes |
| mode: | a duration-cost (or duration-resource) pair |
| s/p: | series/parallel |
| $P_x$: | path in the search tree leading to node x |
| t: | generic designation of a terminal leaf in the search tree |
| $T(P_x)$: | the union of all activities identified by the branches of $P_x$ |

## 1. INTRODUCTION

The specific resource allocation problem addressed in this paper is the discrete time/cost trade-off problem (DTCTP) in project networks of the CPM type, under a single nonrenewable resource. We treat three different objectives: the minimization of the project duration under fixed resource availability; the minimization of the total resource consumption to achieve a target project completion time; and the construction of the efficient time/resource profile over the feasible project durations. The problem is known to be strongly NP-hard, see De et al. (1992). Our aim is to devise efficient procedures that achieve the optima of the stated objectives. This is pertinent from at least three points of view. First, increased computer capabilities bring many problems to within the realm of feasibility: they can be solved to optimality within reasonable time. Optimal solutions provide a benchmark for heuristic and approximating procedures, which has been missing hitherto in this field. Second, optimal procedures are often used as modules in larger problems - for instance, the optimum for a single resource may be used in a BaB procedure for securing the optimum for several resources. Third, optimal procedures constitute the foundation for approximating procedures; and we hope that ours shall serve in this capacity.

The methodology used is that of dynamic programming (DP) - and its modifications - and branch-and-bound (BaB). The paper is self-contained; though we assume knowledge of the rudimentary concepts of activity nets and the methodologies used.

### 1.1 PROJECT REPRESENTATION

The specification of a project is assumed to be given in activity-on-arc (AoA) notation by a directed acyclic graph (dag) $D = (N,A)$ in which N is the set of nodes, representing network "events", and A is the set of arcs, representing network "activities". We assume, without loss of generality, that there is a single start node 1 and a single terminal node n, $n=|N|$. Since D is acyclic, we assume that its nodes are topologically numbered; i.e., $i < j$ whenever there exists an arc joining i to j. The resulting graph is a two-terminal directed acyclic graph, which will be referred to as a *1-n dag*. The duration $y_a$ of activity $a \in A$ is a discrete, nonincreasing function $g_a(x_a)$ of the amount of a single resource allocated to it; i.e., $y_a = g_a(x_a)$. The pair $y_a,x_a$ shall be referred to as a "*mode*", and shall be written as: $y_a(x_a)$. Thus an activity that assumes four different durations according to four possible resource allocations to it shall be said to possess four modes. In this paper we consider three possible objective functions for the DTCTP (see also De et al. 1993). For the first objective function we specify a limit R on the total availability of a single nonrenewable resource type. The problem is then to decide on the vector of activity durations $(y_1,...,y_m)$, $m = |A|$, that completes the project as early as possible under the limited availability of the single nonrenewable resource type. If we denote an activity *a* by its end nodes i and j and if we let $t_i$ denote the (earliest) realization time of node i, then the problem can be formulated as follows:

$$\text{minimize} \quad t_n \tag{1}$$

subject to

$$t_1 = 0 \tag{2}$$

$$t_i + g_{ij}(x_{ij}) <= t_j \qquad \text{for all } (ij) \in A \tag{3}$$

$$\sum_{(ij)\in A} x_{ij} <= R \tag{4}$$

In the objective function (1) we minimize the realization time of the single terminal node n, where equation (2) indicates that the project is started at time 0. Constraint set (3) is used to satisfy the precedence constraints, while constraint (4) indicates that the limit on the resource availability cannot be violated.

A second objective function reverses this problem formulation: now we specify a limit T on the project length and we try to minimize the resource usage. Using the same notation as for the previous formulation, this problem may be stated as follows:

$$\text{minimize} \quad \sum_{(ij)\in A} x_{ij} \tag{5}$$

subject to

$$t_1 = 0 \tag{6}$$

$$t_i + g_{ij}(x_{ij}) <= t_j \qquad \text{for all } (ij) \in A \tag{7}$$

$$t_n <= T \tag{8}$$

In this formulation constraints (6) and (7) are identical to constraints (2) and (3), but now constraint (8) specifies that the maximal project length T cannot be violated. The objective function (5) minimizes the sum of the resource usage over all activities.

For the third and final objective function we have to compute the complete time/cost trade-off function for the total project, i.e., in the case of the DTCTP all the efficient points (T, R) such that with a resource limit R a project length T can be obtained and such that no other point (T', R') exists for which both T' and R' are smaller than or equal to T and R:

$$\text{minimize } [\ \sum_{(ij)\in A} x_{ij} \mid \underline{t}_n <= T <= \bar{t}_n\ ] \tag{9}$$

## 1.2 LITERATURE REVIEW

The early contributions to what we shall call "the basic time-cost trade-off problem" in CPM networks assumed ample resource availability and tried to minimize the total project cost subject to precedence constraints and lower and upper bound constraints on the activity durations. Numerous optimizing procedures have been presented in the literature. When the function g(.) is linear or piece-wise linear, the basic time-cost trade-off problem has been optimally solved by the well-known Fulkerson algorithm (Fulkerson 1961). When g(.) is strictly convex and unbounded from above, the optimum has been characterized by Elmaghraby (1968), who also gave a procedure for its determination. Lamberson & Hocking (1970) presented an optimal algorithm for differentiable convex cost-duration functions and applied decomposition theory to reduce the amount of computation. For the special case where the cost-duration functions are quadratic and convex the Fulkerson labeling algorithm has been extended by Kapur (1973). For the case of quadratic costs, Elmaghraby & Salem (1982) have derived a specific algorithm for the situation of cost curves which are continuously differentiable and for the case of a discontinuous derivative. Elmaghraby & Salem (1981) treated the general convex case and seek a satisficing answer, in which the project is compressed to a desired completion time with prespecified tolerable relative error. When g(.) is concave, the problem can be solved by the BaB procedure of Falk & Horowitz (1972).

For the case where g(.) is discrete (still nonincreasing), Meyer and Shaffer (1965) and Patterson & Harvey (1979) offered an integer programming approach where the objective function is to minimize the cost of completing the project incurred by compressing the project duration to length T or shorter, in the absence of resource constraints. The last two authors also highlight the equivalence with *Decision CPM* as introduced by Crowston & Thompson (1967, 1970).

Subsequent contributions recognized the important role played by limited resources in shaping the final result. Butcher (1967) presents dynamic programming formulations for the problem stated above in eqns. (1) to (4) when the project networks are pure series and pure parallel. When g(.) is arbitrary (still nonincreasing) a dynamic programming procedure was suggested by Robinson (1975) for the same problem in general project networks. Tavares (1990) considers the case where the sequence in which the activities are considered is prespecified and offers a solution, also using dynamic programming. Hindelang & Muth (1979) presented an algorithm for the DTCTP within the more general context of "Decision-CPM". This algorithm may be interpreted in the simpler context of networks of interest to us, as was done by De et al. (1992), who also demonstrated its inadequacy to give the optimum of any of the three objectives defined above. De et al. propose a "correction" which is tantamount to enumeration over the "common activities". As we demonstrate below, this is a highly inefficient way to approach the problem, as was also noted by De et al.. Elmaghraby (1992) presents a dynamic programming procedure in which he exploits the possibilities of "node reduction", explained in section 4.1 below, when the network is not "completely reducible". The same paper also presents an elementary approximating procedure.

Talbot (1982) offers a two-stage optimal solution methodology for the resource-constrained project scheduling problem with time/resource trade-offs. The objective is to minimize project duration subject to finish-start precedence constraints and constraints on both nonrenewable and renewable resources. Computational experience is reported on a series of test problems, which indicates that the methodology can provide optimal solutions to small problems (about 10-activity). Drexl (1991) considers the problem of assigning resources to jobs in order to find the least expensive schedule under which a project is completed by the given time horizon. Each resource can be assigned to one job at a time, and for a limited number of periods only. The author offers a hybrid branch-and-bound/dynamic programming algorithm with a Monte Carlo type heuristic upper bounding technique as well as various relaxation procedures for determining lower bounds.

More recently, Neumann & Zhan (1993) propose an elaborate combination of heuristics for minimizing the project completion time under fixed availability of several resources, when precedence among the activities is specified by minimal and maximal time lags between the start of successive activities. This is a more general setting than discussed here. They report good solution times with projects of up to 1000 activities. They do not report, however, on the performance of their procedure (relative to the optimum), nor do they address the time/cost problem in either its continuous or discrete forms. Nair, Prasad, and Aneja (1993), on the other hand, do address the time/cost problem in its continuous form under the hypothesis of piece-wise linear functions for all the activities. Although this problem was efficiently solved by an elegant procedure in 1961 (see Fulkerson 1961), the authors interpreted the problem as optimization under the "dual-criteria" of minimizing both cost and duration. They approached the construction of the complete functional relationship between project duration and its cost from the point of view of enumerating paths and determining the "min-cost envelope" of activities (or paths) in series and in parallel. We fail to understand the justification of such an approach.

### 1.3 ORGANIZATION OF THE PAPER

In this paper we describe two procedures which may be used to solve the three objective functions of eqns.(1), (5) and (9). The last part of both procedures is the same: it consists of either the determination of the complete function (objective (9)) or of a BaB procedure for the first two objective functions (objectives (1) and (5)). The first part of both procedures, however, is different. For the first algorithm we exploit the basic reduction arguments introduced by Bein, Kamburowski and Stallmann (BKS) (1992), which minimizes the number of "node reductions" and thus the number of levels in the BaB tree of the second part. For the second algorithm we apply a BaB strategy in order to determine the minimal estimated number of leafs that need to be considered in the BaB tree of the second part.

This paper is organized as follows. In section 2 we clarify the methodologies used to solve the DTCTP. In section 3 we introduce the notion of a reduction plan and show how a reduction plan can be used in order to solve the three different objective functions of the DTCTP. Section 4 deals with two different ways of constructing a good reduction plan. Section 5 discusses the computational experience that has been gained with these two procedures. A last section then offers overall conclusions and suggestions for further research.

## 2. SOLUTION METHODOLOGIES

We present the theory underlying the optimal time/cost trade-off procedures. First, it will be established that series/parallel reducible projects (*s/p reducible*, for short) can be optimized through a succession of series and parallel optimizations adapted from a serial and parallel merge algorithm developed by Rothfarb et al. (1970) and by Frank et al. (1971). Projects which cannot be optimized by series/parallel reductions are called "*s/p irreducible*". In terms of the terminology used by the recently emerging techniques for *modular decomposition*, these are project networks which contain so-called *neighbourhood modules* (Buer and Möhring 1983, Müller and Spinrad 1989, Schwarze 1989, Sidney and Steiner 1986). It will subsequently be demonstrated that irreducible projects can be optimized using a procedure for fixing the resource allocation of a subset of the activities in the project network. We shall refer to such "fixings" as *optimal* - following the procedure of BKS - in the sense that they require the minimal number of "node reductions", and not in the sense of requiring the minimal number of "leafs", as shall be detailed below. An alternate series-parallel conversion procedure has been suggested by De et al. (1993).

## 2.1 TIME/COST TRADE-OFFS IN SERIES/PARALLEL NETWORKS

### 2.1.1 Dynamic Programming

A project that is composed solely of a set of activities in series has a length equal to the sum of the individual activity durations. It is easy to show that the optimal allocation of resources is secured by the sequence of extremal equations of dynamic programming:

$$f_m(r) = \min_{x_m} [g_m(x_m)], \qquad 0 <= x_m <= r; \; 0 <= r <= R \qquad (10)$$

$$f_i(r) = \min_{x_i} [g_i(x_i) + f_{i+1}(r - x_i)], \qquad 0 <= x_i <= r; \; 0 <= r <= R \quad i = m-1,..,1 \qquad (11)$$

The optimum is secured when $f_1(R)$ is determined. This serial optimization can be viewed as a series arc reduction: if a = (i,j) is the unique arc into j and b = (j,k) is the unique arc out of j then these two arcs in series are replaced by a single arc c = (i,k).

For a project that is composed of a set of activities in parallel, the activities $a_1,...,a_m$ can be performed in parallel subject only to resource availability. The project duration now equals the maximum activity duration, and the optimal allocation is secured by the sequence of extremal equations of dynamic programming:

$$f_m(r) = \min_{x_m} [g_m(x_m)] \qquad 0 <= x_m <= r; \; 0 <= r <= R \qquad (12)$$

$$f_i(r) = \min_{x_i} [\max \{g_i(x_i), f_{i+1}(r - x_i)\}] \qquad 0 <= x_i <= r; \; 0 <= r <= R \quad i = m-1,..,1 \qquad (13)$$

The optimum is secured when $f_1(R)$ is determined. This parallel optimization process can be viewed as parallel arc reduction: two or more parallel arcs $a_1$, ..., $a_m$ leading from i to j are replaced by a unique arc a = (i,j).

Two theoretical results (due to Robinson 1975) can be invoked to minimize computational effort: (i) The function $f_1(R)$ is nonincreasing in R, and (ii) As the availability of the resource r increases, the value of the optimum allocation $x_i$, viewed as a function of r and therefore written as $x^*_i(r)$, is given by

$$x^*_i(r+\Delta) = x^*_i(r) \text{ if } f_{i+1}(r-x^*_i(r)) > g_i(x^*_i(r))$$

and $$x^*_i(r) + \Delta \text{ otherwise}$$

A project that can be optimized through a succession of series and parallel optimizations (reductions) as indicated by eqns. (10)-(13) is said to be *series/parallel reducible* (*s/p reducible*, for short). A project which cannot be thus optimized is called *s/p irreducible*. Obviously a project may be s/p reducible without being composed solely of activities in series or in parallel. Figure 1 gives a simple example and the steps of its reduction.

Figure 1. A s/p reducible dag and DP recursion equations

2.1.2 Serial and Parallel Merge Operations

Instead of using dynamic programming, we accomplish the series/parallel reductions by applying the serial and parallel merge operations developed by Rothfarb et al. (1970) in the context of the optimal design of off-shore natural-gas pipeline systems, and by Frank et al. (1971) in the context of the optimal capacity assignment in centralized computer networks. Each activity has an associated duration vector such that the k*th* component is the activity duration arising from a choice of the k*th* smallest resource allocation to that activity. Similarly, each activity has an associated cost vector such that the k*th* component is the resource allocation associated with the k*th* largest duration for the activity. The values of the elements of cost are in increasing order and those of duration in decreasing order. The two vectors taken together will be called an *"activity array"*, and an element $y_a(x_a)$ of this array has been referred to as a *mode*.

The procedures are best illustrated by a small example. Consider first the parallel activities shown in Figure 2 together with their corresponding activity arrays. The parallel merge operation now proceeds as follows. A testing block is set up as shown in Table 1. Each activity array being merged has a column in the testing block as indicated. If the index in a column is set to k, then the duration and cost entries in that column are the k*th* components of the list. Initially the indices are set to 1. The procedure locates the largest entry in the duration row of the testing block. In our example this occurs in the first column of the testing block and the entry is shown in bold-face type. We enter the bold-face duration entry and the sum of the cost entries of the testing block in a new list. Since no better resource allocation for activity 2 is possible with activity 1 at its current allocation, we promote the index in the activity 1 column to 2 and continue the process by choosing the duration entry in the second column of the testing block. The process terminates when the largest entry of the duration row of the testing block occurs in a column whose index has been promoted to its maximum value. Further promotions of the other indices would correspond to partial assignments of greater cost and no possible savings in duration. Each entry in the final new array represents an assignment of resources to the activities. For the example problem of Figure 2, the maximum number of possible partial assignments is 3*3 = 9. However, the parallel merge technique will produce an array with at most 3+3-1 = 5 components, one from the original testing block and one each from the testing blocks resulting from a maximum of 4 index promotions. This is precisely the maximum number of possible *outcome modes*. The new activity array can be viewed as the duration and cost vectors of an equivalent activity which replaces those activities whose arrays were merged.

Figure 2. Example of parallel merge

Table 1. Testing block

Next, the serial merge procedure can also be illustrated on a small example. Consider activities in series as depicted in Figure 3. Assuming the same resource-duration functions for the two activities as for the parallel case shown in Figure 2, we now have to evaluate 3*3 = 9 candidates as each of the three alternatives of activity $a_1$ will form a candidate with each of the three alternatives of activity $a_2$. The matrix in Table 2 shows the nine corresponding duration-cost components.

Figure 3.  Example of serial merge

Table 2. Duration-cost components

Each duration-cost component is obtained by simply adding the corresponding components of the two activity arrays. The new activity array is formed by consecutively selecting the candidate having the smallest duration component from this matrix. In case of ties, we select the candidate with the smallest cost component. However, some of the candidates can be eliminated from the new activity array by a simple dominance rule. A candidate which has a cost component greater than or equal to the cost component of an already selected candidate, can be dominated and is not entered in the final new activity array. For the example of Figure 3, the new activity array looks as follows:

| | |
|---|---|
| duration | 14, 15, 16, 17, **17**, 19, **19**, 20, 22 |
| cost | 25, 20, 19, 14, **16**, 10, **13**, 8, 4 |

The candidates indicated in bold can be eliminated due to the dominance rule.

## 2.2 The Optimization Problem For A General dag

### 2.2.1 Total and Partial Enumeration

If a project is s/p irreducible, then a different solution strategy is needed in order to solve the discrete time/cost trade-off problem. One possible method, which we call "*total enumeration*", consists of enumerating all possible combinations of duration/cost assignments for each activity. If activity i has $m_i$ different modes, then the total number of possible combinations is equal to the product of these $m_i$ possibilities over all activities. It is obvious that this number grows exponentially, becoming quickly intractable for even modest networks.

Early in the development of activity networks researchers recognized the impossibility of total enumeration, and the search started for reducing the number of activities that are enumerated. It immediately became obvious that the prime candidates are the so-called "common" activities - i.e., activities that are common to more than one path from 1 to n; see, for instance, Elmaghraby (1989) and the references cited therein. We refer to such endeavors as "*partial enumeration*". Unfortunately, it it easy to come across project nets whose common activities are of the same order of magnitude as the total number of activities in the project, and no saving in effort is realized.

### 2.2.2 Optimal Fixing

A far more efficient method for s/p irreducible networks consists of enumerating the cost assignments for a limited number of activities, preferably the minimum number, such that the resulting network becomes s/p reducible. We refer to this method as "*optimal fixing*", which was described in Elmaghraby (1992) and used in Elmaghraby et al. (1989), and which is based on the fundamental contribution of BKS (1992). Here the number of combinations that need to be considered is equal to the product of the $m_i$ possibilities over this minimal number of activities, which is typically orders of magnitude smaller than for complete enumeration, or even for enumeration over the set of "common activities". We shall refer to this set of arcs to be fixed as the "*index activities*". Both the method of BKS and a new method for determining which activities need to be fixed will be explained more thoroughly in Section 4.

## 3. THE REDUCTION PLAN

A *"reduction plan"* consists of all actions that have to be performed on a network in order to reduce the network to one single arc, including the determination of which activities need to be fixed. As soon as such a plan is constructed it is quite easy to obtain a solution to all of the objective functions that we proposed in the *Introduction*.

In a reduction plan each action consists of three elements:

- the first element describes the construct that has to be applied;
- the second specifies the activities on which this construct is applied;
- and the third specifies what the resulting activity is called.

The only exception to the rule is the EVALUATE construct, which is applied to a single activity that represents the totally reduced project.

All AoA networks can be reduced to a single arc (1,n) by using the following five constructs:

1. SERIES : apply the serial merge as described in section 2.1.2;
2. PARALLEL : apply the parallel merge as described in section 2.1.2;
3. REDUCE1 : fix one activity and then apply a serial merge, adding the cost of the fixed activity to the second;

   **N.B.1**: In the reduction process we shall adopt the convention that whenever action 'reduce1' is used for the first time the corresponding activity shall be fixed at its most expensive mode. This will be beneficial when applying the BaB procedure for the other two objectives.
4. REDUCE2 : fix one activity and then apply a serial merge, without adding the cost of the fixed activity.

   **N.B.2**: By using two constructs for the "reduce" operation in serial merge we avoid double counting the cost of an activity during the reduction process.

   **N.B.3**: We shall also adopt the convention that a series or parallel operation shall be implemented as soon as it is feasible.
5. EVALUATE : used to compare the solution for a totally reduced network with the best solution found.

In the remainder of this section we will use the example of Figure 4 to demonstrate both the complete enumeration and the BaB methods, and to identify the dominance rules that can be applied for the various objective functions. BKS define the *reduction complexity* as the minimum number of node reductions sufficient (along with series and parallel reductions) to reduce a two-terminal acyclic network to a single edge. We adopt the reduction complexity as our definition of the *complexity index CI*. The network of Figure 4 is of CI=3, which means that the minimum number of activities to fix is three (i.e., the index activities). How we arrived at such determination shall be explained more fully when we explain the BKS approach; see section 4.1 below and *Appendix 1*. As will become evident there are usually many ways to accomplish such fixings. We have opted to fix activities $a_2$, $a_3$ and $a_{12}$ *in this order*. Moreover, we shall construct the complete time/cost trade-off function (objective (9)) for the project using optimal fixing.

