ADM I — Exercise Session 11

TU Berlin

January 24, 2014

Announcements

Announcements:

- presentation of PA2 in last week of semester
  your don't have to be done by then$\rightsquigarrow$ we might help you with problems
- last exercise session will repeat the lecture and deal with the written exam
- there will be 14 exercise sheets, last one counts as bonus points

Programming Exercise 1 — Simplex Algorithm

Simplex algorithms results:
code basis ranges from 1.200-10.000 lines of code

Results: solving `boeing2.lp`, timelint 60 minutes:

- 1.5 s, wrong answer
- 6s, ArithmeticException
- 19 s, 190282402289/50000, wrong answer
- 27 s, 776267449034475331069/6772252025000, wrong answer
- 32 s, infeasible, wrong answer
- 1m13s, 9159565743986777325714 43/8036693589487 5000, wrong answer
- 1m40s, infeasible, wrong answer
- 3m2.302s, ArrayIndexOutOfBoundsException
- 4m6s,64974353949012503513 6837/5749033109925 0000 , ✓

Programming Exercise 1 — Simplex Algorithm

Results for the programs handed in time **solving** `boeing2.lp`

- gna-113: 4m6s
- gna-119: 4m52
- gna-108: 6m2s
- gna-101: 6m55s
- gna-109: 11m35s
- gna-114: 27m48s
- others above 60 minutes, wrong or crashing

Programming Exercise 1 — Simplex Algorithm

solving `boeing1.lp` **with these** `6` **programs:**

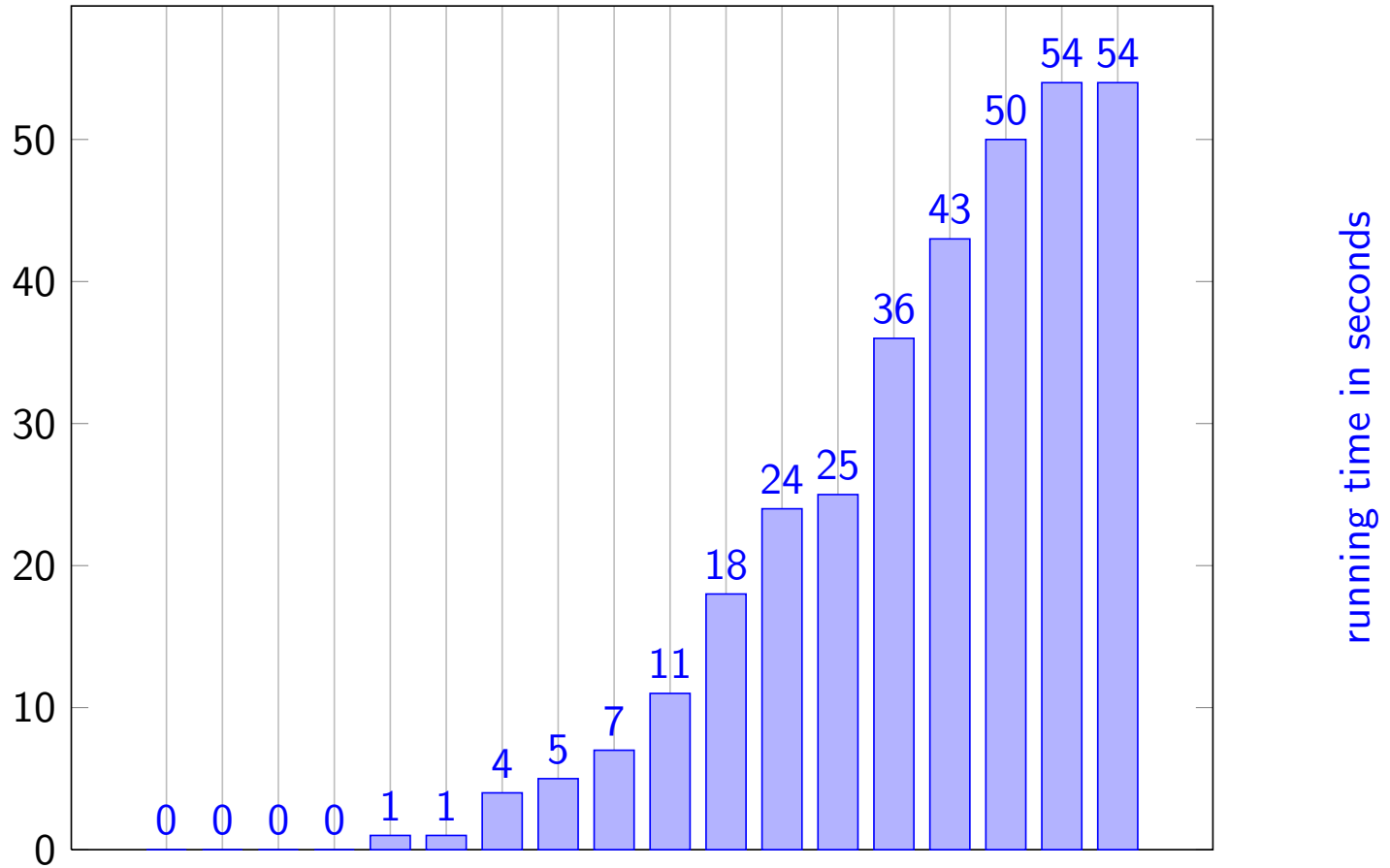
- gna-113: 1h65m
  3.9% away from optimum
- gna-119: 1h49m
  ArrayIndexOutOfBoundsException: 595
- gna-101: 6h23m
  correct
- gna-109: 7h30m
  correct
- gna-114: 9h1m
  correct
- gna-108: 10h18m
  0.25% away from optimum

