Project Scheduling in the Presence of Productivity Functions

Daniel W. Steeneck

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Industrial and Systems Engineering

July 24, 2009
Blacksburg, Virginia

Keywords: Project Scheduling, Operations Research, Management Science

Project Scheduling in the Presence of Productivity Functions

Daniel W. Steeneck

(ABSTRACT)

A project scheduling problem is frequently encountered in real-life. Typical examples of its occurrence arise in construction, manufacturing, and military-related operations. In this thesis, we develop a solution methodology to determine the sequence in which to perform and the amount of resource to allocate to each activity of a project so as to minimize the project duration (makespan). We assume availability of only one resource type, which can be allocated to activities in varying amounts. We consider projects whose activities have durations which are defined by convex, non-increasing time-resource trade-off functions and whose activities are not pre-emptable (i.e., once some amount of resource has been allocated to an activity, this amount cannot be varied while the activity is processed). Also, we assume various forms of productivity (amounts of units produced per unit time) functions with respect to the amount of the resource allocated to an activity. Our solution methodology enables the determination of all potentially optimal sequences for a given project. However, rather than enumerating all possible sequences, we develop special relationships between certain pairs of activities to determine a priori how these pairs will be sequenced in relation to each other in an optimal solution. Then, the optimal resource allocations are determined for each sequence using a nonlinear program and the solution with the smallest makespan is selected. The use of this methodology is illustrated through experimentation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Problem Statement

1.1 Background

A project comprises inter-related activities that are required to accomplish an end goal/product. The process of determining when to perform which activity by using what amount of which type of resource is termed project planning. The objective of project planning is to schedule the activities so as to optimize a performance measure, which is, typically, the minimization of project duration, given the resource levels at hand. Many a time, a project is performed to produce a profit, in which case the objective would be to minimize the cost encountered in performing its activities while still completing the project on time. Project planning is an important step of management decision making in the construction industry. But, it is equally important in other industrial settings as well.

A project usually consists of a large number of activities, and thus, the determination of a schedule may involve intensive computations. Therefore, it is essential to develop efficient algorithms for the solution of project planning problems. It is for these reasons that project scheduling is a popular topic of investigation for management scientists and operations researchers. Probably, the most famous early work on this topic is that of Kelly and Walker[31] on the Critical Path Method in which the desired project completion time is determined by minimizing the longest (or critical) path, which starts at the first node and terminates at the last node of the project. An extensive amount of work has been reported since then on a variety of project planning problems. This includes, among others, consideration of a variety of features such as resource constraints, multiple resources and the cost incurred in performing an activity of a project as a function of the amount of resources allocated to it.

Even though many project planning problems can be addressed by the models and solution methodologies presented in the literature, there are still opportunities available to develop more effective models and to capture additional features that are encountered in real-world project planning problems. It is the goal of this thesis to present a novel model and solution approach for a commonly encountered project planning problem.

1.2 Problem Statement

We consider a directed network, P. It consists of a set of nodes and a set of arcs among the nodes. We represent the former as $R(P)$ and the latter as $A(P)$. A limited amount of a resource type is available to perform the activities of this network. Without loss of generality, we may assume this resource type to be workers. There is a known *time-resource trade-off* for each activity; in particular, the time required to perform an activity is a non-increasing function of the amount of resource allocated to it. We assume this function to be continuous, i.e., a fractional amount of the resource can be allocated to an activity. However, the amount of the resource allocated to an activity cannot be altered once the activity has begun processing, and, moreover, the activities are not pre-emptable.

The problem that we address in this thesis can formally be stated as follows: Given a directed network representing the precedence relationship among activities, the available amount of a resource type, and the time-resource trade-off function for each activity, determine the amount of the resource for allocation to each activity of the project so that the time required to perform all the activities of the project (i.e., project duration) is minimized. We consider a variety of time-resource trade-off functions for the activities of the project.

1.3 Motivation for Addressing the Problem

A project scheduling problem is frequently encountered in real-life. A typical example of its occurrence arises in the construction industry. However, many military-related applications can be viewed as project scheduling problems as well. Invariably, such problems involve a bottleneck resource, which can be the workers required to perform the activities of the project or a particular equipment. A procedure to determine the optimal resource allocation to the activities of a project has been presented in the literature in case the forms of inverse of the time-resource trade-off functions, or productivity functions, for all the activities are identical being either convex, concave or linear [51]. However, this procedure cannot address the situation when their productivity functions are mixed. Our work is motivated by the desire to develop an optimum-seeking method of this more general situation in which each activity could possess a convex, a concave, or a non-convex/concave time-resource trade-off function form.

1.4 Outline of Solution Approach

Our solution approach relies on three main ideas. First, we consider pairs of independent activities and develop conditions under which they can be performed in series or in parallel. Once such a relationship is determined among pairs of activities, sub-optimal sequences are avoided. This reduction in the solution search space can be quite significant depending upon the number of pairs of activities for which such relationships are known. Second, all possible potentially optimal sequences are enumerated using a search tree. As these sequences are enumerated, relationships among the remaining nodes are once again determined to further reduce the solution search space. Third, a network flow-based optimization method is used to determine an optimal allocation of the resource to the activities of each sequence, and the best solution is selected.

1.5 Outline of Remaining Chapters

In the remaining part of this thesis, we present a literature review of the previous work reported in the area of project planning in Chapter 2. A mathematical model of the problem addressed in this thesis is given in Chapter 3. A characterization of productivity functions and a method to estimate the productivity function for a given activity are presented in Chapter 4. In Chapter 5, we present the concept of independent activities and develop conditions for which it is known a priori that pairs of independent activities are sequenced in parallel or in series in an optimal solution. A search tree for finding all possible sequences of the activities in the project is presented in Chapter 6, which also contains our solution methodology. This search tree exploits the special relationships among certain pairs of activities to avoid consideration of many sub-optimal solutions. The solution methodology is illustrated using an example problem. Results and conclusions from some test cases are presented in Chapter 7. Finally, some suggestions for future work are given in Chapter 8.

Chapter 2

Literature Review

2.1 Project Planning

Project planning is the process of scheduling the activities of a project on one or more resources so as to optimize a performance measure. Typically, the performance measure has been the project duration, i.e., the total time required to process all the activities of the project. However, there can be other measures as well, like for instance, variation in the use of resources from one period to the next, and those pertaining to due date (meeting the due date, earliness/tardiness penalty, etc.). The critical path method [31] has been the cornerstone of many approaches proposed for the solution of project planning problems. The critical path of a project is the longest path from the starting to the terminal activities of the project. Building on this concept, many researchers have added to the body of knowledge on project scheduling by developing and solving project scheduling models that incorporate many features encountered in practice.

It is only realistic to assume a limited amount of one or more resources to be available to perform the activities of a project. The resources can either be renewable or non-renewable. Renewable resources continually replenish themselves, and therefore, their usage cannot exceed a given amount at any given time. The consumption of non-renewable resources can be limited over the duration of the project.

Another important feature of an activity of a project is its time-resource trade-off relationship. For many activities, an increase in the allocation of a resource for their performance will reduce their durations. Such a relationship will be nonlinear since an activity cannot have a negative processing time as the amount of resources goes to infinity. The reduction of project duration with increment in the amount of a resource for its processing is termed *project crashing*.

Another important feature of a project's activity is its ability to be or not to be pre-empted. A pre-emptable activity may be paused and resumed at any point in its processing with no effect on its duration. For a non-pre-emptable activity, if it is paused, it must be re-started from the beginning.

2.2 Representation of the Activities of a Project

Activity-on-Arc

Typically, the activities of a project and the relationships among them are represented by an *activity network* using an *activity-on-arc* representation. An activity-on-arc network is a directed graph, or a *digraph* with vertices representing *events* and arcs representing activities.

Consider a directed graph P with event set $R(P)$ and arc set $A(P)$. An individual arc, a_{ij} has tail i and arrowhead j, where i and j are the starting and ending events for a_{ij}. The direction of an arc represents the progression of time between events, and hence, it also represents the precedence relationships between activities. An activity, a_{ki} is an *immediate predecessor* of activity, a_{ij}. In other words, a_{ki} must finish before a_{ij} can begin. Likewise, a_{ij} is an *immediate successor* of a_{ki}. Consider the activity network shown in Figure 2.1. Activities a_{12} and a_{02} are immediate predecessors of a_{23}, and likewise, a_{23} is an immediate successor to both of these activities. There is an event time, t_i, associated with vertex i and a duration d_{ij} associated with activity a_{ij}. This duration may be a function of the amount of a resource allocated to the activity.

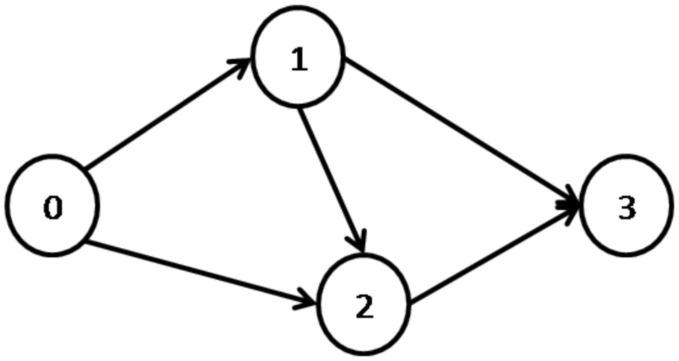


Figure 2.1: An activity-on-arc activity network

Activity-on-node

Another method of modeling activity networks is by using an *activity-on-node* representation. An activity-on-node network is a digraph with its vertices representing activities and its arcs representing precedence relationships. Consider a graph P' with vertex set $R(P')$ and arc set $A(P')$. An arc, a_{ij}, of this digraph indicates that activity i is an immediate predecessor of activity j. Likewise, activity j is an immediate successor of activity i. Consider the activity-on-node activity network shown in Figure 2.2. Note that activity 2 is a predecessor of activity 4. There is a time duration associated with each node. However, arcs represent precedence relationships among the activities, and they have no time duration associated with them.

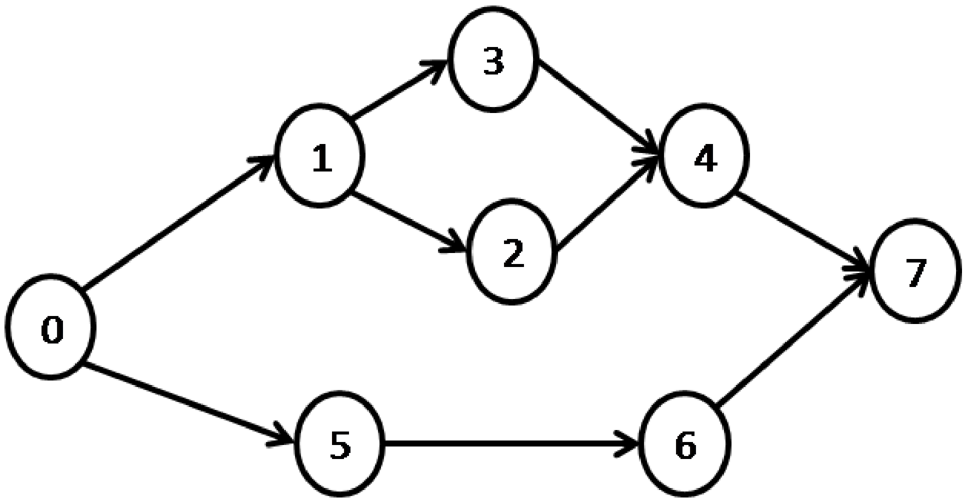


Figure 2.2: An activity-on-node activity network

2.3 The Critical Path Method and Precedence Relationships

The Critical Path Method

The critical path method (CPM) of Kelly and Walker[31] was the first attempt at mathematically modeling precedence relationships and costs associated with project activities. They split project planning into two parts, namely planning and scheduling. The planning aspect involves developing the activity network for the project and identifying the resource/time relationship for each activity. The scheduling aspect involves determining the optimal schedule and allocating resources to the activities of the project in order to minimize the cost incurred in view of a given due date.

The concepts that are central to CPM are *critical* activities and the *critical-path*, which is defined by the critical activities. If we let $t_i^{(0)}$ be the earliest possible realization time of node i, and $t_i^{(1)}$ be the latest possible realization time of node i, then the maximum time available for activity a_{ij} is $t_j^{(1)} - t_i^{(0)}$. If y_{ij}, the duration of (i, j), is equal to the maximum time available for a_{ij}, then a_{ij} is considered critical. If the completion time of a project with n jobs is λ, then λ is also the earliest possible time for the realization of the last event, $t_{n+1}^{(0)}$, and there exists a corresponding critical-path (or a critical sub-network) that consists of all the critical activities of the project. It also consists of the longest path(s) of the network.

If an activity is not on the critical path, then the duration of the activity may be lengthened by an amount called its *free float* before it becomes a critical activity. The idea of free float is important in reducing the resource consumption requirements for the project. Typically, a reduction in the allocation of resources to the activities will increase their durations. Therefore, a decrement in the allocation of a resource to a non-critical activity, in order to lengthen its duration, is advantageous if the objective is to minimize cost. Ideally, all the paths of the network will be critical in order to achieve minimum resource consumption.

To illustrate the idea of a critical activity and a critical path, consider the activity network shown in Figure 2.3. If the project completion time of the project represented in Figure 2.3 is 17, then the critical path of the project is as shown (by bold lines) in Figure 2.4.

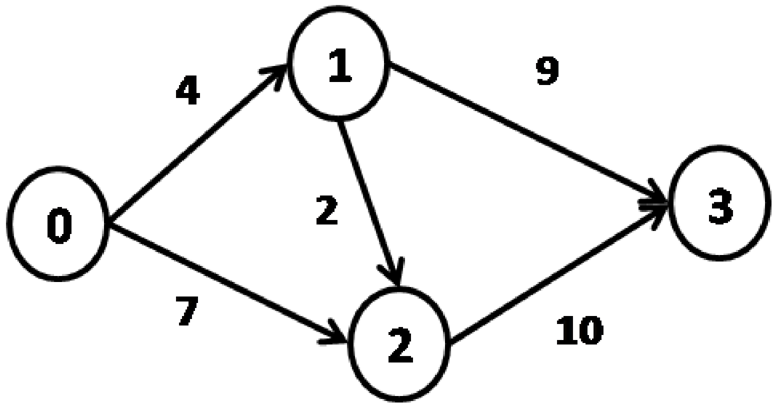


Figure 2.3: Example activity network

To illustrate the mathematical modeling of the precedence relationships of an activity network, consider the case of minimizing the cost incurred for completing the corresponding project by a given due date. Let n be the number of jobs, and d_{ij} be the duration of activity (i, j), a_{ij} be the \$/time cost of decreasing d_{ij}, and D be the project due date. Then, we have

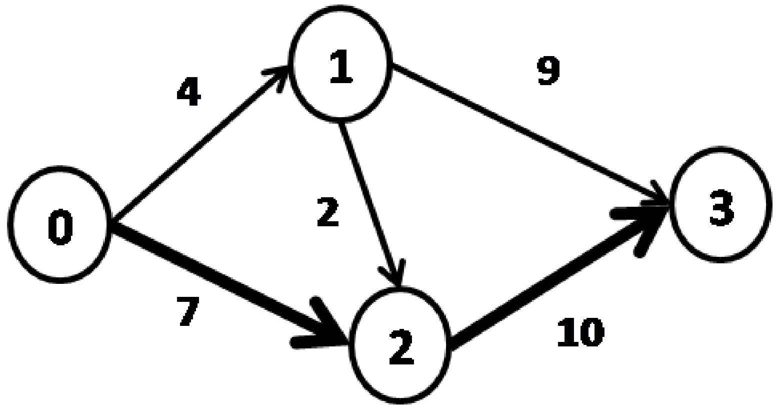


Figure 2.4: Example of a critical path

the following mathematical model:

$$\text{Minimize} \quad \sum_{\forall(i,j)\in \mathrm{A(P)}} -a_{ij}d_{i,j} \tag{2.1}$$

subject to:

$$d_{ij} \leq t_j - t_i, \qquad (i,j) \in A(P) \tag{2.2}$$

$$t_0 = 0 \tag{2.3}$$

$$t_n = \mathrm{D}. \tag{2.4}$$

The constraint set (2.2) defines the precedence relationships among the activities of the project. The constraint sets (2.3) and (2.4) capture the start and completion times of the project. The object is to achieve maximum cost reduction as depicted by (2.1).

Generalized Precedence Relationships

The precedence relationships used in the CPM are *strict* precedence relationships. However, there is another type of precedence relationship called generalized precedence relationships (GPRs) that is discussed in the literature (Elmaghraby and Kamburowski [24]). In the strict precedence model, the finish time of one activity is constrained to be equal to start time of its successor. However, there are other forms of "precedence" relationships between the start and finish times of two activities. They are: start-to-start (SS), start-to-finish (SF), finish-to-start (FS), and finish-to-finish (FF). For example, for a positive finish to start relationship of two activities, there will be some overlap and the second activity's start time will have a lead on the first activity's finish time. If the finish to start relationship is negative, then there will be some lag time between the activities.

GPRs can be represented on an activity-on-arc activity network, however, some more notation needs to be defined. Activity i, where $1 \leq i \leq n$, has a *start-event* node labeled $2i - 1$ and a *finish-event* node labeled $2i$. The start-event node will be called s_k and the finish-event node will be called f_i. Then, the activity network representing the GPRs will have $2n + 2$ nodes. The duration of job i, d_i, has upper and lower bounds u_i and l_i, respectively. Evidently

$$s_i + l_i \leq f_i \leq s_i + u_i \tag{2.5}$$

An arc pointing from node $2i - 1$ to $2i$ will have a length of l_i and an arc pointing in the opposite direction will have a length of $-u_i$. This representation is essential to satisfy the constraint that $t_j - t_i \geq d_{ij}$. An example GPR activity network is shown in Figure 2.5.

Projects with GPRs will not be considered for all the features covered in this literature review; however, projects planning problems with features such as time/cost trade-offs and resource constraints, which have been modeled using strict precedences can easily be modeled

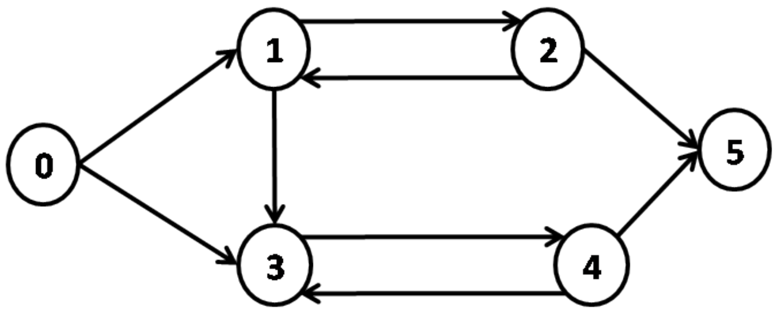


Figure 2.5: Example GPR activity network

for GPRs with small modifications.

2.4 Resource Constraints

An essential feature of almost all projects is that they must be performed using a limited amount of one or more resources. One obvious type of limited resource encountered in any project is money; however, in reality, this is usually a resource whose consumption is minimized rather than constrained. Resources that are constrained include: man-hours, machinery, raw materials, etc. Resources fall into the following categories: renewable, or non-renewable.

Renewable resources are those that are either replenished during each time period (fiscal year, quarter, etc.) or whose usage is available at some constant rate (workers/time, power, trees (provided they are replanted), etc.). In the *Resource Constrained Project Scheduling Problems* or *RCPSP*, time is broken down into periods. All classical formulations of the RCPSP have the same basic structure that Christofides et. al. [10] use to deal with consumption-constrained resources. This includes the objective of minimizing the project duration, cost incurred, and the like, given the precedence constraints among the activities, and the following resource constraints:

$$\sum_{i \in \gamma(t)} r_{ik} \leq R^u, \quad t = 1, \cdots, T, \quad k = 1, \cdots, K \tag{2.6}$$

where $\gamma(t)$ represents the activities in operation during time period t, r_{ik} is amount of resource k required by activity i, and R^u is the total amount of resource type k available for the project in any time period. Typically, this formulation is solved using a branch-and-bound method, which is the most studied technique for solving the RCSPS (Balas [2], Davis and Heidorn [12], Hastings [27], Stinson et al. [48], Talbot and Patterson [50], Radermacher [43], Christofides et al. [10], Bell and Park [4], Carlier and Latapie [7], Mingozzi et al. [35], and Demeulemeester and Herroelen [21]).

Patterson and Huber [40] use a discrete-time-based scheme as well based on the work of Pritsker and Watters [41], and therefore, their resource usage constraint is slightly different:

$$\sum_{j=1}^{n} \sum_{q=t}^{t+d_j-1} r_{jk} x_{jq} \leq R_k^u, \quad t = 1, \cdots, T, \quad k = 1, 2, \cdots, K \tag{2.7}$$

In this case, t refers to a particular time period, d_j represents the duration of activity j in number of time periods, r_{jk} is the amount of resource k required by activity j, x_{jq} is a binary variable and indicates whether or not activity j is to be processed in period q, and R_k^u is the amount of resource k available for the project. Usually, if the problem is formulated in this manner, a zero-one programming technique (Bowman [6], Pritsker et al. [42], and Patterson and Huber [40]) is used to solve the model.

Dynamic programming can also be used for solving the RCSPS as shown by Carruthers and

Battersby[8].

We may also call the above the *usage*-constrained resources following the example of Weglarz [52] for continuous-time-based models. If a resource is usage-constrained, we have

$$\sum_{i=1}^{n} r_i(t) \leq R^u \quad \forall t \geq 0 \tag{2.8}$$

where $r_i(t)$ is the resource usage of job i at time t and N is the maximum resource usage at any given time. Leachman et. al. [32] model resource usage in a similar manner except that they call the resource usage of an activity as its *intensity.*

Resources that are available in limited amounts over the course of the project, are called *consumption-* constrained or *non-renewable* resources. Examples of consumption-constrained resources are money, man-hours, raw materials, among others. Weglarz [52] defines consumption-constrained resources as those possessing the following relationship:

$$\sum_{i=1}^{n} \int_{0}^{\mathrm{T}} r_i(t)dt \leq \mathrm{M} \tag{2.9}$$

Essentially, the resource usage for each activity integrated over the duration of the project is summed to obtain the total resource consumption. M is the maximum resource consumption. Slowinski [47] also uses a method similar to that of Weglarz [52] for the modeling of resources. The model by Patterson and Huber [40] does not consider consumption-constrained resources. However, it can be adapted for a consumption-constrained resource as follows:

$$\sum_{t=1}^{T} \sum_{j=1}^{n} \sum_{q=t}^{t+d_j-1} r_{jk}x_{jq} \leq \mathrm{M}_k, k = 1, 2, \cdots, \mathrm{K}. \tag{2.10}$$

Doubly-constrained resources include both usage and consumption constraints. Doubly-constrained resources include money, power, manpower, among others. To model a resource as doubly constrained, one would simply use both the usage constraints and consumption constraints of the resource in the model. Both Weglarz [52] and Slowinsky [47] feature doubly-constrained resources in their treatment of project planning.

2.5 Time-Resource Trade-offs and Activity Modes

The durations of the activities of a project are, invariably, a function of one or more types of resources. These functional relationships are known as the time-resource trade-offs. Their functional forms are typically linear or convex; however, they may, theoretically, take on any shape. Here, we only consider the linear and convex function forms since the linear case is a good base-case, and modeling diminishing returns requires only convex curves. The time-resource trade-off can also be modeled using discrete activity modes. These discrete activity modes may require different amounts of resources to achieve a particular duration. Allocating additional resources to an activity to reduce its duration is called project crashing. Project crashing is a realistic and, therefore, a popular feature to incorporate into project scheduling models. Various strategies can be used to model the time-resource trade-offs.

Continuous Time-Resource Functions

The linear time-resource trade-off curve is the simplest to handle; however, it is the most unrealistic. Elmaghraby [22], provides the following model for the project planning problem with linear cost-time trade-offs. Consider the following notation.

b_{ij} = y intercept of cost curve for activity (i, j)
u_{ij} = upper bound on duration for activity (i, j)
q_{ij} = lower bound on duration for activity (i, j)

$$\text{Minimize} \quad \sum_{(i,j)\in A(P)} c_{ij} = \sum_{(i,j)\in A} (b_{ij} - a_{ij}d_{ij}) \tag{2.11}$$

subject to:

$$t_j - t_i \geq d_{ij}, \qquad \forall (i, j) \in A(P) \tag{2.12}$$

$$d_{ij} \leq u_{ij}, \qquad \forall (i, j) \in A(P) \tag{2.13}$$

$$d_{ij} \geq q_{ij}, \qquad \forall (i, j) \in A(P). \tag{2.14}$$

The above model is a linear program and is, therefore, easy to solve. Elmaghraby [22] also presents methods for solving problems involving strictly convex cost-time trade-off curves based on the methods of Berman [5]. These methods assume unlimited availability of resources with activities following convex time-resource trade-off curves (see Figure 2.6).

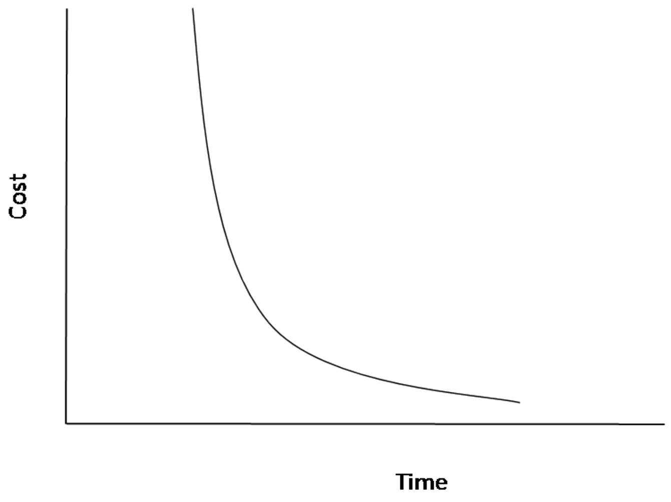


Figure 2.6: A convex time-resource trade-off curve

To complete a series of activities by a given due date, D, Berman [5] has shown that enough resources should be allocated to each activity to ensure that: (1) the project completion time is D, (2) the derivatives of the time-resource curves are less than or equal to zero, and (3) the derivatives of the time-resource curves for all the activities are equal to each other. It is clear that these are the optimal conditions since, at these optimal time-resource points, if we increase the time of one activity and reduce that of another, the total cost would increase. At "complex junctions"(events associated with more than two activities), we must regard the incoming activities at the junction as one job by summing their time-resource functions, and do the same for the outgoing activities. Consequently, the situation reduces to two activities in series and can be addressed by setting D as the time of the next event. The algorithm proposed by Berman [5] is as follows:

1. Set each activity at a fairly steep slope of the time-resource function, leaving enough slack for use at the latter part of the network. Introduce additional slack as required. This step provides an initial feasible solution.
2. Balance the junctions in succession, moving backwards through the network.
3. Iterate Step 2, until an acceptable degree of balance is obtained.