Figure 4. Example network

**OBJECTIVE OF EQUATION (9)**

A possible reduction plan for the network may appear as the first three columns of Table 3 (see also Figure 5). The last column gives the consequence of the action in the form of the array of the newly generated activity, when the "index activities" are fixed at their *shortest (hence most expensive)* duration.

Figure 5. The steps of reduction of Figure 4

Table 3. A reduction plan and its consequences

The first action in the reduction plan, for instance, specifies that activity $a_2$ is to be fixed at one single mode (namely 3(5)), then serially merged with $a_6$ while including its cost, which results in activity $a_{15}$ with array {8(6),7(7)}. Now that the cost of activity $a_2$ has been added to $a_6$, it is not to be added to any other activity with which $a_2$ is serially merged (namely $a_7$); whence the second action involving $a_2$ specifies REDUCE2. Continuing in this fashion as detailed in Table 3 and Figure 5 we obtain the array for the totally reduced project, designated as $a_{31}$, given in Table 3 opposite action 17. As this is the first activity array that we have obtained for $a_{31}$, the action 'EVALUATE 31' only consists of saving this activity array for future use.

Recall that each of the three activities that were fixed assumed its most expensive value, with the last being $a_{12}$. In all, we must evaluate $m_2$*$m_3$*$m_{12}$=8 realizations. This is accomplished by reducing the costs in the reverse order of the selection of the activities. Upon backtracking to the last 'REDUCE1' construct, which is {REDUCE1; [12, 15]; 23}, we fix activity 12 to the next most expensive mode with a duration(cost) assignment of 6(4). Performing all subsequent actions we obtain in step 17 an activity array for $a_{31}$ as follows:

21(39), 20(42), 19(45), 18(46), 16(49), 15(58),14(60)

The 'EVALUATE 31' step now combines this activity array with the one that was previously saved (activity $a_{31}$ of Table 3) to obtain:

21(39), 20(42), 18(45), 16(48), 15(52), 14(57), 13(60), 12(68), 11(71)

This activity array is then saved for future use in the backtracking process. The solution tree for the complete enumeration is drawn in Figure 6, where under each leaf of the tree (i.e., a node where an 'EVALUATE' action is reached) the activity array for $a_{31}$ is shown. The duration(cost) assignments in bold then constitute the complete time/cost trade-off function for the example project:

21(34), 18(40), 17(43), 16(45), 15(49), 14(54), 13(57), 12(65), 11(69)

Figure 6. The optimal fixing tree

### OBJECTIVE OF EQUATION (1)

In order to solve for the first objective optimally, we basically use the same backtracking procedure that was described for objective (9), but now we introduce two dominance rules.

(i) The first is *resource-based*: if fixing an (index) activity at a certain mode causes the already assigned cost (i.e., the sum of the cheapest cost assignments for all activities plus the cost that was added by fixing some activities at a more expensive cost) to exceed the limit R on the total availability of the resource, then backtracking can take place. This dominance rule is especially useful when the limit R is rather small.

(ii) The second dominance rule is *critical path-based*: if the critical path length of the project, using for the index activities their fixed durations and for all other activities the durations of the most expensive modes, is larger than the shortest feasible project length found during the backtracking procedure, then backtracking can take place. This dominance rule is more applicable when the limit R is rather large.

To illustrate, assume that for the example project of Figure 4 the limit R is fixed at a value of 38. Then the following procedure can be followed to determine the minimal project length within this resource constraint. Using all cheapest modes for the activities, the project cost is 34 and using the most expensive modes, the length of the critical path is 11. When executing the action {REDUCE1; [2, 6]; 15}, the cost of $a_2$ is fixed at a value of 5, which is 3 units more than the cost of its cheapest mode. Therefore every solution that will be computed from this node will have a cost of at least 37 (34 + 3 for fixing activity 2). This is still within the availability of the nonrenewable resource type and thus no backtracking takes place, based on the first dominance rule. As the most expensive mode is chosen for $a_2$, the critical path length does not increase and thus remains at a value of 11. On fixing $a_3$ at a cost of 3, which is 2 more than its cheapest cost assignment, the already assigned cost increases to 37 + 2 = 39. As this cost is larger than the limit of 38, backtracking can take place from this node. Fixing $a_3$ at a cost of 1 (its cheapest cost assignment, such that the already assigned cost remains at 37) and fixing $a_{12}$ at a cost of 7, the already assigned cost increases to 37 + 3 = 40, which is again more than the limit of 38: during the backtracking $a_{12}$ is fixed at a cost of 4, which is its minimal cost, and thus the already assigned cost remains at 37. Executing all further actions until the 'EVALUATE 31' action, the shortest project length that can be found within the resource limit of 38 is a length of 21 at a total cost of 37. During the remaining backtracking procedure the first dominance rule can be applied only once more, while for a limit R of 38 the second dominance rule cannot be applied. Therefore the branch-and-bound procedure for this example project visits only four out of the eight leafs that were visited during the complete enumeration procedure for the total time/cost trade-off function for the project. The best solution that is found during the search procedure has a project length of 21 at a total cost of 34: remark that during the solution procedure we are not only looking for the shortest project length, but also for the cheapest cost among all solutions with this shortest project length.

If, however, the limit R on the total availability of the nonrenewable resource type is 72, then in the first leaf of the search tree (after fixing $a_2$, $a_3$ and $a_{12}$ at their most expensive modes) a feasible project length of 11 is found at a cost of 71. As we know that the critical path length of the project with all activities performed in their most expensive mode is 11, no shorter project length can be found during the solution procedure. During the backtracking procedure only one more leaf will be visited, where a project length of 11 is found at a cost of 69, again securing the minimal cost for the minimal project length.

### OBJECTIVE OF EQUATION (5)

In order to solve the second objective function optimally, basically the same two dominance rules can be applied. However, now the comparison for the first dominance rule is against the cheapest feasible cost found for the project, while for the second dominance rule it is against the specified limit T on the project length.

## 4. CONSTRUCTING A GOOD REDUCTION PLAN

In the previous section we have seen how a reduction plan can be used in order to optimally solve the different objective functions of the discrete time/cost trade-off problem. In this section we present two different methods for constructing a good reduction plan. The first method is based on a procedure by Bein, Kamburowski and Stallmann (BKS) (1992) for determining the minimum number of index activities. The second method minimizes the estimated number of possible combinations of activity modes that have to be considered. These two methods will be explained in the sequel of this section.

### 4.1 The Determination of the "Index Activities"

This section gives a heuristic argument for the optimal fixing procedure adopted in our development. Recall that we have coined the name "index activities" to designate the set of activities to be fixed. There is one-to-one correspondence, however, between "fixing the duration of an activity" and "reducing" (i.e., eliminating) a node in a *dag*; (see Colby & Elmaghraby 1984). Indeed, fixing $a_2$ in step 1 above (at whichever value we may select) simply adds its duration to $a_6$ and $a_7$, which is tantamount to "reducing" node c. It no longer exists; and one has the replacement activities $a_{15}$ and $a_{16}$ which link node a directly with nodes f and g, respectively; see Figure 5.

Therefore the question may be posed alternatively in terms of the "minimal node reduction", the so-called "*complexity index (CI)*" of the project network, which is the question resolved by BKS. We have borrowed the word "index" from the definition of the CI to designate the activities to be fixed. The reader should be alerted, however, to the fact that while the *nodes* to be reduced are uniquely defined at the *outset*, the corresponding *index activities can only be defined as the reduction progresses due to the sequential nature of the reduction process and the fact that a number of s/p reductions may follow any particular node reduction*. As a consequence, the arc to be fixed may be a "*composite arc*" that is the result of these intervening s/p reductions. This distinction plays an important role in the construction of our search procedure, as shall be explained below under Plan 2.

The BKS approach is composed of two major steps: the first constructs the "complexity graph" of the given project network (easily accomplished from standard "dominator tree" arguments); and the second determines the minimal node cover of this complexity graph. The key element in their construction is that the complexity graph is *directed and acyclic*, whence its minimal node cover can be easily secured by a simple (i.e., polynomially bounded) "maximum flow" procedure. Also note the possible "embedded" nature of arc fixing, which is caused by the *sequential* nature of node reduction. *Appendix 1* gives a more detailed description of the BKS procedure and illustrates its application to the project network of Figure 4. There, it can be seen that the CI of the project is indeed 3 as indicated above.

### 4.2 THE DETERMINATION OF THE MINIMUM COMPUTATIONAL EFFORT

The node reduction scheme presented above yields the minimum number of node reductions. Such a scheme, however, may not minimize the computational effort required in reducing the network because we should consider not only the *number of node reductions* but also the *order* in which the reductions are performed (oftentimes there is a choice), as well as the *total number of leafs* which have to be evaluated. These additional factors may have a dominant effect on the computation time, as evidenced by our experimentation. In the enumerative scheme adopted we have termed a complete set of resource allocations to all the "index activities" in the project a "*leaf*". In this section we describe a procedure for generating a reduction plan that aims at minimizing the total number of leafs evaluated, which is equivalent to minimizing the computing effort.

The impact on the computation time of the total number of modes evaluated may be illustrated by the "interdictive graph" of Figure 7. It should be clear from the arguments presented above that this network can be reduced to a single arc by reducing either node 2 or node 3. Reducing node 2 is equivalent to fixing the duration of $a_1$, while reducing node 3 corresponds to fixing the duration of $a_5$. The smallest computational effort will result from conditioning on that activity which has the smallest number of modes to be evaluated in executing the reduction plan. This will result in the smallest number of leafs to be visited in the search tree during the execution of the reduction plan. If for example, $a_1$ has 10 modes and $a_5$ only 2, the computational requirements would be reduced to a minimum if the reduction plan would fix the duration of $a_5$. This would lead to only 2 leafs visited during the backtracking process.

Figure 7. The interdictive graph

The procedure we suggest for generating the reduction plan performs a *depth-first search* on a search tree, which is best illustrated by the example of Figure 8. The procedure is characterized by a few distinguishing rules which shall be enunciated as the application to the example proceeds.

The root node of the search tree, *node 0*, corresponds to the original network to be reduced. Every other node in the search tree corresponds to a partially or completely reduced network (at every node all possible series or parallel merges are performed). Thus, node 1 corresponds to the partially reduced network which is obtained after reducing node e, while node 8 corresponds to the partially reduced network which is obtained after reducing node b; etc..

Figure 8. Example network and its search tree

*Branching* from a parent node corresponds to fixing the duration of an activity in the partially reduced network associated with the parent. Several heuristic branching rules may be used. The procedure respects the following two rules:

(i) *the highest numbered node which is eligible for reduction in the parent net is selected for branching,*
and,

(ii) *the duration of the corresponding network activity is fixed to its "median mode",*
where the "median mode" of activity i is the $\lceil m_i/2 \rceil$ 'th mode, where $\lceil k \rceil$ means "the smallest integer larger than or equal to k".

The heuristic rationale of these two rules is as follows. In branching from the highest numbered node eligible for reduction, we hope to pave the way for the application of the fathoming rule based on node duplication to be discussed below as early as possible in the search process. By fixing the duration of an activity to its median mode, we hope to fix the activity to its most representative duration.

In the network associated with the source node of the search tree of Figure 8, node *b* has a unit in-degree and node *e* has a unit out-degree. Both nodes are eligible for reduction. The procedure will select the highest numbered eligible node for branching (rule (i)), *e* in this case. The duration of $a_{11}$ is fixed at its median mode. Executing the network reduction yields the partially reduced network corresponding to node 1 in the search tree. In it, node *b* has a unit in-degree, and node *d* has a unit out-degree. Both nodes are eligible for reduction. Selecting the highest numbered node for branching, *d* in this case, leads into node 2 of the search tree. In the associated net, nodes *b* and *c* are eligible for reduction. Again selecting the highest numbered node for branching, *c* in this case, yields the completely reduced network associated with terminal node 3 in the search tree.

Let $P_x$ denote the unique path in the search tree which leads into node x. For each node x in the search tree, except the root, let $T(P_x)$ denote the union of all activities identified by the branches of $P_x$, and let v(x) denote the number of corresponding arc fixings in the set $T(P_x)$. For instance, in the tree of Figure 8, $T(P_2) = \{e,d\}$, meaning that node 2 in the search tree was reached through reducing nodes *e* and *d*, in that order. The corresponding index activity for node *e* is $a_{11}$, which has $m_{11}$ modes. Suppose it is fixed at mode $y_{11}(x_{11})$, which is the "median mode" for activity $a_{11}$. Activities $a_9 + y_{11}$ and $a_{10}$ may now be reduced to a single activity; call it $a_{12}$ whose modes may be different from either of its constituent activities. We have that $v(2) = m_{11} * m_{12}$. Let "t" be the generic designation of a "terminal" leaf in the search tree. The elements of $T(P_t)$ determine a reduction plan; i.e., the sequence of activity duration fixings required to reduce the original network to a single arc. For instance, $T(P_3) = \{e,d,c\}$, which fully determines the node reductions or, equivalently, activity fixings, to be performed in the reduction plan: reduce node *e* (fix $a_{11}$), followed by node *d* (activity $a_{12}$ in the net of node 1), followed by node *c* (activity $a_{13}$ in the net of node 2 which is the result of the parallel reduction of activity $a_7 + y_{11}$ and $a_8$). Clearly, v(t) gives a reasonably good estimate of the number of fixings required for a complete reduction of the original project net following path $P_t$, for any terminal node t. The aim now is to find that path (reduction plan) which results in the minimum estimated number of leafs to be evaluated when executing the reduction plan.

An obvious estimate on the total number of leafs to be evaluated in the "completion" of path $P_x$ is v(x) itself, given by,

$$v(x) = [ \prod_{i \in T(P_x)} m_i ]$$

Let U be the incumbent upper bound on the required number of modes to be evaluated (initially, $U = \infty$). For each terminal node *t* in the search tree for which $v(t) < U$, the upper bound is updated as $U = v(t)$.

A *node x* in the search tree is *fathomed* when either of three eventualities occur:

(1) we have reached a terminal node, x=t;

(2) v(x) >= U, whence the node is dominated;

(3) the node duplicates a previously enumerated alternative, whence the node is redundant.

*Backtracking* occurs when a node is fathomed. At termination we have the optimal path (with minimum estimated number of leafs). The detailed algorithmic steps may now be described as follows.

*Step 1. Initialization.*

Set the level in the search tree, $q = 0$. Create the root node $\rho = 0$ to correspond to the original project network D, reduce it as much as possible and put $U = \infty$.

*Step 2. Branching.*

a. Set $q = q + 1$. Determine the set of nodes eligible for reduction; call it the set $E_q$.
If $E_q$ is empty, save the reduction plan, update $U = v(t)$ and go to step 4 (backtrack).

b. Select among the nodes in $E_q$ the highest numbered node, denoted by $e^*$, for branching.

c. Create node $\rho = \rho + 1$ and check for dominance: if the path to node $\rho$, $P_\rho$, has appeared in a previous path of the search tree, then set $E_q = E_q - \{e^*\}$ and go to step 4 (backtrack); else, go to step 2d.

d. Reduce node $e^*$: determine the corresponding network activity i to be fixed, update $v(\rho) = LB(q\text{-}1) * m_i$ and fix the duration of i to its median mode. Let $E_q = E_q - \{e^*\}$ and reduce the network as much as possible.

*Step 3. Lower bound calculation.*

Compute the lower bound $LB(q) = v(\rho)$. If $LB(q) >= U$, go to step 4 (backtrack); else, go to step 2a.

*Step 4. Backtracking.*

If the branching level $q = 0$, then STOP.
If $E_q$ is empty, set $q = q - 1$ and repeat step 4; else, go to step 2b.

## 5. COMPUTATIONAL RESULTS

The procedure for executing a reduction plan has been programmed in IBM C Set/2 running under OS/2 for an IBM PS/2 with 486 processor operating at 25 MHz (or compatibles). The reduction plan is constructed either on the basis of the minimum number of reductions as explained in section 4.1, called *Reduction Plan 1* below (or simply *Plan 1*), or on the basis of the minimum estimated number of leafs as explained in section 4.2, called *Reduction Plan 2* (or simply *Plan 2*).

While it is easy to conceive of the number of leafs under Plan 2 - which follows standard BaB terminology, the determination of the number of "leafs" under Plan 1 requires some explanation. Note that if there are k *independent* index activities, and each possesses $m$ modes, then the total number of leafs is $m^k$ - the same as the total number of enumerations required. Note the emphasis on the index activities being *independent*, in the sense that their sequential fixing does not involve a previously-fixed index activity (or activities). As is well known, the index activities are *not necessarily independent*, especially for project nets with large CI's, in which case the number of leafs (i.e., enumerations) shall depend on the number of modes resulting from the intervening series/parallel reductions that follow the fixing of any index activity. In our experimentation we have kept track of the number of leafs generated under both plans.

Experiments 1 through 5 constructed the complete optimal time/cost function over the feasible range of project completion times; only Experiment 6 treated objective (1) subject to the single resource availability constraint. This latter was varied between 0 and 88, which is the maximum required by all activities in the project.

### 5.1 EXPERIMENT 1

The procedure has been tested on the 6 project nets of Figure 9. Each network has been executed 7 times with the number of modes for each activity varying from 4 to 10. The computational results are shown in Table 4. The table entries have the following format:

| | |
|---|---|
| a | b |
| c | d |

where,

a = number of leafs in the search tree visited by Plan 2
b = number of leafs in the search tree visited by Plan 1
c = CPU time in seconds for constructing and executing Plan 2
d = CPU time in seconds for constructing and executing Plan 1

Figure 9. The six sample networks of Experiment 1

Table 4. Computational results of Experiment 1

As expected, for each network the required computation time increases with increasing number of modes per activity. For the first three networks, all of CI=2, both reduction plans require the same number of search tree leafs to be visited for each of the 7 replications, while the CPU time required to construct and execute Plan 1 is generally less than the CPU time for Plan 2, though not by much. The gain of Plan 1 over Plan 2 is typically a small fraction of a second.

The results for networks 4, 5 and 6 are different, however. Network 6 requires the greatest computational effort. This was to be expected, given the complexity of the network structure (CI = 4), and its larger number of nodes and activities. On both networks 4 and 6 Plan 2 generally outperforms Plan 1 in both the number of leafs visited and the CPU time required. This effect is more pronounced for network 6: for 10 modes, Plan 1 evaluates some 18,000 leafs in the search tree in 44.82 seconds of CPU time, while Plan 2 visits only 10,632 leafs in a CPU time of 27.46 seconds. For network 5, Plan 1 outperforms Plan 2 when the activities have 8, 9 and 10 modes by 0.05, 0.99 and 0.60 seconds; or 1.3%, 12.1% and 3.4%; respectively.

In general, the results of the experiment confirm the dominant role played by the CI of the network as a factor in determining the computational effort expended.

### 5.2 EXPERIMENT 2

The same 6 networks as in Experiment 1 are investigated but with the number of modes for each activity determined by drawing from the discrete uniform distribution in the range [1,10]. Each network is solved for 10 different replications. Table 5 gives the computational results. For networks 4, 5 and 6, Plan 1 always requires the same or more leafs to be visited in the search tree than Plan 2. In 13 out of 60 experiments (approximately 20%), however, the CPU time required to do so is somewhat smaller than the CPU time required by Plan 2. The same remarks made above are equally applicable here, with the maximum gain being 0.11 seconds (in network 6 replication 1). However, when Plan 2 consumes less time than Plan 1 it improves significantly on it, indicating that the BaB procedure has been successful in pruning the search tree: the maximum gain is 4.29 seconds (occurring in network 6 replication 6). Again, the dominant role of the CI as a determining factor in the computational effort is evident.

Table 5. Computational results of Experiment 2
Each activity has a random number of modes determined from U(1,10).

### 5.3 Experiment 3

This experiment was conducted on 4 networks adapted from Erenguç et al. (1993); see Figure 10. There are two 12-node networks with 21 and 24 activities and CI's equal to 4 and 6, respectively, and two 20-node networks with 37 and 42 activities and CI's equal to 8 and 12, respectively. For all four networks each activity had two modes and each network was solved for 10 different replications. Table 6 exhibits the results. Interestingly enough; for the 12-node networks both reduction plans visit the same number of leafs in the search tree, resulting in the required computation times for Plan 1 being smaller, as to be expected. However, the two 20-node networks perform differently: in 18 out of 20 experiments Plan 2 requires a significantly smaller number of leafs to be visited, but at the price of much higher CPU-times. These higher computation times are caused by the fact that the BaB procedure to construct the reduction plans for these problems already required some 14 to 15 seconds.

