Networks and Algorithms

CE 377K

February 10, 2015

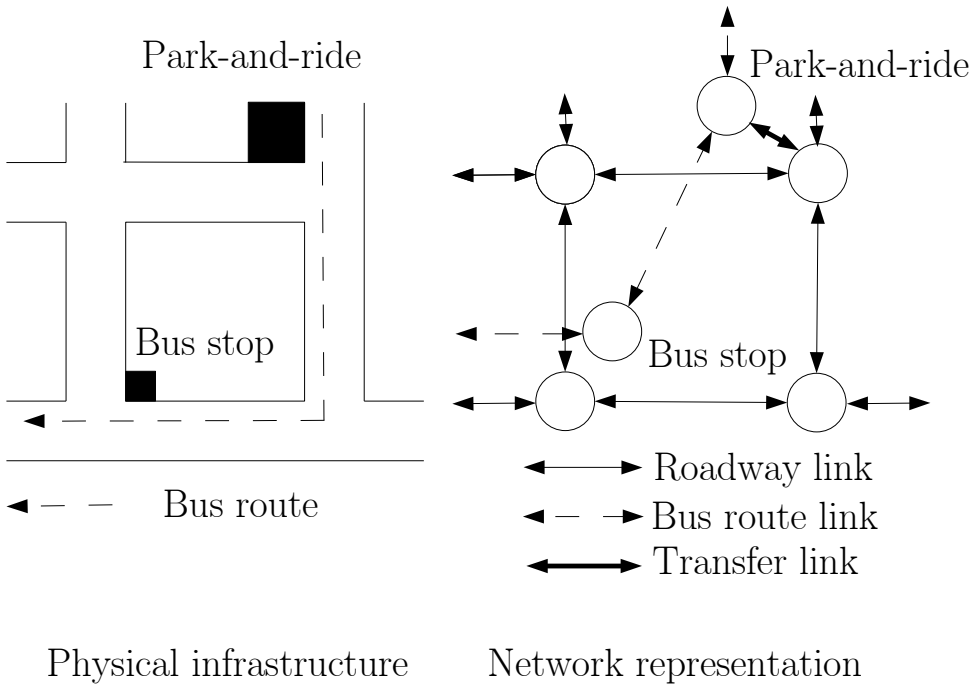
REVIEW

HW 1 due Thursday

Simulated annealing and genetic algorithms

NETWORK OPTIMIZATION

A network is a mathematical object consisting of *nodes* connected by *links*.



The set of nodes is N, the set of links is A, and the network is denoted $G = (N, A)$

Networks are used to represent many types of systems:

- Transportation infrastructure
- Telecommunications and computer networks
- Power grids
- Project structure and task dependencies
- Social connections among people

Since there is a common structure among all of these systems, it is very worthwhile to study optimization problems on abstract networks.

Nodes are typically denoted i, j, etc.

Links are typically denoted (i, j), where i is the upstream node (the *tail*) and j is the downstream ndoe (the *head*).

A *path* is a sequence of nodes, typically denoted $[i, j, \cdots, z]$, where there is a link connecting each consecutive pair of nodes.

A network is *strongly connected* if there is a path between every pair of nodes in the network.

A network is *weakly connected* if adding "mirrors" to each link would make it strongly connected.

A path is a *cycle* if its start and end nodes are the same.

A network is *acyclic* if it contains no cycles.

A network is a *tree* if it contains no cycles even when "mirrors" are added.

A network is a *spanning tree* if it is a tree and is weakly connected.

There are slightly different versions of this terminology in use. This slide is the convention for this class.

In many network problems, we introduce additional information for nodes and links. (Not all problems use all of these.)

A link (i, j) may have a *flow* x_{ij}, a *cost* c_{ij}, and a *capacity* u_{ij}. (Rarely, a link may have a lower bound ℓ_{ij} on its flow.)

