Scheduling in Networks with Time Dependent Arc Lengths Based on a Loop Finder Algorithm

Helga Csordas
Department of Construction Technology and Management, Faculty of Architecture
Budapest University of Technology and Economics, Hungary
hcsordas@ekt.bme.hu

DOI 10.5592/otmcj.2012.2.7
Research paper

NETWORK MODELS ARE OFTEN APPLIED IN PROJECT SCHEDULING. Here a longest path problem has to be solved to get the project duration. There are many generalizations in this theme. One of these is applying time dependent process durations. This potential is very important because this is the key to use calendars in schedules. An other one is applying maximal constraints which result loops in the schedule. If these two potentials are allowed together, the prefix of length of the loops can change according the start time. An algorithm has been already presented for this problem. This study is an opportunity to accelerate it.

Keywords
project scheduling,
time dependent activity,
loop, algorithm

INTRODUCTION

The first scheduling models were presented in the late of '50-s (Bellmann and Ford, 1958; Dijkstra, 1959). The problem of these works is very simplified. There is not allowed negative or changeable process durations and loops. The solutions based on linear programming. Scheduling is a longest path problem. Let this simple model be called the *original model*. For applying a network model in civil engineering practice it has to be suitable for handling two features in consideration of scheduling.

- The first one is the possibility of changing process durations depending on their start times. This is the key to apply calendar. There is already a proper model for the problem (Franck et al., 2001). It is presented in detail later.
- The second one is using maximal constraints for activities and connections. This is useful and important in practice. In the linear programming method it is possible to give only minimal constraints. For applying maximal constraint it must be converted by multiply the assumption with (-1). It effects negative process duration and turning back arc.

Example: There is an expensive machine used in two activities which take 4 and 5 days and follow each other. There is an upper limit (10 days) to rent the machine. The problem can be modelled as it is shown in Fig. 1.

Remark: In schedules activities and connections are usually distinguished. Collectively they can be called processes.

In construction practice there are already available some software which use Activity-On-Node network models. Otherwise in research works Activity-On-Arc models are general because this network is appropriate for modelling more problems. The AOA network model with negative process durations and maximal constraints is more general than the AON model.

Example: The two networks on Fig. 2. show the same problem. The activity durations in AON mean a minimal and a maximal constraints together for the difference of their start and finish times. The connections also can be presented in AOA model.

The difficulties of this problem are analyzed in detail (Csordas, 2008) and the conclusion is that maximal constraints create directed $H=\{i=x_0,x_1,\ldots,x_o\}$ loops which can interlock and create new loops which are unregulated. The algorithm based on an iteration method which finds the first finite solution of the scheduling. Further let it be called the *traditional algorithm*. As this paper deals with the acceleration of this algorithm, the problem and the solution must be demonstrated briefly.

Denote *[N,A]* a directed graph where *N* is the set of nodes and *A* is the set of arcs. Let *n* and *m* be the number of nodes and the number of arcs respectively. There is only one start node *s* and one end node *r*. Arcs are directed only outwards from *s*. Directed graph contains no parallel arcs. For all $k\neq s,r$ there is a directed path $P(s,r)=\{s,x_1,\ldots,r\}$ from *s* to *r* containing node *k*.
Let *T* be an arbitrary integer as the maximal acceptable project duration. Denote τij, $\forall ij\in A$ the number of necessary workdays to realize the object of the process. Let it be called the *effective activity duration*. According to a previously defined resource it is constant. Denote d_{ij}, $\forall ij\in A$ *a T* long vector as the work pattern of the resource assigned to it.

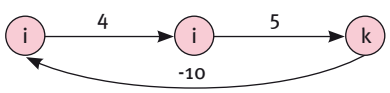

Figure 1
Minimal and maximal constraints

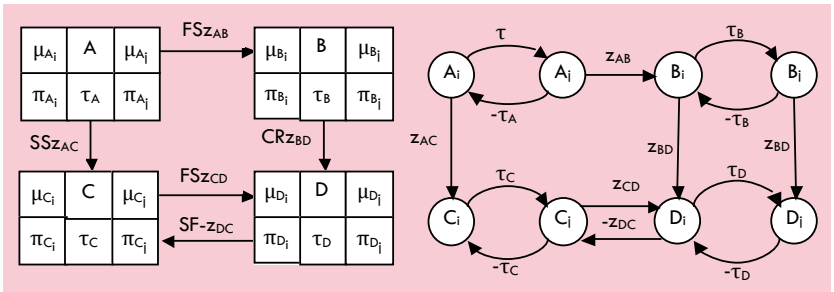

Figure 2 Matching of the AON and AOA models

$$d_{ij}(t)=\begin{cases}1, \text{ if t is workday}\\ 0, \text{ if t is holiday}\end{cases}$$

(1)

According to the definition $d_{ij}(t)=o$ in every other case. Depending on a μ_i start time based on (1) the *real (current) process duration* (denoted with $\theta_{ij}(\mu_i)$) can be defined.

$$\tau_{ij}=\operatorname{sgn}(\tau_{ij})\sum_{t=\mu_i}^{\mu_i+\min\{\theta_{ij}(\mu_i)\}} d_{ij}(t)$$

(2)

Corollary: *The reading of the minimal constraints is: From a given start time what is the minimal processing time which contains exactly the necessary number of workdays.*

The reading of the maximal constraints is: Back from a given finish time what is the maximal processing time which contains exactly the necessary number of workdays.

The aim is to find a μ system, where μi $\forall i\in N$ is the earliest occurrence of *i* and

$$\theta_{ij}(\mu_i)\le\mu_j-\mu_i,\ \forall ij\in A$$

(3)

The algorithm is in Fig. 3. Start μ system is the same like in the original models.

Remark: In practice the value $-\infty$ is a properly large negative number. Because of the calendar the first value before the calendar (generally it is *(-1)*) is proper here.
Parameter *k* follows the number of iterations. In every iterations every arcs are examined and the algorithm gives potentials for every nodes in the network. The algorithm stops if every potential is ready or any of them become over *T*. The control of the right potentials is the last iteration when there is no more change in the values. It means it is only a check counting.
The key issue of the problem is that the network contains loops. The *length of a loop H* is

$$\rho_H=\sum_{ij\in H}\theta_{ij}(\mu_i)$$

(4)

As it is known positive value for ρ_H is not allowed in the maximal path - minimal potential algorithm. If the processes in the loop have different work patterns it effects a new occurrence in the schedule. By counting round the loop a new start time arises according to the length of the loop. This gives new potentials for the nodes in the loop in the new round. Because of the time dependent process durations the length of the loop can change during these iterations. It can be positive again or zero or alter sign. The two last cases give finite solution. According to the paper presented the traditional algorithm the solution can be

- finite where there is at least one path *P(s,r)* where

$$\mu_r=\sum_{ij\in P(s,r)}\theta_{ij}(\mu_i)$$

- finite where there is at least one loop H where

$$\rho_H = \sum_{ij \in H} \theta_{ij}(\mu_i) = 0$$

- infinite which means that the project duration is larger than T

The second solution is not known in the problem with constant process durations. It means a split in scheduling before the loop H. In this case let H be called *critical loop*.

The number of counting round the loops is only depended on the set of the process durations in them and not on the structure of the network (number of nodes or arcs). Unfortunately it is not a good feature to appreciate the run-time of the algorithm but the corresponding is possible.

The paper has four further sections. In the next one the features of loops are shown and some definition are given, then there is the new algorithm. After the run-time of the two algorithms are compared and finally there is an example.

Features of the loops in the network

The loops because of maximal constraints can be distinguished according to their structures.

Definitions:

- *Primary loop is a loop which has exactly two nodes and two arcs. So* $N_H=\{i,j\}$ $N_H \in N$ *and* $A_H=\{(i,j);(j,i)\}$ $A_H \in A$.
- *Maximal loop* is a set of arcs which create a loop without repeating any arc and there is no further arcs to expand it larger.
- Loops are *independent* if every of them contains such arcs which are disjointed. So H_1 and H_2 are independent if $A_{H^1} \setminus (A_{H^1} \cap A_{H^2}) \neq \emptyset$ *and* $A_{H^2} \setminus (A_{H^1} \cap A_{H^2}) \neq \emptyset$.
- *Aggregated loop* is a set of maximal loops which has at least one jointed arc. So H_1 and H_2 create an aggregated loop if $(A_{H^1} \cap A_{H^2}) \neq \emptyset$ and it generates $Q=(N_{H1} \cup N_{H2})$.

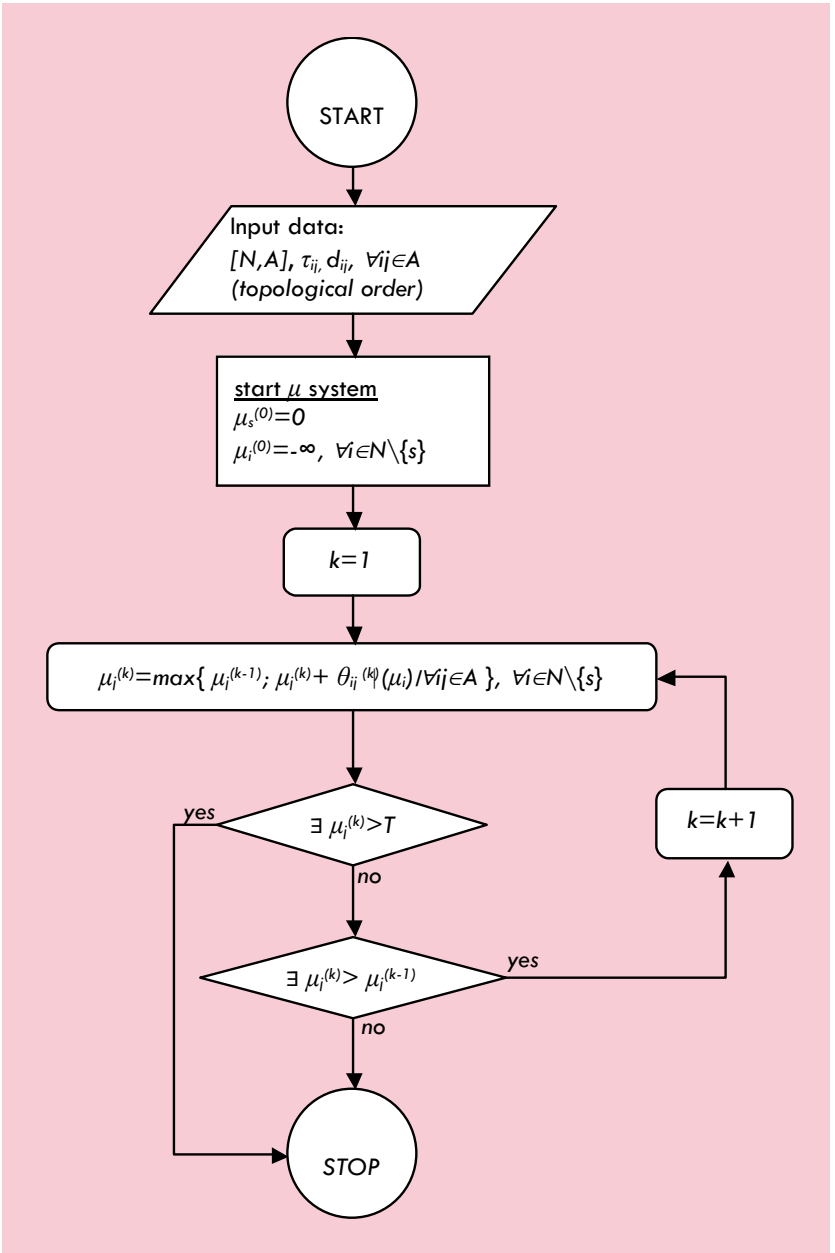

Figure 3 **The traditional algorithm**

- Nodes out of any loops generate *acyclic sets* denoted by W

Example: *Every kind of loops can be found in the network in Fig. 4.*

- Primary loops: $H_1=\{1,6,1\}$, $H_2=\{2,7,2\}$, $H_3=\{4,9,4\}$, $H_4=\{5,r,5\}$, $H_5=\{6,7,6\}$, $H_6=\{9,r,9\}$ signed by broken line.
- Maximal loops: $H_7=\{1,2,7,6,1\}$, $H_8=\{1,6,7,2,7,6,1\}$, $H_9=\{4,5,r,9,4\}$ and $H_{10}=\{4,9,r,5,r,9,4\}$ signed by thick line.
- Independent loops: H_1, H_2, H_3, H_4, H_5, H_6, H_7 and H_9 are independent. H_8 is independent only of loops H_3, H_4, H_6, H_7, H_8 and H_{10}. H_{10} is independent only of loops H_1, H_2, H_5, H_7, H_8 and H_9.
- Aggregated loops: $Q_1=\{1,2,6,7\}$ (which contains H_1, H_2, H_5, H_7 and H_8), $Q_2=\{4,5,9,r\}$ (which contains H_3, H_4, H_6, H_9 and H_{10}) signed by clouds with broken line.
- Acyclic sets: $W_1=\{s\}$, $W_2=\{3,8\}$ signed clouds with dotted line.

Loop finder algorithm

The traditional algorithm determines μ_i potentials for every $i \in N$ nodes in every

iteration. Because of the changeability of the length of the loops it effects a lot of redundant counting for nodes before and behind the loop of which finite solution is looked for over many iterations. This occurrence does not exist in the problem with constant process durations. Avoiding the redundant counting the iterations should be localized to only one loop until finding the right potentials in it. As the interlocked loops can take effects to each other the localization have to be referred to the aggregated loops.

Finding aggregated loops

Finding the nodes belonged to the same aggregated loop is possible by applying a path finding method (Warshall, 1962) (Vattai, 1993). In the solution at first it has to be defined an *n×n* sized adjacency matrix *M*. Connectivity matrix *V* can be derived from *M* by the next logical evaluation

$$V_{ij} = \begin{cases} 0, \text{ if } M_{ik} \cdot M_{kj} = 0, ij \notin A \\ 1, \text{ if } ij \in A \\ 2, \text{ if } M_{ik} \cdot M_{kj} > 0, ij \notin A \end{cases} \quad \forall k \in N$$

(5)

Example: *The matrices M and V for the network in the previous example are in Fig. 4.*

Matrix *V* shows the next information:

- In the main diagonal the nodes of loops can be distinguished. $V_{ii}>o$ means that there exists $P(i,i)=\{i=x_0,x_1,\ldots,x_o\}$ path which is a loop.
- The nodes of loops of which columns have $V_{ij}>o$ values at the same positions belong to the same aggregated loops. The different loops are signed under the matrix *V*.
- In case of the rest nodes the number of $V_{ij}>o$ values sign the order of nodes and the position among the aggregated loops. This is also signed under the matrix *V*.

The result of the counting is the same like in the previous example. If the nodes belonged to the same set *W* or *Q* are not in order, it can be rearranged simply.

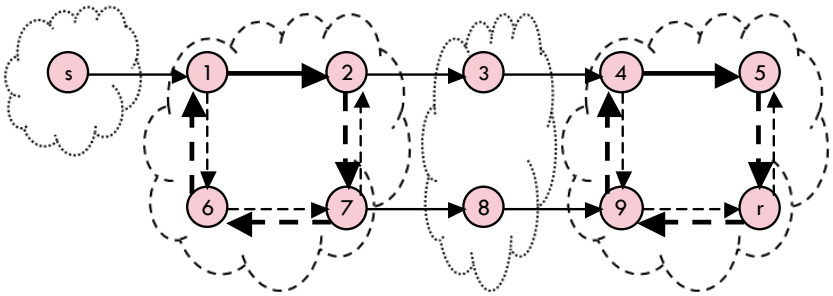

Figure 4 **Loops in the network**

The topological order

The topological order of the nodes generally makes the algorithm simple (Ahuja et al., 1993). Of course it is possible only in case of acyclic graphs. If the aggregated loops are constricted into nodes, the network will be acyclic and the topological order can be determined. The order of nodes inside an aggregated loop is indifferent.

Standardization the model let *N* be defined as the alternate queue of sets *W* and *Q* like $W_1, Q_1, W_2, \ldots, W_q, Q_q, W_{q+1}$, where *q* is the number of aggregated loops in the network. Let every set be marked even if it is empty.

The algorithm

The method for determining potentials is the same like in the traditional algorithm. Otherwise the iterative counting to find the solution is localized to the current set *W* or *Q* which are in topological order. It is very efficient because the sets of nodes do not effect backwards to the others. So after finding the solution in one of them, it does not change any more in the course of counting further. The algorithm is in Fig. 6.

At first it has to set the start μ system and determine the topological order as the traditional algorithm. Parameter *x* follows the sets from *1* to *q*, k the number of iterations. In case of counting in a set *W* it is unnecessary more iteration, so it does not need a loop. In case of a set *Q* the loop can be seen. The end of the iteration in the loop is when there is not any changes in the values of potentials in the current set. So the last iteration is the check counting. The counting step on the next set *W* or *Q* if every potentials in the current set are ready. The algorithm stops if every potential is ready or any of them become over *T*.

Comparing run-times

Up to this point it is presumed that the more aggregated loops are in the network which need more iterations for finding

M	s	1	2	3	4	5	6	7	8	9	r
s		1									
1			1				1				
2				1				1			
3					1						
4						1				1	
5											1
6		1						1			
7			1				1		1		
8										1	
9					1						1
r						1				1	

V	s	1	2	3	4	5	6	7	8	9	r
s		1	2	2	2	2	2	2	2	2	2
1		2	1	2	2	2	1	2	2	2	2
2		2	2	1	2	2	2	1	2	2	2
3					1	2				2	2
4					2	1				1	2
5					2	2				2	1
6		1	2	2	2	2	2	1	2	2	2
7		2	1	2	2	2	1	2	1	2	2
8					2	2				1	2
9					1	2				2	1
r					2	1				1	2
	W_1	Q_1	Q_1	W_2	Q_2	Q_2	Q_1	Q_1	W_2	Q_2	Q_2

Figure 5 **Loop finding by the matrices M and V**

the right potentials, the more efficient the loop finder algorithm is. Assume that the potentials have been determined in sets W_i and Q_i, $i=1...k$ and W_k. In set Q_k potentials can be determined for all nodes according to (3) and get the check potential. If the length of this loop is positive, all potentials have to be counted in set Q_k again according to the check potentials. Denote l_k, $k=1...q$ the necessary number of iterative counting in set Q_k. The algorithms can be compared based on performed actions detailed in Fig. 7.

The rows contains the parts of the network, the columns the necessary number of iteration of these parts accordingly. Obviously in sets W it needs only one counting without any iteration. Signed by "X" the position (i,j) if counting happens on part i by j times. The check counting is also has to be done once distinguished by "+" in the table.

The traditional algorithm counts and checks potentials for all nodes in set A until finding the solution in the whole network. Every potential is determined again and again in every iteration. Thus potentials for nodes in W_k are also determined in the iteration belonged to the nodes in Q_k so it is unnecessary that single counting belonged to them except the first and the last ones as the first and the check counting for the potentials. Based on these considerations the claim for actions of the first algorithm is

$$o_1 = m \cdot \left(2 + \sum_{k=1}^{q} l_k\right) \quad (6)$$

The loop finder algorithm handles aggregated loops one at a time and look for the solution in only the current set of nodes. In this case it has to be taken that single counting in every set W_k. The counting loop of the set Q_k stops with the check counting for the aggregated loop. So the claim for actions of the accelerated algorithm is

$$o_2 = \sum_{k=1}^{q+1} m_{W_k} + \sum_{k=1}^{q} m_{Q_k} \cdot (2 + l_k) \quad (7)$$

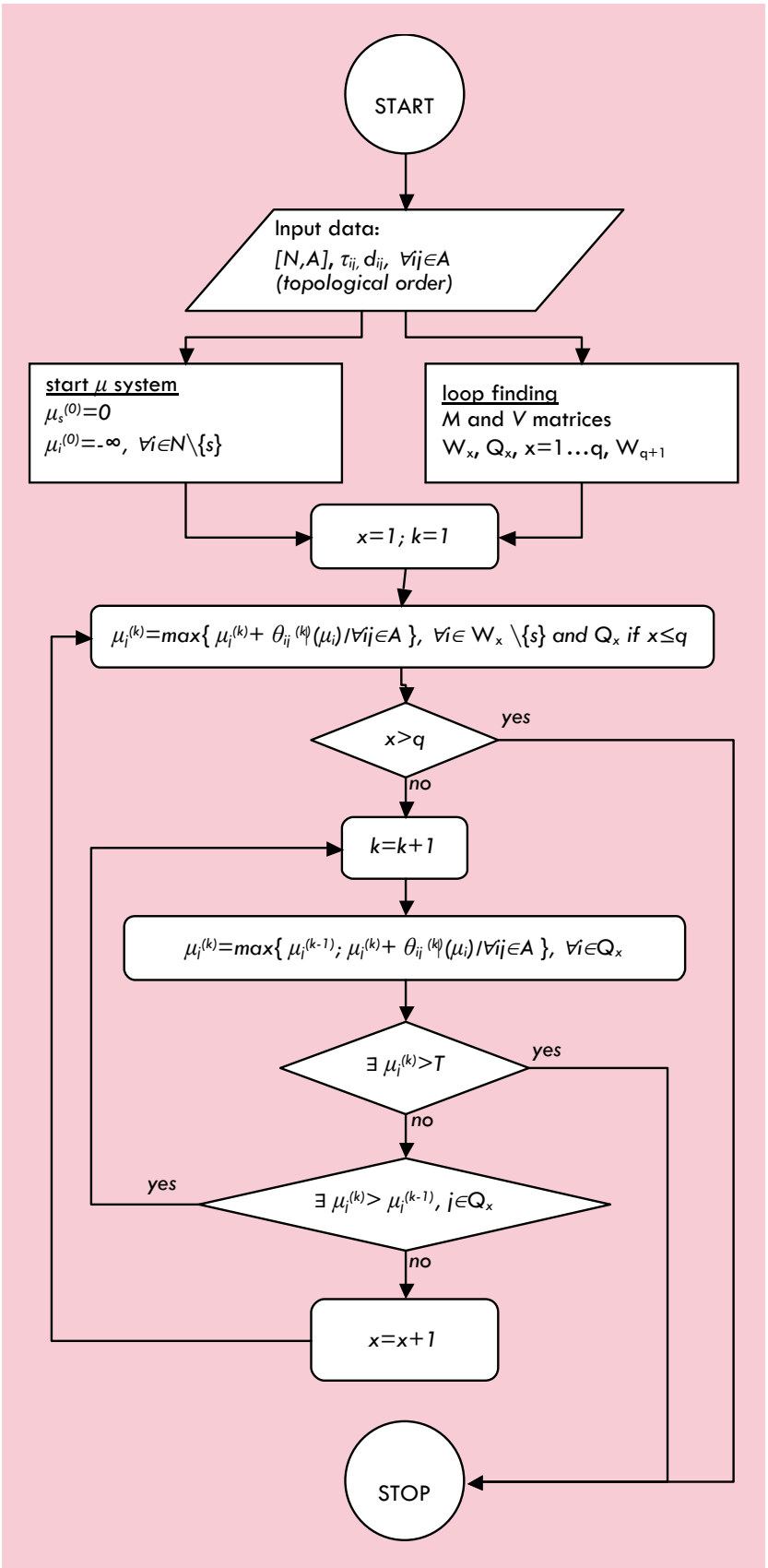

Figure 6 **The loop finder algorithm**

Tr	1	l_1	1	l_2	...	1	l_k	...	1	l_q	1
W_1	X	X		X	...		X	...		X	+
Q_1	X	X		X	...		X	...		X	+
W_2	X	X		X	...		X	...		X	+
Q_2	X	X		X	...		X	...		X	+
⋮	⋮	⋮		⋮	...		⋮	...		⋮	⋮
W_k	X	X		X	...		X	...		X	+
Q_k	X	X		X	...		X	...		X	+
⋮	⋮	⋮		⋮	...		⋮	...		⋮	⋮
W_q	X	X		X	...		X	...		X	+
Q_q	X	X		X	...		X	...		X	+
W_{q+1}	X	X		X	...		X	...		X	+

L-f	1	l_1	1	l_2	...	1	l_k	...	1	l_q	1
W_1	X										
Q_1	X	X	+								
W_2			X								
Q_2			X	X							
...				...	...						
W_k						X					
Q_k						X	X				
...							...	...			
W_q									X		
Q_q									X	X	+
W_{q+1}											X

Figure 7 **Number of actions in case of the traditional and the loop finder algorithms**

	s	1	2	3	4	5	6	7	8	9	r
s		0									
1			2	3							
2					4	1					
3		-3			0						
4			-4	-1			2				
5								1			
6										2	
7									2	5	
8											4
9								-5			0
r									-4	0	

Figure 9 **Weighted adjacency matrix**

Where m_{W_k} and m_{Q_k} are the number of arcs belonged to the denoted set. ($\forall ij \in A_{Qk}$ if i,j$\in Q_k$) So

$$m = \sum_{k=1}^{q+1} m_{W_k} + \sum_{k=1}^{q} m_{Q_k}$$

Generally the difference of the actions of the algorithms is

$$o_1 - o_2 = \sum_{k=1}^{q+1} m_{W_k} \cdot \left(\sum_{i=1}^{q} (l_i) + 1 \right) + \sum_{k=1}^{q} m_{Q_k} \cdot \left(\sum_{i=1}^{q} (l_i) - l_k \right)$$

(8)

The difference is significant and proportional to the number of set Q and the necessary number of iterations in them.

Example

The problem

Let look at Fig. 8. The structure of the network is the same like in Fig. 4. The ordering of the nodes is changed according to the result of loop finding.

The vertical arcs represent the activities and the horizontal ones the connections between them. Activity durations mean the necessary number of workdays (τ_{ij}, $\forall ij \in A$). $T=17$. Look at a simple work pattern, a week with five workdays. The first day is Wednesday.

$d_{1,2}=d_{3,4}=d_{7,8}=d_{9,r}=$[11100111110011110]

This calendar is applied only for vertical arcs as the activities of the project. The real (current) activity durations are in Table 1. depending on their start times.

Remark: For example in the first row there is a 3-day activity. This is its effective activity duration. If it starts on the 1st day it takes 3 calendar days, but if it starts on the 3rd day it takes already 5 calendar days because there is a 2-day period when work is not allowed and it can be finished only on the 7th day. As the work pattern is defined only in the period T, it means that for the other days $d_{ij}(t)=o$. This effects the ∞ values which may be T in practice.

This algorithm is useful if different calendars are in the model. Let the work pattern of the connections be constant *1*. Thus their real process durations are also constant.

Presenting the algorithm is practical on the adjacency matrix. The weighted adjacency matrix of the network is in Fig. 9.

The sets W_1, Q_1, W_2 and Q_2 are visible thanks to the ordering. The sets of the nodes create blocks which are separated by double lines. The results of the iterations can be placed behind the adjacency matrix.

start	times	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
arc id	τ_{ij} \ d_{ij}	1	1	1	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0
1,3	3	3	5	5	5	4	3	3	3	5	5	5	4	3	3	3	∞	∞	∞
3,1	-3	-∞	-∞	-∞	-∞	-∞	-∞	-5	-5	-5	-3	-3	-4	-5	-5	-5	-5	-3	-3
2,4	4	6	6	6	6	5	4	4	6	6	6	6	5	4	4	∞	∞	∞	∞
4,2	-4	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-6	-6	-6	-4	-5	-6	-6	-6	-6	-6	-4
7,9	5	7	7	7	7	6	5	7	7	7	7	7	6	5	∞	∞	∞	∞	∞
9,7	-5	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-7	-7	-7	-8	-9	-7	-7	-7	-7	-7
8,r	4	6	6	6	6	5	4	4	6	6	6	6	5	4	4	∞	∞	∞	∞
r,8	-4	-∞	-∞	-∞	-∞	-∞	-∞	-∞	-6	-6	-6	-4	-5	-6	-6	-6	-6	-6	-4

Table 1 **Real activity durations for every start time**

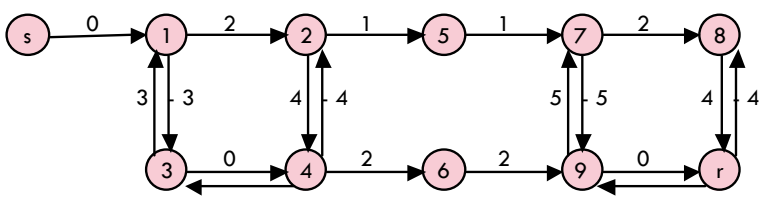

Figure 8 **Example–the problem**

The traditional method

In case of the traditional algorithm the counting is in Fig. 10.

The real activity durations are under their effective activity durations in order to their coming out in the line of the potential belonged to it. The columns behind the matrix show the iterations numbered on the top. The start μ system is in the column number o. If there are more values in a column at a node, it means higher potentials for the node in the same iteration.

The problem is solved in ten iterations. According to the columns it is seemed that the potentials of the first aggregated loop (Q_1=*{1,2,3,4}*) are ready in the fifth iteration. It needs some iterations in the aggregated loop and create a split between the sets W_1=*{s}* and Q_1 as μ_s-μ_s=3-0›0=$\theta_{s,1}(\mu_s)$.

From the sixth iteration the potentials of the second aggregated loop (Q_2=*{7,8,9,r}*) are looked for. There is also a split between the sets W_2=*{5,6}* and Q_2 as μ_7-μ_5=9-6›1=$\theta_{5,7}(\mu_5)$ and μ_8-μ_6=16-11›2=$\theta_{6,8}(\mu_6)$.

So there are two critical loops which get nonpositive lengths before *T*. They are H_6=*{1,2,3,4,1}* with the length ρ_{H6}=2+4+(-1)+(-5)=0 and H_7=*{7,8,9,r,7}* with the length ρ_{H7}=2+5+0+(-7)=0.

	s	1	2	3	4	5	6	7	8	9	r
s		0									
1			2	(3) 3 5 5							
2					(4) 6 5 4	1					
3		(-3) -∞ -5 -5			0						
4			(-4) -6 -6	-1			2				
5								1			
6										2	
7									2	(5) 6 7 7 7 7	
8											(4) 4 6 6 6 5
9								(-5) -9 -7 -7 -7			0
r									(-4) -4 -6 -6 -6 -6	0	

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0
-∞	0	0 2	2	2 3	3	3	3	3	3	3
-∞	2	2	4	4	5	5	5	5	5	5
-∞	3 7	7	7 8	8	8	8	8	8	8	8
-∞	8	8	9	9	9	9	9	9	9	9
-∞	3	3	5	5	6	6	6	6	6	6
-∞	10	10	11	11	11	11	11	11	11	11
-∞	4	4	6	6 7	7	7 8	8	8 9	9	9
-∞	6	6	8	8	9	9	10	10	11	11
-∞	12	12	13 14	14	14 15	15	15 16	16	16	16
-∞	10 12	12	14	14	15	15	16	16	16	16

Figure 10 **Counting in case of the traditional method**

The loop finder method

In case of the loop finder algorithm the counting is shown in Fig. 11.

It is visible very good which sets are counted in the iterations. In case of the acyclic sets *W* it needs only one iteration while the first potentials of the next set *Q* can be determined. Until the right values of the nodes in set *Q* the algorithm takes loops and examines only the current aggregated loop. With the check counting for the set *Q* the next set *W* can be examined. The counting items are separated by double lines.

Of course the result is the same like at the traditional algorithm.

Comparing the run-times

The **traditional algorithm** takes ten iterations. Every arc is checked in every iterations. m=19, so the amount of counting is o_1=19·10=190.

Check: The first aggregated loop (Q_1=*{1,2,3,4}*) needs four more counting round in the loop for finding the right potentials after the iteration number *1*. So I_1=4.

After that, from the iteration number *6*, the second aggregated loop (Q_2=*{5,6,7,8}*) needs also four more counting round. So I_2=4.

Checking the potentials is in the iteration number 10.

According to (6) o_1=19·[2+(4+4)]=190.

The **loop finder algorithm** takes eleven iterations. Here it is need to know the number of arcs belonged to the sets.

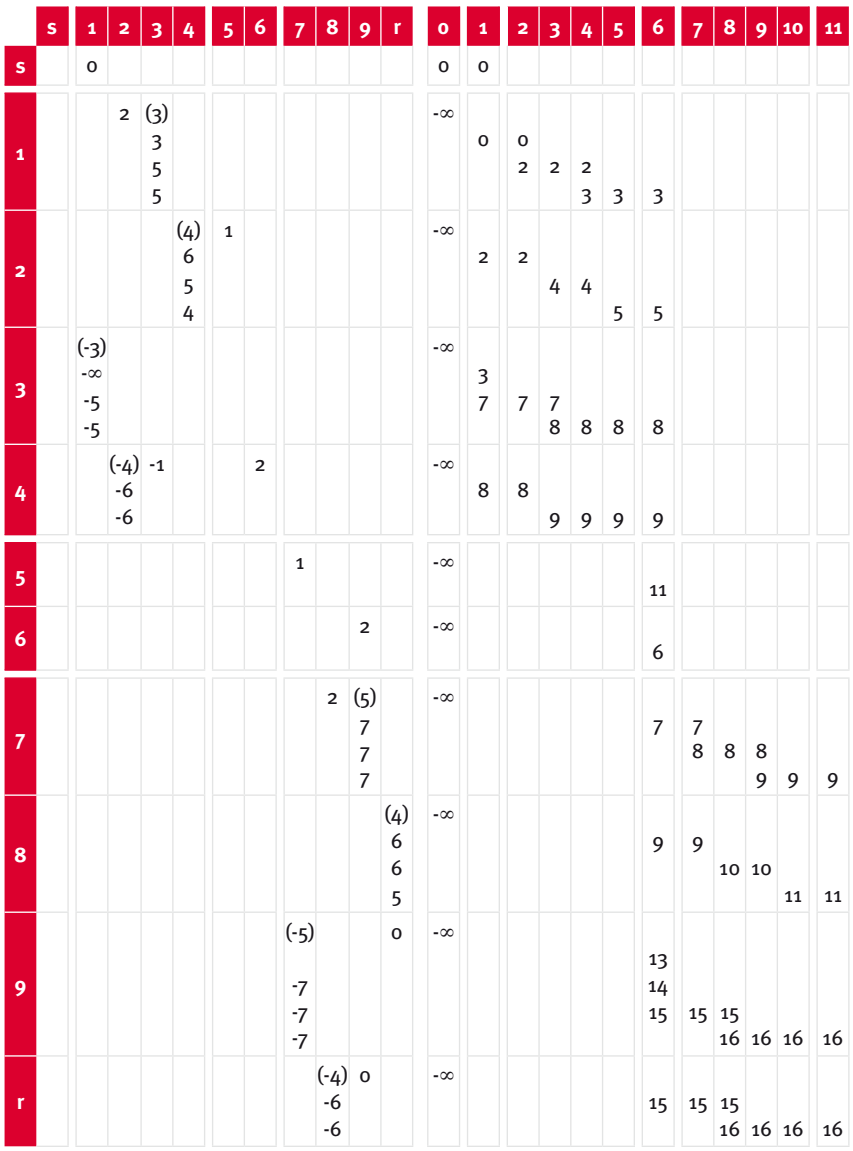

Figure 11 Counting in case of the loop finder method

It can be read out from the blocks of the adjacency matrix. $m_{W_1}=1$, $m_{Q_1}=7$, $m_{W_2}=4$, $m_{Q_2}=7$ and $m_{W_3}=0$. Check: $1+7+4+7+0=19=m$. Summarize the actions:

- In the iteration number 1 the counting of the potential of the set $W_1=\{s\}$ and the first counting in the set $Q_1=\{1,2,3,4\}$ happen. It means $m_{W_1}+m_{Q_1}=1+7=8$ actions.
- In the iterations number 2 – 5 the right potentials in the set Q_1 are looked for. So $I_1=4$ as in the traditional method and it means $I_1 \cdot m_{Q_1}=4 \cdot 7=28$ actions.
- In the iteration number 6 the check counting in the set Q_1, the counting in the set $W_2=\{5,6\}$ and the first counting in the set $Q_2=\{7,8,9,r\}$ happen. It means $m_{Q_1}+m_{W_2}+m_{Q_2}=7+4+7=18$ actions.
- In the iterations number 7 – 10 the right potentials in the set Q_1 are looked for. So $I_2=4$ as in the traditional method and it means $I_2 \cdot m_{Q_2}=4 \cdot 7=28$ actions.
- In the iteration number 11 the check counting in the set Q_2 and the counting for the set $W_3=\{\emptyset\}$ happen. It means $m_{Q_2}+m_{W_3}=7+0=7$ actions.

Summary there are $o_2=8+28+18+28+7=89$ actions here.

Check: According to (7) $o_2=(1+4)+(7 \cdot (2+4)+7 \cdot (2+4))=89$.

According to (8) **the difference** between the amount of actions in case of the two algorithms is $o_1-o_2=(1+4+0) \cdot [(4+4)+1] + (7 \cdot [(4+4)-4]+7 \cdot [(4+4)-4])=101=190-89$.

It points out the efficient of the loop finder algorithm.

Discussion

The paper presents a new theoretical algorithm by combining two algorithms for different known problems. The result is very efficient and suitable for convert into practice as it can handle the features of the models used in applying software.

References

Ahuja, R., Magnati, T. & Orlin, J. (1993), *Network Flows:Theory, Algorithms and Applications.* Englewood Cliffs, NJ: Prentice Hall.

Bellman, R. (1958), *On a Routing Problem*, Quarterly of Applied Mathematics, 16(1), pp. 87-90.

Csordas, H. (2008), Longest Path Problem in Networks with Loops and Time Dependent Edge Lengths. Umag, Croatia, 2008.

Dijkstra, E. W. (1959), *A Note on Two Problems in Connexion With Graphs*, Numerische Mathematik, Vol.1 , S. pp. 269-271.

Franck, B., Neumann, K. & Schwindt, C. (2001), Project scheduling with calendars. OR Spektrum, Vol. 23., pp.325-334.

Vattai, Z. (1993), *Branch & Bound technika alkalmazása építőipari sorolási feladatok megoldására.* PhD Dissertation. Budapest University of Technology and Management.

Warshall, S. (1962), A Theorem on Boolean Matrices. Journal of the ACM, Vol 9.(Iss 1.), pp.11-12.