Programming Exercise 1 — Simplex

Trivial LP:

$$
\begin{aligned}
\min \quad & 0 \cdot x_1 + 0 \cdot x_2 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& 0 \cdot x_1 + 0 \cdot x_2 \geq 0 \\
& \vdots
\end{aligned}
$$

Programming Exercise 1 — Simplex



Solving trivial LP with redundant constrains

- 5 programs not done in 10 minutes
- 2 programs produced an exception

Programming Exercise 2 — Max Flow

Task:

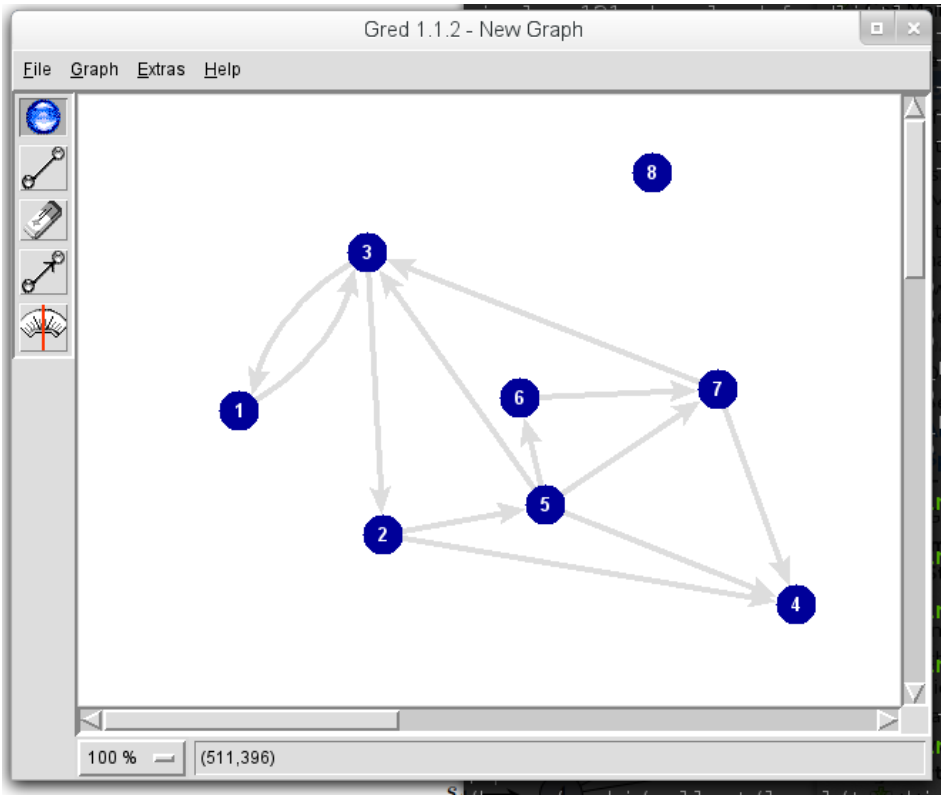
- read the graph from a given `.cat` file
- build the corresponding data structure
- set source s ans sink t
- run the Goldberg-Tarjan / Edmonds-Karp algorithm
- give information about an optimal flow (commandline, file,...)
- (optional:) calc. the max. flow with your simplex algorithm

