CS38
Introduction to Algorithms

Lecture 13

May 13, 2014

Outline

- Network flow
  - – finishing edge-disjoint paths
  - – assignment problem
- Linear programming

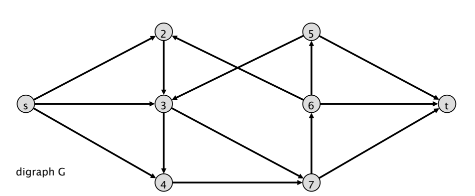
* slides from Kevin Wayne

Edge-disjoint paths

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightarrow t$ paths.



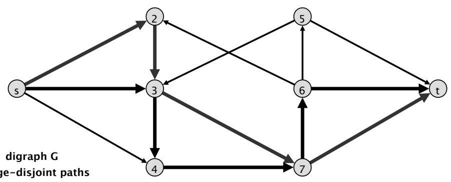
digraph G

Edge-disjoint paths

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem. Given a digraph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint $s \rightsquigarrow t$ paths.

Ex. Communication networks.



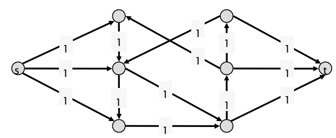
digraph G
2 edge-disjoint paths

Edge-disjoint paths

Max flow formulation. Assign unit capacity to every edge.

Theorem. Max number edge-disjoint $s \rightsquigarrow t$ paths equals value of max flow.
Pf. $\leq$

- Suppose there are k edge-disjoint $s \rightsquigarrow t$ paths $P_1, \ldots, P_k$.
- Set $f(e) = 1$ if e participates in some path P_j ; else set $f(e) = 0$.
- Since paths are edge-disjoint, f is a flow of value k. ▪



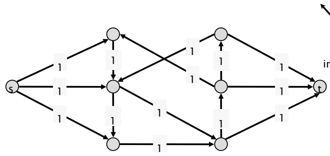
Edge-disjoint paths

Max flow formulation. Assign unit capacity to every edge.

Theorem. Max number edge-disjoint $s \rightsquigarrow t$ paths equals value of max flow.
Pf. $\geq$

- Suppose max flow value is k.
- Integrality theorem implies there exists 0-1 flow f of value k.
- Consider edge (s, u) with $f(s, u) = 1$.
  - by conservation, there exists an edge (u, v) with $f(u, v) = 1$
  - continue until reach t, always choosing a new edge
- Produces k (not necessarily simple) edge-disjoint paths. ▪

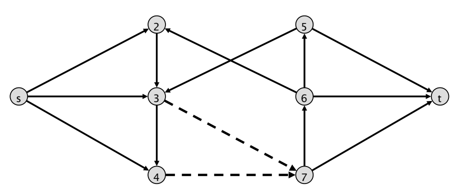
can eliminate cycles
to get simple paths
in O(mn) time if desired
(flow decomposition)



Network connectivity

Def. A set of edges $F \subseteq E$ disconnects t from s if every $s \rightsquigarrow t$ path uses at least one edge in F.

Network connectivity. Given a digraph $G = (V, E)$ and two nodes s and t, find min number of edges whose removal disconnects t from s.

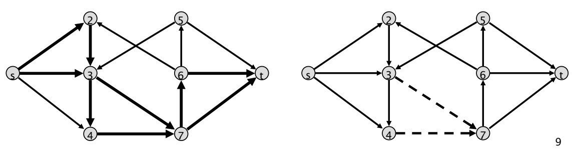


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \to t$ paths is equal to the min number of edges whose removal disconnects t from s.

Pf. $\leq$

- Suppose the removal of $F \subseteq E$ disconnects t from s, and $|F| = k$.
- Every $s \to t$ path uses at least one edge in F.
- Hence, the number of edge-disjoint paths is $\leq k$. ▪

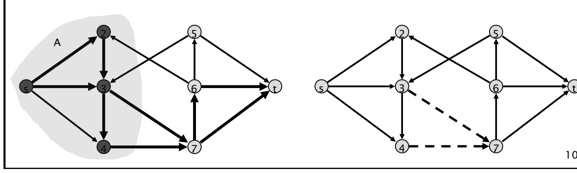


Menger's theorem

Theorem. [Menger 1927] The max number of edge-disjoint $s \to t$ paths equals the min number of edges whose removal disconnects t from s.

