where SS_{ij}^{min} represents a minimum time-lag between the start time of activity *i* and the start time of activity *j* (similar definitions apply for SS_{ij}^{max}, FS_{ij}^{min}, ...), s_i denotes the start time and f_i denotes the finish time of activity *i*.

2.1 *GPR*s in standardised form

The *GPR*s can be represented in *standardised form* by transforming them to, for instance, minimal start-start precedence relations, using the following transformation rules (Bartusch et al. (1988)):

$$s_i + SS_{ij}^{min} \le s_j \Rightarrow s_i + l_{ij} \le s_j \quad \text{with } l_{ij} = SS_{ij}^{min}$$

$$s_i + SS_{ij}^{max} \ge s_j \Rightarrow s_j + l_{ji} \le s_i \quad \text{with } l_{ji} = -SS_{ij}^{max}$$

$$s_i + SF_{ij}^{min} \le f_j \Rightarrow s_i + l_{ij} \le s_j \quad \text{with } l_{ij} = SF_{ij}^{min} - d_j$$

$$s_i + SF_{ij}^{max} \ge f_j \Rightarrow s_j + l_{ji} \le s_i \quad \text{with } l_{ji} = d_j - SF_{ij}^{max}$$

$$f_i + FS_{ij}^{min} \le s_j \Rightarrow s_i + l_{ij} \le s_j \quad \text{with } l_{ij} = d_i + FS_{ij}^{min}$$

$$f_i + FS_{ij}^{max} \ge s_j \Rightarrow s_j + l_{ji} \le s_i \quad \text{with } l_{ji} = -d_i - FS_{ij}^{max}$$

$$f_i + FF_{ij}^{min} \le f_j \Rightarrow s_i + l_{ij} \le s_j \quad \text{with } l_{ij} = d_i - d_j + FF_{ij}^{min}$$

$$f_i + FF_{ij}^{max} \ge f_j \Rightarrow s_j + l_{ji} \le s_i \quad \text{with } l_{ji} = d_j - d_i - FF_{ij}^{max}.$$

The interval $[s_i + l_{ij}, s_i - l_{ji}]$ is called the *time window* of s_j relative to s_i (Bartusch et al. (1988)). Let us borrow from De Reyck (1998) the example activity network given in Figure 64. The numbers above the nodes (activities) denote the activity durations d_i. The labels associated with the arcs indicate the *GPR*s. The bold edges indicate the critical paths (cf. infra). Applying the transformations to this network example results in the activity network in standardised form given in Figure 65. Now the labels associated with the arcs indicate the time-lags l_{ij}. If there is more than one time-lag l_{ij} between two activities *i* and *j*, only the maximal time-lag is retained. The resulting network is called the *constraint digraph,* which is short for *digraph of temporal constraints* (Bartusch et al. (1988)).

It should be noted that activity networks with *GPR*s may contain cycles. A path $\langle i_s, i_k, i_l, \ldots, i_t \rangle$ is called a *cycle* if $s = t$. 'Path' refers to a *directed* path; 'cycle' refers to a *directed* cycle. The *length* of a path (cycle) in a standardised project network is defined as the sum of all the lags associated with the arcs belonging to that path (cycle). To ensure that the dummy start and finish activities correspond to the start and the completion of the project, we assume that there exists at least one path with nonnegative length from

node 1 to every other node and at least one path from every node i to node n which is equal to or larger than d_i. If there are no such paths, we can insert arcs $(1,i)$ or (i,n) with weight zero or d_i respectively.

$P_i=\{k|(k,i)\in E\}$ is the set of *immediate predecessors* of node i in graph $G=(V,E)$, $S_i=\{k|(i,k)\in E\}$ is the set of all its immediate successors. If there exists a path from i to j, then we call i a *predecessor* of j and j a *successor* of i. $P^*(i)$ and $S^*(i)$ denote the set of (not necessarily immediate) predecessors and successors of node i respectively. If the length of the longest path from i to j is nonnegative, i is called a *real predecessor* of j, and j is called a *real successor* of i. Otherwise it is a *fictitious* one.

