JOHANNES KEPLER
UNIVERSITY LINZ

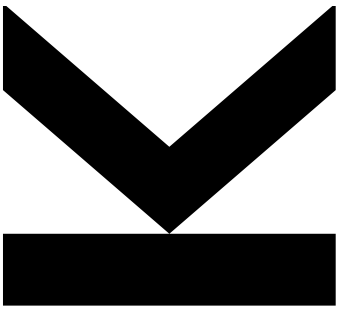
Author
Moritz Willnauer

**Research Institute
for Symbolic
Computation (RISC)**

Supervisor
**Assoc.Univ.-Prof. DI Dr.
Wolfgang Windsteiger**

March 2020

Modelling and Solving a Scheduling Problem by Max-Flow



Bachelor Thesis
for the degree of
Bachelor of Science

Study Programme:
Technische Mathematik

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place/Date

Signature

Abstract. This thesis shows how to specify and solve a specific scheduling problem of workgroups and loading orders. The sections on graph theory and network flows give support to understand the necessary definitions for modelling the scheduling problem as a network problem. For solving the resulting Max-Flow-Problem three different algorithms are presented. Two of them were implemented and tested with Mathmatica to analyse their performance. To do this an auxiliary algorithm for finding a shortest path for the modelled network of a scheduling problem was developed. Finally a pretty complex problem was solved by using Mathematica.

Statutory Declaration 2
1 Introduction 5
2 The Specification of the Scheduling Problem 6
2.1 A Small Sample Problem 6
2.2 The Specification of the Mathematical Problem 6
2.3 The Specification of the Sample Problem 7
3 Formulating the Scheduling Problem as a Network Problem 9
3.1 Graph Theory 9
3.2 Networks and Flows 9
3.3 Modelling the Scheduling Problem as a Max-Flow Problem 10
3.4 Formulating the Sample Problem as a Network Problem 11
4 Solving a Max-Flow Problem 13
4.1 The Ford-Fulkerson-Algorithm 13
4.2 The Correctness of the Ford-Fulkerson-Algorithm 19
4.3 The Edmonds-Karp-Algorithm 21
4.4 The Runtime of the Edmonds-Karp-Algorithm 22
4.5 Implementing and Testing the Edmonds-Karp-Algorithm 24
4.6 The Push-Relabel-Algorithm 29
4.7 Implementing and Testing the Push-Relabel-Algorithm 34
5 Solving Specific Scheduling Problems 36
5.1 Solving our Sample Problem 36
5.2 Solving a Complex Problem 37
6 Conclusion 38
7 Appendix 40
7.1 Implementations 40
7.2 The Workplan of the Complex Problem 44
7.3 The Abilities of the Workgroups 48
7.4 The Workingtimes 51

1 Introduction

The aim of this thesis is to develop a formal model for the following scheduling problem.

We have loading orders (tank cars have to be filled on certain charging stations with petroleum products), which should be finished within a specific working time. At the loading stations there are loading teams. Each order has its own requirements, so that not every workgroup can work on every order. This means that each workgroup has its own abilities. It is known which team can do which assignments. Every assignment has a known machining time and all assignments should be carried out within the specific working time. It is possible that more teams work on one assignment (if the assignment takes 4 hours, it is possible that one team works 3 hours and one team only 1 hour). So on the one hand the jobs can be handled sequentially and on the other hand they can also be executed in parallel. Find a workplan, which establishes the loading teams and their corresponding working time for every assignment.

In general, scheduling problems deal with the distribution of workers to orders by special properties. There are a lot of different formulations.

For modelling the given scheduling problem we need some necessary steps of specifications. First it is important to formulate the mathematical problem, to clarify what we are looking for by fullfilling which restrictions. For solving the specified problem we will formulate it as a network problem and find a valid solution of the scheduling problem by finding a Max-Flow of the specific network. It will be shown how to define such a network. Some sections deal with some theory, which is mainly based on [1]. For solving the problem three different algorithms are presented. Two of them were also implemented and tested with Mathematica. For one of these algorithm an auxiliary algorithm to find a shortest path of the modelled network of a scheduling problem was developed in the frame of this thesis. There are also lemmas that prove the correctness of the presented algorithms. Finally, we want to solve a pretty complex problem by the built-in Mathematica function *FindMaximumFlow* and present the result properly. For better understanding, this thesis presents how to use the gained definitions, on a small sample problem.

2 The Specification of the Scheduling Problem

Throughout this section we want to specify our scheduling problem so that we can solve it by Max-Flow. We will demonstrate all gained definitions on a small sample problem.

2.1 A Small Sample Problem

We have 3 workgroups, which can work 8 hours each. They have to manage 7 orders which take 2,2,11,2,1,3, and 2 hours to be finished. Moreover every workgroup has its own abilities, which means that the first workgroup can work on order 3,4,5 and 6, the second on 1,2,3,5,6 and 7, and the third on 1,2,3 and 5.
Of course one can solve this problem by hand, but the goal is to solve more complex problems like in Section 5.2.

2.2 The Specification of the Mathematical Problem

In this section a scheduling problem, described by words, should be specified as a mathematical model.

Given: m workgroups, as well as n orders with their corresponding working times $t_1, \ldots, t_n$. Moreover, we have m non empty sets $S_i \subseteq \{1, \ldots, n\}$ $i = 1, \ldots, m$, which give us information about, which orders S_i each workgroup i can handle. There is a maximal working time W for all workgroups as well. Of course $m, n \in \mathbb{N}$ and $W, t_1, \ldots, t_n \in \mathbb{R}_+$.

Find: $x_{ij} \in \mathbb{R}_0^+$, which tell us how long workgroup i should work on order j, s.t.

$$\underset{1 \leqslant i \leqslant m}{\forall} \sum_{j=1}^{n} x_{ij} \leqslant W, \qquad \text{(workingtime restrictions)}$$

$$\underset{1 \leqslant j \leqslant n}{\forall} \sum_{i=1}^{m} x_{ij} = t_j, \text{ and} \qquad \text{(every order has to be done)}$$

$$\underset{1 \leqslant i \leqslant m}{\forall} \underset{1 \leqslant j \leqslant n}{\forall} j \notin S_i \Rightarrow x_{ij} = 0. \qquad \text{(ability restrictions)}$$

Finally, we are looking for a $(m \times n)$-matrix, where we are only interested in the non zero entries. Now we specify our small problem from Section 2.1.

2.3 The Specification of the Sample Problem

For our sample problem we get

- $m = 3$ because we have three workgroups,
- $n = 7$ because of seven orders,
- $W = 8$ and
- $t_1 = 2, t_2 = 2, t_3 = 11, t_4 = 2, t_5 = 1, t_6 = 3$, and $t_7 = 2$.

Next we want to specify our sets S_1, S_2 and S_3.

- $S_1 = \{3, 4, 5, 6\}$, $S_2 = \{1, 2, 3, 5, 6, 7\}$ and $S_3 = \{1, 2, 3, 5\}$.

This is all information for the problem. Now we are looking for the entries of the (3×7)-matrix. The zero entries are coming from the ability restrictions.

$$\begin{pmatrix} 0 & 0 & x_{13} & x_{14} & x_{15} & x_{16} & 0 \\ x_{21} & x_{22} & x_{23} & 0 & x_{25} & x_{26} & x_{27} \\ x_{31} & x_{32} & x_{33} & 0 & x_{35} & 0 & 0 \end{pmatrix}$$

Now consider the matrix

$$\begin{pmatrix} 0\,0\,5\,2\,0\,1\,0 \\ 1\,0\,5\,0\,0\,1\,2 \\ 1\,2\,1\,0\,1\,0\,0 \end{pmatrix}.$$

Is this matrix a valid solution for the given problem above? To be a valid solution the entries of the matrix have to fulfill all three restrictions of the preceding specification.

By summing up the sixth column and the second row we see that

$$\sum_{i=1}^{3} x_{i6} = 1 + 1 = 2 < 3 = t_3 \qquad \text{and} \qquad \sum_{j=1}^{7} x_{2j} = 1 + 5 + 1 + 2 = 9 > 8.$$

According to these two unequalities we conclude that this matrix is not a valid solution. For this problem a valid solution would be

$$\begin{pmatrix} 0\,0\,3\,2\,0\,2\,0 \\ 1\,0\,4\,0\,0\,1\,2 \\ 1\,2\,4\,0\,1\,0\,0 \end{pmatrix}.$$

For the correctness of the solution we first have to check the sum of each row, to consider the workingtime restrictions. Thus we see that

$$\sum_{j=1}^{7} x_{1j} = 3+2+2 = 7 \leqslant 8,$$
$$\sum_{j=1}^{7} x_{2j} = 1+4+1+2 = 8 \leqslant 8, \text{ and}$$
$$\sum_{j=1}^{7} x_{3j} = 1+2+4+1 = 8 \leqslant 8.$$

Secondly, we must compute the sum of each column, to check whether each order can be done.

$$\sum_{i=1}^{3} x_{i1} = 1+1 = 2 = t_1 \qquad \sum_{i=1}^{3} x_{i5} = 1 = 1 = t_5$$
$$\sum_{i=1}^{3} x_{i2} = 2 = t_2 \qquad \sum_{i=1}^{3} x_{i6} = 2+1 = 3 = t_6$$
$$\sum_{i=1}^{3} x_{i3} = 3+4+4 = 11 = t_3 \qquad \sum_{i=1}^{3} x_{i7} = 2 = t_7$$
$$\sum_{i=1}^{3} x_{i4} = 2 = t_4$$

Out of this seven equations we can conclude that this solution is valid for the given problem. From now on we will focus on how to find such a valid solution.

3 Formulating the Scheduling Problem as a Network Problem

We will see that we can formulate our problem as a network problem. For this purpose we need some information about graph theory, networks and flows.

3.1 Graph Theory

A *(directed) graph* G is a pair (V, E) where V and E are non empty finite sets and $E \subseteq V \times V$. The elements v of V we call *vertices* and $e \in E$ we call *edges*. An edge $e = (v_1, v_2) \in E$, with $v_1, v_2 \in V$, *starts* starts at v_1 and *ends* at v_2. If $G = (V, E)$ is a graph then $V(G)$ and $E(G)$ refer to the sets of all vertices and edges, i.e. $V(G) = V$ and $E(G) = E$.

A *subgraph* of a graph G is a graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. Additionally, for $v \in V(G)$ we define

$$\delta_G^+(v) := \{(v, y) \in E(G) \mid y \in V(G) \setminus \{v\}\} \text{ and}$$
$$\delta_G^-(v) := \{(x, v) \in E(G) \mid x \in V(G) \setminus \{v\}\}.$$

This means that $\delta_G^+(v)$ with $v \in V(G)$ represents all edges of G that start at v and $\delta_G^-(v)$ is the set of all edges of G that end at v. This is shown graphically in Fig.1.

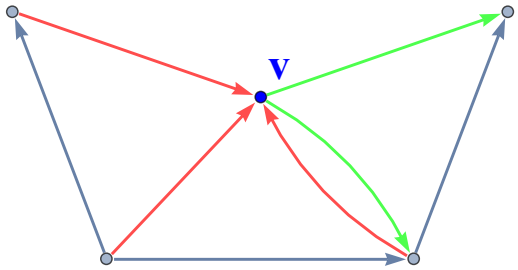


Fig. 1: $\delta_G^+(v)$ is the set of all green edges and $\delta_G^-(v)$ the set of all red edges.

3.2 Networks and Flows

Networks are used to model the transport of goods from one vertex in a graph to another vertex. We call (G, u, h, r) a network, where G is a directed graph and $u\colon E(G) \to \mathbb{R}_+$ is a function of G, which is called the *edge capacity function*. Throughout this thesis we call u the *edge capacities*. Furthermore, $h \in V(G)$ is

called the *source* and $r \in V(G)$ is called the *sink*. The main problem now is to transport as much units as possible from h to r. Thus we have to define a flow and the corresponding properties.

For a given network (G, u, h, r) we call a function $f\colon E(G) \to \mathbb{R}_+$ a *flow* of the network (G, u, h, r) *iff* $f(e) \leqslant u(e)$ for all $e \in E(G)$. The *excess* of f at $v \in V(G)$ is

$$ex_{f,G}(v) := \sum_{e \in \delta_G^-(v)} f(e) - \sum_{e \in \delta_G^+(v)} f(e).$$

A vertex $v \in V(G)$ *preserves* the flow f of (G, u, h, r) *iff* $ex_{f,G}(v) = 0$. Moreover, we call f an h-r-flow of (G, u, h, r) *iff* f is a flow of (G, u, h, r), $ex_{f,G}(h) \leqslant 0$ and $ex_{f,G}(v) = 0$ for all $v \in V(G) \setminus \{h, r\}$. The last condition we call the *flow preservation rule*. Finally, we call

$$val_{(G,u,h,r)}(f) := -ex_{f,G}(h)$$

the *value* of the h-r-flow f of the network (G, u, h, r). Sometimes when it is clear which network is meant, we will only write $val(f)$.

Now we are able to formulate the Maximum-Flow-Problem.

The Maximum-Flow Problem

Given: A network (G, u, h, r).

Find: An h-r-flow f of (G, u, h, r) of maximum value, i.e. for all other h-r-flows f^* of (G, u, h, r), $val_{(G,u,h,r)}(f^*) \leqslant val_{(G,u,h,r)}(f)$.

In Section 4 we will talk about h-r-flows of maximum values and ways to find them, in more detail. Moreover, we will demonstrate this issue on an example later on.

3.3 Modelling the Scheduling Problem as a Max-Flow Problem

To attain the goal of solving a scheduling problem by Max-Flow, we will now formulate our scheduling problem as a network problem. To do this we create vertices $A_1, A_2, \ldots, A_m$ which stand for the m workgroups. The same we do for each order. Thus we get the vertices $W_1, W_2, \ldots, W_n$. Now we can formulate the scheduling problem as a network problem.

Given: A network (G, u, h, r) with $V(G) = \{h, A_1, \ldots, A_m, W_1, \ldots, W_n, r\}$, $E(G) = E_1 \cup E_2 \cup E_3$, where

$$E_1 = \{(h, A_i) \mid i = 1, \ldots, m\},$$
$$E_2 = \{(A_i, W_j) \mid i = 1, \ldots, m \text{ and } j \in S_i\}, \text{ and}$$
$$E_3 = \{(W_j, r) \mid j = 1, \ldots, n\}$$

and $u\colon E(G) \to \{t_1, \ldots, t_n, W\}$ are the edge capacities defined as

$$u(e) = \begin{cases} W & \text{if } e \in E_1 \cup E_2, \\ t_j & \text{if } e = (W_j, r) \in E_3. \end{cases}$$

Find: An h-r-flow f of (G, u, h, r) of maximum value.

Considering the definitions above the x_{ij} from the specification of the scheduling problem are exactly the values of $f(e)$ with $e \in E_2$. This means that a valid solution for our scheduling problem is given by a Max-Flow of the modelled network. This is because an h-r-flow of maximum value transports as much units as possible from h to r. For the scheduling problem this means that the workgroups should work as much as possible. The value of the h-r-flow is restricted by the sum of all working times of the orders. Ideally, the flow reaches these restriction and so there should be no doubt that this solution of the network problem is also a solution of the scheduling problem.

Now we need algorithms to find such a Max-Flow. For solving the problem by Mathematica we use a $((n + m + 2) \times (n + m + 2))$- capacity matrix C s.t. $c_{ij} = u((v_i, v_j))$ if $(v_i, v_j) \in E(G)$ and $c_{ij} = 0$ otherwise. But before talking about solving a network problem, we want to formulate the example problem from Section 2.3 as a network problem like shown above.

3.4 Formulating the Sample Problem as a Network Problem

We have already done all preparations for our sample problem, so that we can start with the network definitions immediately. Firstly, we set
$V = \{h, A_1, A_2, A_3, W_1, W_2, W_3, W_4, W_5, W_6, W_7, r\}$.

Secondly, we define E_1, E_2 and E_3.

$$E_1 = \{(h, A_1), (h, A_2), (h, A_3)\},$$
$$E_2 = \{(A_1, W_3), (A_1, W_4), (A_1, W_5), (A_1, W_6),$$
$$(A_2, W_1), (A_2, W_2), (A_2, W_3), (A_2, W_5), (A_2, W_6), (A_2, W_7),$$
$$(A_3, W_1), (A_3, W_2), (A_3, W_3), (A_3, W_5)\}, \text{ and}$$
$$E_3 = \{(W_1, r), (W_2, r), (W_3, r), (W_4, r), (W_5, r), (W_6, r), (W_7, r)\}.$$

From this we get our set of edges E. Up next we get the capacities for all edges.

$u((W_1, r)) = 2$	$u((W_2, r)) = 2$	$u((W_3, r)) = 11$	$u((W_4, r)) = 2$
$u((W_5, r)) = 1$	$u((W_6, r)) = 3$	$u((W_7, r)) = 2$	

All the other edges from h to the workgroups or from the workgroups to the specific orders get the capacity 8.

$u((h, A_1)) = 8$	$u((h, A_2)) = 8$	$u((h, A_3) = 8$	
$u((A_1, W_3)) = 8$	$u((A_1, W_4)) = 8$	$u((A_1, W_5)) = 8$	$u((A_1, W_6)) = 8$
$u((A_2, W_1)) = 8$	$u((A_2, W_2)) = 8$	.....	$u((A_3, W_5)) = 8$

Finally, we can define our capacity matrix C.

$$C = \begin{pmatrix}
0 & 8 & 8 & 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 8 & 8 & 8 & 8 & 0 & 0 \\
0 & 0 & 0 & 0 & 8 & 8 & 8 & 0 & 8 & 8 & 8 & 0 \\
0 & 0 & 0 & 0 & 8 & 8 & 8 & 0 & 8 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 11 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

As a result of all definitions we get with $G = (V, E)$ our network (G, u, h, r). This network is shown graphically in Fig.2.