Pf. $\geq$

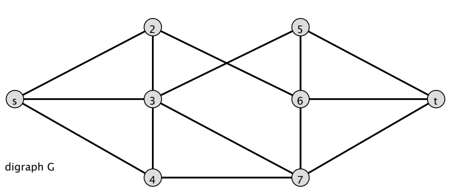
- Suppose max number of edge-disjoint paths is k.
- Then value of max flow $= k$.
- Max-flow min-cut theorem $\Rightarrow$ there exists a cut (A, B) of capacity k.
- Let F be set of edges going from A to B.
- $|F| = k$ and disconnects t from s. ▪



Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

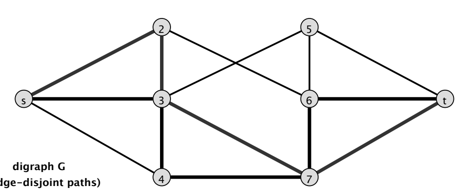


digraph G

Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.

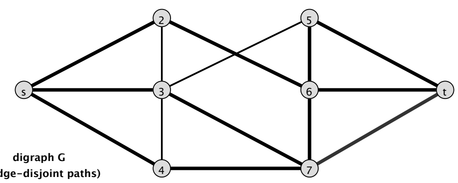


digraph G
(2 edge-disjoint paths)

Edge-disjoint paths in undirected graphs

Def. Two paths are edge-disjoint if they have no edge in common.

Disjoint path problem in undirected graphs. Given a graph $G = (V, E)$ and two nodes s and t, find the max number of edge-disjoint s-t paths.



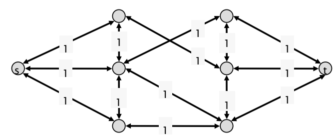
digraph G
(3 edge-disjoint paths)

Edge-disjoint paths in undirected graphs

Max flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Observation. Two paths P_1 and P_2 may be edge-disjoint in the digraph but not edge-disjoint in the undirected graph.

if P_1 uses edge (u, v)
and P_2 uses its antiparallel edge (v, u)



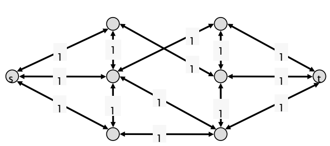
Edge-disjoint paths in undirected graphs

Max flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e', either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

Pf. [by induction on number of such pairs of antiparallel edges]

- Suppose $f(e) > 0$ and $f(e') > 0$ for a pair of antiparallel edges e and e'.
- Set $f(e) = f(e) - \delta$ and $f(e') = f(e') - \delta$, where $\delta = \min \{ f(e), f(e') \}$.
- f is still a flow of the same value but has one fewer such pair. ▪

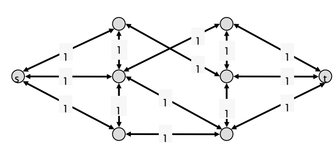


Edge-disjoint paths in undirected graphs

Max flow formulation. Replace each edge with two antiparallel edges and assign unit capacity to every edge.

Lemma. In any flow network, there exists a maximum flow f in which for each pair of antiparallel edges e and e', either $f(e) = 0$ or $f(e') = 0$ or both. Moreover, integrality theorem still holds.

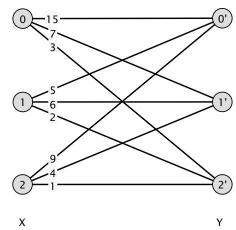
Theorem. Max number edge-disjoint $s \rightsquigarrow t$ paths equals value of max flow.
Pf. Similar to proof in digraphs; use lemma.



Assignment problem
a.k.a.
minimum-weight
perfect matching

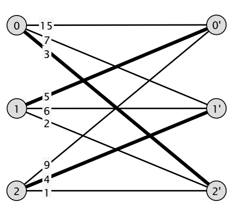
Assignment problem

Input. Weighted, complete bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y|$.
Goal. Find a perfect matching of min weight.