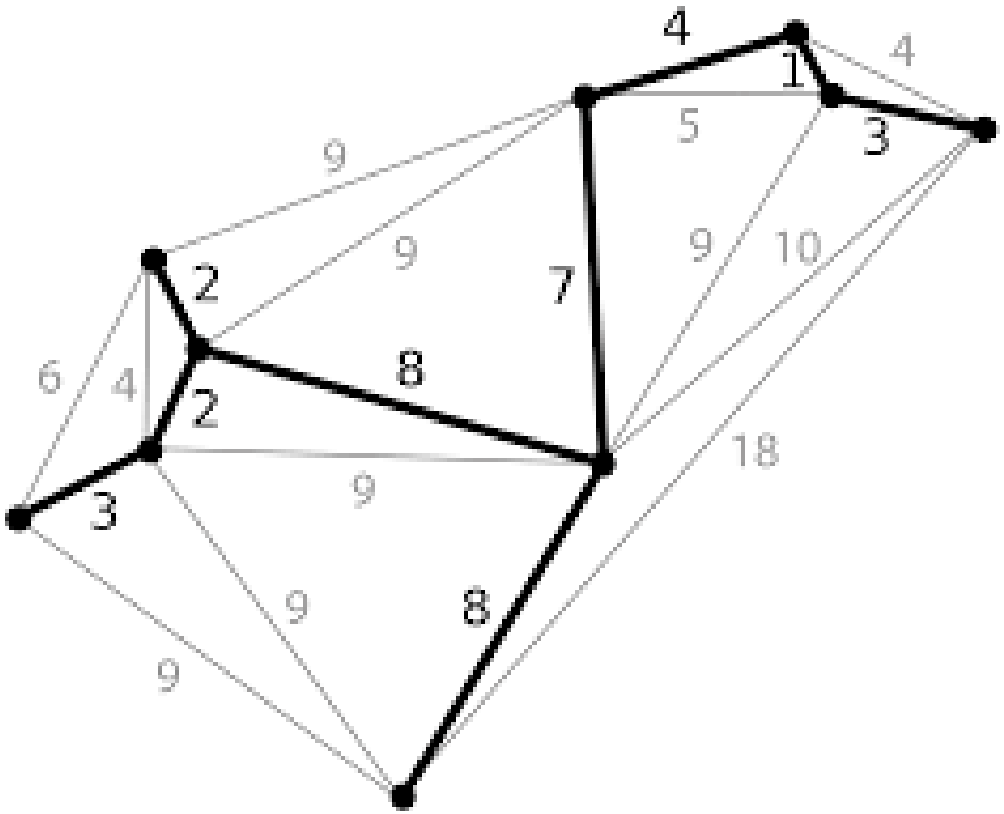
A node i may have a *supply* or *demand* b_i. (Positive for supply, negative for demand.)

A node with supply is a *source*, a node with demand is a *sink*.

We have already seen the notation $A(i)$ and $B(i)$ for the sets of links leaving and entering node i.