Figure 10. The four networks of Experiment 3
The solid arcs give the smaller networks (of 21 and 37 arcs); the dotted arcs give the expanded networks (of 24 and 42 arcs).

Table 6. Computational results of Experiment 3
Each activity has two randomly generated modes; 10 replications.

### 5.4 Experiment 4

This experiment deals with the same networks used in experiment 3, but now each activity possesses 3 modes of operation. The results are given in Table 7. For the two 12-node networks, both reduction plans visit the same number of leafs with a smaller CPU-time required for Plan 1. For the two 20-node networks, however, Plan 2 clearly outperforms Plan 1 in terms of the number of leafs visited. For the CPU-time, however, the result is not that clear: for the network with 37 arcs, the time spent in the BaB procedure for constructing the reduction plan is not offset by a comparable gain in the second part. For the network with 42 activities, however, the construction of the reduction plan by the BaB procedure is clearly beneficiary.

Table 7. Computational results of Experiment 4
Each activity has three randomly generated modes; 10 replications.

### 5.5 EXPERIMENT 5

In our desire to extend the experimentation to a more representative sample of project nets, we randomly generated seven 11-node networks using the network generator described in Demeulemeester et al. (1993), see Figure 11. The number of activities ranges from 15 to 45, in steps of 5 activities. The CI of these nets varied between 1 and 7. The number of modes for each activity was generated from a uniform distribution U(1,10). For an activity with m modes, m different numbers between 1 and 100 were randomly generated and put in increasing order. These numbers were used as the cost values for the different modes. Each network was solved for 10 different replications. For replications i=1,...,10, the duration of the last mode of an activity was randomly chosen from the interval [1, 4*i-1]. For every previous mode, the duration was equal to the duration of the next mode plus a number randomly chosen from the same interval [1, 4*i-1]. Table 8 gives the results. Except for the 11-node 15-activities network in which Plan 1 equals or outperforms Plan 2 relative to the time of execution (albeit by small fraction of a second), the results show Plan 2 outperforming Plan 1 in 45 out of 60 experiments, with the average gain ranging from a minimum of 0.53 sec (34.4%) in the net 11/20 to a maximum of 420.93 sec (123.4%) in the net 11/35.

Figure 11. The incidence matrices of the seven networks of Experiment 5

Table 8. Computational results of Experiment 5
The number of modes of each activity is randomly drawn from U(1,10); 10 replications.

### 5.6 EXPERIMENT 6

To achieve some degree of continuity of thought, this experiment deals with the network of 11 nodes and 20 activities originally treated by Elmaghraby (1992), and depicted in Figure 12. It has CI=6. Each activity has 4 discrete execution modes. First, the complete time/cost trade-off function for the project was calculated (the reduction plan being constructed by Plan 1) and the procedure visited a total of 4096 leafs, requiring 1.65 seconds of computation time. Then the same reduction plan was used, but now the resource availability was considered to be fixed. The procedure that is described in section 3 for the objective of equation (1) was applied on this problem for varying resource availabilities and the results are given in Table 9. The format of this table is different from that of the previous tables, and requires some explanation. The resource availability was varied between 0 and 88 units in steps of one unit, and tabulated as follows: the row and column headings give levels of resource availabilities, with the rows varying in steps of 10 units and the columns providing the 1 unit step. Each cell has two entries: the top entry is the number of leafs visited in the search tree, and the bottom entry is the required CPU time in seconds. The objective is to minimize the project duration subject to the single (nonrenewable) resource constraint equal to the sum of the row and column headings. For instance, consider the cell in the second row and the fifth column: for a resource availability equal to 10 + 4 = 14 units, a total of 422 leafs were visited in the search tree, in a CPU time of 0.22 seconds. The cheapest execution mode for each activity has a cost of 0, which explains why the first entry in the Table is 0, corresponding to the resource constraint of 0 + 0 = 0. The maximal resource utilization is the sum, over the activities, of the maximal resource utilization of each activity, which is 88. This explains why the last entry in the Table corresponds to the resource constraint 80 + 8 = 88. (The last cell of the Table has one unit of redundant resource.)

Table 9 may be viewed as the tabulation of two dependent variables, viz, the number of leafs enumerated, and execution time, as functions of the third independent variable; viz, the resource availability. These relations are best viewed in graphical form as shown in Figure 13. It is evident that both dependent variables increase to a maximum then decrease to their respective minimal values (which are: 1 in the case of leafs, and 0.00 in the case of time - which implies almost negligible time). This stands to reason since at the two extremes of resource availabilities there is little choice - hence no decisions (or very few) to make - while "in the middle of the availability range" there are significantly more alternatives to enumerate, and consequently decision making is more involved. Incidentally, this result confirms the conjecture made by Elmaghraby & Herroelen (1980) in their discussion of the measurement of network complexity.

The small number of leafs for smaller values of the resource constraint is mainly due to the pruning effect caused by dominance rule (1). Dominance rule (2) is mainly responsible for the small number of leafs for the entries corresponding to the larger values of the resource availabilities.

Figure 12. The network of Experiment 6: 11 nodes, 20 activities, CI=6

Table 9. Computational results of Experiment 6

Figure 13. Variation of the execution time and number of leafs with resource availability; Experiment 6

## 6. CONCLUSIONS

We have responded to the following question: can one determine, in a "reasonable" amount of time, the optimal solution to the discrete time/cost trade-off problem (DTCTP) under limited (nonrenewable) resource availability for an arbitrary project network? The answer is a definite yes. We proposed two procedures, both based on dynamic programming logic. The first algorithm is based on a procedure proposed by Bein, Kamburowski and Stallmann for finding the minimal number of node reductions necessary to transform a general network to a series-parallel network. The second algorithm minimizes the computational effort in enumerating alternative modes through a BaB search tree. The suggested procedures were extensively tested through Monte Carlo experimentation on a variety of networks drawn from the literature or generated specifically for this study. The results are most encouraging: for projects of up to 20 nodes and 45 activities, with the complexity index of individual networks varying between 2 and 12, the time required never exceeded about 7 minutes. Furthermore, we recommend the use of Plan 2 for projects with a large number of activities and high CI.

In the process of responding to the above question we have also confirmed the dominant role played by the CI in determining the time to resolve the problem. For instance, as the CI increases from 8 to 12 the average time to solve the almost same network increases from about 28 sec. to about 400 sec! (See Experiment 4.)

This study may be extended in at least three directions: The first is to consider more than one resource; the second is to consider renewable resources and a mixture of nonrenewable and renewable resources; and the third is to consider alternative objective functions, such as the optimization of a value function that depends on "due dates" of certain milestone events in the project.

## REFERENCES

Aho, A.V., J.E. Hopcroft and J.D. Ullman (1975), *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts.

Bein, W.W., J. Kamburowski and M.F.M. Stallman (1992), Optimal Reduction of Two-Terminal Directed Acyclic Graphs, *SIAM Journal on Computing*, 21(6), 1112-1129..

Buer, H. and R.H. Möhring (1983), A Fast Algorithm for the Decomposition of Graphs and Posets, *Mathematics of Operations Research*, 8(2), 170-184.

Butcher, W.S. (1967), Dynamic programming for project cost-time curves, *Journal of the Construction Division, Proceedings of the ASCE*, 93, 59-73.

Colby, A.H. and S.E. Elmaghraby (1984), On the Complete Reduction of Directed Acyclic Graphs, OR Report No. 197, Graduate Program in Operations Research, North Carolina State University, Raleigh NC, 27695-7913.

Crowston, W.B.S. (1970), Decision CPM: Network Reduction and Solution, *Operational Research Quarterly*, 21(1), 435-450.

Crowston, W.B.S. and G.L. Thompson (1967), Decision CPM: A Method for Simultaneous Planning, Scheduling and Control of Projects, *Operations Research*, 15(3), 407-426.

De, P., E.J. Dunne, J.B. Ghosh, and C.E. Wells (1992), Complexity of the Discrete Time-Cost Tradeoff Problem for Project Networks, Tech. Report, Dept. MIS and Dec. Sci., University of Dayton, Dayton, OH 45469-2130.

De, P., E.J. Dunne, J.B. Gosh, and C.E. Wells (1993), The Discrete Time/Cost Trade-Off Problem Revisited, Working Paper 93-04, Dept. MIS and Dec. Sci., University of Dayton, Dayton, OH 45469-2130.

Demeulemeester, E., B. Dodin and W. Herroelen (1993), A Random Activity Network Generator, *Operations Research*, 41(5), 972-980.

Drexl, A. (1991), Scheduling of Project Networks by Job Assignment, *Management Science*, 37(12), 1590-1602.

Elmaghraby, S.E. (1968), The Determination of Optimal Activity Durations in Project Scheduling, *Journal of Industrial Engineering*, 19, 48-51.

Elmaghraby, S.E. (1989), The Estimation of Some Network Parameters in the PERT Model of Activity Networks: Review and Critique, Chapter 1, Part III in Advances in Project Scheduling, R. Slowinski & J. Weglarz, eds., Elsevier Publishing Co.

Elmaghraby, S.E. (1992), Resource Allocation via Dynamic Programming in Activity Networks, *European Journal of Operational Research*, 64, 1-17.

Elmaghraby, S.E. and W.S. Herroelen (1980), On the Measurement of Network Complexity, *European Journal of Operational Research*, 5, 223-234.

Elmaghraby, S.E., J. Kamburowski and M.F.M. Stallman (1989), On the Reduction of Acyclic Digraphs and Its Applications, OR Report N° 233, Graduate Program in Operations Research, North Carolina State University at Raleigh, Raleigh.

Elmaghraby, S.E. and A. Salem (1981), Optimal Linear Approximation in Project Compression, OR Report N° 171, NC State University at Raleigh, Raleigh.

Elmaghraby, S.E. and A. Salem (1982), Optimal Project Compression Under Convex Functions I and II, *Appl. Management Sci.*, 2, 1-39.

Erenguç, S.S., S. Tufekci and C.J. Zappe (1993), Solving Time/Cost Trade-off Problems With Discounted Cash Flows Using Generalised Benders Decomposition, *Naval Research Logistics*, Vol. 40, 25-50.

Falk, J.E. and J.L. Horowitz (1972), Critical Path Problems with Concave Cost-Time Curves, *Management Science*, 19(4), 446-445.

Ford, L.R. and D.R. Fulkerson (1962), *Flows in Networks*, Princeton University Press, Princeton, New Jersey.

Frank, H., I.T. Frisch, R. Van Slyke and W.S. Chou (1971), Optimal Design of Centralized Computer Networks, *Networks*, 1(1), 43-58.

Fulkerson, D.R. (1961), A Network Flow Computation for Project Cost Curves, *Management Science*, 7(2), 167-179.

Hindelang, T.J. and J.F. Muth (1979), A Dynamic Programming Algorithm for Decision-CPM Networks, *Operations Research*, 27, 225-241.

Kapur, K.C. (1973), An Algorithm for the Project Cost-Duration Analysis Problem With Quadratic and Concave Cost Functions, *IIE Transactions*, 5(4), 314-322.

Lamberson, L.R. and R.R. Hocking (1970), Optimum Time Compression in Project Scheduling, *Management Science*, 16(10), B597-B606.

Meyer, W.L. and L.R. Shaffer (1965), Extending CPM for Multiform Project Time-Cost Curves, *Journal of the Construction Division, Proceedings of the ASCE*, 91, 45-65.

Muller, J.H. and J. Spinrad (1989), Incremental Modular Decomposition, *Journal of the ACM*, 36(1), 1-19.

Nair, K.P.K, V.R. Prasad and Y.P. Aneja (1993), Efficient Chains in a Network With Time-Cost Trade-off Function on Each Arc, *European Journal of Operational Research*, 66, 392-402.

Neumann, K. and J. Zhan (1993), Heuristics for Resource-Constrained Project Scheduling With Minimal and Maximal Time Lags, Proc. Intr'l Conf. on Ind. Eng. and Project Management, vol.2, Mons, Belgium, 2-4 June, 561-575.

Patterson, J.H. and R.T. Harvey (1979), An Implicit Enumeration Algorithm for the Time/Cost Trade-Off Problem in Project Network Analysis, *Foundations of Control Engineering*, 6(4), 107-117.

Robinson, D.R. (1975), A Dynamic Programming Solution to Cost-Time Tradeoff for CPM, *Management Science*, 22(2), 158-166.

Rothfarb, B., H. Frank, D.M. Rosenbaum, K. Steiglitz and D.J. Kleitman (1970), Optimal Design of Offshore Natural-Gas Pipeline Systems, *Operations Research*, 18(6), 992-1020.

Schwarze, J. (1980), An Algorithm for Hierarchical Reduction and Decomposition of a Directed Graph, *Computing*, 25, 45-57.

Sidney, J.B. and G. Steiner (1986), Optimal Sequencing by Modular Decomposition: Polynomial Algorithms, *Operations Research*, 34(4), 606-612.

Talbot, F.B. (1982), Resource-Constrained Project Scheduling With Time-Cost Trade-Offs: The Nonpreemptive Case, *Management Science*, 28(10), 1197-1210.

Tavares, L.V. (1990), A Multi-Stage Non-Deterministic Model for Project Scheduling Under Resource Constraints, *European Journal of Operational Research*, 49(1), 92-101.

## Appendix 1. The complexity index

Bein et al. (1992) show that the complexity index of a directed acyclic graph D = (N,A) is equal to the number of nodes in a minimum node cover of its complexity graph C(D). The *complexity graph, C(D),* of a network D = (N,A) is defined as follows: $(i,j) \in C(D)$, i.e., (i,j) is an arc of C(D), if paths $\pi(1,j)$,$\pi(i,n)$, $\pi_1(i,j)$ and $\pi_2(i,j)$ exist such that $\pi(1,j) \cap \pi_1(i,j) = \{j\}$ and $\pi(i,n) \cap \pi_2(i,j) = \{i\}$. Note that the paths $\pi_1(i,j)$ and $\pi_2(i,j)$ may be the same. The definition implies that neither 1 nor n appears as a node in C(D). The same authors have developed an algorithm for constructing the complexity graph in time $O(n^2+M(n))$, where M(n) is the time required for computing the transitive closure of a graph of n vertices (a recent upper bound for M(n) is $O(n^{2.37})$. For ease of understanding, we revise the fundamental concepts underlying the computation of the complexity graph and the complexity index. In doing so, our main concern is on clarity of exposition, rather than on issues of computational efficiency.

### Constructing the dominator trees

In constructing the complexity graph, the first step is to construct the ***dominator tree $T_1(D)$*** and the ***reverse dominator tree $T_2(D)$***. Consider a dag D = (N,A) with *root (initial) node 1*; i.e., there exists a path from 1 to every node in N. Node v is a *dominator* of node w if every path from node 1 to w contains v. Every node is a dominator of itself, and node 1 dominates every node (Aho et al. 1975). Node v is a reverse dominator of node w if every path from w to the end node contains v. The set of dominators of a node w can be linearly ordered by their order of occurrence on a shortest path from the root to w. The dominator of w closest to w (other than w itself) is called the *immediate dominator* of w. Since the dominators of each node are linearly ordered, the relation "v dominates w" can be represented by a tree with root 1. This tree is called the *dominator tree, $T_1(D)$,* for D. The *reverse dominator tree $T_2(D)$* is obtained by reversing all the arcs in D and constructing the dominator tree of the resulting graph. A dag is *series-parallel reducible (s-p)* if and only if for every arc (i,j) either i dominates j or j reverse-dominates i.

Aho et al. (1975) describe a polynomial procedure to compute the dominator tree for a rooted dag with m edges. Harel (1985) presents a procedure for computing the dominator tree T(D) in time O(m+n), where m is the number of edges in D. For our purposes, we rely on the procedure described below, which though less efficient, can be easily incorporated in a computer code. In the description of the procedure we assume that each node w = 2,...,n is considered in sequence and that on considering node w, the immediate dominator for any node $v < w$ has already been determined. The following observations (given here without proof) form the underlying basis of our computer code for transforming D into its dominator tree T(D).

***Observation 1.*** *Let D = (N,A). For each node w = 2,...,n, the number of different paths π(1,w) is equal to the number of immediate predecessors of node w.*

***Observation 2.*** *Let D = (N,A) be a dag. Let node w ∈ N have only one incoming arc (v,w). Then v is an immediate dominator of node w.*

***Observation 3.*** *Let D = (N,A) be a dag. Let node w ∈ N have more than one incoming arc and let v be the highest numbered immediate predecessor of w. Then replacing (v,w) by arc (a,w), where a is the immediate dominator of node v, does not change the dominators of any node in D.*

***Observation 4.*** *Let D = (N,A) be a dag. Let there be an arc (1,w). Then node 1 is the immediate dominator of node w.*

The repeated use of these observations will transform D into its dominator tree T(D). The data structure which allows us to find efficiently the arcs to which to apply the observations uses the upper diagonal part of the incidence matrix for the dag D. Rows 1 and n, as well as columns 1 and n must not be considered (node 1 and n will not appear in the complexity graph). We check each node w = 3,...,n-1 for the number of incoming arcs. If node w has only one incoming arc (v,w), then v is its immediate dominator (Observation 2), and the (v,w) entry is left unchanged in the incidence matrix. If node w has more than one incoming arc, we consecutively replace (v,w), where *v* is the highest numbered predecessor node, by arc (a,w), where *a* is the immediate dominator of node v (Observation 3). The corresponding entries in the incidence matrix are updated.

Figure A1-1(a) represents a 9-node, 14-arc network. The incidence matrix is represented in Figure A1-1(b). The first step is to create the dominator tree and the reverse-dominator tree. In order to create the dominator tree we scan the columns of the incidence matrix. There is no need to explicitly consider the second column: since we assume a 1-n dag, node 1 is always an immediate dominator of node 2 (Observation 2, Observation 4). Neither is there a need to explicitly consider the last column since node 9 will never appear in the complexity graph. In column 3 of the incidence matrix, a single 1 is found; i.e., $a_{13}$ = 1. Node 1 is an immediate dominator of node 3 (Observations 2 and 4). Column 4 shows $a_{14}$ = 1. Again, node 1 is an immediate dominator of node 4 (Observations 2 and 4). The single 1 in column 5 of the incidence matrix occurs for $a_{25}$ = 1. According to Observation 2, node 2 is an immediate dominator of node 5. In column 6 we find $a_{26}$=1, $a_{36}$=1 and $a_{56}$=1. Set $a_{56}$=0 and set $a_{26}$=$a_{25}$=1. Set $a_{36}$=0 and set $a_{16}$=$a_{13}$=1. Node 1 is an immediate dominator of node 6 (Observation 4). In column 7 we have $a_{37}$=1 and $a_{47}$=1. Set $a_{47}$=0 and let $a_{17}$=$a_{14}$=1. Node

1 is an immediate dominator of node 7 (Observation 4). For column 8, set $a_{78}=0$ and let $a_{18}=a_{17}=1$. Node 1 is an immediate dominator of node 8 (Observation 4). The dominator tree is represented in Figure A1-2(a).

Figure A1-1

Figure A1-2

The construction of the reverse-dominator tree is symmetric. It is easy to verify that node 9 is the immediate reverse-dominator of node 8 (Observation 2 and 4). Node 8 is the immediate reverse-dominator of node 7 (Observation 2). Node 9 is the immediate reverse-dominator of node 6 (Observation 2 and 4). Node 9 is the immediate reverse-dominator of node 5 (Observation 4). Node 8 is the immediate reverse-dominator of node 4 (Observation 2 and 3). Node 9 is the immediate reverse-dominator of node 3 (Observation 3 and 4). Node 9 is the immediate reverse-dominator of node 2 (Observation 3 and 4). Node 9 is the immediate reverse-dominator of node 1 (Observation 3 and 4). The reverse-dominator tree is represented in Figure A1-2(b).

***Constructing the complexity graph.***

The data structure for the algorithm is the incidence matrix $[a_{ij}]$ of the dag D with $a_{ij}=1$, if node i directly or indirectly precedes node j. For every node w with immediate dominator b, set $a_{vw} = 0$, for all arcs (v,w), v <= b. For every node v with immediate reverse-dominator c, set all $a_{vw} = 0$ for all arcs (v,w), w >= c. As already mentioned above, it is shown by Bein et al. (1992) that the complexity graph can be computed in time $O(n^{2.37})$.

In order to construct the complexity graph for our problem example, we need to update the incidence matrix of Figure A1-1(b) such that it also contains the indirect arcs. The resulting matrix is represented in Figure A1-3(a). Eliminating all arcs (v,w) with v smaller than or equal to the immediate dominator of node w, yields the matrix in Figure A1-3(b). Eliminating all the arcs (v,w) with w greater than or equal to the immediate reverse-dominator of node v, yields the matrix in Figure A1-3(c). The resulting complexity graph is depicted in Figure A1-3(d).