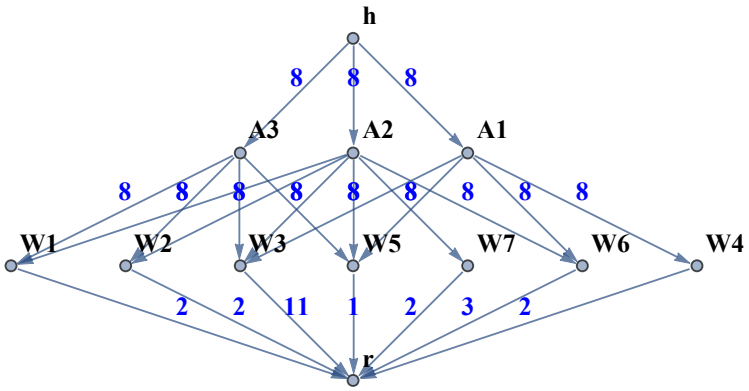


Fig. 2: The network (G, u, h, r) of the sample problem.

4 Solving a Max-Flow Problem

In Section 3 we have modelled a scheduling problem as a Max-Flow problem. Now we investigate some methods to find such flows of maximum value. To understand the problem of finding such a flow a small sample problem is presented in Fig. 3 . This network can be seen as a scheduling problem of two workgroups and three orders. Now our goal is to solve this problem. We can start to send as much flow as needed successively from the first workgroup to the first ordess. This means we set the flow of the edges $(h, A1), (A1, W1)$ and $(W1, r)$ to 8. As a second step we set the flow of the edges $(h, A2), (A2, W3)$ and $(W3, r)$ to 6. Now we cannot send more flow from h to r. This resulting flow is displayed in the left figure of Fig. 4 . But is this a flow of maximum value? If you compare the figures of Fig. 4 you will see that this is not the case. In this section three different algorithms to find flows of maximum value are presented and investigated.

4.1 The Ford-Fulkerson-Algorithm

Ford and Fulkerson developed this algorithm to find a flow of maximum value of a network. The first version of this algorithm was released in 1956, see [2]. This algorithm was the first published procedure to solve the Max-Flow-Problem. Before we can present the algorithm we need some more details.

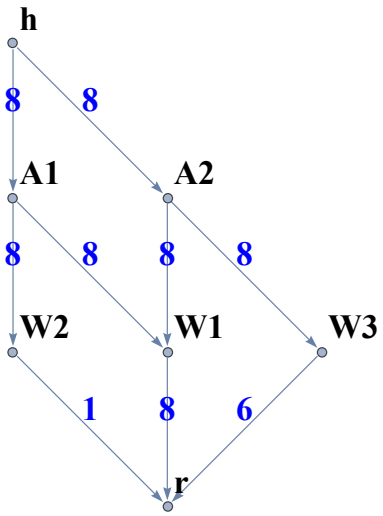


Fig. 3: A sample problem for finding a Max-Flow.

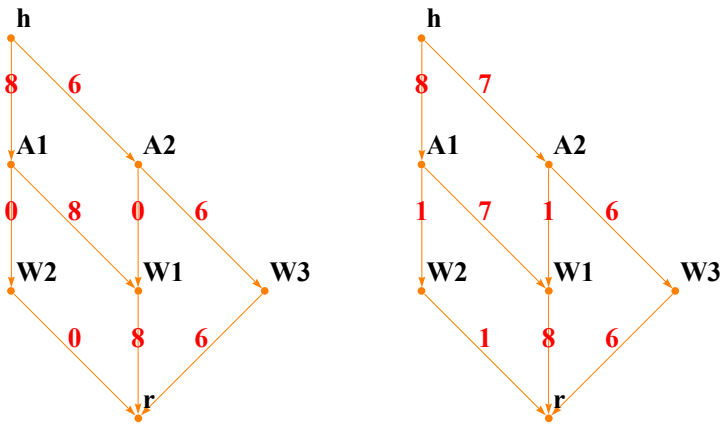


Fig. 4: Left: An h-r-flow for Fig. 3 *NOT* of maximum value. Right: An h-r-flow for Fig. 3 of maximum value.

Let G be again a directed graph. For $e = (u, w) \in E(G)$, we call $\overleftarrow{e} = (w, u)$ the *contrary edge* of e. Furthermore, we define

$$\overleftrightarrow{G} := (V(G), E(G) \cup \{\overleftarrow{e} \mid e \in E(G)\}).$$

Considering edge capacities $u\colon E(G) \to \mathbb{R}_+$ of G and a flow f of (G, u, h, r), we introduce the *residual capacities* $u_f\colon E(\overleftrightarrow{G}) \to \mathbb{R}_+$ of u w.r.t. f, with

$$u_f(e) := u(e) - f(e) \text{ and } u_f(\overleftarrow{e}) := f(e) \text{ for all } e \in E(G).$$

Additionally, we get the *residual graph*

$$G_f := (V(G), \{e \in E(\overleftrightarrow{G}) \mid u_f(e) > 0\})$$

of G w.r.t. f.

To outline the Ford-Fulkerson-Algorithm we also have to clarify the notion of paths. In this thesis we call $P = (v_1, \ldots, v_n)$ a *path* on G *iff* $(v_i, v_{i+1}) \in E(G)$ for $i = 1, \ldots, n-1$. We say P starts at v_1 and ends at v_n. Again $V(P) = \{v_1, \ldots, v_n\}$ and $E(P) = \{(v_1, v_2), \ldots, (v_{n-1}, v_n)\}$. The length of a path is the number of edges visited, namely $|E(P)|$. We say for a graph G and vertices $v, w \in G$, the vertex v is reachable from w in G *iff* there is a path P in G, s.t. P starts at w and ends at v.

Now for a given flow f of a network (G, u, h, r) and a path P on G_f we can *augment the flow* f along P by $\gamma \in \mathbb{R}_+$, which means that we increase $f(e)$ by γ for all $e \in E(P) \cap E(G)$ and decrease $f(\overleftarrow{e})$ by γ if $e \in E(P) \setminus E(G)$. We call P an f-augmenting path of G_f *iff* P starts at h and ends at r and there exist some $\gamma \in \mathbb{R}_+$ s.t. we can augment f along P by γ.

Lemma 1. *For an h-r-flow f of (G, u, h, r). Let f^* be an h-r-flow of the network s.t. you get f^* by augmenting f along an f-augmenting path P. Then*

$$val_{(G,u,h,r)}(f^*) > val_{(G,u,h,r)}(f).$$

Lemma 1 tells us that we can increase an h-r-flow f by looking for f-augmenting paths. Out of this we can formulate the Ford-Fulkerson-Algorithm.

The Ford-Fulkerson-Algorithm

Input: A network (G, u, h, r), with $u\colon E(G) \to \mathbb{N}$.

Output: An h-r-flow f of (G, u, h, r) of maximum value.

1. Set $f(e) := 0$ for all $e \in E(G)$.
2. Take a f-augmenting-path P of G_f. *If* no such path exists *then stop*.
3. Calculate $\gamma := \min_{e \in E(P)} u_f(e)$, augment f along P by γ and *go to* 2.

The Ford-Fulkerson-Algorithm only allows integer capacities. This is because if we allow irrational capacities, the algorithm may will fail to terminate. For this fact you can look at an example in [3]. For a better understanding of the algorithm, we solve a small example by using the Ford-Fulkerson-Algorithm. For this example the directed graph G and the corresponding graph $\overleftrightarrow{G}$ (the contrary edges are colored red) are given in Fig. 5 .

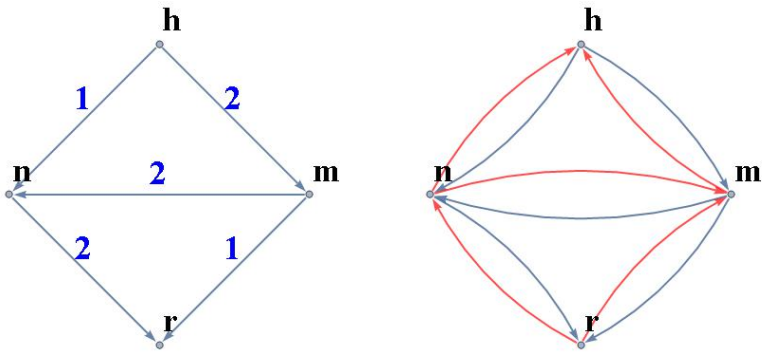


Fig. 5: Left: The example network (G, u, h, r). Right: The graph $\overleftrightarrow{G}$.

Step by step we will consider G_{f_k}, f_k and the path P_k we have found for step k.

At the beginning we get that $G_{f_0} = G$ because our flow (see Fig.6) is zero and therefore $u_f(\overleftarrow{e}) = f(e) = 0$ for $e \in E(G)$. The chosen path P_0 (see Fig.7) is the red marked one. The minimum of the residual capacities of the edges $e \in E(P_0)$ is $\gamma = 2$. Now let us augment the flow f_0 along P_0 by γ so that we get f_1.

After changing the flow we get the residual graph G_{f_1} for the new flow f_1 (see Fig.8). Again we are looking for a path P_1 (see Fig.9) on G_{f_1} from h to r. There is such a path and so we have to calculate $\gamma = 1$. Now we have to augment the flow f_1 along P_1.

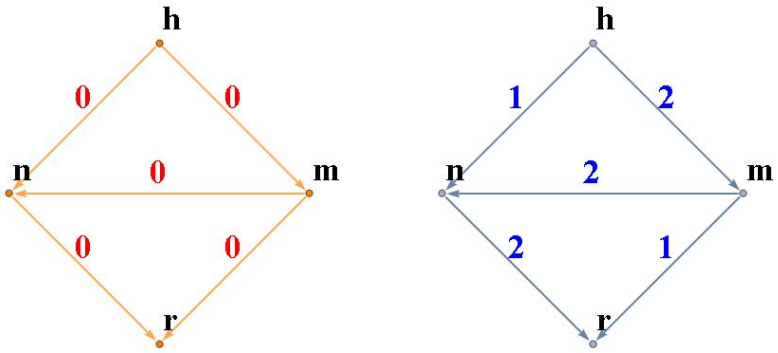


Fig. 6: Left: The flow f_0. Right: The network G_{f_0}.

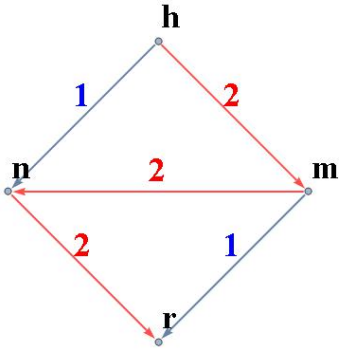


Fig. 7: The path P_0.

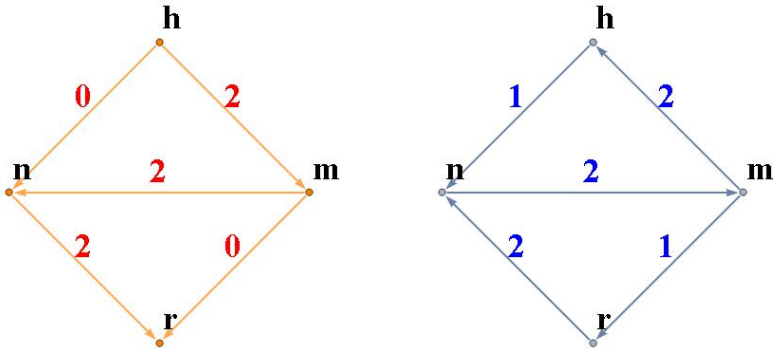


Fig. 8: Left: The flow f_1. Right: The network G_{f_1}.

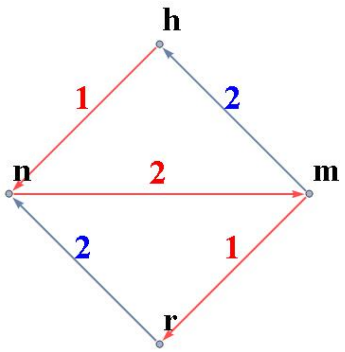


Fig. 9: The path P_1.

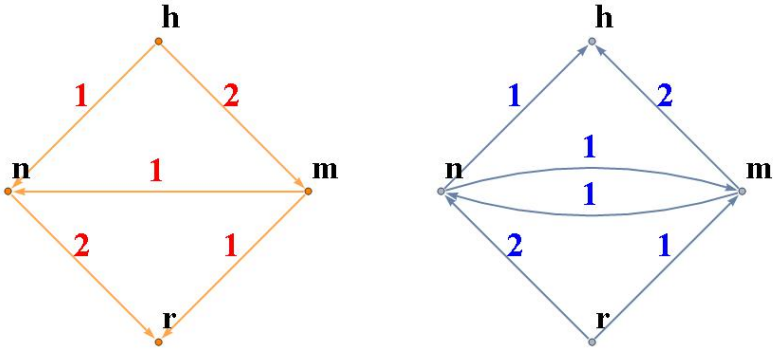


Fig. 10: Left: The flow f_2. Right: The network G_{f_2}.

For G_{f_2} we can find no more path from h to r (see Fig. 10). As a consequence the algorithm terminates and as a result we get our final flow f_2 (see Fig.10) of step 2 with $val_{(G,u,h,r)}(f) = 3$.

4.2 The Correctness of the Ford-Fulkerson-Algorithm

We are going to show that, if the Ford-Fulkerson-Algorithm terminates, the flow will be of maximum value. Before we need some more definitions and two lemmas. For all the lemmas, theorems and their proofs, except the ones about the algorithm for finding a shortest path, there is a similiar version in [1].

For a directed graph G and $X, Y \subseteq V(G)$ we define

$$E_G^+(X,Y) := \{(x,y) \in E(G) \mid x \in X/Y \text{ and } y \in Y/X\}.$$

Furthermore, we can define

$$\delta_G^+(X) := E^+(X, V(G) \setminus X) \text{ and } \delta_G^-(X) := \delta_G^+(V(G) \setminus X)).$$

In Fig.11 you see a small example of the definitions above.

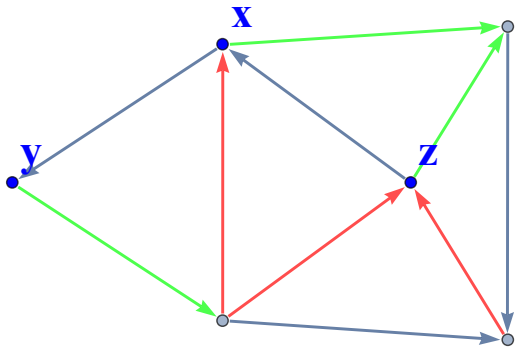


Fig. 11: $\delta_G^+(X)$ (the green edges) and $\delta_G^-(X)$ (the red edges) for $X = \{x, y, z\}$.

Lemma 2. *Given a network (G, u, h, r). Then for all $A \subseteq V(G)$, with $h \in A$ and $r \notin A$, and every h-r-flow f, we get that*

$$val_{(G,u,h,r)}(f) = \sum_{e \in \delta_G^+(A)} f(e) - \sum_{e \in \delta_G^-(A)} f(e).$$

Proof. By the flow preservation rule we get

$$\begin{aligned} val_{(G,u,h,r)}(f) &\overset{(1)}{=} -ex_{f,G}(h) \\ &\overset{(2)}{=} \sum_{v \in A} -ex_{f,G}(v) \\ &\overset{(3)}{=} \sum_{v \in A} \left(\sum_{e \in \delta_G^+(v)} f(e) - \sum_{e \in \delta_G^-(v)} f(e) \right) \\ &\overset{(4)}{=} \sum_{e \in \delta_G^+(A)} f(e) - \sum_{e \in \delta_G^-(A)} f(e). \end{aligned}$$

At (1) we only use the definition of $val_{(G,u,h,r)}(f)$. Step (2) is allowed because f preserves the flow at every vertex $v \in A \setminus \{h\}$. Thus all added terms of the sum are 0. Again we use the definition of the excess at (3). For step (4) we assume that $A = \{v_1, \ldots, v_n\}$. We know that for all edges $e = (v_i, v_j)$, $e \in \delta_G^+(v_i)$ and $e \in \delta_G^-(v_j)$. We can conclude that all edges that are not in $\delta_G^+(A)$ and not in $\delta_G^-(A)$ cancel out. Thus we only have to consider the edges from $\delta_G^+(A)$ and $\delta_G^-(A)$. □

Lemma 3. *Considering a network (G, u, h, r) again. For every set of vertices $A \subseteq V(G)$ with $h \in A$, but $r \notin A$ and every h-r-flow f, we get that*

$$val_{(G,u,h,r)}(f) \leq \sum_{e \in \delta_G^+(A)} u(e).$$

Proof. We know that $0 \leq f(e) \leq u(e)$ for all $e \in E(G)$ and the equality of Lemma 2 holds for A because all conditions are fulfilled. As a consequence we get

$$val_{(G,u,h,r)}(f) = \sum_{e \in \delta_G^+(A)} f(e) - \sum_{e \in \delta_G^-(A)} f(e) \overset{(1)}{\leq} \sum_{e \in \delta_G^+(A)} f(e) \overset{(2)}{\leq} \sum_{e \in \delta_G^+(A)} u(e).$$

The inequality (1) holds because $f(e) \geq 0$ for all $e \in E(G)$ and inequality (2) because $f(e) \leq u(e)$ for all $e \in E(G)$. □

Now we are able to prove the theorem that shows why the Ford-Fulkerson-Algorithm computes a flow of maximum value.

Theorem 1. *An h-r-flow f of the network (G, u, h, r) is of maximum value iff there is no other f-augmenting path in (G, u, h, r).*

Proof. "⇒": Proof by contraposition. Let us assume that there is an f-augmenting path P_0 on G_f. By applying one step of the Ford-Fulkerson-Algorithm we can augment our flow f along P_0 and so f is not a flow of maximum value because after augmentation we get a flow of strictly higher value.