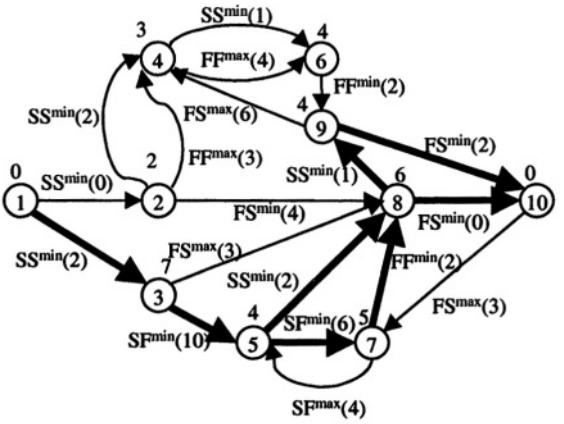

Figure 64. An activity network with GPRs (De Reyck (1998))

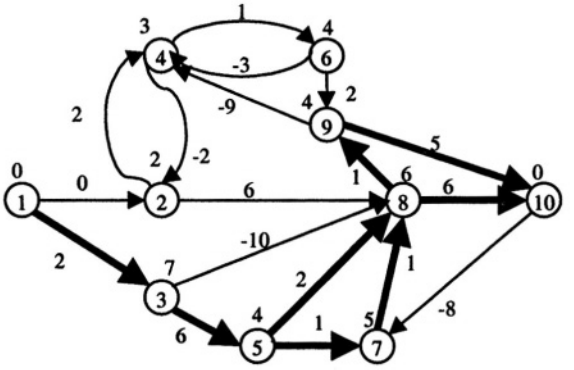

Figure 65. The constraint digraph (De Reyck (1998))

2.2 Computing the earliest start schedule

A schedule $S = (s_1, s_2, \ldots, s_n)$ is called *time-feasible,* if the activity start times satisfy the following conditions:

$$s_i \geq 0 \qquad \forall i \in V \qquad [4.24]$$

$$s_i + l_{ij} \leq s_j \qquad \forall (i,j) \in E \qquad [4.25]$$

where Eqs. [4.24] ensure that no activity starts before the current time (time zero) and Eqs. [4.25] denote the *GPR*s in standardised form.

The resource-unconstrained project scheduling problem with GPRs under the minimum makespan objective (classified as problem $gpr|C_{max}$ following the conventions of Chapter 3) can be mathematically formulated as follows:

$$\min \quad s_n \qquad [4.26]$$

subject to

$$s_i + l_{ij} \leq s_j \qquad \forall (i,j) \in E \qquad [4.27]$$

$$s_i \in \mathbb{N} \qquad i \in V \qquad [4.28]$$

The objective is to minimise the project duration (makespan). As shown in Eq. [4.26], this corresponds to minimising the completion time (or start time, since d_n=0) of the dummy end activity *n.* Constraints [4.27] represent the *GPR*s. Constraints [4.28] ensure that all activity start times assume nonnegative integer values. Solving problem $gpr|C_{max}$ can be accomplished by computing a *time-feasible earliest start schedule* (*ESS*), i.e. the minimum start times $(es_1, es_2, \ldots, es_n)$ satisfying both Eqs. [4.27] and [4.28]. For the example of Figure 65, *ESS* = (0, 0, 2, 2, 8, 3, 9, 10, 11, 16).

