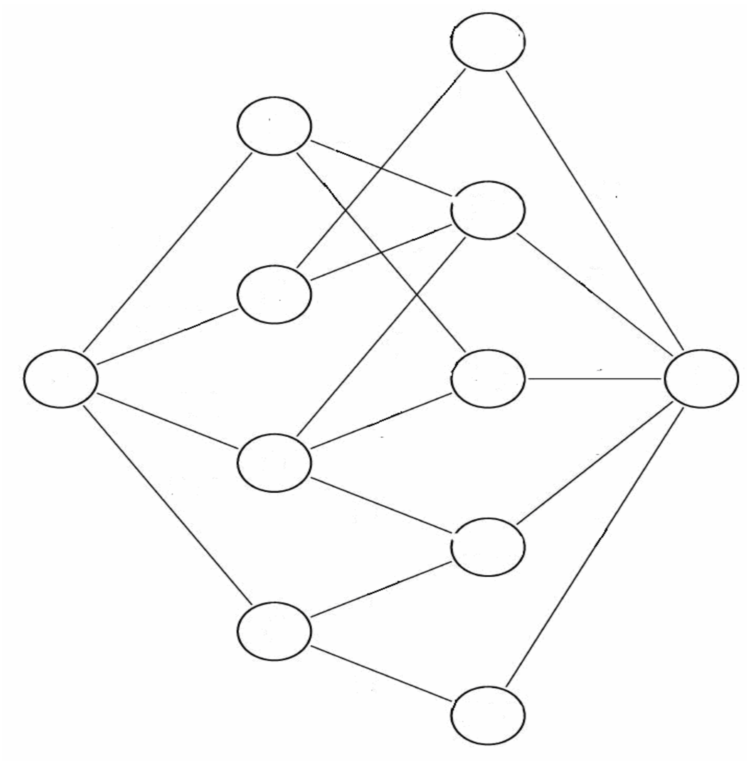
Job Management & Network Flows

How to solve job management problems with networks flow techniques



Francesca Armandillo

BMI paper

Vrije Universiteit
Faculty of Sciences
Business Mathematics and Informatics
De Boelelaan 1081
1081 HV Amsterdam

September 2006

Preface

This paper has been written as part of the course Business Mathematics and Informatics (BMI). The BMI-paper is one of the final subjects. The scope of this work is to investigate the available literature in reference to a topic related to at least two out of the three fields integrated in the BMI course. Specifically, this paper deals with the use of network flow techniques to solve Job Management problems.

This BMI paper first starts with an overview of some networks flow problems. Then several applications are presented that illustrate the practical importance of these models in the Project Management.

I would like to thank Dr. Evert Wattel for supervising this research and giving me this interesting idea for investigating this subject and for the available time, advice and his always enthusiastic feedback!

Francesca Armandillo
Amsterdam, September 2006

Table of contents

1 Introduction

The scheduling of large projects is an important class of network problems. This application context was among the earliest successes of network optimization, and the network flow models of project management continue to be an important management tool used in numerous industries every day. This paper shows that network flow techniques and project management are closely related and how these techniques are used to solve some recent project management problems.

This paper is constructed as follows, first some network flow problems are discussed who will be used later on to solve three project management problems. The three project management problems are:

- How to schedule jobs on uniform parallel machines,
- How to determine the minimum project duration when there are precedence relations between the jobs,
- Just-in Time (JIT) scheduling. This former is an extension to the problem on how to determine the minimum project duration with some extra constraints added. In this problem, the minimum project duration is determined subject to both the precedence constraints and some additional “just-in-time constraints”.

The paper is divided in 4 Chapters. After the introduction, Chapter 2 and 3 illustrate the topics of: *Network flow problems* and *Project Management*. In Chapter 2 several network flow problems are discussed like the Minimum Cost Flow problem, Shortest Path problem, and Maximum Flow problem. These techniques will be needed to solve project management problems in Chapter 3 where typical Project Management problems are presented and discussed. Finally, Chapter 4 provides the summary and conclusion.

In this paper it is assumed there that the reader has basic knowledge concerning graph theory. The basis concepts from the graph theory will be used in this paper thus mostly without further introduction. For more background information concerning graph theory the reader is referred to standard graph books in this area. The second part of the dictations written by the Open University and used at the course Discrete Mathematics at the Vrije Universiteit gives a good introduction in the graph theory.

In this paper no management summary has been incorporated, for this it is referred to the abstract and conclusion in Chapter 4.

2 Network Flow Problems

In this section some network flow models are discussed. These models will be used to solve several applications in *Project Management* that illustrate the practical importance of these models.

2.1 System of difference constraints

The importance of this paragraph is that many network flow problems can be transformed into a *minimum cost flow* problem. The minimum cost flow model is an optimization model that can be written as a mathematical programming formulation. Due to the frequently described mathematical programming formulation and its relation to its associated graph in this paper, a short theoretical text is here introduced to explain how a system of difference constraints is related to its associated constraint graph.

Suppose a linear system has the following set of *m* difference constraints in the *n* variables $x=(x(1),x(2),\ldots,x(n))$:

$$x(j_k) - x(i_k) \le b(k) \qquad \textbf{for each k=1…m}$$

Each system of difference constraints has an associated graph *G*, the *constraint graph*. This graph is used to determine if a system of difference constraints has a feasible solution. The constraint graph has *n* nodes corresponding to the *n* variables and *m* arcs corresponding to the *m* difference constraints. An arc (i_k,j_k) of length *b(k)* in *G* is associated with the constraint $x(j_k) - x(i_k) \le b(k)$.
As an example consider the following system of constraints whose associated graph is shown in Figure 2.1.

$$\begin{aligned}
&x(3) - x(4) \le 5\\
&x(4) - x(1) \le -10\\
&x(1) - x(3) \le 8\\
&x(2) - x(1) \le -11\\
&x(3) - x(2) \le 2
\end{aligned}$$

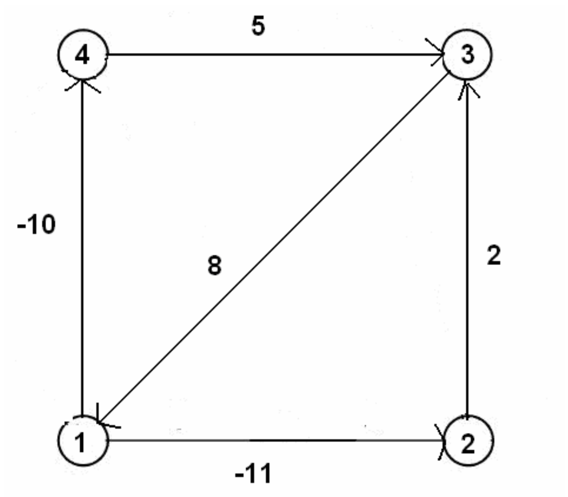


Figure 2.1 Graph corresponding to a system of difference constraints

Later on in this paper, it will be shown that the constraints are identical with the *optimality conditions* for the *shortest path problem.* These conditions can only be satisfied if the network contains no negative cycle. A *negative* cycle is a directed cycle whose total weight is negative.
The network shown in Figure 2.1 contains a negative cycle 1-2-3 of length *-1*, and the corresponding constraints [$x(1) - x(3) \leq 8$, $x(2) - x(1) \leq -11$, $x(3) - x(2) \leq 2$] are inconsistent because summing these constraints yields the invalid inequality 0$\leq$-1. So there is no feasible solution of this system of constraints.

2.2 Minimum Cost Flow Problem

The minimum cost flow model is the most fundamental of all network flow problems. It is possible to transform many network flow problems into a minimum cost flow problem. This is shown in the *Minimum cost problem formulation* paragraph under each different network flow problem.
The minimum cost flow problem is used to solve the Just-in-Time scheduling application discussed later on in this paper (Chapter 3.1.3).

2.2.1 Definition

Let *G=(N,A)* be a directed network defined by a set *N* of nodes and a set *A* of directed arcs. Each arch *(i,j)* $\in A$ has an associated cost c_{ij} that denotes the cost per unit flow on that arc.
Each arc *(i,j)* $\in A$ also has an associated capacity u_{ij} that denotes the maximum amount of flow on the arc and a lower bound l_{ij} that denotes the minimum amount that must flow on the arc.

Each node $i \in N$ has an associated integer number *b(i)* representing its supply/demand.

- If *b(i)>0*, node *i* is a supply node;
- if *b(i)<0*, node *i* is a demand node with demand of *-b(i);*
- if *b(i)=0*, node *i* is a transhipment node.

In most applications, like the ones that are discussed in this paper, the lower bounds on arc flows are assumed to have value zero. An example of a network with *supply* and *demands* nodes is shown in Figure 2.2.1.

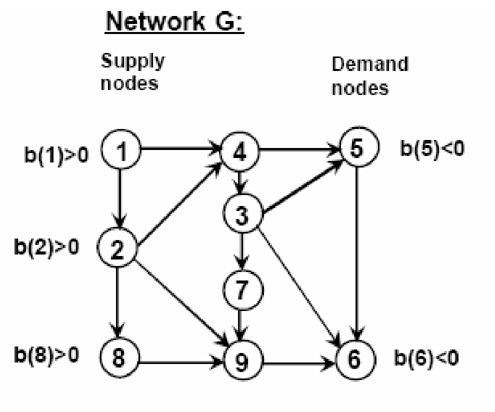


Figure 2.2.1 Network with demands/supply nodes

2.2.2 Mathematical programming formulation

In paragraph 2.2.2.1, the standard Primal and Dual form is explained and paragraph 2.2.2.2 discusses the *primal* and *dual* form of the Minimum Cost flow problem.

2.2.2.1 Standard Primal-Dual form

Linear programming problems are optimization problems (or mathematical programming problems) in which the objective function and the constraints are all linear. Every linear programming problem, referred to as a primal problem, can be converted into a dual problem, which provides an upper bound to the optimal value of the primal problem.

In optimization theory, the duality principle states that optimization problems may be viewed from either of two perspectives, the primal problem or the dual problem. The duality theorem states that in the linear case there is a direct computational equivalence between the solution to the primal problem and the solution to the dual problem: the maximum objective function values of the *dual* equals the minimum objective function of the *primal*.

There is a *dual* for any maximization problem which is a minimization problem. If the primal problem is a *maximizing* problem, then the dual problem is a *minimizing* problem (and reversed).