Note that Berman [5] does, in fact, handle functions that exhibit diminishing marginal productivities (i.e., each unit of addition resource leads to less productivity after a certain point).

Deckro and Hebert [18] have also investigated modeling of diminishing returns in project resource planning. They assume no resource constraints, and the objective is to either: (1)

find the minimal time for a given cost, or conversely, (2) find the minimal cost given a due date. Deckro and Hebert [18] call their model a *base diminishing return model* or BDRM. It can be expressed as follows. Consider the following notation.

LB_{ij}	= the minimum number of resource bundles required by activity (i,j), $\forall(i,j) \in A(P)$
UB_{ij}	= the maximum number of resource bundles that may be allocated to activity(i,j), $\forall(i,j) \in A(P)$
F_{ij}	= the fixed cost associated with activity(i,j), $\forall(i,j) \in A(P)$
OH	= the project overhead cost per period

Model BDRM:

$$\text{Minimize} \qquad OH + \sum_{ij \in A(P)} (c_{ij} r_{ij}) + \sum_{ij \in A} F_{ij} \tag{2.15}$$

Subject to:

$$-t_i + t_j \geq d_{ij}(r_{ij}), \qquad \forall(i,j) \in A(P) \tag{2.16}$$

$$LB_{ij} \leq r_{ij} \leq UB_{ij}, \qquad \forall(i,j) \in A(P) \tag{2.17}$$

$$t_i \geq 0, \qquad i = 1, 2, \cdots, n. \tag{2.18}$$

The constraint set (2.16) ensures that the precedence relationships hold. Constraint set (2.17) captures the upper and lower bounds on the amount of resource allocated to different activities. However, note that the duration functions of the activities are not linear. Deckro and Hebert [18] propose to solve a linear approximation model (LADRM) of the above problem. For the linear approximation model, they suggest to interpolate between the upper bound and lower bound points of the nonlinear duration functions to obtain the corresponding linear approximation values. Consequently, if d_{ij}^{+} is the value of $d_{ij}(LB_{ij})$, one may replace the second constraint of the BDRM with

$$-t_i + t_j \geq d_{ij}^{+} + a_{ij}(r_{ij} - LB_{ij}), \quad \forall (i,j) \in A(P). \tag{2.19}$$

A drawback of the linear approximation method is that it is not very accurate in most cases even though the underlying model is very efficient to solve. Also, for certain types of productivity functions, the nonlinear problem may be solved relatively easily with a commercially available solver. Even though the problem is not linear, it is still possible to decompose it using Benders partitioning or Lagrangian relaxation to reduce the size of the nonlinear part of the problem.

In a previous paper, Deckro et. al. [19] consider the durations of activities as decision variables. The cost of the project varies with the square of the difference between an activity's duration and its "normal" duration. This model is easily solvable using a quadratic programming solver.

Weglarz [52] handled the time-resource trade-off relationship for an activity a bit differently by working with the production rate as a function of the amount of resource allocated rather than directly working with time as a function of resource. The production rate function is defined as:

$$\frac{dX_i(t)}{dt} = \begin{cases} f_1(r_i(t)) & t \in (t_i, T_i) \\ 0 & \text{otherwise} \end{cases}, \quad i = 1, 2, \cdots, n, \tag{2.20}$$

where X_i is a measure of work related to the performance of activity, i. For example, if X_i is the number of acres to be plowed for activity i, then $X_i(t)$ is the number of acres that are plowed at time t. Then, Equation (2.20) represents the rate at which activity i is performed at time t with resource level $r_i(t)$ allocated to it. This is called the *productivity function*. Interestingly, if the productivity functions of all the activities are of the same form (either linear, concave or convex), then there is a fairly straight-forward algorithm to determine the optimal resource allocation to each activity. Since this section pertains only to time-resource trade-offs, this method will be discussed in a later section of the literature review.

Activity Modes

An activity may only be performed in a limited number of ways. These are called the *modes* of an activity. For example, if waiting for concrete to dry is an activity of a project network, then that activity has two modes: use of fast drying concrete or normal concrete. Each option has a different cost and/or resource requirement and duration. When there is a time-resource trade-off in the presence of activity modes, the problem is called the *Discrete Time-Cost Trade-off Problem* or *DTCTP*.

Model DTCTP (De et al. [14]):

$$\text{Minimize} \quad \sum_{1 \le i \le n} \sum_{1 \le j \le m(i)} c_{ij} g_{ij} \tag{2.21}$$

subject to:

$$\sum_{1 \le j \le m(i)} g_{ij} = 1, \qquad i = 1, \cdots, n \tag{2.22}$$

$$\sum_{1 \le j \le a(i)} d_{ij} g_{ij} + t_i \le t_k, \qquad \forall k \in S(i) \tag{2.23}$$

$$i = 1, \cdots, n$$

$$t_{n+1} \le d, \tag{2.24}$$

where c_{ij} is the cost of activity i in mode j, $g_{i,j}$ is 0 or 1 depending on whether or not activity i is performed in mode j, $m(i)$ is the set of modes for activity i, $d_{i,j}$ is the duration of job i in mode j, $S(i)$ are the successors of activity i, and t_i is the start time of activity i.

A few methods have been developed to solve this problem. Panagiotakopoulos [39] and Harvey and Patterson [26] have developed special solution algorithms; however, Elmaghraby [23] and Demeulemeenster et al. [21] have developed hybrid branch-and-bound/dynamic programming methods to solve the problem. De et al. [13] has proven the problem to be NP hard.

Deckro and Hebert [18] have considered a modification of the DTCTP in the presence of strict precedence relations, which features modes but not resource constraints. Let

$L(P) =$	the set of all activities (i, j) whose $d_{ij}(r_{ij})$ is a linear function
$N(P) =$	the set of all activities (i, j) whose $d_{ij}(r_{ij})$ is a non-linear function
$z_{ij} =$	the number of options available for activity $(i, j) \in N(P)$

Model DDRM:

$$\text{Minimize} \quad OHx_n + \sum_{(ij)\in N(P)} \sum_{k=1}^{z_{ij}} (y_{ijk} c_{ik} r_{ij}(k)) + \sum_{(ij)\in L} (r_{ij} c_{ij}) \tag{2.25}$$

subject to:

$$-x_i + x_j - \sum_{k=1}^{z_{ij}} d_{ijk} y_{ijk} \geq 0, \qquad \forall (i,j) \in N(P) \tag{2.26}$$

$$\sum_{k=1}^{z_{ij}} y_{ijk} = 1, \qquad \forall (i,j) \in N(P) \tag{2.27}$$

$$-x_i + x_j - S_{ij} r_{ij} \geq D^{+}_{ij} - S_{ij} LB_{ij}, \qquad \forall (i,j) \in L(P) \tag{2.28}$$

$$LB_{ij} \leq r_{ij} \leq UB_{ij}, \qquad \forall (i,j) \in A(P) \tag{2.29}$$

$$r_{ij}(k) \in I, \qquad \forall (i,j), \in N(P) \tag{2.30}$$

$$y_{ij}(k) \in \{0,1\}, \qquad \forall (i,j), \in N(P) \tag{2.31}$$

$$x_i \geq 0, \qquad i = 1, 2, \cdots, n, \tag{2.32}$$

where $y_{ijk} = 1$ if $r_{ij}(k)$ bundles of resource (i,k) are allocated to activity (i,k), and 0, otherwise. Note that this model is similar to model BDRM. For this model, there are different modes in which each activity can be performed. These modes are set by changing the value of y_{ijk} to one or zero. The model is an integer program and may be solved by a number of commercial solvers.

Sakellaropoulos and Chassiakos [46] present a model that is similar to the DDRM of Deckro and Hebert's [18]. It is based on an activity-on-node model representation:

ϵ	a very small number designed to drive f_0 to 0
f_0	project finish time
f_i	finish time of activity $i \quad i, \cdots, n$
s_0	project start time $\quad i, \cdots, n$
s_i	start time activity i

$$\text{Minimize} \qquad \sum_{i \in R(P)} \sum_{k \in M(i)} c_{ik} m_{ik} + OHf_0 + \epsilon f_0 \tag{2.33}$$

Subject to:

$$f_i - s_i = \sum_{k \in M(i)} d_{ik} m_{ik}, \qquad \forall i \in R(P) \tag{2.34}$$

$$\sum_{k \in M(i)} m_{ik} = 1, \qquad \forall i \in R(P) \tag{2.35}$$

$$f_0 \geq f_i, \qquad \forall i \in R(P) \tag{2.36}$$

$$s_0 \leq s_i, \qquad \forall i \in R(P) \tag{2.37}$$

$$s_0 = 0, \tag{2.38}$$

$$s_i \geq 0, \qquad \forall i \in R(P) \tag{2.39}$$

$$f_i \geq 0, \qquad \forall i \in R(P) \tag{2.40}$$

$$m_{ik} = \{0,1\}, \qquad i = 1, 2, \cdots, n \tag{2.41}$$

The model is a strict precedence model. Constraint set (2.34) maintains the precedence constraints. Constraint set (2.35) ensures that some mode of performance is chosen for each activity. Constraint sets (2.36) to (2.41) ensure that feasible start and finish times for the activities are found in the solution. This model may be modified easily to handle GPRSs by adding the appropriate constraints as follows:

Start no earlier than time D:	$s_i \geq D$
Start no later than time D:	$s_i \leq D$
Start on time D:	$s_i = D$
Finish no earlier than time D:	$f_i \geq D$
Finish no later than time D:	$f_i \leq D$
Finish on time D:	$f_i = D$

Many other researchers have used activity modes when modeling project planning problems. These include Nudtasomboom and Randhawa [37], Erenguc, Ahn, and Conway [25], Deckro and Hebert again [17], and De Reyck and Herroelen [16]. These models consider the resource constraint problem, and we will present them in the next section.

2.6 Resource-Constrained Project Planning with Time-Resource Trade-offs

The most general case of project planning problem includes resource constraints along with time-resource trade-offs. The most common versions of these are the *Discrete Time-Resource Trade-off Problem (DTRTP)*, and the *Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP)*. Also, the DTCTP can be a resource constrained problem if the objective is to minimize project duration subject to a budget constraint. These involve modes and discrete times rather than a smooth time-cost or time-resource function. The DTRTP features discrete time-resource trade-offs as well as a constraint for the usage of a renewable resource. This last feature lends itself to a model using time-periods rather than continuous time in its formulation. The DTRTP model as presented by De Reyck et al. [15] is as follows:

$$\text{Minimize} \qquad \sum_{t=e_n}^{l_n} t \times x_{nlt} \tag{2.42}$$

Subject to:

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} x_{imt} = 1, \qquad i = 1, \cdots, n \tag{2.43}$$

$$(-\sum_{m=1}^{M_j} \sum_{t=e_j}^{j_j} t \times x_{jmt}) + \tag{2.44}$$

$$\sum_{m=1}^{M_i} \sum_{t=e_i}^{l_i} (t + d_{im}) x_{imt} \leq 0, \qquad i = 1, 2, \cdots, n$$

$$x_{ik} \geq 0, M_k \geq 0 \qquad \forall (i,j) \in A(P) \tag{2.45}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{im} \sum_{s=max(t-d_{im},e_i)}^{min(t-l,l_i)} x_{ims} \leq R, \qquad t = 1, \cdots, T \tag{2.46}$$

$$x_{imt} \in \{0, 1\} \qquad i = 1, \cdots, n \quad m = 1, \cdots, M \tag{2.47}$$
$$t = 0, \cdots, T.$$

Here, x_{imt} indicates if activity i is performed in mode m and started at time t, and $e_i(l_i)$ is the earliest (latest) start time of activity i based on the modes with the smallest duration. Since the problem is NP-hard, there is no polynomial-time algorithm available for its solution. The various methods that have been used for its solution are: branch-and-bound (Demeulemeester et al [20]), tabu search (De Reyck et al. [15]), genetic algorithm (Ranjibar and Kianfar [45]), hybrid scatter search (Ranjibar et al. [44]). Ranjibar et al. [44] have shown that, for many modes, the scatter search method performed the best; however, for very few modes the branch-and-bound method of Demeulemeester et al.'s [20] performs better. Notice that this formulation is similar to Talbot's [49] MRCPSP model presented next. This follows because the DTRTP is a special case of the MRCPSP with no non-renewable resources.

Talbot[49] presents a resource-constrained time-resource trade-off, model which is mode-based, treats time discretely, and models both multiple renewable and nonrenewable resources.

$$\text{Minimize} \quad \sum_{m=1}^{M_n} \sum_{t=E_N}^{L_N} t \times x_{Ntm} \tag{2.48}$$

$$\text{subject to} \quad \sum_{m=1}^{M_n} \sum_{t=E_N}^{L_N} x_{jtm} = 1, \qquad j = 1, \cdots, N \tag{2.49}$$

$$-\sum_{m=1}^{M_a} \sum_{t=E_a}^{L_a} t \times x_{atm} + \tag{2.50}$$

$$\sum_{m=1}^{M_b} \sum_{t=E_b}^{L_b} (t - d_{bm}) x_{jtm} \geq 0, \qquad i = 1, 2, \cdots, n$$

$$\sum_{j=1}^{N} \sum_{m=1}^{M_j} \sum_{q=t}^{t+d_{jm}-1} r_{jkm} x_{jqm} \leq R^u_{kt}, \qquad k = 1, \cdots, K, t = 1, \cdots, H \tag{2.51}$$

$$\sum_{j=1}^{N} \sum_{m=1}^{M_j} \sum_{t=E_j}^{L_j} r_{jim} d_{jm} x_{jtm} \leq R^c_i, \qquad i = 1, \cdots, I \tag{2.52}$$

where $e_i(l_i)$ is the earliest finish time for activity i. For this formulation, E_j and L_j maybe determined by using a method presented by Levy and Weist [33]. In order to solve this model, Talbot [49] develops an algorithm that uses integer programming with network cuts, which is a more efficient method than solving it as a straightforward integer program. This problem has also been addressed by Chang et al. [9] (ant colony optimization), Wei-cun and Kai [53] (ant colony and particle swarm optimization), Liu et al. [34] (particle swarm optimization), Yu [54] (hybrid genetic algorithms) and Pan et al. [38] (clonal selection optimization), Mori and Tseng [36] and later Alcaraz et al. [1](genetic algorithms), Jarboui et al. [28] (particle swarm algorithm), and Jozeforwka et al. [30] (simulated annealing). Most recently, Ranjbar et al. [44] have developed a hybrid search procedure, which performs better than the algorithms by Alcaraz et al.[1] and Jozeforwka et al.[30].

Erenguc, Ahn and Conway [25] present a formulation and a solution method for the resource-constrained project planning problem with activity modes and time periods. The formulation is as follows:

$$\text{Minimize} \quad \sum_{j=2}^{n-1} \sum_{m_j \in M(j)} x_{jm_j} \times \{NC_{jm_j} + MCjm_j \times (ND_{jm_j} - d_j)\} + \bar{P} \times D_{over} \tag{2.53}$$

subject to:

$$\sum_{m_j \in M(j)} x_{jm_j} = 1, \qquad j = 2, \cdots, n-1, m_j \in M(j) \tag{2.54}$$

$$s_i + d_i \leq s_j, \qquad \forall (i,j) \in A(P) \tag{2.55}$$

$$\sum_{j \in A_t} \sum_{m_j \in M(j)} r_{jm_jk}, \qquad \forall k \text{ and } t \tag{2.56}$$

$$x_{jm_j} \leq R_k, \qquad \forall k \text{ and } t \tag{2.57}$$

$$CD_{jm_j} x_{jm_j} \leq d_j x_{jm_j} D_{jm_j} x_{jm_j}, \qquad j = 2, \cdots, n-1, m_j \in M(j) \tag{2.58}$$

$$x_{jm_j} \in \{0,1\}, \qquad j = 2, \cdots, n-1, m_j \in M(j) \tag{2.59}$$

$$s_1 = 0, d_1 = 0, d_n = 0 \tag{2.60}$$

$$z \geq 0 \tag{2.61}$$

$$z \geq s_j - D \tag{2.62}$$

$$d_j \text{ is integer}, \qquad j = 1, 2, \cdots, n. \tag{2.63}$$

Here, $ND_{jm_j}(CD_{jm_j})$ represents the longest(shortest) duration of m_j, $\bar{P}$ is the per unit time cost of tardiness, and D_over is the tardiness of project. This formulation is similar to the MRCPSP except that in addition to considering different activity modes, one can also crash an activity by increasing cost; however, there is no non-renewable resource constraint. The time-cost trade-off is considered to be linear. The authors solve the problem by a branch-and-bound procedure.

Nudtasomboom and Randhawa [37] develop a formulation very similar to MRCPSP as well except that they allow all types of constrained resources (renewable, non-renewable, and doubly constrained). Their solution algorithms are based on the methods presented by Talbot [49] and Davis [11], and Johnson [29].

Weglarz's [52] treatment of the time-resource trade-off problem with resource constraints problem uses a doubly constrained, continuously divisible resource and continuous time-resource trade-off curves. For sake of completion, Equations (2.20), (2.8), and (2.9) are presented again below along with some others.

$$\frac{dX_i(t)}{dt} = \begin{cases} f_1[r_i(t)] & \text{if } t \in (t_i, T_i), \quad i = 1, 2, \cdots, n \\ 0 \quad \text{otherwise} \end{cases} \tag{2.64}$$

$$\sum_{i=1}^{n} r_i(t) \leq N, \forall t \geq 0 \tag{2.65}$$

$$\sum_{i=1}^{n} \int_0^T r_i(t)dt \leq M \tag{2.66}$$

$$C(T) = \sum_{i=1}^{n} \int_0^T r_i(t)dt. \tag{2.67}$$

$C(T)$ represents the total resource consumption over the duration of the project. Weglarz [52] assumes that all the activities in the project will either have concave, convex or linear productivity functions. If the productivity functions are concave, and all the activities are independent (i.e., there are no precedence relations among them), then they should be performed in parallel. If, in finding the minimum project duration T^*, the positive root of the equation, $T^* \sum_{i=1}^{n} f_i(x_i/T^*) = M$, does not violate the resource usage constraint $\sum_{i=1}^{n} f_i^{-1}(x_i/T^*) \leq N$, then one can allocate $r_i* = f^{-1}(x_i/T^*)$ for $i = 1, 2, \cdots, n$. If the usage constraint does not hold, then instead of T^*, we must find T^*_{∞}, which is the positive root of $\sum_{i=1}^{n} f_i(x_i/T^*) = N$. This will ensure that the usage constraints are not violated. On the other hand, if the productivity functions are all convex, then we perform the activities in series in an optimal solution.

Weglarz [52] also presents a procedure to allocate resources to dependent activities (i.e., activity networks with precedence relationships). Weglarz [52] asserts that it is possible to label the nodes of an activity-on-arc activity network such that the occurrence of node i is before the occurrence of node j if $i < j$ and that, the activity network should be labeled as such. Then, denote the indices of the activities, which may be performed between nodes k and $k+1$ as $I_k, k = 1, 2, \cdots, s-1$, where s is the number of nodes in the network. Additionally, denote x_{ij} the part of x_i performed in I_k. Also let $\Delta_k(\{x_{ik}\}_{i \in I_k}, M_k)$ be the duration of I_k. The idea here is that the parts of activities performed in any given I_k are independent, and so, we can use previous results. If the production function for all the activities are convex, we simply perform all of them in series. However, to find the optimal solution for the concave case, we can solve the following nonlinear program (note that Δ_k can be calculated as if all x_{ik} are independent):

$$\text{Minimize} \qquad T = \sum_{k=1}^{s-1} \Delta_k(\{x_{ik}\}_{i \in I_k}, M_k) \tag{2.68}$$

subject to:

$$\sum_{k=1}^{s-1} M_k \leq M \tag{2.69}$$

$$\sum_{k \in K_i} x_{ik} = x_i, \qquad i = 1, 2, \cdots, n \tag{2.70}$$

$$x_{ik} \geq 0, M_k \geq 0, \qquad i = 1, 2, \cdots, n; k \in K_i. \tag{2.71}$$

Then, the resource allocations, r_i^*, $i = 1, \cdots, n$, are determined from the optimal values of the decision variables.

Chapter 3

Model Formulation and Solution Approach

3.1 Accessibility and Sequences

As stated in Chapter 1, the problem that we address in this thesis is to determine an appropriate amount of resource to allocate to each activity of a project so as to minimize the project duration (makespan). The resource is usage-constrained (renewable), and a time-resource function is associated with each activity. Once an activity begins its operation, it cannot be pre-empted and resource levels cannot change.

In addition to the precedence digraph, there are two other important networks associated with projects: the accessibility digraph and the sequence digraph. Clearly, the precedence structure of the project is a factor that restricts how different amounts of resources may "flow" through the project. The precedence relationships specify which activities are possible successors of a given activity. If an activity j is a possible successor to activity i, then activity j is *accessible* from i. An accessibility digraph can be developed to explicitly identify which activities are accessible from other activities by adding disjunctive arcs. Consider the precedence digraph shown in Figure 3.1. Its corresponding accessibility digraph is depicted in Figure 3.2.

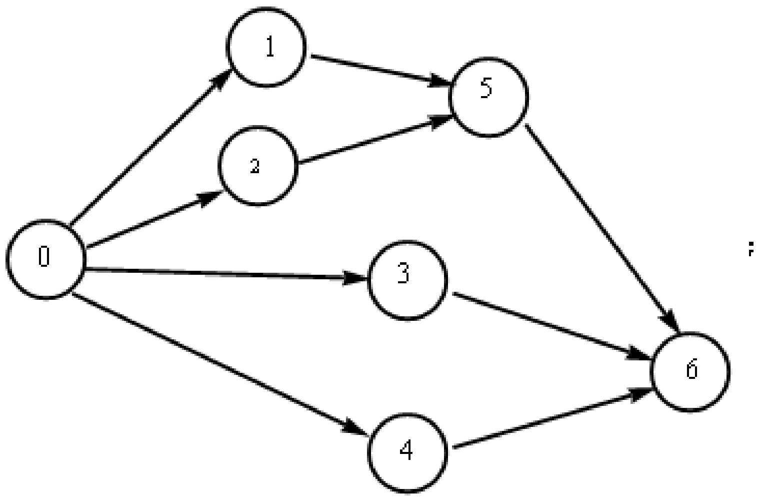


Figure 3.1: A precedence digraph

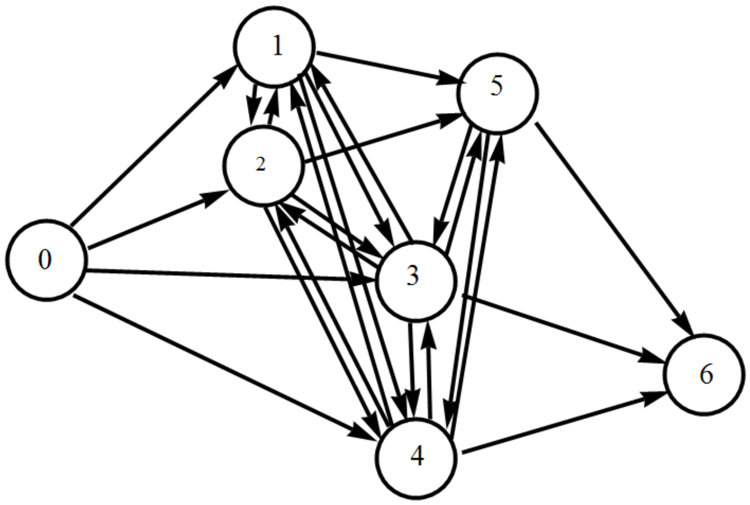


Figure 3.2: The accessibility digraph of the network in Figure 3.1

Since the accessibility digraph has too many arcs to be viewed easily, the accessibility matrix (an adjacency matrix corresponding to the accessibility digraph) is a better way to represent the accessibility relationships in the project. The accessibility matrix for Figure 3.2 is as follows.

$$Q = \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 & 1 & 0 \\ 3 & 1 & 1 & 0 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 0 & 1 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 & 1 \end{array} \tag{3.1}$$

If $Q_{ij} = 1$, then activity j is accessible from activity i. Note that there is no column associated with the 0^{th} activity since it will not be accessible from any activity. Likewise, there is no row associated with the last activity since no activities are accessible from the sink. It is important to note that simply because $Q_{ij} = 1$, does not imply that resources from i can flow to j. Activity j's predecessor must be completed first before this is possible.

A sequence digraph for a project, or simply a sequence of the project, shows a possible ordering of the activities. An arc (i, j) of a sequence diagraph represents a flow-carrying arc from activity i to activity j. For example, one possible sequence for the project shown in Figure 3.1 is depicted in Figure 3.3.

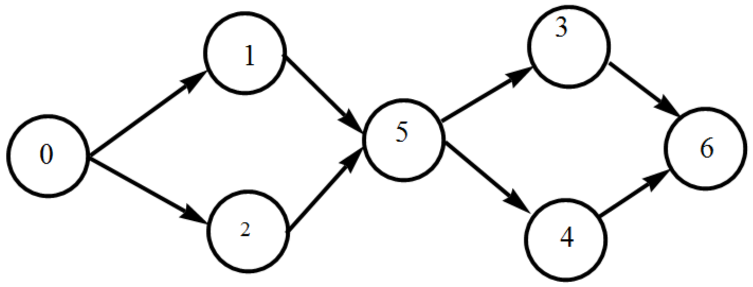


Figure 3.3: A sequence digraph

3.2 Overview

A flow-based network representation is used to model the problem in which the resource "flows" through the project network from one activity to another. The decision variables are (1) the selection of flow-carrying disjunctive arcs from the accessibility matrix, and (2) the determination of the amount of flow on each of the conjunctive and selected disjunctive arcs. These will give rise to each activity's completion, and hence, the project completion time. In order to satisfy the resource usage constraint, the amount of resource flow into the project is equal to the maximum resource level, and the flow-in should be equal to flow-out for each activity. Additionally, no two-way flows may exist among activities. The accessibility matrix will ensure the desired precedence relationships among the activities as well as the assignment of the resource to each activity. Additional resources may not be added to an activity once its operation has begun.