SOME NETWORK PROBLEMS

Minimum Spanning Tree

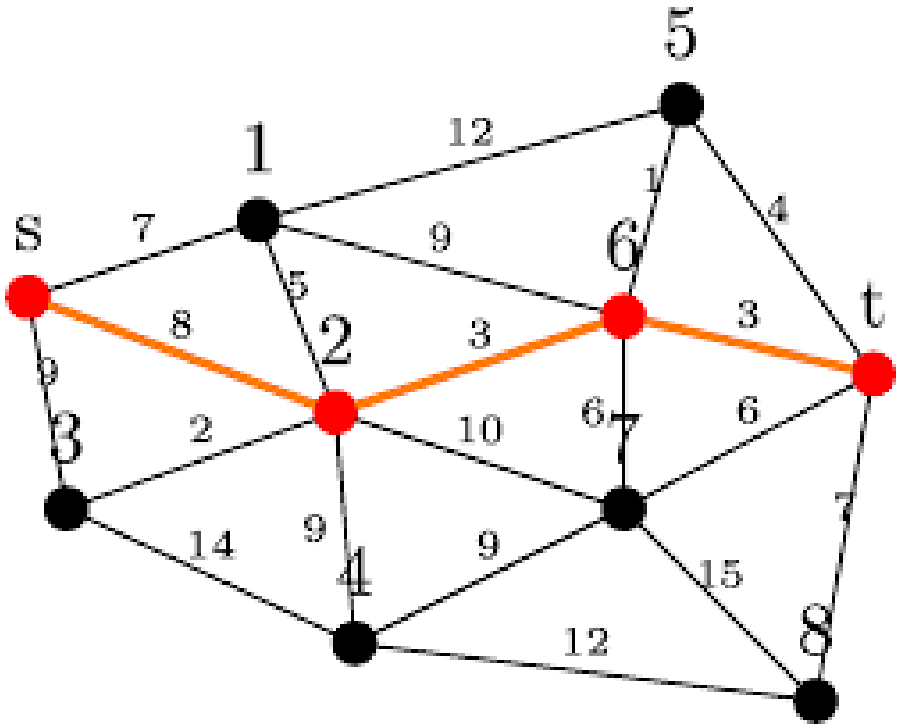


Identify a subset of links which form a spanning tree, where the total cost of the links in the tree is minimized.

Applications

- Building roads in rural areas
- Providing utilities and other infrastructure
- Espionage networks, passing messages between spies

Shortest Path Problem

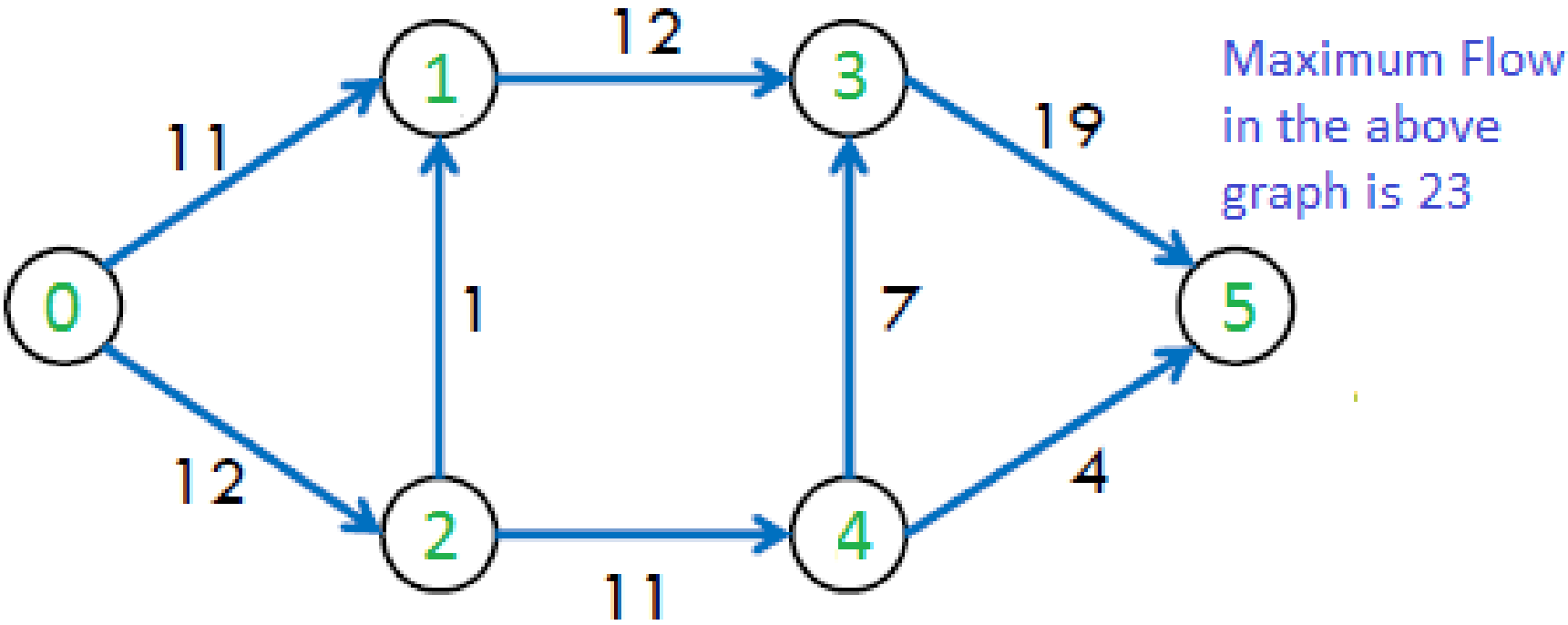


Identify a path connecting a given origin and destination, where the total cost of the links in the path is minimized.