The standard primal form of a linear programming can be expressed as follows:

Maximize $\sum_{j=1}^{n} c_j x_j$

subject to $\sum_{j=1}^{n} a_{ij} x_j$ **{ $\leq$, or = or $\geq$} b_i for i=1,…m**

and $x_j \geq$ **0 for j=1,..n**

The standard primal form is a maximizing problem with $\leq$,$\geq$or = constraints and the objective function is a linear combination of *n* variables and *m* constraints. The goal is to maximize the value of the objective function subject to the constraints. A solution is a vector (a list) of *n* values that achieves the maximum value for the objective function.

In the dual problem, the objective function is a linear combination of the *m* values that are the limits in the *m* constraints from the primary problem. There are *n* dual constraints, each of which places a lower bound on a linear combination of *m* dual variables.

The *dual* problem of the *primal* problem stated above, can be formulated as:

Minimize $\sum_{i=1}^{m} b_i y_i$

subject to $\sum_{i=1}^{m} a_{ij} y_i \geq c_j$ **for j=1,…n (with a,b&c being the same as in the primal problem)**

and

- **$y_i \leq$ 0 if the ith primal constraint is a "$\geq$" constraint**
- **$y_i \geq$ 0 if the ith primal constraint is a "$\leq$" constraint**
- **y_i can be as well negative as positive (unrestricted) if the ith primal constraint is a "=" constraint**

As already mentioned, the *dual* form has *m* decision variables and *n* constraints. A new variable y_i is associated with each constraint *i* in the primal problem.

So summarized, from a standard maximum problem, the *dual* problem is found as follows:
1. A variable is associated to every constraint in the *primal*.
2. The coefficients on the right side of the main constraints of the *primal* become the coefficients of the objective function of the *dual*.
3. The coefficients of the variable x_1 in the *primal* become the coefficients of the first constraint of the *dual* (etc).
4. The inequalities are reversed.

Here below an example is given to illustrate how the *dual* problem is worked out.

Maximize $7x_1 + 5x_2 + 4x_3$

subject to $\begin{cases} 2x_1 + 3x_2 + 5x_3 \geq 8 \\ x_1 + x_2 + x_3 = 2 \end{cases}$

and $x_1, x_2, x_3 \geq 0$

Then the dual is:

Minimize $8y_1 + 2y_2$

subject to $\begin{cases} 2y_1 + y_2 \geq 7 \\ 3y_1 + y_2 \geq 5 \\ 5y_1 + y_2 \geq 4 \end{cases}$

and $y_1 \leq 0$ **and** y_2 **unrestricted**

Another example is given that illustrates the *dual* problem.

Maximize $c_1 x + c_2 y$

subject to $\begin{cases} Ax \leq b \\ By = d \end{cases}$

and $x \geq 0$

A variable is associated to each constraint in the *primal*, obtaining two variables *u* and *v*. Then the maximum problem above is turned into a *dual* problem:

Minimize $ub + vd$

subject to $\begin{cases} uA \geq c_1 \\ vB = c_2 \end{cases}$

and $u \geq 0$ **,**v **unrestricted**

The book *Modelbouw in de Operations Research* (*Tijms 1994*) can give more insight about this subject.

2.2.2.2 Primal-Dual form of Minimum Cost Flow problem

The decisions variables in the minimum cost flow problem are arc flows and the flow on an arc *(i,j)* $\in A$ is represented by x_{ij}. The minimum cost flow problem is the problem to find a flow from *s* to *t* with minimum cost.
This flow problem is an optimization model formulated as follows:

$$\text{Minimize} \quad \sum_{(i,j)\in A} c_{ij} x_{ij} \qquad \textbf{(2.2.2 a)}$$

$$\text{subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \text{ for all } i \in N \qquad \textbf{(2.2.2 b)}$$

$$0 \le x_{ij} \le u_{ij} \text{ for every arc } (i,j) \in A \qquad \textbf{(2.2.2. c)}$$

where $\sum_{i=1}^{N} b(i) = 0$

Constraint (*2.2.2 b*) is referred as the *mass balance constraints.*
It just states that the outflow minus inflow must equal supply/demand of the node. If the node is a *demand* node, its inflow exceeds its outflow; and if the node is a *supply* node, its outflow exceeds its inflow. If *b(i)=0*, the inflow equals its outflow.
Constraint (*2.2.2 c*) is referred as the *flow bound constraints*, which states that a flow must also satisfy the lower bound and the capacity constraints.

In this paper it is assumed that the networks only have transhipment nodes, in this way *b(i)* always equals zero. This type of linear problem can be referred as a *primal* problem. The dual problem can be found as illustrated in paragraph 2.2.2.1; for the minimum cost flow problem, the variable *π(i)* can be associated with the mass balance constraint of node *i*, and the variable α_{ij} with the capacity constraint of arc *(i,j).* In terms of these variables, the dual minimum cost flow problem can be stated as follows according to *Orlin (1993)*:

Maximize $\sum_{i\in N} b(i)\pi(i) - \sum_{(i,j)\in A} u_{ij}\alpha_{ij}$

subject to $\pi(i) - \pi(j) - \alpha_{ij} \le c_{ij}$ **for all (i,j)** $\in$ **A**

$\alpha_{ij} \ge 0$ **for all (i,j)** $\in$ **A**

and $\pi(j)$ **unrestricted for all j** $\in$ **N**

2.2.3 Successive Shortest Path Algorithm

Many algorithms for solving the minimum cost flow problems combine ingredients of both *shortest path* and *maximum flow* algorithms. Here only one, the *successive shortest path* algorithm, is discussed. This algorithm is one of the most fundamental algorithms for solving the *minimum cost flow* problem. This algorithm incrementally loads flow on the network from some source to some sink node, each time selecting an appropriately defined shortest path.
It selects at each step a node *s* with excess supply (i.e. supply not yet sent to some demand node) and a node *t* with unfilled demand and sends flow from *s* to *t* along a shortest path in the residual network. With residual network the "remaining flow network "is meant.
The algorithm terminates when the current solution satisfies all the mass balance constraints.

Definition

A *residual network* denoted *G(x)* corresponding to a flow *x* is defined as follows. Replace each arc $(i,j) \in A$ by two arcs *(i,j)* and *(j,i).* The arc *(i,j)* has cost c_{ij} and *residual capacity* $r_{ij}=u_{ij}-x_{ij}$, and the arc *(j,i)* has cost $c_{ji}=-c_{ij}$ and residual capacity $r_{ji}=x_{ij}$. The residual network consists only of arcs with positive residual capacity.

Figure 2.2.3 gives an example of a residual network.

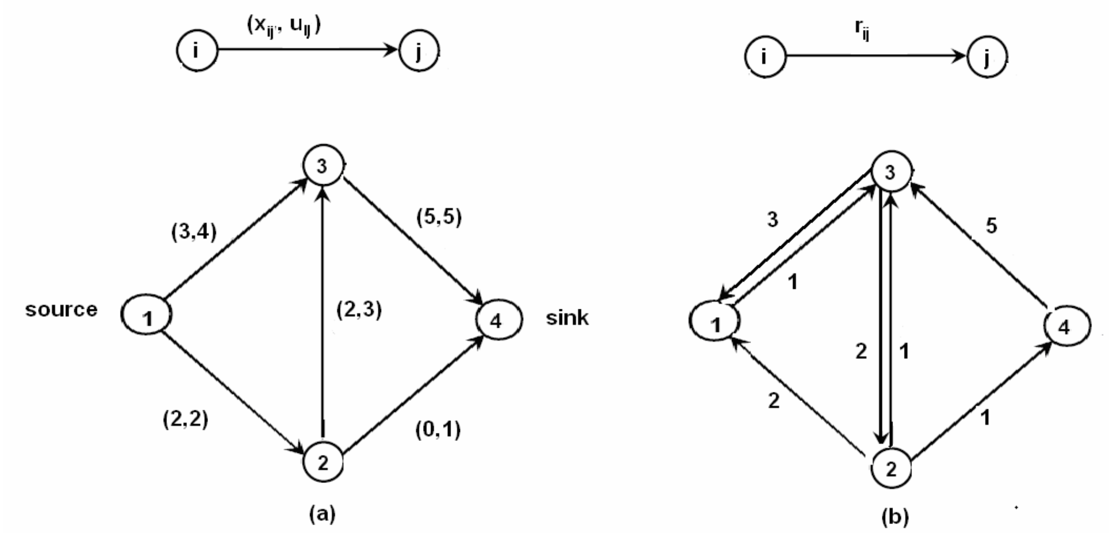


Figure 2.2.3 Illustrating a residual network: (a) original network G with flow x; (b) residual network G(x)

First some concepts are introduced before the algorithm is described.
In the discussion about the *Shortest Path Problem* in Chapter 2.3.1, the *shortest path optimality conditions* are defined.
These optimality conditions can be written in the following way:

$c_{ij}^{d} = c_{ij} + d(i) - d(j) \geq 0$ **for all arcs (i,j) $\in$ A**

This expression has the following interpretation: c_{ij}^{d} is an optimal “reduced cost” for arc *(i,j)* in the sense that it measures the cost of this arc relative to the shortest path distances *d(i)* and *d(j).*

Suppose that a real number π *(i),* is associated to each node *i* $\in$ *N*. This number π *(i)* is referred as the potential of node *i* and is the linear programming dual variable corresponding to the mass balance constraints for node *i* (paragraph *2.2.2).* For a given set of nodes potentials, de r*educed* cost of an arc *(i,j)* is defined as : $c_{ij}^{\pi} = c_{ij} - \pi(i) + \pi(j)$. This because π*=-d* (see Chapter 2.3.2).
These reduced costs are applicable to the residual network as well as the original network.

The *successive shortest path* algorithm maintains “optimality” of the solution at every step and terminates when the current solution satisfies all the mass balance constraints. The term *optimality* is now described with the use of the next theorem.

Theorem(Reduced Cost optimality Conditions) A feasible solution *x** is an *optimal* solution of the minimum cost flow problem if and only if some set of node potentials π satisfy the following reduced cost optimality conditions.

$c_{ij}^{\pi} \geq 0$ **for every arc (i,j) in G(x*)**

To describe the *successive shortest path* algorithm, the concept of *pseudo-flows* is introduced. A *pseudo-flow* is a "flow" vector *x* such that *0 < x < u*. It satisfies only the capacity and nonnegativity constraints: it need not satisfy the mass balance constraints.
For any *pseudo-flow x*, the imbalance of node *i* is defined as.

$$e(i) = b(i) + \sum_{\{j:(j,i)\in A\}} x_{ji} - \sum_{\{J:(i,j)\in A\}} x_{ij} \qquad \textbf{for all i} \in \textbf{N}$$

If *e(i)>0* for some node *i* , *e(i)* is referred as the *excess* of node *i*; if *e(i)<0*, then *–e(i)* is called the node’s *deficit*. A node i with *e(i)=0* is called *balanced*. The residual network corresponding to a *pseudo-flow* is defined in the same way that the residual network is defined for a flow.