The *earliest start* of an activity *i* can be calculated by finding the longest path from node 1 to node *i*. The calculation of an earliest start schedule (*ESS*) can be related to the test for existence of a time-feasible schedule. A time-feasible schedule for *G* exists iff *G* has no cycle of positive length (Bartusch et al. (1988)). Such cycles would unable us to compute start times for the activities which satisfy conditions [4.27] and [4.28]. Therefore, if we compute the *longest path matrix* $\Pi = [\pi_{ij}]$, where π_{ij} denotes the longest path length from node *i* to node *j*, a positive path length from node *i* to itself indicates the existence of a cycle of positive length, and consequently, the non-existence of a time-feasible schedule. In the literature (Bartusch et al. (1988)), the matrix Π is often referred to as the *distance matrix.*

The computation of the matrix Π can be performed by standard graph algorithms for computing longest paths in networks. The Floyd-Warshall algorithm, for example, is of time complexity $O(n^3)$ (see e.g. Lawler (1976)). The possible existence of cycles should be taken into account. Let $\pi_{ij}^{(k)}$

represent the length of a longest path from node i to node j subject to the condition that this path uses only the nodes 1, 2, ..., k-1 as internal nodes. Clearly, $\pi_{ij}^{(n+1)}$ represents the actual longest path distance from node i to node j. If we start with the matrix $\Pi^1 = [\pi_{ij}^{(1)}](i, j = 1,2,...,n)$ with

$$\pi_{ij}^{(1)} = \begin{cases} 0 & \text{if } i = j \\ l_{ij} & \forall (i,j) \in E \\ -\infty & \text{otherwise} \end{cases},$$

we can compute the matrix $\Pi = \Pi^{(n+1)}$ according to the updating formula $\pi_{ij}^{(k+1)} = \max\{\pi_{ij}^{(k)}, \pi_{ik}^{(k)} + \pi_{kj}^{(k)}\}$. If $\pi_{ii}=0$ for all i=1,2,...,n (the numbers in the diagonal of Π), there exists a time-feasible schedule. The *ESS* is given by the numbers in the upper row of Π: $ESS=(\pi_{11}, \pi_{12}, \pi_{13},..., \pi_{1n})$. The matrix Π for the problem example is found to be the following:

$$\Pi = \begin{bmatrix} 0 & 0 & 2 & 2 & 8 & 3 & 9 & 10 & 11 & 16 \\ -\infty & 0 & -4 & 2 & 2 & 3 & 4 & 6 & 7 & 12 \\ -\infty & -2 & 0 & 0 & 6 & 1 & 7 & 8 & 9 & 14 \\ -\infty & -2 & -6 & 0 & 0 & 1 & 2 & 4 & 5 & 10 \\ -\infty & -8 & -8 & -6 & 0 & -5 & 1 & 2 & 3 & 8 \\ -\infty & -5 & -9 & -3 & -3 & 0 & -1 & 1 & 2 & 7 \\ -\infty & -9 & -9 & -7 & -3 & -6 & 0 & 1 & 2 & 7 \\ -\infty & -10 & -10 & -8 & -4 & -7 & -2 & 0 & 1 & 6 \\ -\infty & -11 & -12 & -9 & -6 & -8 & -3 & -2 & 0 & 5 \\ -\infty & -17 & -17 & -15 & -11 & -14 & -8 & -7 & -6 & 0 \end{bmatrix}$$

A π_{ij} value equal to $-\infty$ represents the non-existence of a path from activity i to activity j. The zeroes in the diagonal indicate the existence of a time-feasible schedule.

The *ESS* can be computed more efficiently by using the *Modified Label Correcting Algorithm* (Ahuja et al. (1989)), which is of time complexity $O(|V||E|)$. A label correcting algorithm is iterative and assigns tentative distance labels to nodes at each step. The distance labels are estimates of (i.e. lower bounds on) the longest path distances and are considered as temporary until the final step where π_j is the longest path length from the source node 1 to node j. The algorithm maintains a *LIST* of nodes with the property that if there is an arc (i,j) for which $\pi_j < \pi_i + l_{ij}$ then *LIST* must contain node i. The algorithm runs as follows:

Step 1. Set the distance labels for the nodes as follows: $\pi_1 = 0; \pi_j = -\infty$ for j = 2,3,...,n. Initialise the list of nodes *LIST*= [1].