3.3 Mathematical Model

We use activity-on-node representation. Consider the following notation:

Parameters

$R(P)$	= Set of activities in the project P
F	= Set of potential resource flows among the activities in the project i.e., arcs of the accessibility digraph
$S(i)$	= Set of activities accessible from activity i, $i \in R(P)$
$\rho(i)$	= Set of activities for which i is accessible for, $i \in R(P)$
$0, n$	= Indices of first (source) and last (sink) activity in $R(P)$, respectively
W	= Maximum number of workers available
$d_i(w)$	= Duration of activity i if the number of workers assigned for its processing is w

Decision Variables

$$\begin{aligned} t_i &= \text{completion time of activity } i,\ \forall i \in R(P) \\ w_{ij} &= \text{amount of resource that flows from activity } i \text{ to activity } j,\ \forall (i,j) \in F \\ x_{ij} &= \begin{cases} 1, & \text{if flow occurs on } (i,j) \\ 0, & \text{otherwise} \end{cases} \quad , \quad (i,j) \in F \end{aligned}$$

Model PSPPF:

$$\begin{aligned} \text{minimize} \quad & t_n && && (3.2) \\ \text{subject to:} \quad & && && \\ & t_j - t_i \geq d(\sum_{k \in \rho(j)} w_{kj}) && \forall i \in \rho(j), \forall j \in R(P) \backslash \{0\} && (3.3) \\ & \sum_{j \in S(0)} w_{0j} = W && && (3.4) \\ & \sum_{i \in \rho(n)} w_{in} = W && && (3.5) \\ & \sum_{i \in \rho(j)} w_{ij} = \sum_{k \in S(i)} w_{jk} && \forall j \in R(P) \backslash \{0, n\} && (3.6) \\ & w_{ij} \leq W x_{ij} && \forall (i,j) \in F && (3.7) \\ & x_{ij} \leq (1 - x_{ji}) && \forall (i,j) \in F && (3.8) \\ & t_0 = 0 && && (3.9) \\ & w_{ij} \geq 0 && \forall (i,j) \in F && (3.10) \end{aligned}$$

The constraints sets (3.4) to (3.6) are the flow-based constraints that maintain conservation of flow through the network. The constraint set (3.3) ensures that the completion time of an activity j, $j \in R(P)\backslash\{0\}$, is greater than or equal to its processing time plus the completion times of all of its predecessors. The constraint set (3.7) ensures that if no flow is to occur between activities i, j, the w_{ij} does, in fact, equal 0. Constraint set (3.8) disallows two-way flows between activities. The constraint set (3.9) arbitrarily assigns the completion time of the source node to zero, while the constraint set (3.10) enforces non-negative flows on each pair of activities, $(i, j) \in A$.

The values of the variables x_{ij}, $(i, j) \in F$ determine the sequence in which to perform the project. In particular arcs $(i, j) \in F$ with $x_{ij} = 1$ are the flow carrying arcs from the disjunctive arcs that determine the sequence. If the sequence for a project is known before hand, i.e, values variables x_{ij}, $(i, j) \in F$ are known a priori, the model PSPPF reduces to only a flow determination problem (model PSPPF').

Model PSPPF':

$$\text{minimize} \quad t_n \tag{3.11}$$

subject to:

$$t_j - t_i \geq d(\sum_{k \in \rho(j)} w_{kj}), \qquad \forall i \in \rho(j), \forall j \in R(P)\backslash\{0\} \tag{3.12}$$

$$\sum_{j \in S(0)} w_{0j} = W \tag{3.13}$$

$$\sum_{i \in \rho(n)} w_{in} = W \tag{3.14}$$

$$\sum_{i \in \rho(j)} w_{ij} = \sum_{k \in S(i)} w_{jk}, \qquad \forall j \in R(P)\backslash\{0, n\} \tag{3.15}$$

$$t_0 = 0 \tag{3.16}$$

$$w_{ij} \geq 0, \qquad (i, j) \in SEQ \tag{3.17}$$

where SEQ is the set of flow-carrying arcs (for which $x_{ij} = 1$). In this case, $\rho(i)$ is now the set of activities that immediately precede i and $S(i)$ is the activities that immediately succeed i.

3.4 Solution Approach

Our approach for the solution of model PSPPF (Project Scheduling in the Presence of Production Functions) involves finding certain, potentially optimal sequences in which the project may be performed, and then, selecting the sequence whose optimal resource allocation gives the smallest makespan for the project. These sequences are found using a search tree. The differentiating feature of our approach is the use of some special properties of the independent activities (defined in section 5.1) of a project that reduces the size of the search tree, and consequently, enumeration of a fewer number of sequences. We discuss this in Chapters 4 and 5 where we study different forms of productivity functions and identify some special properties of independent activities. In Chapter 6, we present an algorithm for enumerating all possible sequences. The optimal resource allocation for each sequence is determined using a standard nonlinear programming algorithm (Augmented Lagrangian Penalty Function method).

Chapter 4

Discussion of Productivity Functions

4.1 Characteristics of an Activity Duration Function

The duration of an activity, generally, decreases as an additional amount of a resource is allocated to it. However, it is well-known that, after a certain point, an additional amount of the resource no longer remains as productive. In other words, each activity has a resource level at which the maximum marginal productivity rate is attained. An increment of resource beyond that level will produce a *diminishing return* of marginal productivity rates. This phenomena is called *the Law of Diminishing Returns.* For example, if the activity is to saw 2x4 boards in half and stack them, then the greatest marginal productivity rate might be attained by 2 workers: one cutting the boards, and the other stacking them. The inclusion of another worker without adding a saw and a saw-horse, will not impact the productivity rate significantly. On the other hand, one man working alone may take more than twice as long as with having a partner. A hypothetical marginal productivity function, a productivity function, and a time/unit function associated with sawing the 2x4's are shown in Figures 4.1, 4.2, and 4.3, respectively. Note that the time/unit function is inverse of

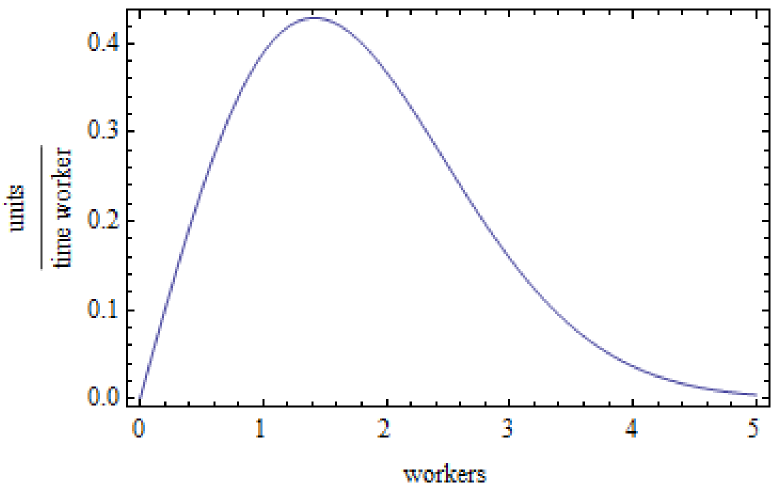


Figure 4.1: Marginal productivity function

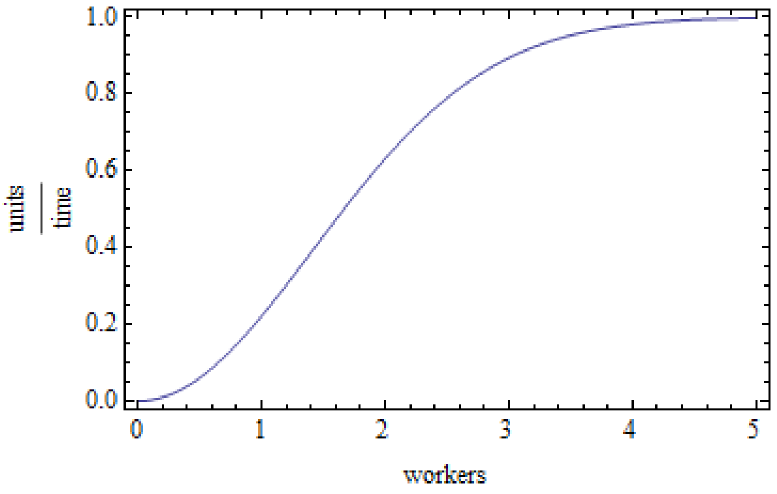


Figure 4.2: Productivity function

the productivity function, and that, it is convex. If an activity exhibits diminishing returns, then the productivity function may actually start decreasing as resources are added because the resources may interfere with each other. Consider, for instance, a group of 10 students trying to work together on a homework. They would all be chatting together and not getting any work done! Clearly, it is not advantageous to allocate resources in this manner.

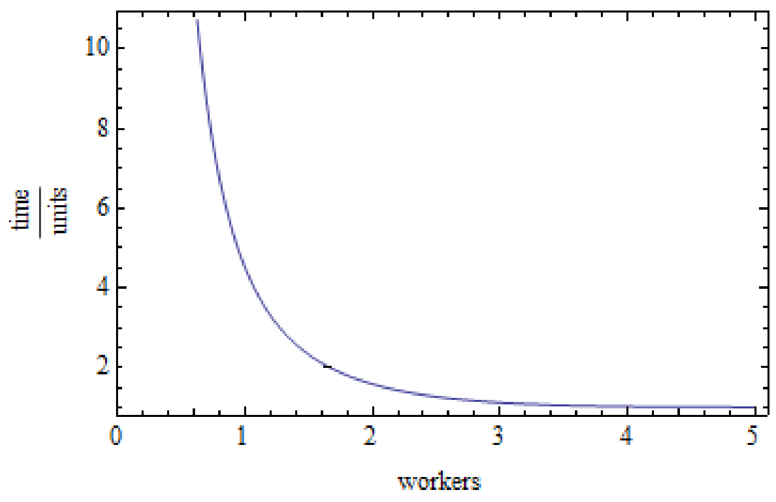


Figure 4.3: Time/unit function

The resource allocation that gives the minimum task time will constitute an upperbound on resource allocation for that task. Figure 4.4 depicts a variety of marginal productivity functions.

Figure 4.4(a) displays a flat marginal productivity function with a marginal productivity rate of 1 $unit/\ (time\ workers)$ which indicates that each additional worker will add the same amount of productivity. The function in Figure 4.4(b) exhibits diminishing returns starting from the initial stage of resource allocation. For the function in Figure 4.4(c), the point of diminishing returns is too large to be shown on the graph (within the range of the number workers shown in the figure). Consequently, for this range of workers, there is no point of diminishing returns. In Figure 4.4(d), the function peaks at 5 workers. Therefore, an additional worker beyond 5 will cause a drop in the average productivity of a worker. The corresponding productivity and time/unit functions for the marginal productivity functions in Figure 4.4 are shown in Figures 4.5 and 4.6, respectively.

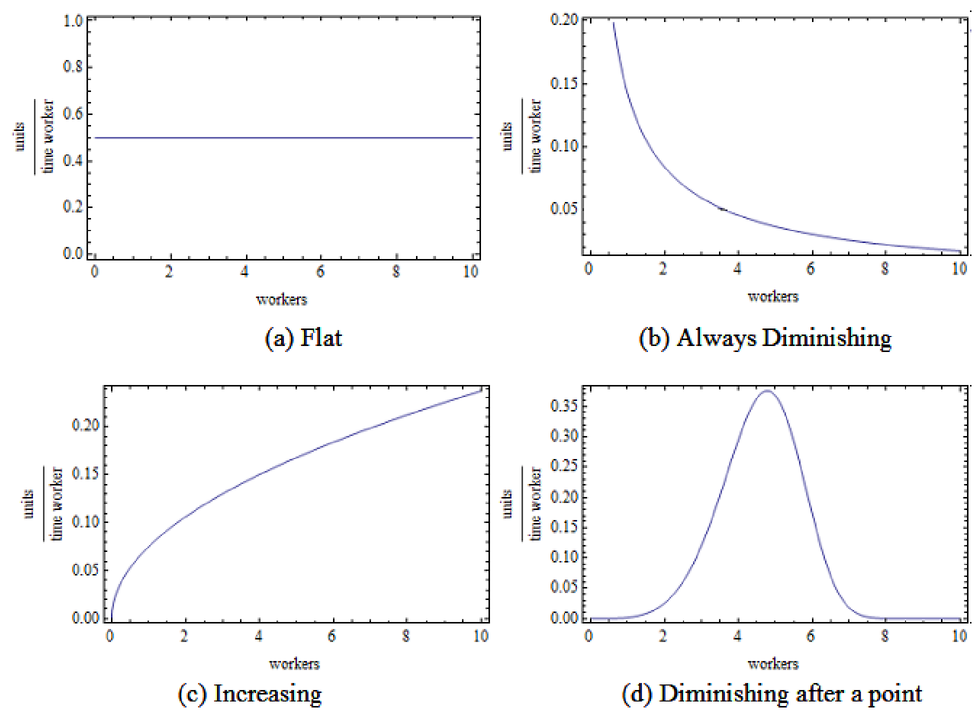
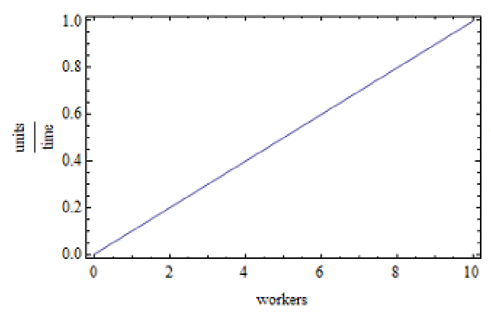
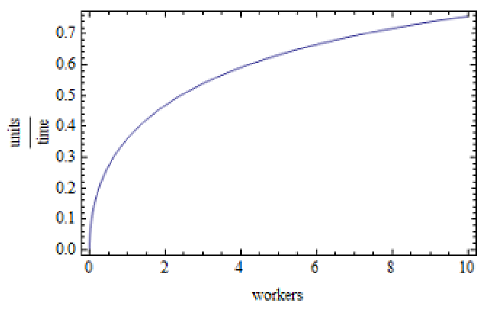


Figure 4.4: Marginal productivity functions

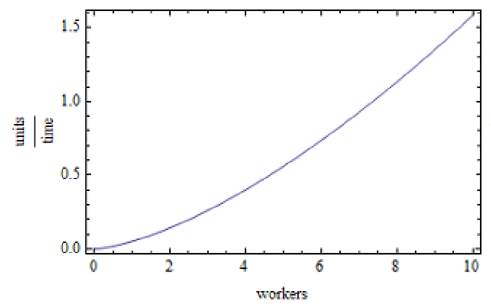
Notice that the unit of the functional value for the functions in Figure 4.6 is time/unit. The time required per unit is the ratio of the time required by a specified number of workers to that required by one worker to accomplish the task. In the sequel, we will call the time/unit function simply as the time function. Note that, with the above definition, the time function can be transformed into a true activity duration function by multiplying the value obtained from the function by the time required by one person to accomplish the task. Also, note that, the activity duration functions are convex and decreasing. The activity duration functions will be decreasing if the marginal productivity functions are greater than zero for all worker allocation levels. For some activities, their marginal productivity function may become negative after some particular resource allocation level, indicating a decreasing



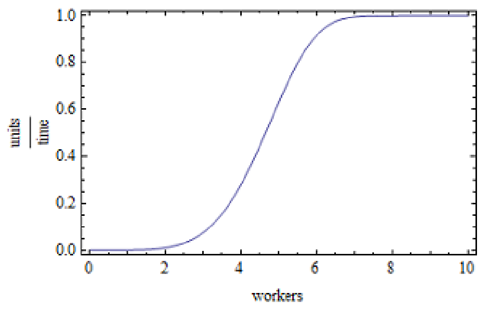
(a) Linear(flat)



(b) Concave (always Diminishing)



(c) Convex (increasing)



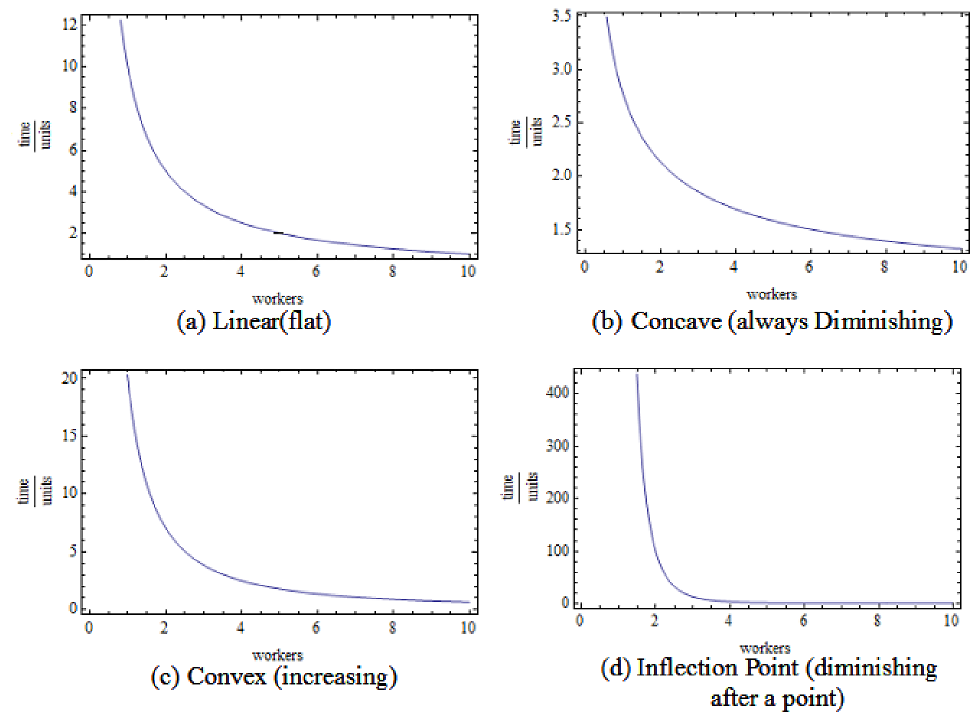
(d) Inflection Point (diminishing after a point)

Figure 4.5: Productivity functions

productivity function and an increasing activity duration function after this point. Clearly, it is not advantageous to consider resource allocations beyond these points. Therefore, resource allocation levels, beyond the point at which the marginal productivity function equals zero, be artificially forced to not increase the activity duration. In other words, if workers are added at levels where the marginal productivity function goes negative, then the extra workers sit idle for the duration of the activity.

4.2 Estimation of Production Functions

A productivity function may take on a variety of forms. As seen above, this function may be convex, concave, or it may consist of an inflection point. Therefore, the class of functions



(a) Linear(flat)

(b) Concave (always Diminishing)

(c) Convex (increasing)

(d) Inflection Point (diminishing after a point)

Figure 4.6: Time/unit functions

that we intend to use to estimate a productivity function must be able to take these three shapes, at least at non-extreme values of resource levels. Also, the class of functions must be able to capture the following three important features of the production rate function. Firstly, the productivity for one worker must be equal to 1 unit/time. Secondly the second derivative of the productivity function at the resource level corresponding to the inflection point, $w_{inflection}$, must equal 0. Thirdly, the desired productivity at the maximum number of workers must be captured. Since there are 3 characteristics of the production function that must be modeled, the class of functions used must have 3 parameters. A modified Weibull Cumulative Distribution Function is capable of capturing all the required features. This function is as follows:

$$\theta(w) = \lambda(1 - e^{-(w/\beta)^{\alpha}}), \tag{4.1}$$

where λ and β are scaling parameters, and α is a shape parameter.

Now, it is possible to estimate each of the parameters of $P(w)$ by numerically solving the following system of equations:

$$\begin{array}{ll} \theta(1) & = 1, \\ \theta''(w_{inflection}) & = 0, \\ \theta(w_{max}) & = MaxProd, \end{array} \tag{4.2}$$

where $w_{inflection}$ is the value of w corresponding to the inflection point, w_{max} is the maximum number of workers, and $MaxProd$ is the productivity at the maximum number of workers. Note that

$$P''(w) = \lambda(\frac{e^{-(w/\beta)^{\alpha}}\alpha(\alpha-1)(\frac{w}{\beta})^{\alpha-2}}{\beta^2} - \frac{-e^{-(w/\beta)^{\alpha}}\alpha^2(\frac{w}{\beta})^{2\alpha-2}}{\beta^2}) \tag{4.3}$$

.

Chapter 5

Independent Activities and Their Properties

5.1 Characterization of Independent Activities

Consider a set of activities, β, which are the possible successors to the already scheduled set of activities, γ. Let p be the set of activities in β that have common set of successors, S_p. Then, a pair of activities in p is independent if none of these or other activities in p have predecessors in β. In other words, such a pair is a bottleneck of sorts in that if they and their successors are not completed, no progress can be made after the completion of non-independent activities in β. If no precedence relationships exist in a project then all the activities are independent.

For example, consider the precedence digraph (activity-on-node representation) shown in Figure 5.1 for a project with activities $0, 1, 2, 3, 4, 5$, and 6. Note that, initially, the only sequenced activity is the source, 0 and therefore $\beta = (1, 2, 3, 4, 5, 6)$. Consider $p = (1, 2)$, which share a common successor, 5. Clearly, 1 and 2 are independent since they have no predecessors. Note that activities 3 and 5 share a common successor, 6, but they are not independent since activity 5, has predecessors 1 and 2, which are also in β. However, as a sequence of activities is developed, the relationships among them may change. For instance, if activities 1 and 2 have been sequenced after 0, then activities 3, 4, 5 and 6 become independent.

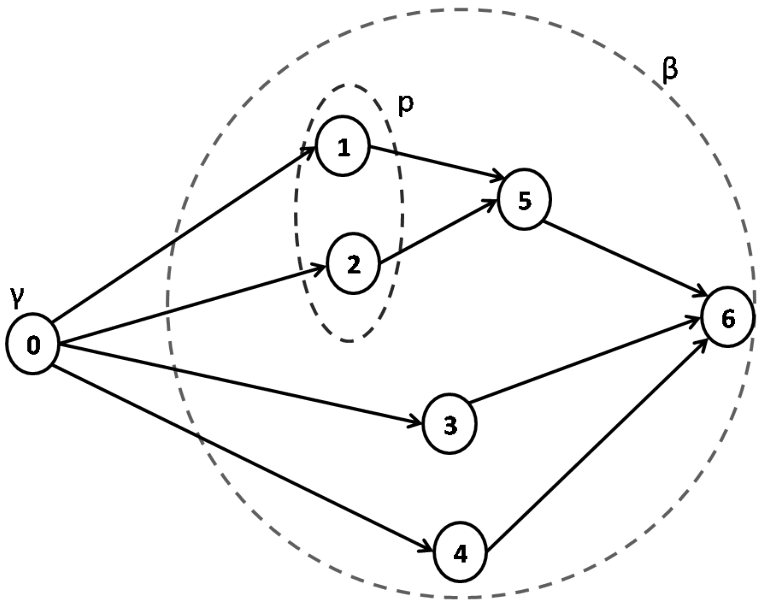


Figure 5.1: A precedence digraph (activity-on-node)

5.2 Scheduling Properties of Independent Activities

Consider a project consisting of n activities all of which are independent and pre-emptable, and a tuple $s = (s_1, s_2, \cdots, s_n)$, where s_i is the production rate at which activity i is processed. Let S be the set of all feasible points, s. This set of points may or may not be convex. For example, for a project consisting of two independent activities with concave productivity rates, the set S will form a convex set as shown in Figure 5.2. For a project consisting of two independent activities with concave productivity rates, the set S will form a non-convex set as shown in Figure 5.3. However, if both productivity functions are not concave, then their set of possible productivities will form a non-convex set like the one shown in Figure 5.4. Note that any point on the boundary of S may be achieved through allocating all available resources. All other points in S can be generated by the partial allocation of resources. For the non-convex set of points in Figures 5.3 and 5.4, let $\text{conv}(S)$ be the convex hull of S. It is important to note that any point on these convex hulls may be obtained by a convex combination of points on the boundary of S.

Recall that x_i is the amount of work to be done for activity i, d_i is the duration of activity i and t_n is the realization time of the project's sink node. The quantity x_i/d_i is a production rate of activity i. If all the activities in a project are independent and they are performed in parallel, it would be best to allocate the available resources such that all the activities start and finish at the same time, because otherwise, the resource can be reallocated to reduce the time at which these activities are completed. Therefore, the productivity for a given activity processed in parallel to all others will be given as x_i/t_n. The set of production rates $T = \{x_1/t, x_2/t, \cdots, x_n/t\}$ for different values of t produces a line, through the set S, and it comprising all the points for which each activity starts and finishes at the same time, t. This is shown in Figures 5.2, 5.3, and 5.4.

Note that the smallest feasible project duration for the given quantity of the available resource, when all the independent activities have concave productivity functions and are performed in parallel, is obtained at the intersection of T and the boundary of S as shown in Figure 5.2.

However, when S is not a convex set, then the boundary of S may have points both inside and on the boundary of conv(S). If the line T passes through a point that is both on the boundary of S and conv(S), then activities will be performed in parallel. However, if T passes through a point that is on the boundary of the conv(S), but outside the boundary of S, then in order to generate that point, a convex combination of points on the boundary of S is used. Note that a convex combination of points, in reality, corresponds to a reallocation of resources at certain times in the project. For example, for the first 40% of the project duration, one worker maybe working on activity 1 and three workers may be working on activity 2. Then, for the remaining 60% of the project duration, all 4 workers may be working on activity 2. This may be how s^* is achieved in Figure 5.4.