"⇐": Now assume that there is no f-augmenting path. That means that we cannot go from h to r on G_f. Let B be the set of all vertices that are reachable from h on G_f. By the definiton of the residual capacities we get

$$f(e) = u(e) \quad \text{for all } e \in \delta_G^+(B) \quad \wedge \quad f(e) = 0 \quad \text{for all } e \in \delta_G^-(B). \qquad (1)$$

Otherwise there would be a vertex $v_0 \in V(G_f)$, s.t. there is a path from h to v_0 on G_f. Finally, we have that $B \subseteq V(G)$, $h \in B$, but $r \notin B$ and f is an h-r-flow of (G, u, h, r). Considering this, all conditions of Lemma 2 are fulfilled and we get

$$val_{(G,u,h,r)}(f) = \sum_{e \in \delta_G^+(B)} f(e) - \sum_{e \in \delta_G^-(B)} f(e) \overset{(1)}{=} \sum_{e \in \delta_G^+(B)} u(e).$$

As a result of Lemma 3 we get that for all h-r-flows f^* of (G, u, h, r), $val_{(G,u,h,r)}(f^*) \leqslant val_{(G,u,h,r)}(f)$.Thus f is an h-r-flow of maximum value. □

But what about the runtime of the algorithm? If you take a close look at Fig.12 you will understand the problem. Each time you choose the path from right figure Fig.12 at step (2) of Ford-Fulkerson, we can only augment our flow by 1. After augmenting the flow along the chosen path, the edge with capacity 1 will now go from m to n. If you choose again the path of length 3 you can increase your flow by only 1 again.

4.3 The Edmonds-Karp-Algorithm

The Edmonds-Karp-Algorithm avoids such problems. This algorithm is similar to the Ford-Fulkerson-Algorithm but chooses instead of an abitrary f-augmenting path at step (2) of the algorithm, the shortest one. This algorithm was invented in 1969 from Edmonds and Karp, see [2].

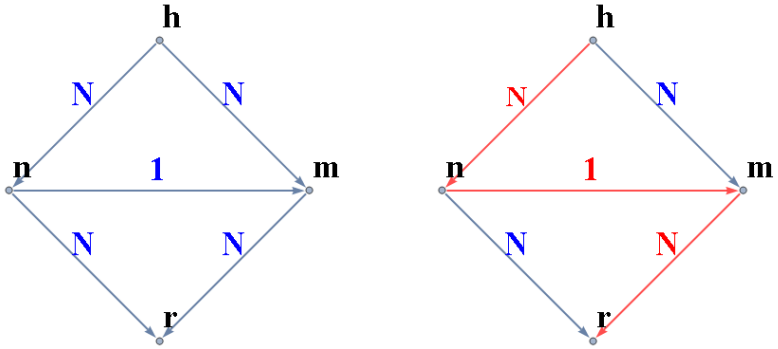


Fig. 12: Left: A possible network (G, u, h, r). Right: A possible path P.

The Edmonds-Karp-Algorithm

Input: A network (G, u, h, r).

Output: An h-r-flow of (G, u, h, r) of maximum value.

1. Set $f(e) := 0$ for all $e \in E(G)$.
2. Take a shortest f-augmenting-path P. *If* no such path exists *then stop*.
3. Calculate $\gamma := \min_{e \in E(P)} u_f(e)$, augment f along P by γ and *go to* 2.

4.4 The Runtime of the Edmonds-Karp-Algorithm

By this idea, using the shortest path, Edmonds and Karp found the first polynomial algorithm for the Max-Flow-Problem. To prove this, we need a lemma again.

Lemma 4. *Let (G, u, h, r) be a network and $f_1, f_2, \ldots$ be h-r-flows of the network, s.t. you get f_{i+1} by augmenting f_i along the shortest f_i-augmenting path P_i. Additionally, $E(P_k) \cup E(P_l)$ contains a pair of contrary edges, i.e. for some $e \in E(G)$: $e, \overleftarrow{e} \in E(P_k) \cup \in E(P_l)$. Then we have for all $k \leq l$*

$$|E(P_k)| + 2 \leqslant |E(P_l)|.$$

Proof. You can find this proof in [1].

As a result we can show an important theorem for the Edmonds-Karp-Algorithm. Later we will deal in detail with the exactness of the following estimation.

Theorem 2. *Given a network (G, u, h, r). The Edmonds-Karp-Algorithm stops after a maximum of $\frac{mn}{2}$ steps, where m is the number of edges and n the number of vertices of the graph G.*

Proof. Let (G, u, h, r) be a network and let $n = |V(G)|$ and $m = |E(G)|$. Let $P_1, P_2, \ldots$ be the shortest paths genereted by Edmonds-Karp. For all these paths there is at least one edge $e \in E(P_i)$ such that $u_{f_i}(e) = \gamma_i$. This edges at each step of the algorithm are named Bottleneck-edges. For an abitrary edge e_0 let $P_{i_1}, P_{i_2}, \ldots$ be the subsequence of shortest f_{i_t}-augmenting paths, for which e_0 is Bottleneck. Now we choose j arbitrary but fix. Between two paths P_{i_j} and $P_{i_{j+1}}$ there has to be an $\overleftarrow{e_0}$ consisting path P_k, with $i_j \leqslant k \leqslant i_{j+1}$. This is because after the augmentation along P_{i_j}, $u_f(e_0) = 0$. If there had not been the path P_k, no augmentation along e_0 would has been possible again. Moreover, we have seen that $e_0 \in P_{i_j}$ and $\overleftarrow{e_0} \in P_k$. So we get that $\overleftarrow{e_0}$, $e_0 \in P_{i_j} \cup P_k$ and $\overleftarrow{e_0}$, $e_0 \in P_k \cup P_{i_{j+1}}$. Now we can use Lemma 4 because $i_j \leqslant k \leqslant i_{j+1}$ holds as well and we get the following inequality.

$$|E(P_{i_j})| + 2 \leqslant |E(P_k)| \ \wedge \ |E(P_k)| + 2 \leqslant |E(P_{i_{j+1}})|$$

$$\Rightarrow |E(P_{i_j})| + 4 \leqslant |E(P_k)| + 2 \leqslant |E(P_{i_{j+1}})| \tag{2}$$

Case 1: We consider the case that e_0 neither starts at h nor ends at r. We can make the estimation that

$$3 \leqslant |E(P_{i_j})| \leqslant n - 1.$$

As a result from (2) we know that at each step $|E(P_{i_j})|$ grows at least by 4. For e_0, not starting at h or ending at r, we get a maximum of $\frac{n}{4}$ augmentations.

Case 2: On the contrary, let e_0 start at h. Because there will be no flow that comes back to h along $\overleftarrow{e_0}$ there will be one shortest f-augmenting path that contains e_0 as Bottleneck at most.

Case 3: Moreover, if e_0 ends at r there will be no flow away from r. Again, there will be at most one shortest f-augmenting path that contains e_0 as Bottleneck again. We conclude that for these edges the maximum of $\frac{n}{4}$ augmentations

holds as well.

In fact, for every shortest f-augmenting path, at least one edge of $\overleftrightarrow{G}$ has to be Bottleneck. For this purpose the maximal number of augmentations is

$$|E(\overleftrightarrow{G})|\frac{n}{4} = 2m\frac{n}{4} = \frac{mn}{2}. \quad \square$$

Theorem 2 proves that the algorithm stops after at most $\frac{mn}{2}$ augmentations. So the algorithm will always come to an end for a given problem. But does the algorithm really needs this estimated number of steps?

4.5 Implementing and Testing the Edmonds-Karp-Algorithm

Mathematica provides an implemented algorithm for solving a Max-Flow problem, mainly the *FindMaximumFlow* function. Using different options you can get the *FlowMatrix*, the *FlowGraph*, the *FlowValue*, etc.. Most time you will deal with the *FlowMatrix*, because it gives information about the flows between the various vertices. Unfortunately, Mathematica does not provide the option to use different algorithms to solve the task. That is why this thesis provides the implementations of two different algorithms in order to compare the different runtimes and results.

For implementing the Edmond-Karp-Algorithm we have to define an auxiliary algorithm that computes the shortest f-augmenting path. Additionally, we need functions that update the flow and the residual capacities. To find the shortest f-augmenting paths on residual graphs that result from modelling a scheduling problem, an own algorithm was developed. For a faster computation the algorithm uses the special structure ot networks that represent a scheduling problme. For this networks only the edges between workers and orders are interesting. The particular steps are listed below. For the following algorithm, $\asymp$ stands for the operation that joins vertices and/or paths to a new path.

Find a Shortest f-augmenting Path for the Modelled Network of a Scheduling Problem

Input: A network (G, u, h, r) and an h-r-flow f of (G, u, h, r).

Output: A shortest f-augmenting path $P = (h, \ldots, r)$ if one exists. Otherwise the algorithm stops and returns $P = ()$.

1. – Set $V = \{h\}$
   – Set $A = \{a \in V(G_f) \mid (h,a) \in \delta^+_{G_f}(h)\}$.
   – Set $W = \{w \in V(G_f) \mid (w,r) \in \delta^-_{G_f}(r)\}$.
   if $A = \{\} \vee W = \{\}$ *then stop* because no path exists and return $P = ()$.
   – Let $p : V(G_f) \to \{p \mid p \text{ is a path of } G_f\}$
   – Set $p(a) = (h)$ for all $a \in V(G_f)$.
2. *While* $A \neq \{\}$ *do*:
   Check if there is an edge $e = (a^*, w^*) \in E(G_f)$ with $a^* \in A$ and $w^* \in W$.
   If no such edge exists
   - $V = V \cup A$
   - Set $B = \{b \in V(G_f) \mid (a,b) \in \delta^+_{G_f}(a) \text{ for some } a \in A\} \setminus V$
   - For all $b \in B$, with $(a,b) \in \delta^+_{G_f}(a)$ for some $a \in A$, set $p(b) = p(a) \asymp a$.
   - Set $A = B$
   else stop and return $P = p(a^*) \asymp a^* \asymp w^* \asymp r$.
   Stop because no path exists and return $P = ()$.

The algorithm starts by defining a set A of all vertices that are reachable from h and defines a set W of all edges from where you can reach r. The set V should store all vertices of the network from where you cannot reach a vertex of W. Now at each step the algorithm will check, if you can already reach a vertex of W. If so, the algorithm will stop. Otherwise the algorithm computes a new set of vertices which are reachable from any vertex of A. Now our A is updated to all new computed vertices without the previous ones, because you already know that from a previous vertex you cannot reach a vertex of W. Furthermore, this condition avoids that you loop between two vertices that are connected by an edge and its contrary one. Finally, you save the preceding path for every vertex of A.

Now we proceed by showing that the algorithm will supply a shortest f-augmenting path if possible.

Theorem 3. *The algorithm for finding a shortest f-augmenting path of the modelled network of a scheduling problem is correct.*

Proof. Finding a shortest f-augmenting path for the network means that we have to find a shortest path for G_f. Let us consider the sequences $V_0, V_1, V_2, \ldots$ of V and $A_0, A_1, A_2, \ldots$ of A. First we want to show that if the algorithm com-

putes a path P, it will be a correct one from h to r. At each step of the algorithm we only add a new vertex to the path P that is reachable from the previous one. As a result the path P computed by the algorithm is a correct one. Now we assume that the algorithm stops without finding a path. Now we want to show that there cannot be a path from h to r. This means there is an index s such that $A_s = \{\}$. Hence, in the previous step we got that

$$\{b \in V(G_f) \mid (a,b) \in \delta^+_{G_f}(a) \text{ for some } a \in A_{s-1}\} \subseteq V_s = \{h\} \cup \bigcup_{i=0}^{s-1} A_i.$$

This means that all new reachable vertices were already tested if there is an edge to a vertex of W. Furthermore, all reachable vertices from h were tested as well. In conclusion we can say that either the algorithm produces a f-augmenting path for the modelled network of a scheduling problem, or there is no such path at all.

As a next step we want to show that the computed path is a shortest one. Let $P = (h, a_1, \ldots, a_m, r)$ be a path computed by the algorithm again. Now we assume that there is a strictly shorter path $P_0 = (h, v_1, \ldots, v_n, r)$ with

$$|E(P_0)| = n + 1 < m + 1 = |E(P)|. \tag{3}$$

It should be clear that $v_1 \in A_0$ and $v_n \in W$. For the path P we have that

$$a_i \in A_{i-1} \text{ for } i = 1, \ldots, m-1.$$

Case 1: We see that if $v_j \in A_{j-1}$ for $j = 1, \ldots, n-1$, there will be an edge from A_{n-2} to W. Now there will be an index $k < m-2$ such that $A_k = A_{n-2}$. This is a contradiction to the condition of the algorithm because there is an edge from A_k to W but the algorithm proceeds.

So we can conclude that there is a smallest index $2 \leqslant t \leqslant n-1$ such that

$$\text{for all } j < t: \quad v_j \in A_{j-1} \quad \wedge \quad v_t \notin A_{t-1}.$$

From the condition $v_t \notin A_{t-1}$ we get that

$$(v_t \notin A_{t-1} \quad \wedge \quad v_t \in V_{t-1}) \quad \vee \quad (v_t \notin A_{t-1} \quad \wedge \quad v_t \notin V_{t-1}).$$

If the first condition holds we will get a set A_z with $z \leqslant t-2$ such that $v_t \in A_z$. But we know that $v_z \in A_{z-1}$. As a consequence of the choice of the A_i's we get that there has to be an edge $e = (v_z, v_t) \in E(G_f)$. Therefore, we can define the path $P^* = (h, v_1, \ldots, v_z, v_t, \ldots, r)$. Thus we get that

$$|E(P^*)| \leqslant n < n + 1 = |E(P_0)|, \quad \lightning$$

which is a contradiction to the assumption that P_0 is a shortest path.

Case 2: We have that P_0 is a shortest path and therefore there has to be an edge $e = (v_{t-1}, v_t) \in \delta^+_{G_f}(v_{t-1})$ with $v_{t-1} \in A_{t-2} \neq \{\}$. Furthermore, there is no edge from A_{t-2} to W, because otherwise P_0 would not be a shortest path. We get that

$$v_t \in \{b \in V(G_f) \mid (a,b) \in \delta^+_{G_f}(a)) \text{ for some } a \in A_{t-2}\} \quad \wedge \quad v_t \notin V_{t-1}$$
$$\Rightarrow v_t \in \{b \in V(G_f) \mid (a,b) \in \delta^+_{G_f}(a) \text{ for some } a \in A_{t-2}\} \setminus V_{t-1} = A_{t-1}. \quad \lightning$$

As a result we get that P is a shortest f-augmenting path for the modelled network of a scheduling problem. □

We can show even more for the presented algorithm. The following theorem will show you that the algorithm stops after a maximum of $n - 3$ steps.

Theorem 4. *The presented algorithm for finding a shortest f-augmenting path of the modelled network of a scheduling problem terminates after a maximum of $n - 3$ steps, where $n = |V(G_f)|$.*

Proof. Let $V_0, V_1, V_2, \ldots$ be the genereted sequence of sets V and $A_0, A_1, A_2, \ldots$ the one of A. Now we consider the case that the algorithm is looping. We know that $|V_0| = 1$ and $|W| \geqslant 1$ because otherwise the algorithm will stop. Notice that $A_i \cap W = \{\}$ during the whole algorithm. While the algorithm is looping $|A_i| \geqslant 1$ because $A_i \neq \{\}$ and $A_i \cap V_i = \{\}$. Thus, $|V_{i+1}| \geqslant |V_i| + 1$ and we get that $|V_{n-3}| \geqslant n - 2$. We know that $r \notin A_i, V_i, W$ and $|V_{n-3}| + |A_{n-3}| + |W_{n-3}| + 1 \leqslant n$ because all these sets are pairwise distinct. Now we get that

$$|V_{n-3}| + |A_{n-3}| + |W_{n-3}| + 1 \geqslant n - 3 + 1 + 1 + 1 = n + 1 > n. \quad \lightning$$

As a logical consequence we get that

$$|A_{n-3}| = 0 \quad \vee$$
$$\text{there is an edge } e = (a^*, w^*) \in E(G_f) \text{ with } a^* \in A_{n-4} \text{ and } w^* \in W.$$

So we get that the algorithm terminates after a maximum of $n - 3$ loops. □

Now we will compare the implemented Edmond-Karp-Algorithm and the provided *FindMaximumFlow* function with respect to the solution. For small problems the runtimes are similar but most time we are getting different results. Take a look at the problem in Fig.13 and compare the two different results in Fig. 14 and Fig. 15. Both results present a flow of maximum value, but use different edges and different flows.

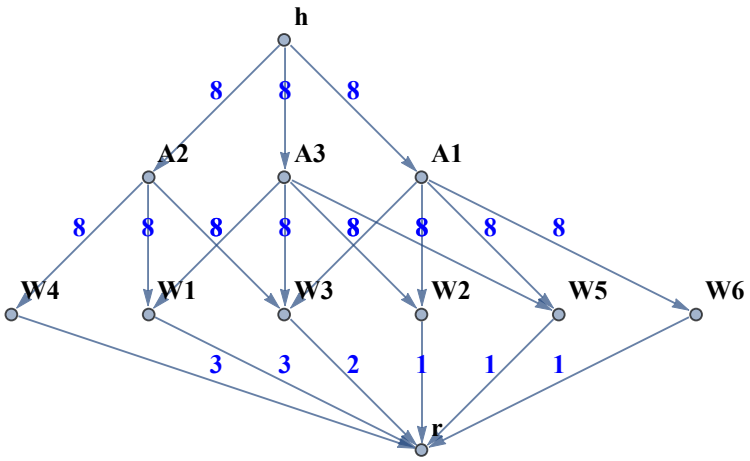


Fig. 13: An example that has to be solved by *FindMaximumFlow* and the Edmonds-Karp-Algorithm.