The successive shortest path algorithm is shown in Figure 2.2.3(i) and will be illustrated with the example shown in Figure 2.2.3 (ii) where the initial residual network is shown.

```
algorithm successive shortest path;
begin
    x:=0 and $\pi$:=0;
    e(i):=b(i) for all i ∈ N
    initialize the sets E:={i:e(i)>0} and D:={i:e(i)<0};
    while (E≠∅) do
        begin
        select a node s ∈ E and a node t ∈ D;
        determine shortest path distances d(j) from node s to all other
        nodes in G(x) with respect to the reduced costs $c_{ij}^{\pi}$;
        let P denote a shortest path from node s to node t;
        update $\pi$ = $\pi$ -d ;
        δ:= min[e(s),-e(t),min{$r_{ij}$:(i,j) ∈ P}];
        augment δ units of flow along path P;
        update x,G(x),E,D, and the reduced costs;
        end;
end;
```
Figure 2.2.3 (i) Successive shortest path algorithm

Initially, *E={1}* and *D={4}*. Therefore, in the first iteration, *s=1* and *t=4*. The shortest path distances *d* (with respect to reduced costs) are *d= (0, 2, 2, 3)* and the shortest path from node *1* to node *4* is *1-3-4*.

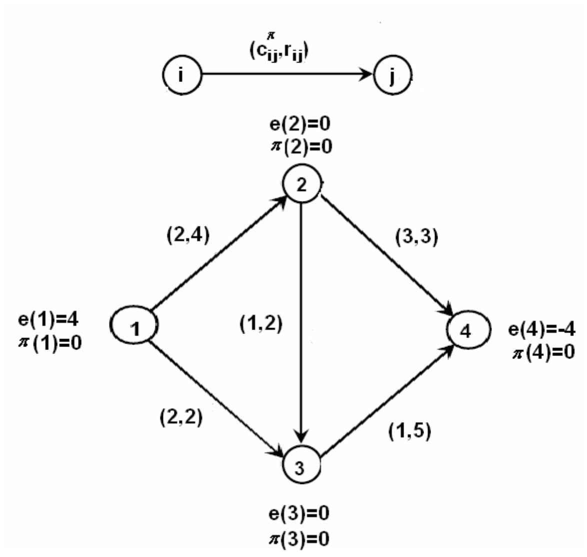


Figure 2.2.3 (ii) Initial residual network for x=0 and π=0

Figure 2.2.3 (iii) (a) shows the updated node potentials and reduced costs, and Figure 2.2.3 (iii) (b) shows the solution after the flow has been augmented min{e(1),-e(4),r_{13},r_{34}}=min{*4,4,2,5*}=*2* along path *1-3-4*.

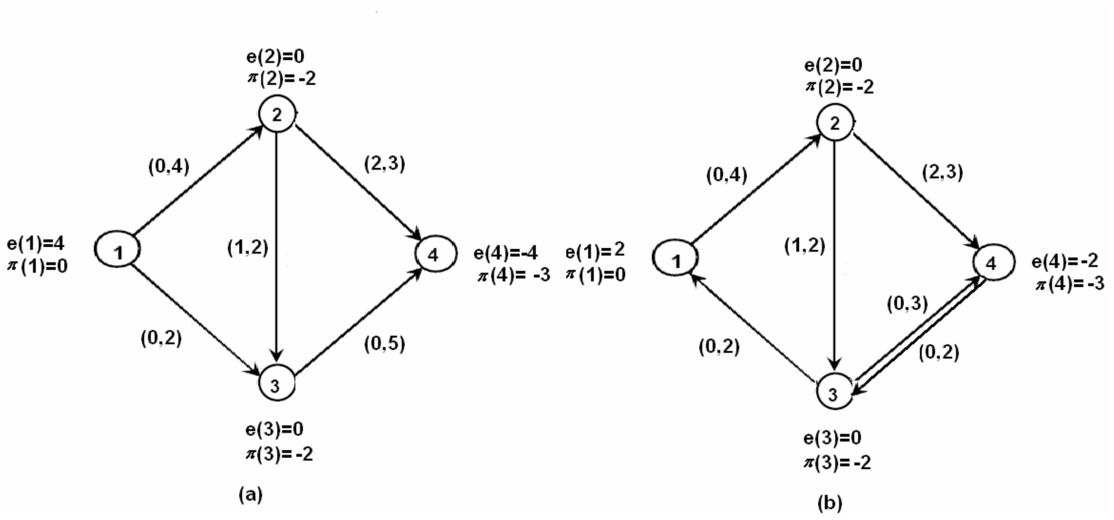


Figure 2.2.3 (iii) (a) Network after updating the potentials π
(b) Network after augmenting 2 units along path 1-3-4

In the second iteration, *s=1,t=4, d=(0,0,1,1)* and the shortest path from node *1* to node *4* is *1-2-3-4*. Figure 2.2.3 (iv) (a) shows the updated node potentials and reduced costs and Figure 2.2.3 (iv) (b) shows the solution after the flow has been augmented min{$e(1),-e(4),r_{12},r_{23},r_{34}$}=min{*2,2,4,2,3*}=*2* units of flow. At the end of the iteration all imbalances become zero and the algorithm terminates and a minimum cost flow from *s* to *t* is found.

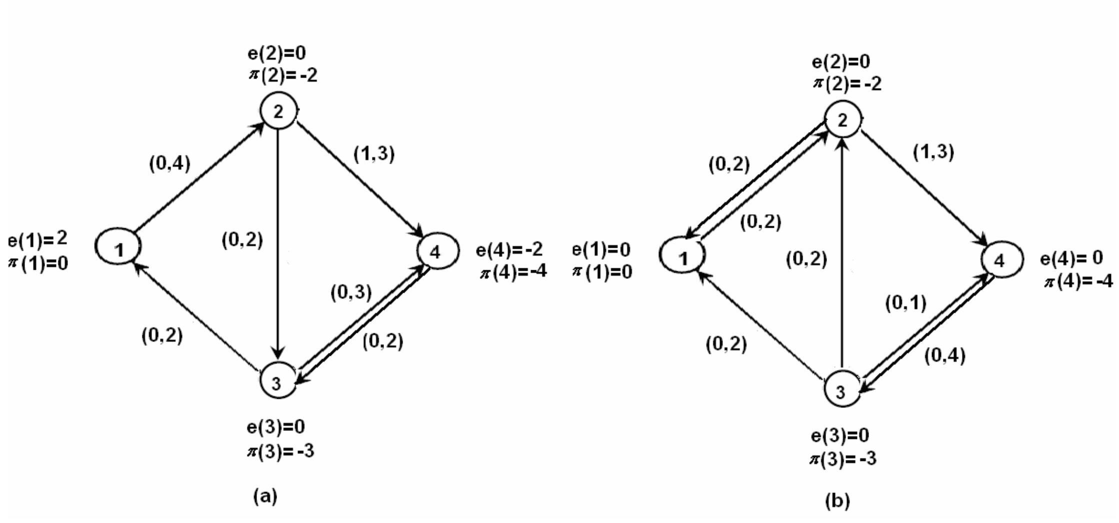


Figure 2.2.3 (iv) (a) Network after updating the potentials π
(b) Network after augmenting 2 units along path 1-2-3-4

2.3 Shortest path problem

The shortest path problem is used to solve the Minimum Project Duration application in Chapter 3.1.2 and the JIT problem in Chapter 3.1.3. The networks used for determining the Minimum Project Duration application contain no cycles due to the characteristics of these graphs. This will be discussed in Chapter 3.1.2. Due to this acyclic property of these graphs, the *shortest path* problems are very easy to solve with the algorithm described in paragraph 2.3.3.
The networks used to solve the JIT problem contain positive directed cycles. This will be discussed in Chapter 3.1.3. For this reason another algorithm has to be used to handle these types of networks: The *generic-labelling algorithm* is discussed in paragraph 2.3.4.

2.3.1 Definition

Consider a directed network *G=(N,A)* with an arc length (or arc cost) c_{ij} associated with each arc $(i,j) \in A$. The network has a source *s* and a sink *t*. The *length* of a directed path is defined as the sum of the lengths of arcs in the path. The *shortest path* problem is to determine a path of minimum length (or cost) from a specified source node *s* to another specified sink node *t*. Denote *d(i)* as the distance from the source node *s* to node *i* along the shortest path. Now the next theorem about the s*hortest path optimality conditions* is defined.

Theorem (Shortest path Optimality conditions)
For every node $j \in N$, let *d(j)* denote the length of some directed path from the source node to node *j*. Then the numbers *d(j)* represent shortest path distances if and only if they satisfy the following shortest path optimality conditions:

$$d(j) \le d(i) + c_{ij} \qquad \textbf{for all } (i,j) \in A$$

These optimality conditions are useful in several aspects. First, they give a simple validity check to see whether a given set of distance labels does indeed define shortest paths. Similarly, the optimality conditions provide a method for determining whether or not a given set of paths, one from node *s* to every other node in the network, constitutes a set of shortest paths from node *s*. Simply the lengths of these paths are computed and it is seen of these distances satisfy the optimality conditions. In both cases, the optimality conditions provide a “certificate’ of optimality, that is, an assurance that a set of distance labels or a set of paths is optimal. The optimality conditions are also valuable for other reasons; they can suggest algorithms for solving a shortest path problem. For example the *generic label algorithm* discussed in paragraph 2.3.4 uses the simple idea of repeatedly replacing *d(j)* by *d(j)* $+c_{ij}$ if d*(j)>d(i)+cij* for some arc *(i,j).*

2.3.2 Minimum cost problem formulation

The *shortest path* problem can also be seen as sending *1* unit of flow as cheaply as possible (with arc flow costs c_{ij}) from node *s* to node *t* in an uncapacitated network.
The shortest flow problem can then be formulated as minimum cost problem in the following way according to *Orlin*(1993):

- *Set b(s)=1, b(t)= -1* and *b(i)=0* for all other nodes in the minimum cost flow problem.