Figure A1-3

### *The minimum node cover of C(D)*

Having constructed the complexity graph C(D), the next step is to determine its minimum node cover $N^*$. Because C(D) is a transitive dag, $N^*$ can be computed by reducing the problem to finding the maximum matching in a bipartite graph, since the complement of a minimum node cover is a maximum independent set, which in a transitive dag corresponds to a Dilworth chain decomposition (Ford & Fulkerson 1962). Hopcroft & Karp (1973) offer an $O(n^{5/2})$ algorithm for maximum matchings in bipartite graphs. In the sequel, we follow the original arguments of Ford & Fulkerson (1962) who offer a simple procedure that runs in time O(mn), where m is the number of edges and n is the number of vertices.

Following Ford & Fulkerson's notation, let P be a finite partially ordered set with elements 1,2,...,n and order relation ">". A *chain* in P is a set of one or more elements $i_1,i_2,...,i_k$ with $i_1 > i_2 > ... > i_k$. If we associate a directed graph with P by taking nodes 1,2,...,n and arcs (i,j) corresponding to i>j, this notion of a chain coincides with the notion of a chain in the graph, except that now we allow a single node to be a chain. A *decomposition* of P is a partition of P into chains. Thus P always has the trivial decomposition into *n* 1-element chains. A decomposition with the smallest number of chains is *minimal.*

Two distinct members i,j of P are unrelated if neither i>j nor j>i. The maximal number of mutually unrelated elements of P is less than or equal to the number of chains in a minimal decomposition of P, since two members of a set of mutually unrelated elements cannot belong to the same chain. Ford & Fulkerson (1962) establish the proof that the problem of constructing a minimal decomposition can be solved by their labeling algorithm for constructing a maximal independent set of admissible cells in an array. In the context of the minimum node cover problem on hand, the array to be considered is the (nxn) incidence matrix for the complexity graph C(D), in which a cell is *admissible* if the corresponding arc appears in C(D).

The algorithm proceeds as follows. Let i = 1,...,n index the rows of the array, j = 1,...,n the columns. The maximal flow problem becomes that of placing as many 1's as possible in admissible cells, with the proviso that at most one 1 can be placed in any row and column.

**Initialize:**

```
For rows i=1,...,n
    For columns j=1,...,n
        If cell[i,j] admissible then place a 1
           Delete row i and column j;
```

```
Label with dashes:

labeled_set = ∅;

For rows i=1,...,n
    If no 1 placed in row i then
       labeled_set = labeled_set + {row i};
       Label row i with '-';

Labeling:

While labeled_set ≠∅  do

   Choose labeled row i;
   Labeled_set = labeled_set - {row i};
   For columns j=1,...,n
       If cell[i,j] is admissible and column j not labeled then
          label column j with row number i;
          If column j contains no 1
             then
                BREAKTHROUGH:
                   Place a 1 in cell [i,j];
                   While row label i ≠ '-' do
                     j = row label i;
                     Remove 1 from cell[i,j];
                     i = column label j;
                     Place a 1 in cell[i,j];
                   Erase labels and go to Label with dashes
              else
                Label row with j;
                labeled_set = labeled_set + {row i}

NON-BREAKTHROUGH:
Minimal covering set = {unlabeled rows} + {labeled columns};
```

Figure A1-4 represents the array corresponding with the complexity graph C(D) depicted in Figure A1-3(d). Admissible cells (correponding to an arc of C(D)) are blank and inadmissible cells are crossed out. The 1's shown constitute an initial placement using the **Initialize** procedure. Rows 2 and 3 contain 1's. The procedure **Label with dashes** labels the other rows with '-'. Labeled row 4 has admissible cell[4,7]. Column 7 is labeled with 4. Scanning column 7, we find a 1 in row 3. Row 3 is labeled with 7. Labeled row 5 has admissible cell[5,6]. Column 6 is labeled with 5. Scanning column 6, we find a 1 in row 2. Row 2 is labeled with 6. Labeled rows 6, 7 and 8 have no admissible cells. Labeled row 3 has an admissible cell[3,8]. Column 8 is labeled with 3. Column 8 has no 1. We have a breakthrough. The arrows in Figure A1-4 indicate the resulting sequence of changes in the placement of 1's.

Figure A1-4

After making the indicated changes and relabeling, we obtain the array depicted in Figure A1-5. The rows 1,5,6,7,8,9 are labeled with '-'. Scanning labeled row 5, we find admissible cell[5,6]. Label column 6 with 5. Scanning column 6, we find a 1 in cell[2,6]. Label row 2 with 6. No further labeling is possible, and we have a non-breakthrough. A minimal cover is found to consist of unlabeled rows 3 and 4, and labeled column 6. The nodes to be reduced in the network of Figure A1(a) are nodes 3, 4 and 6. Hence, CI = 3.

Figure A1-5

## FIGURE CAPTIONS

Figure 1. A s/p reducible dag and DP recursion equations

Figure 2. Example of parallel merge

Figure 3. Example of serial merge

Figure 4. Example network

Figure 5. The steps of reduction of Figure 4

Figure 6. The optimal fixing tree

Figure 7. The interdictive graph

Figure 8. Example network and its search tree

Figure 9. The six example networks of Experiment 1

Figure 10. The four networks of Experiment 3. The solid arcs give the smaller networks (of 21 and 37 arcs); the dotted arcs give the expanded networks (of 24 and 42 arcs)

Figure 11. The incidence matrices of the seven networks of Experiment 5

Figure 12. The network of Experiment 6: 11 nodes, 20 activities, CI = 6

Figure 13. Variation of the execution time and number of leafs with resource availability; Experiment 6

Figure A1-1. Problem example

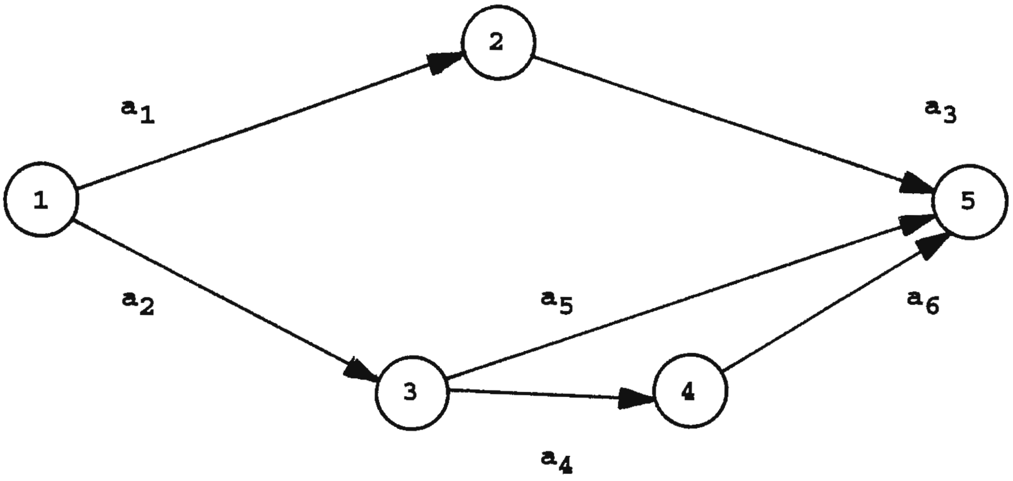
Figure A1-2. Dominator and reverse dominator tree for the problem example

Figure A1-3. Complexity graph for the problem example

Figure A1-4. Array of admissible cells

Figure A1-5. Relabeled array

**TABLE CAPTIONS**



$L_1(X) = g_1(x1) + g_3(x_3)$

$L_2(X) = g_2(x_2) + g_5(x_5)$

$L_3(X) = g_2(x_2) + g_4(x_4) + g_6(x_6)$

$\max \{L_k(X)\} = \max \{L_1(X), g_2(x_2) + \max \{g_5(x_5); g_4(x_4)+g_6(x_6)\}\}$

Dynamic programming resursion equations (all $r \in [0,R]$):

$$f_1(r) = \min_{x_4} [g_4(x_4) + g_6(r-x_4)]$$

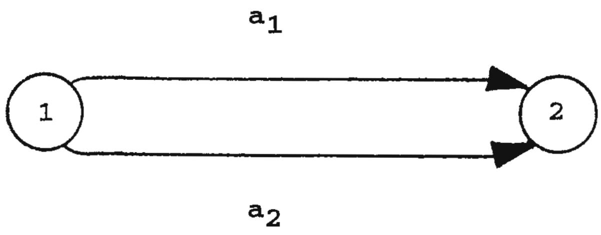
$$f_2(r) = \min_{x_5} [\max\{g_5(x_5), f_1(r-x_5)\}]$$

$$f_3(r) = \min_{x_2} [g_2(x_2) + f_2(r-x_2)]$$

$$f_4(r) = \min_{x_1} [g_1(x_1) + g_3(r-x_1)]$$

$$f_5(R) = \min_{r} [\max\{f_4(r), f_3(R-r)\}]; \ 0 < r <R$$

Figure 1



| | Allocation $x_a$ | Duration $g_a(x_a)$ |
|---|---|---|
| Activity $a_1$ | 3 | 12 |
| | 9 | 9 |
| | 15 | 7 |
| Activity $a_2$ | 1 | 10 |
| | 5 | 8 |
| | 10 | 7 |

Figure 2

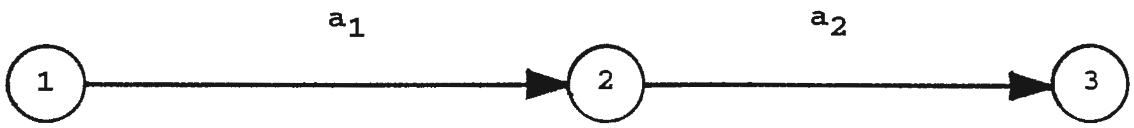
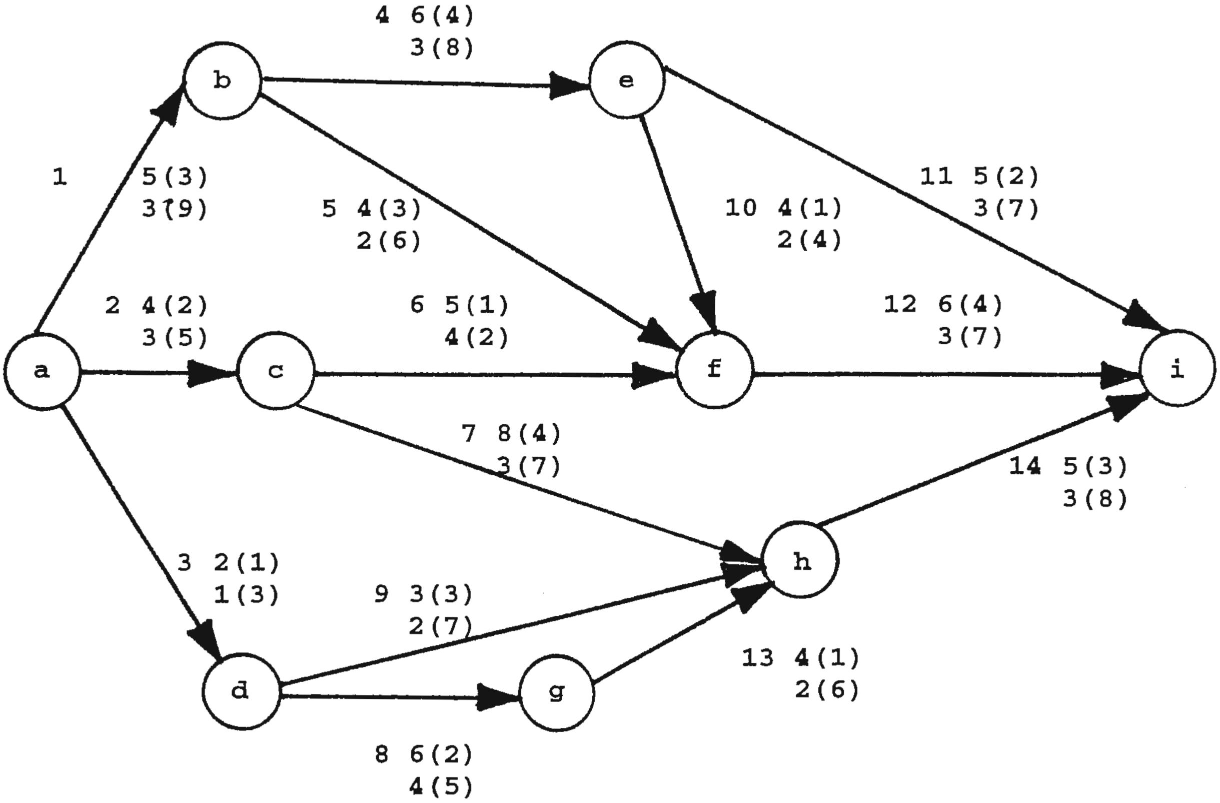


Figure 3



Project 11(69)
12(65)
13(57)
14(54)
15(49)
16(45)
17(43)
18(40)
21(34)

Figure 4

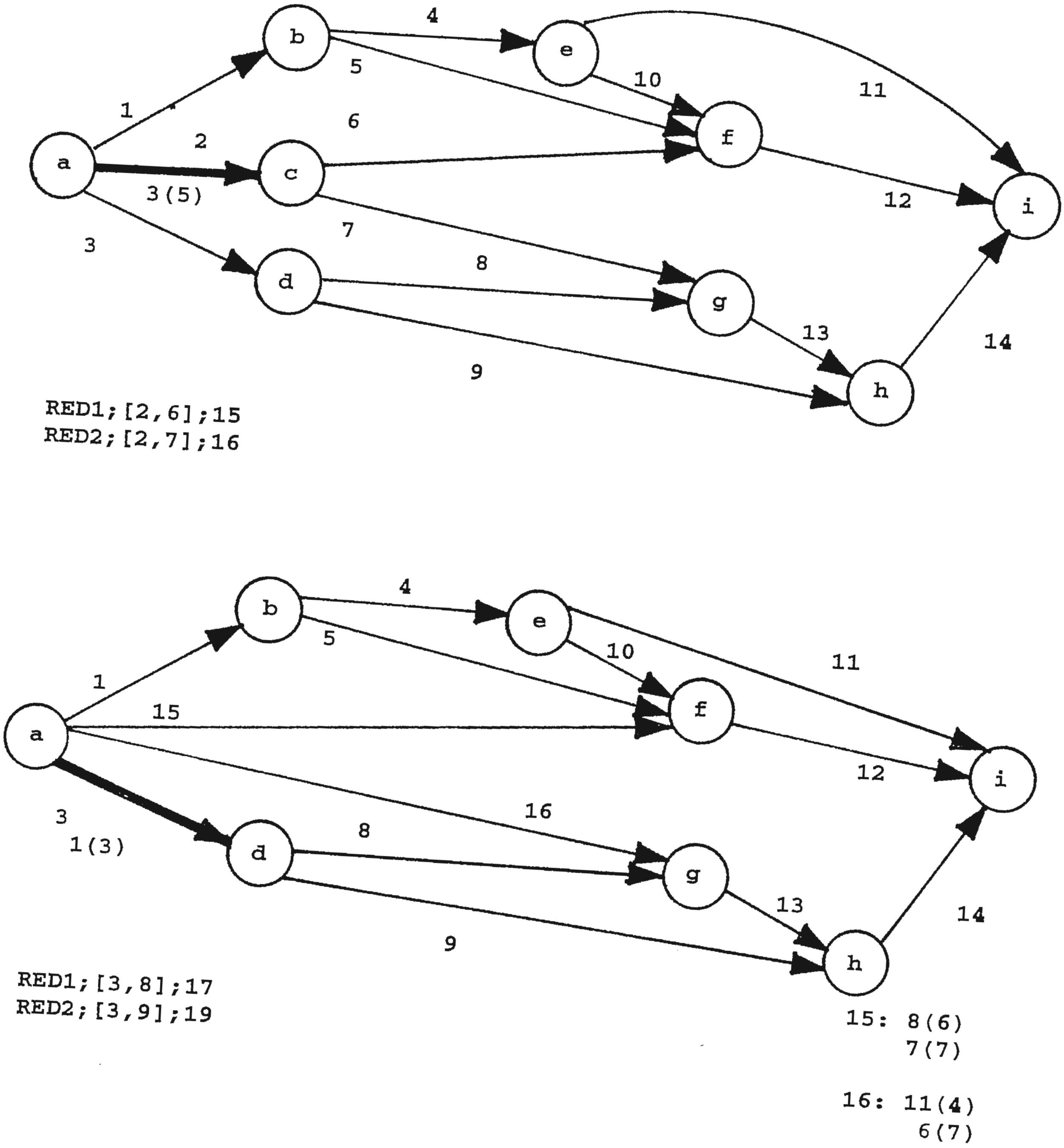
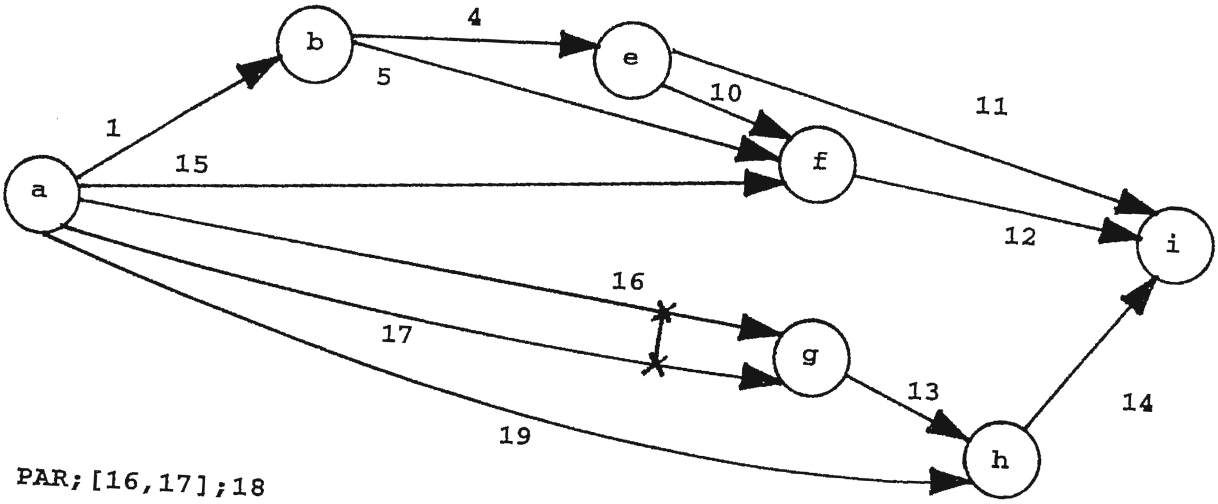

b
4
e
5
10
11
1
6
f
2
a
c
3(5)
12
i
7
3
8
d
g
13
14
9
h
RED1;[2,6];15
RED2;[2,7];16
b
4
e
5
10
11
1
15
f
a
12
16
3
1(3)
8
d
g
13
14
9
h
RED1;[3,8];17
RED2;[3,9];19
15: 8(6)
7(7)
16: 11(4)
6(7)

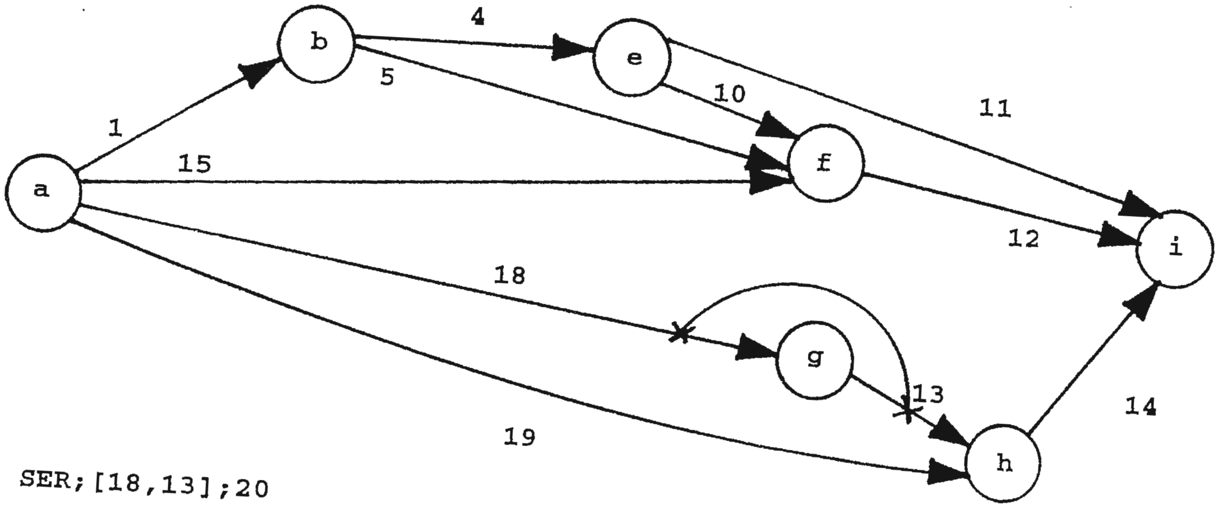
Figure 5



PAR;[16,17];18

17: 7(5)
5(8)