Assignment problem

Input. Weighted, complete bipartite graph $G = (X \cup Y, E)$ with $|X| = |Y|$.
Goal. Find a perfect matching of min weight.



min-cost perfect matching
M = { 0-2', 1-0', 2-1' }
cost(M) = 3 + 5 + 4 = 12

Applications

Natural applications.
- Match jobs to machines.
- Match personnel to tasks.
- Match students to writing seminars.

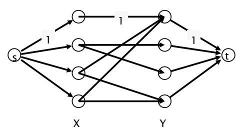
Non-obvious applications.
- Vehicle routing.
- Kidney exchange.
- Signal processing.
- Earth-mover's distance.
- Multiple object tracking.
- Virtual output queueing.
- Handwriting recognition.
- Locating objects in space.
- Approximate string matching.
- Enhance accuracy of solving linear systems of equations.

Bipartite matching

Bipartite matching. Can solve via reduction to maximum flow.

Flow. During Ford-Fulkerson, all residual capacities and flows are 0-1; flow corresponds to edges in a matching M.

Residual graph G_M simplifies to:
- If $(x, y) \notin M$, then (x, y) is in G_M.
- If $(x, y) \in M$, then (y, x) is in G_M.



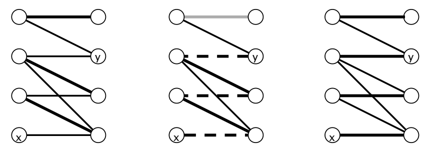
Augmenting path simplifies to:
- Edge from s to an unmatched node $x \in X$,
- Alternating sequence of unmatched and matched edges,
- Edge from unmatched node $y \in Y$ to t.

Alternating path

Def. An alternating path P with respect to a matching M is an alternating sequence of unmatched and matched edges, starting from an unmatched node $x \in X$ and going to an unmatched node $y \in Y$.

Key property. Can use P to increase by one the cardinality of the matching.
Pf. Set $M' = M \oplus P$.

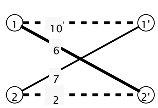
symmetric difference



matching M   alternating path P   matching M'

Assignment problem: successive shortest path algorithm

Cost of alternating path. Pay $c(x, y)$ to match x-y; receive $c(x, y)$ to unmatch.



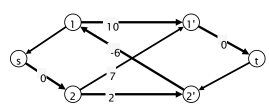
P = 2 → 2' → 1 → 1'
cost(P) = 2 - 6 + 10 = 6

Shortest alternating path. Alternating path from any unmatched node $x \in X$ to any unmatched node $y \in Y$ with smallest cost.

Successive shortest path algorithm.
- Start with empty matching.
- Repeatedly augment along a shortest alternating path.

Finding the shortest alternating path

Shortest alternating path. Corresponds to minimum cost $s \rightarrow t$ path in G_M.



Concern. Edge costs can be negative.

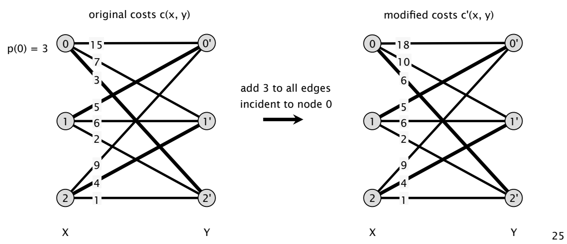
Fact. If always choose shortest alternating path, then G_M contains no negative cycles ⇒ can compute using Bellman-Ford.

Our plan. Avoid negative edge costs (and negative cycles) ⇒ can compute using Dijkstra.

Equivalent assignment problem

intuition. Adding a constant $p(x)$ to the cost of every edge incident to node $x \in X$ does not change the min-cost perfect matching(s).

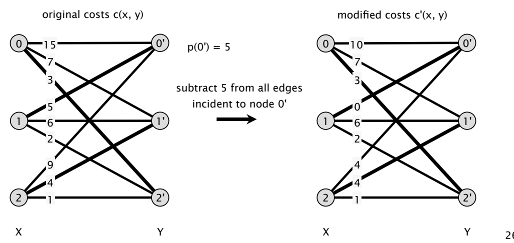
Pf. Every perfect matching uses exactly one edge incident to node x. ▪



Equivalent assignment problem

intuition. Subtracting a constant $p(y)$ to the cost of every edge incident to node $y \in Y$ does not change the min-cost perfect matching(s).

