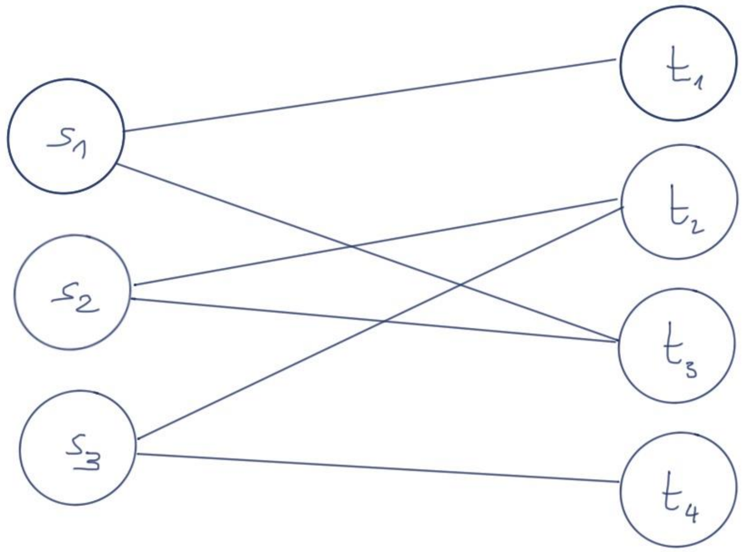
Algorithmic Matching



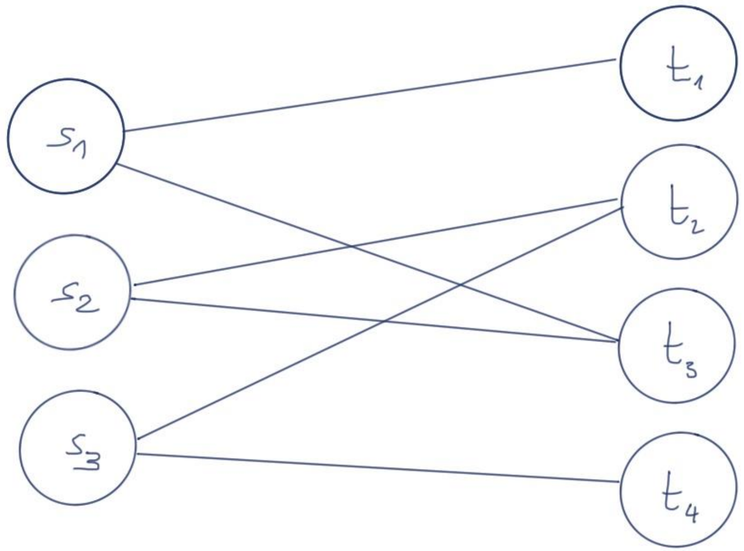
Mathematical approaches to optimally allocate topics to students

Frederik Heck

Coached by Simon Kramer

The Student-Topic-Assignment Problem

Students



Topics

… want to be assigned to …

Solution for Assignement Problems?

$G = (V, E)$

$e = (v_1, v_2) = (v_2, v_1).$

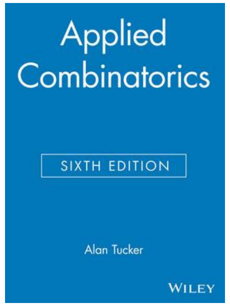
$e^{\rightarrow} = (v_1^{\rightarrow}, v_2).$

Mathematics!

$0 \leq f(e^{\rightarrow}) \leq k(e^{\rightarrow})$

$(v_{IN}^{\rightarrow}, v_g)$

$(b^{\pm}, \triangle(q))$

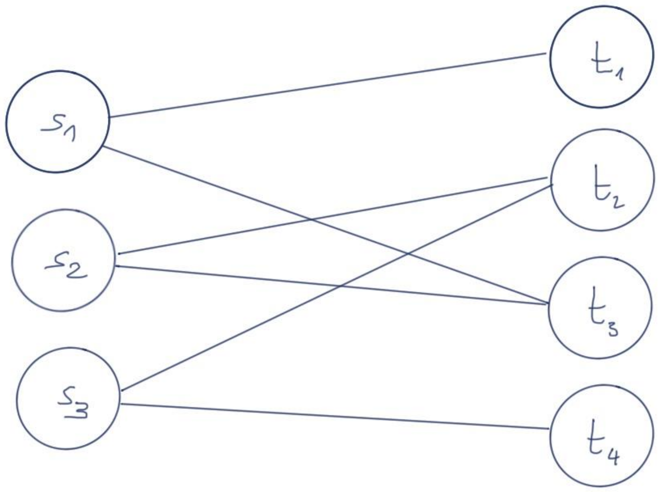


Most of the information bases on
Allen Tuckers - ***Applied Combinatorics***

[***https://www.wiley.com/en-us/Applied+Combinatorics%2C+6th+Edition-p-9781118210116***](https://www.wiley.com/en-us/Applied+Combinatorics%2C+6th+Edition-p-9781118210116)

The Approach

Phase I: Model the Problem



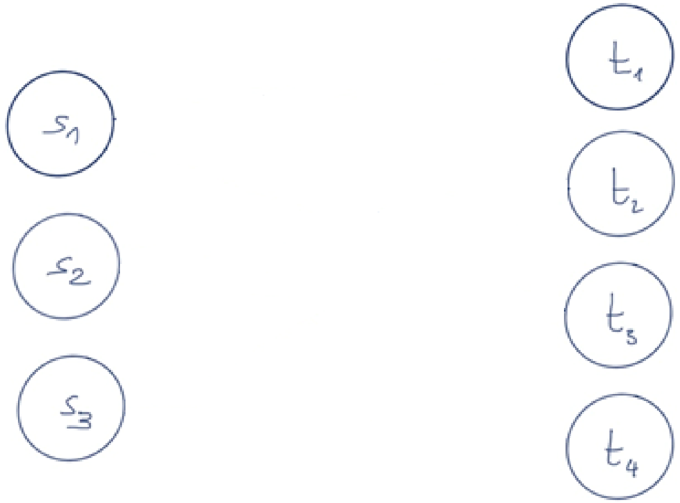
Phase II: Solve the Model

```
// start procedure
for (i=1; i<=n; i++)
  if (sol[i] == head) {
    nn = 0;
    cwk2 = big;
    for (j=1; j<=n; j++) {
      min = i;
      max = j;
      if (i != j) {
        if (i > j) {
          max = i;
          min = j;
        }
```