Requirements:

- similar to PA1
- no need for an output file
- find the correct value for the given examples, at least for the smaller ones
- no additional packages (eg. flows/graph/algorithms) allowed

Programming Exercise 2 — Max Flow

Fileformat — Gato Tool



- add arcs/edges
- changes weights, 1 to 3 edge weights possible
- random grap generation possible
- saves graph as `.cat` file
- free software for windows/mac/linux
- **Tip:** build simple graphs to test your code

http://gato.sourceforge.net/

Programming Exercise 2 — Max Flow

Fileformat — `.cat` Files

```
 1 graph:
 2 dir:1; simp:1; eucl:0; int:1; ew:3; vw:0;
 3 scroller:
 4 vdim:1000; hdim:1000; vlinc:10; hlinc:10; vpinc:50; hpinc:50;
 5 vertices:4;
 6 n:1; x:150; y:150;
 7 n:2; x:200; y:150;
 8 n:3; x:300; y:300;
 9 n:4; x:100; y:350;
10 edges:1;
11 h:2; t:1; e:2; w:50; w:0; w:23000;
```

- **line2:** dir: 1 $\rightsquigarrow$ directed graph, simp: 1 $\rightsquigarrow$ simple graph, eucl: 1 $\rightsquigarrow$ euclidean graph, int: 1 $\rightsquigarrow$ weights are integral, ew: number of edge weights, vw: number of vertex weights
- **line4:** information about how to draw the graph
- **line5:** vertices: number of vertices
- **line6+:** n: index of vertex, x/y: coordinates(for drawing)
- edges: number of edges
- **edge:** h/t: head/tail of an edge, w: weights, ignore "`e:2;`"

Programming Exercise 2 — Max Flow

What you need to do:
implement a graph structure, needed operations

- finding all neighbors of a vertex(δ^+/δ^-)
- select that antiparallel arc of a given arc
- store capacities/ flow values/ residual capacities

residual graph not necessarily needed

graph can be stored as adjacency matrix or as adjacency lists

Algorithms:

- breath first search(shortest s-t path)
- Edmonds-Karp algorithm
- Goldberg-Tarjan algorithm

Min Cost Flow — Optimality

Min Cost Flow Problem:

Given: $D = (V, A), u : A \to \mathbb{R}_{\geq 0}, c : A \to \mathbb{R}_{\geq 0}, b : V \to \mathbb{R}$

Task: find min cost feasible b-transshipment
(flow x, such that $\text{ex}_v = b(v)$)

Optimality Criterion:
b-transshipment x is **optimal** $\Leftrightarrow$ **no dicircuit** in D_x has **neg. cost**

Proof: . . .