The solution to the problem will send *1* unit of flow from node *s* to node *t* along the shortest path. The shortest path formulation can be written as:

Minimize $\sum_{j:(i,j)\in A} c_{ij} x_{ij}$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & \text{for } i = s, \\ 0 & \text{for } i \neq s, t \\ -1 & \text{for } i = t, \end{cases}$$

$$x_{ij} \geq 0 \quad \text{for every arc } (i, j) \in A$$

The shortest path *dual* problem is a special form of the *dual* minimum cost flow problem. The shortest path problem contains no arc capacities, so the α_{ij} variables can be eliminated from the dual minimum cost problem defined in paragraph 2.2.2.
Denote *d(i)* as the distance from the source node *s* to node *i* along the shortest path.
Let $d(i) = -\pi(i)$ *and b(s)=1, b(t)*=-1 and *b(i)=0* for the other nodes, then according to *Orlin* (1993) the *dual minimum cost flow* problem becomes the *shortest path dual* problem. This dual can be written as:

Maximize $d(t) - d(s)$
subject to $d(j) - d(i) \leq c_{ij}$ **for all (i,j)** $\in$ **A**

2.3.3 Shortest path problems in acyclic networks

A network is acyclic if it contains no directed cycle. The networks used for the determination of the minimum project duration (Chapter 3.1.2) only deal with acyclic networks. This paragraph will show how to solve the shortest path problem on acyclic network even though the arc length might be negative.

Let us label the nodes of a network *G=(N,A)* by distinct numbers from *1* through *n* and represent the labelling by an array *order* (i.e. *order(i)* gives the label of node *i*). This labelling is a *topological ordering* of nodes if every arc

joins a lower-labelled node to a higher- labelled node. That is, for every arc (i,j) $\in A$, $order(i)<order(j)$.
For example, for the network shown in Figure 2.2.3(a), the labelling shown in Figure 2.3.3(b) is not a topological order because *(5,4)* is an arc and *order(5)>order(4)*, However, the labelling shown in Figure 2.2.3(c) is a topological ordering.

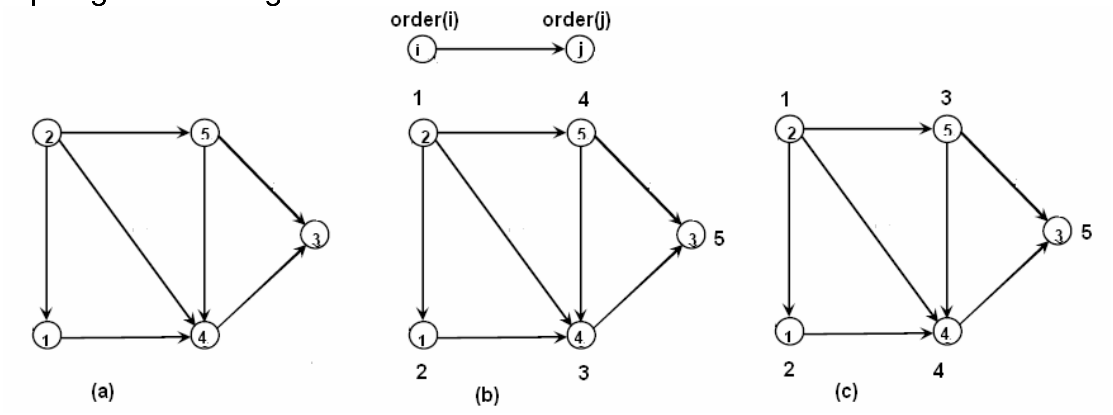


Figure 2.3 3 Topological ordering of nodes

Some networks cannot be topologically ordered. For example, the network shown in Figure 2.3.3(i) has no such ordering. This network is cyclic because it contains a directed cycle and for any directed cycle W, the condition *order(i)<order(j)* for each $(i,j) \in W$ can never be satisfied.

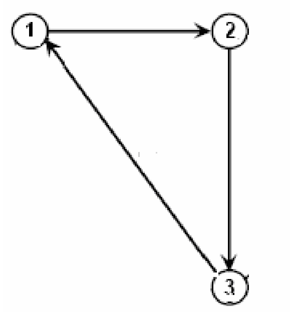


Figure 2.3 3 (i) Network without a topological ordering of the nodes

Acyclic networks and topological ordering are closely related. A network that contains a directed cycle has no topological ordering, and conversely, a network that possesses a topological order cannot contain a cycle. This observation shows that a network is acyclic if and only if it posses a topological ordering of its nodes.

The next algorithm is now used to solve the shortest path problem on acyclic networks, with *d(i),* the shortest path distance from the source to node i,(*Orlin 1993*):

1) Set $d(s)=0$ and the remaining distance labels to a very large number.
2) Examine the nodes in a topological order and for each node *i* being examined, the set of arcs emanating from node *i* are scanned.
3) If for any arc $(i,j)\in A$, there is found that $d(j)>d(i) + c_{ij}$, then set $d(j)=d(i)+c_{ij}$.

When the algorithm has examined all the nodes **once** in this order, the distance labels are optimal. This algorithms works even when the graph has negative lengths.

2.3.4 Shortest path problems in positive cyclic networks

For the types of problems discussed in this paper. only positive cycled network are encountered. A *positive* cycle is a directed cycle whose total weight is positive.
The algorithm used to solve the shortest path problem for the type of networks with positive cycles is similar to the one used for the acyclic network. The only difference is that the acyclic algorithm examines each node and arc exactly ones in contrast to the algorithm for the positive cyclic network.

The algorithm used for these types of network is called the ***generic label-correcting algorithm***.
This algorithm maintains a set of distance labels $d(.)$ at every stage. The label $d(j)$ is either ∞ , indicating that a directed path from the source to node j is yet to be discovered, or it is the length of some directed path from the source to node j. The algorithm is as follows:

1. Set $d(s)=0$ and the remaining distance labels to ∞
2. If for any arc $(i,j) \in A$, there is found that $d(j)>d(i) + c_{ij}$, then set $d(j)=d(i)+c_{ij}$.

The *generic label-correcting* algorithm is a general procedure for successively updating the distance labels until they satisfy the shortest path optimality conditions of $d(j) \leq d(i))+c_{ij}$ for all $(i,j) \in A$.

2.4 Longest Path Problem

The longest path and shortest path are closely related. The longest path problem can be transformed to a shortest path problem by defining arc costs equal to the negative of the arc length.
To solve the problem, multiply each arc length by -1 and then solve a shortest path problem. If the network is acyclic the corresponding shortest path problem is efficiently solved with the algorithm described in paragraph 2.3.3. If the longest path problem contains any positive length directed cycles, the resulting shortest path problem contains a negative cycle and it cannot be solved by any of the techniques discussed in this paper. However, if all directed cycles in the longest path problem, have nonpositive lengths, then in the corresponding shortest path problem all directed cycles have nonnegative lengths and this problem can be efficiently solved with the *generic label-correcting algorithm* described in paragraph 2.3.4.

2.5 Maximum Flow problem

2.5.1 Definition

The maximum flow problem is the problem of determining the maximum amount of flow *v* that can be sent in a given directed network G with arc capacities given by u_{ij}'s from a source node *s* to a sink node *t*.

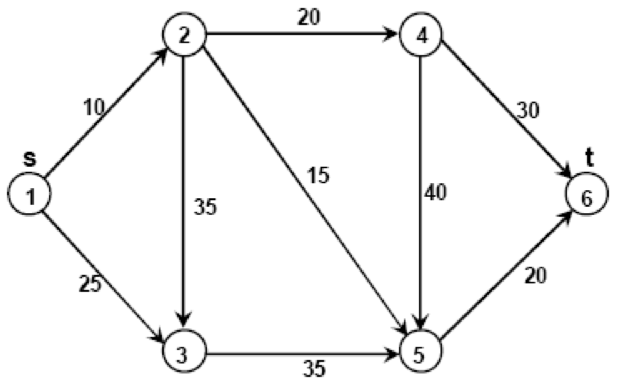


Figure 2.5.1 Maximum flow problem

The decision variables in the maximum flow problem are flow x_{ij} on arc (i, j), and the flow *v* entering the sink node.
The maximum flow problem can be formulated as the following linear programming problem:

Maximize v

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} v & \text{for } i = s, \\ 0 & \text{for } i \neq s, t \\ -v & \text{for } i = t, \end{cases}$$

$$0 \le x_{ij} \le u_{ij} \text{ for every arc } (i, j) \in A$$

2.5.2 Minimum cost problem formulation

The maximum flow problem can be formulated as *minimum cost* problem in the following way according to *Orlin*(1993):
Set $b(i)=0$ for all $i \in N$, $c_{ij}=0$ for all $(i,j) \in A$ in the minimum cost flow problem and introduce an additional arc *(t,s)* with cost $c_{ts}=-1$ and flow bound $u_{ts}=\infty$.
See Figure 2.5.2 as illustration.

Then the linear programming problem is written as:

$$\textbf{Minimize} \quad -x_{ts}$$

$$\textbf{subject to} \quad \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0$$

$$l_{ij} \le x_{ij} \le u_{ij} \text{ for every arc } (i, j) \in A$$

Now the minimum cost flow solution maximizes the flow on arc *(t,s).* But because any flow on arc *(t,s)* must travel from node *s* to node *t* through the arcs in *A* (because *b(i)=0*) , the solution to the minimum cost flow problem will maximize the flow from node *s* to node *t* in the original network.

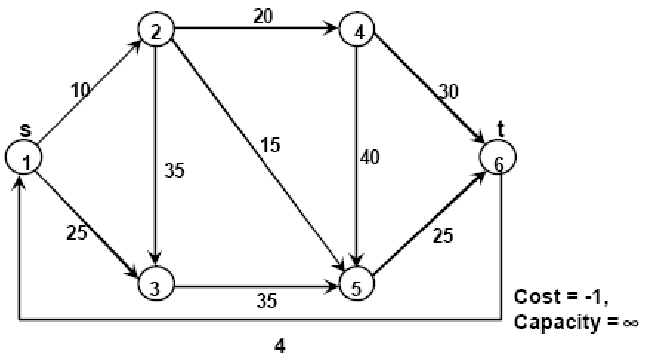


Figure 2.5.2 Maximum flow as minimum cost problem

2.5.3 Augmenting path algorithm

There exist several algorithms for finding a maximum flow in a network.
For example: the *labelling* algorithm, *Ford and Fulkerson* algorithm and *augmenting path* algorithm. Only one, the *augmenting path* algorithm, will here be discussed. For more information about these algorithms, *Even (1979*) and *Orlin (1993*) provide more insight.

With the *augmenting path* algorithm, which is one of the simplest algorithms to solve the maximum flow problem, a maximum flow is found. This algorithm searches for an *augmenting* path in the residual network. An *augmenting* path is a path from *s* to *t* in the residual network along which more flow can be pushed in the flow network.

The *augmenting path* algorithm is described in Figure 2.5.3 (i) and the maximum flow problem given in Figure 2.5.3 (ii) is used to illustrate the algorithm. This algorithm proceeds by identifying augmenting paths and augmenting flows on these paths until the network contains no such path.

```
algorithm augmenting path;
begin
x:=0;
while G contains  directed path from node s to node t do
begin
      identify an augmenting path P from node s to node t
       $\delta$:= min{$r_{ij}$:(i,j)$\in$ P};
      augment $\delta$ units of flow along P and update G(x)
      end;
end;
```
Figure 2.5.3 (i) Augmenting path algorithm

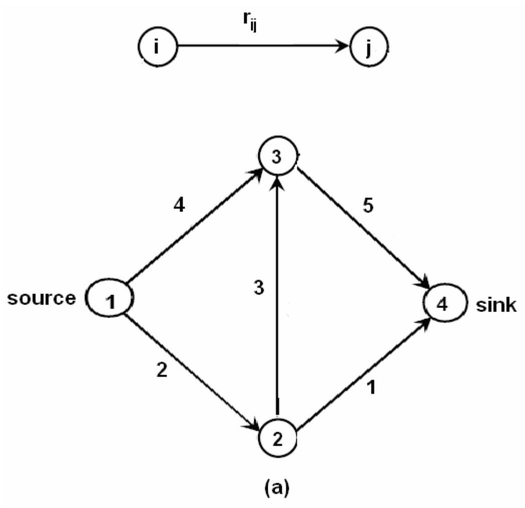


Figure 2.5.3 (ii) (a) Residual network for the zero flow

Suppose that the algorithms select the path *1-3-4* for augmentation. The residual capacity of this path is δ =min {r_{13},r_{34}}=min{*4,5*}=*4*. This augmentation reduces the residual capacity of arc *(1,3)* to *0* (thus it is deleted from the residual network) and increases the residual capacity of arc *(3,1)* to *4* (so this arc is added to the residual network). The augmentation also decreases the residual capacity of arc *(3,4)* from *5* to *1* and increases the residual capacity of arc *(4,3)* from *0* to *4*. Figure 2.5.3 (ii) (b) shows the residual network at this stage.

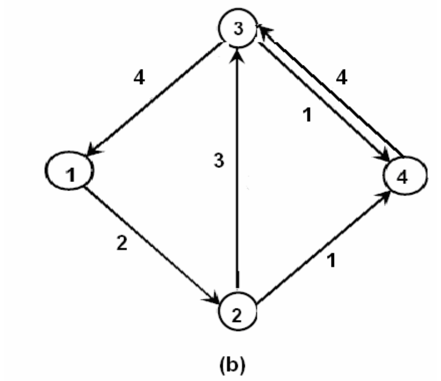


Figure 2.5.3 (ii) (b) Network after augmenting 4 units along path 1-3-4

In the second iteration, suppose that the algorithm selects the path *1-2-3-4*. The residual capacity of this path is δ=min{*2,3,1*}=*1*. Augmenting *1* unit of flow along this path yields the residual network shown in Figure 2.5.3 (ii) (c).

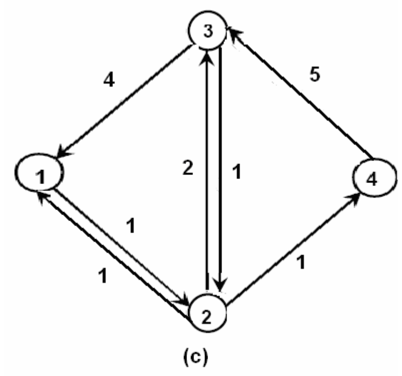


Figure 2.5.3 (ii) (c) Network after augmenting 1 unit along path 1-2-3-4

In the third iteration, the algorithm augments *1* unit of flow along the path *1-2-4*. Figure 2.5.3 (ii) (d) shows the corresponding residual network. Now the residual network contains no augmenting path, so the algorithm terminates.

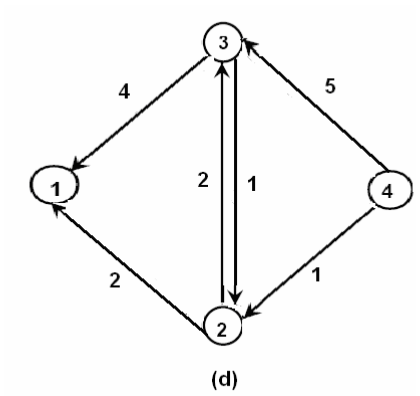


Figure 2.5.3 (ii) (d) Network after augmenting 1 unit along path 1-2-3-4

With the use of the next theorem, it is shown that the flow resulted from the *augmenting path* algorithm is a maximum flow.

Theorem (Augmenting Path Theorem)
A flow x is a maximum flow if and only if the residual network G(x*) contains no augmenting path.*

Proof: If the residual network G(x*) contains an augmenting path, clearly the flow x* is not a maximum flow. Conversely, if the residual network G(x*) contains no augmenting path, then the cut (B,P)'s capacity equals the maximum flow, thereby implying that the flow must be a maximum (*Ford & Fulkerson).* The set of nodes in B contains all the nodes that can be reached from the source *s* through an unsaturated path, and the set of nodes in P contains the nodes that cannot be reached from the source *s* through an unsaturated path.

2.6 Feasible Flow Problem

The *feasible flow* problem is formulated as follows:
Does there exist a flow in *G* that satisfies the capacity constraints?
The *feasible flow* problem requires the identification of a flow *x* in a network *G= (N, A*) with nodes *N*, edges *A*, capacity u_{ij} on edges *(i,j),*source *s* and a sink *t* that satisfies the following constraints:

- *Flow conservation:*

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \text{ for all } i \in N$$

As before it is assumed that $\sum_{i=1}^{n} b(i) = 0$

- *Capacity constraint*:

$$0 \le x_{ij} \le u_{ij} \text{ for every arc } (i, j) \in A.$$

This *feasible flow* problem can be solved by solving a *maximum flow* problem on a network as follows according to *Orlin (1993)*.

- Introduce two nodes, a source node *s* and a sink node *t*.
  For each node *i* with *b(i)>0*, add an arc *(s,i)* with capacity *b(i)* and for each node *i* with *b(i)<0*, add an arc *(i,t)* with capacity *–b(i).* This new network is called the *transformed* network. See Figure 2.6 as example.

- Solve a *maximum flow* problem from node *s* to node *t* in the transformed network.

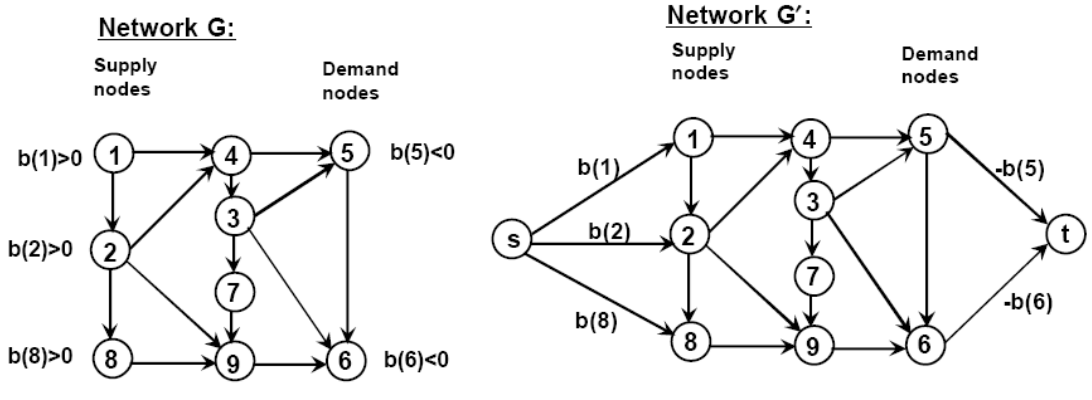


Figure 2.6 Feasible flow problem as maximum flow formulation

If the maximum flow saturates all the source and sink arcs then the feasible problem has a feasible solution, otherwise it is infeasible.

This conclusion can be made because if *x* is a flow satisfying the above constraints, the same flow with $x_{si}=b(i)$ for each source arc *(s,i)* and $x_{it}=-b(i)$ for each sink arc *(i,t)* is a maximum flow in the transformed network (since it saturates all the source and sink arcs).
Similarly, if *x* is a maximum flow in the transformed network that saturates all the source and sink arcs, this flow in the original network satisfies the above constraints. Therefore the original network contains a feasible flow if and only if the transformed network contains flow that saturates all the source and sink arcs. This observation shows how a maximum flow problem arises whenever a feasible solution in a network is needed to be found.

3 Project Management

3.1 Applications

The planning and scheduling of large projects is an important class of network problems, such as constructing a building or a highway, planning and launching a new product, installing and debugging a computer system, or developing and implementing a space exploration program. This application context was among the earliest successes of network optimization, and the network flow models of project management continue to be an important management tool used in numerous industries every day.

In this chapter, three basic models of project management are considered:

- Model for scheduling jobs on uniform parallel machines,
- Shortest path technique for scheduling projects to achieve the earliest possible completion,
- Network flow model for Just-in-Time scheduling of jobs in a project.

3.1.1 Scheduling on Uniform Parallel Machines

In this application the problem of scheduling a set J of jobs on *M* uniform parallel machines is considered according to the procedure as described in *Orlin* (1993). The scheduling problem is to determine a feasible schedule that completes all jobs before their due dates or to show that no such schedule exists.

Each job $j \in J$ has:

- a processing requirement p_j , denoting the number of machine days required to complete the job
- A release date r_j, representing the beginning of the day when job *j* becomes available for processing
- A due date $d_j \geq r_j + p_j$, representing the beginning of the day by which the job must be completed

The assumptions are that a machine can work on only one job at a time and that each job can be represented by at most one machine at a time. However *preemptions* are allowed. This means that a job can be interrupted and processed on different machines on different days.
For some problems a schedule is only feasible if preemptions are allowed. An example of a schedule that is only feasible when preeemptions are allowed is illustrated in Figure 3.1.1 (a).

Job (j)	1	2	3	4
Processing time (p_j)	1.5	3	4.5	5
Release time (r_j)	2	0	2	4
Due date (d_j)	5	4	7	9

Figure 3.1.1 (a) Scheduling Problem 1

As you can see from Table 1 no schedule is possible unless *preemption* is allowed. In Table 2 a feasible schedule is shown when preemption is allowed.