Step 2. If *LIST* is empty, go to Step 5. Select the first node i from *LIST*. Delete i from *LIST*.

Step 3. If i has no uncorrected successors (successors for which the distance label has not been corrected), go to Step 2. Otherwise, select an arbitrary uncorrected successor j of i.

Step 4. $\pi_j = \max\{\pi_j, \pi_i + l_{ij}\}$. If the distance label π_j has changed and $j \notin LIST$, add node j to the end of *LIST* and go to Step 3.

Step 5. Stop.

The algorithm assumes the presence of a single starting node to start the calculations in *Step* 1. In general, this poses no problem given our assumption that the network contains a single dummy start and end node.

Let us illustrate the algorithm on the simple constraint digraph of Figure 66.

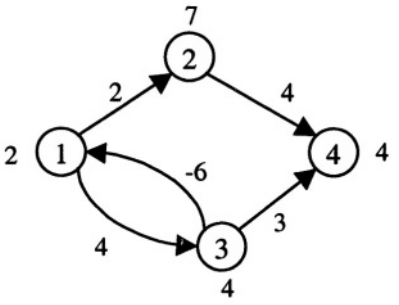

Figure 66. An example constraint digraph

Step 1. $\pi_1 = 0; \pi_i = -\infty$ for i = 2,3,4; *LIST*=[1].
Step 2. Select node 1 from *LIST*; *LIST*=[].
Step 3. Select node 2 as a successor node of node 1.
Step 4. $\pi_2 = \max\{\pi_2; \pi_1 + l_{12}\} = \max\{-\infty; 0+2\} = 2$; *LIST*=[2].
Step 3. Select node 3 as a successor of node 1.
Step 4. $\pi_3 = \max\{\pi_3; \pi_1 + l_{13}\} = \max\{-\infty; 0+4\} = 4$; *LIST*=[2,3].
Step 3. No uncorrected successors left.
Step 2. Select node 2 from *LIST*; *LIST*=[3].
Step 3. Select node 4 as a successor node of node 2.
Step 4. $\pi_4 = \max\{\pi_4; \pi_2 + l_{24}\} = \max\{-\infty; 2+4\} = 6$; *LIST*=[3,4].
Step 3. No uncorrected successors left.
Step 2. Select node 3 from *LIST*; *LIST*=[4].
Step 3. Select node 1 as a successor node of node 3.

Step 4. $\pi_1 = \max\{\pi_1; \pi_3 + l_{31}\} = \max\{0; 4 - 6\} = 0.$
Step 3. Select node 4 as a successor of node 3.
Step 4. $\pi_4 = \max\{\pi_4; \pi_3 + l_{34}\} = \max\{6; 4 + 3\} = 7;$ *LIST*=[4].
Step 3. No uncorrected successors left.
Step 2. Select node 4 from *LIST*; *LIST*=[].
Step 3. No uncorrected successors left.
Step 2. *LIST* is empty.
Step 5. Stop.
The corresponding early start schedule is obtained as *ESS*=(0, 2, 4, 7). We leave it as an exercise for the reader to apply the algorithm to the constraint digraph of Figure 65.

A *latest allowable start schedule* (*LSS*) can be computed by a similar method, based on the network with all arcs reversed and with the condition that $es_n = ls_n$. Note that for calculating the *LSS,* the maximal time-lags between activities have to be taken into account.

2.3 Activity criticality in activity networks with *GPR*s

The notions of critical paths and activity floats studied earlier have to be modified when used for activity networks with *GPR*s. A *critical path* is still the longest path from node 1 to node *n.* All the activities on this longest path are called *critical.* For the standardised example of Figure 65, there are four critical paths (indicated in bold), namely <1-3-5-8-10>, <1-3-5-8-9-10>, <1-3-5-7-8-10> and <1-3-5-7-8-9-10>. The length of each of these critical paths equals 16.