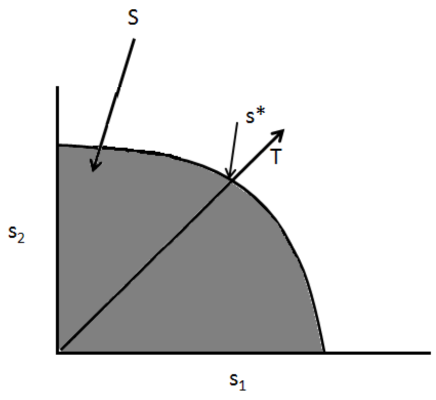


Figure 5.2: Both production functions are concave with pre-emptable activities

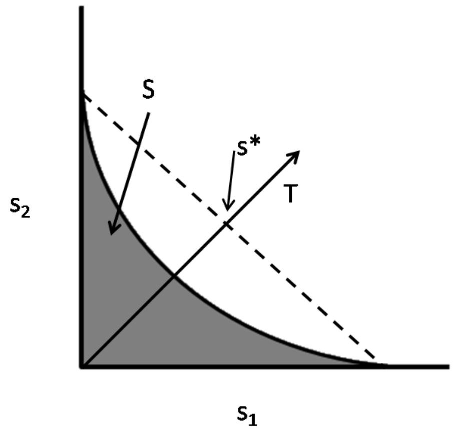


Figure 5.3: Both production functions are convex with pre-emptable activities

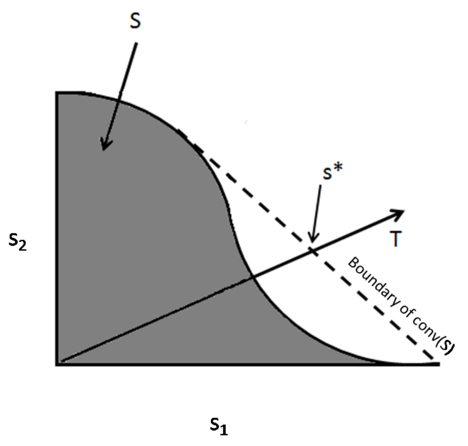


Figure 5.4: Mixed convex and concave production functions with pre-emptable activities

For the non-pre-emptable case, reallocation of resources between activities while they are being processed is forbidden. Consequently, achieving s^* in Figure 5.4 is impossible. Since the activities may only be processed in series or in parallel at invariable resource levels, the production rate points in S and the convex combinations of points on the boundary of S correspond to serial relationships between activities. For example, in a project with 3 independent events, if 1 and 2 are to be processed in series and 3 is to be processed in parallel to 1 and 2, the production rate point achieved would be some convex combination of the points $(s_1, 0, s_3)$ and $(0, s_2, s_3)$ where s_3 is constant. With these restrictions imposed by the non-pre-emptability constraint, the set of possible combinations of production rates, including those achieved using convex combinations of points, is reduced. For example, Figure 5.5 depicts the non-pre-emptable case of Figure 5.4. Area 1 represents the points found in the set S. Area 2 is the unshaded area beneath the thick dotted line and not in S, and can be found by using a convex combination of points corresponding to processing activities 1 and 2 in series. Area 3 is the unshaded area between the dotted lines and represents the points that may be achieved in the pre-emptable case, but not in the non-pre-emptable case.

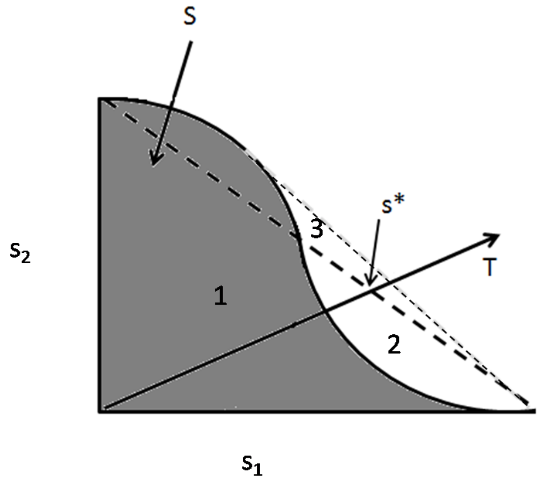


Figure 5.5: Mixed convex and concave production functions with non-pre-emptable activities

Note that for certain sets of independent activities, the set of possible production rate combinations that may be achieved are identical. This is the case in Figure 5.2 and in Figure 5.3.

We may now state the following remarks. Consider the following notation.

$$\begin{aligned}
\theta_i(w_i) =& \quad \text{Production rate of activity } i \text{ with } w_i \text{ workers}\\
I =& \; Set of independent activities\\
S(W) =& \quad \text{Set of feasible production rates for activities in } A \text{ for a given } W(=\\
& \quad \{(s_1, s_2, \cdots, s_n) : s_1 = \theta_1(w_1), s_2 = \theta_2(w_2), \cdots, s_n = \theta_n(w_n);\\
& \qquad \textstyle\sum_{\forall i} w_i = W\} \text{ where} |I| = n)\\
T =& \quad \text{“Equal time line”, line whose intersection with the boundary of}\\
& \quad conv(S) \text{ is the optimal production rate point, } s^*\\
=& \quad \{x_1/t, x_2/t, \cdots, x_n/t\}\\
x_i =& \quad \text{amount of work in activity } i
\end{aligned}$$

Remark 1. *If an activity requires no work, this activity can be eliminated from consideration.*

Remark 2. *Let w_{other} be the resources available to allocate to activities 1 and 2 with W as the maximum resources available for all the activities. Thus, $0 \leq w_{other} \leq W$. The* *boundary of the of the projection of S onto the s_1, s_2 plane is defined such that for a given s_1, $s_2 = \theta_2(w_{other} - \theta_1^{-1}(s_1))$, where $\theta^{-1}(s_1)$ is the number of workers required to process activity 1 at production rate s_1.*

Remark 3. *If the point $s_{parallel}$ is in S, then $s_{parallel}$ is achieved by processing all the activities in parallel. If $s_{parallel}$ does not fall in S, but is generated by a convex combination of points on the boundary of S, then a pair of activities whose production rates are not 0 in any of the points in the convex combination are processed in parallel.*

Remark 4. *If the point s_{series} is generated by using the convex combination $s_{series} = \lambda s^1 + (1-\lambda)s^2$, where $s^1 = (s_1, 0, s_3, \cdots, s_n)$ and $s^2 = (0, s_2, s_3, \cdots, s_n)$, then s_{series} is achieved by processing activities 1 and 2 in series (activity 1 will be processed for $\lambda \times 100\%$ of the duration and activity 2 will take the rest of the time). Note that additional activities may be processed in series with activities 1 and 2 and so additional terms maybe needed in the convex combination which generates s_{series}.*

Remark 5. *By Weglarz [52] we know that s^* is found at the intersection of T and conv(S).*

Weglarz [51] has shown that if all the activities in a network of independent activities have convex(concave) productivity functions, then all the activities will be performed in series(parallel) in an optimum solution. We extend this result and show that in a network of independent activities, if a pair of activities has convex(concave) productivity curves, then that pair of activities will be performed in series(parallel) irrespective of the other activities.

Proposition 1. *Let $h : \mathbb{R} \to \mathbb{R}$ be convex and non-decreasing, $g : \mathbb{R} \to \mathbb{R}$ be convex and non-increasing, and C be a constant scalar. Then, $f(x) = g(C - h^{-1}(x))$ is convex.*

Proof. A graph of h is shown in Figure 5.6. Note that by rotating it 90° clockwise, and then, flipping it vertically so that the axis labels are switched, we get the inverse of h, h^{-1}, which is concave. Therefore, $-h^{-1}$ is convex. Since C is constant, and hence, has no effect on the shape of the function, $f(x) = g(C - h^{-1}(x))$ must be convex since $f(x)$ is a convex function of a convex function (Bazaraa et al. [3]). □

Proposition 2. *If a set of independent activities, $I = \{1, 2, \cdots, n\}$, contains two activities, 1 and 2, whose productivity functions, $\theta_1(w_1)$ and $\theta_2(w_2)$, are convex, then there exists a makespan minimizing resource allocation in which activities 1 and 2 are performed in series.*

Proof. (1) By Remark 1, an optimum production rate combination, s^*, will have all production rates greater than zero. (2) By Remark 2, the boundary of the of the projection of S

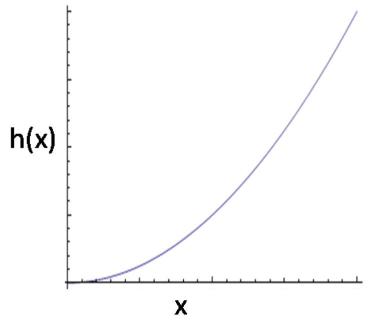


Figure 5.6: Non-decreasing, convex function, $h(x)$

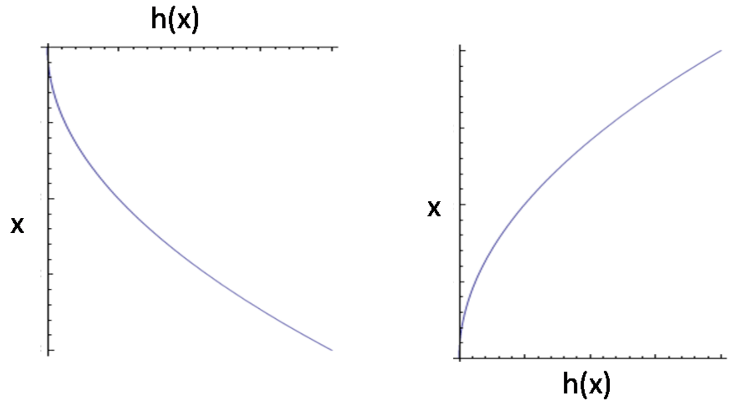


Figure 5.7: Transformation of the non-decreasing convex function, $h(x)$

onto the s_1, s_2 plane is defined such that for a given s_1, $s_2 = \theta_2(w_{other} - \theta_1^{-1}(s_1))$. If θ_1 and θ_2 are convex, then $\theta_2(w_{other} - \theta_1^{-1}(s_1))$ is convex by Proposition 1. This implies that the hypograph of $\theta_2(w_{other} - \theta_1^{-1}(s_1))$, H, is not convex and, moreover, no subset of H, which includes the boundary points of H, is convex. Consequently, the convex hull of H (conv(H)) must be created from a convex combination of its extreme points. Theses points will be $(\theta_1(w_{other}), 0)$ and $(0, \theta_2(w_{other}))$. (3) Consequently, since H is the projection of S onto the s_1, s_2 plane, the conv(S) must be created by using a convex combination of points including $(\theta_1(w_{other}), 0, s_3, ..., s_n)$ and $(0, \theta_2(w_{other}), s_3, ..., s_n)$. (4) By Remark 5, we know that s^* is found at the intersection of T and conv(S), and so s^* must be generated by the points given in (3). By Remark 4, a convex combination including these points corresponds to processing activities 1 and 2 in series. □

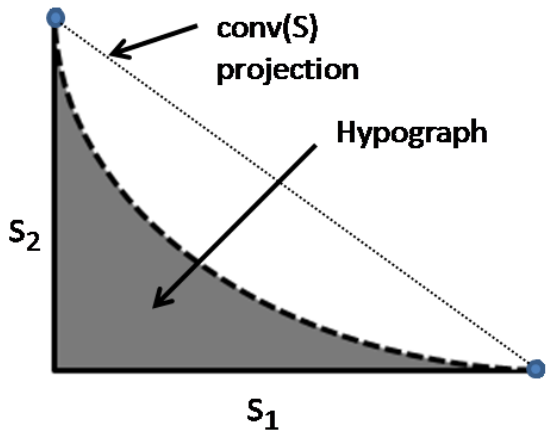


Figure 5.8: Hypograph of s_2

Likewise, if a pair of activities have concave productivity functions, they will be processed in parallel. We formally state and prove the result. But first, we show the following result.

Proposition 3. *Let $h : \mathbb{R} \to \mathbb{R}$ be concave and non-decreasing, $g : \mathbb{R} \to \mathbb{R}$ be concave and non-increasing, and C be a constant scalar. Then, $f(x) = g(C - h^{-1}(x))$ is concave.*

Proof. A graph of h is shown in Figure 5.9. Note that by rotating it 90° clockwise, and then, flipping it vertically so that the axis labels are switched we get the inverse of h, h^{-1}, which is convex. Therefore, $-h^{-1}$ is concave. Since C is constant, and hence, has no effect on the shape of the function, $f(x) = g(C - h^{-1}(x))$ must be concave since $f(x)$ is a concave function of a concave function (Bazaraa et al. [3]). □

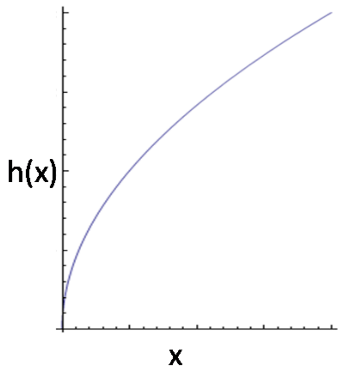


Figure 5.9: Non-decreasing, concave function, $h(x)$

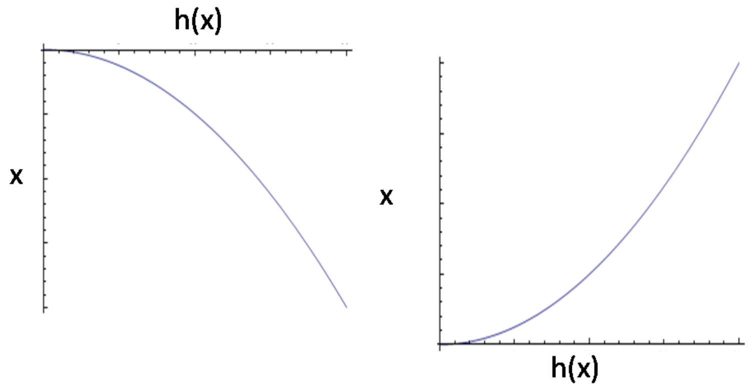


Figure 5.10: Transformation of the non-decreasing concave function, $h(x)$

Proposition 4. *If a set of independent activities, $I = \{1, 2, \cdots, n\}$, contains two activities, 1 and 2, whose productivity functions, $\theta_1(w_1)$ and $\theta_2(w_2)$, are concave, then there exists a makespan minimizing resource allocation in which activities 1 and 2 are performed in parallel.*

Proof. (1) By Remark 1, an optimum production rate combination, s^*, will have all production rates greater than zero. (2) By Remark 2, the boundary of the of the projection of S onto the s_1, s_2 plane is defined such that for a given s_1, $s_2 = \theta_2(w_{other} - \theta_1^{-1}(s_1))$. If θ_1 and θ_2 are concave, then $\theta_2(w_{other} - \theta_1^{-1}(s_1))$ is concave by Proposition 3. This implies that the hypograph of $\theta_2(w_{other} - \theta_1^{-1}(s_1))$, H, is convex. (3) Consequently, conv(H) is in the boundary of H, which is defined by the points $(\theta_1(w_1), \theta_2(w_2))$, where $w_1 + w_2 = w_{other}$. Since H is the projection of S onto the s_1, s_2 plane, the conv(S) includes points of the form $(\theta_1(w_1), \theta_2(w_2), s_3, ..., s_n)$. (4) By Remark 5, we know that s^* is found at the intersection of T and conv(S), and so s^* must be generated by the points given in (3). By Remark 3, these points correspond to processing activities 1 and 2 in parallel. $\square$

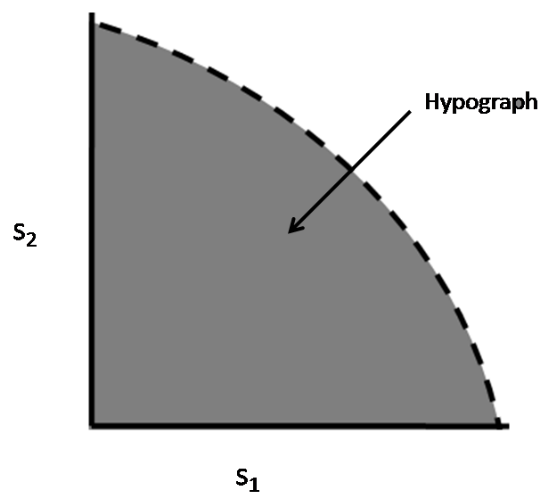


Figure 5.11: Hypograph of s_2

5.3 Exploitation of Scheduling Properties of Independent Activities

For some partial sequence of the activities, γ, of a project, consider the set of activities, β, that have not been sequenced. Based on the above results, the optimal sequence among certain pairs of activities in β may be known a priori. Consequently, this information limits how activities may be sequenced after γ. Consider the precedence digraph shown in Figure 5.12. The activities which are circled (activities 1 and 2) are independent activities. Based on the results presented in the previous section, suppose activities 1 and 2 will be performed in series in the optimal sequence. This may be accomplished by allowing only activity 1 or activity 2 to be sequenced after activity 0, and then, forcing a serial relationship between them in which the activities are performed back-to-back. An example sequence illustrating this is shown in Figure 5.13.

It may be noted that in order for activities 1 and 2 to have a serial relationship, they do

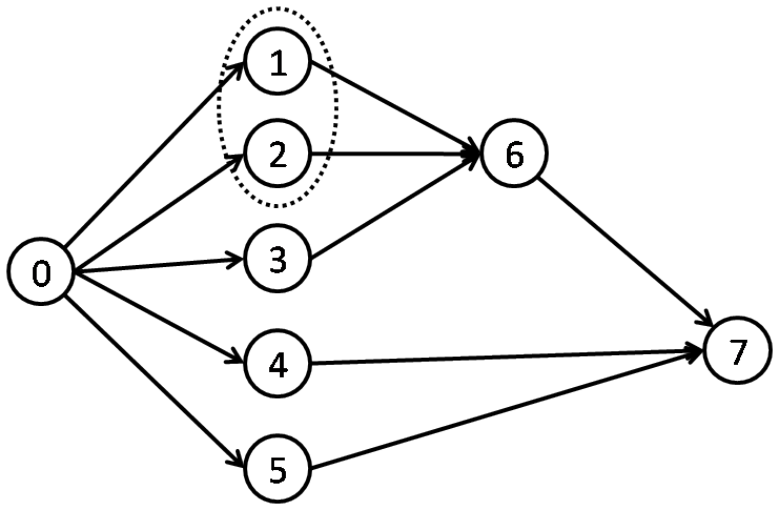


Figure 5.12: A project digraph with different forms of productivity functions

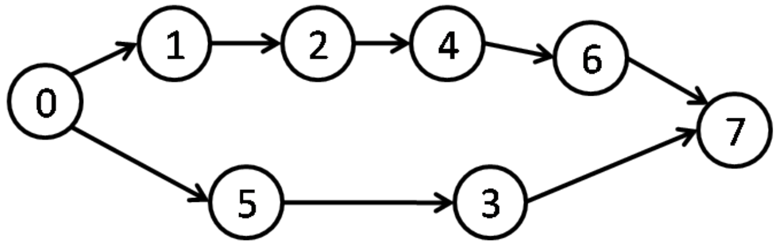


Figure 5.13: A possible sequence in which activities 1 and 2 are processed in series

not need to be processed consecutively. If activities 1 and 2 are processed in series, but not consecutively, then they have serial relationships with the activities processed between them. Since the duration of a project is not affected by the order of activities with serial relationships, then there exists an equivalent sequence in which activities 1 and 2 are processed non-consecutively to that in which they are processed consecutively. Note that the sequence depicted in Figure 5.13 is equivalent to the sequence depicted in 5.14.

Consider again the precedence digraph shown in Figure 5.12, however, activities 1 and 2 have concave productivity functions. In this case, if we process activities 1 and 2 in parallel immediately after activity 0, we get a sequence that is shown in Figure 5.15.

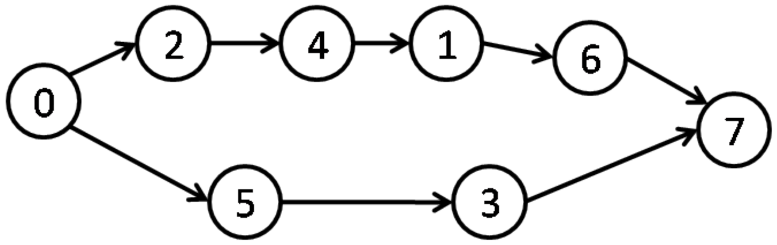


Figure 5.14: A sequence equivalent to Figure 5.13 where activities 1 and 2 are processed in series but not consecutively

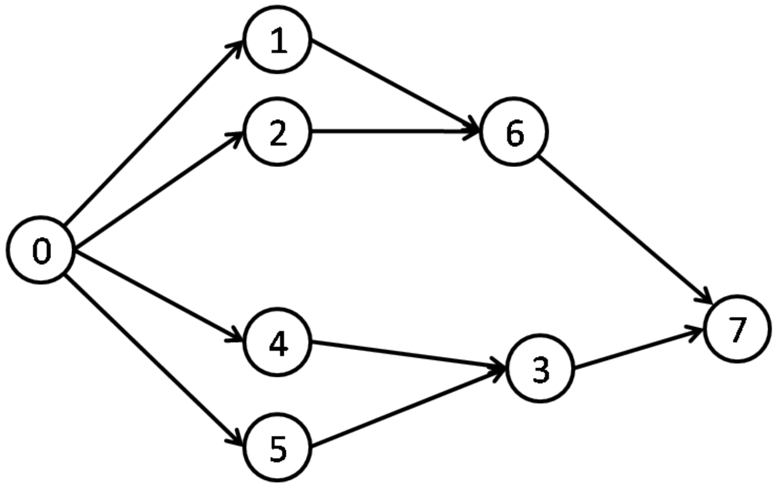


Figure 5.15: Possible sequence in which activities 1 and 2 are processed in parallel

It may be noted that in order for activities 1 and 2 to have a parallel relationship, they do not both need to be processed immediately after activity 0. Such a sequence only differs from the sequence in which activities 1 and 2 are processed immediately after 0 by the order in which activities 1 and 2 are processed with respect to the activities they have serial relationships with. Since the duration of a project is not affected by their relative order with activities having serial relationships, the two types of sequences are equivalent. Note that the sequence depicted in Figure 5.15 is equivalent to the sequence depicted in 5.16.

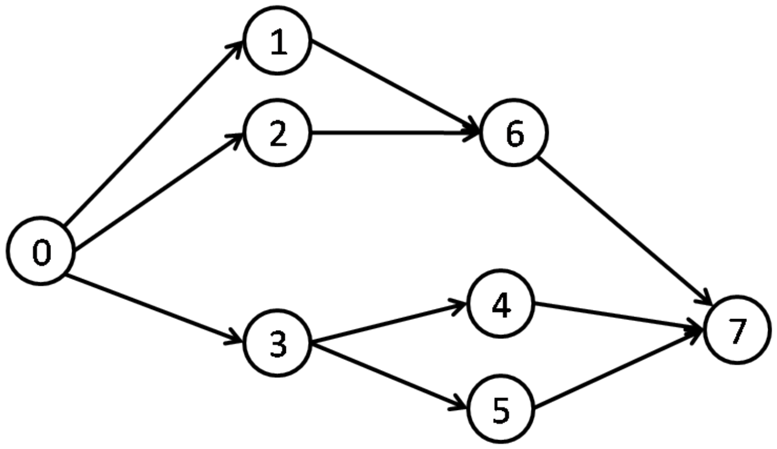


Figure 5.16: A sequence that is equivalent to that in Figure 5.15

Chapter 6

Solution Algorithm

In general, there are many possible sequences in which the activities of a project may be executed. The following method can be used to find these possible sequences.

Enumeration of Sequences

In order to determine an optimal sequence in which to process the activities of a project, it is essential to enumerate explicitly or implicitly all possible sequences. We present a tree search procedure to do that. Our aim is to exploit the properties developed in Chapter 5, and other properties, so that the size of the search tree is as small as possible.

Search Tree

This search tree will generate all possible sequences if the procedure is fully executed and to completion. A node of this tree consists of a *partial sequence*, that is, an updated accessibility matrix, $A_{current}$, in which a set of activities, γ, has been sequenced, and the set of activities, ξ, from which flow assignments are to be made. For instance, a node is shown below in which the activity that is already scheduled (but not yet completed) is 0, and the arrow points to the activity in ξ. The node depicted in Figure 6.1 is associated with an project consisting of three independent activities.

$$\begin{array}{cccccc} & - & 1 & 2 & 3 & 4 \\ \longrightarrow & 0 & 1 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 1 & 1 \\ & 2 & 1 & 0 & 1 & 1 \\ & 3 & 1 & 1 & 0 & 1 \end{array}$$

$$\psi = \{\}$$

Figure 6.1: An example node

If this node does not represent a full sequence (for a full sequence, the only possible next activity in the sequence is the last activity), then one or more *branches*, as the case may be, are created from the node. These branches create new nodes based on the next possible activities that are accessible from the current node. These next possible activities are all the activities which may be performed now that their predecessors have been completed. For example, from the current node shown in Figure 6.2, the possible branches are: $0 \rightarrow 1$, $0 \rightarrow 2$, $0 \rightarrow 3$, $0 \rightarrow \{1, 2\}$, $0 \rightarrow \{1, 3\}$, $0 \rightarrow \{2, 3\}$, $0 \rightarrow \{1,2, 3\}$ as depicted in Figure 6.2.

However, if the set of the next possible activities contains pairs of independent activities whose productivity functions are all either convex or concave, we can reduce the number of possible branches. In case that activities 1 and 2 are independent with convex productivity functions, a flow must not go to both 1 and 2 and, consequently, branches $0 \rightarrow \{1, 2\}$ and

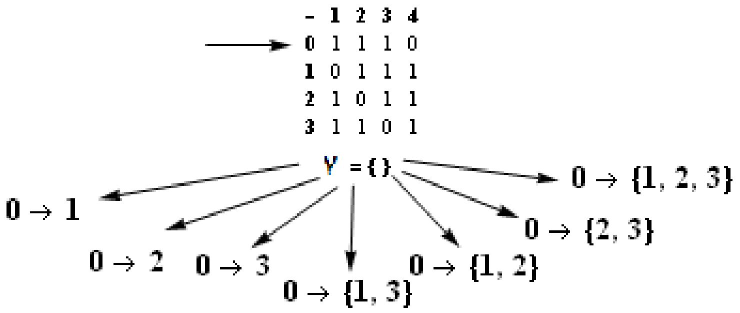


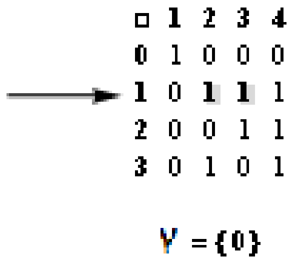
Figure 6.2: Possible branching from the initial partial sequence $\{0\}$

$0 \to \{1{,}2, 3\}$ are fathomed.

It is important to pass to the next node a list of the old independent activities with a serial relationship to enable enforcement of serial relationships between the independent activities having convex productivity functions.

Likewise, if the independent activities, 1 and 2, have concave productivity functions, then we eliminate any branches that do not contain flows to both activities 1 and 2 and, consequently, all branches except $0 \to \{1, 2\}$ and $0 \to \{1{,}2, 3\}$ are fathomed. No information about these activities needs to be passed to the next node in this case.

Suppose we pick the $0 \to 1$ branch. We must modify $A_{current}$ to reflect two facts. First, activity 1 is no longer accessible from any other activity than activity 0. Second, activity 0 may not access any other activities other than activity 1. In general, we must modify $A_{current}$ to reflect that the newly assigned activities are only accessible from the activities they follow, and that, the newly assigned activities are the only activities accessible to the activities they follow. Figure 6.3 shows how $A_{current}$ is modified.



□	1	2	3	4
0	1	0	0	0
1	0	1	1	1
2	0	0	1	1
3	0	1	0	1

$\xi = \{0\}$

Figure 6.3: Node modification for partial sequence $\{0, 1\}$

Each branch of each node is traversed in this manner, and the branching process is repeated for each new node. If the only branching possible from a node is the last activity, then the node is a full sequence and is recorded as such.