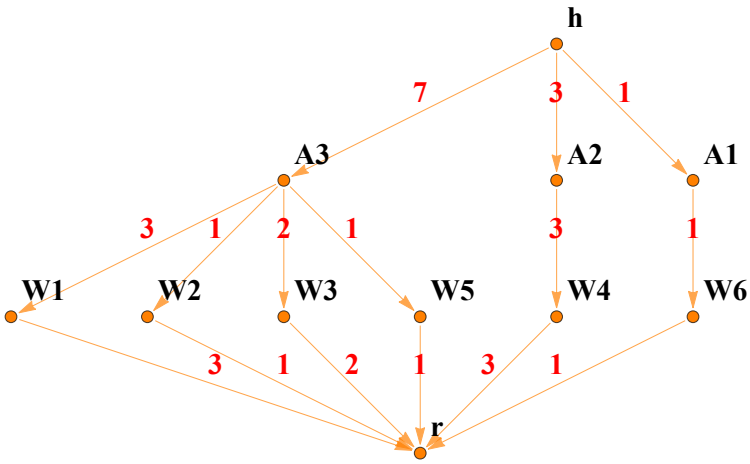


Fig. 14: The result of the problem in Fig.13 by using *FindMaximumFlow*.

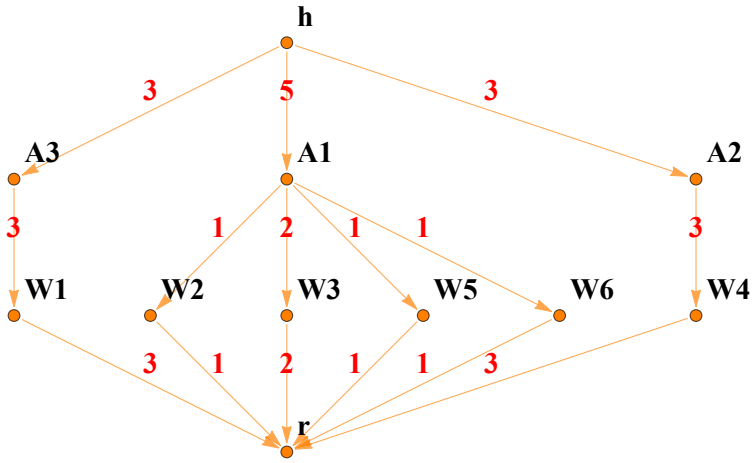


Fig. 15: The result of the problem in Fig.13 by using the Edmonds-Karp-Algorithm.

Furthermore, we want to look at how much augmentations the implemented algorithm really needs. For this purpose the algorithm has been tested on various problems. In Fig.16 you see the difference between the counted number of augmentations and the maximum number of augmentations for all the different problems. The x-axis presents the estimated number of augmentations $\frac{mn}{2}$, whereas the y-axis presents the number of augmentations actually needed. The numbers refer to various different problems. For each test the number of workgroups, the number of orders and the abilities of the workgroups were changed randomly.

For the purpose of solving a specified scheduling problem, there is a significant difference between the number of augmentations the algorithm needs to solve the problem and the maximal number of augmentations the algorithm could require as you can learn from Fig.16. Even though the Edmonds-Karp-Algorithm can solve rather complex problems, the runtime in contrast to the runtime of the *FindMaximumFlow* function is much bigger. For this purpose you have to consider the different implementation of self implemented functions and provided functions in Mathematica.

4.6 The Push-Relabel-Algorithm

The third algorithm we want to deal with was introduced by Goldberg and Tarjan in 1986, see [4]. This algorithm differs from the others by the condition for having a flow of maximum value. Before explaining the main difference of the

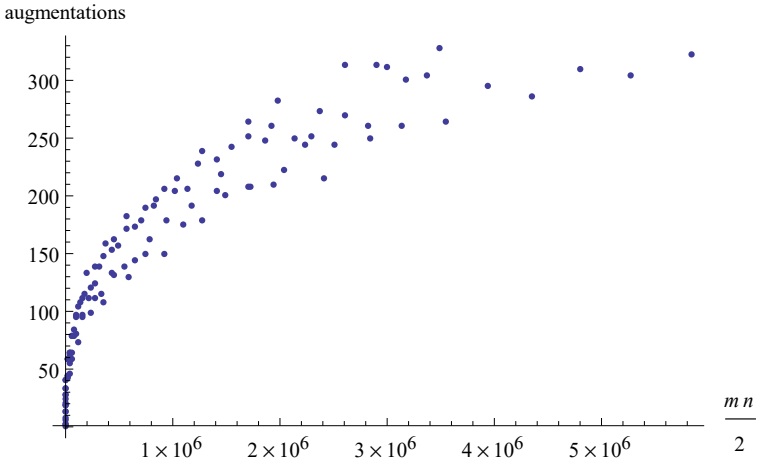


Fig. 16: Comparing the estimated to the needed number of augmentations.

algorithms we need some more definitions.

Given a network (G, u, h, r). An *h-r-preflow* of (G, u, h, r) is a function $f\colon E(G) \to \mathbb{R}_+$, s.t. $f(e) \leqslant u(e)$ for all $e \in E(G)$ and $\mathrm{ex}_f(v) \geqslant 0$ for all $v \in V(G) \setminus \{h\}$. Moreover, we call $v \in V(G) \setminus \{h, r\}$ an *active vertex iff* $ex_{f,G}(v) > 0$.

Both, the Ford-Fulkerson-Algorithm and the Edmonds-Karp-Algorihtm, start with an *h-r*-flow f, only change the flow such that the flow stays an *h-r*-flow and end if there is no f-augmenting path anymore. Considering this, the Push-Relabel-Algorithm goes the other way around. At each step the algorithm provides an *h-r*-preflow f such that there is no f-augmenting path and ends if the flow an *h-r*-flow .

To establish the Push-Relabel-Algorithm we need two more definitions. Again, we deal with a network (G, u, h, r). A *distance mark* ψ of G_f w.r.t. f is a function $\psi\colon V(G) \to \mathbb{N}$, with $\psi(r) = 0$, $\psi(h) = |V(G)|$ and $\psi(v) \leqslant \psi(w) + 1$ for $(v, w) \in E(G_f)$. An edge $e = (v, w) \in E(\overleftrightarrow{G})$ is called a *permitted edge* w.r.t. ψ *iff* $e \in E(G_f)$ and $\psi(v) = \psi(w) + 1$.

If ψ is a distance mark, then $\psi(v)$ (with $v \neq h$) will be the minimal number of edges in G_f for a shortest path from v to r.

The Push-Relabel-Algorithm

Input: A network (G,u,h,r).

Output: An h-r-flow of (G, u, h, r) of maximum value.

1. Set $f(e) := u(e)$ for all $e \in \delta_G^+(h)$.
   Set $f(e) := 0$ for all $e \in E(G) \setminus \delta_G^+(h)$.
2. Set $\psi(h) = |V(G)|$ and $\psi(v) = 0$ for all $v \in V(G) \setminus \{h\}$.
3. *While* there is an active vertex *do*:
   Let v be an active vertex.
   If no $e \in \delta_{G_f}^+(v)$ is permitted
   then RELABEL(v).
   else let $e \in \delta_{G_f}^+(v)$ be a permitted edge and *PUSH*(v).

RELABEL(v)

1. Set $\psi(v) :=\min\{\psi(w) + 1 \mid (v, w) \in \delta_G^+(G_f]v)\}$

PUSH(v)

1. Set $\gamma :=\min\{\mathrm{ex}_f(v), u_f(e)\}$, with e starting at v.
2. Augment f along e by γ.

This algorithm can be implemented with a runtime of $\mathcal{O}(n^2\sqrt{m})$, with $n = |V(G)|$ and $m = |E(G)|$, see [1]. We are going to show that this algorithm is correct. Thus we have to introduce some more lemmas.

Lemma 5. *During the whole Push-Relabel-Algorithm f stays an h-r-preflow of (G, u, h, r) and ψ stays a distance mark w.r.t. f.*

Proof. We have a network (G, u, h, r), an h-r-preflow f and a distance mark ψ. So the only thing that can happen, is that *PUSH* or *RELABEL* violates the claimed properties of a h-r-preflow or a distance mark.

Case 1: Consider the *PUSH*-operation, which only influences our h-r-preflow f. To apply *PUSH* we need an active vertex v and a permitted edge $e \in \delta_{G_f}^+(v)$ that starts at v. Now we know that $ex_{f,G}(v) > 0$. On the one hand consider the case that $\gamma = ex_{f,G}(v)$. Thus $ex_{f,G}(v)$ would get zero, but the flow f stays an h-

r-preflow, because $ex_{f,G}(v) = 0 \geqslant 0$. On the other hand if $\gamma = u_f(e)$ we would get that $f(e) = u(e)$ and the flow f stays an h-r-preflow again because $ex_{f,G}(v)$ would increase. As a next step we have to show that ψ stays a distance mark w.r.t. our new preflow f after applying the *PUSH*-operation. Hence, we have to expose that $\psi(a) \leqslant \psi(b) + 1$ for all new edges (a, b) in G_f. The condition for $PUSH(v)$ is that there is a permitted edge $e = (v, w) \in \delta^+_{G_f}(v)$. Thus we get that $\psi(v) = \psi(w) + 1$. Now the only new edge we can get by augmenting our flow is the contrary one. But for (w, v) holds $\psi(w) = \psi(v) - 1 \leqslant \psi(v) + 1$. Finally, ψ stays a distance mark.

Case 2: Let us focus on the *RELABEL*-operation, which only affects the distance mark ψ. After applying *RELABEL* on v, ψ stays a distance mark because no $e \in \delta^+_{G_f}(v)$ was permitted. For this case ψ increases strictly and we get that

$$\psi(v) = \min\{\psi(w) + 1 \mid (v, w) \in \delta^+_{G_f}(v)\} \leqslant \psi(w) + 1 \text{ for all } (v,w) \in E(G_f). \quad \square$$

Using Lemma 5, we can conclude that at each step of the algorithm we have a distance mark ψ for our h-r-preflow f. This is a big benefit because of the following lemma.

Lemma 6. *Let (G, u, h, r) be a network, f an h-r-preflow of (G, u, h, r) and ψ a distance mark w.r.t. f. The following three statements hold:*

5.1 *h is always reachable from any active vertex $v \in G_f$ in G_f.*

5.2 *If there are two vertices v,$w \in V(G)$, with the property that you can reach w from v in G_f, so we have $\psi(v) \leqslant \psi(w) + n - 1$, with $n = |V(G)|$.*

5.3 *r is not reachable from h in G_f.*

Proof. For **5.1** we choose an active vertex v arbitrary but fixed. Additionally, we define $B = \{w \in G \mid w \text{ is reachable from } v \text{ in } G_f\} \cup \{v\}$ as the set of all vertices that are reachable from v in G_f with $v \in B$. Now we assume that $h \notin B$. Because of the fact that f is an h-r-preflow and v is active we get

$$ex_{f,G}(w) \geqslant 0 \quad \text{for all } w \in B \quad \wedge \quad ex_{f,G}(v) > 0.$$

Similary to (1) of Theorem 1 we get that $f(e) = 0$ for $e \in \delta_G^-(B)$. Moreover, we get by remembering step (4) of Lemma 2 that

$$\begin{aligned} 0 < \sum_{w \in B} ex_{f,G}(w) &= \sum_{e \in \delta_G^-(B)} f(e) - \sum_{e \in \delta_G^+(B)} f(e) \\ &= 0 - \sum_{e \in \delta_G^+(B)} f(e) \leqslant 0. \quad \lightning \end{aligned}$$

Due to this contradiction we get that $h \in B$.

For **5.2** we know that w is reachable from v. For this purpose there has to be a path $v = v_0, v_1, \ldots, v_{k-1}, v_k = w$ from v to w in G_f. Considering that ψ is a distance mark w.r.t. f we can conclude that $\psi(v_i) \leqslant \psi(v_{i+1}) + 1$ for $i = 0, \ldots, k-1$. As a result we get that $\psi(v) \leqslant \psi(w) + k$ with $k \leqslant n-1$ because the shortest path from one vertex to an other can only consist of n vertices in G_f. Thus we get

$$\psi(v) \leqslant \psi(w) + k \leqslant \psi(w) + n - 1.$$

Finally, we can show **5.3** . Because of the definiton of ψ we have that $\psi(h) = n$ and $\psi(r) = 0$. We assume that r is reachable from h in G_f. So we can use **5.2** . As a result we get that

$$n = \psi(h) \leqslant \psi(r) + n - 1 = n - 1. \quad \lightning$$

This is a contradicition to our assumption and as a result we get that r is not reachable from h in G_f. □

By the help of the previous lemmas we can formulate Theorem 5 about the correctness of the Push-Relabel-Algorithm.

Theorem 5. *Given a network (G, u, h, r). If the Push-Relabel-Algorithm terminates, the resulting flow f of (G, u, h, r) is an h-r-flow of maximum value.*

Proof. Assume the algorithm stops because there are no active vertices anymore. As a result the flow f is an h-r-flow. Lemma 5 provides the conditions for Lemma **5.3**. Thus r is not reachable from h in G_f and so there is no f-augmenting path. As a consequence f is an h-r-flow of maximum value. □

4.7 Implementing and Testing the Push-Relabel-Algorithm

We have already seen, that the Edmonds-Karp-Algorithm and the *FindMaximumFlow* function calculate different solutions. For this purpose the Push-Relabel-Algorithm was tested on the problem in Fig.13. It is interesting to see that the Push-Relabel-Algorithm supplies the same result as the *FindMaximumFlow* function. The resulting flow is presented in Fig.17.

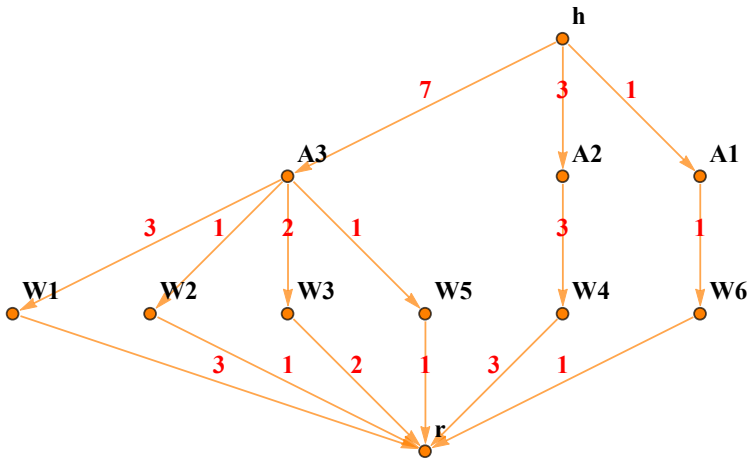
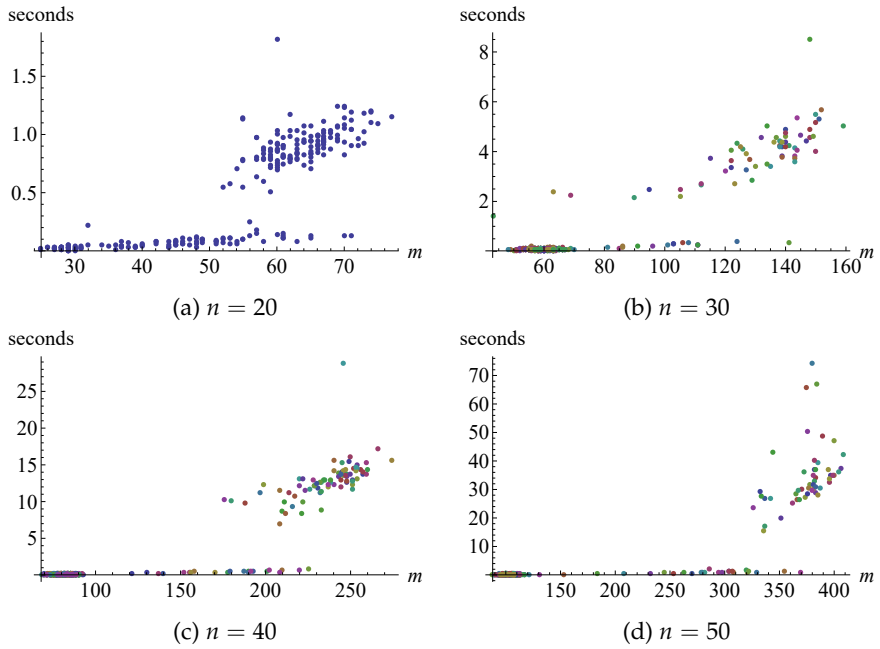


Fig. 17: The result of the problem in Fig.13 by using the Push-Relabel-Algorithm.

We have already mentioned that the algorithm has a complexity of $\mathcal{O}(n^2\sqrt{m})$. For investigating the runtime, the algorithm was tested on various problems with a fixed number of vertices n but a changing number of edges m. Remember that the number of vertices is the result of the number of workgroups plus the number of orders plus two. For the number of edges the abilities of the workgroups are crucial. In Fig.18 you see the runtime for various problems with a fixed number of vertices. Additionally, Fig.19 shows off the runtime of the Edmonds-Karp-Algorithm for the same problems as well. As you will see, the Push-Relabel-Algorithm is much slower than the Edmonds-Karp-Algorithm.



(a) $n = 20$ (b) $n = 30$

(c) $n = 40$ (d) $n = 50$

Fig. 18: The development of the runtime of the Push-Relabel-Algorithm with respect to m for $n = 20, 30, 40$ and 50.

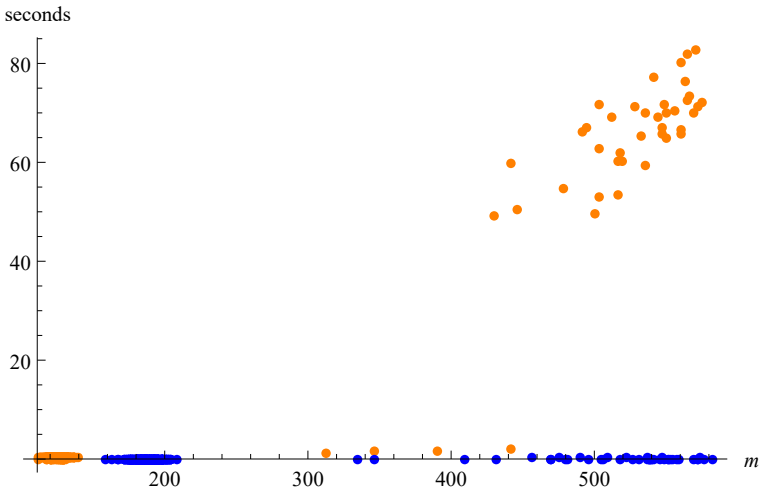


Fig. 19: Comparing the runtime of the Edmonds-Karp-Algorithm (blue points) to the Push-Relabel-Algorithm (orange points) for $n = 60$.