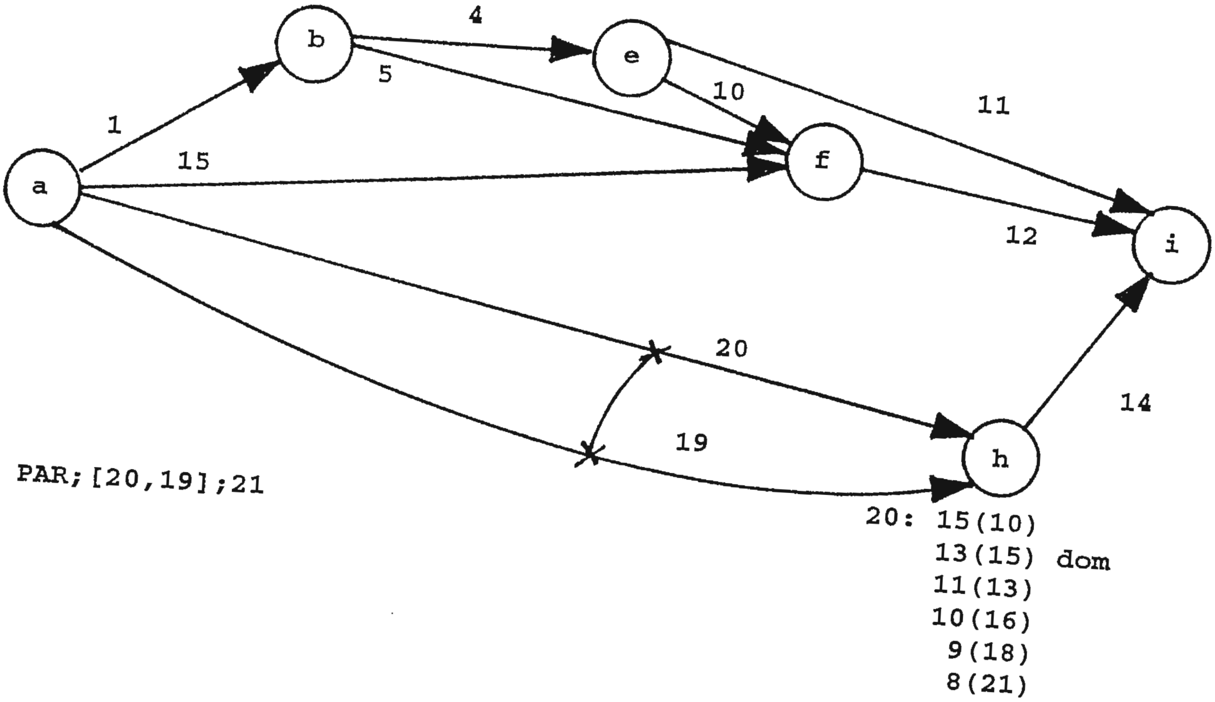
19: 4(3)
3(7)



SER;[18,13];20

18: 11(9)
7(12)
6(15)

Figure 5 : Cont'd 1



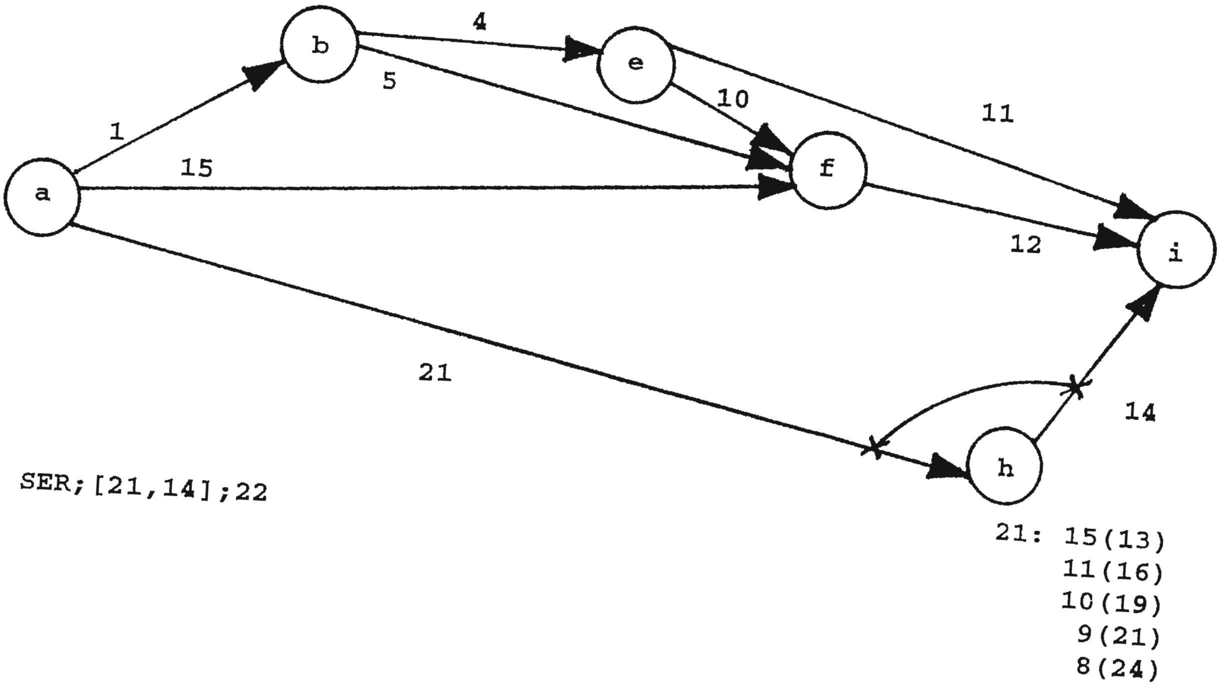
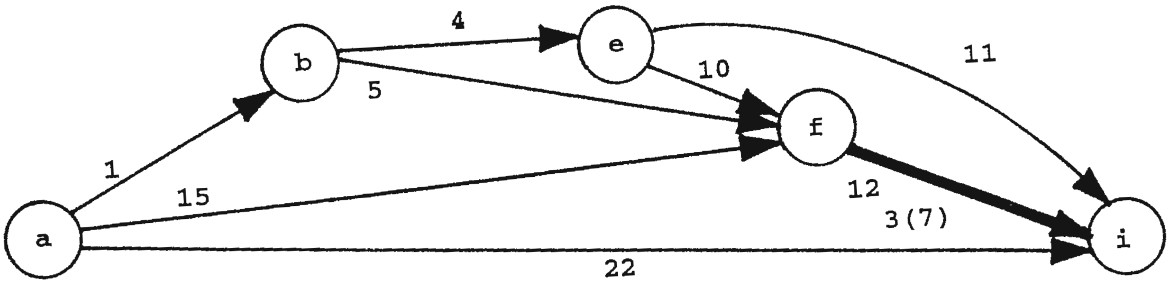
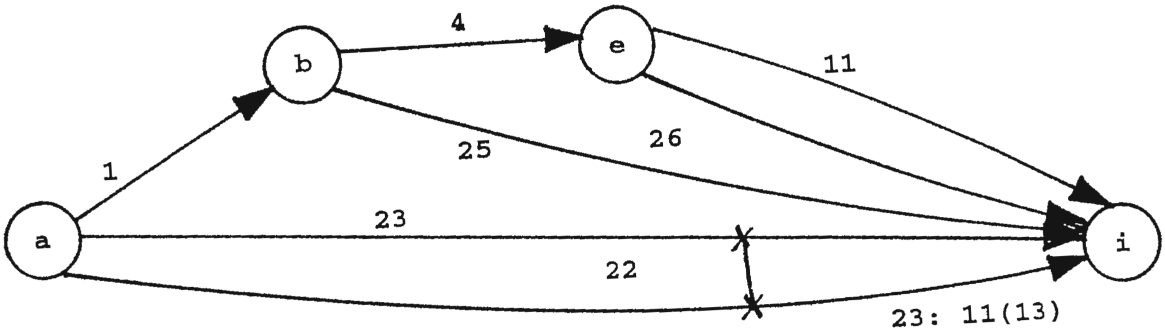


Figure 5 : Cont'd 2



```
RED1;[12,15];23
RED2;[12, 5];25
RED2;[12,10];26
```

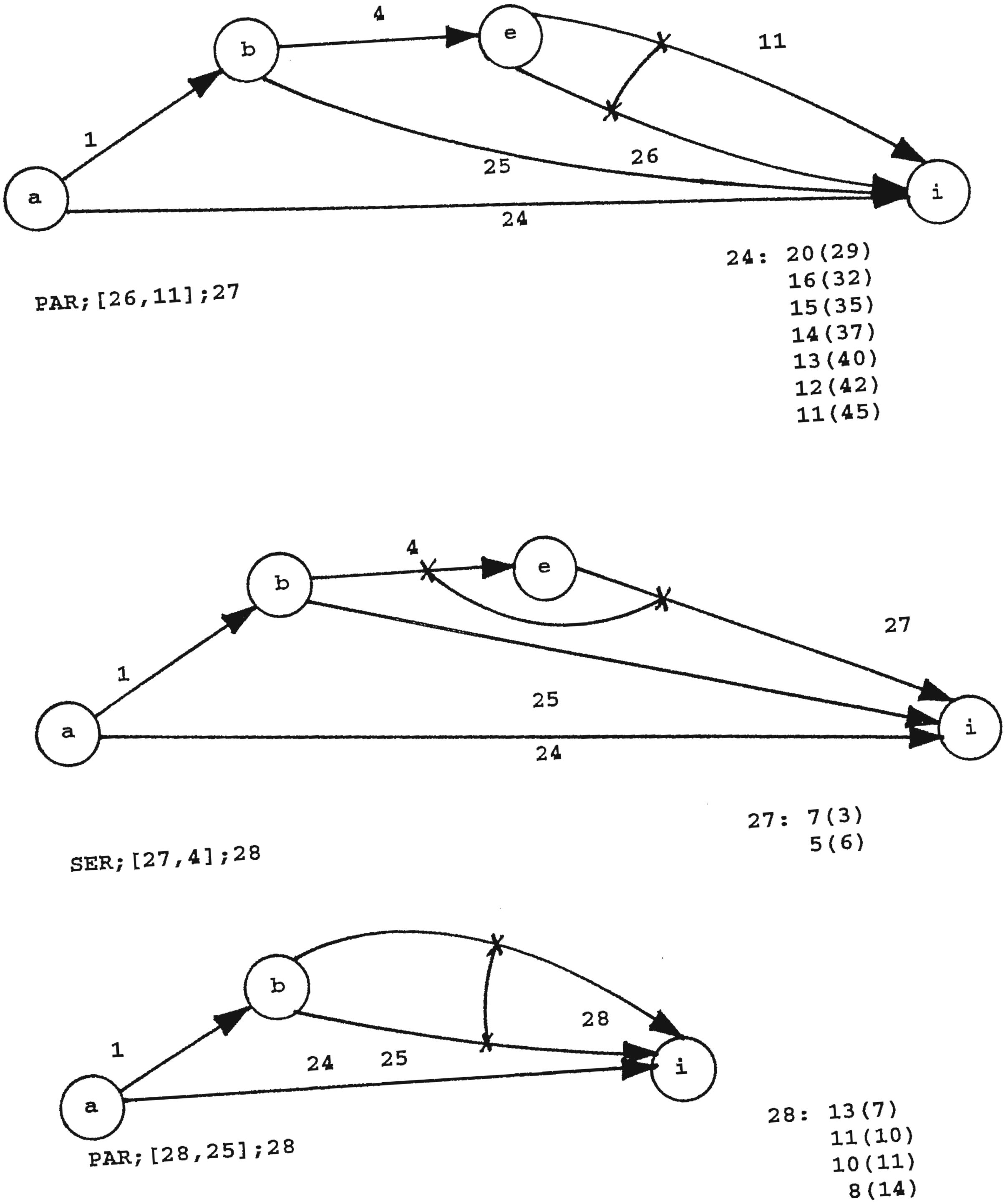
```
22: 20(16)
     18(21) dom
     16(19)
     15(22)
     14(24)
     13(27)
     14(24) repl.
     12(29)
     13(27) repl.
     11(32)
```



```
PAR;[23,22];24
```

```
23: 11(13)
    10(14)
25:  7(3)
     5(6)
26:  7(1)
     5(4)
```

Figure 5 : Cont'd 3



PAR;[26,11];27

24: 20(29)
16(32)
15(35)
14(37)
13(40)
12(42)
11(45)

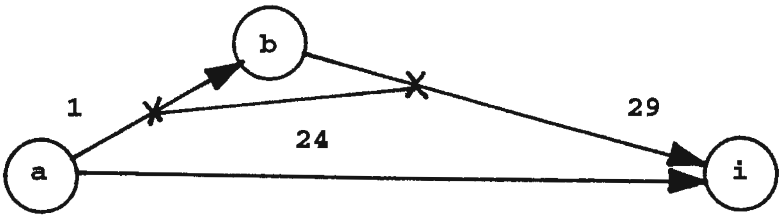
SER;[27,4];28

27: 7(3)
5(6)

PAR;[28,25];28

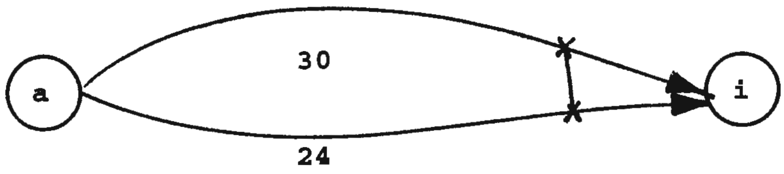
28: 13(7)
11(10)
10(11)
8(14)

Figure 5: Cont'd 4



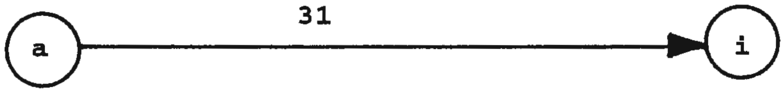
SER;[29,1];30

```
29: 13(10)
    11(13)
    10(14)
     8(17)
```



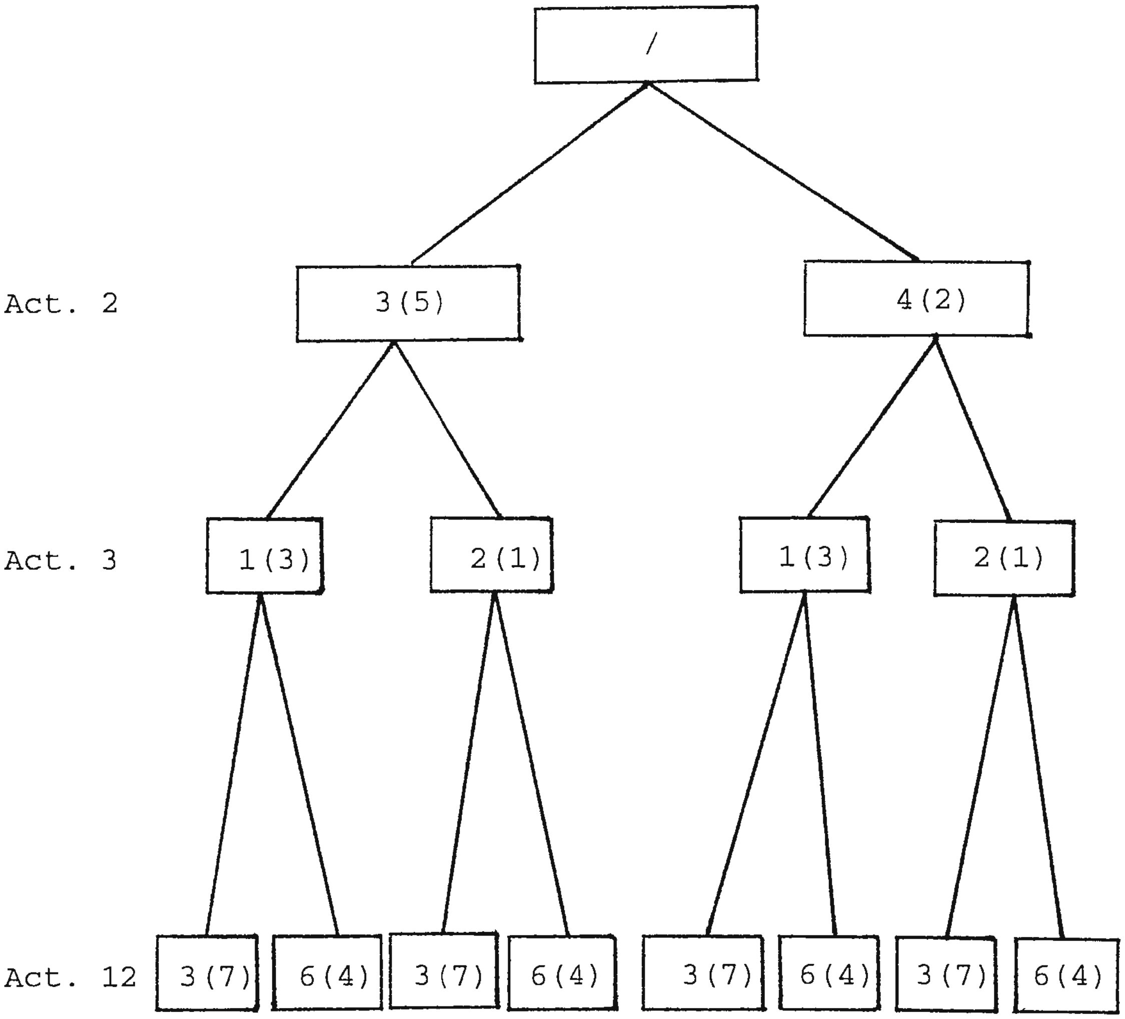
PAR;[30,24];31

```
30: 18(13)
    16(16)
    15(17)
    14(22) dom
    13(20)
    11(26)
```



```
31: 20(42)
    18(45)
    16(48)
    15(52)
    14(57)
    13(60)
    12(68)
    11(71)
```

Figure 5 : Cont'd 5



**Project:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 20(42) | 21(39) | 20(40) | 21(37) | 21(39) | 21(36) | 21(37) | **21(34)** |
| 18(45) | 20(42) | 18(43) | 20(40) | 18(42) | 19(42) | **18(40)** | 19(40) |
| 16(48) | 19(45) | 17(46) | 19(43) | **16(45)** | 18(43) | **17(43)** | 18(41) |
| 15(52) | 18(46) | 16(49) | 18(44) | 15(51) | 16(46) | 16(46) | 17(44) |
| 14(57) | 16(49) | 15(50) | 17(47) | **14(54)** | 15(57) | **15(49)** | 16(47) |
| 13(60) | 15(58) | 13(58) | 16(50) | 13(59) | 14(58) | 14(55) | 15(55) |
| 12(68) | 14(60) | **11(69)** | 15(56) | **12(65)** | | **13(57)** | 14(59) |
| 11(71) | | | 14(61) | | | 12(57) | |

Figure 6

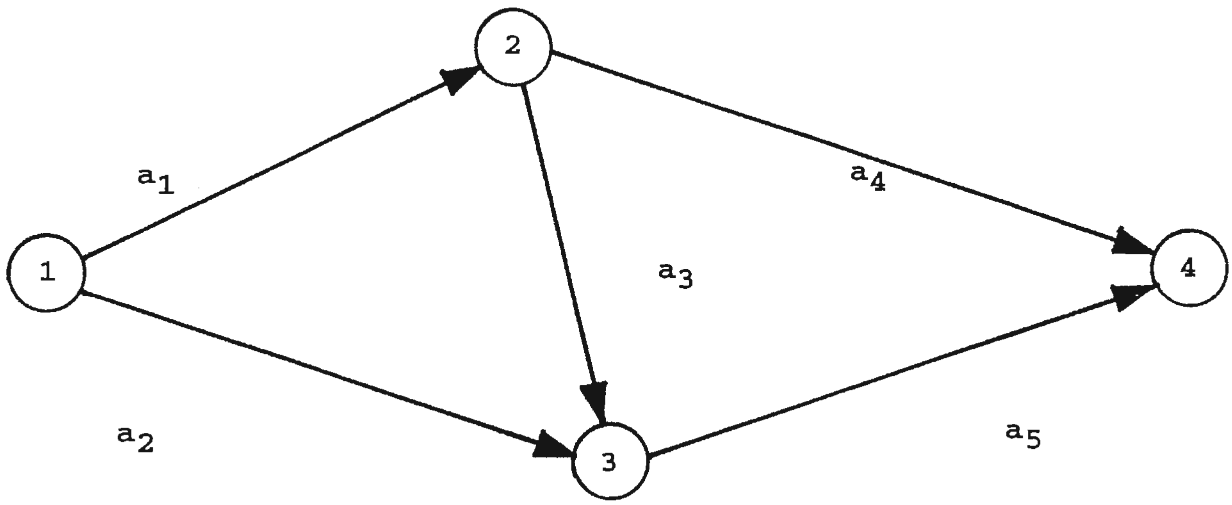


Figure 7

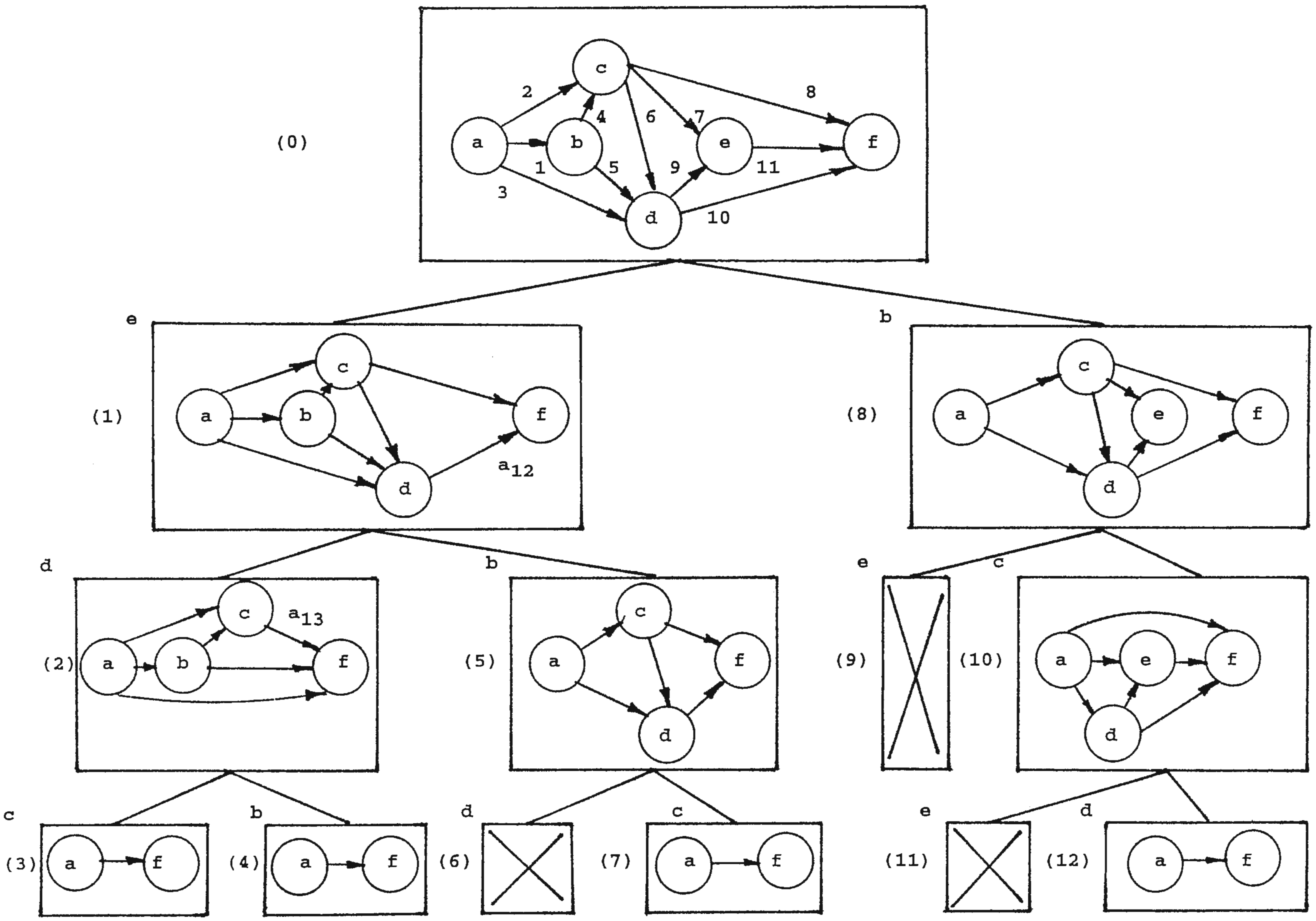
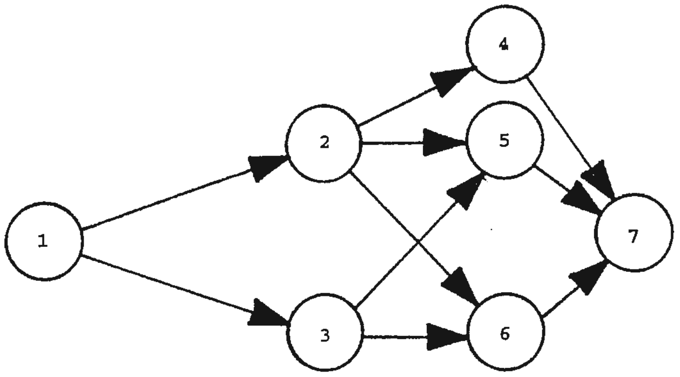
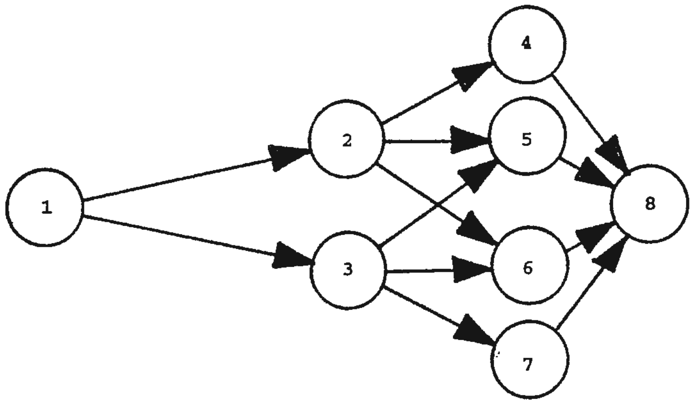

(0)
a
b
c
d
e
f
1
2
3
4
5
6
7
8
9
10
11
e
(1)
a
b
c
d
f
$a_{12}$
b
(8)
a
c
d
e
f
d
(2)
a
b
c
f
$a_{13}$
b
(5)
a
c
d
f
e
(9)
c
(10)
a
d
e
f
c
(3)
a
f
b
(4)
a
f
d
(6)
c
(7)
a
f
e
(11)
d
(12)
a
f

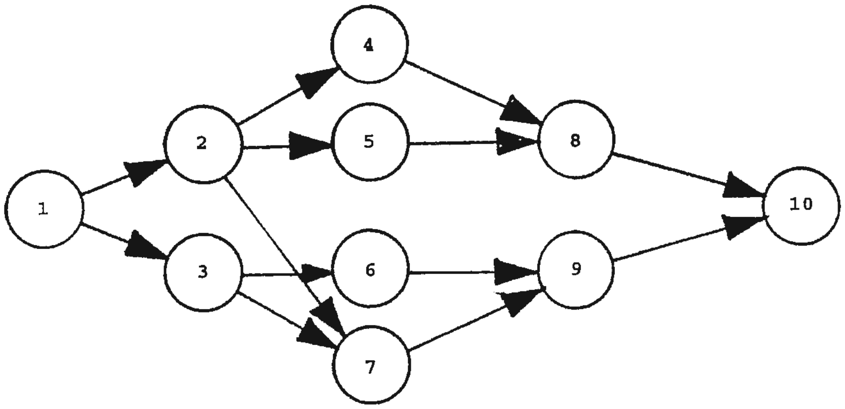
Figure 8



(1) Network 1 (CI = 2)

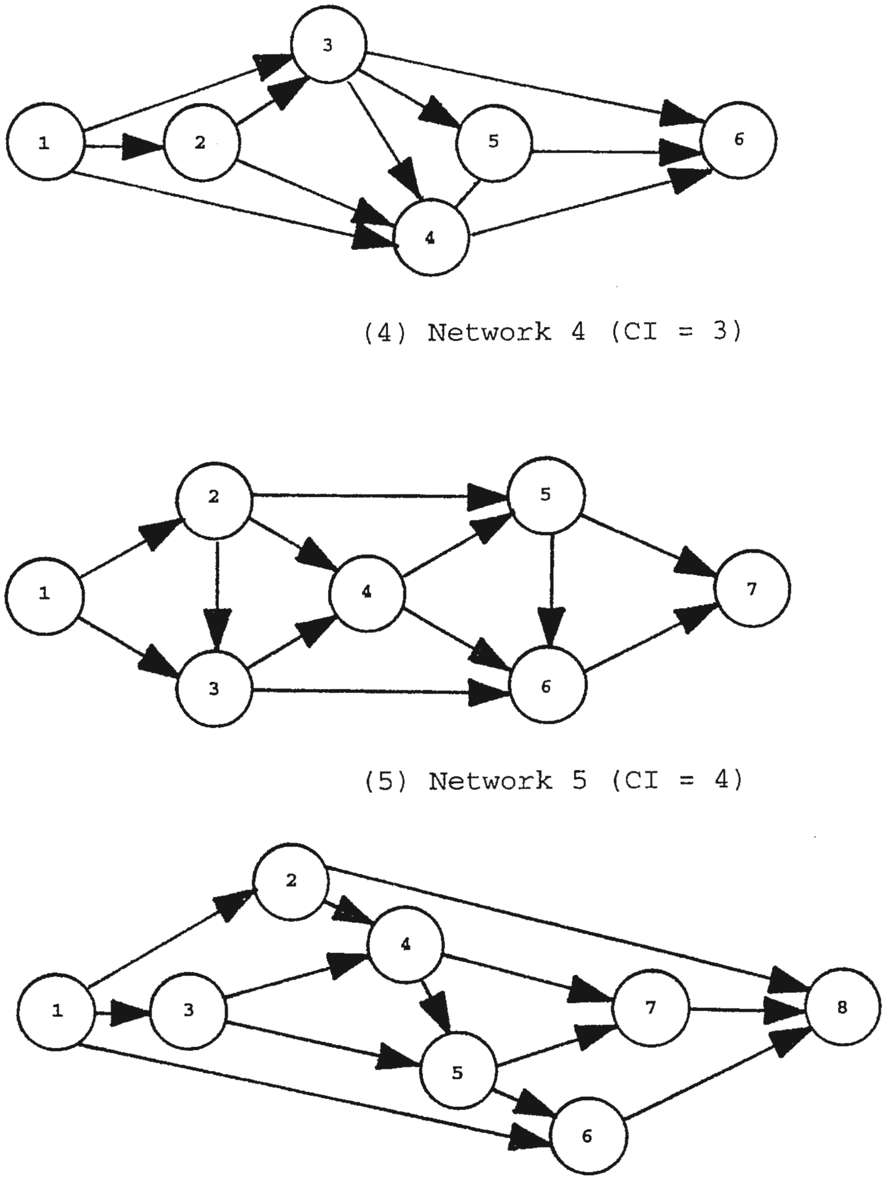


(2) Network 2 (CI = 2)



(3) Network 3 (CI = 2)

Figure 9



(4) Network 4 (CI = 3)

(5) Network 5 (CI = 4)

(6) Network 6 (CI = 4)

Figure 9

```
12/21 : CI = 4    [2,3,4,6]
12/24 : CI = 6    [2,3,4,9,10,11]
```

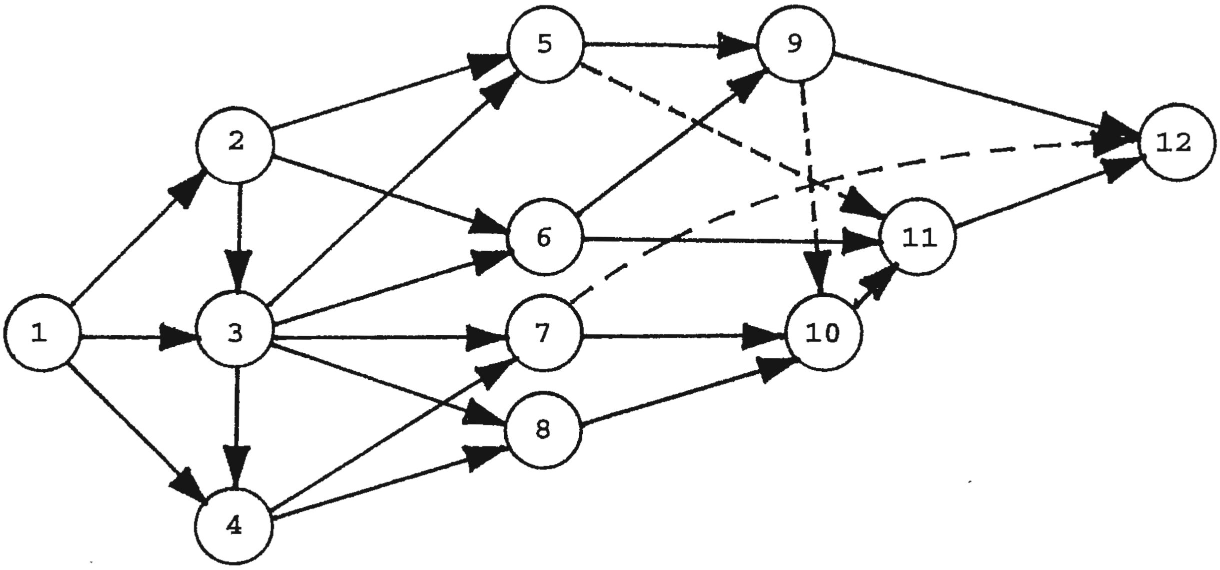


Figure 10(a)

```
20/37 : CI = 8    [2,3,4,5,6,7,9,12]
20/42 : CI = 12   [2,3,4,5,6,7,8,9,10,12,13,14]
```

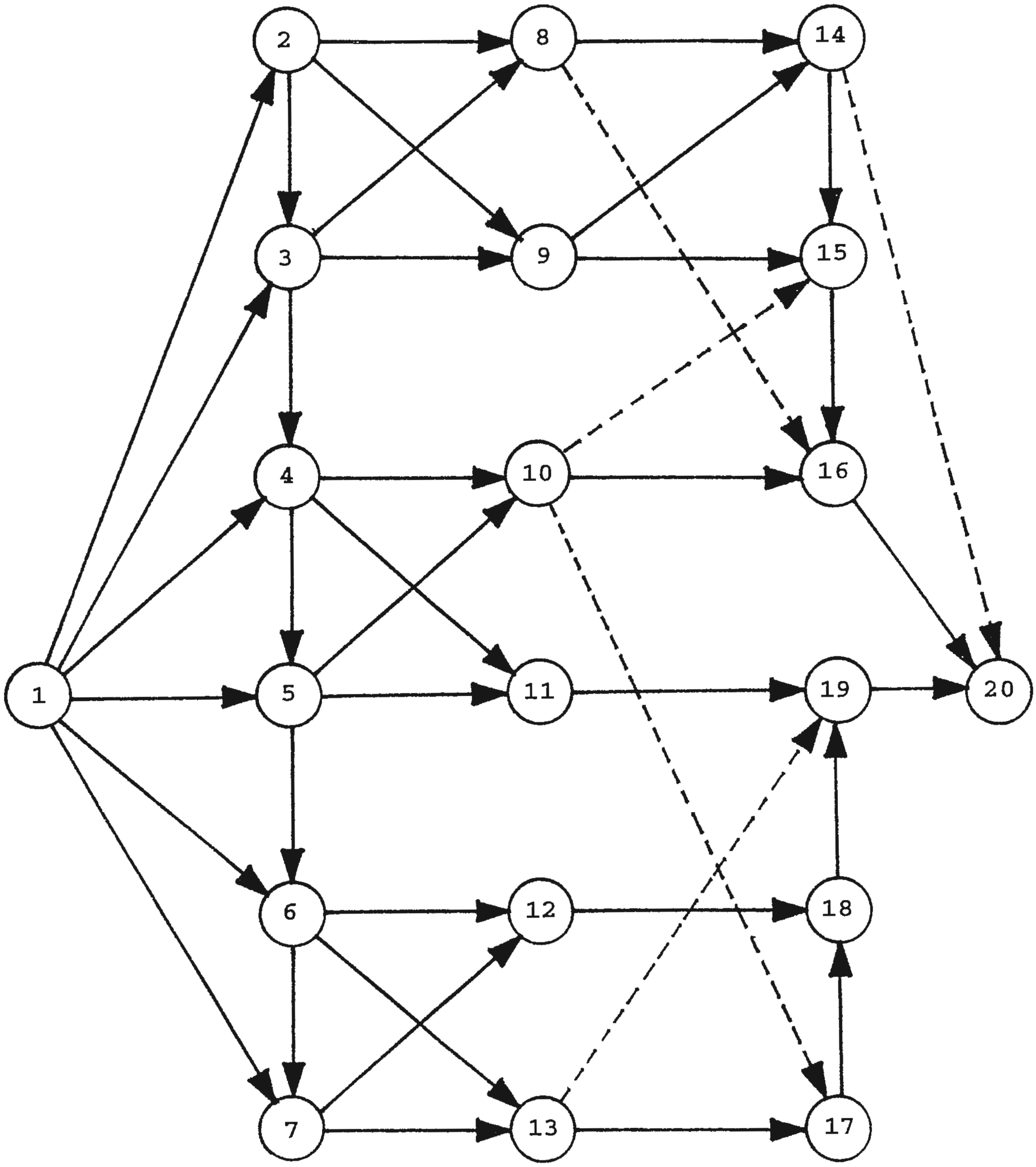


Figure 10(b)

**11/15 CI = 1 [2]**

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 |  |  | 1 |  |  |  |  |  |  |
| 2 |  | - | 1 | 1 |  |  |  | 1 |  | 1 |  |
| 3 |  |  | - |  |  | 1 |  |  | 1 |  |  |
| 4 |  |  |  | - |  |  | 1 |  |  |  |  |
| 5 |  |  |  |  | - |  | 1 |  |  |  |  |
| 6 |  |  |  |  |  | - |  |  |  | 1 |  |
| 7 |  |  |  |  |  |  | - |  |  | 1 |  |
| 8 |  |  |  |  |  |  |  | - |  |  | 1 |
| 9 |  |  |  |  |  |  |  |  | - | 1 |  |
| 10 |  |  |  |  |  |  |  |  |  | - | 1 |
| 11 |  |  |  |  |  |  |  |  |  |  | - |

**11/20 CI = 4 [2,5,6,7]**

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 |  | 1 |  |  |  |  |  |  |  |
| 2 |  | - | 1 |  | 1 | 1 | 1 |  |  | 1 | 1 |
| 3 |  |  | - |  |  | 1 |  |  |  |  |  |
| 4 |  |  |  | - |  |  |  | 1 |  |  |  |
| 5 |  |  |  |  | - | 1 | 1 |  |  |  |  |
| 6 |  |  |  |  |  | - | 1 |  | 1 |  |  |
| 7 |  |  |  |  |  |  | - | 1 |  |  | 1 |
| 8 |  |  |  |  |  |  |  | - |  |  | 1 |
| 9 |  |  |  |  |  |  |  |  | - |  | 1 |
| 10 |  |  |  |  |  |  |  |  |  | - | 1 |
| 11 |  |  |  |  |  |  |  |  |  |  | - |

Figure 11

**11/25 CI = 5 [2,3,5,7,8]**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 | | 1 | | 1 | 1 | | 1 | | |
| 2 | | - | 1 | | 1 | | 1 | 1 | | | 1 |
| 3 | | | - | | 1 | 1 | | | | 1 | 1 |
| 4 | | | | - | | | | 1 | | | 1 |
| 5 | | | | | - | 1 | | | | 1 | |
| 6 | | | | | | - | | 1 | | | 1 |
| 7 | | | | | | | - | 1 | | 1 | |
| 8 | | | | | | | | - | | | 1 |
| 9 | | | | | | | | | - | | 1 |
| 10 | | | | | | | | | | - | 1 |
| 11 | | | | | | | | | | | - |

**11/30 CI = 5 [2,3,4,6,7]**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 | 1 | 1 | 1 | | | | 1 | | |
| 2 | | - | | 1 | | 1 | 1 | 1 | | | 1 |
| 3 | | | - | 1 | | | 1 | 1 | | 1 | |
| 4 | | | | - | 1 | 1 | 1 | 1 | 1 | | |
| 5 | | | | | - | | | 1 | | | |
| 6 | | | | | | - | 1 | 1 | | 1 | 1 |
| 7 | | | | | | | - | | 1 | 1 | 1 |
| 8 | | | | | | | | - | | | 1 |
| 9 | | | | | | | | | - | 1 | |
| 10 | | | | | | | | | | - | 1 |
| 11 | | | | | | | | | | | - |