2.3.1 Cycles and tree structures

In a standardised project network with *GPRs,* a critical path may contain cycles, provided they are of length zero. When determining the critical path(s) in nonstandardised networks with *GPRs,* the maximal time-lags require special attention. A critical path may not be a path in the strict sense, but rather be a tree structure, because it may include an activity connected with a minimal and maximal time-lag in the same direction, ensuring that the start or completion of the activity should be exactly equal to the start or finish of a predecessor plus a certain time-lag. The critical path will always contain a chain from the start to the end activity, but it can contain several branches connected to this chain. De Reyck (1998) provides the example of Figure 67 to illustrate this point.

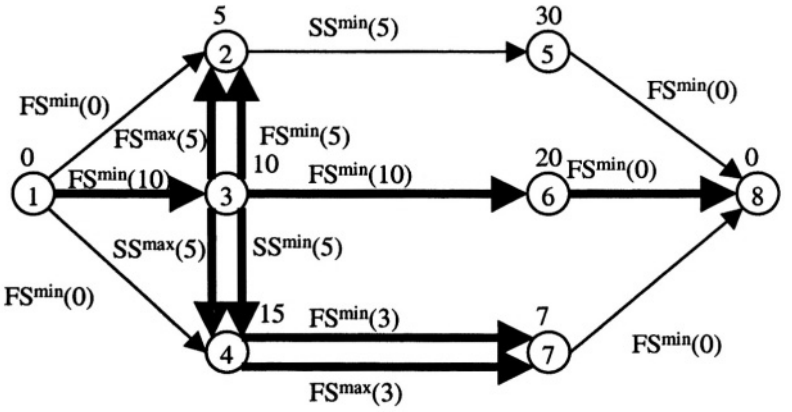

Figure 67. A project network with *GPR*s (De Reyck (1998))

Figure 68 shows the standardised version of the network in Figure 67, with the critical paths indicated in bold. The longest path from source to sink in the network of Figure 68 has a length of 50, being the critical path length. There are, however, several different critical paths, amongst others <1,3,6,8>, <1,3,2,3,6,8>, <1,3,4,3,6,8> and <1,3,4,7,4,3,6,8>. The last three paths contain a cycle. Only activity 5 is non-critical. The critical path in the non-standardised project network of Figure 67 contains several branches and has the shape of a tree rather than a path.

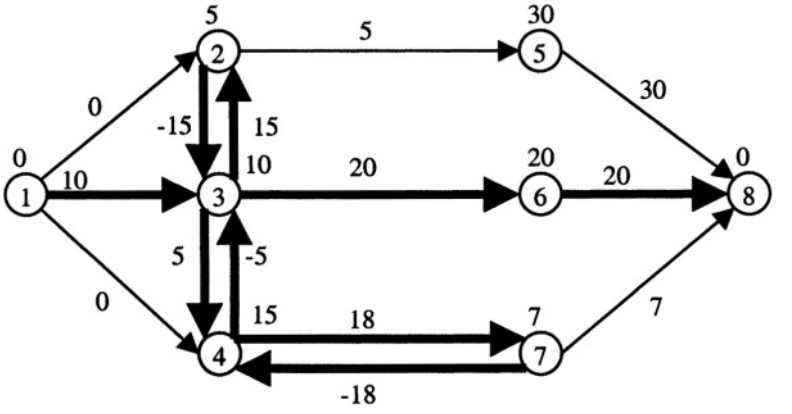

Figure 68. A standardised project network

However, transforming the maximal time-lags into minimal ones (while still distinguishing between the four types of precedence relations), yields

multiple critical paths which do not exhibit a tree structure, as shown in Figure 69. We alert the reader for the fact that criticality analysis should be performed with extreme care when *GPRs* with maximal time-lags come into play. Therefore, we agree with the argument of De Reyck (1998) who advises that a temporal analysis should be performed on project networks in which the maximal time-lags are transformed into minimal ones in the opposite direction. The constraint digraph should not be used for criticality analysis since it does not correctly represent the impact of altering activity durations on the earliest start schedule or on the total project duration.