Min Cost Flow — Min Mean Cycle Canceling

Minimum Mean- Cycle Canceling Algorithm

- find feasible b-transshipment x or determine that none exists
- while D_x contains a dicircuit with negative cost
  - find a minimum mean-cost dicircuit C in D_x
  - send flow along C, such that at least on edge in C vanishes D_x

Finding Negative Cycles:

- any: $\rightsquigarrow$ easy, done with Bellman-Ford Algorithm
  cycle cancling has pseudopolyn. running time
- min. cost: $\rightsquigarrow$ extremely difficult(NP-complete), most likely not possible in polynomial time
- minimum mean-cost: $\rightsquigarrow$ doable in polynomial time with modification of Bellman-Ford

Min Cost Flow — Min Mean Cycle

Assumption: digraph D, $s \in V(D)$, every vertex v is reachable from s

$c_{\text{mean}}(C) := \frac{\sum_{e \in C} c(e)}{|C|}$, mean cost of dicircuit C

$\mu(D) = \min\{c_{\text{mean}}(C) | C \text{ is a dicircuit in } D\}$,
value of min mean cycle

$F_k(v) :=$ min. length of a path from s to v with exactly k arcs,
∞ if no such path exists

Claim: $\mu(D) = \min_{v \in V(D)} \max_{0 \le k \le n-1, F_k \neq \infty} \frac{F_n(v) - F_k(v)}{n-k}$

Proof: ...

Min Cost Flow — Min Mean Cycle

Min Cost Flow — Min Mean Cycle

Min Cost Flow — Min Mean Cycle Algorithm

Algorithm 1 Minimum Mean Cycle

add vertex s to D, add arcs (s, v) with $c(s, v) = 0$ to D
$F_0(s) = 0, F_0(v) = \infty$ {initialize values}
for $k = 1$ to n **do**
  for all $v \in V(G)$ **do**
    $F_k(v) = \infty$
    for all $w \in \delta^-(v)$ **do**
      if $F_{k-1}(w) + c((w, v)) < F_k(v)$ **then**
        $F_k(v) = F_{k-1}(w) + c((v, w))$
        $\text{pred}_k(v) = u$
      end if
    end for
  end for
end for

Running time: $\mathcal{O}(n \cdot m)$

Min Cost Flow — Min Mean Cycle Algorithm

Finding cycle with distance values:

- $F_n(v) = \infty$ for all $v \in V \Rightarrow D$ is acyclic
- find vertex v, which minimizes $\max_{0 \le k \le n-1, F_n(v) \neq \infty} \frac{F_n(v) - F_k(v)}{n-k}$
- determine simple cycle in $\text{pred}_n(v), \text{pred}_{n-1}(\text{pred}_n(v)), \ldots$

Running time: $\mathcal{O}(n \cdot n)$, in total $\mathcal{O}(n \cdot m)$

Min Cost Flow — Min Mean Cycle

Correctness:

- adding vertex s and edges (v, s) does not create new cycles, every vertex reachable from s
- $F_k(v)$ calculated correctly via recursion

$$F_k(v) = \min\{F_{k-1}(w) + c(w, v) | w \in \delta^-(v)\}$$

- detection of acyclic graphs correct: no circuit $\rightsquigarrow F_n(v) = \infty$, because no path has length of n
- algorithm finds min. mean cycle with values of $F_k(v)$:

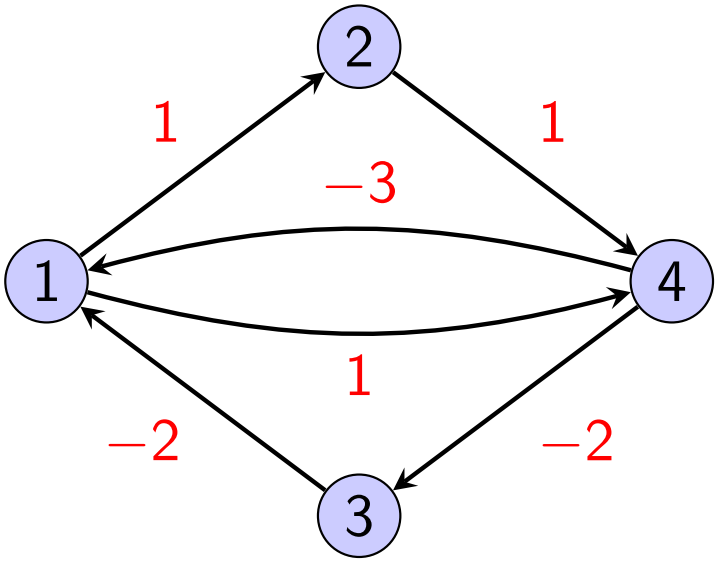
Min Cost Flow — Min Mean Cycle

Correctness contd.: algorithm finds min. mean cycle with values of $F_k(v)$:

- consider (D, c') with $c'(e) = c(e) - \mu(D)$
- algorithms runs analog on this instance, $F'_k(v) = F_k(v) - k \cdot \mu(D)$
- v chosen such that $F'_k(v) = \min\{F'_k(v) | 0 \leq k \leq n-1\}$
- $\mu(D) = 0$ wrt. c' (no negative cycle in D')
- $s - v$ path with n arcs and length $F'_n(v)$:
  shortest $s - v$ path + negative cycle(s) of length 0
- those cycles have min mean cost $\mu(D)$ in (D, c), because cycles in (D, c) and (D, c') differ wrt. mean cost by $\mu(D)$
- min mean cycle is found

Min Cost Flow — Min Mean Cycle

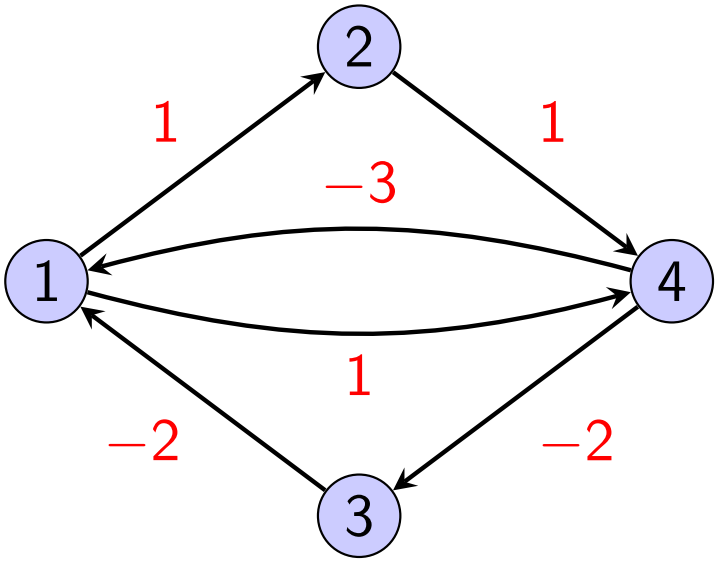
Example:



k	0	1	2	3	4
$F_k(1)$	∞	∞	∞	∞	∞
$F_k(2)$	∞	∞	∞	∞	∞
$F_k(3)$	∞	∞	∞	∞	∞
$F_k(4)$	∞	∞	∞	∞	∞

Min Cost Flow — Min Mean Cycle

Example:



⟶ : pred

k	0	1	2	3	4
$F_k(1)$	0	∞	-2	-3	-4
$F_k(2)$	∞	1	∞	-1	-2
$F_k(3)$	∞	∞	-1	0	-3
$F_k(4)$	∞	1	2	-1	-2