Figure 11: Cont'd1

**11/35 CI = 7 [2,3,4,5,6,7,9]**

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | – | 1 | 1 | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 |
| 2 |  | – | 1 | 1 | 1 |  |  | 1 |  | 1 | 1 |
| 3 |  |  | – |  | 1 |  |  |  | 1 | 1 | 1 |
| 4 |  |  |  | – | 1 |  |  |  | 1 | 1 | 1 |
| 5 |  |  |  |  | – | 1 |  | 1 |  | 1 |  |
| 6 |  |  |  |  |  | – | 1 | 1 |  |  |  |
| 7 |  |  |  |  |  |  | – | 1 | 1 | 1 |  |
| 8 |  |  |  |  |  |  |  | – | 1 |  |  |
| 9 |  |  |  |  |  |  |  |  | – | 1 | 1 |
| 10 |  |  |  |  |  |  |  |  |  | – | 1 |
| 11 |  |  |  |  |  |  |  |  |  |  | – |

**11/40 CI = 7 [2,3,4,5,8,9,10]**

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | – | 1 | 1 |  | 1 |  | 1 | 1 |  | 1 |  |
| 2 |  | – | 1 | 1 | 1 | 1 | 1 |  |  | 1 | 1 |
| 3 |  |  | – |  | 1 |  | 1 | 1 | 1 | 1 | 1 |
| 4 |  |  |  | – |  | 1 |  | 1 |  | 1 | 1 |
| 5 |  |  |  |  | – | 1 | 1 |  | 1 | 1 | 1 |
| 6 |  |  |  |  |  | – |  | 1 | 1 | 1 | 1 |
| 7 |  |  |  |  |  |  | – | 1 |  | 1 | 1 |
| 8 |  |  |  |  |  |  |  | – | 1 |  | 1 |
| 9 |  |  |  |  |  |  |  |  | – | 1 | 1 |
| 10 |  |  |  |  |  |  |  |  |  | – | 1 |
| 11 |  |  |  |  |  |  |  |  |  |  | – |

Figure 11: Cont'd2

**11/45 CI = 7 [2,3,4,5,6,7,10]**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 | | 1 | | 1 | 1 | 1 | | 1 | 1 |
| 2 | | - | 1 | | | | 1 | 1 | | 1 | 1 |
| 3 | | | - | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | | | | - | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | | | | | - | 1 | | 1 | 1 | 1 | 1 |
| 6 | | | | | | - | 1 | 1 | 1 | | 1 |
| 7 | | | | | | | - | | 1 | 1 | 1 |
| 8 | | | | | | | | - | | 1 | 1 |
| 9 | | | | | | | | | - | 1 | 1 |
| 10 | | | | | | | | | | - | 1 |
| 11 | | | | | | | | | | | - |

Figure 11: Cont'd3

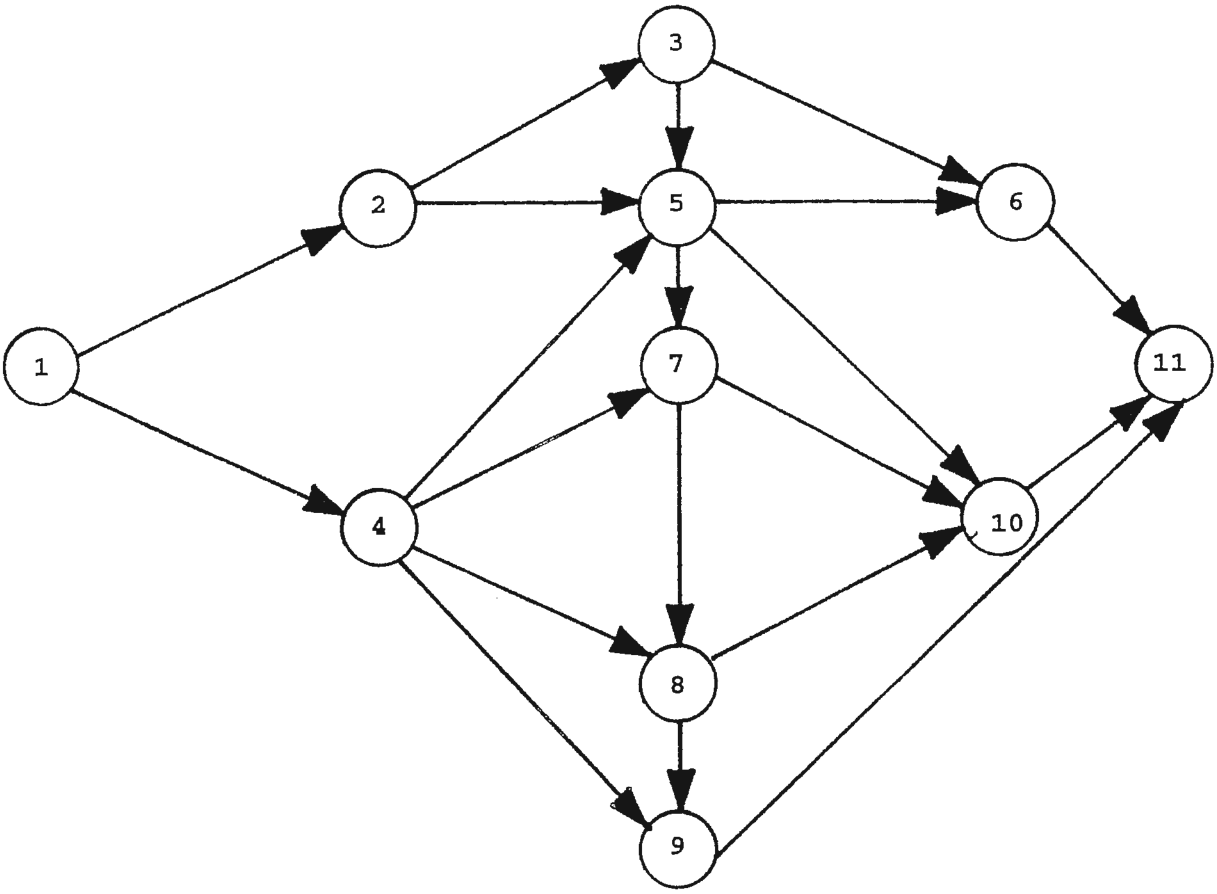
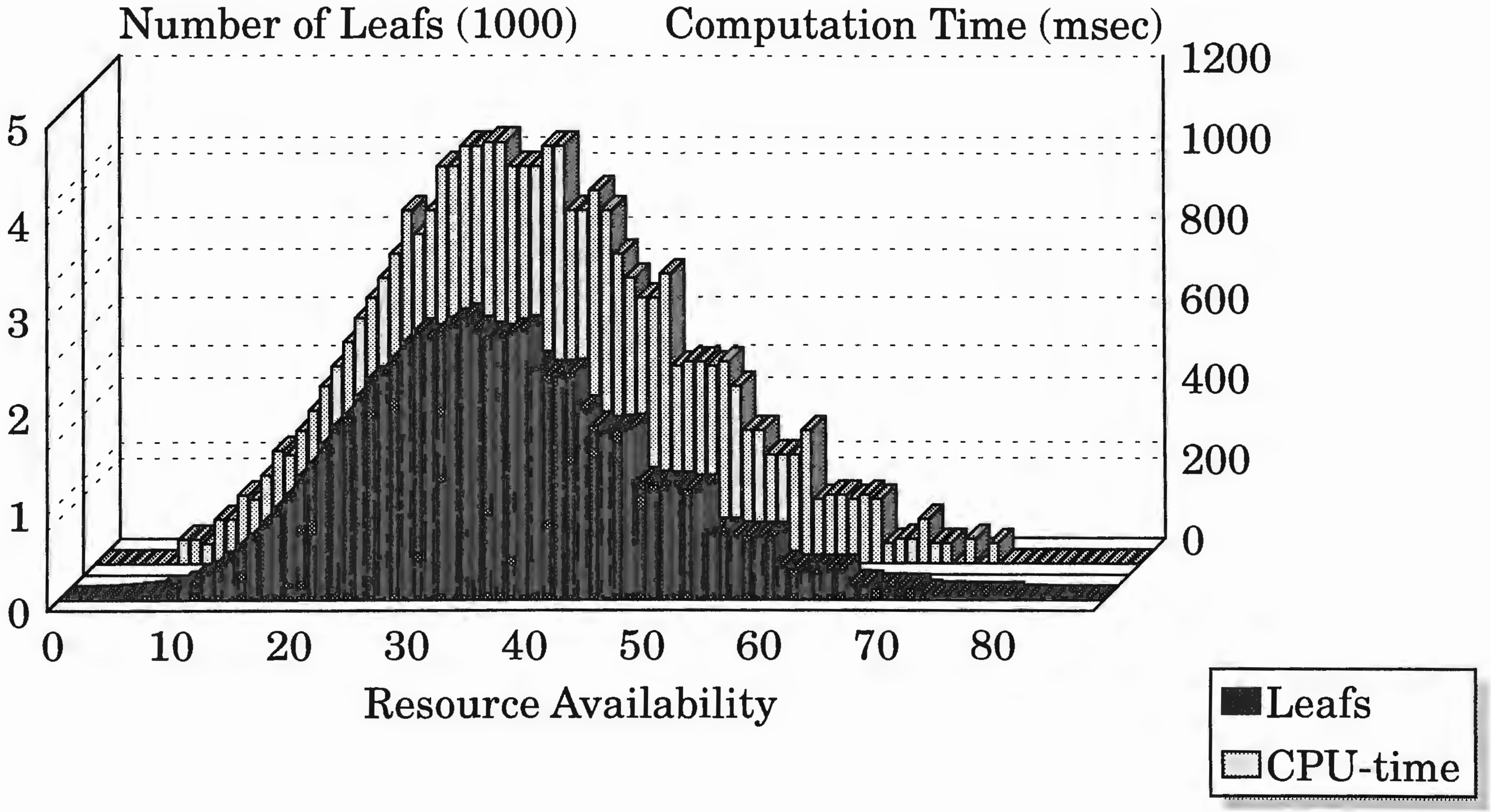
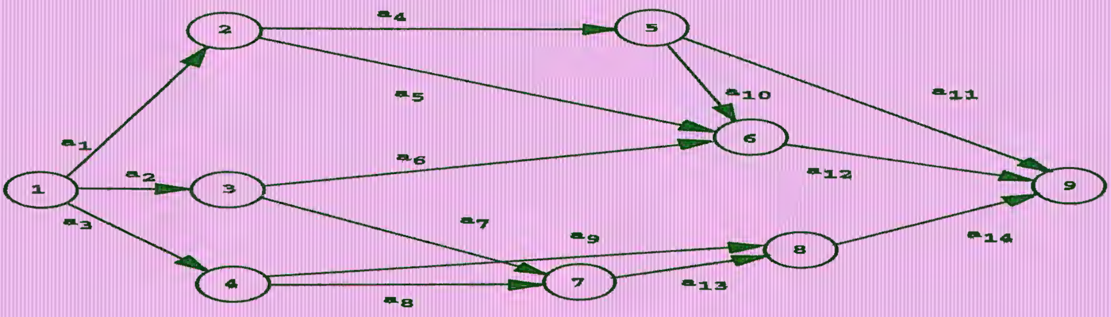


Figure 12

Figure 13


Number of Leafs (1000)
Computation Time (msec)
5
4
3
2
1
0
1200
1000
800
600
400
200
0
0
10
20
30
40
50
60
70
80
Resource Availability
■ Leafs
□ CPU-time

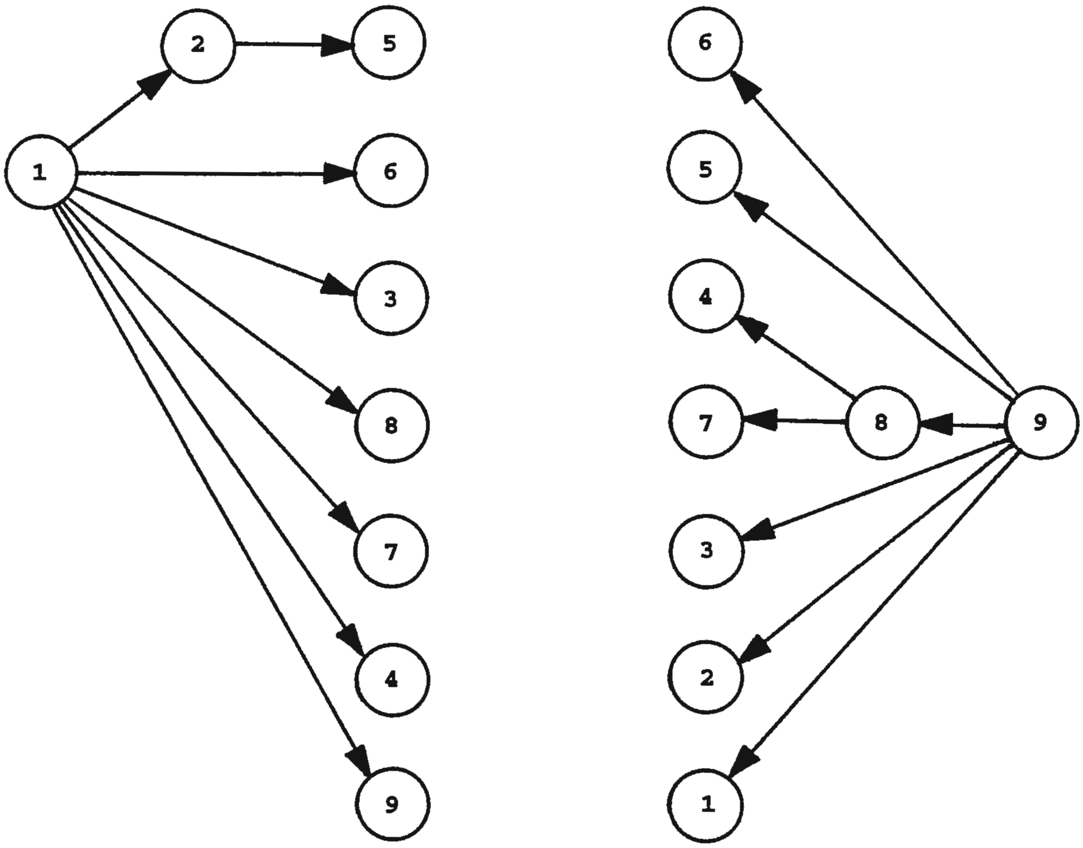


(a) Activity-on-the-arc network

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | – | 1 | 1 | 1 | | | | | |
| 2 | | – | | | 1 | 1 | | | |
| 3 | | | – | | | 1 | 1 | | |
| 4 | | | | – | | | 1 | 1 | |
| 5 | | | | | – | 1 | | | 1 |
| 6 | | | | | | – | | | 1 |
| 7 | | | | | | | – | 1 | |
| 8 | | | | | | | | – | 1 |
| 9 | | | | | | | | | – |

(b) Incidence matrix

Fig. A1-1



(a) Dominator tree

(b) Reverse-dominator tree

Fig. A1-2

| Nodes | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 |  | - | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 3 |  |  | - | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 |  |  |  | - | 0 | 0 | 1 | 1 | 1 |
| 5 |  |  |  |  | - | 1 | 0 | 0 | 1 |
| 6 |  |  |  |  |  | - | 0 | 0 | 1 |
| 7 |  |  |  |  |  |  | - | 1 | 1 |
| 8 |  |  |  |  |  |  |  | - | 1 |
| 9 |  |  |  |  |  |  |  |  | - |

**(a) Updated incidence matrix**

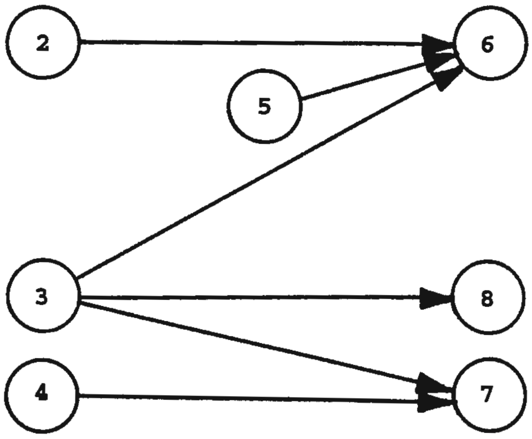
| Nodes | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 |  | - | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 3 |  |  | - | 0 | 0 | 1 | 1 | 1 | 1 |
| 4 |  |  |  | - | 0 | 0 | 1 | 1 | 1 |
| 5 |  |  |  |  | - | 1 | 0 | 0 | 1 |
| 6 |  |  |  |  |  | - | 0 | 0 | 1 |
| 7 |  |  |  |  |  |  | - | 1 | 1 |
| 8 |  |  |  |  |  |  |  | - | 1 |
| 9 |  |  |  |  |  |  |  |  | - |

**(b) Updated matrix**

Fig. A1-3

| Nodes | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | - | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 |  | - | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 |  |  | - | 0 | 0 | 1 | 1 | 1 | 0 |
| 4 |  |  |  | - | 0 | 0 | 1 | 0 | 0 |
| 5 |  |  |  |  | - | 1 | 0 | 0 | 0 |
| 6 |  |  |  |  |  | - | 0 | 0 | 0 |
| 7 |  |  |  |  |  |  | - | 0 | 0 |
| 8 |  |  |  |  |  |  |  | - | 0 |
| 9 |  |  |  |  |  |  |  |  | - |

(c) Updated matrix



(d) Complexity graph

Fig. A1-3 (continued)

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | x | x | x | x | x | x | x | x | x | - |
| 2 | x | x | x | x | x | 1 | x | x | x | 6 |
| 3 | x | x | x | x | x |  | 1 | ← | x | 7 |
| 4 | x | x | x | x | x | x | ↧ | x | x | - |
| 5 | x | x | x | x | x |  | x | x | x | - |
| 6 | x | x | x | x | x | x | x | x | x | - |
| 7 | x | x | x | x | x | x | x | x | x | - |
| 8 | x | x | x | x | x | x | x | x | x | - |
| 9 | x | x | x | x | x | x | x | x | x | - |
|  |  |  |  |  |  | 5 | 4 | 3 |  |  |

Fig. A1-4

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |  |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | x | x | x | x | x | x | x | x | x | - |
| 2 | x | x | x | x | x | 1 | x | x | x | 6 |
| 3 | x | x | x | x | x |  |  | 1 | x |  |
| 4 | x | x | x | x | x | x | 1 | x | x |  |
| 5 | x | x | x | x | x |  | x | x | x | - |
| 6 | x | x | x | x | x | x | x | x | x | - |
| 7 | x | x | x | x | x | x | x | x | x | - |
| 8 | x | x | x | x | x | x | x | x | x | - |
| 9 | x | x | x | x | x | x | x | x | x | - |
|  |  |  |  |  |  | 5 |  |  |  |  |

Fig. A1-5

Table 1

| | Activity $a_1$ | Activity $a_2$ |
|---|---|---|
| Duration | **12** | 10 |
| Cost | 3 | 1 |
| Index | 1 | 1 |
| Duration | 9 | **10** |
| Cost | 9 | 1 |
| Index | 2 | 1 |
| Duration | **9** | 8 |
| Cost | 9 | 5 |
| Index | 2 | 2 |
| Duration | 7 | **8** |
| Cost | 15 | 5 |
| Index | 3 | 2 |
| Duration | **7** | 7 |
| Cost | 15 | 10 |
| Index | 3 | 3 |

| | | |
|---|---|---|
| New array: | Duration | 12, 10, 9, 8, 7 |
| | Cost | 4, 10, 14, 20, 25 |

Table 2

| Act. $a_2$ Act. $a_1$ | 12<br>3 | 9<br>9 | 7<br>15 |
|---|---|---|---|
| 10<br>1 | 22<br>4 | 19<br>10 | 17<br>16 |
| 8<br>5 | 20<br>8 | 17<br>14 | 15<br>20 |
| 7<br>10 | 19<br>13 | 16<br>19 | 14<br>25 |

Table 3: A reduction plan and its consequences

| Action | Activities | New | Array |
|---|---|---|---|
| 1. REDUCE1 | [ 2, 6] | : 15 | 8(6),7(7) |
| 2. REDUCE2 | [ 2, 7] | : 16 | 11(4),6(7) |
| 3. REDUCE1 | [ 3, 8] | : 17 | 7(5),5(8) |
| 4. PARALLEL | [17, 16] | : 18 | 11(9),7(12),6(15) |
| 5. REDUCE2 | [ 3, 9] | : 19 | 4(3), 3(7) |
| 6. SERIES | [18, 13] | : 20 | 15(10),11(13),10(16),9(18),8(21) |
| 7. PARALLEL | [20, 19] | : 21 | 15(13),11(16),10(19),9(21),8(24) |
| 8. SERIES | [21, 14] | : 22 | 20(16),16(19),15(22),14(24),13(27),12(29),11(32) |
| 9. REDUCE1 | [12, 15] | : 23 | 11(13),10(14) |
| 10. PARALLEL | [23, 22] | : 24 | 20(29),16(32),15(35),14(37),13(40),12(42),11(45) |
| 11. REDUCE2 | [12, 5] | : 25 | 7(3),5(6) |
| 12. REDUCE2 | [12, 10] | : 26 | 7(1),5(4) |
| 13. PARALLEL | [26, 11] | : 27 | 7(3),5(6) |
| 14. SERIES | [27, 4] | : 28 | 13(7),11(10),10(11),8(14) |
| 15. PARALLEL | [28, 25] | : 29 | 13(10),11(13),10(14),8(17) |
| 16. SERIES | [29, 1] | : 30 | 18(13),16(16),15(17),13(20),11(26) |
| 17. PARALLEL | [30, 24] | : 31 | 20(42),18(45),16(48),15(52),14(57),13(60),12(68),11(71) |
| EVALUATE | 31 | | |

Table 4. Computational results of Experiment 1.

| Network | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nodes \|N\| | 7 | | 8 | | 10 | | 6 | | 7 | | 8 | |
| Arcs \|A\| | 10 | | 12 | | 14 | | 11 | | 12 | | 13 | |
| Complexity Index | 2 | | 2 | | 2 | | 3 | | 4 | | 4 | |
| Modes | 16 | 16 | 16 | 16 | 16 | 16 | 94 | 64 | 256 | 256 | 256 | 416 |
| 4 | 0.00 | 0.00 | 0.00 | 0.00 | 0.05 | 0.00 | 0.06 | 0.05 | 0.05 | 0.11 | 0.16 | 0.22 |
| 5 | 25 | 25 | 25 | 25 | 25 | 25 | 140 | 140 | 625 | 625 | 625 | 975 |
| | 0.00 | 0.00 | 0.00 | 0.00 | 0.11 | 0.11 | 0.11 | 0.11 | 0.33 | 0.33 | 0.50 | 0.60 |
| 6 | 36 | 36 | 36 | 36 | 36 | 36 | 252 | 281 | 1296 | 1296 | 1296 | 2196 |
| | 0.05 | 0.05 | 0.11 | 0.05 | 0.22 | 0.17 | 0.22 | 0.28 | 0.71 | 0.77 | 1.37 | 1.92 |
| 7 | 49 | 49 | 49 | 49 | 49 | 49 | 434 | 563 | 2450 | 2401 | 2401 | 4022 |
| | 0.06 | 0.11 | 0.16 | 0.11 | 0.39 | 0.39 | 0.44 | 0.71 | 1.71 | 1.81 | 3.13 | 4.51 |
| 8 | 64 | 64 | 64 | 64 | 64 | 64 | 712 | 1040 | 4400 | 4096 | 4104 | 7352 |
| | 0.17 | 0.11 | 0.22 | 0.17 | 0.72 | 0.65 | 0.99 | 1.54 | 3.84 | 3.79 | 6.86 | 10.49 |
| 9 | 81 | 81 | 81 | 81 | 81 | 81 | 990 | 1583 | 6804 | 6804 | 6662 | 11466 |
| | 0.28 | 0.17 | 0.39 | 0.22 | 1.21 | 1.21 | 1.98 | 2.74 | 9.17 | 8.18 | 13.46 | 21.20 |
| 10 | 100 | 100 | 100 | 100 | 100 | 100 | 1410 | 2584 | 10800 | 11238 | 10632 | 18014 |
| | 0.44 | 0.27 | 0.55 | 0.33 | 1.98 | 1.92 | 3.57 | 5.38 | 18.40 | 17.80 | 27.46 | 44.82 |

Table 5. Computational results of Experiment 2.

| Network | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nodes \|N\| | 7 | | 8 | | 10 | | 6 | | 7 | | 8 | |
| Arcs \|A\| | 10 | | 12 | | 14 | | 11 | | 12 | | 13 | |
| Complexity Index | 2 | | 2 | | 2 | | 3 | | 4 | | 4 | |
| Replicate | 16 | 16 | 27 | 27 | 35 | 35 | 350 | 753 | 882 | 1323 | 1968 | 1968 |
| 1 | 0.00 | 0.00 | 0.05 | 0.06 | 0.16 | 0.17 | 0.33 | 0.55 | 0.71 | 0.88 | 3.08 | 2.97 |
| 2 | 28 | 36 | 12 | 12 | 6 | 6 | 56 | 105 | 474 | 2156 | 824 | 1062 |
| | 0.06 | 0.06 | 0.05 | 0.06 | 0.06 | 0.00 | 0.06 | 0.06 | 0.38 | 0.83 | 0.82 | 0.72 |
| 3 | 6 | 12 | 2 | 2 | 70 | 70 | 37 | 53 | 996 | 996 | 334 | 806 |
| | 0.00 | 0.00 | 0.00 | 0.00 | 0.16 | 0.17 | 0.06 | 0.06 | 0.88 | 0.88 | 0.28 | 0.82 |
| 4 | 12 | 36 | 32 | 32 | 48 | 48 | 96 | 96 | 44 | 418 | 979 | 4516 |
| | 0.05 | 0.06 | 0.05 | 0.05 | 0.55 | 0.49 | 0.05 | 0.05 | 0.11 | 0.44 | 1.10 | 3.62 |
| 5 | 18 | 18 | 6 | 6 | 3 | 3 | 256 | 343 | 731 | 2225 | 1427 | 1427 |
| | 0.06 | 0.00 | 0.05 | 0.00 | 0.05 | 0.06 | 0.33 | 0.33 | 1.21 | 1.86 | 2.37 | 2.20 |
| 6 | 8 | 16 | 14 | 14 | 12 | 12 | 507 | 701 | 168 | 2226 | 3579 | 4358 |
| | 0.00 | 0.00 | 0.06 | 0.06 | 0.17 | 0.11 | 0.88 | 0.99 | 0.28 | 0.77 | 3.46 | 7.75 |
| 7 | 49 | 49 | 5 | 5 | 72 | 72 | 213 | 276 | 894 | 1750 | 1046 | 2438 |
| | 0.11 | 0.11 | 0.06 | 0.05 | 0.27 | 0.17 | 0.28 | 0.33 | 0.66 | 1.04 | 1.37 | 2.25 |
| 8 | 60 | 60 | 8 | 15 | 10 | 10 | 30 | 55 | 1234 | 1234 | 2048 | 4405 |
| | 0.11 | 0.00 | 0.05 | 0.00 | 0.05 | 0.05 | 0.00 | 0.05 | 1.81 | 1.75 | 2.47 | 4.50 |
| 9 | 4 | 4 | 18 | 18 | 8 | 8 | 162 | 218 | 708 | 2244 | 775 | 1398 |
| | 0.06 | 0.11 | 0.11 | 0.06 | 0.05 | 0.05 | 0.17 | 0.16 | 0.83 | 1.27 | 0.93 | 1.48 |
| 10 | 32 | 32 | 1 | 4 | 100 | 100 | 354 | 686 | 218 | 2020 | 795 | 795 |
| | 0.06 | 0.05 | 0.00 | 0.00 | 0.32 | 0.33 | 0.77 | 0.77 | 0.55 | 1.15 | 0.72 | 0.71 |
| Average | 23.3 | 27.9 | 12.5 | 13.5 | 36.4 | 36.4 | 206 | 328 | 634 | 1659 | 1377 | 2317 |
| | 0.06 | 0.04 | 0.05 | 0.03 | 0.18 | 0.16 | 0.29 | 0.34 | 0.74 | 1.09 | 1.66 | 2.70 |

Table 6. Computational results of Experiment 3.

| Network | 1 | | 2 | | 3 | | 4 | |
|---|---|---|---|---|---|---|---|---|
| Nodes \|N\| | 12 | | 12 | | 20 | | 20 | |
| Arcs \|A\| | 21 | | 24 | | 37 | | 42 | |
| Complexity Index | 4 | | 6 | | 8 | | 12 | |
| Replicate | 20 | 20 | 64 | 64 | 320 | 352 | 4096 | 13520 |
| 1 | 0.16 | 0.00 | 0.11 | 0.00 | 15.49 | 0.22 | 13.79 | 4.07 |
| 2 | 16 | 16 | 64 | 64 | 256 | 256 | 4608 | 6400 |
| | 0.16 | 0.00 | 0.11 | 0.06 | 15.11 | 0.27 | 13.84 | 2.26 |
| 3 | 30 | 30 | 96 | 96 | 256 | 256 | 4096 | 8320 |
| | 0.22 | 0.05 | 0.17 | 0.06 | 15.82 | 0.22 | 13.79 | 3.52 |
| 4 | 26 | 26 | 96 | 96 | 392 | 524 | 4096 | 5760 |
| | 0.16 | 0.00 | 0.11 | 0.06 | 15.98 | 0.50 | 14.28 | 2.70 |
| 5 | 20 | 20 | 64 | 64 | 256 | 320 | 4608 | 12688 |
| | 0.22 | 0.05 | 0.16 | 0.00 | 15.55 | 0.44 | 14.06 | 6.38 |
| 6 | 30 | 30 | 88 | 88 | 320 | 480 | 4096 | 4608 |
| | 0.22 | 0.00 | 0.17 | 0.06 | 15.82 | 0.44 | 13.89 | 2.26 |
| 7 | 24 | 24 | 96 | 96 | 256 | 320 | 5120 | 5760 |
| | 0.17 | 0.00 | 0.11 | 0.05 | 15.82 | 0.44 | 14.22 | 2.75 |
| 8 | 20 | 20 | 104 | 104 | 256 | 384 | 7488 | 16256 |
| | 0.16 | 0.06 | 0.17 | 0.06 | 15.82 | 0.44 | 14.67 | 5.94 |
| 9 | 16 | 16 | 64 | 64 | 256 | 320 | 4096 | 8640 |
| | 0.16 | 0.06 | 0.11 | 0.06 | 15.76 | 0.33 | 14.12 | 3.46 |
| 10 | 16 | 16 | 88 | 88 | 256 | 320 | 5184 | 11880 |
| | 0.16 | 0.06 | 0.17 | 0.05 | 15.82 | 0.33 | 14.23 | 5.54 |
| Average | 21.8 | 21.8 | 82.4 | 82.4 | 282.4 | 353.2 | 4748 | 9383 |
| | 0.18 | 0.03 | 0.14 | 0.05 | 15.70 | 0.36 | 14.09 | 3.89 |

Table 7. Computational results of Experiment 4.

| Network | 1 | | 2 | | 3 | | 4 | |
|---|---|---|---|---|---|---|---|---|
| Nodes \|N\| | 12 | | 12 | | 20 | | 20 | |
| Arcs \|A\| | 21 | | 24 | | 37 | | 42 | |
| Complexity Index | 4 | | 6 | | 8 | | 12 | |
| Replicate | 99 | 99 | 864 | 864 | 6561 | 8100 | 610173 | 2026992 |
| 1 | 0.27 | 0.11 | 0.44 | 0.33 | 20.76 | 7.03 | 234.04 | 1034.03 |
| 2 | 129 | 129 | 840 | 840 | 6561 | 6561 | 839808 | 2402190 |
| | 0.33 | 0.17 | 0.54 | 0.44 | 24.60 | 9.23 | 364.32 | 1839.73 |
| 3 | 94 | 94 | 972 | 972 | 6561 | 8748 | 632772 | 1650348 |
| | 0.27 | 0.11 | 0.60 | 0.61 | 26.53 | 17.08 | 297.31 | 1251.20 |
| 4 | 144 | 144 | 945 | 945 | 7290 | 11664 | 782217 | 5343975 |
| | 0.38 | 0.22 | 0.66 | 0.55 | 24.22 | 18.02 | 364.98 | 3619.92 |
| 5 | 201 | 201 | 1271 | 1271 | 8505 | 10890 | 551124 | 831060 |
| | 0.60 | 0.33 | 0.82 | 0.77 | 26.48 | 14.89 | 344.22 | 837.07 |
| 6 | 150 | 150 | 840 | 840 | 8019 | 15552 | 708588 | 4839912 |
| | 0.49 | 0.22 | 0.61 | 0.55 | 30.38 | 25.65 | 452.75 | 3482.00 |
| 7 | 155 | 155 | 756 | 756 | 8748 | 14580 | 656100 | 1692738 |
| | 0.66 | 0.39 | 0.60 | 0.49 | 29.50 | 26.47 | 354.05 | 1416.96 |
| 8 | 114 | 114 | 1015 | 1015 | 8181 | 14703 | 634230 | 1831500 |
| | 0.33 | 0.16 | 0.77 | 0.60 | 31.08 | 28.84 | 347.07 | 1271.09 |
| 9 | 157 | 157 | 945 | 945 | 7776 | 13860 | 1148175 | 2843910 |
| | 0.54 | 0.28 | 0.66 | 0.55 | 31.59 | 26.91 | 640.55 | 2323.62 |
| 10 | 147 | 147 | 870 | 870 | 8019 | 15795 | 852930 | 5141565 |
| | 0.61 | 0.33 | 0.60 | 0.43 | 33.78 | 24.71 | 556.56 | 4833.28 |
| Average | 139.0 | 139.0 | 931.8 | 931.8 | 7622 | 12045 | 741611 | 2860419 |
| | 0.45 | 0.23 | 0.63 | 0.53 | 27.89 | 19.88 | 395.59 | 2190.89 |

Table 8. Computational results of Experiment 5.

| Network | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Nodes \|N\| | 11 | | 11 | | 11 | | 11 | | 11 | | 11 | | 11 | |
| Arcs \|A\| | 15 | | 20 | | 25 | | 30 | | 35 | | 40 | | 45 | |
| Complexity Index | 1 | | 4 | | 5 | | 5 | | 7 | | 7 | | 7 | |
| Replicate | 8 | 8 | 204 | 846 | 24 | 288 | 12422 | 12422 | 16160 | 458092 | 768791 | 821736 | 247971 | 200572 |
| 1 | 0.06 | 0.06 | 0.66 | 0.93 | 0.11 | 0.38 | 23.51 | 23.12 | 27.41 | 146.26 | 675.03 | 937.30 | 329.01 | 172.41 |
| 2 | 6 | 6 | 1228 | 5530 | 4590 | 29160 | 4560 | 8202 | 124285 | 1396092 | 373527 | 373527 | 215820 | 268101 |
| | 0.11 | 0.11 | 4.12 | 6.59 | 14.39 | 30.92 | 7.25 | 8.63 | 140.39 | 701.68 | 651.09 | 638.29 | 214.54 | 246.50 |
| 3 | 5 | 5 | 424 | 1350 | 56 | 624 | 6220 | 9029 | 102448 | 2506721 | 102872 | 400200 | 43380 | 43380 |
| | 0.06 | 0.00 | 2.14 | 1.48 | 0.27 | 0.94 | 12.96 | 16.70 | 157.30 | 1475.08 | 196.69 | 670.70 | 68.11 | 116.55 |
| 4 | 5 | 5 | 504 | 1330 | 4998 | 18326 | 3415 | 3415 | 12701 | 287372 | 108936 | 108936 | 589550 | 1086616 |
| | 0.06 | 0.06 | 0.88 | 1.31 | 13.84 | 32.24 | 12.03 | 11.70 | 24.17 | 107.93 | 200.54 | 191.58 | 1148.00 | 1520.72 |
| 5 | 5 | 5 | 132 | 2310 | 930 | 5016 | 147 | 203 | 140697 | 618426 | 328174 | 339020 | 157437 | 157437 |
| | 0.11 | 0.06 | 0.98 | 1.43 | 2.34 | 6.32 | 0.55 | 0.65 | 1351.61 | 798.61 | 756.71 | 826.35 | 315.55 | 403.59 |
| 6 | 2 | 2 | 54 | 1224 | 960 | 2160 | 10202 | 54728 | 36774 | 104751 | 351528 | 351528 | 230832 | 279800 |
| | 0.05 | 0.06 | 0.22 | 0.82 | 1.81 | 2.80 | 48.12 | 89.20 | 98.09 | 114.14 | 762.36 | 758.85 | 410.85 | 612.42 |
| 7 | 2 | 2 | 2025 | 4149 | 1175 | 12250 | 1244 | 2187 | 540500 | 3959135 | 719550 | 1066135 | 232680 | 612360 |
| | 0.11 | 0.00 | 3.08 | 3.63 | 4.94 | 12.52 | 3.40 | 3.07 | 754.90 | 3348.75 | 2248.96 | 2237.78 | 637.19 | 1363.15 |
| 8 | 1 | 1 | 234 | 268 | 963 | 12840 | 3208 | 3208 | 44274 | 104171 | 442295 | 376320 | 89706 | 89706 |
| | 0.06 | 0.06 | 0.39 | 0.27 | 2.41 | 12.47 | 15.05 | 14.28 | 45.86 | 72.45 | 601.87 | 443.14 | 171.37 | 187.57 |
| 9 | 8 | 8 | 1440 | 3075 | 16320 | 24072 | 24107 | 24107 | 41024 | 419558 | 60672 | 60672 | 73802 | 73802 |
| | 0.17 | 0.11 | 2.47 | 2.96 | 42.51 | 37.57 | 85.30 | 80.68 | 304.68 | 321.75 | 174.38 | 178.46 | 251.34 | 353.56 |
| 10 | 5 | 5 | 165 | 923 | 960 | 4080 | 3759 | 6621 | 243756 | 463266 | 221568 | 221568 | 111945 | 111945 |
| | 0.06 | 0.05 | 0.44 | 1.32 | 2.91 | 8.08 | 9.72 | 8.84 | 508.00 | 535.09 | 299.84 | 304.12 | 242.33 | 327.52 |
| Average | 4.7 | 4.7 | 641 | 2100 | 3097 | 10881 | 6928 | 12412 | 130261 | 1031758 | 347791 | 411964 | 199312 | 292371 |
| | 0.09 | 0.06 | 1.54 | 2.07 | 8.55 | 14.42 | 21.79 | 25.69 | 341.24 | 762.17 | 656.75 | 718.66 | 378.83 | 530.40 |

Table 9. Computational results of Experiment 6.

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1<br>0.00 | 1<br>0.00 | 6<br>0.00 | 6<br>0.00 | 18<br>0.00 | 22<br>0.00 | 41<br>0.06 | 54<br>0.06 | 80<br>0.05 | 109<br>0.05 |
| 10 | 150<br>0.11 | 205<br>0.11 | 260<br>0.17 | 350<br>0.16 | 422<br>0.22 | 545<br>0.28 | 642<br>0.27 | 783<br>0.33 | 924<br>0.38 | 1068<br>0.44 |
| 20 | 1256<br>0.49 | 1389<br>0.55 | 1588<br>0.61 | 1773<br>0.66 | 1896<br>0.71 | 2082<br>0.77 | 2265<br>0.88 | 2335<br>0.82 | 2470<br>0.88 | 2633<br>0.99 |
| 30 | 2763<br>0.99 | 2702<br>1.04 | 2778<br>1.04 | 2862<br>1.05 | 2918<br>1.05 | 2787<br>0.99 | 2726<br>0.99 | 2702<br>0.99 | 2758<br>1.04 | 2793<br>1.04 |
| 40 | 2422<br>0.88 | 2321<br>0.88 | 2325<br>0.93 | 2339<br>0.88 | 1960<br>0.77 | 1795<br>0.71 | 1749<br>0.66 | 1757<br>0.66 | 1769<br>0.72 | 1219<br>0.49 |
| 50 | 1183<br>0.50 | 1176<br>0.50 | 1178<br>0.49 | 1142<br>0.50 | 1145<br>0.44 | 694<br>0.33 | 696<br>0.33 | 657<br>0.27 | 654<br>0.27 | 654<br>0.27 |
| 60 | 654<br>0.33 | 351<br>0.16 | 316<br>0.17 | 316<br>0.17 | 315<br>0.16 | 312<br>0.17 | 312<br>0.16 | 154<br>0.05 | 151<br>0.06 | 121<br>0.06 |
| 70 | 121<br>0.11 | 118<br>0.05 | 118<br>0.05 | 55<br>0.00 | 40<br>0.06 | 40<br>0.00 | 33<br>0.05 | 33<br>0.00 | 33<br>0.00 | 33<br>0.00 |
| 80 | 33<br>0.00 | 8<br>0.00 | 8<br>0.00 | 5<br>0.00 | 5<br>0.00 | 5<br>0.00 | 5<br>0.00 | 5<br>0.00 | 1<br>0.00 | |