Note that in the tree search procedure, many partial sequences may be generated that are identical. For instance, a partial sequence $3 \to 1$ is the same as $1 \to 3$. Therefore, if partial sequence $1 \to 3$ exists in a previously found sequence, then any sequence containing $3 \to 1$ is equivalent to that sequence, and hence, should be avoided.

Also, if there is more than one activity in ξ, not all the activities in ξ must have an activity sequenced after them for each branch. For a branch, the activities in ξ which do not have resource flows out of them will be included in the ξ of the next node associated with that branch. For example, as depicted in Figure 6.4, if we choose the branching $0 \to \{1, 3\}$ at the initial node, $\{0\}$, then our new ξ would contain activities $\{1, 3\}$. It is feasible to sequence 2 after activity 1, while no activities are sequenced after 3 for this node. In this newest node, not only is activity 2 in ξ, but, activity 3 continues to be in ξ.

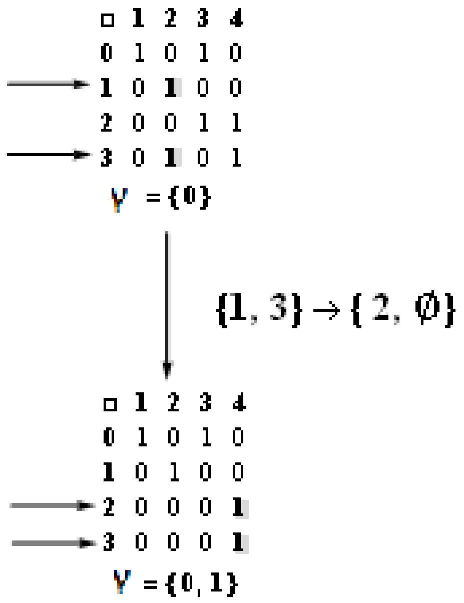


Figure 6.4: Nodes {0}, {1, 3} and {0}, {1, 3}, {2}

All the features of the search tree are illustrated in Figures 6.5 and 6.6 using a simple example. The matrix at each node is the modified accessibility matrix. The arrows pointing to particular rows of the accessibility matrix at each node represent ξ. The bold 1's in the accessibility matrix at each node indicate the accessible flows. The label, in the middle to end of the arc, shows the resource flow. The tree should be read from left to right following each branch to its leaf.

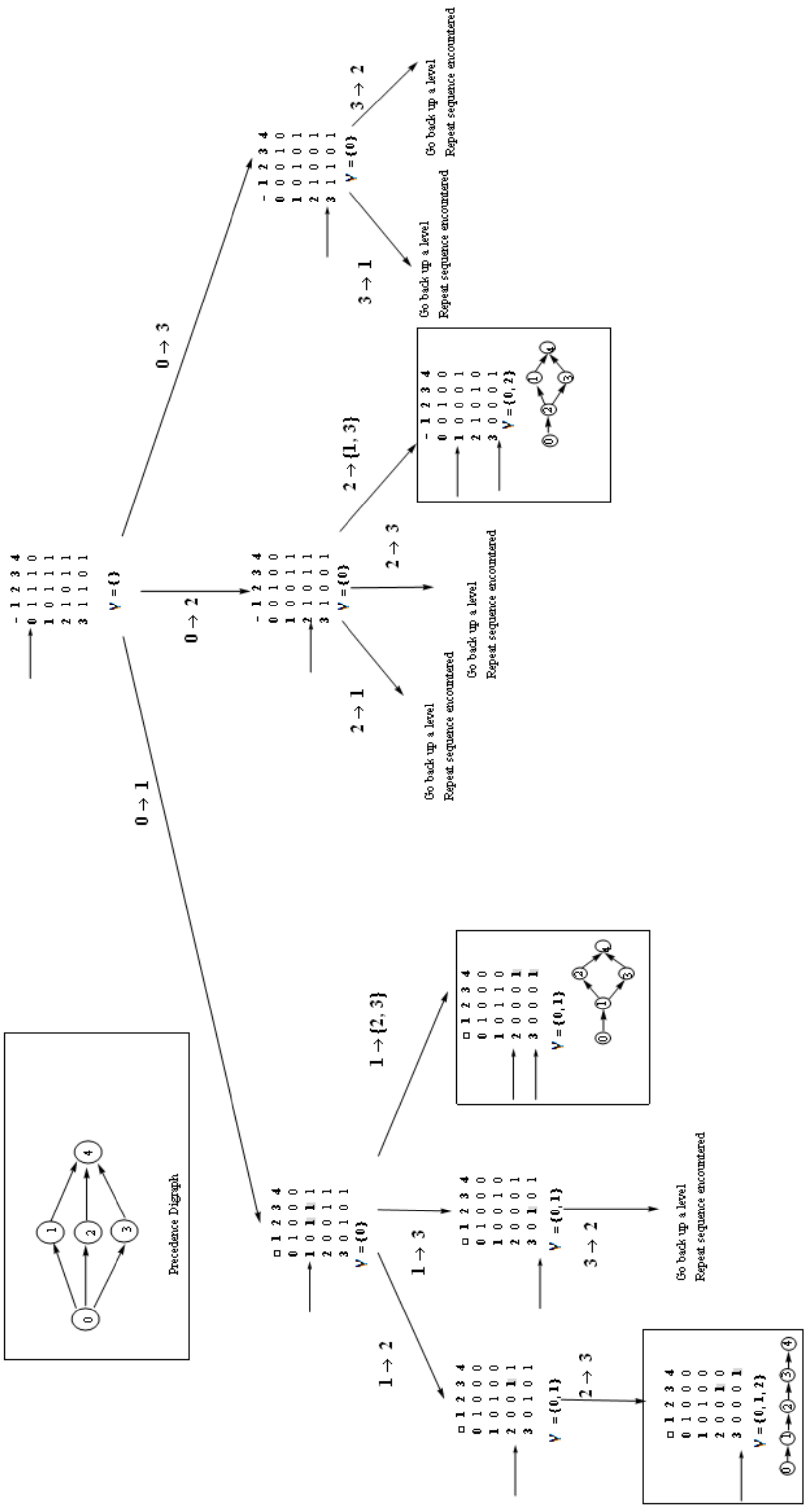


Figure 6.5: Search tree

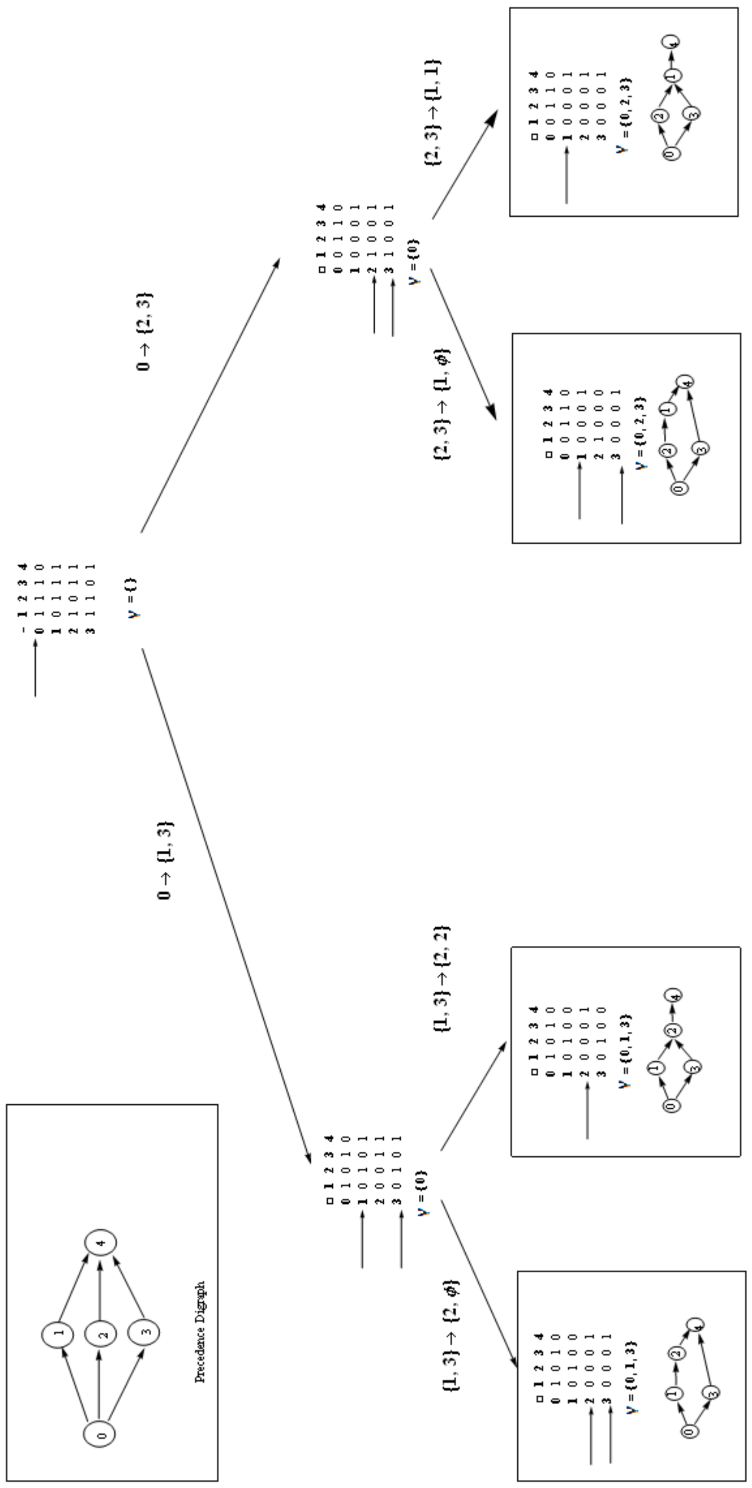

Precedence Digraph
$0 \rightarrow \{1, 3\}$
$0 \rightarrow \{2, 3\}$
$\{1, 3\} \rightarrow \{2, \phi\}$
$\{1, 3\} \rightarrow \{2, 2\}$
$\{2, 3\} \rightarrow \{1, \phi\}$
$\{2, 3\} \rightarrow \{1, 1\}$
$V = \{\}$
$V = \{0\}$
$V = \{0, 1, 3\}$
$V = \{0, 2, 3\}$

Figure 6.6: Search tree

6.1 Determination of Minimum Makespan of a Sequence

Once a feasible sequence is determined, it is necessary to determine the allocation of available resources to its activities so that the makespan is minimized. We do this by solving model PSPPF'. Note that this is a nonlinear program since the constraint set (3.3) is nonlinear, and we use an available software for its solution. The mathematical model for the reduced PSPPF is presented in Section 3.3. We use an Augmented Lagrangian Penalty Function method developed by M. Asghar Bhatti using Mathematica for the solution of model PSPPF'.

An Illustrative Example

Consider the precedence digraph depicted in Figure 6.7 and a possible sequence for this precedence shown in Figure 6.8.

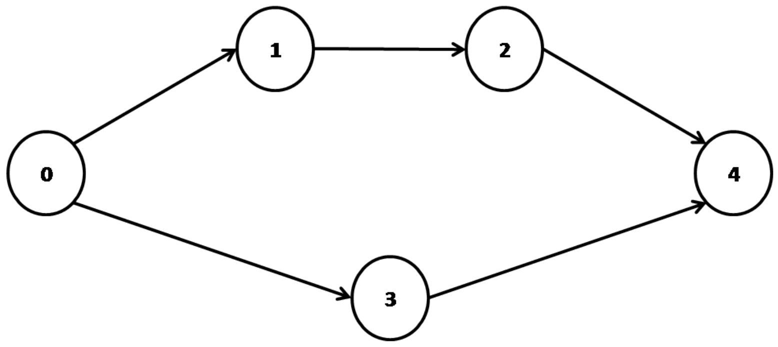


Figure 6.7: A precedence digraph

Table 6.1 gives the duration functions for each activity of the project.

Model PSPPF' is developed for the sequence in consideration. For instance, for one of the

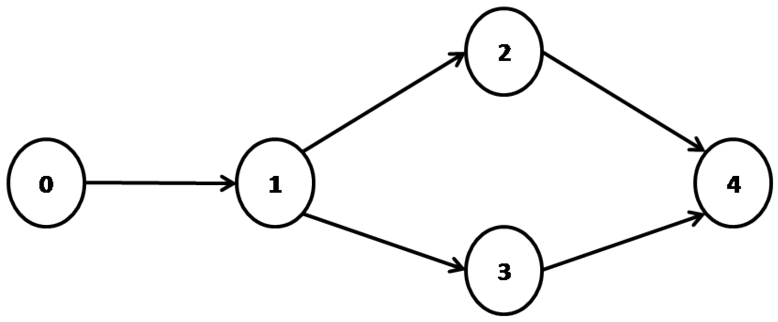


Figure 6.8: A sequence of the network in Figure 6.7

Table 6.1: Table of Duration Functions

Activity	Function
1	0
2	$67.2229\left(1-e^{-0.0149876w_1^{1.47539}}\right)$
3	$13.0783\left(1-e^{-0.0795438w_2^{1.80227}}\right)$
4	$3208.92\left(1-e^{-0.000791346w_3^{0.858637}}\right)$
5	0

sequences, we have the following constraints:

$$\begin{aligned}
&w_{0,1} && = 5 \\
&-w_{0,1}+w_{1,2}+w_{2,3} && = 0 \\
&-w_{1,2}+w_{2,4} && = 0 \\
&-w_{1,3}+w_{3,4} && = 0 \\
&w_{2,4}+w_{3,4} && = 5 \\
&-t_0+t_3 && \geq \frac{0.0148759}{1-e^{-0.0149876w_{1,2}^{1.47539}}} \\
&-t_1+t_2 && \geq \frac{0.0764625}{1-e^{-0.0795438w_{2,3}^{1.80227}}} \\
&-t_1+t_3 && \geq \frac{0.000311631}{1-e^{-0.000791346w_{2,4}^{0.858637}}} \\
&-t_2+t_4 && \geq 0 \\
&-t_3+t_4 && \geq 0 \\
&t_0 && = 0 \\
&w_{i,j} && \geq 0 \qquad \forall (i,j) \in SEQ
\end{aligned} \tag{6.1}$$

For this project, there are 4 possible sequences. After solving model PSPPF' for all four sequences, the objective values are as follows: (1):0.299937, (2):0.29931,(3):0.343879 ,(4):0.344739. The sequence corresponding to constraints (6.1) gives the least makespan value, and, is therefore, the optimal sequence. The flow of resource for this sequence is as follows: $\{w_{0,1} = 4.99999, w_{1,2} = 2.76479, w_{1,3} = 2.2352, w_{2,4} = 2.7648$, and $w_{3,4} = 2.2352\}$. The optimal sequence and the flows are depicted in Figure 6.9.

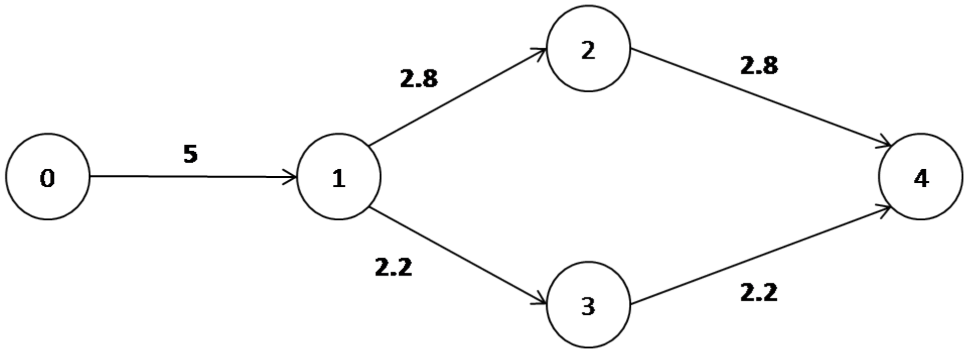


Figure 6.9: Solution to the problem

6.2 Solution Algorithm

We can now summarize the proposed method for solving the problem defined and modeled in Chapter 3. It consists of the following main steps:

1. **Enumerate all possible activity sequences.** The tree search algorithm for finding all possible activity sequences was described in section 6. This uses as input the reduced project network developed above.

2. **Find optimal resource allocations.** We use the reduced model PSPPF to determine optimal allocation of resources to activities for each sequence developed in step 1. The model is soved using a nonlinear program.

3. **Use the solution with the smallest project duration.** There may be one or

more sequences that give the best solution, and several near-optimal solutions. Based on expert knowledge of the underlying process, the "best" sequence is selected.

6.3 Some Illustrative Examples

Example 1. Consider a project with the precedence digraph as shown in Figure 6.10 and productivity functions as shown in Table 6.2.

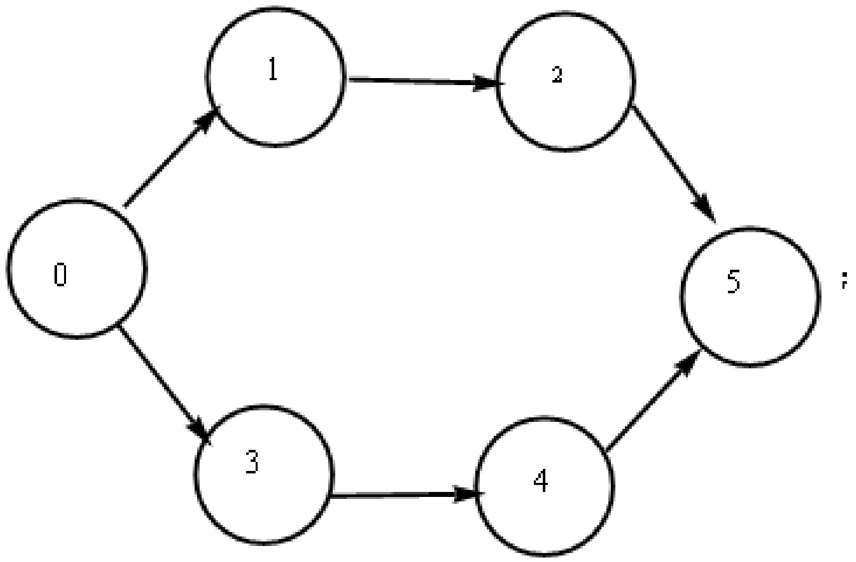


Figure 6.10: Precedence digraph for Example 1

Table 6.2: Table of productivity functions for the project in Figure 6.10

Activity	Function
0	0
1	$13\left(1-e^{-0.08w_1^{1.8}}\right)$
2	$67\left(1-e^{-0.015w_1^{1.5}}\right)$
3	$3209\left(1-e^{-0.0008w_3^{0.89}}\right)$
4	$68\left(1-e^{-0.015w_1^{1.5}}\right)$
6	0

After implementing the sequence enumeration algorithm without using the results developed in Chapter 5, we find 16 different sequences. In this implementation, some partial sequences that result in identical solutions are not deleted from these sequences.

After solving for the minimum makespan for each sequence, it is found that the optimal solution is as follows: $w_{0,1} = 2.7504$, $w_{0,3} = 2.2496$, $w_{1,4} = 2.7504$, $w_{2,5} = 5$, $w_{3,4} = 2.2496$, $w_{4,2} = 5$, $t_0 = 0$,$t_1 = 0.196609$, $t_2 = 0.395302$, $t_3 = 0.196557$, $t_4 = 0.295179$, $t_5 = 0.395425$ with a project duration of 0.395425. Figure 6.11 presents the optimal solution obtained.

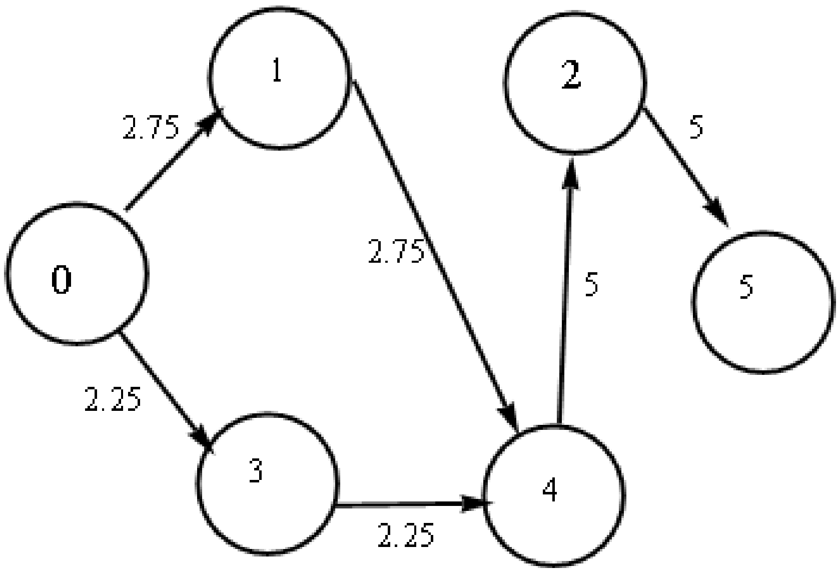


Figure 6.11: Project solution for Example 1

Note that activities 2 and 4 are processed in series. However, using the search tree exploiting properties developed in Chapter 5, it could be determined a priori that activities 1 and 2 are independent and they have convex production functions, and therefore, should be processed in series in an optimal solution. If this were applied, the number of sequences would be 8, which amounts to a reduction of 50%. However, as expected, the optimal solution obtained for this digraph would be the same as before.

Example 2. Consider a project with 4 independent activities and their productivity functions as shown in Table 6.3.

By implementing the sequence enumeration algorithm with out using the results developed in Chapter 5, we find 208 different sequences. In this implementation, some partial sequences that result in identical solutions are not deleted from these sequences.

After solving for the minimum makespan for each sequence, it is found that the optimal solution is as follows: $w_{0,1} = 4$, $w_{0,3} = 1$, $w_{1,4} = 4$, $w_{2,5} = 4$, $w_{3,5} = 1$, $w_{4,2} = 4$, $t_0 = 0$,

Table 6.3: Table of productivity functions for the project with 6 independent activities

Activity	Function
0	0
1	$68\left(1-e^{-0.015w_1^{1.5}}\right)$
2	$67\left(1-e^{-0.015w_1^{1.5}}\right)$
3	$3209\left(1-e^{-0.0008w_3^{0.89}}\right)$
4	$13\left(1-e^{-0.08w_1^{1.8}}\right)$
6	0

$t_1 = 0.134121$, $t_2 = 0.393779$, $t_3 = 0.393779$, $t_4 = 0.257631$, $t_5 = 0.393953$ with a project duration of 0.393953. Figure 6.12 presents the optimal solution obtained.

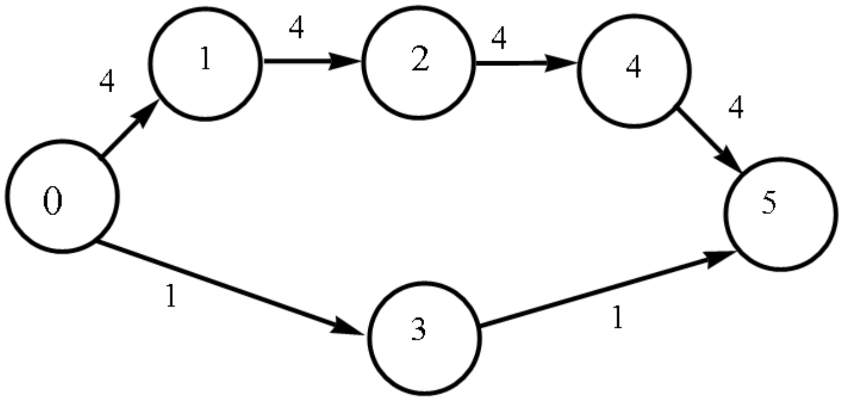


Figure 6.12: Project solution for Example 2

Note that activities 2 and 4 are processed in series. However, using the search tree exploiting properties developed in Chapter 5, it could be determined a priori that activities 1 and 2 are independent and they have convex production functions, and therefore, should be processed in series in an optimal solution. If this were applied the number of sequences is 128, which amounts to a reduction of 48.5%. However, as expected, the optimal solution obtained for this digraph is the same as before.

Chapter 7

Results and Conclusions

7.1 Testing Methodology

The solution methodology is developed using Mathematica and the test runs were made on a computer (Dell Precision 690) with 3 GHz Pentium Xeon processors and 3 GB of RAM running Windows XP. All possible sequences are generated for each problem; however, the resource allocations to each of the sequences is not found. Since finding the optimal resource allocation requires the solution of a nonlinear linear program, and solving each nonlinear program requires about the same amount of time for a given number of activities, the overall computational time can be easily estimated from the number of sequences developed by the search tree.

7.2 Rationale For Choice of Test Cases

There are three important characteristics of a project that determine the computational effort required to determine the optimal solution. The first characteristic is the number of activities in the project or the project size. The second is the project structure or how much "parallelism" the precedence structure exhibits. The third characteristic is the distribution of the various types of productivity functions for the activities of the project.

The number of activities chosen for the test cases range between 3 and 8. Except for very special project structures, the number of possible sequences becomes large enough to make it difficult to calculate all of them in a reasonable amount of time. This maybe due to the fact that the algorithm is developed in Mathematica, which is an interpretive programming language, rather than using a language like C, which would make it more efficient.

The "parallelism" of a project is a soft concept, but the idea is that the parallelism of a project increases with the number of activities which may be performed in parallel. The two extreme cases of parallelism are a project with a serial precedence digraph and a project with no precedence relationship whatsoever. Projects with precedence digraphs with low parallelism are easy to handle, even for projects with large numbers of activities. Therefore, test cases with "medium" and "high" amounts of parallelism are chosen.

For each combination of project size and project structure, a number of different activity duration function distributions are possible. The distributions for the test cases are chosen so that they impact the number of sequences found. This is important since there are some activity duration functions that will not impact the number of sequences found by the search tree. In general, these productivity functions can be thought of as being mapped to activities located somewhere between the front and the back of the project. If an activity is located at the front of the project, its immediate precedence requirement is that project is started.

If an activity is located at the back of the project, it is an immediate predecessor to the sink of the project

7.3 Effects of Parallelism

To illustrate the effect of the amount of parallelism in a project's precedence digraph, consider the projects in Table 7.1. All of these project have 6 activities, and no special productivity function distributions. Their respective number of sequences generated, and time required to enumerate the search tree are also shown. Please refer to Appendix A to find the precedence digraphs corresponding to the projects listed in Table7.1. A plot of the number of sequences generated and computational time required to search the tree are presented in Figure 7.1.