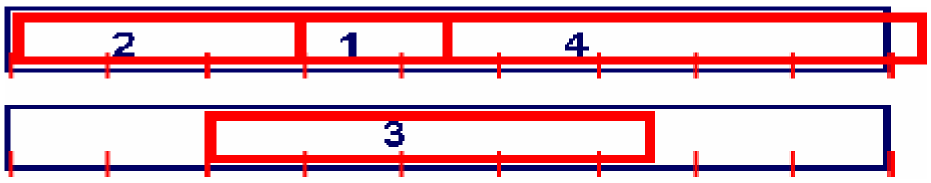


Table 1 No pre-emption & No feasible schedule

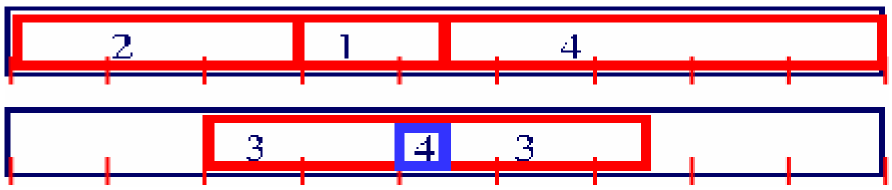


Table 2 Preemption & Feasible Schedule

The feasible scheduling problem, described in paragraph 2.6, is a fundamental problem in this situation. The feasible scheduling problem will now be formulated as a maximum flow problem.
The formulation will be illustrated using Scheduling Problem 2 described in Figure 3.1.1 (b) with *M=2* machines.

Job (j)	1	2	3	4
Processing time (p_j)	1.5	1.25	2.1	3.6
Release time (r_j)	3	1	3	5
Due date (d_j)	5	4	7	9

Figure 3.1.1 (b) Scheduling Problem 2

This scheduling problem is solved as follows according to *Orlin* (1993):

- Rank all release and due dates, r_j and d_j for all j , in ascending order.
- Determine $P \leq 2|J|-1$ mutually disjoint intervals of dates between consecutive milestones. Let $T_{k,l}$ denote the interval that starts at the beginning of date *k* and ends at the beginning of date *l+1*.

For the example of Figure 3.1.1 (b), the order of release and due dates is *1, 3, 4 ,5,7,9.*
There are five intervals, represented by $T_{1,2}$, $T_{3,3}$, $T_{4,4}$, $T_{5,6}$, $T_{7,8}$.
Notice that within each interval, the set of available jobs (i.e. those released but not yet due) does not change: all jobs j with $r_j \le k$ and $d_j \ge l+1$ can be processed in the interval.

The scheduling problem can be formulated as a maximum flow problem on a bipartite network G as follows according to *Orlin* (1993):

- Introduce a source node *s*, a sink node *t*, a node corresponding to each job *j* , and a node corresponding to each interval $T_{k,l}$, as shown in Figure 3.1.1 (c) .
- The source node is connected to every job node j with an arc with capacity p_j, indicating that you need to assign p_j days of machine time to job *j* .
- Each interval node $T_{k,l}$ is connected to the sink node *t* by an arc with capacity *(l-k+1)M*, representing the total number of machine days available on the days from *k* to *l*.
- Finally connect a job node j to every interval node $T_{k,l}$ if $r_j \le k$ and $d_j \ge l+1$ by an arc with capacity *(l-k+1)* which represents the maximum number of machines day you can assign to job *j* on the days from *k* to *l*.

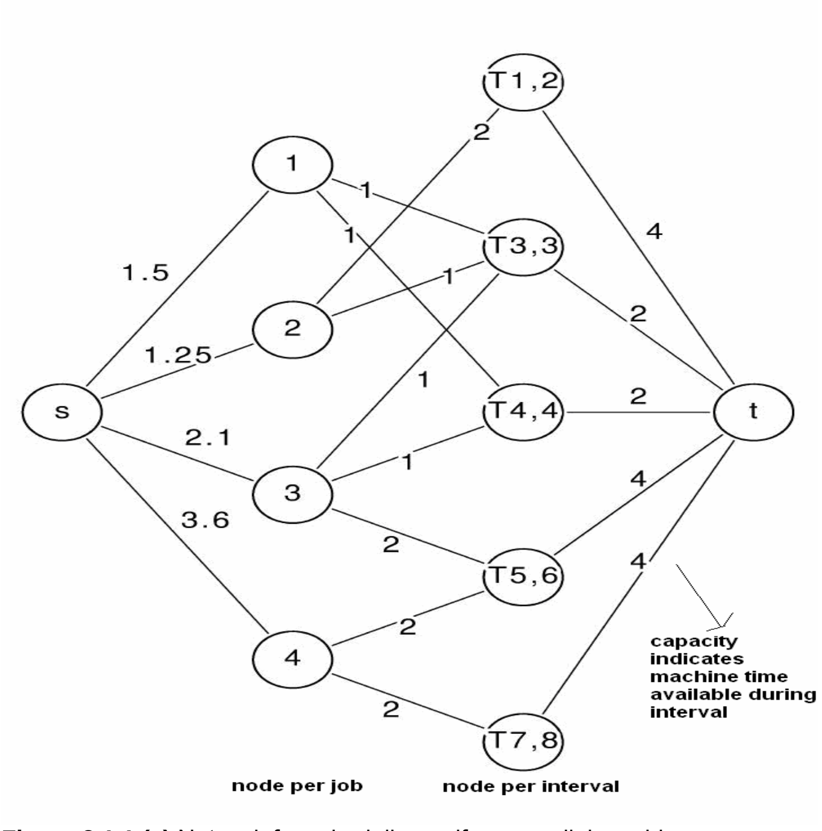


Figure 3.1.1 (c) Network for scheduling uniform parallel machines

Now a maximum flow problem is solved on this network: the scheduling problem has a feasible schedule if and only if the maximum flow equals $\sum_{j \in J} p_j$ (alternatively, the flow on every arc *(s,j)* is p_j).

The validity of this formulation is easy to establish by showing a 1-1 correspondence between feasible schedules and flows of value $\sum_{j \in J} p_j$ from the source to the sink; only if every arc *(s,j)* is p_j (so all the arcs *(s,j)* are saturated) a feasible schedule exist. This because if one of these arcs is not saturated, it means that the job corresponding to that arc cannot be completed.

Figure 3.1.1 (d) illustrates the formulation of a scheduling problem on uniform parallel machines. It is obvious that the arc (s, 1) will not be saturated in any maximum flow (due to *mass balance* constraint). Hence a feasible schedule does not exist.

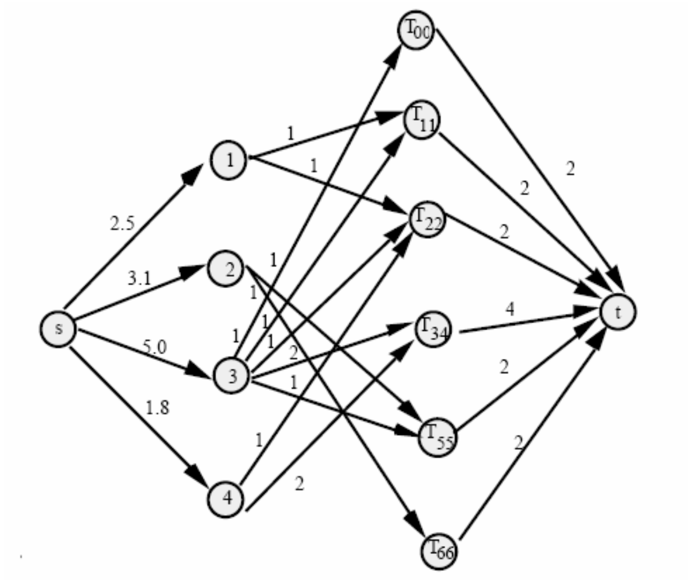


Figure 3.1.1 (d) Network with no feasible schedule

Because these are acyclic networks, the augmenting path algorithm (see paragraph 2.5.3) is used to find a maximum flow. For the Scheduling Problem 2, the maximum flow of 1.25+2+1.5+2.1+1.6=8.45 is found with the augmenting path algorithm. See Figure 3.1.1 (e).

Because this maximum flow equals $\sum_{j \in J} p_j$, there is a feasible schedule.

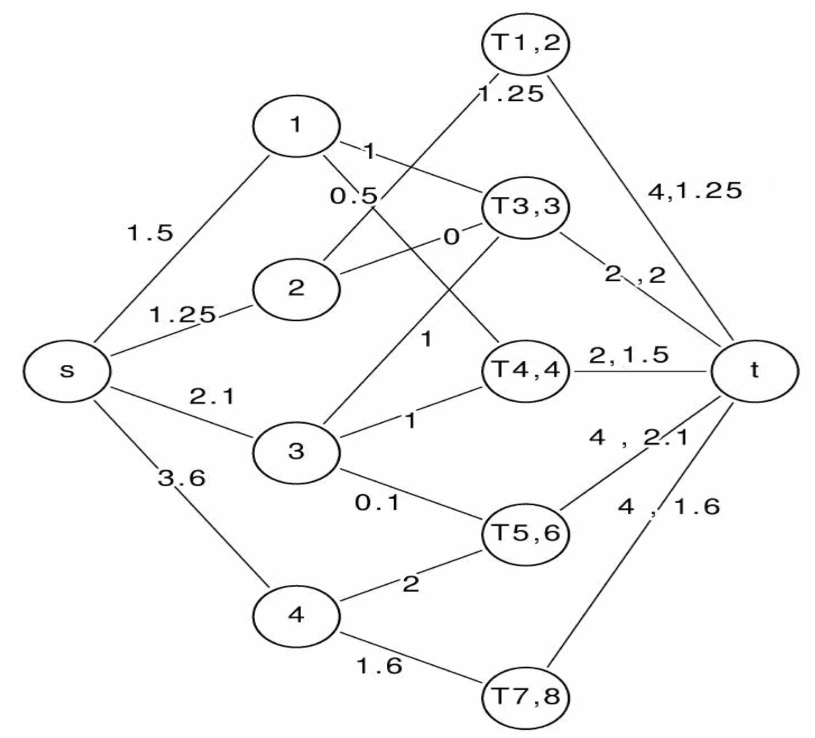


Figure 3.1.1 (e) Network with maximum flow

Flow decomposition can now be used to transform flows into schedules.
The feasible schedule can be interpreted as follows:
For job 1: assign 1 unit of machine time in period $T_{3,3}$ and 0.5 unit of machine time in period $T_{4,4}$.
For job 2: assign 1.25 unit of machine time in period $T_{1,2}$.
For job 3: assign 1 unit of machine time in period $T_{3,3}$, 1 unit of machine time in period $T_{4,4}$ and 0.1 unit of machine time in period $T_{5,6}$.
For job 4: assign 2 unit of machine time in period $T_{5,6}$ and 1.6 unit of machine time in period $T_{7,8}$.
A feasible schedule is shown in Table 4.