5 Solving Specific Scheduling Problems

The previous section has dealt in detail with possibilities of solving a Max-Flow problem. We have already formulated the scheduling problem as a network problem as well. In the last section we want to solve the sample problem as well as a problem with higher numbers of workgroups and loading orders.

5.1 Solving our Sample Problem

We can solve our small sample Problem by using the capacity matrix C. Using the option *FlowMatrix* we get the following matrix as a result.

$$\begin{pmatrix}
0&8&8&7&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&2&2&1&3&0&0\\
0&0&0&0&0&0&6&0&0&0&2&0\\
0&0&0&0&2&2&3&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&11\\
0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&3\\
0&0&0&0&0&0&0&0&0&0&0&2\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{pmatrix}$$

The distribution of the Max-Flow is visualized in Fig.20.

From the resulting matrix we get our x_{ij} from Section 2.2 by considering the submatrix consisting of the rows 2–4 and the columns 4–11. This matrix is exactly the one that was asked for in Section 2.3.

$$\begin{pmatrix}
0 & 0 & x_{13} & x_{14} & x_{15} & x_{16} & 0\\
x_{21} & x_{22} & x_{23} & 0 & x_{25} & x_{26} & x_{27}\\
x_{31} & x_{32} & x_{33} & 0 & x_{35} & 0 & 0
\end{pmatrix} = \begin{pmatrix}
0&0&2&2&1&3&0\\
0&0&6&0&0&0&2\\
2&2&3&0&0&0&0
\end{pmatrix}$$

By implementing some auxiliary functions we get our workplan where you can read out the time t, how long workgroup i should work on order j. This information is given in the following form: $\{i, j\} \to t$.

$\{1,3\} \to 2$	$\{1,4\} \to 2$	$\{1,5\} \to 1$	$\{1,6\} \to 3$
$\{2,3\} \to 6$	$\{2,7\} \to 2$		
$\{3,1\} \to 2$	$\{3,2\} \to 2$	$\{3,3\} \to 3$	

Another interesting aspect of the problem is the efficiency. That means how much time stays unused and which work cannot be finished within the eight

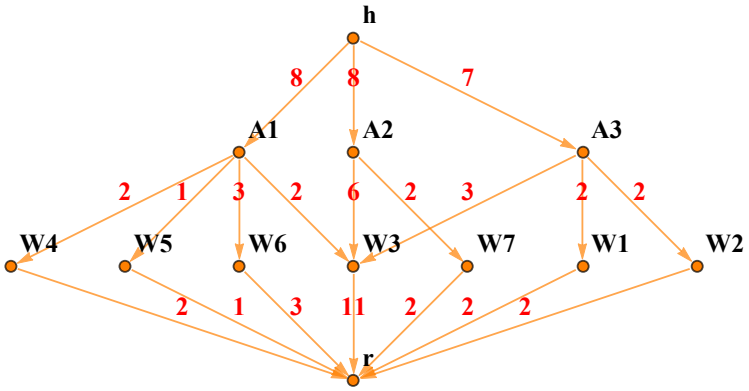


Fig. 20: The maximum flow of the sample problem.

hours of work. Thus the empty set for the undone work means that all orders can be finished.

Efficiency:

- unused hours of work: 1
- undone work: {}

5.2 Solving a Complex Problem

In the end we can solve a big problem by the built-in function *FindMaximumFlow* in Mathematica again. For the problem we have 150 workgroups, who have a maximal worktime of eight hours each again, and 450 orders. Because of the large numbers of orders and workgroups we get a lot of data for the abilities of the workgroups and for the workingtimes, see Sections 7.3 and 7.4. For solving the problem by Mathematica we have to define the capacity matrix C. As a result we get a (602×602)-matrix. The workplan is listed in Section 7.2. Moreover, we get information about the efficiency again.

Efficiency:

- unused hours of work: 17
- undone work: {}

6 Conclusion

It should be clear that the chosen specification of the mathematical problem as well as the model of the network problem is not unique. There are several methods to specify the given scheduling problem. Moreover, you can model the network the other way around as well, which means that you first go from h to the loading orders and afterwards to the workgroups. Perhaps there is also a possibility to solve the problem without graph theory. This would be an interesting point, to try different models and compare the different results. Maybe some algorithm works faster on a different model. Of course it would always be an interesting point to outline all different algorithms and find specific problems which some algorithm can solve faster than the other. We could also think about the best solution, because a flow of maximum value is not unique. Maybe you want that all workgroups should work the same time, or one workgroup should work less because it is very expensive. For this purpose we could think about methods to modify a correct solution to the best solution.

This thesis has shown that it is fairly straight forward to model a scheduling problem as a network problem. Moreover, the problems and solutions can be presented nicely by graphs. For the aspect of solving network problems, there are many other algorithms. The Edmonds-Karp- and the Push-Relabel-Algorithm are really interesting because they use a rather diverse strategy to solve the problem. As a further aspect we have seen that the runtime of the algorithms can strongly depend on the structure of the network.

Even though there are a lot of algorithms to find a shortest path on a graph and maybe there are some similar algorithms in some literature, it was a desire to try to design an own algorithm to find a shortest path of the modelled network of a scheduling problem. The lemmas and theorems for the algorithm are an outcome of the preceding thoughts.

References

1. Bernhard Korte & Jens Vygen, *Kombinatorische Optimierung*, Springer Spektrum, 2018.
2. Herbert S. Wilf, *Algorithms and Complexity*, University of Pennsylvania Philadelphia, 1994
3. Uri Zwick, *The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate*, Tel Aviv University, 1993
4. Andrew V. Goldberg & Robert E. Tarjan, *A New Approach to the Maximum-Flow Problem*, 1986

7 Appendix

7.1 Implementations

```
------------------------------------------------------------------------------------------------------------------------

the function NewAugmentedFlow  augments a given flow along a
given path by a given value
myAdMatrix:   is the CapacityMatrix of the given network
myflow:     is the flow of the network
mypath_List:    is the given f-augmenting path
Min:      is the value gamma for the augmentation
------------------------------------------------------------------------------------------------------------------------


Clear[NewAugmentedFlow];
NewAugmentedFlow[myAdMatrix_, myflow_, mypath_List, Min_] :=
Module[{myFlow = myflow, myPath = mypath, myMin = Min, resultFlow = myflow},
For[i = 1, i < Length[mypath], i++,
If[myAdMatrix[[mypath[[i]]]][[mypath[[i + 1]]]] != Infinity,
resultFlow[[mypath[[i]]]][[mypath[[i + 1]]]] += myMin, ]];
For[i = 1, i < Length[mypath], i++,
If[myAdMatrix[[mypath[[i + 1]]]][[mypath[[i]]]] != Infinity,
resultFlow[[mypath[[i + 1]]]][[mypath[[i]]]] -= myMin, ]];
Return[resultFlow]];

------------------------------------------------------------------------------------------------------------------------

the function NewResidualMatrix updates the residual capacities by a
specific flow
myAdMatrix:   is the CapacityMatrix of the given network
FlowMatrix:     is the flow of the network
------------------------------------------------------------------------------------------------------------------------


Clear[NewResidualMatrix];
NewResidualMatrix[myAdMatrix_, FlowMatrix_] :=
Module[{ResidualMatrix = GetCapacityMatrix[myAdMatrix],
myFlow = GetCapacityMatrix[FlowMatrix]}, ResidualMatrix -= myFlow;
ResidualMatrix += Transpose[myFlow]; Return[GetAdMatrix[ResidualMatrix]]];

------------------------------------------------------------------------------------------------------------------------

the function GetWeigths computes all capactiies of a path
myAdMatrix:   is the CapacityMatrix of the given network
mypath_List:    is the given path
------------------------------------------------------------------------------------------------------------------------


Clear[GetWeights];
GetWeights[myAdMatrix_, mypath_List] :=
Table[myAdMatrix[[mypath[[i]]]][[mypath[[i + 1]]]], {i, Length[mypath] - 1}];

------------------------------------------------------------------------------------------------------------------------

the function GoFrom gives you all vertices that are reachable from a in Gf
Gf:     is the CapacityMatrix of the residual graph
a:      is a vertice of V(G)
n:      is the number of all vertices
------------------------------------------------------------------------------------------------------------------------


Clear[GoFrom];
GoFrom[Gf_, a_, n_] :=
Return@Flatten@
Reap[Table[If[Gf[[a]][[i]] != Infinity, Sow[i], i], {i, n}]][[2]];

------------------------------------------------------------------------------------------------------------------------

the function Go updates the the set A and the paths
Gf:     is the CapacityMatrix of the residual graph
PathReachable_List: is a list of tuples of paths and reachable vertices
n:      is the number of all vertices
Visited:    is the set of all vertices that were already ''visited'' (V)
------------------------------------------------------------------------------------------------------------------------


Clear[Go];
Go[Gf_, PathReachable_List, n_, Visited_] :=
Module[{path = PathReachable[[1]], Reachable = PathReachable[[2]],
visited = Visited},
Return@DeleteCases[
Table[{Join[path, {Reachable[[i]]}],
Complement[GoFrom[Gf, Reachable[[i]], n], visited]}, {i,
Length[Reachable]}], {List_, {}}]];
```

```
-----------------------------------------------------------------------------------------------------------------

the function PathToR tests if there is a path from A to r, and returns one if so, moreover this
function updates the set Visited if no path exists and returns the set
Gf:     is the CapacityMatrix of the residual graph
PathReachable:  is a tuple of a path and reachable vertices
n:      is the number of all vertices
Visited:    is the set of all vertices that were already ''visited'' (V)
w:      is the set W that saves all vertices from where r is reachable
-----------------------------------------------------------------------------------------------------------------


Clear[PathToR];
PathToR[Gf_, PathReachable_, n_, Visited_, w_] :=
Module[{a = PathReachable[[2]], M, p, myPath = PathReachable[[1]], visited},
M = Gf[[a]][[All, w]]; p = Position[M, _Integer, {2}];
If[p == {}, visited = Union[Visited, a]; Return@{visited, {}}, p = p[[1]]];
myPath = Join[myPath, {a[[p[[1]]]], w[[p[[2]]]], n};
Return@{Visited, myPath}];

-----------------------------------------------------------------------------------------------------------------

the function PathChecking checks if there is a path from A to W
Gf:     is the CapacityMatrix of the residual graph
PathReachable_List: is a list of tuples of paths and reachable vertices
n:      is the number of all vertices
Visited:    is the set of all vertices that were already ''visited'' (V)
w:      is the set W that saves all vertices from where r is reachable
-----------------------------------------------------------------------------------------------------------------


Clear[PathChecking];
PathChecking[Gf_, PathReachable_List, n_, Visited_, w_] :=
Module[{p = {}, visited = Visited},
For[i = 1, i <= Length[PathReachable],
i++, {visited, p} = PathToR[Gf, PathReachable[[i]], n, visited, w];
If[p == {}, , Return@{visited, p}]]; Return@{visited, p}];

-----------------------------------------------------------------------------------------------------------------

the function FindShortestPathNetwork computes the shortest path in Gf,
if no path exists, it returns {}
Gf:     is the CapacityMatrix of the residual graph
n:      is the number of all vertices
-----------------------------------------------------------------------------------------------------------------


FindShortestPathNetwork[Gf_, n_] :=
Module[{w =
Flatten@Reap[
Table[If[Gf[[All, n]][[i]] != Infinity, Sow[i], i], {i, n}]][[2]],
PathReachable = {{}, {1}}, path = {}, VisitedOld = {1}, VisitedNew, a},
a = Go[Gf, PathReachable, n, VisitedOld];
While[path == {}, {VisitedNew, path} =
PathChecking[Gf, a, n, VisitedOld, w];
If[path != {} || VisitedNew == VisitedOld, Return@path,];
VisitedOld = VisitedNew;
a = Flatten[Map[Go[Gf, #, n, VisitedNew] &, a], 1]]];


-----------------------------------------------------------------------------------------------------------------

the function EdmondsKarp computes a flow of maximum value for a given network
by the Edmonds-Karp-Algorithm, the Algorithm also returns the counted number
of augmentations
myNetworkMatrix:  The AdjacencyMatrix of the Network
-----------------------------------------------------------------------------------------------------------------


Clear[EdmondsKarp];
EdmondsKarp[myNetworkMatrix_] :=
Module[{myFlow, myPath, myWeights, myMin, counter = 0, ResidualM,
n = Dimensions[myNetworkMatrix][[1]]}, myFlow = Table[0, {i, n}, {j, n}];
myPath = FindShortestPathNetwork[myNetworkMatrix, n];
ResidualM = myNetworkMatrix;
While[myPath != {}, myWeights = GetWeights[ResidualM, myPath];
myMin = Min[myWeights];
myFlow = NewAugmentedFlow[myNetworkMatrix, myFlow, myPath, myMin];
ResidualM = NewResidualMatrix[myNetworkMatrix, myFlow];
myPath = FindShortestPathNetwork[ResidualM, n]; counter++];
Return[{GetAdMatrix@myFlow, counter}]];

-----------------------------------------------------------------------------------------------------------------

the function AllReachable2 computes a set of all vertices that a reachable
from a given vertice
NetworkMatrix:  The AdjacencyMatrix of the Network
vertex_Integer: a given vertice
N:      the number of all vertices in the network
```

```
-----------------------------------------------------------------------------------------------------------------------


Clear[AllReachable2];
AllReachable2[NetworkMatrix_, vertex_Integer, N_] :=
Return[Flatten@
Reap[Table[
If[NetworkMatrix[[vertex]][[i]] != Infinity, Sow[i], i], {i, N}]][[2]]];

-----------------------------------------------------------------------------------------------------------------------

the function AllGoTo2 computes the set of all vertices from where you can reach a
given vertex
NetworkMatrix:  The AdjacencyMatrix of the Network
vertex_Integer: a given vertex
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[AllGoTo2];
AllGoTo2[NetworkMatrix_, vertex_Integer, N_] :=
Return[Flatten@
Reap[Table[
If[NetworkMatrix[[i]][[vertex]] != Infinity, Sow[i], i], {i, N}]][[2]]];

-----------------------------------------------------------------------------------------------------------------------

the function Excess2 computes the excess of a given vertex for a given flow
given vertex
FlowMatrix:   the given flow
vertex_Integer: a given vertex
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[Excess2];
Excess2[FlowMatrix_, vertex_Integer, N_] :=
Module[{In = AllGoTo2[FlowMatrix, vertex, N],
Out = AllReachable2[FlowMatrix, vertex, N], InFlow, OutFlow},
InFlow = Sum[FlowMatrix[[In[[i]]]][[vertex]], {i, 1, Length[In]}];
OutFlow = Sum[FlowMatrix[[vertex]][[Out[[i]]]], {i, 1, Length[Out]}];
Return[InFlow - OutFlow]];

-----------------------------------------------------------------------------------------------------------------------

the function AllActive2 computes a set of all vertices that are active for a
given flow. It does not consider the vertex r
given vertex
FlowMatrix:   the given flow
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[AllActive2];
AllActive2[FlowMatrix_, N_] :=
Return@Flatten@
Reap[Table[If[Excess2[FlowMatrix, j, N] > 0, Sow[j], j], {j, N - 1}]][[2]];

-----------------------------------------------------------------------------------------------------------------------

the function AllActive3 updates a given set of all vertices that are active for a
given flow. It does not consider the vertex r. This function uses the property that
only the excess of the vertices of the given edge could have changed.
FlowMatrix:   the given flow
ActiveVertices: a set of all active vertices before changing the flow
edge_List:    a tuple {a,b} - an edge from a to b
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[AllActive3];
AllActive3[FlowMatrix_, ActiveVertices_, edge_List, N_] :=
Module[{AllActive = ActiveVertices},
If[Excess2[FlowMatrix, edge[[1]], N] <= 0,
AllActive = Drop[AllActive, Flatten@Position[ActiveVertices, edge[[1]]],];
If[Excess2[FlowMatrix, edge[[2]], N] > 0 && edge[[2]] != N,
If[MemberQ[AllActive, edge[[2]]], , AppendTo[AllActive, edge[[2]]],];
Return@AllActive];

-----------------------------------------------------------------------------------------------------------------------

the function Permissed gives True if a given edge is permissed for a given
residual graph and a given distance mark
edge_List:    a tuple {a,b} - an edge from a to b
ResidualMatrix: the Adjacency Matrix for a redsidual graph
Distance_List:  is a distance mark
-----------------------------------------------------------------------------------------------------------------------


Clear[Permissed];
```

```
Permissed[edge_List, ResidualMatrix_, Distance_List] :=
Return[ResidualMatrix[[edge[[1]]]][[edge[[2]]]] != Infinity &&
Distance[[edge[[1]]]] == Distance[[edge[[2]]]] + 1];

-----------------------------------------------------------------------------------------------------------------------

the function AllPermissed2 computes a set of all permissed edges for
a given residual graph and a given distance mark
ResidualMatrix: the Adjacency Matrix for a redsidual graph
vertex_Integer: a given vertex
Distance_List:  is a distance mark
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[AllPermissed2];
AllPermissed2[ResidualMatrix_, vertex_Integer, Distance_List, N_] :=
Module[{ReachableV = AllReachable2[ResidualMatrix, vertex, N]},
Return[Flatten[
Reap[Table[
If[Permissed[{vertex, ReachableV[[i]]}, ResidualMatrix, Distance],
Sow[{vertex, ReachableV[[i]]}], {vertex, ReachableV[[i]]}], {i,
Length[ReachableV]}]][[2]], 1]]];

-----------------------------------------------------------------------------------------------------------------------

the function Relabel2 is the Relabel-operation of the Push-Relabel-Algorithm
ResidualMatrix: the Adjacency Matrix for a redsidual graph
Distance_List   is a distance mark
vertex_Integer: a given active vertex
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[Relabel2];
Relabel2[ResidualMatrix_, Distance_List, vertex_Integer, N_] :=
Module[{M, ReachableV = AllReachable2[ResidualMatrix, vertex, N],
Psi = Distance},
M = Table[Distance[[ReachableV[[i]]]] + 1, {i, Length[ReachableV]}];
Psi[[vertex]] = Min[M]; Return@Psi];

-----------------------------------------------------------------------------------------------------------------------

the function Push2 is the Push-operation of the Push-Relabel-Algorithm
NetworkMatrix:  The AdjacencyMatrix of the Network
ResidualMatrix: the Adjacency Matrix for a redsidual graph
FlowMatrix:   the given flow
edge_List   a given permissed edge
N:      the number of all vertices in the network
-----------------------------------------------------------------------------------------------------------------------


Clear[Push2];
Push2[NetworkMatrix_, ResidualMatrix_, FlowMatrix_, edge_List, N_] :=
Module[{excess = Excess2[FlowMatrix, edge[[1]], N],
residualcapacity = ResidualMatrix[[edge[[1]]]][[edge[[2]]]], gamma,
NewFlow}, gamma = Min[excess, residualcapacity];
NewFlow = NewAugmentedFlow[NetworkMatrix, FlowMatrix, edge, gamma];
Return@NewFlow];

-----------------------------------------------------------------------------------------------------------------------

the function PushRelabel2 computes a flow of maximum value for a given network
by the Push-Relabel-Algorithm, the Algorithm also returns the counted number loops
NetworkMatrix:  The AdjacencyMatrix of the Network
-----------------------------------------------------------------------------------------------------------------------


Clear[PushRelabel2];
PushRelabel2[NetworkMatrix_] :=
Module[{G = NetworkMatrix, Gf = NetworkMatrix,
n = Dimensions[NetworkMatrix][[1]], Active, v, PermissedEdges, e,
ReachableH, myFlow, Phi, counter = 0},
myFlow = Table[Table[0, {n}], {n}];
Phi = Join[{n}, Table[0, {n - 1}]];
ReachableH = AllReachable2[NetworkMatrix, 1, n];
For[k = 1, k <= Length[ReachableH], k++,
myFlow[[1]][[ReachableH[[k]]]] = G[[1]][[ReachableH[[k]]]];
Active = AllActive2[myFlow, n]; Gf = NewResidualMatrix[G, myFlow];
While[Active != {}, v = Active[[1]];
PermissedEdges = AllPermissed2[Gf, v, Phi, n]; counter++;
If[PermissedEdges == {}, Phi = Relabel2[Gf, Phi, v, n],
e = PermissedEdges[[1]]; myFlow = Push2[G, Gf, myFlow, e, n];
Gf = NewResidualMatrix[G, myFlow];
Active = AllActive3[myFlow, Active, e, n]]];
Return[{GetAdMatrix[myFlow], counter}]];
```