We can make the following observations about the precedence digraphs of 6A, 6B2, and 6B and 6D, respectively. They each have a different number of "rungs". Activities with predecessors and successors to each other, excluding the source and sink, are a part of the same rung. For instance, project 6B has 4 rungs: $\{1, 2, 6\}$, $\{3\}$, $\{4\}$, and $\{5\}$. Since all of these rungs have, roughly, the same amount of parallelism within themselves, they each contribute fairly evenly to the parallelism of overall project. So, the parallelism of a project

Table 7.1: Projects with six activities and varying parallelism

Project	Num of Activities	Parallelism	Prod Func Dist	Num of Sequences	Search Tree Time
6A	6	Low	None	188	0.375
6B2	6	Medium	None	1685	3.171
6B	6	High	None	24095	45.687
6D	6	Ind	None	159524	1175.81

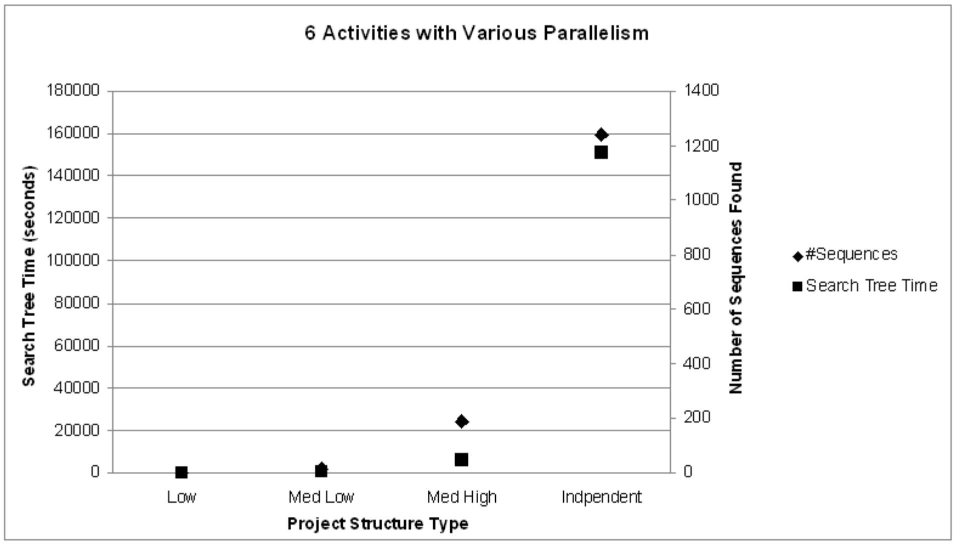


Figure 7.1: Computational complexity for projects with six activities and various levels of parallelism

depends, in a large part, on the number of rungs between the source and sink of a project's precedence digraph and, less so, on the number of rungs within a rung, and so forth. For example, consider the projects in Table 7.2 in which all projects have 2 rungs, but the parallelism in one of the rungs is increasing. A plot of the number of sequences generated and the computational time required to search the tree as a function of the number of rungs within rungs is shown in Figure 7.2. Please refer to Appendix A for the precedence digraphs of the projects in Table 7.2. Note that computational complexity does not increase as

Table 7.2: Projects with six activities and varying numbers of sub-rungs within rungs

Project	Num of Activities	Rungs within a Rung	Prod Func Dist	Num of Sequences	Search Tree Time
6C	6	1	None	24095	45.687
6B3	6	2	None	188	0.375
6B4	6	3	None	1685	3.171
6B5	6	4	None	24095	45.687

quickly in this case.

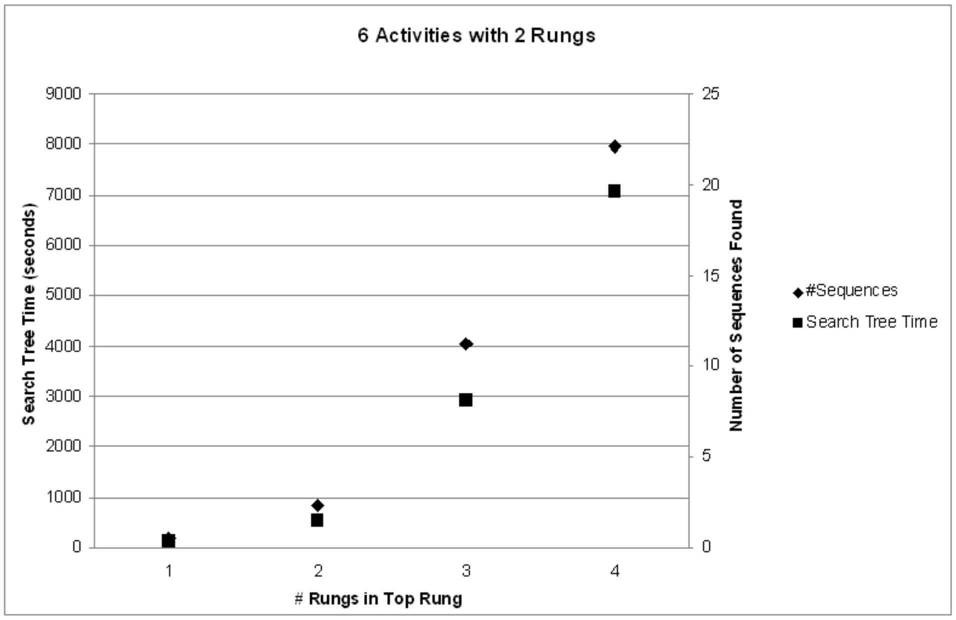


Figure 7.2: Computational complexity for projects with 2 rungs and various numbers of sub-rungs within the top rung.

However, in many cases, with more parallelism, the opportunities for creating independent events increases giving rise to an opportunity for reducing the problem's computational complexity. Consider the projects presented in Table 7.1 with all activities having concave productivity functions. A plot of the number of sequences generated and the computational time required to search the tree is shown in Figure 7.3.

Note that as the parallelism increases, the computational complexity of the problem does not increase quite as quickly as in the previous case. In fact, at very high levels of parallelism, the computational complexity decreases.

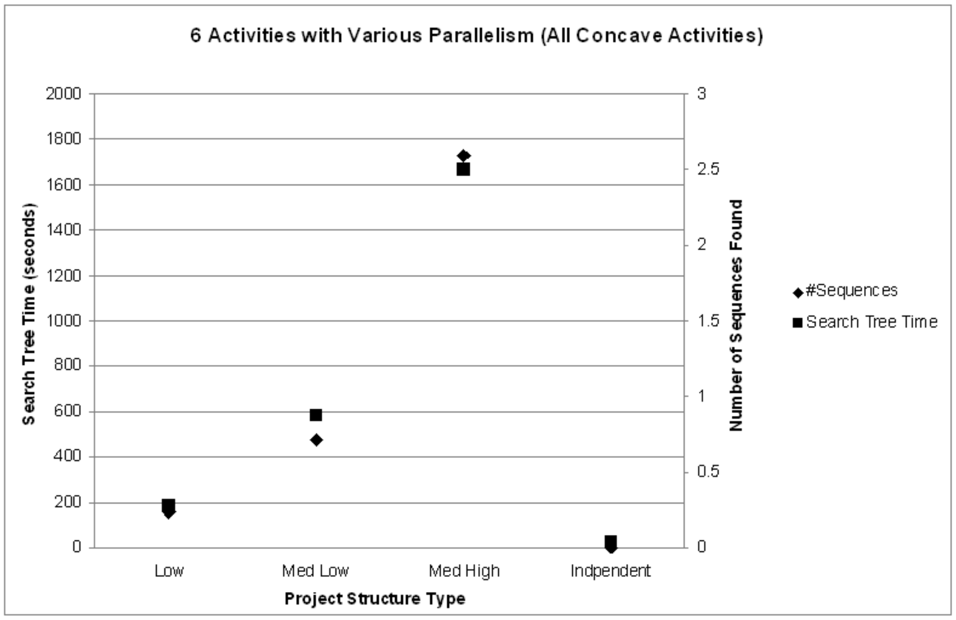


Figure 7.3: Computational complexity for projects with six activities and various levels of parallelism

7.4 Effects of Productivity Function Distribution

Consider a project with six independent events. Variations on this project may include productivity function distributions in which either 0 or 1 activities have convex or concave productivity functions (i.e., no exploitable relationships) or between 2 and 6 activities have concave or convex productivity functions. The projects are given in Table 7.3 Another variation on this project is that there are 2 activities with concave and 2 activities with convex productivity functions. A plot of the number of sequences generated and the computational time required to search the tree as a function of the distribution of productivity functions is shown in Figure 7.4. The computational complexity quickly decreases as the number activities with concave/convex productivity functions increases for projects with all independent activities. Note that even indices correspond to concave productivity function distributions,

Table 7.3: Projects with six independent activities and varying productivity function distributions

Index	Project	Num of Activities	Prod Func Dist	Num of Sequences	Search Tree Time
1	6D	6	None	159524	1175.81
2	6D	6	2 Concave	17529	24.25
3	6D	6	2 Convex	81416	349.063
4	6D	6	3 Concave	3792	4.468
5	6D	6	3 Convex	18709	29.781
6	6D	6	4 Concave	517	0.594
7	6D	6	4 Convex	4420	7.375
8	6D	6	5 Concave	32	0.063
9	6D	6	5 Convex	841	2.453
10	6D	6	6 Concave	1	0.031
11	6D	6	6 Convex	1	0.25
12	6D	6	2 Concave/2 Convex	5368	6.469

except for index 12; the odd indices correspond to convex productivity function distributions, except for index 1. The projects with concave productivity function distributions are consistently computationally less intensive than for the case with convex productivity function distributions.

However, for activities with more complicated structures, the position, and not just the number, of these special productivity functions, is important. Consider a project of type 6B (please see Appendix A). Activities 1 and 2 are in the “front”(F) and activities 3, 4, 5 and 6 are in the “back”(F). The projects are given in Table 7.4. A plot of the number of sequences generated and the computational time required to search the tree as a function of the distribution of productivity functions is shown in Figure 7.5.

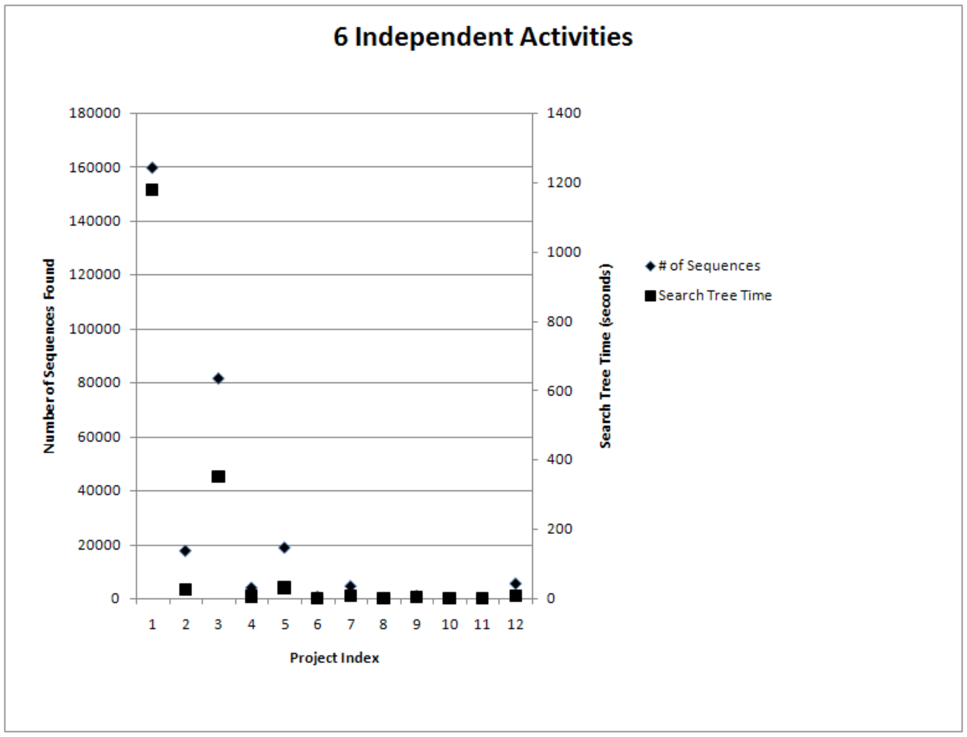


Figure 7.4: Computational complexity for projects with six independent activities and various productivity function distributions

Note that the easiest projects to develop solutions for have independent activity pairs with concave/convex productivity functions in the front of project. The pairs in the back do not have nearly as strong an effect.

7.5 Effects of Project Size

Consider the projects presented in Table 7.5. The number of activities in each of these projects varies between 5 and 8, however, they all have 3 rungs and all activities have concave productivity functions. Figure 7.6 depicts the computational effort required for these problems. It is clear that the effort required to find a solution increases exponentially with the number of activities for a given level of parallelism.

Table 7.4: Project with varying six activities and various productivity function distributions

Index	Project	Num of Activities	Prod Func Dist	Num of Sequences	Search Tree Time
1	6D	6	None	24095	45.687
2	6D	6	2 F Concave	7529	9.891
3	6D	6	2 F Convex	6758	9.453
4	6D	6	2 B Concave	19725	35.438
5	6D	6	2 B Convex	20397	36.296
6	6D	6	3 B Concave	15918	27.438
7	6D	6	3 B Convex	15963	28.203
8	6D	6	3 B &6 Concave	11245	19.187
9	6D	6	3 B& 6 6 Convex	8190	19.14
10	6D	6	2 F Convex and 2 B Concave	1365	1.828
11	6D	6	All Convex	1730	2.5
12	6D	6	All Convex	2231	4.547

Table 7.5: Projects with various number of activities

Project	Num of Activities	Prod Func Dist	Num of Sequences	Search Tree Time
5B	5	All Concave	93	0.11
6B2	6	All Concave		
7B	7	All Concave	3690	12.063
8B	8	All Concave	18884	1741.9

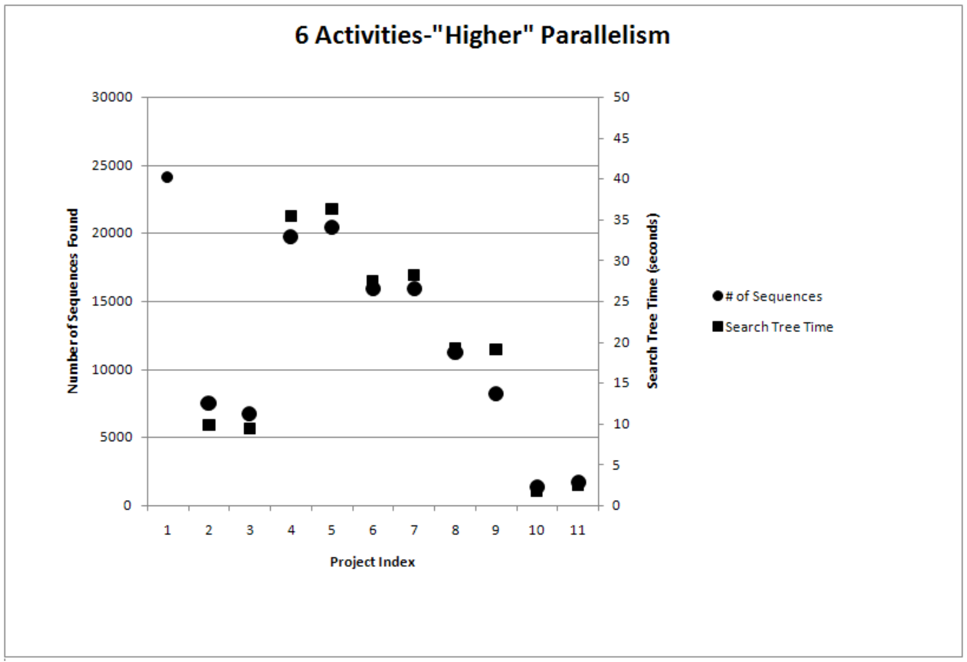


Figure 7.5: Computational complexity for projects with six activities and various productivity function distributions

In general, increasing the project size without affecting the other parameters will increase the computational complexity of the problem exponentially.

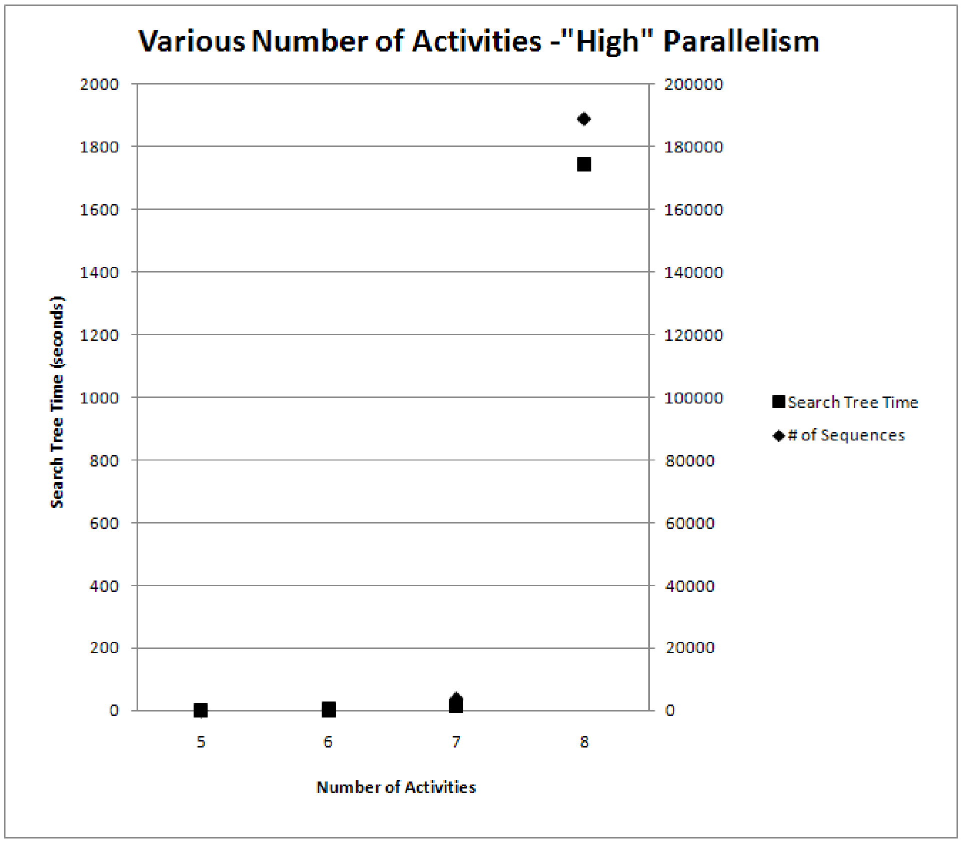


Figure 7.6: Computational complexity for projects with various number of activities

7.6 Conclusions

Our computational experiments show that the proposed solution methodology is most effective on projects in which there are many pairs of independent activities with concave/convex productivity functions towards the front of a project's precedence digraph. However, even if there is a small number activities in a project with exploitable properties, the decrease in computational complexity is substantial. Also, due to the nature of the solution methodology, very large-size projects with low parallelism are relatively easy to solve. Examples of relatively easy to solve for digraphs are depicted in Figures 7.7, 7.8,7.9. All activities have concave productivity functions. However, for many projects, our solution methodology

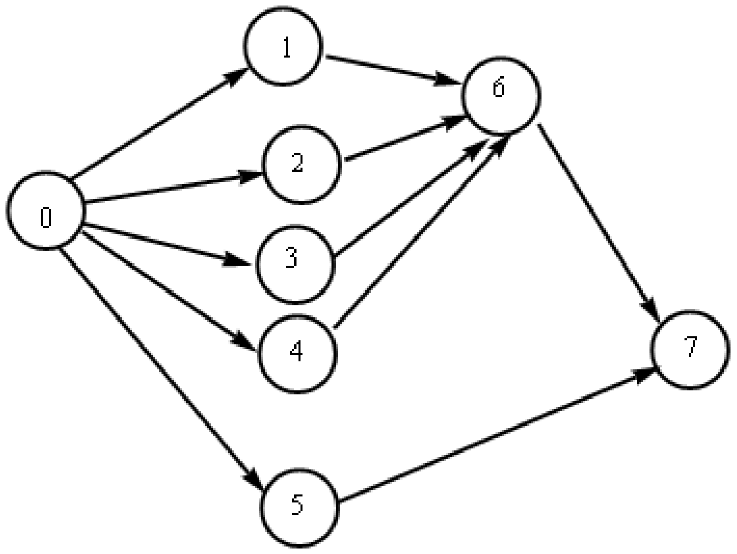


Figure 7.7: Precedence digraph 6A4

may not be satisfactory due to the size and nature of the activities involved. In this case, heuristics must be developed so that a good quality solution may be found in a reasonable amount of time. Potential heuristics are discussed in Chapter 8.

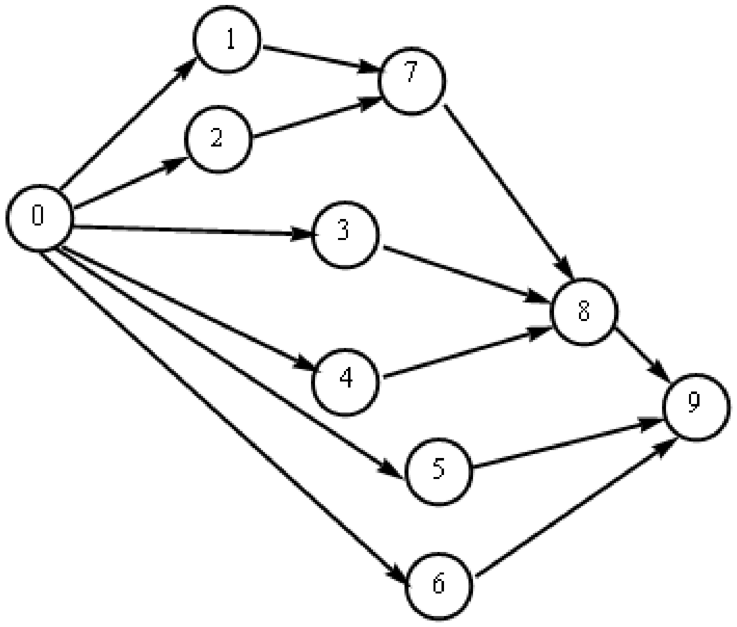


Figure 7.8: Precedence digraph 8B

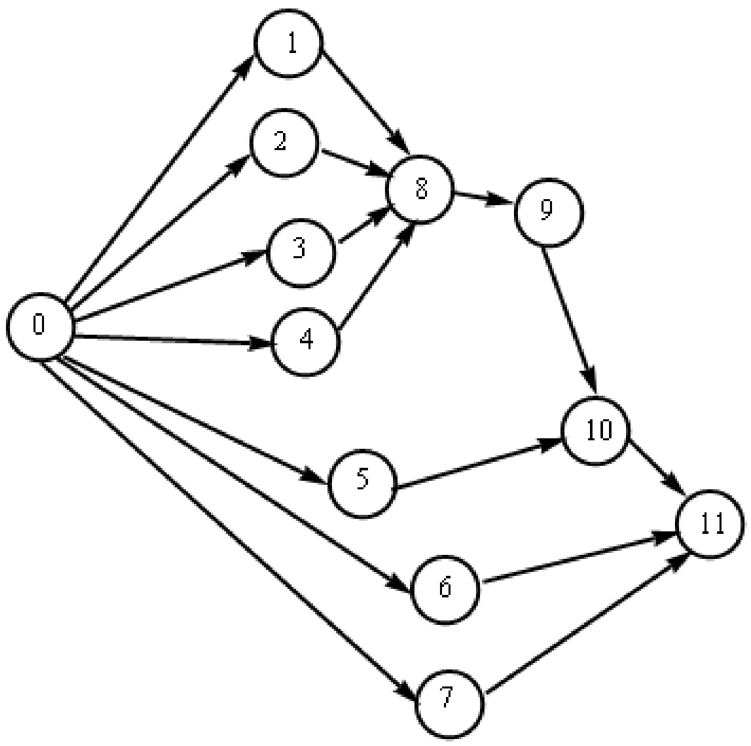


Figure 7.9: Precedence digraph 10C

Chapter 8

Future Work

8.1 Theoretical Work

Currently, the method developed can be used for projects in which there is only one infinitely divisible resource. However, additional resource flows may be added to the model along with their corresponding multi-dimensional time-resource curves. The challenge will be to develop similar rules for reducing the solution of to the problem.

Also, many projects have other constraints than those pertaining to resources and precedences. These include spatial constraints (e.g. two activities may not be performed in parallel since they must be performed in the same physical locate) and tooling constraints (e.g. two activities may not be performed in parallel since there is only one of a needed type of tool), among others. In reality, these are similar to resource constraints, and one may consider modeling them as such.

Another limiting characteristic of this model is that these activities cannot be pre-empted. Although this is realistic for many projects, it would be a more general model if this constraint were relaxed. This may be difficult to do with the flow based nature of the model. However, as presented in the literature review (see chapter 2), other researchers who have considered jobs to be pre-emptable have modeled the problem with discrete time by breaking the project down into time periods. Since the addition of time periods will add another integer variable to the problem, it may be best to avoid incorporating time period in this model.

Additionally, it would be beneficial to consider the cost minimization problem. Additional information on activity cost curves and/or resource usage rate cost would be needed to solve this problem.

8.2 Heuristic Development

A network reduction technique can be used at each node (including the initial node) of the search tree presented in Section 6, in which independent activities may be grouped into a "super-activity". For example, consider the precedence digraph in Figure 8.1. The circled nodes represent independent activities with both convex production rate functions. The other activities have productions functions which are neither concave nor convex. Based on the results presented in the previous section, activities 1 and 2 will be performed in series in an optimal sequence. As a part of the heuristic, activities 1 and 2 can be replaced by a super activity, say $1-2$, which is equivalent to 1 and 2 in series. The transformed precedence digraph for the project is shown in Figure 8.2.