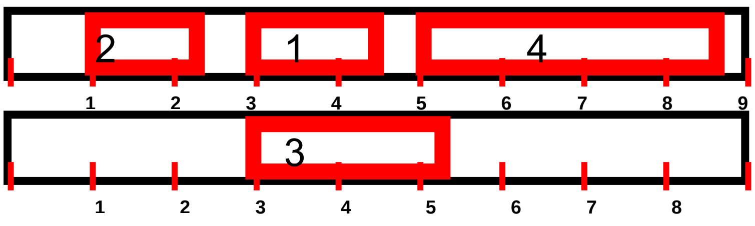


Table 4 A feasible schedule for Scheduling Problem 2

3.1.2 Determining Minimum Project Duration

In this application a shortest path technique for scheduling projects to achieve the earliest possible completion time will here be discussed according to the procedure as described in *Orlin* (1993).

A project is viewed as a collection of jobs and a set *A* of precedence relations between the jobs. A job is represented by a node and the precedence relationship between two activities is represented by a directed arc in which the direction of the arrow specifies the precedence. A characteristic of this network graph:

- By construction, the network contains no cycles, since otherwise by tracing around the cycle it would be concluded, by the transitivity property of precedence, that a task must precede itself, which is an impossibility.

If $(i,j) \in A$, job *i* is to be completed before beginning job *j*. Each job *j* has also a known duration d_j. The problem is to determine a project schedule (i.e., the start time of each job) that will satisfy the precedence relations between the jobs and complete the project in the least possible amount of time. This gives the least possible project duration.

Consider, for example, the project planning problem given in Table 3.1.2. According to *Orlin* (1993), this project planning problem can be formulated as a shortest path problem where the jobs are represented by nodes as follows:

- Define a project network by associating a node *j* with each job *j* and by including arc *(i,j)* whenever job *i* is an immediate predecessor of job *j.*
- Set the length c_{ij} of arc *(i,j)* equal to d_i, the duration of job *i*.
- Introduce a source node *s*, the beginning of the project and connect it to every other node that has no incoming arc (So the jobs without predecessors) by zero-length arcs.
- Similarly introduce a sink node *t*, the end of the project, and connect every node *i* with no outgoing arc to this sink node by an arc *(i,t)* whose length equals the duration of job *i*.

Job	Duration	Immediate predecessors
a	14	---
b	3	---
c	3	a,b
d	7	a
e	4	d
f	10	c,e

Table 3.1.2 Project planning problem

Figure 3.1.2 (a) gives the network corresponding to the project planning example shown in Table 3.1.2.

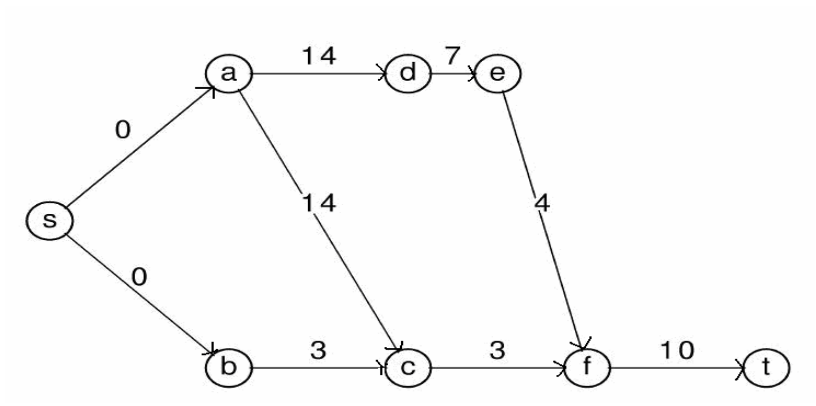


Figure 3.1.2 (a) Shortest path formulation of the project planning problem

Note that the network corresponding to any project planning model must be acyclic because a network containing a cycle could never be completed, since otherwise by tracing around the cycle it would be concluded that a task must precede itself, which is an impossibility.

Let *u(j)* denote the earliest possible starting time of job *j* in a project planning schedule that satisfies the precedence constraints. Notice that with respect to quantities *u(j),* the project duration is *u(t)-u(s)*.
The project planning problem can be stated as the following optimization model:

$$\textbf{Minimize} \quad u(t)-u(s)$$

$$\textbf{subject to} \quad u(j)-u(i) \geq c_{ij} \text{ for all } (i,j) \in A$$

The constraint models the precedence constraint by stating that if job *i* is an immediate predecessor of job *j*, then job *j* can start only after $c_{ij}=u_i$ units of time have elapsed since the start of job *i*. This optimization model has for the attentive reader already been mentioned in this paper. It is the dual form of the shortest path problem.

To bring this problem into a more familiar network flow form, the dual is taken (so the primal is obtained).
If the variables x_{ij} are associated with the constraint, the linear program is:

$$\textbf{Maximize} \sum_{\{j:(i,j)\in A\}} c_{ij} x_{ij}$$

subject to

$$\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = \begin{cases} 1 & \text{for } i = s, \\ 0 & \text{for } i \neq s, t \\ -1 & \text{for } i = t, \end{cases}$$

$$x_{ij} \geq 0 \quad \text{for every arc } (i, j) \in A$$

Clearly, this is a longest path problem with c_{ij} as the length of arc *(i,j):* the

objective is to send *1* unit of flow from node *s* to node *t* along the longest path. To solve the problem, multiply each arc length by -1 and then solve a shortest path problem. This is possible because the network is acyclic so no negative cycles are created.
For the project planning example shown in Table 3.1.2, the network becomes as shown in Figure 3.1.2(b) and now a shortest path problem is solved with the algorithm described in Chapter 2.3.3.

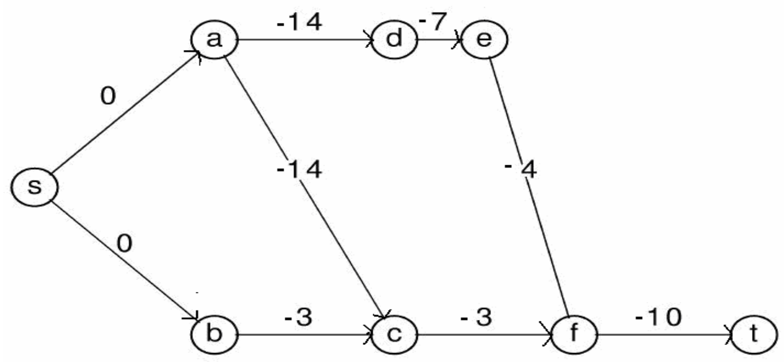


Figure 3.1.2 (b) Shortest path formulation of the project planning problem

First the nodes are labelled in a topological ordered. Recall from Chapter 2.3.3 that nodes are in a topological order if *$i<j$* for every *$(i,j)\in A$*.
An example of a topological order is shown in Figure 3.1.2 (c) where the order number of each node is shown above the node for the upper part of the graph, and under the node for the lower part of the graph.

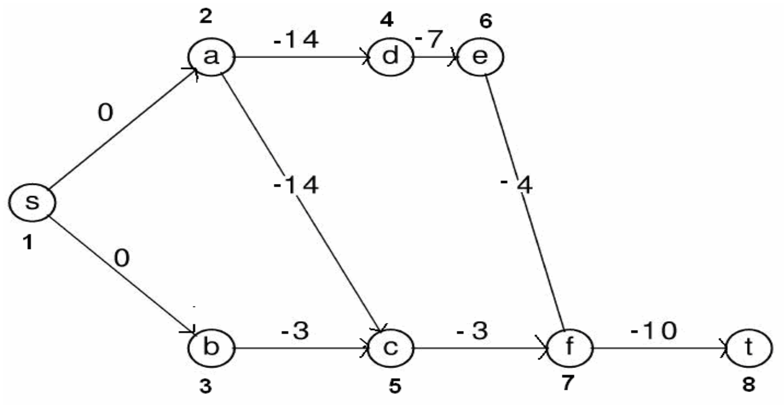


Figure 3.1.2 (c) Project planning problem with nodes labelled in a topological ordered

Now set *$d(s)=0$* and the remaining distance labels to ∞.
Then nodes are now examined in a topological order and for each node *i* being examined, the arcs emanating from node *i* are scanned.
If for any arc *$(i,j)\in A$*, there is found that *$d(j)>d(i)+c_{ij}$*, then set *$d(j)=d(i)+c_{ij}$*
After applying this algorithm to the project planning problem, Figure 3.1.2 (d) is obtained and the optimal solution is found of 35. So the minimum project duration is 35.

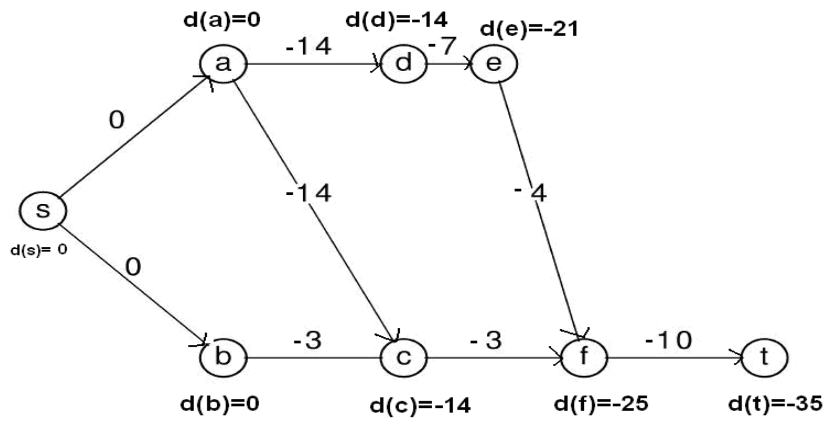


Figure 3.1.2 (d) Project planning problem with shortest path at every node.

3.1.3 Just in Time Scheduling

The just-in-time scheduling is an extension of the project planning problem discussed in the former paragraph. In the JIT problem, the minimum project duration is determined subject to both the precedence constraints and some additional "just-in-time constraints".
In this problem a subset $S \subseteq A$ and a number α_{ij} for each $(i,j) \in S$ is given. The just in time constraints state that for each $(i,j) \in S$, job *j* must start within α_{ij} units of time from the start of job *i*.

Denote with *u(i)* the earliest start times of job *i*, then the just-in-time constraints require that:

$$u(j) \le u(i) + \alpha_{ij} \qquad \textbf{for all (i,j)} \in \textbf{S}$$

or, equivalently,

$$u(i) - u(j) \ge -\alpha_{ij} \qquad \textbf{for all (i,j)} \in \textbf{S}$$

The start times must also satisfy the usual precedence constraints:

$$u(j) - u(i) \ge c_{ij} \qquad \textbf{for all (i,j)} \in \textbf{A}$$

In the just-in-time scheduling problem, the objective is:

$$\textbf{Minimize} \qquad \textbf{u(t)-u(s)}$$

$$\textbf{subject to} \quad u(i) - u(j) \ge -\alpha_{ij} \qquad \textbf{for all (i,j)} \in \textbf{S}$$

$$u(j) - u(i) \ge c_{ij} \qquad \textbf{for all (i,j)} \in \textbf{A}$$

Like in the former application problem for determining the minimum project duration, this can be solved as a longest path problem , in this case on an network *G'=(N,A')* whose arc set *A'* includes an arc *(i,j)* of cost c_{ij} for each *(i,j)* $\in$ *A* and an arc *(j,i)* of cost $-\alpha_{ij}$ for each *(i,j)* $\in$ *S* according to *Orlin (1993*).

Job	Duration	Immediate predecessors	JIT constraints
a	14	---	---
b	3	---	---
c	3	a,b	----
d	7	a	14
e	4	d	7
f	10	c,e	---

Table 3.1.3 Project planning problem with JIT constraints

To transform this longest path problem into a shortest path problem, multiply each arc cost by -1.For the project planning example shown in Table 3.1.3, the network becomes as shown in Figure 3.1.3 (a).

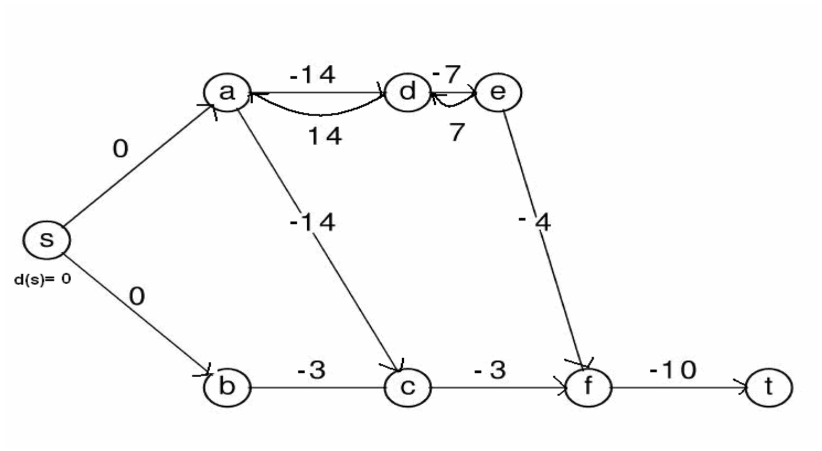


Figure 3.1.3 (a) Project planning problem

Notice that in this case the augmented network G' might not be acyclic and the resulting *shortest path* problem might contain a negative cycle (so a positive cycle in the longest path problem). The presence of a negative cycle in the shortest path problem (or positive cycle in longest path problem) indicates that the JIT problem has no feasible solution. This is explained as follows.
The just-in-time constraints require that:

$$u(i) - u(j) \geq -\alpha_{ij} \quad \textbf{for all (i,j)} \in \textbf{S}$$
$$u(j) - u(i) \geq c_{ij} \quad \textbf{for all (i,j)} \in \textbf{A}$$

This can be seen as that in the longest path problem:
$c_{ij} - \alpha_{ij} \leq 0$ **for all (i,j) $\in$ S**
This can be seen in the shortest path problem as:
$\alpha_{ij} - c_{ij} \geq 0$ **for all (i,j) $\in$ S**

If there would be a positive cycle in the longest path problem, then $c_{ij} - \alpha_{ij} \geq 0$ so this would be in contradiction with the just-in-time constraint that $c_{ij} - \alpha_{ij} \leq 0$ and so there would not be a feasible solution.

If there would be a negative cycle in the shortest path problem, then $\alpha_{ij} - c_{ij} \leq 0$ so this would be in contradiction with the just-in-time constraint for the shortest path problem that $\alpha_{ij} - c_{ij} \geq 0$ and so there would not be a feasible solution.

When the resulting shortest path problem has no negative cycle, the negative of the shortest path distances provide optimal start time for the jobs. For graph shown in Figure 3.1.3 (a), it is seen that the network contains no negative cycles. So the *generic label-correcting* algorithm can be used to solve this problem. By applying this algorithm the minimum duration of 35 is obtained. In Figure 3.1.3 (b) the graph is shown with at every node the shortest path distances.

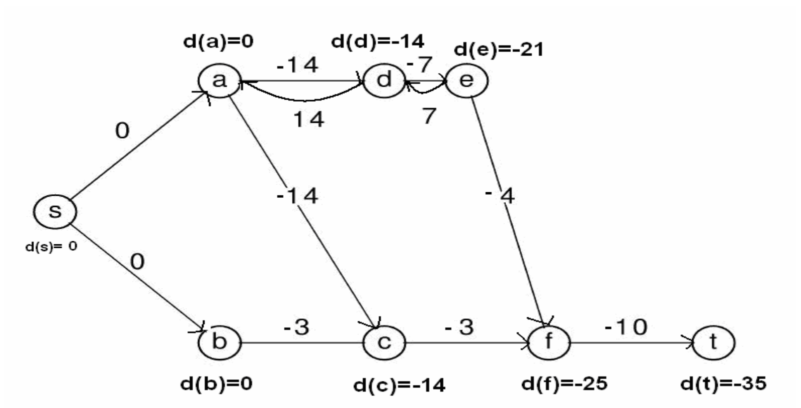


Figure 3.1.3 (b) Project planning problem with shortest path at every node

Suppose now that instead of imposing an upper bound on when job *j* should start after the start of job i, the time difference between the completion of job *i* and the start of job *j* is penalized using a penalty factor of d_{ij}.
The objective is to determine the start times of jobs that will minimize this penalty and yet satisfy the restriction that the project duration is at most a specified constant *λ.*
The following linear program models this problem:

$$\textbf{Minimize} \qquad \sum_{(i,j)\in A} (u(j) - u(i) - c_{ij}) d_{ij}$$

$$\textbf{subject to} \qquad -u(t) + u(s) \ge -\lambda \text{ ,}$$
$$u(j) - u(i) \ge c_{ij} \qquad \textbf{for all (i,j)} \in \textbf{A}$$
$$u(j) \qquad \textbf{unrestricted for all j} \in \textbf{N}$$

Let $D_i = \sum_{\{j:(j,i)\in A} d_{ji} - \sum_{\{j:(i,j)\in A} d_{ij}$

The dual of this model according to *Orlin* (1993) is:

$$\textbf{Maximize} \qquad \sum_{\{j:(i,j)\in A\}} c_{ij} x_{ij} - \lambda x_{ts}$$

$$\textbf{subject to} \qquad \sum_{\{j:(j,s)\in A} x_{js} - \sum_{\{j:(s,j)\in A} x_{sj} + x_{ts} = D_s \text{ ,}$$
$$\sum_{\{j:(j,i)\in A} x_{ji} - \sum_{\{j:(i,j)\in A} x_{ij} = D_i \quad \textbf{for all } \boldsymbol{i} \ne \ \boldsymbol{s} \textbf{ or } \boldsymbol{t}$$

$$\sum_{\{j:(j,t)\in A} x_{jt} - \sum_{\{j:(t,j)\in A} x_{tj} - x_{ts} = D_t \text{ ,}$$

$$x_{ij} \ge 0 \qquad \textbf{for all } \boldsymbol{(i,j)} \in \boldsymbol{A}$$

This problem is a minimum cost network flow problem with an arc *(t,s)* from the end node *t* to the start node s and can be solved using the Successive Shortest path algorithm.

3.2 Relevant literature

There is considerable amount of literature concerning Job Management and Network flows that can be found in technical libraries or on the Internet. Among the available literature, the best available resource is without doubt: *Network Flows* by Orlin, Ahuja and Magnanti. This book has been the winner of the 1993 Lanchester Prize for the best English publication in Operations Research. This book is rich in theory, as well as in algorithms and applications dealing with all the topics addressed in this paper. Orlin's book appears as the most complete and relevant publication, it is also the most referenced book in the literature. Other publications of interest that can be mentioned for more additional information are respectively for:

" Scheduling on Uniform parallel machines"

- *Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Techniques* by Groenevelt.H, Federgruen A. Management Science (1986)

" Determining minimum project duration"

- *Activity Networks:Project Planning and Control by Network Models* by Elmaghraby S. (1977)

" Just In Time Scheduling"

- *Activity Networks:Project Planning and Control by Network Models* by Elmaghraby S. (1977)
- *A network flow algorithm for Just-In-Time scheduling* by Levner E.V., Nemirovsky, A.S

4 Abstract

This paper provides a basis for solving project management problems by means of network flow techniques. At first, some network flow problems are discussed which will be used later to solve three project management problems.

The three project management problems that are here discussed are:

1. How to schedule jobs on uniform parallel machine such that there is a feasible schedule (all the jobs finish on time).
2. How to determine the minimum project duration when there are precedence relations between the jobs.
3. Just-in-Time scheduling problem. This is an extension to the problem of how to determine the minimum project duration with some extra constraints added

The first project management problem will be solved using a network flow technique called the maximum flow problem. Before being able to do this, the feasible scheduling problem will have to be formulated as a maximum flow problem. Thereafter a maximum flow problem is solved on the network. It will show that the scheduling problem has a feasible schedule if and only if the maximum flow equals the sum of the flows on every arc emanating from the source.

The second project management problem will be solved using a technique called, the shortest path problem. First the project planning problem is formulated as a shortest path problem and hereafter it is solved using this technique.

The third project management problem will be also solved using the shortest path problem. In this problem, the minimum project duration is determined subject to both the precedence constraints and some additional “just-in-time constraints”. An extended version of this problem is also considered. This version also includes a penalty for the time difference between the completion of a job and the start of another job. This problem is solved according to the minimum cost flow technique.

References

Combinatoriek, grafentheorie en getaltheorie (deel 2);Dictaat Open Universiteit
Elmaghraby S. (1977);*Activity Networks: Project Planning and Control by Network Models*
Even, S (1979*); Graph Algorithms*; Pitman publishing
Frank H., Frisch I. (1971); *Communication, Transmission and Transportation Networks*; Addison-Wesley Publishing
Groenevelt H., Federgruen A. (Mar., 1986); *Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Techniques;* Management Science, Vol. 32, No.3, pp.341-349
Orlin J.B., Ahuja R.K., Magnanti T.L. (1993); *Network Flows: Theory, Algorithms and Applications*; Prentice Hall