Min Cost Flow — Min Mean Cycle

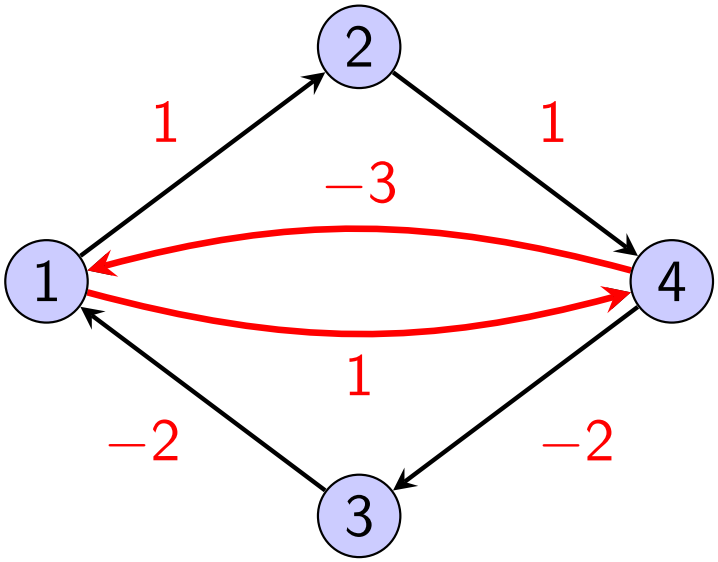
Example:

k	0	1	2	3
$\frac{F_n(1)-F_k(1)}{n-k}$	-1		-1	-1
$\frac{F_n(2)-F_k(2)}{n-k}$		-1		-1
$\frac{F_n(3)-F_k(3)}{n-k}$			-1	-3
$\frac{F_n(4)-F_k(4)}{n-k}$		-1	-2	-1

k	0	1	2	3	4
$F_k(1)$	0	∞	-2	-3	-4
$F_k(2)$	∞	1	∞	-1	-2
$F_k(3)$	∞	∞	-1	0	-3
$F_k(4)$	∞	1	2	-1	-2

$\rightsquigarrow$ select vertex 3 for minimal maximum

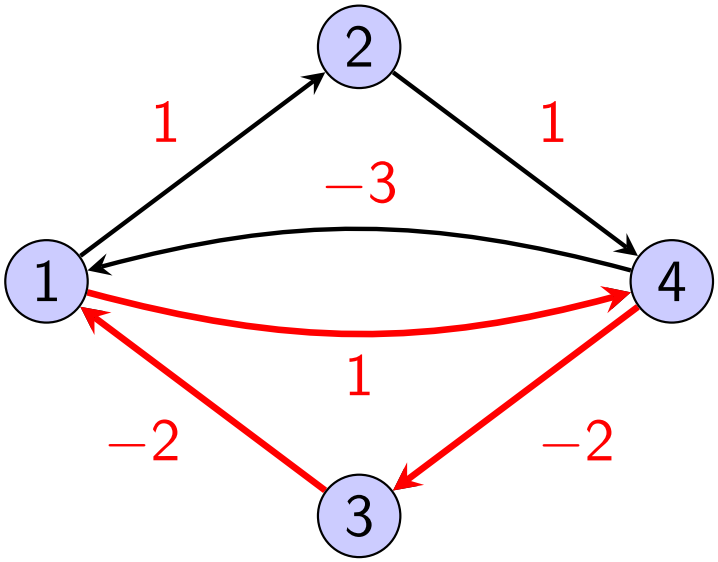
Min Cost Flow — Min Mean Cycle



min mean cycle for $v = 3$

k	0	1	2	3	4
$F_k(1)$	0	∞	−2	−3	−4
$F_k(2)$	∞	1	∞	−1	−2
$F_k(3)$	∞	∞	−1	0	−3
$F_k(4)$	∞	1	2	−1	−2

Min Cost Flow — Min Mean Cycle



min mean cycle for $v = 2$

k	0	1	2	3	4
$F_k(1)$	0	∞	-2	-3	-4
$F_k(2)$	∞	1	∞	-1	-2
$F_k(3)$	∞	∞	-1	0	-3
$F_k(4)$	∞	1	2	-1	-2