After implementing this sequence enumeration algorithm (see Chapter 6), we find 390 different sequences for this network shown in Figure 8.1. In this preliminary implementation,

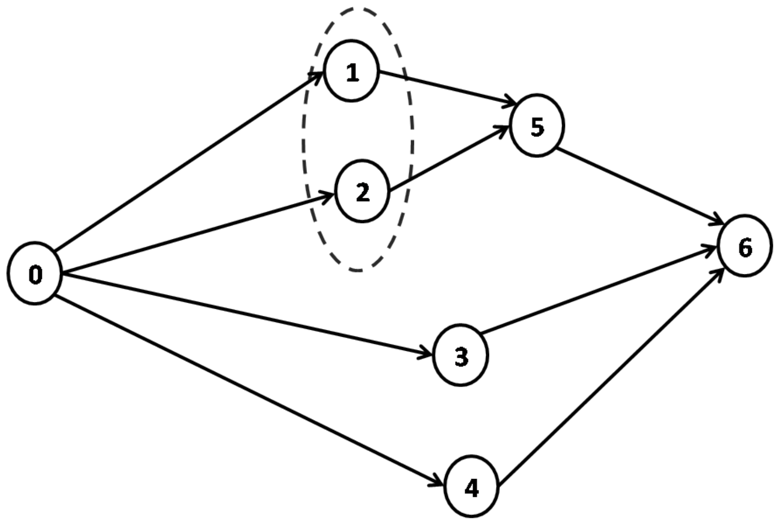


Figure 8.1: A project digraph with different forms of productivity functions

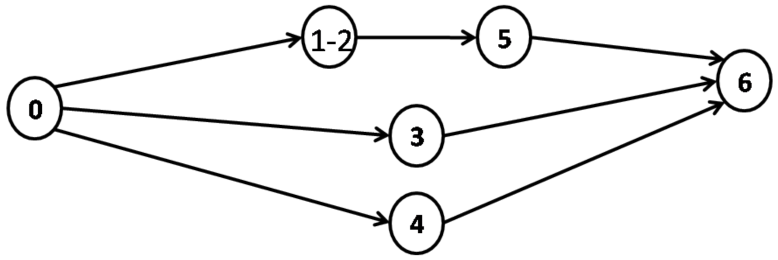


Figure 8.2: Transformed project digraph

some partial sequences that result in identical solutions are not deleted from these sequences. The number of sequences for the network in Figure 8.2 is 42, which amounts to a reduction of 89%. These heuristics can be very effective for the solution of real-life problems as it would generate a good and feasible solution very quickly.

Another potential heuristic is to consider activities with productivity functions which are "more convex/concave", i.e. their inflection point is shifted far to the right/left, to be convex/concave. This strategy may be used in conjunction with the network reduction heuristic just presented.

8.3 Software Development

As noted earlier, Mathematica is not the ideal language for the implementation of this solution methodology. It would be beneficial to rewrite the algorithm in C, or C++, which are lower level languages. Additional functions could be written to develop accessibility matrices from the precedence digraph. For ease of use, a GUI could be developed, which would enable ease of use, efficient data transfer, and provision of several options for the user.

Bibliography

[1] J. Alcaraz, C. Maroto, and R. Ruiz. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society*, 54(6):614–626, 2003.

[2] E. Balas and Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group. Project scheduling with resource constraints., 1968.

[3] M.S. Bazaraa, H.D. Sherali, and Shetty C.M. *Nonlinear Programming Theory and Algorithms*. Wiley, 2 edition, 1993.

[4] C.E. Bell and K. Park. Solving resource-constrained project scheduling problems by A* search. *Naval Research Logistics*, 37(1), 1990.

[5] E.B. Berman. Resource allocation in a pert network under continuous activity time-cost functions. *Management Science*, 10(4):734–745, July 1964.

[6] E.H. Bowman. The schedule-sequencing problem. *Operations Research*, pages 621–624, 1959.

[7] J. Carlier and B. Latapie. Un méthode arborescente pour résoudre les problèmes cumulatifs. *RAIRO. Recherche opérationnelle*, 25(3):311–340, 1991.

[8] J.A. Carruthers and A. Battersby. Advances in critical path methods. *OR*, pages 359–380, 1966.

[9] C. Chiang, Y. Huang, and W. Wang. Ant colony optimization with parameter adaptation for multi-mode resource-constrained project scheduling. *Journal of Intelligent and Fuzzy Systems*, 19(4-5):345–358, 2008.

[10] N. Christofides, R. Alvarez-Valdes, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

[11] E.W. Davis. *An exact alogorithm for the multiple constrained-resouce project scheduling problem*. PhD thesis, Department of Administrative Sciences, Yale Universtiy, 1968.

[12] E.W. Davis and G.E. Heidorn. An algorithm for optimal solutions in resource-constrained project sscheduling with multiple resource constraints. *Management Science*, 17:803–816, 1971.

[13] P. De, E.J. Dunne, J.B. Ghosh, and C.E. Wells. Complexity of the discrete time-cost tradeoff problem for project networks. *Operations Research*, pages 302–306, 1997.

[14] P. De, E. James Dunne, J.B. Ghosh, and C.E. Wells. The discrete time-cost tradeoff problem revisited. *European Journal of Operational Research*, 81(2):225–238, 1995.

[15] B. De Reyck, E. Demeulemeester, and W. Herroelen. Local search methods for the discrete time/resource trade-off problem in project networks. *Naval Research Logistics*, 45(6), 1998.

[16] B. de Reyck and W. Herroelen. Multi-mode resouce-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 119(2):538–56, December 1999.

[17] R.F. Decrko and J.E. Hebert. Resource constrained project crashing. *Omega International Journal of Management Science*, 17(1):69–79, 1989.

[18] R.F. Decrko and J.E. Hebert. Modeling diminishing returns in project resource planning. *Computers and Industrial Engineering*, 44:19–33, 2002.

[19] R.F. Decrko, J.E. Hebert, W.A. Verdini, and S. Venkateshwar P.H. Grimsrud. Nonlinear time-cost tradeoff models in project management. *Computers and Industrial Engineering*, 28(2):219–229, 1995.

[20] E. Demeulemeester, B. De Reyck, and W. Herroelen. The discrete time/resource trade-off problem in project networks: a branch-and-bound approach. *IIE transactions*, 32(11):1059–1069, 2000.

[21] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, pages 1803–1818, 1992.

[22] S.E. Elmaghraby. *Activity Networks.* New York: Wiley, 1977.

[23] S.E. Elmaghraby. Resource allocation via dynamic programming in activity networks. *European Journal of Operational Research*, 64(2):199–215, 1993.

[24] S.E. Elmaghraby and J. Kamburowski. The analysis of activity networks under generalized precedence relations (gprs). *Management Science*, 38(9):1245–63, September 1992.

[25] S.S. Erenguc, T. Ahn, and D.G. Conway. The resource constrained project scheduling problem with multiple crashable modes: An exact solution method. *Naval Research Logistics*, 48:107–127, 2001.

[26] R.T. Harvey and J.H. Patterson. An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis. *Foundations of Control Engineering*, 4(2):107–117, 1979.

[27] N.A.J Hastings. On resource allocation in project networks. *Operational Research Quarterly*, pages 217–221, 1972.

[28] B. Jarboui, N. Damak, P. Siarry, and A. Rebai. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Applied Mathematics and Computation*, 195(1):299–308, 2008.

[29] T.J.R. Johnson. *An algorithm for the resource-constrained project scheduling problem.* PhD thesis, School of Management, 1967.

[30] J. Józefowska, M. Mika, R. Różycki, G. Waligóra, and J. Weglarz. Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research*, 102(1):137–155, 2001.

[31] J.E. Kelley Jr. and M.R. Walker. Critical path planning and scheduling. In *Eastern Joint Computing Conference 16*, pages 160–172, 1959.

[32] R.C. Leachman, A. Dincerler, and S. Kim. Resource-constrained scheduling of projects with variable-intensity activities. *IIE Transactions*, 22(1), March 1990.

[33] J. Levy and J. Wiest. *A Management Guide to PERT/CPM.* Prentice-Hall, Cliffs, N.J., 2 edition, 1977.

[34] M. Liu, X. Cheng, M. Ge, S. An, and H. Li. Study on multi-mode resource-constrained project scheduling problem based on particle swarm optimization. *Nongye Jixie Xuebao/Transactions of the Chinese Society of Agricultural Machinery*, 39(2):134–138, February 2008.

[35] A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, pages 714–729, 1998.

[36] M. Mori and C.C. Tseng. A genetic algorithm for multi-mode resource constrained project scheduling problem. *European Journal of Operational Research*, 100(1):134–141, 1997.

[37] N. Nudtasomboom and S.U. Randhawa. Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. *Computers and Industrial Engineering*, 32(1):227–242, 1996.

[38] X. Pan, F. Liu, and L. Jiao. Clonal selection optimization for multi-mode resource constrained project scheduling problem. *Moshi Shibie yu Rengong Zhineng/Pattern Recognition and Artificial Intelligence*, 21(3):303–309, June 2008.

[39] D. Panagiotakopoulos. A CPM time-cost computational algorithm for arbitrary activity cost functions. *INFOR*, 15(2):183–195, 1977.

[40] J.H. Patterson and W. Huber. A horizon-varying, zero-one approach to project scheduling. *Management Science*, 20(6):990, February 1974.

[41] A.B. Pritsker and L.J. Watters. A zero-one programming approach to scheduling with limited resources. *The RAND Corporation*, RM-5561, January 1968.

[42] A.B. Pritsker, L.J. Watters, and P.M. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, 16(1):93, September 1969.

[43] F.J Radermacher. Scheduling of project networks. *Annals of Operations Research*, 4(1):227–252, 1985.

[44] M. Ranjbar, B. De Reyck, and F. Kianfar. A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research*, 193(1):35–48, 2007.

[45] M.R. Ranjbar and F. Kianfar. Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms. *Applied Mathematics and Computation*, 191(2):451–456, 2007.

[46] S. Sakellaropoulos and A.P. Chassiakos. Project time-cost analysis under generalised precedence relations. *Advances in Engineering Software*, 35:715–25, 2004.

[47] R. Slowinski. Two approaches to problems of resource allocation among project activities - a comparative study. *Journal of Operational Research Society*, 31:711–23, 1980.

[48] J.P. Stinson, E.W. Davis, and B.M. Khumawala. Multiple resource–constrained scheduling using branch and bound. *IIE Transactions*, 10(3):252–259, 1978.

[49] F. B. Talbot. Resource-constrained project scheduling with time-resource tradeoffs: the non-preemptive case. *Management Science*, 28(10):1197–1210, 1982.

[50] F.B. Talbot and J.H. Patterson. An efficient integer programming algorithm with network cuts for solving resource-constrained scheduling problems. *Management Science*, pages 1163–1174, 1978.

[51] J. Weglarz. Time-optimal control of resource allocation in a complex of operations framework. *IEEE Trans. Systems, Man and Cybernetics*, 6:783–788, 1976.

[52] J. Weglarz. Project scheduling with continuously divisible resources. *Management Science*, 27(9):1040–53, September 1981.

[53] Z Wei-cun and K Kai. Ant colony and particle swarm optimization algorithm-based solution to multi-mode resource-constrained project scheduling problem. *Computer Engineering and Applications*, 43(34):213–216, December 2008.

[54] Y. Yu. Hybrid genetic algorithm for multi-mode resource-constrained project scheduling problems. *Dongnan Daxue Xuebao (Ziran Kexue Ban)/Journal of Southeast University (Natural Science Edition)*, 38(4):736–740, July 2008.

Appendix A

Precedence Digraphs Used in Testing

3A, 4B, 5D, 6C, and 7C are all precedence digraphs with independent events.

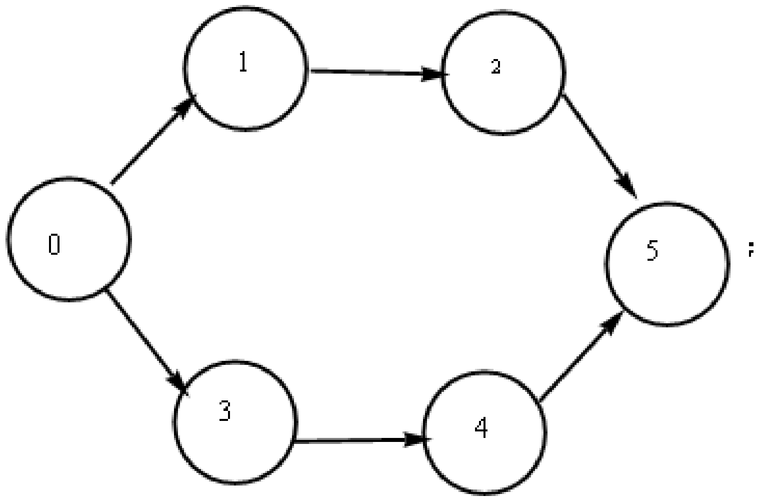


Figure A.1: 4A

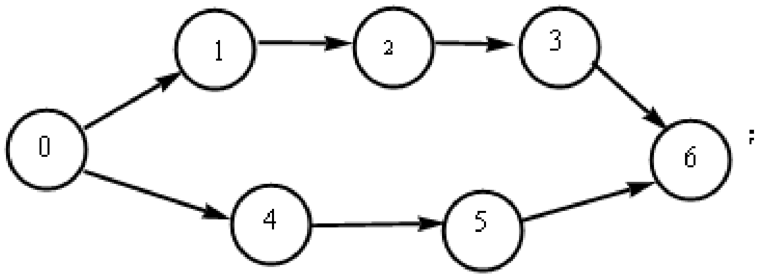


Figure A.2: 5A

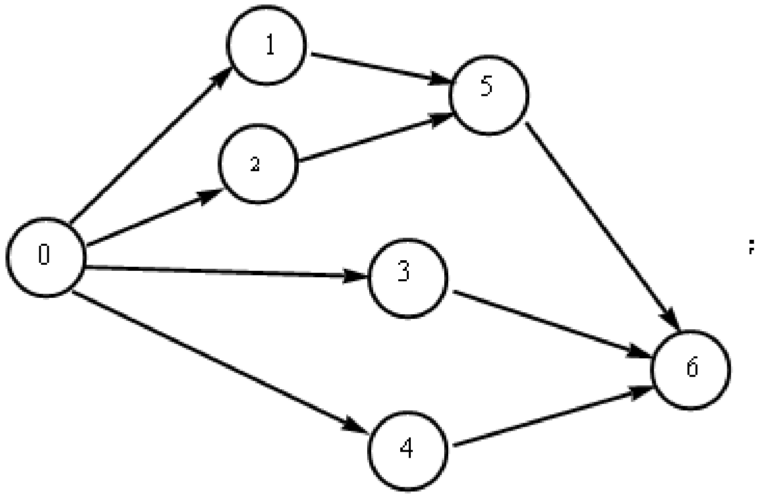


Figure A.3: 5B

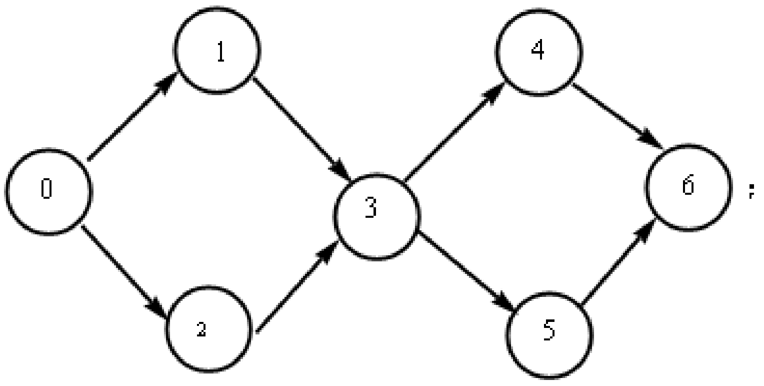


Figure A.4: 5C

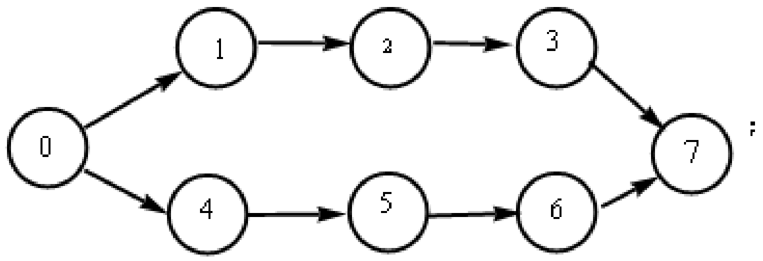


Figure A.5: 6A

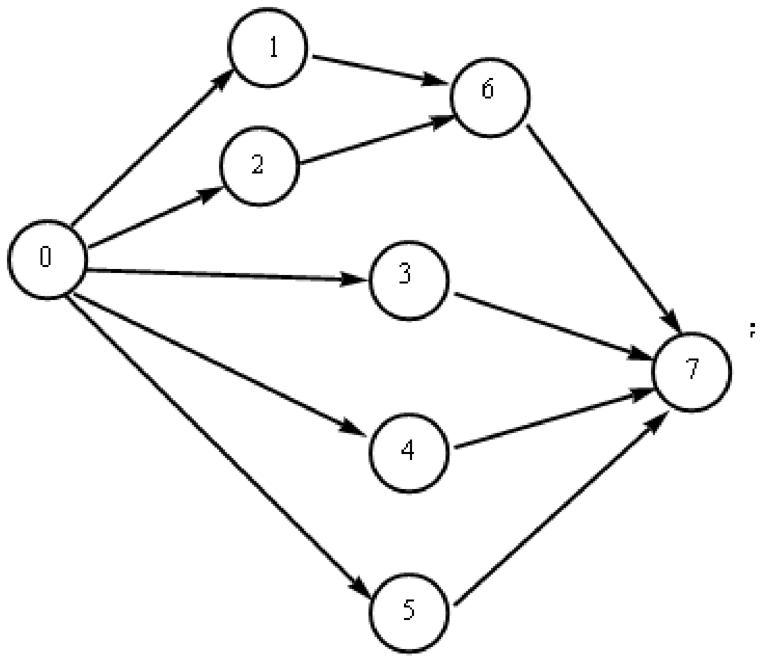


Figure A.6: 6B

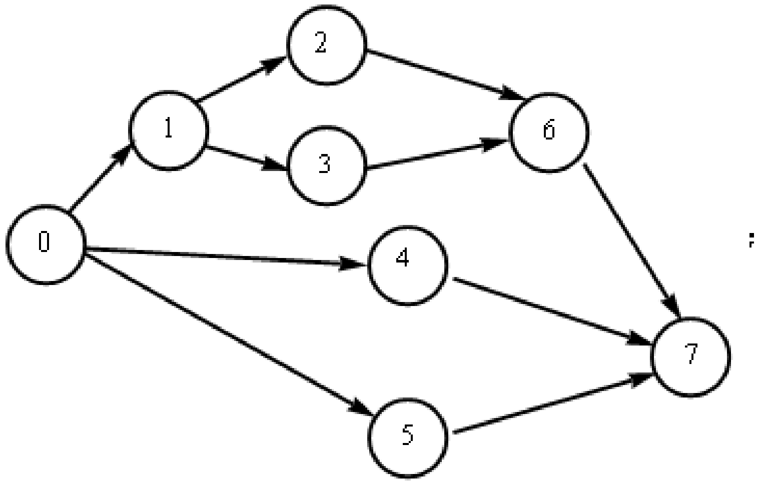


Figure A.7: 6B2

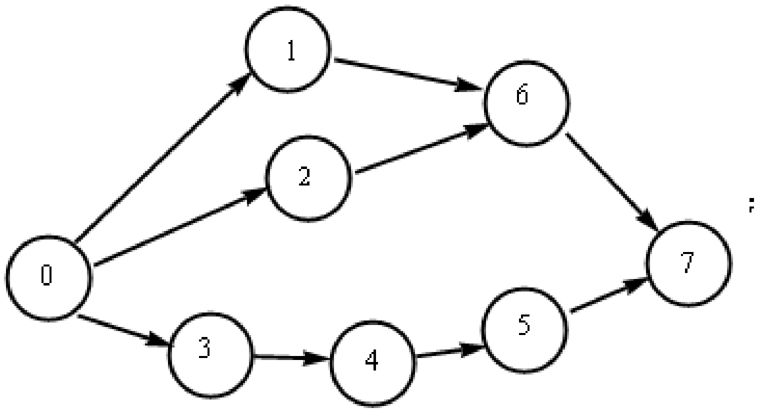


Figure A.8: 6A2

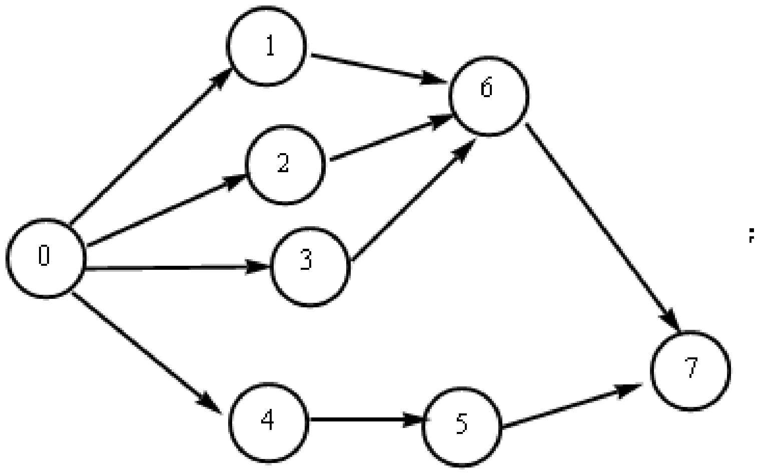


Figure A.9: 6A3

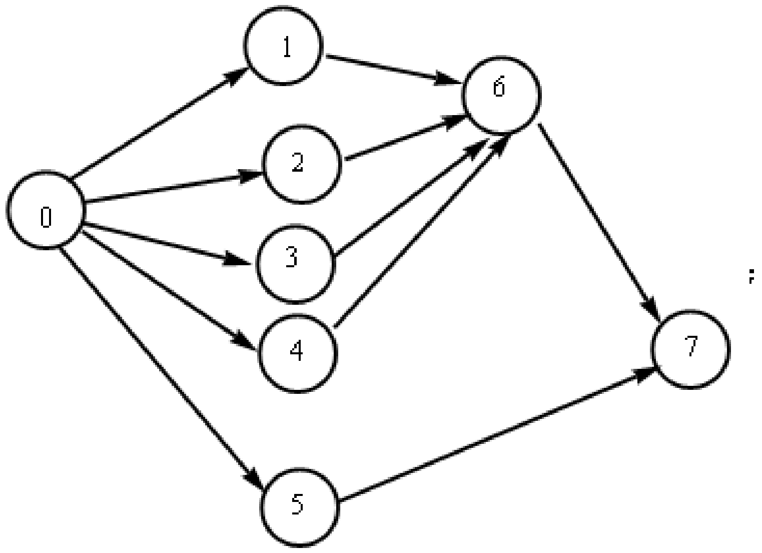


Figure A.10: 6A4

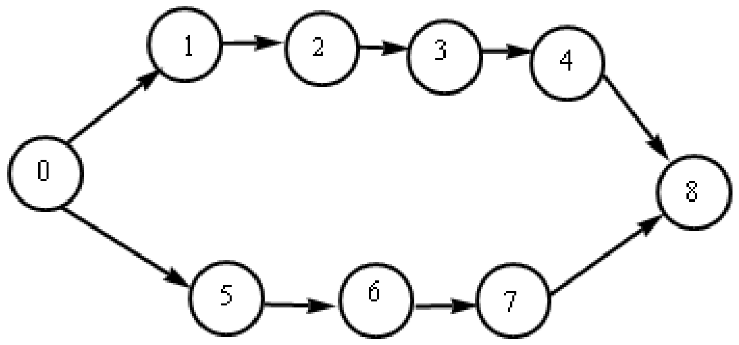


Figure A.11: 7A

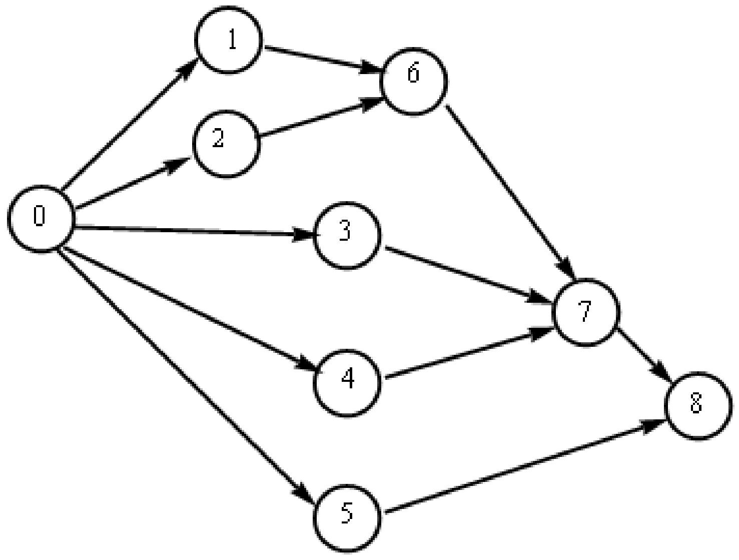


Figure A.12: 7B

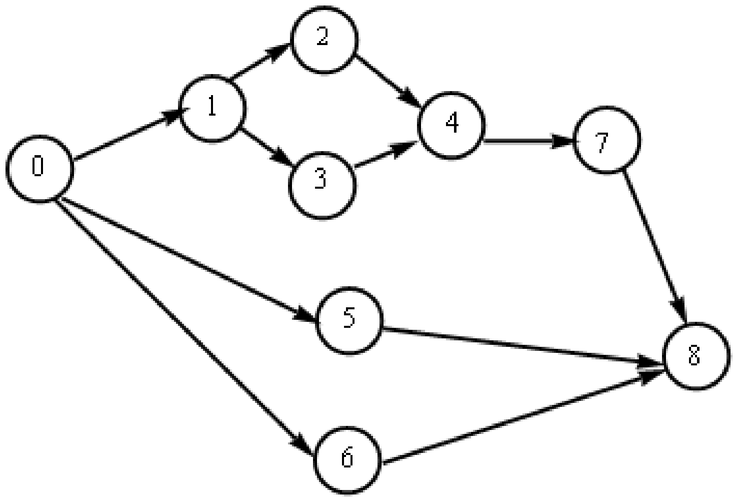


Figure A.13: 7B2

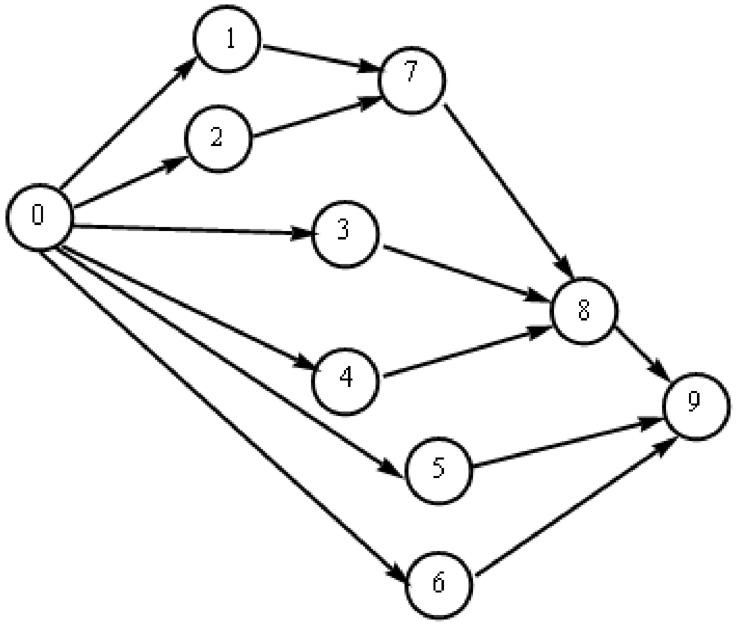


Figure A.14: 8B

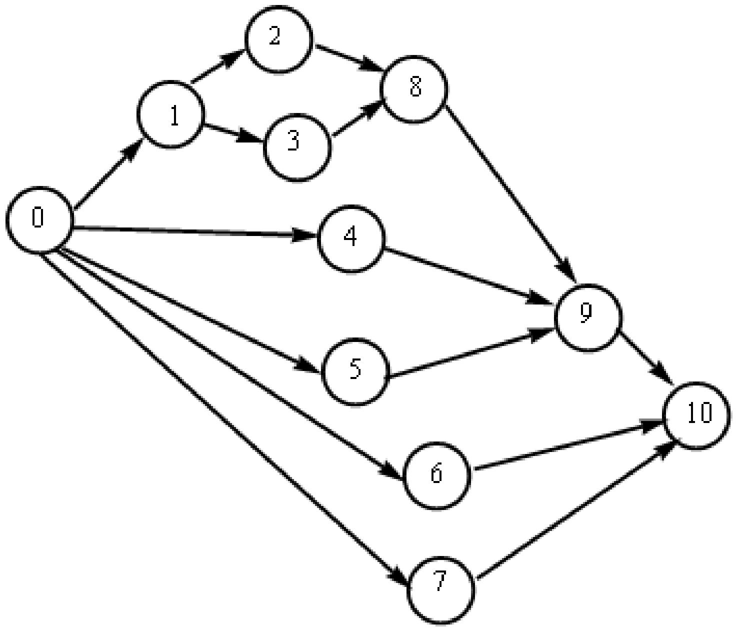


Figure A.15: 9B

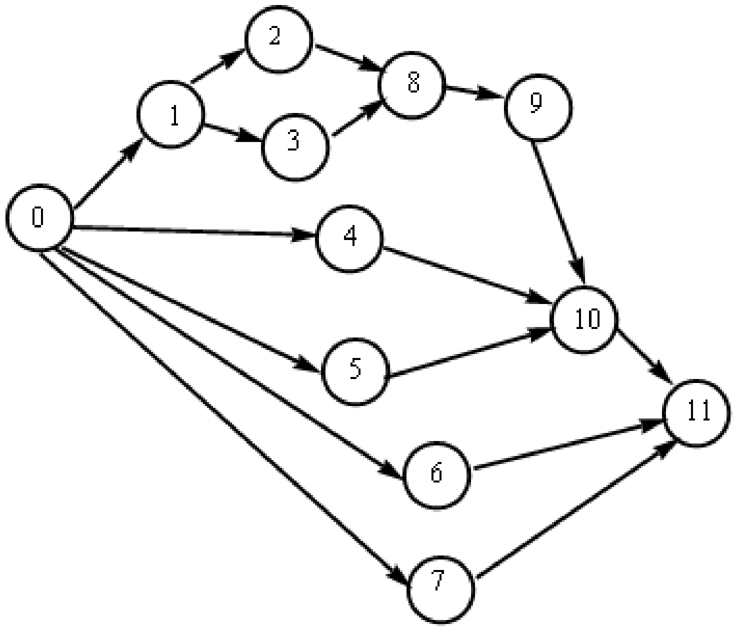


Figure A.16: 10B

Appendix B

Pseudo and Mathematica Code for Enumeration Algorithm

B.1 Pseudo Code

B.1.1 Precedence Path Algorithm

Inputs:

- A set of partial paths, *paths*. This is a jagged array containing all partial precedence paths.
- An array of immediate predecessors for each activity, *Precs*. This is a jagged array with the first entry being the predecessors of the sink, 0, which is defined to be -1. The i^th entry corresponds to activity i.

Outputs:

The precedence paths of a project (i.e. the number of paths from the source to the sink)

Algorithm:

- Find the number of paths in the partial path, l
- Set a counter, c, to be equal to 0. For each path, or element, of l if the last activity of that path equals 0, then increase c, by 1.
- For each path,j, in $paths$, find the position of the last activity in the path, $lp2$, the index of the activity in that position, $Opnum$, and the number of precedences of activity $Opnum$, numPrec.
- Nested: For each predecessor, i, of $path(j, lp)$, if the predecessor is -1 (indicating we are at the source), then append this path to the new set of partial paths, s. Else, append i to $path(j, lp)$ and append this to the new set of partial path s.
- If the $c = l$ then all partial paths are full paths and we return s. Else, we run this function again inputing s.

B.1.2 Search Tree

Main Function:

Inputs:

- Set of just previously sequenced activities, R
- Current Accessibility Matrix, $Acur$

- Set of activities of which no flows were assigned out of, $RowHold$
- Set of activities which have been sequenced, $Done$
- 0/1 variable indicating if the previously sequenced are not serial,$BigY$
- Set of previously found independent activities, $oldIndConv$
- An array of immediate predecessors for each activity, $Precs$. This is a jagged array with the first entry being the predecessors of the sink, 0, which is defined to be -1. The i^th entry corresponds to activity i
- An array of immediate successors for each activity, $Successors$. The i^th entry corresponds to activity i.
- Set of forbidden serial sequences, $Serial$
- number of activities, n
- Set presently found sequences, $Branches$
- Precedence paths for the project, $FullPrecs$
- 0/1 variable indicating the serial case has been found, $SerialDone$

Outputs:

All possibly optimal sequences for a project)

Algorithm:

- If $Acur = \{0\}$ this indicates that the branch previously traversed is not to be traversed for some reason. Exit this instance of the Main function to go back to the previous node and choose another branch.
- Append to R, $RowHold$
- For each activity in R, find the activities accessible to it. Record these potential next activities in Y, in which the i^th element of Y corresponds to the activities accessible to the activity of the i^th element of R. An activity,j, is accessible to another activity,k, if j's precedences have been met and $Acur(k, j) == 1$.
- If the first element of Y equals the index of the sink, then we will append $Acur$ to branches in most cases. However, if $SerialDone$ is 1 and $BigY = 0$, then we would be repeating a serial sequencing of the activities and so we go back up a level. If it is the first serial sequencing found, then set ”SerialDone” equal to 1.
- Find the concave and convex independent activities by setting $IndConvConc$ equal to $FindIndAct$. $IndConv$ equals the sorted first element of $IndConvConc$ and $IndConc$ equals the sorted second element of $IndConvConc$.
- Create an array in which each element is an array of the subsets of each element of Y. Denote this array as $Ysubsets$.
- Find $Ytuples$, all the tuples of the elements of $Ysubsets$
- If the length of $Ysubsets$ is 1 then, drop the null set from the first element $Ysubsets$

- If, $c1$, the number of activities in R complement $RowHold$ is greater than 0, for each element in R complement $RowHold$ find all subsets of its successors, $SubsetsSucCur$. For each element in $SubsetsSucCur$, remove the element from Ytuples which is the tuple will all positions empty except the position corresponding to the current element of R and that will be equal to current subset of $SubsetsSucCur$.
- If there are more than 1 independent activities or more *OldIndConv* is greater than 1, then set $Ytuples$ equal to $FathomNodes$
- Run Main again for each element of $Ytuples$

"Else" Function:

Inputs:

- Set of just previously sequenced activities, R
- Current Accessibility Matrix, $Acur$
- A possible branchings from a node, $Ytuples$
- Set of activities which have been sequenced, $Done$
- 0/1 variable indicating if the previously sequenced are not serial,$BigY$
- Set of previously found independent activities, $oldIndConv$
- An array of immediate predecessors for each activity, $Precs$. This is a jagged array

with the first entry being the predecessors of the sink, 0, which is defined to be -1. The i^th entry corresponds to activity i

- An array of immediate successors for each activity, $Successors$. The i^th entry corresponds to activity i.
- Set of forbidden serial sequences, $Serial$
- number of activities, n
- Precedence paths for the project, $FullPrecs$
- 0/1 variable indicating the serial case has been found, $SerialDone$

Outputs:

None

Algorithm:

- If the length of $Ytuple$ is 1 or if the length of the Flattened $Ytuple$ is 1 then $BigY = 1$
- Check to see if any of flows in $Ytuple$ are null. If so, add the corresponding R to $RowHold$.
- Check to see if any of the activities to be flowed to have activities that are already in progress, i.e. if the any of the predecessors of an activity in $Ytuple$ are also in $RowHow$

then we cannot flow to that activity yet. Set $Anew = 0$ and go to the main function call.

- Check to see $Ytuple$ is of size 1. If it is, then check to see if an equivalent serial sequence have been done already. If it has, then set $Anew$ to 0 and go to the main function call. If not, add this sequence to the $Serial$ array.
- Find $Yunion$, a list of the activities which will be flowed to next.
- Eliminate the null positions from $Ytuple$ and the corresponding elements of R. The new sets will be called $Ytuplemod$ and $Rmod$
- Call the Amod function. Store the output in B.
- Set $Anew$ equal to the first element of B.
- Set $Done$ equal to the second element of B.
- Call the main fucntion.

Amod Function:

Inputs:

- Set of just previously sequenced activities, R
- Current Accessibility Matrix, $Acur$
- A possible branchings from a node, $Ytuple$

- List of activities to be flowed to next, $Yunion$
- Set of activities which have been sequenced, $Done$
- An array of immediate predecessors for each activity, $Precs$. This is a jagged array with the first entry being the predecessors of the sink, 0, which is defined to be -1. The i^th entry corresponds to activity i
- An array of immediate successors for each activity, $Successors$. The i^th entry corresponds to activity i.
- Set of forbidden serial sequences, $Serial$
- number of activities, n
- Set presently found sequences, $Branches$
- Precedence paths for the project, $FullPrecs$
- 0/1 variable indicating the serial case has been found, $SerialDone$

Outputs:

Modified A matrix

Algorithm:

- For each activity with an out flow, look at each activity in the network, j. If the outgoing activity is not flowing to j, zero that position in $Acur$.

- For each position in *Ytuple*, k, and then for each position in *Ytuple*, j, for each (k, j) and i, $i = 1, ..., n$, do not zero $Acur(i, Ytuple(k, j)$ if i is in R. If i is not in R then add $Ytuple(k, j)$ to *Done*.

- Return (Acur, Done)

```
findPath[path_,Pre_]:=
Module[{p={},s,c,m,lp1,lp2,Opnum,numPrec,z,l,Precs=Pre},
   s={};
   l=Length[path];
   c=0;
   For[m=1,m<=l,
      lp1=Length[path[[m]]];
      If[path[[m]][[lp1]]==0,c++
      ];
      m++
   ];
   For[j=1,j<=l,
      lp2=Length[path[[j]]];
      Opnum=path[[j]][[lp2]];
      numPrec=Length[Precs[[Opnum+1]]];
      For[i=1,i<=numPrec,
         z=path[[j]];
         If[Precs[[Opnum+1]][[i]]==-1,AppendTo[s,z],
            AppendTo[z,Precs[[Opnum+1]][[i]]]; AppendTo[s,z]];
      i++];
   j++];
   If[c==l,p=s,findPath[s,Precs]]
]
Else[FromNode_]:=
Module[{Acur=(*A*) FromNode[[3]],Ytuple=FromNode[[2]],
        R=FromNode[[1]],  Done=FromNode[[4]],B,Yunion,Anew,
        c2,c21,c211,c3,c12, FlatYtuple, UnionFlatYtuple,
        Ytuplemod,Ytuplemod1,Rmod1,Rmod,RowHold,RowHold1,
        LengthYtuple, LengthUnionFlatYtuple,LengthR,
        BigY=FromNode[[5]],OldIndConv=FromNode[[6]]},
   RowHold={};
   LengthR=Length[R];
   (*If the entering Tuple of new assignments is null,
   then signal Branching to go back up a level*)
   FlatYtuple=Flatten[Ytuple];
   If[FlatYtuple=={},Anew={0};Yunion={};Goto["c"]
   ];
   If[Length[Ytuple]||Length[FlatYtuple]>1,
      BigY=1
   ];
   (*If a particlur position of a Tuple is null then that
     indicates that we are making no flow this round out
     of the corresponding previously assigned activity.
     We should hold this activity over and treat it as
     if it were assigned this round*)
   LengthYtuple=Length[Ytuple];
   RowHold=Complement[Table[If[Ytuple[[i]]=={},
                      R[[i]]],{i,LengthYtuple}],{Null}];
   (*Check to see if the Tuple has an in progress activity*)
   UnionFlatYtuple=Union[FlatYtuple];
   LengthUnionFlatYtuple=Length[UnionFlatYtuple];
   c12=1;
   While[c12<=Length[UnionFlatYtuple],
      If[Intersection[Precedences[[UnionFlatYtuple[[c12]]+1]],
                  RowHold]!={},
         Anew={0};Yunion={};Goto["c"]
      ];
      c12++
   ];
   (*Print["Here?4"];*)
   (*Print["ForcedMove=",ForcedMove];*)

   (*Check To See if the positions in the tuple are identical*)
   If[LengthR==1 && LengthUnionFlatYtuple==1,
      If[MemberQ[Serial,{R,Ytuple[[1]]}]==False,
         AppendTo[Serial,{Ytuple[[1]],R}],
         Yunion={}; Anew={0}; Goto["c"]
      ]
   ];
   (*Compile a list of all activities to be flowed to next*)
   Yunion=Ytuple[[1]];
   For[j=2, j<= LengthR,j++,
      Yunion=Union[Yunion,Ytuple[[j]]]
   ];
   (*If position c3 is null in the Ytuple the we will elimnate it
     from the Ytuple and carry R[[c3]] over as just assigned*)
   c3=1;
   Ytuplemod={};
   Rmod={};
   While[c3<= LengthYtuple,
      If[Ytuple[[c3]]!={},
         AppendTo[Ytuplemod,Ytuple[[c3]]];AppendTo[Rmod,R[[c3]]]
      ];
   c3++
   ];
   (*Print["Here?8"];*)
   (*Pause[2];*)
   B=Amod[Rmod,Ytuplemod,Yunion, Acur, Done];
   (*Print["Here?9"];*)
   Anew = B[[1]];
   Done=B[[2]];
   Label["c"];
   Branching[Yunion, Anew,Done, RowHold,
            BigY,OldIndConv]
]

Amod[Row_,Combo_,Uni_,A_,Assigned_]:=
Module[{Acur = A,Ytuple=Combo, Yunion=Uni,
        R=Row,Done=Assigned,k,Unidentical,LengthYunion,LengthYtuple},
   LengthYunion=Length[Yunion];
   LengthYtuple=Length[Ytuple];
   (*For each activity with an out flow, look at each activity
     in the network, j.  If the outgoing activity is not following
     to j, zero that position in the accessibility matrix*)
   For[i=1, i<= Length[R],i++,
      For[j = 1, j <= n,j++,
         If [Intersection[{j}, Ytuple[[i]]]=={},
            Acur[[R[[i]]+1,j]]=0
         ]
      ]
   ];

   (*Look at each position in the incoming Ytuple*)
   For[k=1,k<= LengthYtuple,k++,
      (*Look at each activity in the kth position in
        the Ytuple*)
      For[j=1,j<= Length[Ytuple[[k]]],j++,
         (*Look at each activity in the project*)
         For[i =1,i<= n, i++,
            (*If activity i is not in NoKill, then zero that
              position out in the accessibility matrix*)
            If[Intersection[{i-1},R(*NoKill*)]=={},
               Acur[[i,Ytuple[[k,j]]]]=0]];
               AppendTo[Done,Ytuple[[k,j]]
            ]
         ]
      ]
   ];
   {Acur,Done}
]
FindIndAct[NextAct_,Finished_]:=
Module[{Y=NextAct,Yflat,IndEv,count1,count2,IsInd,count3,IndAct,count4,
      Done=Finished,IndConv,IndConc,count5,count6,count21,count211,z,y,
      LengthYflat,LengthIndEv,LengthIndAct},
    Yflat =Union[Flatten[Y]];
   LengthYflat=Length[Yflat];
   (*Find the Potential Independent Activities*)
   IndEv={};
   count1 =1;
   (*We need to loop through all the potential next activities*)
   While[count1<=LengthYflat,
      count2=1;
      IsInd=1;
      (*We need to loop through them again so pairwise
         comparisons can be made*)
      While[count2<=LengthYflat,
         count21=1;

         (*Loop through the successors of the current next
           possible activity
           for the first loop*)
         While[count21<=Length[Successors[[Yflat[[count1]]+1]]],
            count211=1;
            (*Inspect a successor of the current next possible
              activity for the first loop*)
            z=Successors[[Yflat[[count1]]+1,count21]];
            (*Loop throught the successors of the current next
              possible activity for the second loop*)
            While[count211<=Length[Successors[[
                                      Yflat[[count2]]+1]]],
               (*Inspect a successor of the current next
                  possible activity for the second loop*)
               y=Successors[[Yflat[[count2]]+1,count211]];
               (*If the successor to the current next possible
                 activity from the first loop is a Successor to
                 one of the successors of the next possible
                 activities in the second loop, then the next
                 possible activity from the first loop is not
                 independent*)
               If[MemberQ[FullPrec[[z]], y(*Successors[[y+1]],z*)]
                  ==True && count1!=count2,
                  (*if y+1 is on z's precedence path!!!*)
                  IsInd=0
               ];
            (*Look at the next successor of the current next
             activity in the second loop*)
            count211++;
            ]
         (*Look at the next successor of the current next
           activity in the first loop*)
         count21++
         ];
         (*Look at the next next activity in this second loop*)
         count2++
      ];
      (*If the this next activity for the first loop is independent,
        record it as such*)
      If[IsInd==1,
         AppendTo[IndEv, Yflat[[count1]]]
      ];
      (*Look at the next next activty for this first loop*)
    count1++;
    ];

   (*Find Actual Independent Activities*)
   (*Can't figure out why this is here.
     Seems like it should work with out it but
     I don't want to mess up something that
     is working*)
   count3=1;
   IndAct={};
```

```
LengthIndEv=Length[IndEv];
While[count3<=LengthIndEv,
   count4=count3+1;
   While[count4<=LengthIndEv,
      If[Complement[Successors[[IndEv[[count3]]+1]],Done]==
         Complement[Successors[[IndEv[[count4]]+1]],Done],
      AppendTo[IndAct,IndEv[[count3]]];
      AppendTo[IndAct,IndEv[[count4]]]
      ];
      count4++
   ];
   count3++
];

(*Check each pair independent activities for a reducable
  relationship*)
IndConv={};
IndConc={};
count5=1;
IndAct=Flatten[IndAct];
LengthIndAct=Length[IndAct];
If[LengthIndAct>1,
   While[count5<=LengthIndAct,
      count6=1;
      While[count6<=LengthIndAct,
         If[Curve[[IndAct[[count5]]+1]]==
            Curve[[IndAct[[count6]]+1]]&&count5!=count6,
            If[Curve[[IndAct[[count5]]+1]]==1,
               AppendTo[IndConv,
                     {IndAct[[count5]],IndAct[[count6]]}],
               If[Curve[[IndAct[[count5]]+1]]==2,
                  AppendTo[IndConc,
                        {IndAct[[count5]],
                         IndAct[[count6]]}]
               ]
            ]
         ];
         count6++
      ];
      count5++
   ]
];
```

```
   IndConv=Union[Flatten[IndConv]];
   IndConc=Union[Flatten[IndConc]];
   {IndConv,IndConc}
]
FathomNodes[Conv_,Conc_,Tuple_,Row_,oldIndConv_]:=
Module[{IndConv=Sort[Conv],IndConc=Sort[Conc],Ytuples=Tuple,YtuplesNew,
      T,P,Fathom,PFathom,count10,count11,count12,
      LengthIndConv, LengthIndConc, LengthYtuples, R=Row,
      OldIndConv=oldIndConv,pos,
LengthOutYtuples},
   LengthIndConv=Length[IndConv];
   LengthIndConc=Length[IndConc];
   LengthYtuples=Length[Ytuples];
   (*Print["R=",R];
   Print["Ytuples=",MatrixForm[Ytuples]];
   Print["IndCov=",IndConv];
   Print["OldIndConv=",OldIndConv];*)
   If[IndConv=={},IndConv=OldIndConv];
   (*Print["IndCov=",IndConv];*)
   If[Length[OldIndConv>1]||Length[IndConv]>1,
      If[OldIndConv == {},
      Ytuples=Select[Ytuples,If[Length[Union[Flatten[#]]\
                   [Intersection]IndConv]==1,True,False]&],
      pos=Flatten[Position[R,(R\[Intersection]OldIndConv)[[1]]]];
      Ytuples=Select[Ytuples,  If[And[Length[Union[Flatten[#]]
                                \[Intersection]IndConv]==1,
              #[[pos[[1]]]]\[Intersection]IndConv==Union[Flatten[#]]\
              [Intersection]IndConv],True,False]&]
      ]
   ];

   If[LengthIndConc>1,
      Ytuples=Select[Ytuples, Flatten[#]\
                    [Intersection]IndConc==IndConc &]
   ];

   (*If[LengthIndConc>1,
      count10 =1;
      While[count10<=LengthYtuples,
         Fathom=0;
         T=Ytuples[[count10]];
         count11=1;
         While[count11<=Length[T],
            P=T[[count11]];
            PFathom=1;
               If[Intersection[IndConc,P]!=Sort[IndConc],
                  PFathom=0;
               ];
               If[PFathom==1,
                  Fathom=1
               ];
            count11++;
         ];

         If[Fathom==1,
            AppendTo[YtuplesNew,T]
         ];
         count10++;
      ];
      Ytuples=YtuplesNew
   ];*)
   {Ytuples}
]

Branching[Row_,A_,Assigned_,AddToRow_(*,MustMove_*),bigY_,oldIndConv_]:=
Module[{R=Join[Row,AddToRow],RowHold=AddToRow,Acur=A,Done=Assigned,
               (*Limb=Branch,*)RSubsets,p,Y,Ysubsets,Ytuples,
               aZeroRow, Yflat,count1,count2,
               count3,count4, count5, count6,count7,count8,
               count9,count10,count11,count12,IsInd,IndEv,
               IndAct,IndConv,IndConc,YtuplesNew,Act,Act1,
               P,T,Fathom,IndConvConc,
               LengthR,LengthYtuples,LengthRowHold,SucCurR, CurR,pos,
               SubsetsSucCurR, ForbiddenMoves,c1,c2,BigY=bigY,
               ElseTable,Ytuples2,OldIndConv=oldIndConv},
   count++;
   If[Mod[count,100000]==0,Print[count]];
   LengthR=Length[R];
   LengthRowHold=Length[RowHold];
   (*If the Acur == {0} then it is a sign to go back up a level*)
   If[Acur=={0}, Return[1]
   ];
   (*Find all the possible successors of previously assigned
   activities*)
   Y={}; (*Variable which will hold possible successors of
         previously assigned activities*)
   (*For each previously assigned activity, search through
    it's respective row in the accessibility matrix for
   accessible jobs make sure the precedence requirments
   have been met*)
   For[i=1, i<= LengthR, i++,
      AppendTo[Y,{}];
      For[j =1, j<= n,j++,
         h=R[[i]]+1;
         If[And[Acur[[h,j]] ==1,And[Intersection[{j}, Done]!={j},
               Sort[Intersection[Precedences[[j+1]],Done]]
                           ==Sort[Precedences[[j+1]]]]],
            AppendTo[Y[[i]],j]
         ]
      ]
   ];
   Do[If[Flatten[Y[[i]]]==0,Return[1]],{i,1,Length[Y]}];
   If[Y[[1]]=={n},
      If[BigY==0 && SerialDone==1,
         Return[1]
      ];
      If[BigY==0 && SerialDone==0,
            SerialDone=1
      ];
      AppendTo[Branches,Acur];
      Return[1]
   ];
   IndConvConc=FindIndAct[Y,Done];
   IndConv=Sort[IndConvConc[[1]]];
   IndConc=Sort[IndConvConc[[2]]];
   (*ForcedMove1={};*)
   (*If[IndConv!={},
      ForcedMove1 = {IndConv[[1]],IndConv[[2]]};
   ];*)
   (*Variable to hold all subsets for each Y
               Form:{ { {},{something},...,{something}},...
                   ,{ {},{something},...,{something}} }*)
   Ysubsets=Table[Subsets[Y[[i]]],{i,LengthR}];
   (*We do not want to include the null set in our subset
    of Y if there is only one previously assigned activity
```

```
     which only has one possible flow*)
   (*Print["LengthYsubsets=",Length[Ysubsets]];*)
   If[Length[Ysubsets]==1, Ysubsets[[1]]=Complement[Ysubsets[[1]],{{}}]];
   (*Print["Ysubsets=",Ysubsets];*)
   Ytuples=Tuples[Ysubsets];
   If[LengthR-LengthRowHold>0,
      c1=LengthR-LengthRowHold+1;
         While[c1<=LengthR,
            CurR = R[[c1]];
            c2=1;
            SucCurR=Successors[[CurR+1]];
            SubsetsSucCurR = Complement[Subsets[SucCurR],{}];
            While[c2<=Length[SubsetsSucCurR],
               Ytuples=Complement[Ytuples,
                              {Table[If[i==c1,SubsetsSucCurR[[c2]],{}],
                               {i,1,LengthR}]}];
            c2++
            ];
            c1++
         ]
   ];

   (*Reduce Network*)
   If[Length[IndConv]>1 || Length[IndConc]>1||Length[OldIndConv]>1,
      Ytuples=FathomNodes[IndConv,IndConc,Ytuples,R,OldIndConv][[1]]
   ];
   LengthYtuples=Length[Ytuples];
   ElseTable=Table[{R,Ytuples[[i]],Acur,Done
  ,BigY,IndConv},{i,1,LengthYtuples,1}];
   Map[Else,ElseTable];
]
SearchTree[A_,Precs_,AllPrec_,Sucs_,CC_,NumJobs_]:=
   Module[{Acur=A},
   Curve=CC;
   Precedences=Precs;
   FullPrec=AllPrec;
   Successors=Sucs;
   n=NumJobs;
   Serial={};
   SerialDone=0;
   count=0;
   Branches={};
   (*SetSharedVariable[Precedences,Successors,n, Serial];*)
   (*SetSharedFunction[Else,Amod,FathomNodes,FindIndAct];*)
   T=Timing[Branching[{0},Acur,{0},(*{1,0},*){}(*,{}*),0,{}]];
   (*Print[Curve,Length[Branches],T];*)
   {Branches,T}
 ];
```