Acyclic Digraphs and Topological Orderings

Definition 5.19.

Consider a digraph $D = (V, A)$.

a. An ordering $v_1, v_2, \ldots, v_n$ of V so that $i < j$ for each $(v_i, v_j) \in A$ is called a topological ordering.

b. If D has a topological ordering, then D is called acyclic.

Observations:

- Digraph D is acyclic if and only if it does not contain a dicircuit.
- Let D be acyclic and $\mathcal{S}$ an ordering of A such that (v_i, v_j) precedes (v_k, v_ℓ) if $i < k$. Then every dipath of D is embedded in $\mathcal{S}$.

Theorem 5.20.

The shortest path problem on acyclic digraphs can be solved in time $O(m)$.

Proof: ... □

Still open: how to find a topological order?

Claim: Can be done in $O(m)$ time

1) compute indegree $b_v := |\delta^-(v)|$ for all $v \in V$ $O(m)$

put all nodes v with $b(v) = 0$ into a queue Q

2) while $Q \neq \emptyset$ do

extract $v \in V$

delete it from digraph D together with all arcs (v,w) } $O(\text{outdegree}(v))$

update b_w for all these w

if $b_w = 0$, enter it in Q

The order of deletion is a topological order

If all all vertices are deleted

Otherwise the remaining vertices contain a dicycle

In total $O(m)$

Dijkstra's Algorithm

Consider the special case of nonnegative costs, i. e., $c_a \geq 0$, for each $a \in A$.

Dijkstra's Algorithm

i initialize y, p (see Ford's Algorithm); set $S := V$;

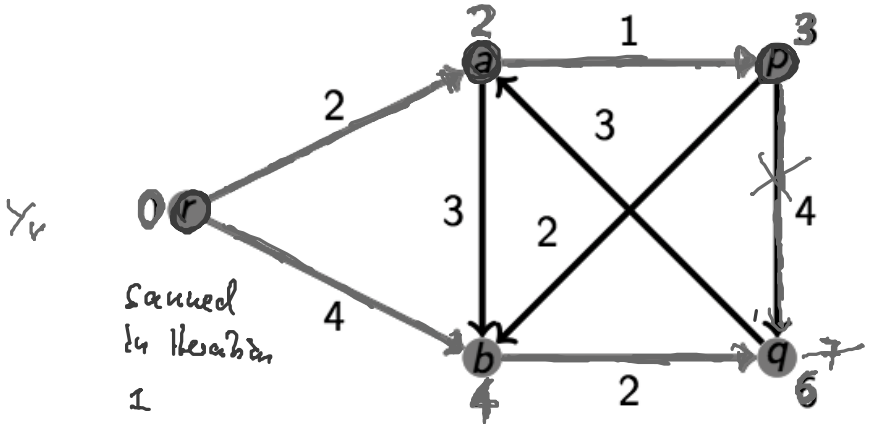
ii while $S \neq \emptyset$ do

iii choose $v \in S$ with y_v minimum and delete v from S;

iv for each $w \in V$ with $(v,w) \in A$ do ← scanning

v if $y_w > y_v + c_{(v,w)}$, then set $y_w := y_v + c_{(v,w)}$ and $p(w) := v$; ← labelling

Example:



y_v

Scanned in Iteration 1

Correctness of Dijkstra's Algorithm

Lemma 5.21.

For each $w \in V$, let y'_w be the value of y_w when w is removed from S. If u is deleted from S before v, then $y'_u \leq y'_v$.

Proof: ... key lemma □

Theorem 5.22.

If $c \geq 0$, then Dijkstra's Algorithm solves the shortest paths problem correctly in time $O(n^2)$. A heap-based implementation yields running time $O(m \log n)$.

Proof: ... □

Remark: The for-loop in Dijkstra's Algorithm (step iv) can be modified such that only arcs (v, w) with $w \in S$ are considered.

↳ Scanning !

→ Proof

By contradiction, assume that $y'_v < y'_u$ but that v is deleted after u

among all possible such v, choose the earliest one after u

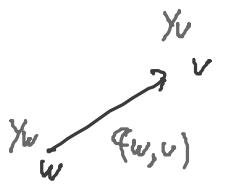
When u was deleted, we had $\underbrace{y_u}_{=: y'_u \text{ by definition of } y'} \leq y_v$

so y_v was decreased later, maybe several times

Consider the last time when it was decreased

$\Rightarrow$ v was the endpoint of an arc (w,v)

$y_w + c_{(w,v)} < y_v$ upon checking (w,v)



$\Rightarrow$ $y_w + c_{(w,v)} = y_v = y_v'$

$\uparrow$ because y_v is not changed afterwards

$\uparrow$ ≥ 0

$= y_w'$

$\Rightarrow$ $y_w' \leq y_v'$ and w is deleted before v

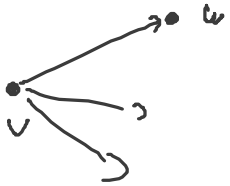
$\Rightarrow$ contradiction to the fact that v is the earliest to be deleted after u $\square$

Proof of Thm 5.22

show that, at termination
- y' is a feasible potential
- p gives a r–v-dipath with $y_v' = c(P)$

upon deletion of v, all (v,w) arcs are checked

and $y_w \leq y_v + c_{(v,w)}$ afterwards



Lemma 5.21 y_v' is never decreased again

$\Rightarrow$ triangle ineq. is valid at termination

$\Rightarrow$ y' is a feasible potential

Thm. 5.12, no negative cycles $\Rightarrow$ p defines min cost r–v-dipaths with $c(P) = y_v'$ $\square$

Feasible Potentials and Nonnegative Costs

Observation 5.23.

For given arc costs $c \in \mathbb{R}^A$ and node potential $y \in \mathbb{R}^V$, define arc costs $c' \in \mathbb{R}^A$ by $c'_{(v,w)} := c_{(v,w)} + y_v - y_w$. Then, for all $v, w \in V$, a least-cost v-w-dipath w.r.t. c is a least-cost v-w-dipath w.r.t. c', and vice versa.

≙ subtract potential diff. $y_w - y_v$ of arc (v,w)

Proof: Notice that for any v-w-dipath P it holds that

$$c'(P) = c(P) + y_v - y_w \ . \qquad \square$$

Corollary 5.24.

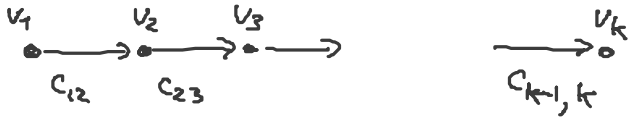
For given arc costs $c \in \mathbb{R}^A$ (not necessarily nonnegative) and a given feasible potential $y \in \mathbb{R}^V$, one can use Dijkstra's Algorithm to solve the shortest paths problem. $\square$

Definition 5.25.

For a digraph $D = (V, A)$, arc costs $c \in \mathbb{R}^A$ are called conservative if there is no negative-cost dicircuit in D, i. e., if there is feasible potential $y \in \mathbb{R}^V$.

Observation 5.23

Path P original cost



Path P with modified cost $\quad c_{12} + y_1 - \cancel{y_2} + c_{23} + \cancel{y_2} - y_3 + \cdots$

Efficient Algorithms

What is an efficient algorithm?

- efficient: consider running time
- algorithm: Turing Machine or other formal model of computation

Java virtual machine + arbitrary long numbers in one memory cell

Simplified Definition

An algorithm consists of

- "elementary steps" like, e. g., variable assignments
- simple arithmetic operations

which only take a constant amount of time. The running time of the algorithm on a given input is the number of such steps and operations.

Bit Model and Arithmetic Model

Two ways of measuring the running time and the size of the input I of A:

Bit Model

Count bit operations; e. g., adding two n-bit numbers takes $n(+1)$ steps; multiplying them takes $O(n^2)$ steps.

Size of input I is the total number of bits needed to encode "structure" and numbers.

Arithmetic Model

Simple arithmetic operations on arbitrary numbers can be performed in constant time.

Size of input I is total number of bits needed to encode "structure" plus # numbers in the input.

assume an encoding of numbers w.r.t. base $b \geq 2$

a number N has $\lfloor \log_b N \rfloor + 1$ digits

Denote input size of I by $\langle I \rangle$

Polynomial vs. Strongly Polynomial Running Time

Definition 5.26.

i. An algorithm runs in polynomial time if, in the bit model, its (worst-case) running time is polynomially bounded in the input size.

ii. An algorithm runs in strongly polynomial time if, in the bit model as well as in the arithmetic model, its (worst-case) running time is polynomially bounded in the input size.

Examples:

- Prim's and Kruskal's Algorithm as well as the Ford-Bellman Algorithm and Dijkstra's Algorithm run in strongly polynomial time.
- The Euclidean Algorithm runs in polynomial time but not in strongly polynomial time. — for $\gcd(a,b)$

Def: $\exists$ polynomial p s.t. $\forall I$ the running time of algorithm A has a run time

$$T_A(I) \leq p(\langle I \rangle)$$

$T_A(I)$: run time of A for input I

$\langle I \rangle$: encoding length of I

Pseudopolynomial Running Time

- In the bit model, we assume that numbers are binary encoded, i. e., the encoding of the number $n \in \mathbb{N}$ needs $\lfloor \log n \rfloor + 1$ bits.
- Thus, the running time bound $O(C\, n^2)$ of Ford's Algorithm where $C := 2 \max_{a \in A} |c_a| + 1$ is not polynomial in the input size.
- If we assume, however, that numbers are unary encoded, then $C\, n^2$ is polynomially bounded in the input size.

Definition 5.27.

An algorithm runs in pseudopolynomial time if, in the bit model with unary encoding of numbers, its (worst-case) running time is polynomially bounded in the input size.

there is no polynomial p s.t. $C \leq \lfloor \log C \rfloor + 1$

size of number C — encoding length in binary encoding

(size is exponential in encoding length)

"Bierdeckel" encoding

coaster encoding

Chapter 6:
Maximum Flow Problems

(cp. Cook, Cunningham, Pulleyblank & Schrijver, Chapter 3)

Maximum s-t-Flow Problem

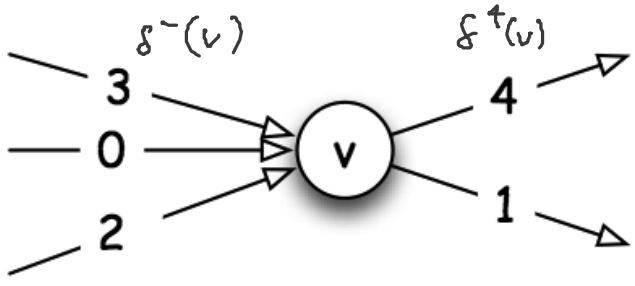
Given: Digraph $D = (V, A)$, arc capacities $u \in \mathbb{R}^A_{\geq 0}$, nodes $s, t \in V$.

Definition 6.1.

A flow in D is a vector $x \in \mathbb{R}^A_{\geq 0}$. Moreover, a flow x in D

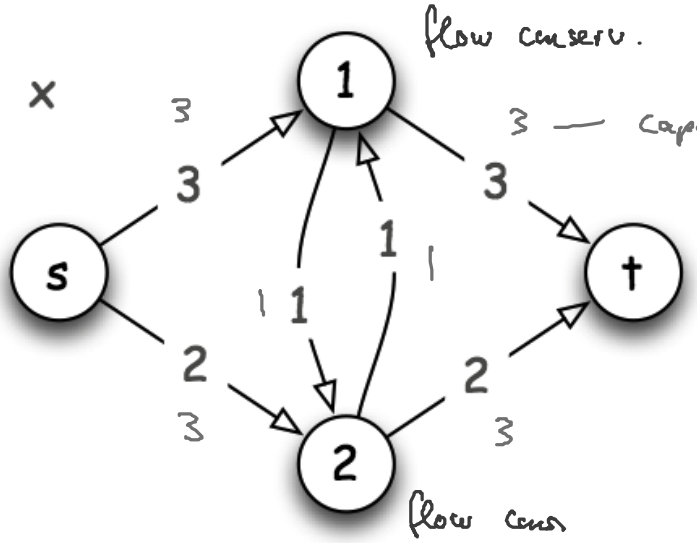
i. obeys arc capacities and is called feasible, if $x_a \leq u_a$ for each $a \in A$;
ii. has excess $\mathrm{ex}_x(v) := x(\delta^-(v)) - x(\delta^+(v))$ at node $v \in V$;
iii. satisfies flow conservation at node $v \in V$ if $\mathrm{ex}_x(v) = 0$;
iv. is a circulation if it satisfies flow conservation at each node $v \in V$;
v. is an s-t-flow of value $\mathrm{ex}_x(t)$ if it satisfies flow conservation at each node $v \in V \setminus \{s, t\}$ and if $\mathrm{ex}_x(t) \geq 0$.

The maximum s-t-flow problem asks for a feasible s-t-flow in D of maximum value.



flow conservation

inflow into v = 5 = outflow out of v


flow conserv.
x
3
3
3
3 — capacity u_a
1
1
1
1
s
t
2
2
3
3
2
flow cons
$ex_t(x) = 5$