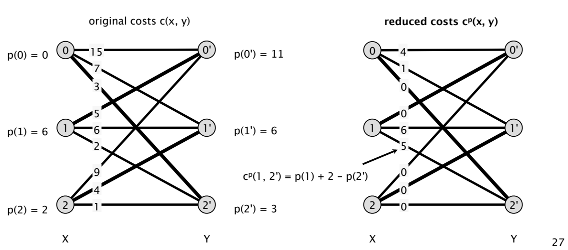
Pf. Every perfect matching uses exactly one edge incident to node y. ▪



Reduced costs

Reduced costs. For $x \in X, y \in Y$, define $c^p(x, y) = p(x) + c(x, y) - p(y)$.

Observation 1. Finding a min-cost perfect matching with reduced costs is equivalent to finding a min-cost perfect matching with original costs.



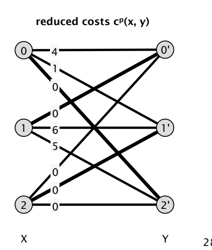
Compatible prices

Compatible prices. For each node $v \in X \cup Y$, maintain prices $p(v)$ such that:

- $c^p(x, y) \geq 0$ for all $(x, y) \notin M$.
- $c^p(x, y) = 0$ for all $(x, y) \in M$.

Observation 2. If prices p are compatible with a perfect matching M, then M is a min-cost perfect matching.

Pf. Matching M has 0 cost. ▪



Successive shortest path algorithm

SUCCESSIVE-SHORTEST-PATH (X, Y, c)

$M \leftarrow \varnothing$.
FOREACH $v \in X \cup Y : p(v) \leftarrow 0$.

prices p are compatible with M
$c^p(x, y) = c(x, y) \geq 0$

WHILE (M is not a perfect matching)
  $d \leftarrow$ shortest path distances using costs c^p.
  $P \leftarrow$ shortest alternating path using costs c^p.
  $M \leftarrow$ updated matching after augmenting along P.
  FOREACH $v \in X \cup Y : p(v) \leftarrow p(v) + d(v)$.

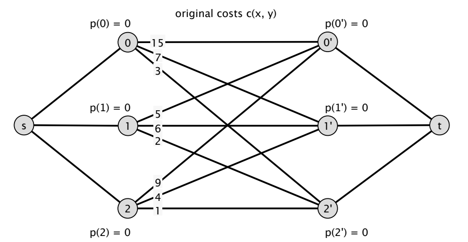
RETURN M.

29

Successive shortest path algorithm

Initialization.

- $M = \varnothing$.
- For each $v \in X \cup Y : p(v) \leftarrow 0$.

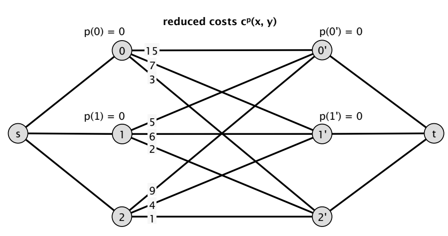


30

Successive shortest path algorithm

Initialization.

- $M = \varnothing$.
- For each $v \in X \cup Y : p(v) \leftarrow 0$.

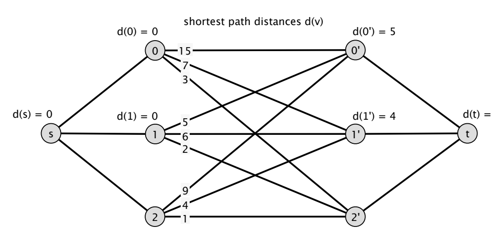


31

Successive shortest path algorithm

Step 1.

- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

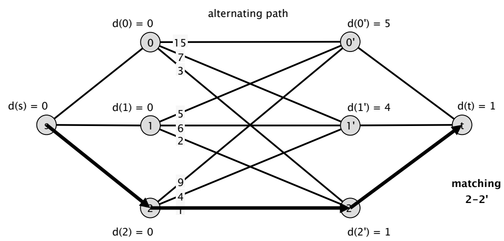


32

Successive shortest path algorithm

Step 1.

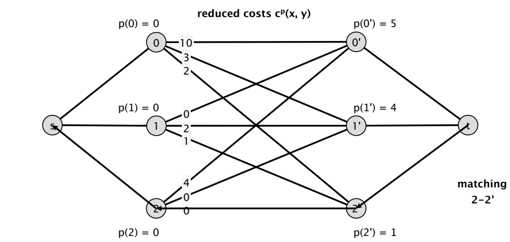
- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.



Successive shortest path algorithm

Step 1.

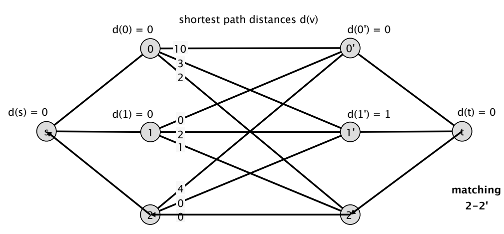
- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.



Successive shortest path algorithm

Step 2.

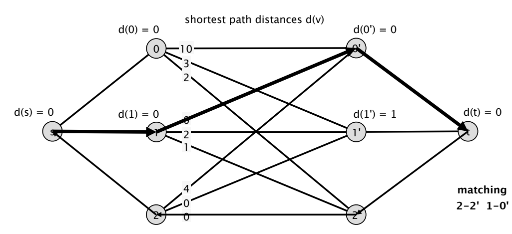
- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.



Successive shortest path algorithm

Step 2.

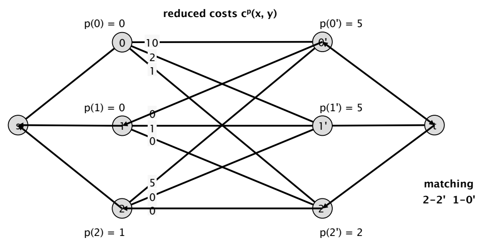
- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.



Successive shortest path algorithm

Step 2.

- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

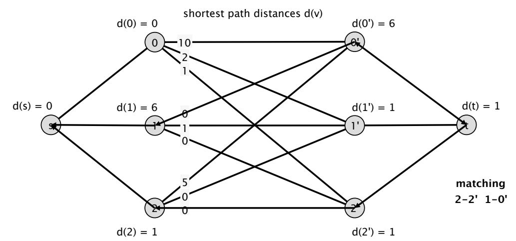


37

Successive shortest path algorithm

Step 3.

- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

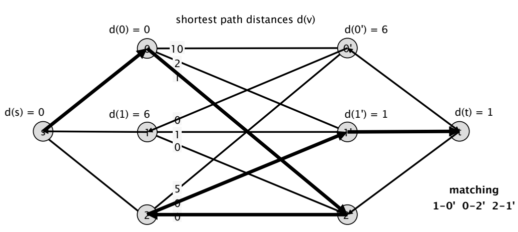


38

Successive shortest path algorithm

Step 3.

- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

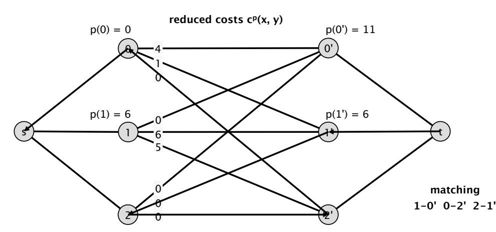


39

Successive shortest path algorithm

Step 3.

- Compute shortest path distances $d(v)$ from s to v using $c^p(x, y)$.
- Update matching M via shortest path from s to t.
- For each $v \in X \cup Y$: $p(v) \leftarrow p(v) + d(v)$.

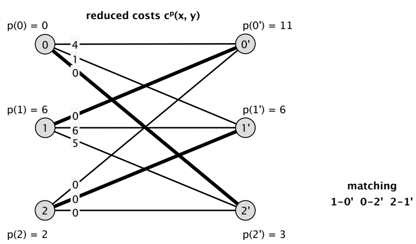


40

Successive shortest path algorithm

Termination.

- M is a perfect matching.
- Prices p are compatible with M.



Maintaining compatible prices

Lemma 1. Let p be compatible prices for M. Let d be shortest path distances in G_M with costs c^p. All edges (x, y) on shortest path have $c^{p+d}(x, y) = 0$.

forward or reverse edges

Pf. Let (x, y) be some edge on shortest path.

- If $(x, y) \in M$, then (y, x) on shortest path and $d(x) = d(y) - c^p(x, y)$;
  If $(x, y) \notin M$, then (x, y) on shortest path and $d(y) = d(x) + c^p(x, y)$.
- In either case, $d(x) + c^p(x, y) - d(y) = 0$.
- By definition, $c^p(x, y) = p(x) + c(x, y) - p(y)$.
- Substituting for $c^p(x, y)$ yields $(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) = 0$.
- In other words, $c^{p+d}(x, y) = 0$. ▪

Given prices p, the reduced cost of edge (x, y) is
$c^p(x, y) = p(x) + c(x, y) - p(y)$.

Maintaining compatible prices

Lemma 2. Let p be compatible prices for M. Let d be shortest path distances in G_M with costs c^p. Then $p' = p + d$ are also compatible prices for M.

Pf. $(x, y) \in M$

- (y, x) is the only edge entering x in G_M. Thus, (y, x) on shortest path.
- By LEMMA 1, $c^{p+d}(x, y) = 0$.

Pf. $(x, y) \notin M$

- (x, y) is an edge in $G_M \Rightarrow d(y) \le d(x) + c^p(x, y)$.
- Substituting $c^p(x, y) = p(x) + c(x, y) - p(y) \ge 0$ yields
  $(p(x) + d(x)) + c(x, y) - (p(y) + d(y)) \ge 0$.
- In other words, $c^{p+d}(x, y) \ge 0$. ▪

Prices p are compatible with matching M:

- $c^p(x, y) \ge 0$ for all $(x, y) \notin M$.
- $c^p(x, y) = 0$ for all $(x, y) \in M$.

Maintaining compatible prices

Lemma 3. Let p be compatible prices for M and let M' be matching obtained by augmenting along a min cost path with respect to c^{p+d}. Then $p' = p + d$ are compatible prices for M'.

Pf.

- By LEMMA 2, the prices $p + d$ are compatible for M.
- Since we augment along a min-cost path, the only edges (x, y) that swap into or out of the matching are on the min-cost path.
- By LEMMA 1, these edges satisfy $c^{p+d}(x, y) = 0$.
- Thus, compatibility is maintained. ▪

Prices p are compatible with matching M:

- $c^p(x, y) \ge 0$ for all $(x, y) \notin M$.
- $c^p(x, y) = 0$ for all $(x, y) \in M$.

Successive shortest path algorithm: analysis

Invariant. The algorithm maintains a matching M and compatible prices p.
Pf. Follows from LEMMA 2 and LEMMA 3 and initial choice of prices. ▪

Theorem. The algorithm returns a min-cost perfect matching.
Pf. Upon termination M is a perfect matching, and p are compatible prices.
Optimality follows from OBSERVATION 2. ▪

Theorem. The algorithm can be implemented in $O(n^3)$ time.
Pf.

- Each iteration increases the cardinality of M by $1 \Rightarrow n$ iterations.
- Bottleneck operation is computing shortest path distances d.
  Since all costs are nonnegative, each iteration takes $O(n^2)$ time
  using (dense) Dijkstra. ▪

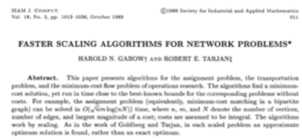
45

Weighted bipartite matching

Weighted bipartite matching. Given a weighted bipartite graph with n nodes and m edges, find a maximum cardinality matching of minimum weight.

Theorem. [Fredman-Tarjan 1987] The successive shortest path algorithm solves the problem in $O(n^2 + m\,n \log n)$ time using Fibonacci heaps.

Theorem. [Gabow-Tarjan 1989] There exists an $O(m\,n^{1/2} \log(nC))$ time algorithm for the problem when the costs are integers between 0 and C.


SIAM J. COMPUT.
Vol. 18, No. 5, pp. 1013-1036, October 1989

©1989 Society for Industrial and Applied Mathematics
011

FASTER SCALING ALGORITHMS FOR NETWORK PROBLEMS*

HAROLD N. GABOW† AND ROBERT E. TARJAN‡

Abstract. This paper presents algorithms for the assignment problem, the transportation problem, and the minimum-cost flow problem of operations research. The algorithms find a minimum-cost solution, yet run in time close to the best-known bounds for the corresponding problems without costs. For example, the assignment problem (equivalently, minimum-cost matching in a bipartite graph) can be solved in $O(\sqrt{n}m \log(nN))$ time, where n, m, and N denote the number of vertices, number of edges, and largest magnitude of a cost; costs are assumed to be integral. The algorithms work by scaling. As in the work of Goldberg and Tarjan, in each scaled problem an approximate optimum solution is found, rather than an exact optimum.

46

Linear programming

May 12, 2014 CS38 Lecture 13 47

Linear Programming

Linear programming. Optimize a linear function subject to linear inequalities.

$$
\begin{aligned}
\text{(P)} \quad \max \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j = b_i \quad 1 \le i \le m \\
& x_j \ge 0 \quad 1 \le j \le n
\end{aligned}
$$

$$
\begin{aligned}
\text{(P)} \quad \max \quad & c^T x \\
\text{s.t.} \quad & Ax = b \\
& x \ge 0
\end{aligned}
$$

48

Linear Programming

Linear programming. Optimize a linear function subject to linear inequalities.

Generalizes: $Ax = b$, 2-person zero-sum games, shortest path, max flow, assignment problem, matching, multicommodity flow, MST, min weighted arborescence, …

Why significant?
- Design poly-time algorithms.
- Design approximation algorithms.
- Solve NP-hard problems using branch-and-cut.

Ranked among most important scientific advances of 20th century.

Linear programming running example

Brewery Problem

Small brewery produces ale and beer.
- Production limited by scarce resources: corn, hops, barley malt.
- Recipes for ale and beer require different proportions of resources.

Beverage	Corn (pounds)	Hops (ounces)	Malt (pounds)	Profit ($)
Ale (barrel)	5	4	35	13
Beer (barrel)	15	4	20	23
constraint	480	160	1190	

How can brewer maximize profits?
- Devote all resources to ale: 34 barrels of ale ⇒ $442
- Devote all resources to beer: 32 barrels of beer ⇒ $736
- 7.5 barrels of ale, 29.5 barrels of beer ⇒ $776
- 12 barrels of ale, 28 barrels of beer ⇒ $800

Brewery Problem

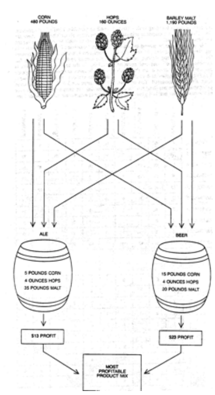
objective function

Ale Beer

$$
\begin{array}{lrcrcrl}
\max & 13A & + & 23B & & & \text{Profit} \\
\text{s. t.} & 5A & + & 15B & \leq & 480 & \text{Corn} \\
& 4A & + & 4B & \leq & 160 & \text{Hops} \\
& 35A & + & 20B & \leq & 1190 & \text{Malt} \\
& A & , & B & \geq & 0 &
\end{array}
$$

constraint

decision variable



Linear programming standard form

Standard Form LP

"Standard form" LP.
- Input: real numbers a_{ij}, c_j, b_i.
- Output: real numbers x_j.
- n = # decision variables, m = # constraints.
- Maximize linear objective function subject to linear inequalities.

$$
\begin{array}{llrcll}
\text{(P)} & \max & \sum_{j=1}^{n} c_j x_j & & & \\
& \text{s.t.} & \sum_{j=1}^{n} a_{ij} x_j & = & b_i & 1 \le i \le m \\
& & x_j & \ge & 0 & 1 \le j \le n
\end{array}
$$

$$
\begin{array}{llrcl}
\text{(P)} & \max & c^T x & & \\
& \text{s.t.} & Ax & = & b \\
& & x & \ge & 0
\end{array}
$$

Linear. No x^2, $x\,y$, $\arccos(x)$, etc.
Programming. Planning (term predates computer programming).

Brewery Problem: Converting to Standard Form

Original input.

$$
\begin{array}{lrcrcr}
\max & 13A & + & 23B & & \\
\text{s.t.} & 5A & + & 15B & \le & 480 \\
& 4A & + & 4B & \le & 160 \\
& 35A & + & 20B & \le & 1190 \\
& A & , & B & \ge & 0
\end{array}
$$

Standard form.
- Add slack variable for each inequality.
- Now a 5-dimensional problem.

$$
\begin{array}{lrcrcccccrcr}
\max & 13A & + & 23B & & & & & & & & \\
\text{s.t.} & 5A & + & 15B & + & S_C & & & & & = & 480 \\
& 4A & + & 4B & & & + & S_H & & & = & 160 \\
& 35A & + & 20B & & & & & + & S_M & = & 1190 \\
& A & , & B & , & S_C & , & S_H & , & S_M & \ge & 0
\end{array}
$$

Equivalent Forms

Easy to convert variants to standard form.

$$
\begin{array}{llrcl}
\text{(P)} & \max & c^T x & & \\
& \text{s.t.} & Ax & = & b \\
& & x & \ge & 0
\end{array}
$$

Less than to equality:

$x + 2y - 3z \le 17 \quad \Rightarrow \quad x + 2y - 3z + s = 17, s \ge 0$

Greater than to equality:

$x + 2y - 3z \ge 17 \quad \Rightarrow \quad x + 2y - 3z - s = 17, s \ge 0$

Min to max:

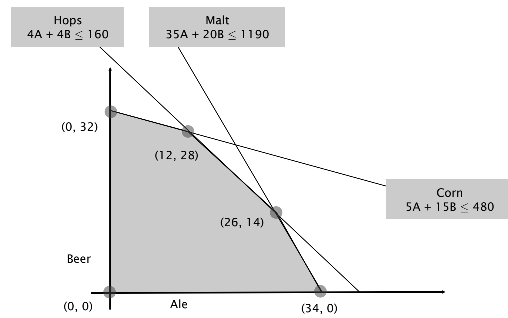
$\min\ x + 2y - 3z \quad \Rightarrow \quad \max\ -x - 2y + 3z$

Unrestricted to nonnegative:

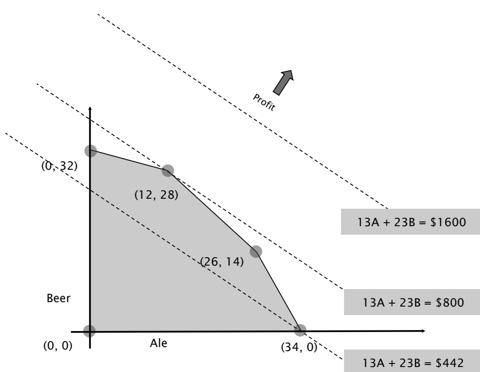
x unrestricted $\quad \Rightarrow \quad x = x^+ - x^-,\ x^+ \ge 0, x^- \ge 0$

Linear programming geometric perspective

Brewery Problem: Feasible Region

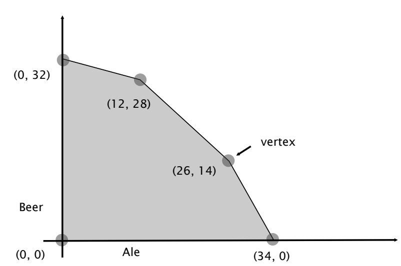


Brewery Problem: Objective Function



Brewery Problem: Geometry

Brewery problem observation. Regardless of objective function coefficients, an optimal solution occurs at a vertex.

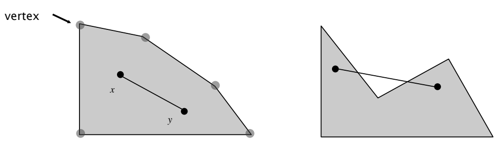


Convexity

Convex set. If two points x and y are in the set, then so is $\lambda\ x + (1-\lambda)\ y$ for $0 \le \lambda \le 1$.

convex combination

Vertex. A point x in the set that can't be written as a strict convex combination of two distinct points in the set.



Observation. LP feasible region is a convex set.

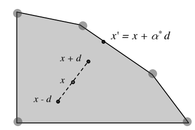
61

Geometric perspective

Theorem. If there exists an optimal solution to (P), then there exists one that is a vertex.

$$
\begin{array}{llrcl}
\text{(P)} & \max & c^T x & & \\
 & \text{s.t.} & Ax & = & b \\
 & & x & \ge & 0
\end{array}
$$

Intuition. If x is not a vertex, move in a non-decreasing direction until you reach a boundary. Repeat.



62