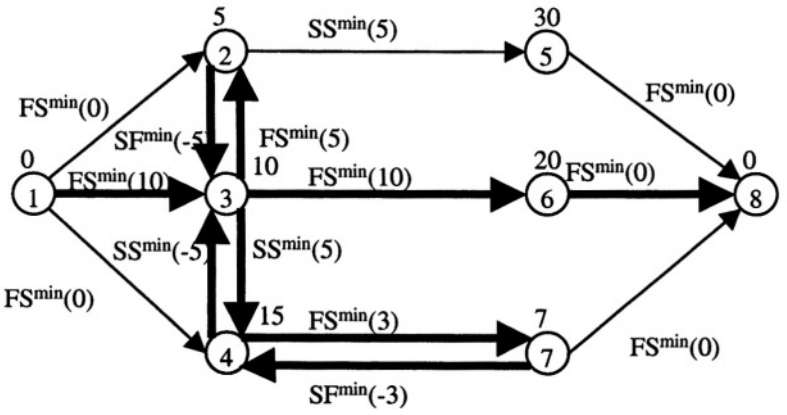

Figure 69. A project network with minimal time-lags

2.3.2 Types of criticality

In traditional CPM terminology, an activity is considered to be critical if *delaying* that activity causes a project delay. This implies that an *increase* of the *duration* of a critical activity results in an increase of the project duration with the same amount. This is not always the case when the project network contains *GPRs.* In such networks, *delaying a critical activity* will *always* result in an equal increase in the project duration, but a *duration increase* of a critical activity will *not always* lead to an increase in the duration of the project. In the example of Figure 69, delaying critical activity 3 with one time unit will result in a unit increase in the duration of the project. However, a unit increase in the duration of activity 3 will not cause a project delay at all. This is due to the fact that the (minimal) time-lags of the critical path(s) containing activity 3 are of the *SS*- and *FS*-type for its (immediate) predecessors, and are of the *SS*- or *SF*-type for its (immediate) successors. Therefore, only the start of activity 3 really matters and should not be

delayed in order to avoid a project delay, whereas the finish of activity 3 has some float. Actually, as we will explain below, activity 3 is *start-critical*: increasing its duration does not affect the project duration, delaying its start leads to a project duration increase.

Elmaghraby and Kamburowski (1992) distinguish between different criticality types. The following definitions apply. An activity is labelled:

- *Critical* when it is located on a critical path;
- *Start-critical* if (a) it is critical, and (b) if the project duration increases when the start time of the activity is delayed;
- *Finish-critical* if (a) it is critical, and (b) if the project duration increases when the finish time of the activity is delayed;
- *Forward-critical* (described as normal-critical by Hajdu (1997)) if (a) it is start-critical, and (b) when the project duration increases when the activity's duration is increased;
- *Backward-critical* (described as reverse-critical by Hajdu (1997)) if (a) it is finish-critical, and (b) when the project duration increases when the activity's duration is decreased;
- *Bi-critical* if (a) it is start- and finish-critical, and (b) when the project duration increases when the activity's duration is either increased or decreased.

The criticality type of an activity can be determined by looking at the precedence relations originating from all critical activities leading to the activity in question and originating from the activity to all other critical activities. Table 6 relates the criticality type of a critical activity to the time-lag between the activity and its (immediate) predecessors and (immediate) successors. The second column of Table 7 shows the criticality type for the critical activities of the project network of Figure 64.