Phase I - Model the Problem

First Step: Make a graph

3 students



4 topics

This graph has 7 vertices and no edges

A **graph** *G* consists of two sets *V* and *E*

$$G = (V, E)$$

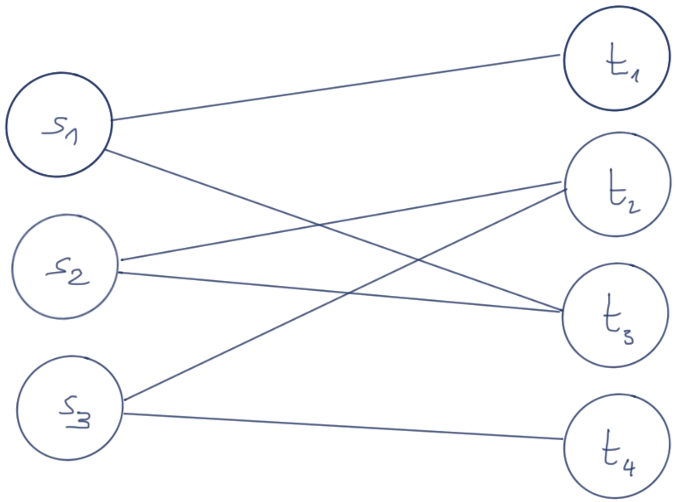
V: Vertices
E: Edges (connecting vertices)

Phase I - Model the Problem

First Step: Make a graph

Edges represent topic wishes

3 students



4 topics

This graph has 7 vertices and 6 edges

A **graph** *G* consists of two sets *V* and *E*

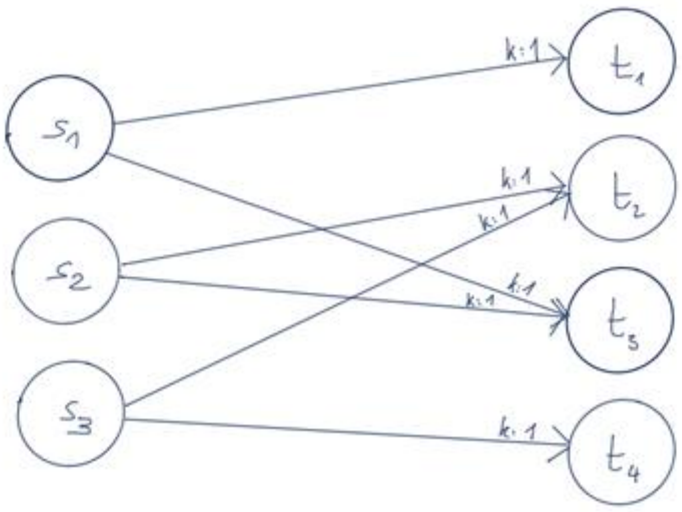
$$G = (V, E)$$

V: Vertices
E: Edges (connecting vertices)

Phase I - Model the Problem

Second Step: Transform the graph into a network

Edges are directed now



Each edge has a capacity $k(e^{\rightarrow}) = 1$

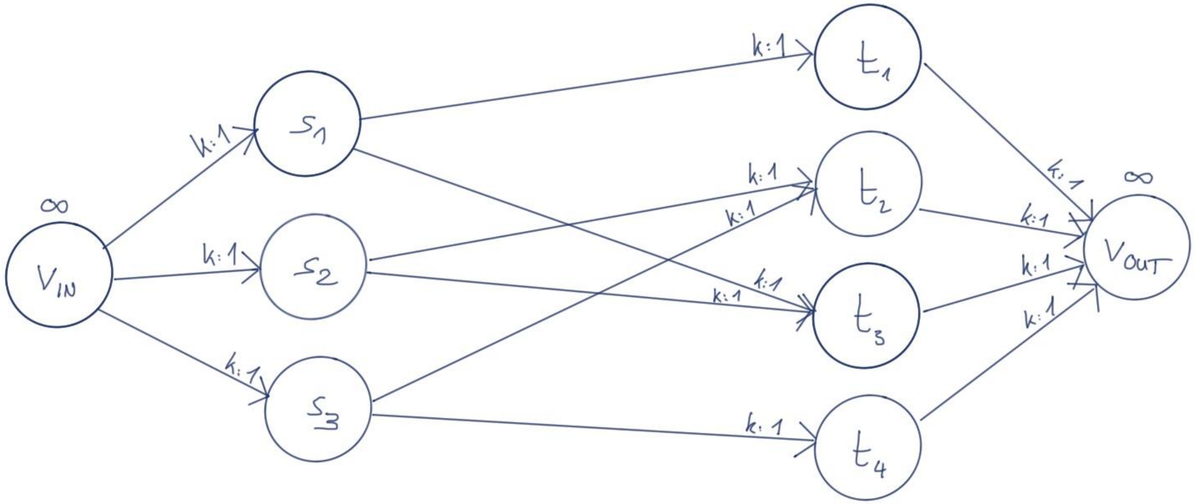
A **network** *N* is a directed graph, where:
Each edge $e^{\rightarrow}$ has a capacity $k(e^{\rightarrow})$
Each edge $e^{\rightarrow}$ has s flow $f(e^{\rightarrow})$

$$0 \leq f(e^{\rightarrow}) \leq k(e^{\rightarrow})$$

Phase I - Model the Problem

Second Step: Transform the graph into a network

We add a super-source and -sink



Each edge has a capacity $k(e^{\rightarrow}) = 1$

A **network** *N* is a directed graph, where:
Each edge $e^{\rightarrow}$ has a capacity $k(e^{\rightarrow})$
Each edge $e^{\rightarrow}$ has s flow $f(e^{\rightarrow})$

$$0 \leq f(e^{\rightarrow}) \leq k(e^{\rightarrow})$$

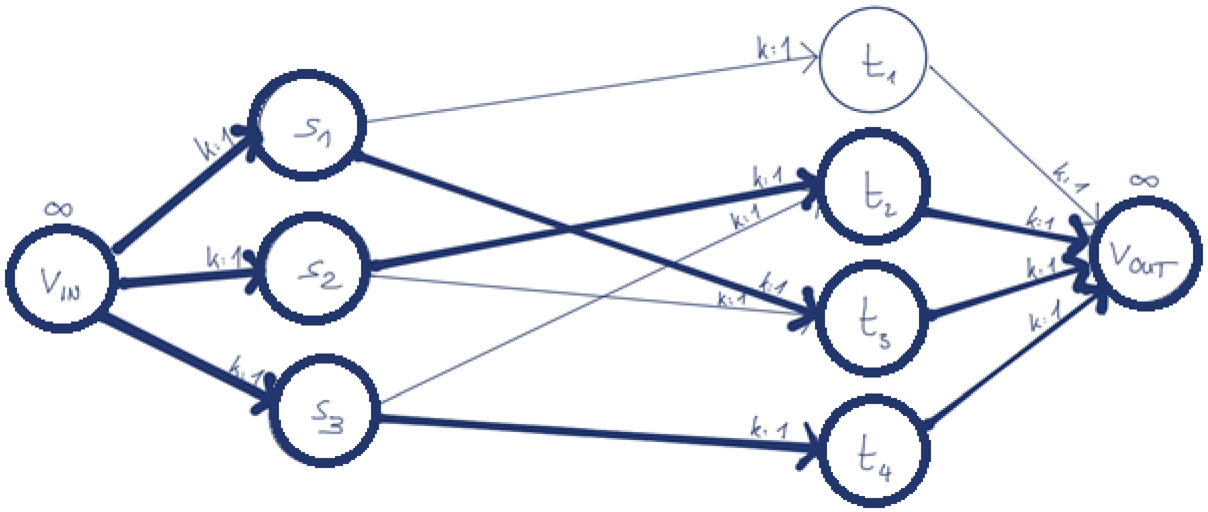
A network often has a super-source v_{IN} as well as a super-sink v_{OUT}:

v_{IN} generates all flow
v_{OUT} swallows all flow

Phase II – Solve the Model

First Step: Understand Matchings

Thick lines represent a matching



Every student is matched: Optimal!