7.2 The Workplan of the Complex Problem

$\{1,6\} \to 2$	$\{1,256\} \to 3$	$\{1,263\} \to 1$	$\{1,337\} \to 2$		
$\{2,270\} \to 7$	$\{2,379\} \to 1$				
$\{3,362\} \to 8$					
$\{4,2\} \to 1$	$\{4,249\} \to 1$	$\{4,345\} \to 2$	$\{4,377\} \to 3$		
$\{5,43\} \to 1$	$\{5,69\} \to 1$	$\{5,316\} \to 6$			
$\{6,24\} \to 2$	$\{6,303\} \to 6$				
$\{7,10\} \to 1$	$\{7,11\} \to 2$	$\{7,174\} \to 1$	$\{7,229\} \to 2$	$\{7,260\} \to 2$	
$\{8,188\} \to 2$	$\{8,244\} \to 1$	$\{8,296\} \to 3$	$\{8,396\} \to 2$		
$\{9,106\} \to 2$	$\{9,136\} \to 1$	$\{9,227\} \to 2$	$\{9,267\} \to 2$	$\{9,280\} \to 1$	
$\{10,22\} \to 3$	$\{10,87\} \to 1$	$\{10,209\} \to 1$	$\{10,212\} \to 2$	$\{10,318\} \to 1$	
$\{11,44\} \to 1$	$\{11,104\} \to 1$	$\{11,137\} \to 1$	$\{11,151\} \to 1$	$\{11,217\} \to 3$	$\{11,341\} \to 1$
$\{12,61\} \to 3$	$\{12,156\} \to 1$	$\{12,170\} \to 2$	$\{12,181\} \to 1$	$\{12,375\} \to 1$	
$\{13,44\} \to 2$	$\{13,150\} \to 3$	$\{13,183\} \to 1$	$\{13,256\} \to 1$	$\{13,303\} \to 1$	
$\{14,222\} \to 6$	$\{14,363\} \to 2$				
$\{15,44\} \to 1$	$\{15,193\} \to 7$				
$\{16,8\} \to 1$	$\{16,85\} \to 5$	$\{16,139\} \to 1$	$\{16,147\} \to 1$		
$\{17,46\} \to 2$	$\{17,196\} \to 1$	$\{17,218\} \to 3$	$\{17,291\} \to 2$		
$\{18,2\} \to 2$	$\{18,232\} \to 6$				
$\{19,150\} \to 3$	$\{19,168\} \to 2$	$\{19,266\} \to 1$	$\{19,387\} \to 2$		
$\{20,12\} \to 1$	$\{20,158\} \to 1$	$\{20,176\} \to 1$	$\{20,223\} \to 2$	$\{20,288\} \to 1$	$\{20,362\} \to 2$
$\{21,265\} \to 7$	$\{21,348\} \to 1$				
$\{22,394\} \to 8$					
$\{23,243\} \to 4$	$\{23,308\} \to 2$	$\{23,321\} \to 2$			
$\{24,180\} \to 7$	$\{24,213\} \to 1$				
$\{25,60\} \to 2$	$\{25,117\} \to 3$	$\{25,397\} \to 3$			
$\{26,32\} \to 1$	$\{26,154\} \to 1$	$\{26,179\} \to 4$	$\{26,399\} \to 2$		
$\{27,41\} \to 3$	$\{27,74\} \to 1$	$\{27,91\} \to 3$	$\{27,200\} \to 1$		
$\{28,112\} \to 3$	$\{28,120\} \to 2$	$\{28,187\} \to 3$			
$\{29,203\} \to 1$	$\{29,220\} \to 1$	$\{29,316\} \to 6$			
$\{30,77\} \to 1$	$\{30,163\} \to 2$	$\{30,205\} \to 1$	$\{30,265\} \to 4$		
$\{31,319\} \to 1$	$\{31,328\} \to 5$	$\{31,334\} \to 2$			
$\{32,192\} \to 6$	$\{32,247\} \to 2$				
$\{33,18\} \to 1$	$\{33,222\} \to 1$	$\{33,231\} \to 1$	$\{33,290\} \to 2$	$\{33,358\} \to 3$	
$\{34,64\} \to 5$	$\{34,248\} \to 1$	$\{34,275\} \to 2$			
$\{35,200\} \to 2$	$\{35,208\} \to 6$				
$\{36,180\} \to 7$	$\{36,189\} \to 1$				
$\{37,175\} \to 4$	$\{37,196\} \to 2$	$\{37,259\} \to 2$			
$\{38,309\} \to 3$	$\{38,312\} \to 1$	$\{38,323\} \to 1$	$\{38,353\} \to 2$	$\{38,367\} \to 1$	
$\{39,309\} \to 8$					
$\{40,273\} \to 3$	$\{40,280\} \to 2$	$\{40,336\} \to 2$			

$\{41,102\} \to 7$	$\{41,203\} \to 1$				
$\{42,33\} \to 1$	$\{42,72\} \to 1$	$\{42,114\} \to 2$	$\{42,188\} \to 1$	$\{42,195\} \to 2$	$\{42,318\} \to 1$
$\{43,121\} \to 2$	$\{43,127\} \to 3$	$\{43,142\} \to 1$	$\{43,228\} \to 2$		
$\{44,95\} \to 3$	$\{44,134\} \to 3$	$\{44,177\} \to 2$			
$\{45,7\} \to 1$	$\{45,39\} \to 1$	$\{45,83\} \to 3$	$\{45,157\} \to 3$		
$\{46,20\} \to 2$	$\{46,72\} \to 1$	$\{46,123\} \to 3$	$\{46,202\} \to 2$		
$\{47,22\} \to 2$	$\{47,41\} \to 3$	$\{47,270\} \to 3$			
$\{48,81\} \to 1$	$\{48,87\} \to 2$	$\{48,110\} \to 4$	$\{48,390\} \to 1$		
$\{49,102\} \to 1$	$\{49,182\} \to 2$	$\{49,192\} \to 3$	$\{49,219\} \to 2$		
$\{50,99\} \to 5$	$\{50,210\} \to 3$				
$\{51,106\} \to 2$	$\{51,191\} \to 6$				
$\{52,143\} \to 1$	$\{52,161\} \to 1$	$\{52,224\} \to 3$	$\{52,226\} \to 2$		
$\{53,14\} \to 4$	$\{53,295\} \to 1$	$\{53,317\} \to 1$	$\{53,327\} \to 2$		
$\{54,70\} \to 2$	$\{54,86\} \to 3$	$\{54,128\} \to 1$	$\{54,135\} \to 2$		
$\{55,14\} \to 2$	$\{55,235\} \to 5$				
$\{56,73\} \to 3$	$\{56,141\} \to 1$	$\{56,221\} \to 1$	$\{56,246\} \to 2$	$\{56,342\} \to 1$	
$\{57,64\} \to 1$	$\{57,78\} \to 3$	$\{57,109\} \to 2$	$\{57,153\} \to 2$		
$\{58,53\} \to 4$	$\{58,59\} \to 1$	$\{58,133\} \to 1$	$\{58,215\} \to 1$		
$\{59,76\} \to 2$	$\{59,98\} \to 2$				
$\{60,111\} \to 1$	$\{60,141\} \to 5$	$\{60,169\} \to 2$			
$\{61,22\} \to 5$	$\{61,121\} \to 1$	$\{61,389\} \to 2$			
$\{62,102\} \to 2$	$\{62,126\} \to 1$	$\{62,302\} \to 1$	$\{62,307\} \to 2$	$\{62,310\} \to 2$	
$\{63,79\} \to 1$	$\{63,99\} \to 7$				
$\{64,214\} \to 2$	$\{64,236\} \to 6$				
$\{65,15\} \to 2$	$\{65,34\} \to 2$	$\{65,58\} \to 2$	$\{65,269\} \to 1$	$\{65,355\} \to 1$	
$\{66,55\} \to 1$	$\{66,118\} \to 1$	$\{66,125\} \to 2$	$\{66,167\} \to 2$	$\{66,376\} \to 2$	
$\{67,103\} \to 3$	$\{67,154\} \to 1$	$\{67,197\} \to 3$	$\{67,378\} \to 1$		
$\{68,55\} \to 1$	$\{68,67\} \to 1$	$\{68,113\} \to 3$	$\{68,116\} \to 3$		
$\{69,211\} \to 1$	$\{69,303\} \to 1$	$\{69,317\} \to 3$	$\{69,333\} \to 3$		
$\{70,250\} \to 5$	$\{70,298\} \to 3$				
$\{71,251\} \to 8$					
$\{72,271\} \to 8$					
$\{73,61\} \to 2$	$\{73,92\} \to 3$	$\{73,309\} \to 3$			
$\{74,82\} \to 1$	$\{74,148\} \to 3$	$\{74,171\} \to 3$	$\{74,207\} \to 1$		
$\{75,7\} \to 2$	$\{75,22\} \to 3$	$\{75,88\} \to 1$	$\{75,122\} \to 1$	$\{75,251\} \to 1$	
$\{76,44\} \to 3$	$\{76,100\} \to 2$	$\{76,222\} \to 3$			
$\{77,13\} \to 2$	$\{77,49\} \to 3$	$\{77,314\} \to 2$	$\{77,399\} \to 1$		
$\{78,248\} \to 2$	$\{78,304\} \to 2$	$\{78,351\} \to 4$			
$\{79,30\} \to 2$	$\{79,89\} \to 5$	$\{79,211\} \to 1$			
$\{80,16\} \to 1$	$\{80,173\} \to 3$	$\{80,270\} \to 1$	$\{80,300\} \to 3$		

$\{81,17\} \to 1$	$\{81,149\} \to 3$	$\{81,160\} \to 1$	$\{81,248\} \to 2$	$\{81,340\} \to 1$
$\{82,53\} \to 6$	$\{82,93\} \to 2$			
$\{83,17\} \to 1$	$\{83,52\} \to 1$	$\{83,178\} \to 3$	$\{83,265\} \to 1$	$\{83,368\} \to 2$
$\{84,373\} \to 3$	$\{84,374\} \to 3$	$\{84,382\} \to 2$		
$\{85,82\} \to 2$	$\{85,102\} \to 1$	$\{85,146\} \to 1$	$\{85,250\} \to 3$	$\{85,348\} \to 1$
$\{86,61\} \to 7$	$\{86,261\} \to 1$			
$\{87,52\} \to 2$	$\{87,111\} \to 2$	$\{87,124\} \to 1$	$\{87,144\} \to 2$	$\{87,275\} \to 1$
$\{88,82\} \to 2$	$\{88,230\} \to 5$	$\{88,385\} \to 1$		
$\{89,40\} \to 6$	$\{89,252\} \to 1$	$\{89,397\} \to 1$		
$\{90,53\} \to 1$	$\{90,56\} \to 3$	$\{90,101\} \to 2$	$\{90,273\} \to 2$	
$\{91,27\} \to 5$	$\{91,42\} \to 1$	$\{91,90\} \to 2$		
$\{92,274\} \to 1$	$\{92,292\} \to 2$	$\{92,346\} \to 2$	$\{92,395\} \to 3$	
$\{93,330\} \to 2$	$\{93,349\} \to 2$	$\{93,350\} \to 2$	$\{93,369\} \to 2$	
$\{94,193\} \to 1$	$\{94,297\} \to 1$	$\{94,324\} \to 1$	$\{94,355\} \to 2$	$\{94,361\} \to 3$
$\{95,194\} \to 1$	$\{95,238\} \to 2$	$\{95,315\} \to 1$	$\{95,331\} \to 3$	$\{95,335\} \to 1$
$\{96,64\} \to 3$	$\{96,150\} \to 3$	$\{96,254\} \to 1$	$\{96,311\} \to 1$	
$\{97,143\} \to 1$	$\{97,155\} \to 2$	$\{97,184\} \to 3$	$\{97,190\} \to 2$	
$\{98,204\} \to 5$	$\{98,326\} \to 1$	$\{98,339\} \to 2$		
$\{99,7\} \to 1$	$\{99,30\} \to 1$	$\{99,129\} \to 2$	$\{99,240\} \to 2$	$\{99,241\} \to 2$
$\{100,28\} \to 2$	$\{100,50\} \to 3$	$\{100,235\} \to 1$	$\{100,299\} \to 1$	$\{100,352\} \to 1$
$\{101,47\} \to 6$	$\{101,237\} \to 2$			
$\{102,81\} \to 1$	$\{102,193\} \to 1$	$\{102,216\} \to 1$	$\{102,295\} \to 3$	$\{102,384\} \to 2$
$\{103,51\} \to 3$	$\{103,130\} \to 2$	$\{103,365\} \to 3$		
$\{104,255\} \to 4$	$\{104,314\} \to 1$	$\{104,322\} \to 2$	$\{104,346\} \to 1$	
$\{105,262\} \to 8$				
$\{106,54\} \to 2$	$\{106,70\} \to 4$	$\{106,150\} \to 2$		
$\{107,21\} \to 1$	$\{107,26\} \to 4$	$\{107,36\} \to 1$	$\{107,204\} \to 2$	
$\{108,57\} \to 1$	$\{108,71\} \to 1$	$\{108,140\} \to 1$	$\{108,253\} \to 4$	$\{108,288\} \to 1$
$\{109,44\} \to 2$	$\{109,232\} \to 6$			
$\{110,138\} \to 2$	$\{110,289\} \to 1$	$\{110,320\} \to 2$	$\{110,332\} \to 2$	$\{110,347\} \to 1$
$\{111,287\} \to 1$	$\{111,302\} \to 1$	$\{111,330\} \to 6$		
$\{112,53\} \to 1$	$\{112,115\} \to 3$	$\{112,254\} \to 1$	$\{112,302\} \to 1$	$\{112,364\} \to 2$
$\{113,9\} \to 2$	$\{113,38\} \to 1$	$\{113,65\} \to 1$	$\{113,159\} \to 2$	$\{113,393\} \to 2$
$\{114,37\} \to 3$	$\{114,199\} \to 1$	$\{114,323\} \to 1$	$\{114,372\} \to 3$	
$\{115,5\} \to 1$	$\{115,23\} \to 3$	$\{115,94\} \to 3$	$\{115,96\} \to 1$	
$\{116,29\} \to 1$	$\{116,60\} \to 1$	$\{116,77\} \to 2$	$\{116,100\} \to 1$	$\{116,234\} \to 3$
$\{117,41\} \to 3$	$\{117,55\} \to 1$	$\{117,89\} \to 2$	$\{117,97\} \to 2$	
$\{118,75\} \to 2$	$\{118,84\} \to 3$	$\{118,164\} \to 2$		
$\{119,143\} \to 1$	$\{119,165\} \to 4$	$\{119,248\} \to 1$	$\{119,289\} \to 2$	
$\{120,45\} \to 1$	$\{120,66\} \to 1$	$\{120,258\} \to 2$	$\{120,272\} \to 2$	$\{120,277\} \to 2$

$\{121, 41\} \to 3$	$\{121, 82\} \to 2$	$\{121, 166\} \to 3$			
$\{122, 25\} \to 2$	$\{122, 48\} \to 3$	$\{122, 356\} \to 2$	$\{122, 371\} \to 1$		
$\{123, 5\} \to 1$	$\{123, 108\} \to 3$	$\{123, 145\} \to 1$	$\{123, 172\} \to 2$	$\{123, 268\} \to 1$	
$\{124, 31\} \to 3$	$\{124, 35\} \to 3$	$\{124, 398\} \to 2$			
$\{125, 201\} \to 2$	$\{125, 264\} \to 2$	$\{125, 398\} \to 4$			
$\{126, 107\} \to 2$	$\{126, 226\} \to 1$	$\{126, 233\} \to 1$	$\{126, 276\} \to 1$	$\{126, 278\} \to 2$	$\{126, 348\} \to 1$
$\{127, 68\} \to 1$	$\{127, 360\} \to 4$	$\{127, 366\} \to 3$			
$\{128, 104\} \to 1$	$\{128, 106\} \to 1$	$\{128, 152\} \to 4$	$\{128, 378\} \to 1$	$\{128, 392\} \to 1$	
$\{129, 64\} \to 5$	$\{129, 239\} \to 3$				
$\{130, 3\} \to 3$	$\{130, 311\} \to 1$	$\{130, 323\} \to 3$	$\{130, 338\} \to 1$		
$\{131, 162\} \to 2$	$\{131, 255\} \to 2$	$\{131, 385\} \to 2$	$\{131, 398\} \to 1$		
$\{132, 267\} \to 1$	$\{132, 303\} \to 1$	$\{132, 357\} \to 2$	$\{132, 391\} \to 2$	$\{132, 400\} \to 2$	
$\{133, 129\} \to 1$	$\{133, 283\} \to 2$	$\{133, 286\} \to 1$	$\{133, 325\} \to 3$	$\{133, 344\} \to 1$	
$\{134, 41\} \to 2$	$\{134, 80\} \to 2$	$\{134, 119\} \to 1$	$\{134, 131\} \to 2$	$\{134, 254\} \to 1$	
$\{135, 45\} \to 2$	$\{135, 249\} \to 1$	$\{135, 282\} \to 1$	$\{135, 293\} \to 3$	$\{135, 319\} \to 1$	
$\{136, 107\} \to 5$	$\{136, 261\} \to 2$				
$\{137, 7\} \to 2$	$\{137, 79\} \to 5$				
$\{138, 164\} \to 1$	$\{138, 198\} \to 2$	$\{138, 270\} \to 2$	$\{138, 284\} \to 1$	$\{138, 286\} \to 1$	$\{138, 356\} \to 1$
$\{139, 44\} \to 3$	$\{139, 257\} \to 2$	$\{139, 279\} \to 1$	$\{139, 305\} \to 2$		
$\{140, 185\} \to 2$	$\{140, 257\} \to 1$	$\{140, 268\} \to 3$			
$\{141, 281\} \to 1$	$\{141, 285\} \to 7$				
$\{142, 66\} \to 1$	$\{142, 105\} \to 5$	$\{142, 375\} \to 2$			
$\{143, 186\} \to 1$	$\{143, 194\} \to 2$	$\{143, 370\} \to 1$	$\{143, 388\} \to 3$		
$\{144, 19\} \to 3$	$\{144, 306\} \to 1$	$\{144, 329\} \to 1$	$\{144, 359\} \to 3$		
$\{145, 63\} \to 2$	$\{145, 242\} \to 1$	$\{145, 245\} \to 4$	$\{145, 381\} \to 1$		
$\{146, 172\} \to 1$	$\{146, 198\} \to 1$	$\{146, 206\} \to 1$	$\{146, 301\} \to 1$	$\{146, 337\} \to 1$	$\{146, 343\} \to 3$
$\{147, 4\} \to 2$	$\{147, 132\} \to 1$	$\{147, 380\} \to 3$	$\{147, 383\} \to 1$		
$\{148, 29\} \to 2$	$\{148, 133\} \to 1$	$\{148, 281\} \to 1$	$\{148, 282\} \to 2$	$\{148, 313\} \to 2$	
$\{149, 1\} \to 2$	$\{149, 62\} \to 1$	$\{149, 249\} \to 1$	$\{149, 386\} \to 4$		
$\{150, 82\} \to 2$	$\{150, 225\} \to 2$	$\{150, 294\} \to 2$	$\{150, 354\} \to 1$	$\{150, 378\} \to 1$	

7.3 The Abilities of the Workgroups

```
S1 = {6, 31, 54, 56, 59, 94, 100, 196, 230, 242, 256, 262, 263, 337, 360}
S2 = {46, 79, 87, 112, 130, 151, 167, 216, 240, 258, 270, 279, 289, 294, 356, 357, 379}
S3 = {31, 38, 62, 65, 91, 114, 118, 149, 156, 167, 173, 181, 254, 268, 269, 270, 286, 362}
S4 = {2, 4, 46, 58, 82, 90, 116, 169, 178, 216, 249, 252, 262, 279, 281, 296, 313, 337, 345, 346, 377}
S5 = {1, 34, 43, 58, 69, 90, 131, 143, 150, 174, 223, 250, 294, 316, 350, 359, 363, 382}
S6 = {24, 38, 54, 78, 86, 88, 90, 137, 149, 160, 173, 205, 217, 280, 301, 303, 317, 365, 387, 400}
S7 = {10, 11, 36, 52, 79, 94, 113, 149, 174, 181, 229, 252, 254, 260, 293, 304, 362, 375, 388}
S8 = {29, 39, 52, 91, 99, 106, 109, 147, 149, 176, 179, 188, 195, 244, 282, 290, 296, 313, 326, 331, 343,
      362, 394, 396}
S9 = {1, 19, 22, 25, 38, 60, 79, 106, 131, 136, 195, 201, 206, 217, 227, 257, 260, 263, 267, 280, 286, 333,
      346, 361, 377}
S10 = {10, 22, 31, 35, 49, 72, 87, 135, 182, 190, 202, 209, 212, 224, 241, 271, 300, 318, 326, 364}
S11 = {42, 44, 58, 63, 104, 109, 117, 123, 134, 137, 151, 209, 217, 230, 285, 302, 319, 341, 380, 381}
S12 = {3, 12, 19, 31, 39, 40, 61, 77, 78, 93, 94, 97, 101, 117, 122, 126, 156, 170, 181, 207, 221, 254, 261,
       269, 280, 287, 299, 301, 331, 339, 340, 363, 364, 375}
S13 = {42, 43, 44, 70, 83, 115, 118, 126, 134, 150, 183, 214, 225, 256, 266, 303, 335, 351, 364}
S14 = {25, 91, 100, 124, 172, 222, 275, 306, 329, 332, 350, 363, 368}
S15 = {8, 28, 38, 41, 44, 66, 92, 98, 125, 130, 136, 175, 189, 193, 263, 264, 272, 275, 288, 304, 315, 318,
      320, 336, 342, 346, 351, 353, 370, 371}
S16 = {5, 8, 20, 42, 46, 69, 70, 80, 85, 119, 138, 139, 147, 205, 227, 271, 295, 296, 349, 361, 370}
S17 = {13, 21, 29, 34, 46, 76, 80, 104, 125, 168, 177, 180, 185, 196, 198, 218, 271, 291, 294, 312, 377,
      389, 393}
S18 = {2, 12, 45, 90, 95, 144, 165, 184, 232, 291, 294, 295, 310, 333, 353}
S19 = {37, 59, 71, 79, 84, 125, 150, 168, 191, 217, 266, 287, 339, 387}
S20 = {12, 66, 75, 96, 109, 158, 170, 176, 218, 223, 258, 288, 293, 336, 342, 362}
S21 = {5, 96, 117, 119, 122, 129, 132, 138, 157, 160, 166, 171, 216, 220, 235, 237, 259, 265, 282, 304, 344,
       348, 350, 362}
S22 = {26, 66, 96, 113, 116, 161, 168, 175, 184, 204, 259, 298, 299, 350, 352, 354, 374, 394}
S23 = {14, 19, 45, 47, 69, 100, 119, 125, 126, 132, 162, 196, 197, 201, 235, 243, 259, 307, 308, 319, 321}
S24 = {32, 41, 75, 154, 180, 183, 213, 224, 264, 308, 324, 387, 388, 394}
S25 = {43, 60, 80, 86, 117, 158, 176, 191, 202, 217, 230, 270, 291, 295, 320, 392, 397}
S26 = {32, 75, 76, 90, 103, 133, 154, 179, 213, 244, 247, 260, 266, 311, 335, 339, 387, 396, 399}
S27 = {13, 22, 41, 63, 67, 74, 89, 91, 128, 159, 169, 200, 204, 207, 208, 209, 217, 258, 290, 304, 307, 318,
       321, 343, 358, 384, 388, 398}
S28 = {45, 60, 67, 75, 112, 120, 163, 187, 190, 249, 252, 258, 259, 302, 303, 327, 343, 379, 388}
S29 = {3, 33, 60, 63, 75, 100, 147, 149, 185, 200, 203, 220, 230, 282, 283, 287, 289, 304, 313, 316, 356,
      364}
S30 = {29, 77, 110, 117, 163, 205, 265, 338, 393, 399}
S31 = {85, 100, 105, 108, 126, 142, 171, 211, 225, 232, 233, 278, 319, 328, 334, 380}
S32 = {9, 45, 48, 54, 97, 113, 151, 172, 174, 192, 207, 241, 245, 247, 284, 321, 336, 360, 387}
S33 = {18, 34, 56, 60, 62, 79, 101, 103, 110, 124, 196, 210, 222, 231, 272, 290, 358, 395}
S34 = {51, 56, 61, 64, 85, 89, 94, 136, 172, 216, 218, 230, 248, 261, 262, 266, 275, 280, 286, 295, 329,
      338, 348}
S35 = {8, 32, 68, 84, 94, 142, 159, 165, 200, 208, 263, 266, 295, 301, 322, 325, 328, 331, 348, 350, 397}
S36 = {17, 25, 35, 50, 87, 146, 153, 161, 169, 180, 189, 232, 237, 254, 256, 262, 267, 269, 271, 279, 288,
      291, 293, 312, 349, 351, 352, 362, 370, 376, 379, 397}
S37 = {65, 84, 86, 131, 133, 162, 175, 182, 196, 212, 214, 218, 231, 242, 244, 258, 259, 294, 330, 364, 382,
       389, 395}
S38 = {25, 71, 72, 93, 100, 118, 128, 137, 166, 176, 241, 244, 261, 265, 272, 285, 309, 312, 313, 323, 353,
      367}
S39 = {16, 30, 31, 36, 40, 66, 95, 98, 100, 106, 137, 168, 170, 197, 202, 218, 225, 227, 234, 237, 261, 309,
       350, 355, 380}
S40 = {9, 66, 74, 78, 101, 200, 216, 273, 280, 315, 330, 336}
S41 = {1, 28, 34, 36, 102, 126, 152, 156, 183, 192, 202, 203, 214, 223, 240, 244, 255, 270, 291, 308, 312,
      325, 357, 375}
S42 = {33, 36, 72, 86, 90, 114, 136, 174, 180, 182, 188, 195, 262, 294, 302, 318, 364, 391}
S43 = {13, 74, 85, 121, 125, 127, 142, 228, 300, 303, 357, 371, 389, 391}
S44 = {60, 71, 87, 95, 116, 134, 177, 257, 274, 282, 326, 369, 373, 379, 397}
S45 = {7, 35, 39, 83, 109, 157, 171, 188, 196, 227, 242, 255, 264, 285, 315, 316, 339, 352, 364, 383, 397}
S46 = {19, 20, 65, 72, 123, 130, 139, 152, 174, 202, 207, 241, 287, 318, 337, 349, 361, 375, 382}
S47 = {12, 22, 26, 41, 51, 70, 72, 100, 103, 159, 171, 173, 178, 184, 187, 207, 234, 241, 270, 314, 328,
      334, 337, 365}
S48 = {33, 51, 52, 60, 63, 68, 81, 87, 110, 116, 181, 202, 218, 235, 256, 266, 283, 307, 322, 329, 340, 347,
       356, 369, 377, 388, 390}
S49 = {27, 35, 56, 76, 102, 117, 125, 131, 145, 156, 182, 192, 219, 226, 353, 364, 365}
S50 = {14, 22, 25, 29, 45, 48, 67, 71, 99, 119, 201, 206, 210, 213, 323, 361, 364, 368, 372, 378, 384, 396}
S51 = {8, 41, 47, 79, 106, 131, 180, 191, 206, 257, 293, 297, 381}
S52 = {5, 9, 49, 63, 78, 89, 94, 100, 106, 143, 161, 181, 196, 224, 226, 231, 290, 326, 339, 376, 398}
S53 = {14, 17, 37, 56, 57, 79, 82, 117, 205, 261, 278, 282, 295, 317, 322, 327, 334, 337}
S54 = {7, 13, 21, 30, 31, 47, 70, 86, 128, 135, 247, 267, 274, 283, 293, 350, 369, 377}
S55 = {14, 24, 36, 47, 58, 65, 78, 91, 181, 191, 202, 221, 234, 235, 264, 266, 324, 341, 363, 376, 380, 387,
       397}
S56 = {1, 12, 14, 73, 77, 87, 95, 103, 117, 123, 141, 156, 198, 218, 221, 246, 299, 311, 312, 314, 342, 356,
       363}
S57 = {40, 52, 55, 56, 64, 78, 109, 153, 171, 223, 224, 268, 307, 308, 343, 353, 367, 377, 396}
S58 = {14, 46, 53, 59, 105, 133, 175, 183, 189, 215, 258, 259, 264, 283, 284, 296, 367, 371, 398}
S59 = {4, 41, 48, 57, 63, 76, 93, 98, 126, 127, 165, 167, 171, 174, 178, 185, 214, 219, 225, 261, 270, 271,
      296, 355, 385, 387, 396, 397}
S60 = {32, 55, 64, 66, 111, 116, 141, 148, 161, 168, 169, 207, 218, 246, 264, 291, 297, 313, 396}
S61 = {22, 24, 29, 50, 62, 70, 94, 106, 108, 109, 121, 168, 187, 204, 229, 239, 245, 267, 280, 297, 306,
      314, 354, 378, 389}
S62 = {6, 33, 52, 84, 90, 102, 126, 235, 245, 252, 257, 264, 302, 307, 310, 360, 373}
S63 = {18, 79, 99, 147, 168, 169, 213, 225, 249, 264, 265, 294, 344, 368, 370, 390}
S64 = {7, 8, 12, 121, 127, 141, 147, 166, 213, 214, 236, 246, 272, 292, 312, 322, 328, 339, 349, 356, 372,
       387}
S65 = {15, 34, 58, 174, 181, 243, 266, 269, 278, 279, 294, 320, 355, 394}
S66 = {1, 15, 27, 30, 43, 55, 100, 118, 125, 165, 167, 202, 218, 219, 220, 228, 262, 263, 293, 301, 323,
      346, 353, 376}
```

```
S67 = {27, 46, 76, 90, 103, 109, 121, 154, 167, 197, 214, 215, 216, 219, 231, 233, 247, 249, 280, 293, 308,
      330, 345, 378, 393}
S68 = {44, 55, 67, 113, 116, 191, 237, 259, 283, 297, 299, 301, 350, 364, 373, 376}
S69 = {12, 80, 125, 150, 152, 156, 194, 195, 211, 218, 224, 303, 317, 333}
S70 = {37, 59, 62, 114, 126, 137, 138, 145, 171, 202, 216, 250, 257, 258, 275, 297, 298, 354, 360, 362}
S71 = {15, 16, 65, 149, 150, 168, 172, 183, 225, 251, 287, 329, 356, 359, 368, 383, 388}
S72 = {2, 3, 10, 23, 57, 77, 97, 98, 115, 129, 158, 177, 179, 189, 221, 238, 271, 272, 275, 276, 281, 342,
      354, 355, 357, 366, 368, 397}
S73 = {17, 27, 56, 61, 92, 152, 171, 176, 187, 194, 195, 212, 221, 265, 302, 309, 321, 340, 354, 374, 376,
      381, 388}
S74 = {25, 69, 78, 82, 87, 91, 99, 103, 133, 148, 171, 207, 254, 276, 297, 319, 328, 334, 347, 398}
S75 = {7, 20, 22, 54, 58, 88, 90, 122, 131, 135, 142, 150, 162, 180, 183, 190, 205, 233, 251, 254, 285, 316,
       366, 374, 396}
S76 = {26, 34, 44, 100, 120, 128, 220, 222, 233, 246, 252, 270, 277, 278, 340, 349, 366, 371, 372, 378, 380}
S77 = {13, 49, 58, 95, 105, 119, 131, 133, 138, 154, 164, 183, 219, 228, 245, 262, 265, 303, 308, 311, 314,
      317, 352, 376, 396, 399, 400}
S78 = {9, 23, 32, 36, 91, 151, 207, 211, 219, 226, 227, 239, 246, 248, 277, 296, 304, 351}
S79 = {11, 20, 30, 71, 89, 186, 211, 213, 218, 267, 290, 311, 329, 335, 371, 374, 376, 384}
S80 = {16, 55, 77, 88, 146, 151, 173, 202, 213, 218, 226, 245, 265, 270, 300, 338, 347}
S81 = {17, 35, 43, 47, 71, 74, 121, 130, 149, 160, 183, 219, 248, 262, 278, 283, 315, 319, 328, 340, 351,
      367, 382}
S82 = {9, 27, 53, 93, 125, 134, 180, 196, 215, 217, 238, 251, 287, 289, 290, 316, 323, 350, 366, 367, 387,
      391}
S83 = {17, 52, 56, 62, 91, 94, 101, 149, 167, 173, 176, 178, 257, 261, 263, 265, 328, 368, 386}
S84 = {79, 87, 127, 129, 137, 153, 156, 189, 200, 206, 209, 217, 238, 304, 308, 318, 325, 330, 332, 333,
      347, 362, 373, 374, 382}
S85 = {1, 3, 6, 9, 28, 48, 82, 97, 102, 122, 144, 146, 169, 223, 225, 228, 233, 250, 295, 348}
S86 = {12, 28, 48, 61, 66, 93, 94, 129, 132, 151, 154, 240, 261, 302, 321, 326, 334, 352, 385, 392}
S87 = {2, 33, 34, 52, 62, 93, 111, 114, 124, 127, 144, 170, 206, 238, 248, 261, 275, 299, 317, 318, 344,
      370}
S88 = {4, 18, 20, 38, 70, 82, 83, 180, 194, 195, 230, 287, 299, 343, 385, 390}
S89 = {40, 92, 113, 168, 231, 234, 241, 245, 249, 252, 374, 397}
S90 = {3, 23, 32, 42, 53, 56, 101, 164, 166, 168, 184, 197, 251, 273, 291, 307, 355, 359, 389, 399}
S91 = {15, 20, 26, 27, 42, 86, 90, 110, 116, 252, 290, 299, 317, 366, 382, 383, 387, 390, 394}
S92 = {15, 78, 87, 119, 131, 178, 182, 204, 208, 213, 218, 230, 236, 274, 280, 292, 324, 346, 395}
S93 = {51, 52, 62, 103, 219, 227, 238, 256, 262, 277, 303, 313, 316, 330, 349, 350, 369}
S94 = {4, 50, 85, 98, 118, 129, 130, 153, 161, 166, 177, 187, 192, 193, 202, 206, 234, 241, 250, 254, 265,
      266, 291, 297, 308, 321, 324, 337, 351, 355, 361}
S95 = {3, 10, 38, 101, 111, 162, 164, 194, 221, 238, 306, 311, 314, 315, 319, 325, 331, 335, 365}
S96 = {24, 47, 59, 64, 78, 80, 93, 125, 134, 150, 164, 171, 173, 178, 182, 189, 249, 254, 311, 334, 344,
      351, 384}
S97 = {14, 27, 57, 62, 67, 93, 118, 143, 148, 155, 164, 177, 184, 190, 292, 316, 319, 321, 330, 360, 370,
      385, 386}
S98 = {46, 53, 67, 92, 98, 118, 136, 151, 181, 190, 203, 204, 275, 282, 288, 289, 296, 299, 301, 304, 320,
      323, 326, 339, 347, 383}
S99 = {7, 23, 30, 51, 66, 93, 99, 127, 129, 184, 195, 207, 212, 240, 241, 279, 282, 310, 331, 343, 380, 383,
       398}
S100 = {24, 28, 50, 57, 59, 81, 87, 88, 101, 113, 122, 125, 133, 147, 153, 154, 157, 177, 183, 197, 202,
      222, 235, 259, 298, 299, 307, 313, 333, 349, 352, 382}
S101 = {47, 87, 127, 133, 167, 175, 205, 211, 222, 226, 237, 286, 289, 349, 364, 365, 378, 393}
S102 = {48, 52, 71, 76, 81, 98, 123, 141, 193, 198, 216, 219, 236, 284, 286, 295, 332, 347, 370, 384, 394}
S103 = {49, 51, 99, 130, 153, 183, 219, 227, 233, 242, 250, 271, 287, 313, 360, 365, 382}
S104 = {15, 16, 47, 69, 72, 107, 146, 148, 175, 196, 202, 231, 249, 255, 314, 322, 346, 383}
S105 = {32, 42, 53, 112, 115, 117, 145, 158, 176, 180, 211, 262, 305, 317, 388, 391}
S106 = {17, 34, 54, 57, 70, 87, 113, 147, 150, 163, 164, 167, 217, 272, 281, 294, 304, 305, 318, 388}
S107 = {19, 21, 26, 36, 109, 116, 176, 192, 195, 196, 204, 238, 241, 245, 263, 272, 301, 328, 340, 353, 383,
       388}
S108 = {9, 12, 16, 57, 71, 140, 242, 253, 255, 274, 277, 281, 288, 295, 339, 391, 397}
S109 = {44, 55, 88, 94, 122, 123, 166, 171, 178, 189, 192, 206, 219, 232, 262, 275, 323, 336, 340, 375}
S110 = {49, 56, 69, 85, 121, 128, 138, 140, 166, 172, 186, 211, 216, 222, 235, 278, 279, 280, 289, 301, 316,
       320, 322, 332, 347}
S111 = {3, 42, 47, 51, 79, 97, 121, 122, 126, 127, 130, 140, 178, 182, 193, 196, 239, 248, 251, 266, 270,
      284, 285, 287, 299, 302, 327, 330, 346, 373}
S112 = {53, 113, 115, 148, 154, 196, 208, 217, 227, 228, 229, 254, 292, 302, 319, 339, 353, 355, 364}
S113 = {9, 38, 65, 125, 128, 155, 159, 173, 186, 205, 209, 240, 249, 251, 260, 262, 283, 297, 369, 393}
S114 = {37, 93, 95, 109, 113, 121, 138, 145, 149, 199, 203, 219, 222, 223, 234, 240, 263, 269, 305, 316,
      323, 326, 347, 372, 393, 394, 395}
S115 = {1, 5, 14, 23, 78, 94, 96, 110, 148, 219, 239, 245, 246, 252, 263, 266, 270, 280, 290, 328, 354, 383,
       389, 394}
S116 = {1, 29, 37, 55, 60, 77, 89, 100, 110, 120, 138, 151, 189, 191, 212, 234, 264, 277, 307, 312, 320,
      321, 329, 333, 335, 341, 343, 351, 374, 375}
S117 = {2, 41, 55, 89, 97, 147, 245, 249, 293, 295, 314, 316, 317, 335, 345, 360, 398, 399}
S118 = {32, 75, 79, 84, 87, 126, 163, 164, 191, 202, 213, 222, 229, 267, 270, 276, 280, 296, 339, 344, 368,
      382, 389}
S119 = {9, 37, 50, 95, 115, 116, 133, 143, 145, 148, 150, 155, 165, 181, 194, 206, 219, 222, 246, 248, 286,
      289, 369}
S120 = {4, 45, 50, 66, 77, 112, 143, 223, 233, 253, 258, 272, 277, 322, 336, 338, 386}
S121 = {36, 41, 66, 72, 82, 122, 131, 140, 146, 149, 166, 227, 235, 268, 269, 286, 292, 316, 317, 333, 348,
      353, 359, 386}
S122 = {25, 34, 35, 48, 54, 116, 137, 149, 156, 165, 166, 168, 171, 187, 201, 202, 214, 215, 247, 274, 277,
      314, 356, 357, 366, 371}
S123 = {2, 5, 11, 17, 96, 97, 104, 108, 116, 120, 141, 145, 172, 181, 217, 246, 268, 341, 349, 383, 400}
S124 = {31, 35, 128, 130, 163, 193, 196, 198, 217, 245, 265, 270, 285, 294, 308, 330, 377, 398}
S125 = {58, 90, 120, 155, 169, 188, 193, 201, 226, 245, 254, 260, 264, 351, 363, 390, 398}
S126 = {10, 15, 40, 69, 95, 106, 107, 143, 201, 226, 233, 276, 278, 322, 343, 348, 357}
S127 = {15, 38, 50, 68, 118, 143, 156, 159, 210, 222, 263, 274, 324, 360, 366}
S128 = {23, 50, 75, 94, 98, 104, 106, 116, 125, 134, 151, 152, 194, 195, 220, 238, 258, 281, 285, 291, 293,
      294, 304, 353, 378, 392}
S129 = {25, 64, 90, 104, 133, 175, 183, 197, 206, 217, 230, 239, 282, 306, 356, 382}
```

```
S130 = {3, 39, 47, 95, 101, 131, 136, 151, 153, 172, 175, 190, 200, 203, 212, 224, 234, 257, 282, 311, 323,
      332, 338, 360}
S131 = {27, 39, 60, 62, 101, 112, 133, 143, 148, 156, 162, 166, 181, 207, 208, 215, 234, 239, 255, 293, 306,
       325, 332, 353, 367, 385, 393, 398}
S132 = {42, 47, 51, 59, 108, 110, 112, 141, 147, 151, 185, 201, 203, 234, 239, 240, 261, 264, 267, 303, 327,
       351, 357, 383, 388, 391, 400}
S133 = {36, 101, 104, 129, 131, 194, 213, 239, 254, 283, 286, 325, 344}
S134 = {5, 9, 14, 15, 19, 38, 41, 80, 119, 131, 206, 252, 254, 278, 297, 313, 319, 339, 364, 366, 382, 384,
      387, 389}
S135 = {13, 14, 43, 45, 112, 122, 123, 146, 177, 181, 231, 246, 247, 249, 258, 282, 290, 293, 314, 319, 332,
       361, 391}
S136 = {14, 45, 107, 121, 123, 144, 179, 190, 222, 238, 241, 261, 278, 294, 381}
S137 = {7, 14, 59, 65, 70, 79, 96, 116, 129, 134, 169, 174, 193, 214, 237, 260, 265, 268, 292, 315, 346,
      374, 375, 383, 393}
S138 = {6, 21, 49, 86, 98, 99, 113, 143, 159, 164, 198, 267, 270, 284, 286, 356}
S139 = {3, 18, 21, 44, 58, 72, 76, 90, 93, 107, 148, 155, 164, 183, 190, 222, 257, 264, 279, 305, 312, 319,
      332, 346, 347, 351, 384}
S140 = {13, 34, 37, 94, 106, 122, 134, 141, 155, 156, 158, 183, 185, 221, 250, 257, 268, 319, 389, 398}
S141 = {17, 32, 65, 77, 95, 111, 140, 178, 220, 226, 236, 260, 266, 281, 285, 304, 310, 349, 350, 355, 358,
      363, 390, 391, 400}
S142 = {8, 25, 41, 57, 62, 66, 102, 105, 153, 178, 180, 219, 226, 259, 275, 356, 367, 375}
S143 = {13, 36, 54, 58, 80, 86, 88, 95, 152, 162, 165, 186, 194, 326, 341, 342, 345, 370, 380, 388}
S144 = {3, 19, 58, 95, 111, 114, 149, 158, 179, 215, 271, 278, 294, 306, 307, 315, 329, 343, 347, 359, 384,
      393}
S145 = {20, 27, 55, 63, 110, 118, 123, 124, 129, 156, 166, 202, 242, 244, 245, 257, 276, 277, 284, 290, 320,
       322, 381, 387}
S146 = {4, 54, 76, 85, 97, 121, 171, 172, 198, 206, 286, 301, 325, 337, 343, 373}
S147 = {4, 79, 131, 132, 144, 206, 209, 214, 219, 253, 262, 264, 303, 321, 354, 362, 380, 383}
S148 = {29, 85, 113, 133, 175, 206, 258, 281, 282, 313, 365, 374, 392}
S149 = {1, 36, 40, 48, 50, 62, 79, 158, 183, 185, 198, 238, 249, 258, 276, 289, 306, 326, 341, 342, 369,
      372, 381, 386}
S150 = {18, 48, 73, 75, 80, 82, 95, 101, 105, 110, 144, 148, 151, 173, 225, 228, 294, 299, 306, 336, 337,
      352, 354, 373, 378}
```

7.4 The Workingtimes

t1 = 2	t2 = 3	t3 = 3	t4 = 2	t5 = 2	t6 = 2	t7 = 6	t8 = 1	t9 = 2	t10 = 1
t11 = 2	t12 = 1	t13 = 2	t14 = 6	t15 = 2	t16 = 1	t17 = 2	t18 = 1	t19 = 3	t20 = 2
t21 = 1	t22 = 13	t23 = 3	t24 = 2	t25 = 2	t26 = 4	t27 = 5	t28 = 2	t29 = 3	t30 = 3
t31 = 3	t32 = 1	t33 = 1	t34 = 2	t35 = 3	t36 = 1	t37 = 3	t38 = 1	t39 = 1	t40 = 6
t41 = 14	t42 = 1	t43 = 1	t44 = 12	t45 = 3	t46 = 2	t47 = 6	t48 = 3	t49 = 3	t50 = 3
t51 = 3	t52 = 3	t53 = 12	t54 = 2	t55 = 3	t56 = 3	t57 = 1	t58 = 2	t59 = 1	t60 = 3
t61 = 12	t62 = 1	t63 = 2	t64 = 14	t65 = 1	t66 = 2	t67 = 1	t68 = 1	t69 = 1	t70 = 6
t71 = 1	t72 = 2	t73 = 3	t74 = 1	t75 = 2	t76 = 2	t77 = 3	t78 = 3	t79 = 6	t80 = 2
t81 = 2	t82 = 9	t83 = 3	t84 = 3	t85 = 5	t86 = 3	t87 = 3	t88 = 1	t89 = 7	t90 = 2
t91 = 3	t92 = 3	t93 = 2	t94 = 3	t95 = 3	t96 = 1	t97 = 2	t98 = 2	t99 = 12	t100 = 3
t101 = 2	t102 = 11	t103 = 3	t104 = 2	t105 = 5	t106 = 5	t107 = 7	t108 = 3	t109 = 2	t110 = 4
t111 = 3	t112 = 3	t113 = 3	t114 = 2	t115 = 3	t116 = 3	t117 = 3	t118 = 1	t119 = 1	t120 = 2
t121 = 3	t122 = 1	t123 = 3	t124 = 1	t125 = 2	t126 = 1	t127 = 3	t128 = 1	t129 = 3	t130 = 2
t131 = 2	t132 = 1	t133 = 2	t134 = 3	t135 = 2	t136 = 1	t137 = 1	t138 = 2	t139 = 1	t140 = 1
t141 = 6	t142 = 1	t143 = 3	t144 = 2	t145 = 1	t146 = 1	t147 = 1	t148 = 3	t149 = 3	t150 = 11
t151 = 1	t152 = 4	t153 = 2	t154 = 2	t155 = 2	t156 = 1	t157 = 3	t158 = 1	t159 = 2	t160 = 1
t161 = 1	t162 = 2	t163 = 2	t164 = 3	t165 = 4	t166 = 3	t167 = 2	t168 = 2	t169 = 2	t170 = 2
t171 = 3	t172 = 3	t173 = 3	t174 = 1	t175 = 4	t176 = 1	t177 = 2	t178 = 3	t179 = 4	t180 = 14
t181 = 1	t182 = 2	t183 = 1	t184 = 3	t185 = 2	t186 = 1	t187 = 3	t188 = 3	t189 = 1	t190 = 2
t191 = 6	t192 = 9	t193 = 9	t194 = 3	t195 = 2	t196 = 3	t197 = 3	t198 = 3	t199 = 1	t200 = 3
t201 = 2	t202 = 2	t203 = 2	t204 = 7	t205 = 1	t206 = 1	t207 = 1	t208 = 6	t209 = 1	t210 = 3
t211 = 2	t212 = 2	t213 = 1	t214 = 2	t215 = 1	t216 = 1	t217 = 3	t218 = 3	t219 = 2	t220 = 1
t221 = 1	t222 = 10	t223 = 2	t224 = 3	t225 = 2	t226 = 3	t227 = 2	t228 = 2	t229 = 2	t230 = 5
t231 = 1	t232 = 12	t233 = 1	t234 = 3	t235 = 6	t236 = 6	t237 = 2	t238 = 2	t239 = 3	t240 = 2
t241 = 2	t242 = 1	t243 = 4	t244 = 1	t245 = 4	t246 = 2	t247 = 2	t248 = 6	t249 = 3	t250 = 8
t251 = 9	t252 = 1	t253 = 4	t254 = 3	t255 = 6	t256 = 4	t257 = 3	t258 = 2	t259 = 2	t260 = 2
t261 = 3	t262 = 8	t263 = 1	t264 = 2	t265 = 12	t266 = 1	t267 = 3	t268 = 4	t269 = 1	t270 = 13
t271 = 8	t272 = 2	t273 = 5	t274 = 1	t275 = 3	t276 = 1	t277 = 2	t278 = 2	t279 = 1	t280 = 3
t281 = 2	t282 = 3	t283 = 2	t284 = 1	t285 = 7	t286 = 2	t287 = 1	t288 = 2	t289 = 3	t290 = 2
t291 = 2	t292 = 2	t293 = 3	t294 = 2	t295 = 4	t296 = 3	t297 = 1	t298 = 3	t299 = 1	t300 = 3
t301 = 1	t302 = 3	t303 = 9	t304 = 2	t305 = 2	t306 = 1	t307 = 2	t308 = 2	t309 = 14	t310 = 2
t311 = 2	t312 = 1	t313 = 2	t314 = 3	t315 = 1	t316 = 12	t317 = 4	t318 = 2	t319 = 2	t320 = 2
t321 = 2	t322 = 2	t323 = 5	t324 = 1	t325 = 3	t326 = 1	t327 = 2	t328 = 5	t329 = 1	t330 = 8
t331 = 3	t332 = 2	t333 = 3	t334 = 2	t335 = 1	t336 = 2	t337 = 3	t338 = 1	t339 = 2	t340 = 1
t341 = 1	t342 = 1	t343 = 3	t344 = 1	t345 = 2	t346 = 3	t347 = 1	t348 = 3	t349 = 2	t350 = 2
t351 = 4	t352 = 1	t353 = 2	t354 = 1	t355 = 3	t356 = 3	t357 = 2	t358 = 3	t359 = 3	t360 = 4
t361 = 3	t362 = 10	t363 = 2	t364 = 2	t365 = 3	t366 = 3	t367 = 1	t368 = 2	t369 = 2	t370 = 1
t371 = 1	t372 = 3	t373 = 3	t374 = 3	t375 = 3	t376 = 2	t377 = 3	t378 = 3	t379 = 1	t380 = 3
t381 = 1	t382 = 2	t383 = 1	t384 = 2	t385 = 3	t386 = 4	t387 = 2	t388 = 3	t389 = 2	t390 = 1
t391 = 2	t392 = 1	t393 = 2	t394 = 8	t395 = 3	t396 = 2	t397 = 4	t398 = 7	t399 = 3	t400 = 2