Table 6. Five types of criticality (De Reyck (1998))

	Minimal time-lag between all critical activities in the set of predecessors of activity *i* of type	Minimal time-lag between activity *i* and all critical activities in the set of its successors of type
Start-critical	*SS* or *FS*	*SF* or *SS*
Finish-critical	*SF* or *FF*	*FF* or *FS*
Forward-critical	*SS* or *FS*	*FS* or *FF*
Backward-critical	*SF* or *FF*	*SF* or *SS*
Bi-critical	(*SS* or *FS*) and (*SF* or *FF*)	(*FS* or *FF*) and (*SF* or *SS*)

Table 7. Activity criticality and flexibility for the problem example of Figure 64 (De Reyck (1998))

Activity	Criticality	Flexibility
1	start-critical	bi-inflexible
2	non-critical	backward-inflexible
3	start-critical	bi-flexible
4	non-critical	forward-inflexible
5	backward-critical	backward-inflexible
6	non-critical	bi-flexible
7	finish-critical	bi-flexible
8	bi-critical	bi-inflexible
9	forward-critical	forward-inflexible
10	forward-critical	bi-inflexible

2.4 Activity flexibility in activity networks with *GPR*s

As mentioned by Elmaghraby and Kamburowski (1992), the concept of criticality of an activity has been wedded to that of project duration. However, increasing or decreasing the duration of an activity in the presence of *GPRs* may not be permissible since the resulting project may become time-infeasible. The criticality of an activity says nothing about the effect of shortening or extending the duration of an activity on the time-feasibility of the project. It determines the effect on the total project duration, given that the change is feasible with respect to the temporal constraints. In the presence of *GPRs* one may be interested in varying an activity duration in order to achieve network feasibility, rather than affect the project duration. This novel consideration inspired Elmaghraby and Kamburowski (1992) to introduce the concept of *flexibility* to denote the freedom to manipulate the activity duration to achieve feasibility. The absence of such flexibility results in denoting the activity as *inflexible.*

An activity is said to be *forward-inflexible* (*backward-inflexible*) if extending (shortening) the activity duration by a small amount ε (or in the discrete case, 1) results in a time-infeasible project or a project completion time exceeding the shortest project duration of the original project. An activity is *bi-inflexible* if it is forward- and backward-inflexible. Consequently, if an activity is forward-critical (backward-critical) (bi-critical), then it is also forward-inflexible (backward-inflexible) (bi-inflexible). If an activity is start- or finish-critical, then it is also bi-inflexible. The last column of Table 7 denotes the flexibility type for each activity in the network example of Figure 64.

If one aims at shortening the project duration by acting on the activity durations, one should focus on those activities that are backward-flexible and forward-critical (such as activity 9 in the network of Figure 64), or forward-flexible and backward-critical (such as activity 5 in the network of Figure

64). The duration of the former should be reduced, the duration of the latter should be extended. Obviously, in order to reduce the duration of a project, (backward-flexible and forward-critical or forward-flexible and backward-critical) activities in every critical path should be tackled.

2.5 Activity floats in activity networks with *GPR*s

The notion of activity floats for activity networks with *GPRs* are similar to those of standard CPM networks. The *total float* of an activity is nothing else than the maximal amount of time that an activity can be delayed without causing a project delay. An activity is critical if its total float equals zero. However, in CPM networks an activity may be delayed by the late start or a duration increase of a predecessor activity. In activity networks with *GPRs,* only late starts are used for computing activity floats, since activity duration increases may not lead to a project delay or may not be permissible at all because of time-infeasibilities.

3. EXERCISES

1. Consider the following project activities with corresponding data:

Activity	*Duration*	*Immediate successors*
1	0	2, 3, 4
2	2	5, 6
3	1	9
4	10	10
5	5	7, 11
6	4	8, 9
7	1	10
8	6	12
9	4	12
10	1	12
11	3	12
12	0	-

All precedence relations are of the finish-start type with zero time-lag. Draw the network in *AoN* format. Perform the forward and backward critical path calculations. Derive for each activity the total, free and safety float values.

2. Consider the following data for a project with five activities: