Minimum Cost Flow

Bibliografia

- Network Flows, Ahuja , Magnanti e Orlin

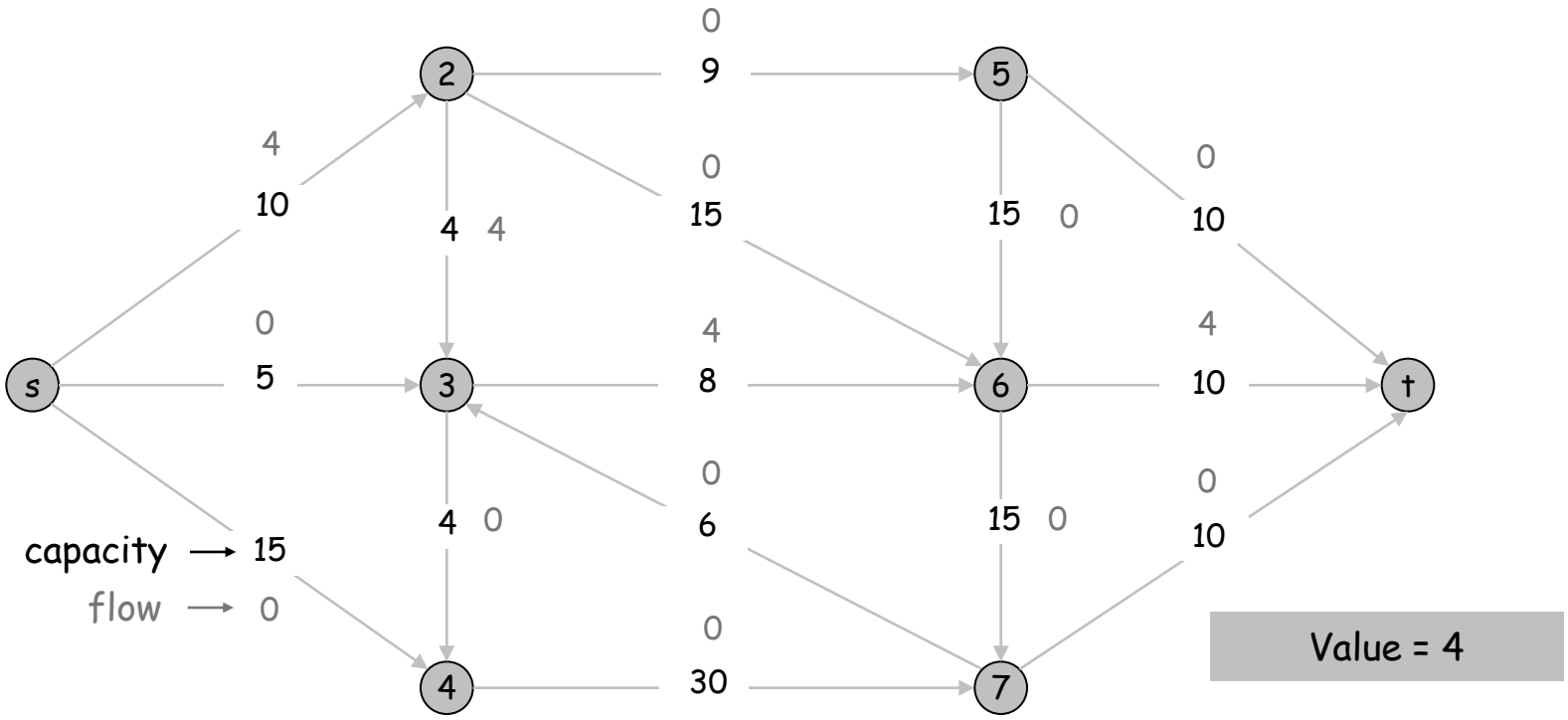
- Capítulos 1 e 2
- Seção 3.5
- Seções 9.1, 9.2 and 9.3, 9.6 and 9.7
- Seções 10.1,10.2

Flows

Def. An s-t flow is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ (capacity)
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ (conservation)

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



Minimum Cost Flows

Goal: Build a cheap network to satisfy the flow requirement.

- A network G
- A demand/supply b on each vertex, i.e. $b: N \rightarrow R$
  - $b(v)>0$ *supply node*
  - $b(v)<0$ *demand node*
  - $b(v)=0$ *transhipment node*
  - ***Inverted with respect to max flow lecture***
- A capacity function c on the edges, i.e. $c: E \rightarrow R$
- A cost function w on the edges, i.e. $w: E \rightarrow R$

Output: a feasible flow f which **minimizes** $\Sigma f(e)\, w(e)$

Minimum Cost Flow Problem

$$Min \sum_{e} f(e) w(e)$$

$$\sum_{e\ out\,\mathrm{i}} f(e) - \sum_{e\ \mathrm{in}\ i} f(e) = b(i) \ \forall \ \mathrm{node\,i}$$

$$0 \le f(e) \le c(e)$$

Minimum Cost Flows

Assumptions

- All data are integral
  - Assumption is not restrictive in practice

- There is a balance between supplies and demands
  - Necessary for feasibility

Special cases

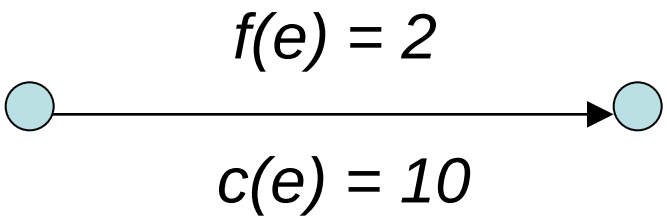
Shortest path: find a shortest path between *s* and *t*

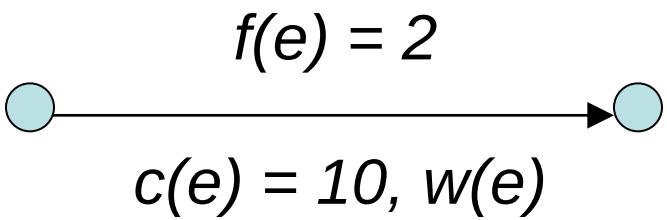
- A minimum cost flow b(s)=1 an b(t)=-1

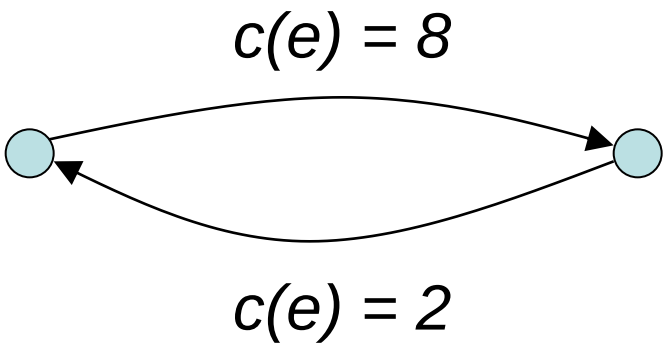
Disjoint paths: connect *s* and *t* by *k* paths with min # of edges

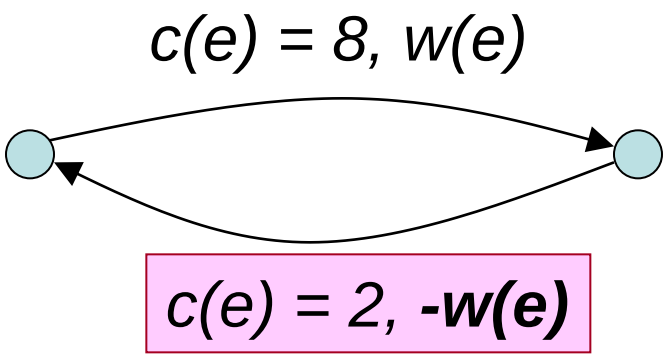
- Every edge in the original graph has cost *1* and capacity *1*.
- b(s)=k and b(t)=-k

Residual Graph









Max Flow

Min-cost Flow

What is the augmenting structure?

Maximum flows

- Directed paths in residual graphs
- **Idea**: **Keep flow conservations** and **increase the flow from s**.

What is the augmenting structure?

Minimum cost flows

- Strategy: start with a feasible flow and improve the cost?
- Try: **Imagine a cheaper flow**.

What would happen in the residual graphs?

Flows: Working with the residual network (p44)

Residual Network Property: *If x^0 and x are flows in the network G then*

$$x = x^0 + x'$$

where x' is a flow in the residual network $G(x^0)$.

In addition, for every arc (i,j) in G,

$$w(i,j)x(i,j) = w(i,j)\,x^0(i,j) + w'(i,j)x'(i,j) + w'(j,i)x'(j,i),$$

where w' is the cost in $G(x^0)$.

Em palavras ... Dados dois fluxo x e x^0, x pode ser decomposto em x^0 mais um fluxo x' na rede residual de x^0

Flows: Working with the residual network

Proof. Let (i,j) be an arc in G.

(i) If $x(i,j) >= x^0(i,j)$ define $x'(i,j)= x(i,j)- x^0(i,j)$

(ii) If $x(i,j) < x^0(i,j)$ define $x'(j,i)= x^0(i,j) - x(i,j)$

a) x' respects the capacity constraints in $G(x^0)$

- *If $x(i,j) > x^0(i,j)$*

➔ *$x'(i,j)= x(i,j) - x^0(i,j) <= c(i,j) - x^0(i,j)$ =capacity of (i,j) in $G(x^0)$*

- *If $x(i,j) < x^0(i,j)$*

➔ *$x'(j,i)=x^0(i,j)-x(i,j)<=x^0(i,j)$=Cap de (j,i) in $G(x^0)$*

Flows: Working with the residual network

(b) x' respects the mass balance constraint

Let j be a node of G.

In(x) = sum of flows (x) that enter j

In(x^0) =sum of flows (x^0) that enter j

Out(x)= *sum of flows (x) that leave j*

Out(x^0) =sum of flows (x^0) that leave j

Flows: Working with the residual network

(b) Como x e x^0 são viáveis

Inflow(x) – Outflow(x) = Inflow(x^0) – Outflow(x^0) = b(j) ➔

Inflow(x) – Inflow(x^0) = Outflow(x)– Outflow(x^0)

Flows: Working with the residual network

(b) Defina

D: contribuição para o fluxo x' das arestas que entram em j na rede $G(x^0)$ proveniente das arestas que entram no nó j na rede G

$$D = \sum_{\substack{(i,j)\in E \\ x(i,j)\geq x^0(i,j)}} (x(i,j) - x^0(i,j))$$

D': contribuição para o fluxo x' das arestas que deixam j na rede $G(x^0)$ proveniente das arestas que entram no nó j na rede G

$$D' = \sum_{\substack{(i,j)\in E \\ x(i,j)< x^0(i,j)}} (x^0(i,j) - x(i,j))$$

Temos:

$$D - D' = Inflow(x) - Inflow(x^0) \quad (i)$$

Flows: Working with the residual network

(b) De forma análoga, defina

D'': contribuição para o fluxo x' das arestas que saem de j na rede $G(x^0)$ proveniente das arestas que deixam o nó j na rede *G*

D''': contribuição para o fluxo x' das arestas que entram em j na rede $G(x^0)$ proveniente das arestas que deixam o nó j na rede *G*

Temos:

$$D''-D''' = Outflow(x) - Outflow(x^0) \quad (II)$$

Diminuindo (i) de (ii) temos que D+D'''=D'+D''

➔ ***Conservação de Fluxo***

Flows: Working with the residual network

(c) custos

Seja um arco (i,j) em G.

Se $x(i,j) >= x^0(i,j)$

$$w(i,j)x(i,j) = w(i,j)x^0(i,j) + w(i,j)(x(i,j) - x^0(i,j)) =$$

$$w(i,j)x^0(i,j) + w'x(i,j)'(i,j)$$

Se $x(i,j) < x^0(i,j)$

$$w\,x(i,j) = w(i,j)\,x^0(i,j) - w(i,j)(x^0(i,j) - x(i,j)) =$$

$$w(i,j)\,x^0(i,j) + w'(j,i)x'(j,i)$$

Flow Decomposition

Figura 3.9 , p 79, Ahuja

Flow Decomposition

- *e(i) = inflow in i - outflow i*
- *e(i) > 0 excess node*
- *e(i) < 0 deficit node*
- *e(i)=0 balanced node*

EXAMPLE

Flow Decomposition

Flow Decomposition Theorem (p80 Ahuja)

Every non-negative arc flow can be represented as a path and cycle flow (not necessarily uniquely) with the following two properties

(i) Every directed path with positive flow connects a deficit node to an excess node

(ii) At most (n+m) paths and cycles have nonzero flow: out of these at most m cycles have nonzero flow.

Flow Decomposition

Corollary

If there are no deficit or excess nodes in the network then a non-negative arc flow can be represented as at most m cycles with nonzero flow.

Procedure to decompose a flow into paths and cycles

C = empty set

While there is a deficit node *i*

Follow arcs with positive flows starting from *i* until either an excess node *j* is found or a node is visited twice

If an excess node *j* is found then

P ← path from *i to j*

flow(P) ← min{ -e(i),e(j), flow of some edge in P}

C ← C ∪ P

Reduce the flow along P

If a node *k* is visited twice

W ← Cycle from k to k

flow(W) ← min{flow of some edge in W}

C ← C ∪ W

Reduce the flow along W

End While

Procedure to decompose a flow into paths and cycles (cont'd)

While there is an arc *e* with positive flow

Follow backward arcs with positive flows starting from *e* until a node is visited twice

Let W be the obtained cycle

flow(W) ← min{flow of some edge in W}
Reduce the flow along W
C ← C ∪ W

End While

End Procedure

Procedure to decompose a flow into paths and cycles (cont'd)

Correctness

- A deficit node exists if and only if na excess node exists ➔ after the execution of the first while block there are no more imbalanced nodes.
- At each iteration either one node becomes balanced or the flow of some arc goes to 0 ➔ There are at most n+m iterations and, as a consequence, n+m paths or cycles

- Whenever a cycle is obtained the flow of some arc goes to 0 ➔ There are at most m cycles

Flow Decomposition

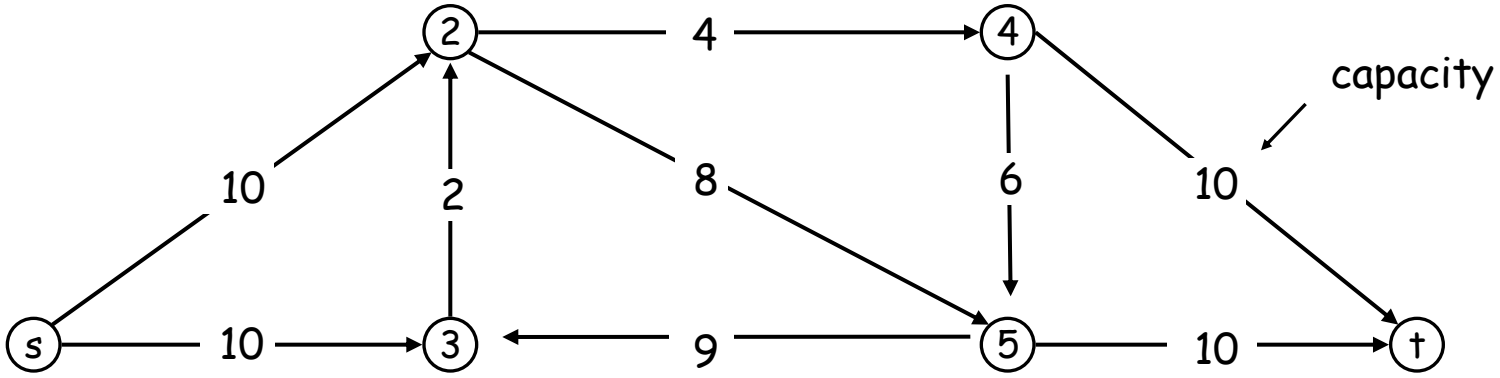


Figura 3.10, p 82

Optimality conditions

Augmenting Cycle Theorem (Theorem 3.7)

Let x and x^0 be two feasible flows. Then x equals x^0 plus the flow on at most m directed cycles in $G(x^0)$. Furthermore, the cost of x equals the cost of x^0 plus the cost of the flows on these augmenting cycles

Proof.

- *We have that $x = x^0 + x'$, where x' is a flow in the residual network $G(x^0)$.*
- *Since $e(j) = 0$ for every node j in the residual network $G(x^0)$ it follows from the flow decomposition theorem that x' can be decomposed in at most m directed flow cycles in $G(x^0)$*

Optimality conditions

Theorem (Negative Cycle Optimality, 3.8)

A feasible solution x of the minimum cost flow problem is an optimal solution iff the residual network G(x*) contains no negative cost directed cycle*

Proof

- *If G(x*)* has a negative cost directed cycle then by sending flow along the cycle we can improve the cost of the solution while keeping feasibility. Thus, if x* is optimal then G(x*) *contains no negative cost directed cycle*
- Let x be a a fesible solution. Then x = x* + the flows of at most m directed cycles in G(x*). If G(x*) does not contain a negative cost directed cycle then the cost of x is at least the cost of solution x* → x* is optimal

Cycle Canceling Algorithm (Section 9.6)

Start with a feasible flow f

***While** there is a negative cost directed cycle C in the residual graph*

e← bottleneck capacity of C

augment flow f by e units along C

Update the residual graph

End While

Return f

Cycle Canceling Algorithm

Figura 9.8 (a), p 318

Cycle Canceling Algorithm

Theorem. *If all arc capacities and suplies/demands nodes are integer, the minimum cost flow problem admits an integer solution*

Running Time

- If W is the maximum cost in the network and U is the maximum capacity then mWU is an upper bound on any feasible flow and -mWU (allowing negative costs) is a lower bound. Since the cost is reduced by at least one unit at each iteration we have at most O(mWU) iterations.

- A negative cost cycle can be found in O(mn) (Bellman-Ford) → The algorithm runs in $O(nm^2WU)$

Cycle Canceling Algorithm

Algorithm does not specify which cycle shall be used at each iteration

- *Different choices lead to different theoretical worst case behaviour*

Cycle Canceling Algorithm

- cycle of maximum improvement

  - $O(m \log(mWU))$ iterations, but it is hard to find such a cicle (NP-complete).

  - A variation of this approach provides a polytime algorithm

- minimum mean cost cycle

  - the mean cost of a cycle is the sum of the cost of its edges over the number of edges

  - $O(n^2 m^3 \log n)$ time algorithm

Extra Assumptions

Assumption 9.4: the network G contains an uncapcitated directed path between every pair of nodes.

- If necessary, we can transform the network G into a network G' by fixing a node **1** adding artificial arcs (**1**,j) and (j,**1**) for all nodes j, and assigning large cost and infinite capacity to these arcs.
- If G has a feasible solution, then the optimal solution of G' will not send flow along the artificial edges so that the optimal solution of G' is also an optimal solution for G.

Main implication: for every flow x, there is a directed path between (i,j) in $G(x)$.

Optimality conditions: reduced costs

Reduced costs. Let ***p*** be a vector of prices (potentials) associated with the nodes of G. For every (i,j) in G, define $w^p(i, j) = w(i,j) - p(i) + p(j)$.

Theorem *(Reduced Cost Optimality)* A feasible solution x^* is an optimal solution of the minimum cost flow problem if and only if there are a set of prices ***p*** *that* satisfies the following reduced cost optimality conditions

$$w^p(i, j) >= 0 \text{ for every arc } (i,j) \text{ in } G(x^*) \qquad (*)$$

Proof

- Via the Negative Cycle Optimality condition

- $\leftarrow$ If x^* satisfies (*) then let W be a directed cycle in $G(x^*)$

We have that

$$w^p(W) >= 0 \rightarrow w(W) >= 0,$$

So that there are no negative cycles in $G(x^*)$ which implies that x^* is optimal

Optimality conditions: reduced costs

➔ If x^* is an optimal solution then $G(x^*)$ contains no negative cycles.

Fix a node **1** in $G(x^*)$ and let d(j) be the distance (shortest path) from **1** to node j in $G(x^*)$.

The previous assumption (9.4) and the fact that there are no negative cycles guarantee that these distances are well defined.

For every arc (i,j) we have

$$d(j) <= d(i) + w(i,j) \quad \rightarrow$$

$$w(i,j)-d(j) +d(i) >=0$$

Define p(i)=-d(i) for every node → $w^p(i,j) = w(i,j) + d(i) - d(j) >= 0$

Extra Assumptions

Assumption 9.5: All the arcs in the network G have non-negative costs

- This assumption can be dropped with a proper network transformations

Successive Shortest Path Algorithm (Section 9.7)

Pseudoflow: *satisfies capacity and nonnegativity constraints*

e(i) = b(i) + inflow(i) - outflow(i)

e(i) > 0 excess node

e(i) < 0 deficit node

e(i)=0 balanced node

Strategy: *Maintains the optimality of the solution according to the reduced cost condition at every step and change the solution toward feasibility.*

Successive Shortest Path Algorithm

x,**p** ←0; e(i) = b(i) for every node i

E = excess nodes D = deficit nodes

While there is an excess node in *G*

select k in E and l in D

Determine shortest path distances ***d*** from k to all other nodes in G(x) w.r.t reduced cost w^p

Let P be the shortest path from k to l in G(x) w.r.t. the reduced costs (existence follows from Assumption 9.4 and 9.5)

p ←p – d

$\delta \leftarrow$ min{ e(k),-e(l), capacities of edges of P}

Augment δ units of flow along P

Update x,G(x), E, D

End While

Successive Shortest Path Algorithm

FIGURA 9.10, p 322

Successive Shortest Path Algorithm

Correctness.

Lemma 1: Suppose that a pseudoflow (flow) satisfies the reduced cost optimality conditions w.r.t some prices ***p****. Let* ***d*** *be the vector of shortest path distances from some node k in G(x) with arc costs* w^p *. Then, the following properties are valid*

(a) the pseudoflow x also satisifies the reduced cost optimality conditions w.r.t. node potentials $\mathbf{p'=p-d}$

(b) The reduced costs $w^{p'}$ *are 0 for all arcs in a shortest path from k to every other node.*

Successive Shortest Path Algorithm

Proof.

(a) We have to prove that $w^{p'}(i, j) >= 0$ for every arc (i,j) in G(x).

$$w^{p'}(i, j)=w(i,j)-p'(i)+p'(j)=w(i,j)-(p(i)-d(i)) +p(j)-d(j) =$$

$$w^{p}(i, j) + d(i) -d(j)$$

Since d is a vector of shortest paths in G(x) w.r.t. arc costs w^p we have that $w^p(i, j) + d(i) -d(j) >=0$

(b) Consider the shortest path from k to some node l in G(x). For each arc (i,j) in this path, $d(j)=d(i)+w^p(i,j)$. Substituting $w^p(i,j)$ by $w^{p'}(i,j) - d(i) + d(j)$ we conclude that

$$w^{p'}(i,j)=0$$

Successive Shortest Path Algorithm

Lemma 2. *The flow x' obtained from x by sending* δ units of *flow along P in the algorithm satisifies the reduced cost optimality conditions w.r.t to p'*

Proof. *We have to prove that* $w^p(i, j) >= 0$ for every arc (i,j) in G(x').

The only arcs that are modified from G(x) to G(x') are the arcs between consecutive nodes in path P. There are three cases:

(a) arc (i,j) does exist in G(x) and does not exist in G(x'). We are fine.

(b) arc (i,j) does exist in both G(x) and G'(x'). reduced cost optimality remain satisfied

Successive Shortest Path Algorithm

Lemma 2. *The flow x' obtained from x by sending* δ units of *flow along P in the algorithm satisifies the reduced cost optimality conditions w.r.t to p.*

Proof. *We have to prove that* $w^p(i, j) >= 0$ for every arc (i,j) in G(x').

(c) arc (i,j) does not exist in G(x) and is created in G'x').

The flow sent along P might created an edge (i,j) in G(x') that does not exist in G(x).

The item (b) from the previous lemma implies that $w^{p'}(i,j) = 0$ *in G(x). Thus,* $w^{p'}(j,i) = - w^{p'}(i,j) = 0$ *so that the new arc satisfies the reduced cost optimality condition.*

Successive Shortest Path Algorithm

Correctness

flow =0,p=0 respects the reduced cost optimality condition (Assumption 9.5)

At each iteration the current flow and potential vector satisfies the reduced cost optimality condition

Successive Shortest Path Algorithm

Running Time

- *The Total imbalance (sum of excess) decreases by at least one unit → the algorithm finishes in at most nU iterations, where U is the absolute value of the maximum demand/supply*

- *At every iteration the algorithm has to run Dijkstra shortest path algorithm from node k → Time complexity O(nU S(n,m,C)),*

where S(n,m,C) is the time complexity to solve the shortest path problem in a network with *m* arcs and *n* nodes.

Successive Shortest Path Algorithm

Practical Improvement

- It is possible to abort Dijkstra algorithm when the first deficit node v is found and update, for each node i, p(i)=p(i)-d(i) if the shortest path to node i has already been determined and p(i)=p(i)-d(v) , otherwise

Other Algorithms

Primal Dual

- *Work with pseudo-flows and maintain reduced optimality conditions as sucessive shortest path. algorithm*
- *Use max flow problem instead of shortest path problem to modify the flow along the network*

Out of Kilter

- *Work with pseudo-flows that maintain mass balance constraints but do not necessarily respect capacity constraints and reduced cost optimaility*
- *The algorithm iteratively modifies the flow and the potentials to reduce the infeabisility and moves toward to satisfy the reduced optimality conditions.*

Capacity Scaling Algorithm (section 10.2)

Approach

- *At every iteration reduce the total Imbalance by a large amount.*

Capacity Scaling Algorithm

x=0, p=0

Δ = largest power of 2 smaller than or equal to U

While Δ >=1 do

For every arc (i,j) in G(x)

if residual capacity r(i,j) >= Δ and w^p (i,j)<0

send r(i,j) unit of flow along arc (i,j) in G

update x and the imbalances

(p,x)←SuccessivePath(x, Δ,p)

Δ ← Δ /2

End While

Capacity Scaling Algorithm

Sucessive Path(flow x, Parameter Δ, prices p)

E(Δ)= excess nodes with imbalance >= Δ

D(Δ) = deficit nodes with imbalance <= - Δ

While E(Δ)<>∅ and D(Δ)<> ∅

select k in E(Δ) and l in D(Δ)

Determine shortest path distances ***d*** from k to all other nodes in G(x, Δ) w.r.t reduced cost w^p

Let P be the shortest path from k to l in G(x, Δ) w.r.t. The reduced costs

p ←p – d

Augment Δ units of flow along P

Update x,G(x, Δ), E(Δ), D(Δ)

End While

Capacity Scaling Algorithm

Correctness

Lemma. *The reduced cost optimality is respected for every arc in* $G(x, \Delta)$ during the call to SuccessivePath (x, Δ,p).

Proof .

- At the begginning of the Δ-phase the arcs from $G(x, \Delta)$ satisfy the reduced cost optimality because the red code of the algorithm removes every edge that violates the condition
- In addition, it follows from Lemmas 1 and 2 that the reduced cost optimality is also respected along the phase.

CQD

Capacity Scaling Algorithm

Running Time

- *We have log U external loops*
- *At the end of a 2* Δ-phase either all excess node have excess smaller than 2Δ or all deficities nodes have deficits larger than -2 Δ. Thus, the sum of excess can be upper bounded by 2nΔ (we use the fact that the sum of excess equals the sum of deficits)
- In the red code, only arcs with *Δ* <=r(i,j)<= *2Δ send flow*
  - *Note that arcs with* r(i,j)> *2Δ do not send flow because they already satisfy the reduced optimaility conditions due to the 2Δ-phase*
  - Thus, the sum of excess right before the beginning of Δ-phase is at most 2Δn+mΔ. Since each iteration of the Capacity Scaling phase reduces the sum by at least Δ, the number of iteration is at most 2(m+n).
- The running time is O(*log U (m+n) S(m,n))*

Capacity Scaling Algorithm

Running Time

- Since the sum of excess at the end of phase Δ is at most $n\Delta$ and every iteration of a Δ-phase reduces this sum by at least Δ it follows that the Δ-phase performs at most (m+n) iterations.

➔ The algorithm runs in $O(m \log U S(m,n))$

Applications: The Transportation Problem

Input:

- *p* plants, each has supply *s(i)*
- *q* warehouses, each has demand *t(j)*
- cost of shipping from plant *i* to warehouse *j* is *d(i,j)*

Goal: Find a cheapest shipping plan to satisfy all the demands.

Car Production

Input

- Several manufacturer plants
- Several car models
- Retail centers throughout the country, each of them with a certain demand for each car
- p(i): cost of producing a model i in plant p is known
- d(i,j): cost of delivering model i from plant p to retail j is d(i,j)

Output

- Production Plan and shipping pattern that minimizes the overall cost

Car Production

Network

- Four layers of nodes: manufacturer plants; manufacturer plants/models; retail centers/models; retail centers
- Arcs between layer 1 and 2: cost of producing a car model in a plant
- Arcs between layer 2 and 3. cost of delivering a model from a plant to a retail center.
- Arcs between layer 2 and 3. these arc have cost 0 and a lower bound corresponding to the demand
- Solution is the minimum cost flow in the network

Racial Balancing of Schools (two ethnics groups)

Input

- A set of schools and a set of districts
- District i has p(i) students from ethnic group p and q(i) from ethnic group q
- Distance from district i to school j is d(i,j)
- School i can enroll at most u(i) students
- Let l(i) and l'(i) be, respectively, the minimum and maximum number of students from group p that can be enrolled in school i

Output

- An assignment of students to scholl that minimizes the total distance students have to travel in order to reach the school.

Optimal Loading of Hoping Airplane (p302)

Input

- An airplane with capacity to carry at most ***p*** passengers on a hoping flight that visits cities 1,2,...,n in a fixed sequence.
- b(i,j) : number of passengers available at city i who want to go to city j
- f(i,j) – fare per passenger from i to j

Output

- The number of passangers that should travel each leg so that the revenue of the airline is maximized

Scheduling with Deferral Costs (p 303)

Input

P - #jobs

q - #processors

a - processing time of all jobs

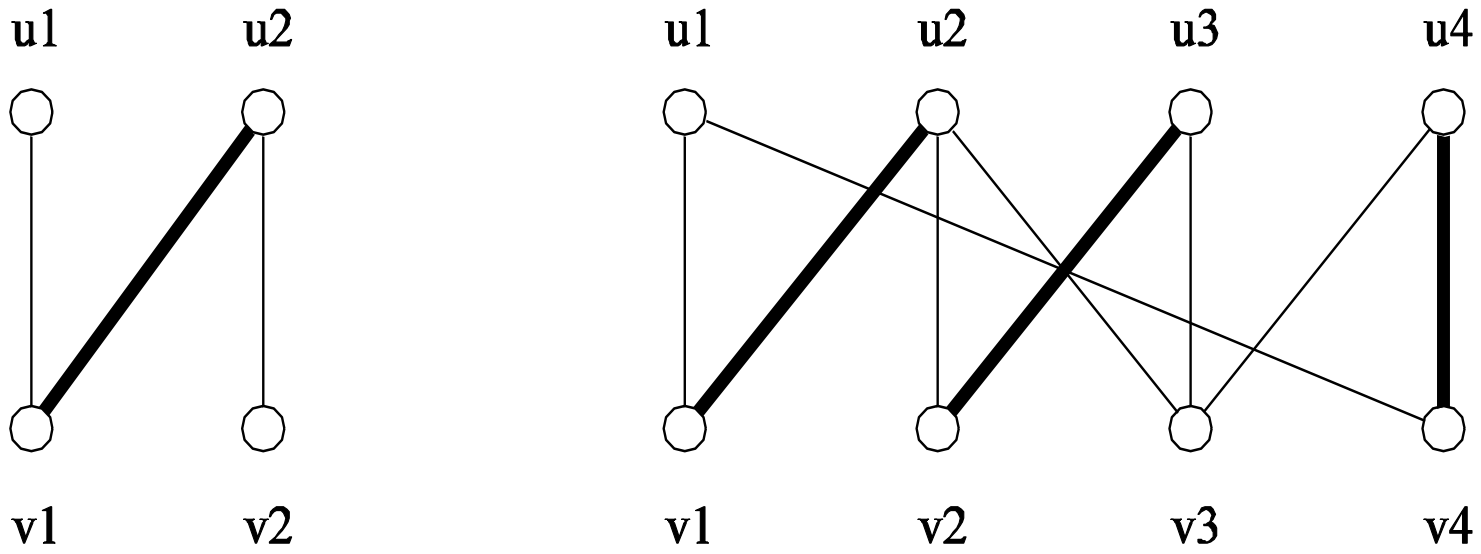
c(j,t) - deferral costs of job j if it completes at time t
Necessary Assumption c(j,t) >=c(j,t') for all t>t'

Output

A scheduling that minimizes the sum of deferral costs

Weighted Bipartite Matchings

Goal: Find a matching with maximum total weight



Reduce to min-cost flow by adding a source and a sink.