A **matching** *M* is a set of independant edges in a graph G = (S, T, E), where:

S: First vertex set (Students)
T: Second vertex set (Topics)
E: Edges, that match vertices in *S* with vertices in *E*

Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

INIT

1. Give the super-source v_{IN} the labels (-, ∞)

LABEL

2. Now scan a vertex. Let us call the vertex being scanned p and its second label $\triangle(p)$. Initially, $p = v_{IN}$.
   (a) Check each incoming edge $e = (q^{\rightarrow}, p)$. If $f(e^{\rightarrow}) > 0$ and q is unlabeled, then label q with $(p^{-}, \triangle(q))$, where $\triangle(q) = \min[\triangle(p), f(e^{\rightarrow})]$
   (b) Check each outgoing edge $e = (p^{\rightarrow}, q)$. If $s(e^{\rightarrow}) = k(e^{\rightarrow})$ - $f(e^{\rightarrow}) > 0$ and q is unlabeled, then label q with $(p^{+}, \triangle(q))$, where $\triangle(q) = \min[\triangle(p), s(e^{\rightarrow})]$

CHECK

3. If v_{OUT} has been labeled, go to step 4. Otherwise choose another labeled vertex to be scanned (which was not yet scanned) and go to Step 2. If there are no more labled vertices to scan, let P be the set of labled vertices, and now $(P, \overline{P})$ form a saturated cut. The flow from P to $\overline{P}$ will then be maximum. From this we can conclude that we have found the maximum flow of the network, since every cut must contain the whole flow of a network. Stop the algorithm.
4. Find a v_{IN} - v_{OUT} chain of vertices, where every $\triangle(q)$ of a vertex has a positive value. The minimal number of $\triangle(q)$ in the chain, tells us by how many flow-units the flow may be increased. We increase it by adding flow of this number on all forwardly-directed edges, and removing flow of this number on all backwardly-directed edges.

REPEAT

5. Start again at 1.

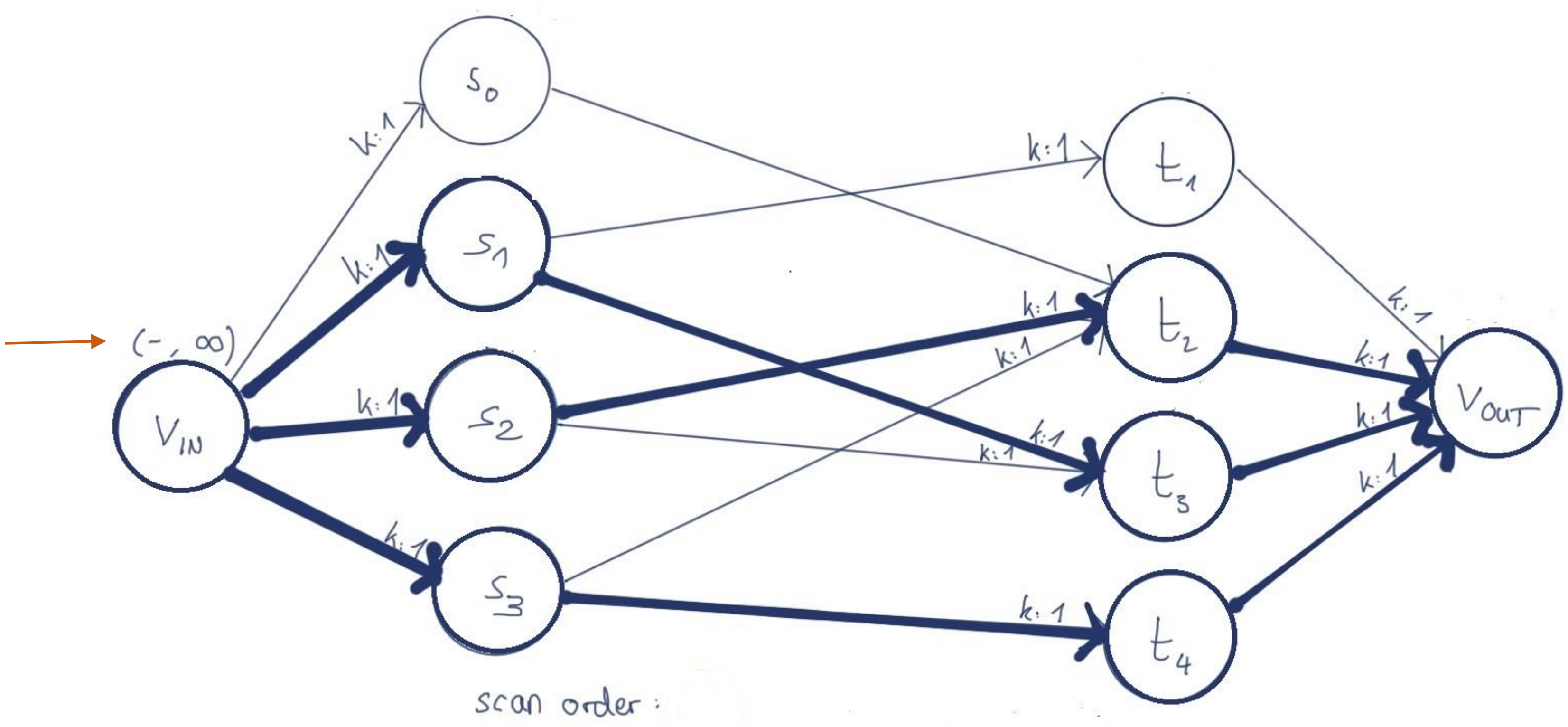
Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

INIT

Give v_{IN} a label



Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

LABEL

Scan a labled vertex:

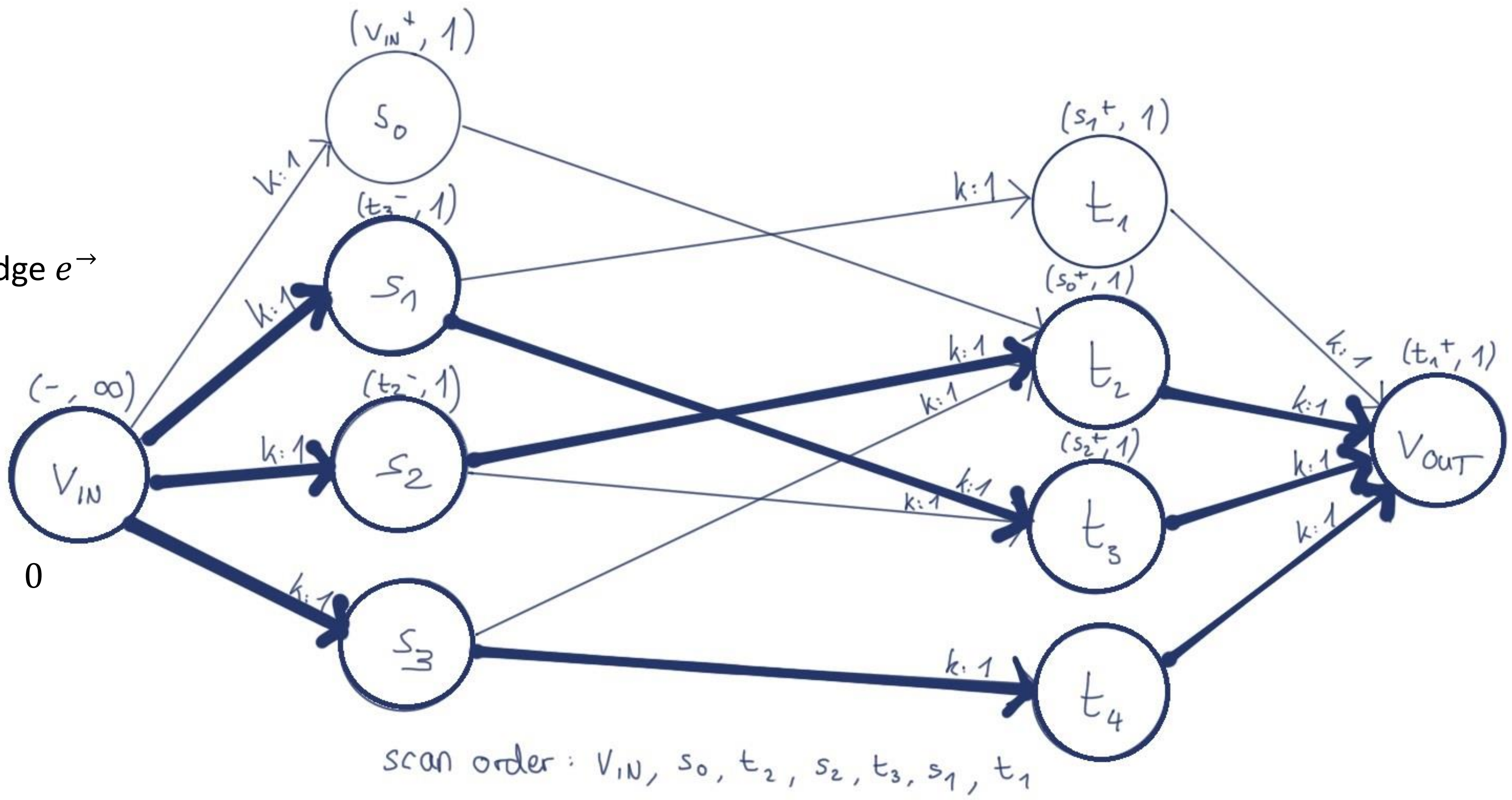
Check each incoming edge $e^{\rightarrow}$
Lable it if:

- $f(e^{\rightarrow}) > 0$
- $e^{\rightarrow}$ is unlabled

Check each outgoing edge $e^{\rightarrow}$, label it if:

- $k(e^{\rightarrow}) - f(e^{\rightarrow}) > 0$
- $e^{\rightarrow}$ is unlabled

Repeat until all labled vertices are scanned



Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

CHECK

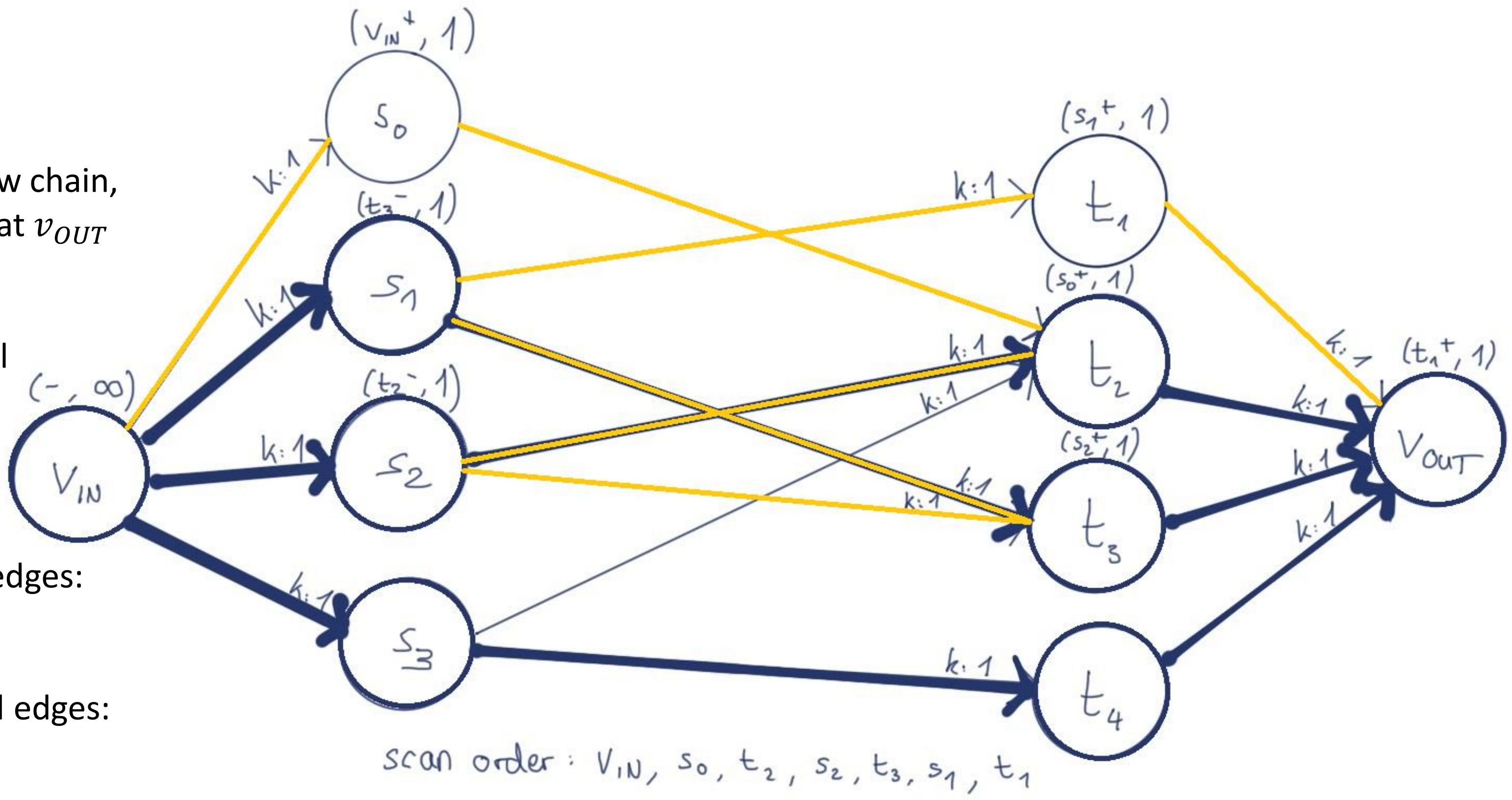
Find an augmenting flow chain,
Starting at v_{IN}, ending at v_{OUT}

If no chain is found:
The matching is optimal
End algorithm

If a chain is found:

On forwardly directed edges:
- Add flow

On backwardly directed edges:
- Remove flow



Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

CHECK

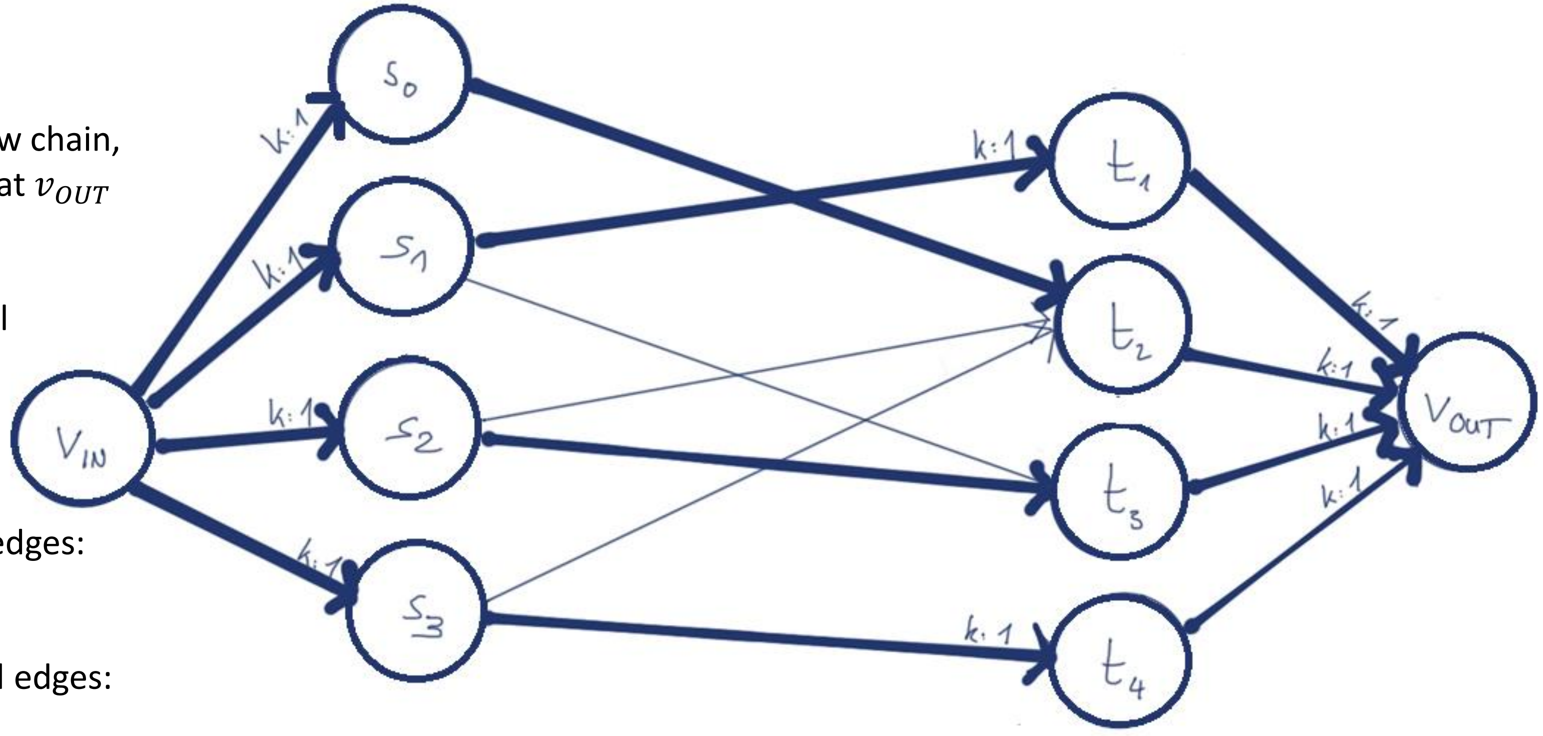
Find an augmenting flow chain,
Starting at v_{IN}, ending at v_{OUT}

If no chain is found:
The matching is optimal
End algorithm

If a chain is found:

On forwardly directed edges:
- Add flow

On backwardly directed edges:
- Remove flow



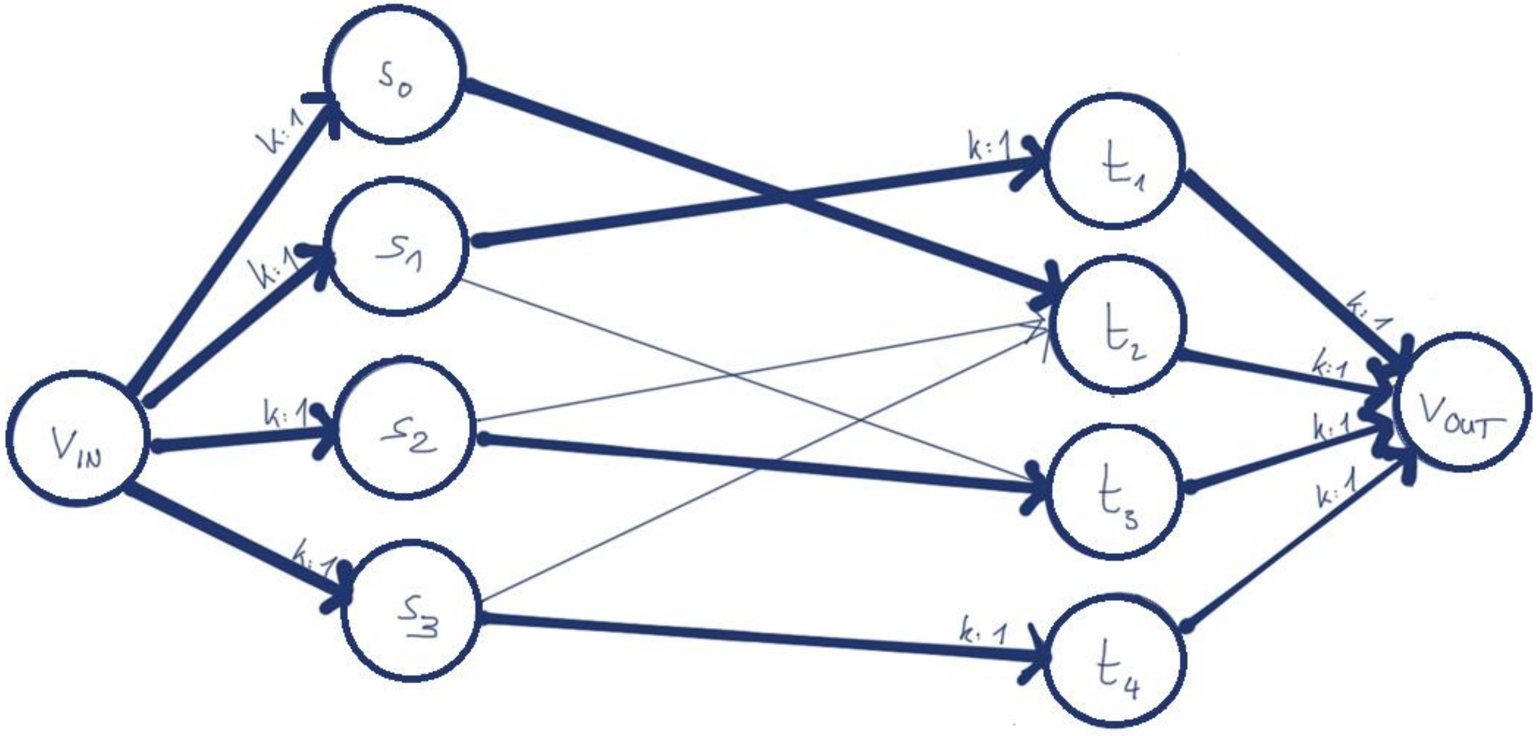
Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

REPEAT

Start again with INIT



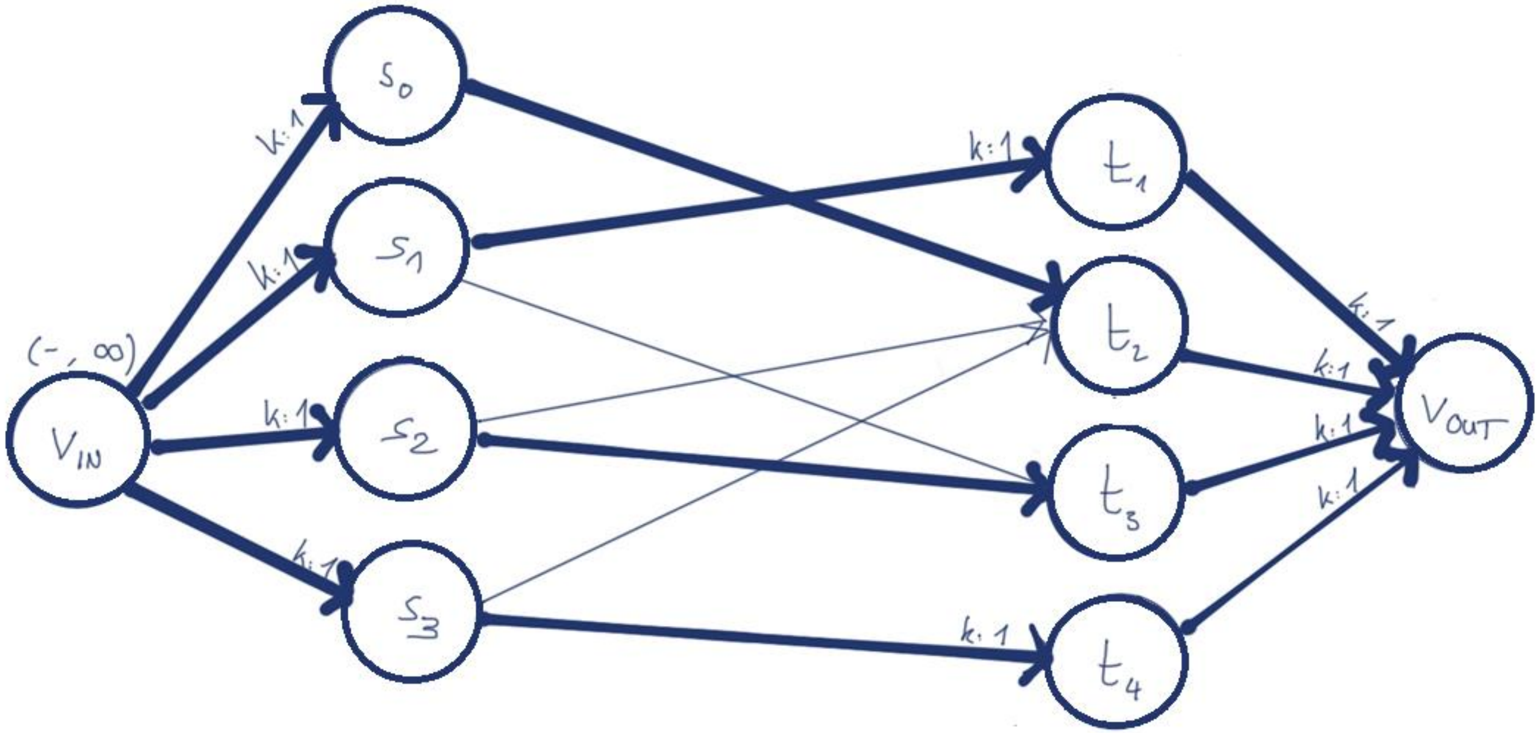
Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

INIT

Give v_{IN} a label



Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

LABEL

Scan a labled vertex:

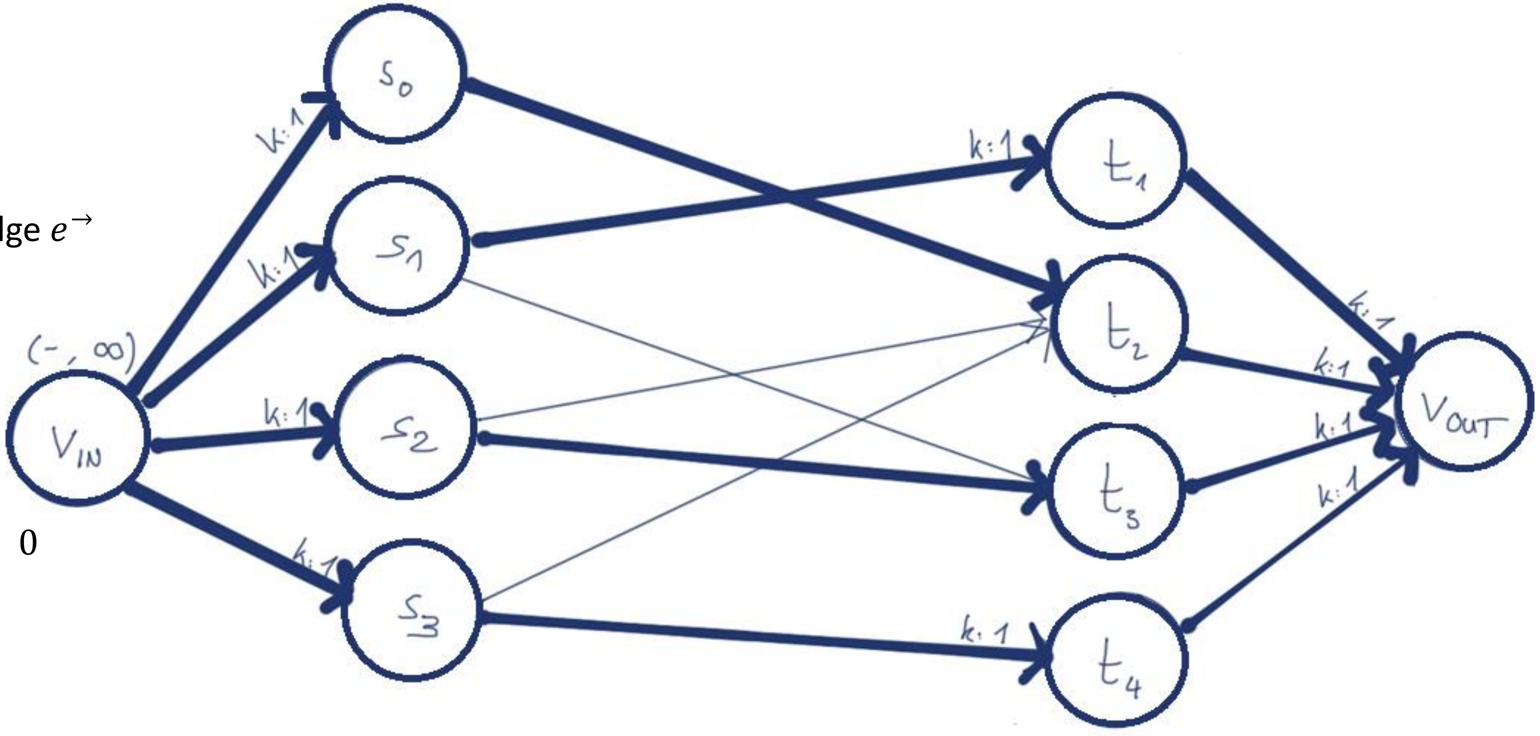
Check each incoming edge $e^{\rightarrow}$
Lable it if:

- $f(e^{\rightarrow}) > 0$
- $e^{\rightarrow}$ is unlabled

Check each outgoing edge $e^{\rightarrow}$, label it if:

- $k(e^{\rightarrow}) - f(e^{\rightarrow}) > 0$
- $e^{\rightarrow}$ is unlabled

Repeat until all labled vertices are scanned



Phase II – Solve the Model

Second Step: Develop an Algorithm

Augmenting Flow Algorithm

CHECK

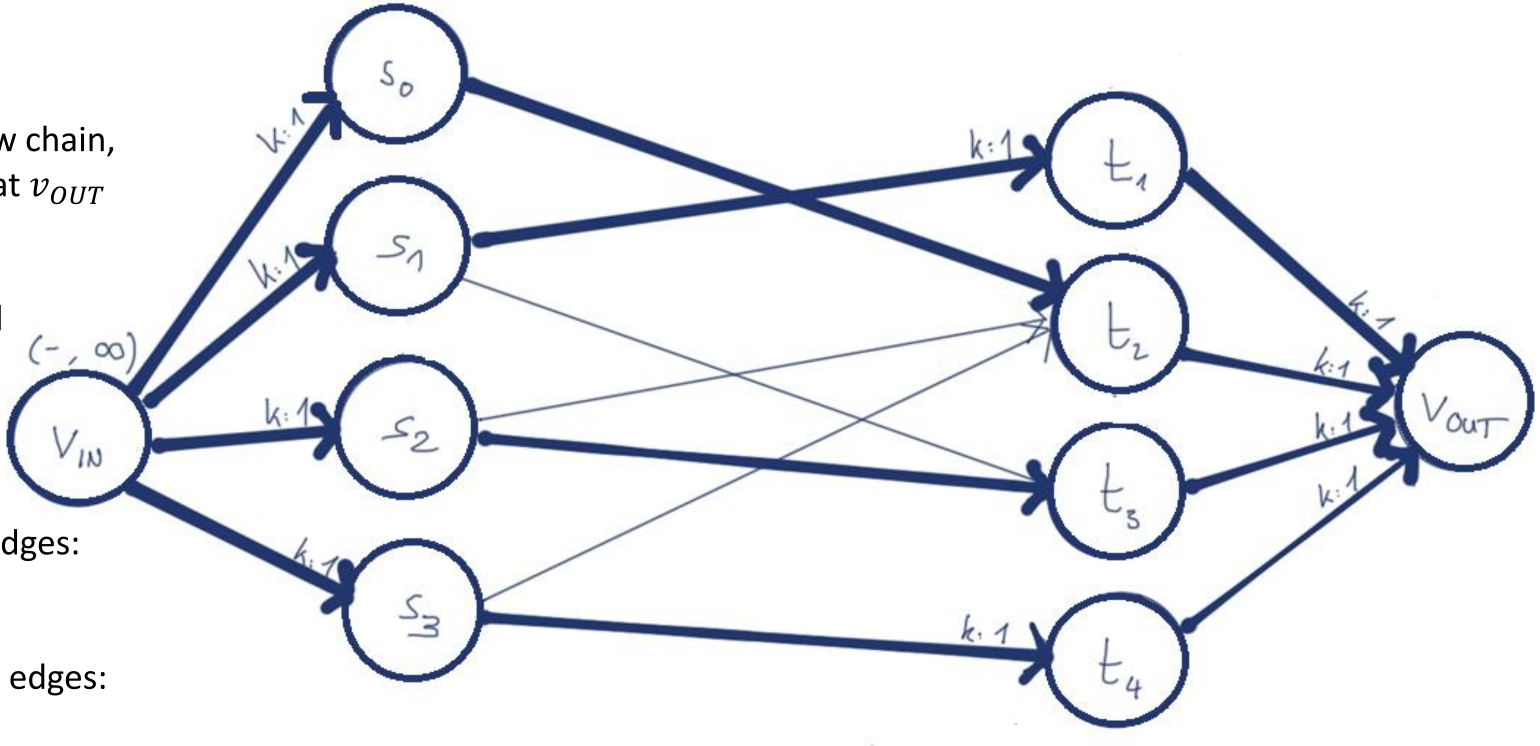
Find an augmenting flow chain,
Starting at v_{IN}, ending at v_{OUT}

If no chain is found:
The matching is optimal
End algorithm

If a chain is found:

On forwardly directed edges:
- Add flow

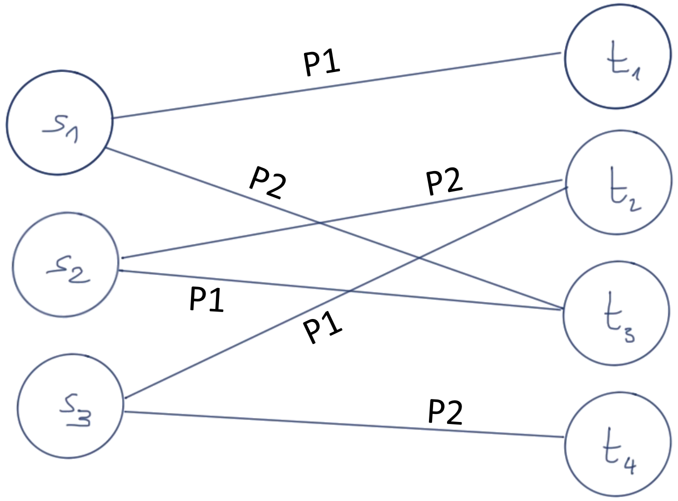
On backwardly directed edges:
- Remove flow



After-math: Improvements

We must think of two additional constraints

What about priorities?



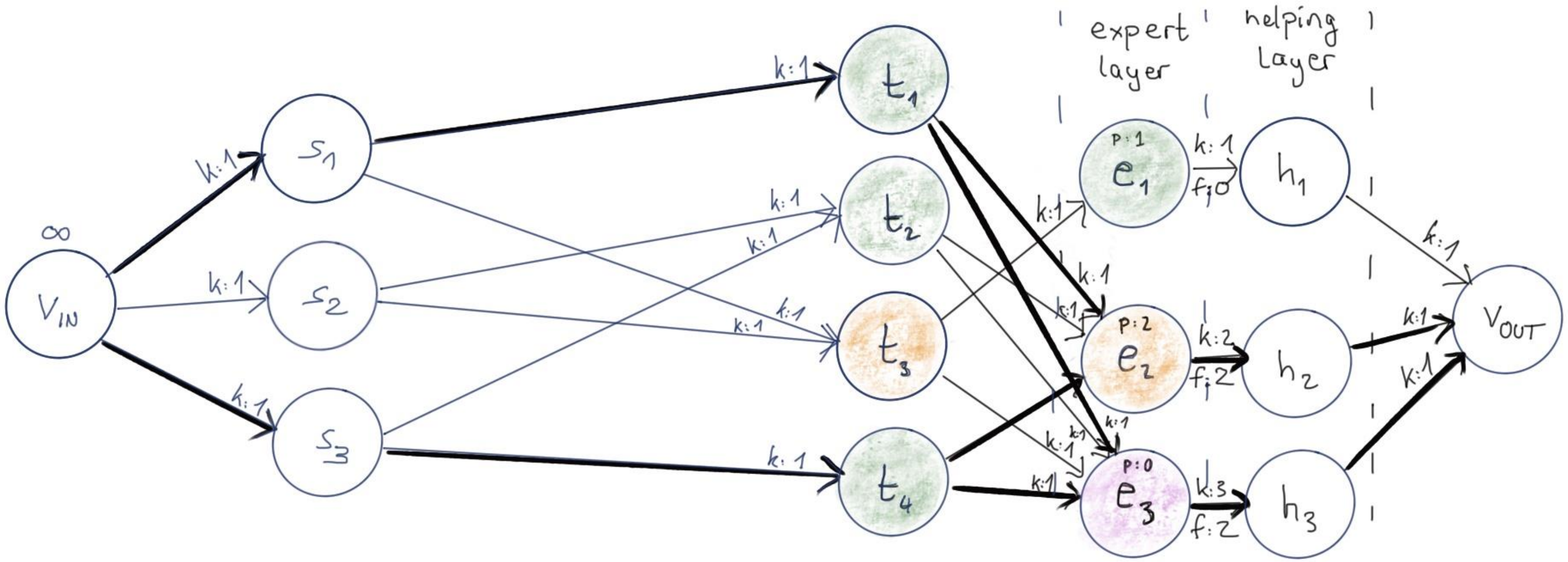
We make use of a minimization algorithm. It will find the matching with the minimal priority-sum on all optimal matchings.

Not explictly treated in my work

After-math: Improvements

We must think of two additional constraints

Experts have a minimum pensum!

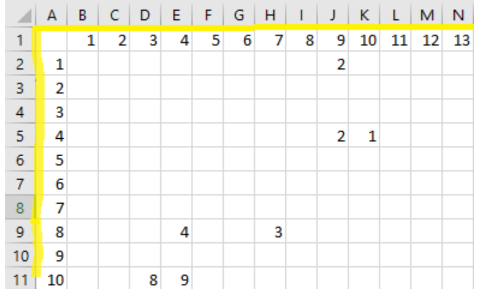


More complicated concepts are needed, explictly treated in my work

After-math: Programming

We’re ready: Let’s program!

CSV-File



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		1	2	3	4	5	6	7	8	9	10	11	12	13
2	1									2				
3	2													
4	3													
5	4									2	1			
6	5													
7	6													
8	7													
9	8				4			3						
10	9													
11	10			8	9									

JAVA-Algorithm

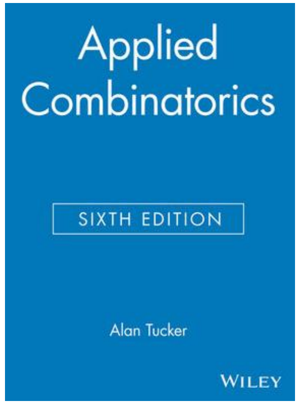
```
// start procedure
for (i=1; i<=n; i++)
  if (sol[i] == head) {
    nn = 0;
    cwk2 = big;
    for (j=1; j<=n; j++) {
      min = i;
      max = j;
      if (i != j) {
        if (i > j) {
          max = i;
          min = j;
        }
```

Mostly adopted from a library

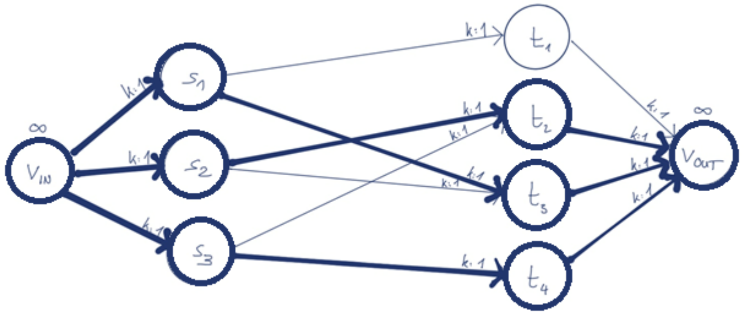
Optimal Matching

```
Optimal matching:
 s1 -- t16 [P1]
 s2 -- t36 [P1]
 s3 -- t42 [P1]
 s4 -- t10 [P1]
 s5 -- t37 [P2]
 s6 -- t22 [P1]
 s7 -- t17 [P1]
 s8 --  no matching possible
 s9 -- t23 [P1]
 s10 -- t26 [P3]
```

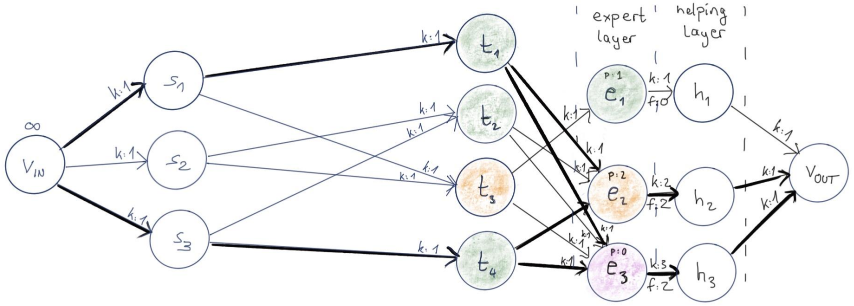
Review: How to assign students to topics



1. Math gives us a model



2. We model some specific constraints




```
// start procedure
for (i=1; i<=n; i++)
  if (sol[i] == head) {
    nn = 0;
    cwk2 = big;
    for (j=1; j<=n; j++) {
      min = i;
      max = j;
      if (i != j) {
        if (i > j) {
          max = i;
          min = j;
        }
```

3. We implement an algorithm

Questions