Applications

- Vehicle routing
- Critical path analysis in project management
- Six degrees of Kevin Bacon

Maximum Flow Problem

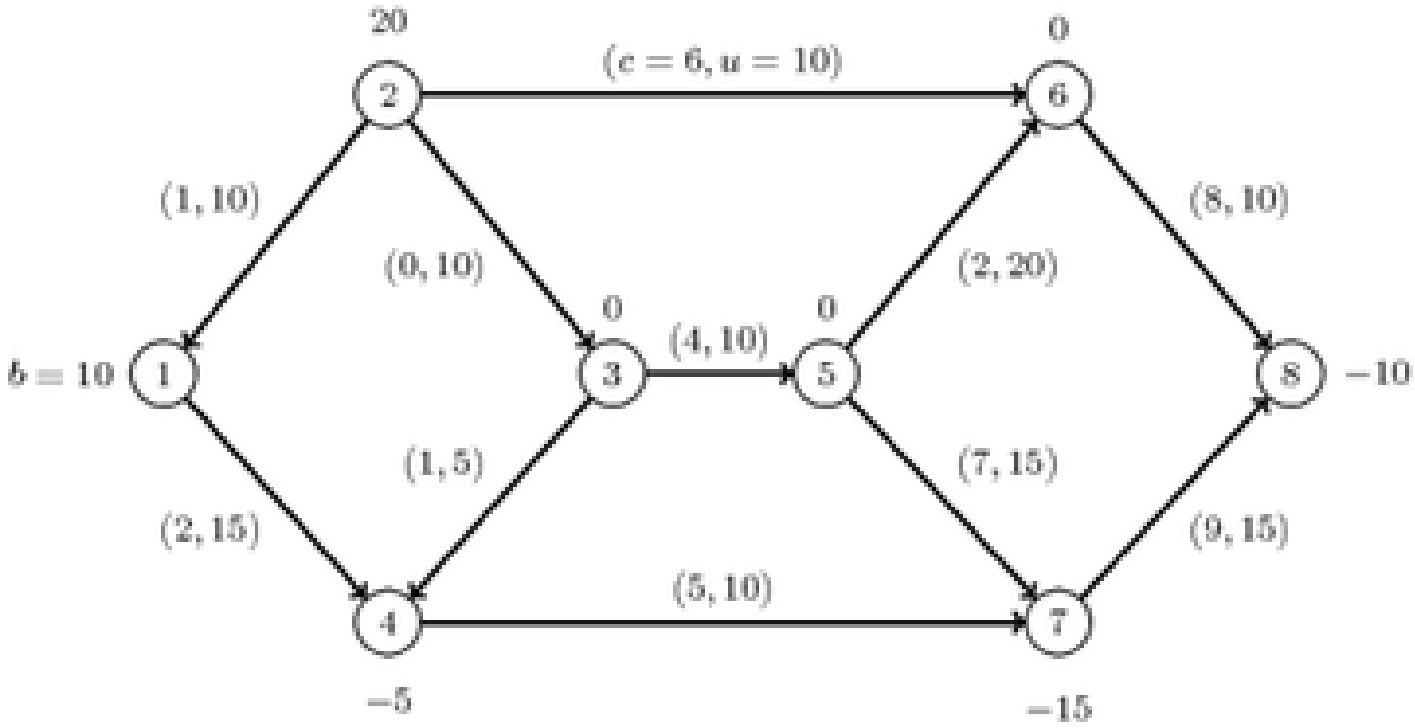


What is the greatest amount of flow that can be shipped between a given source and sink without exceeding link capacities?

Applications

- Determining the capacity of a network
- Identifying critical links in a network
- Rounding a matrix

Minimum Cost Flow Problem

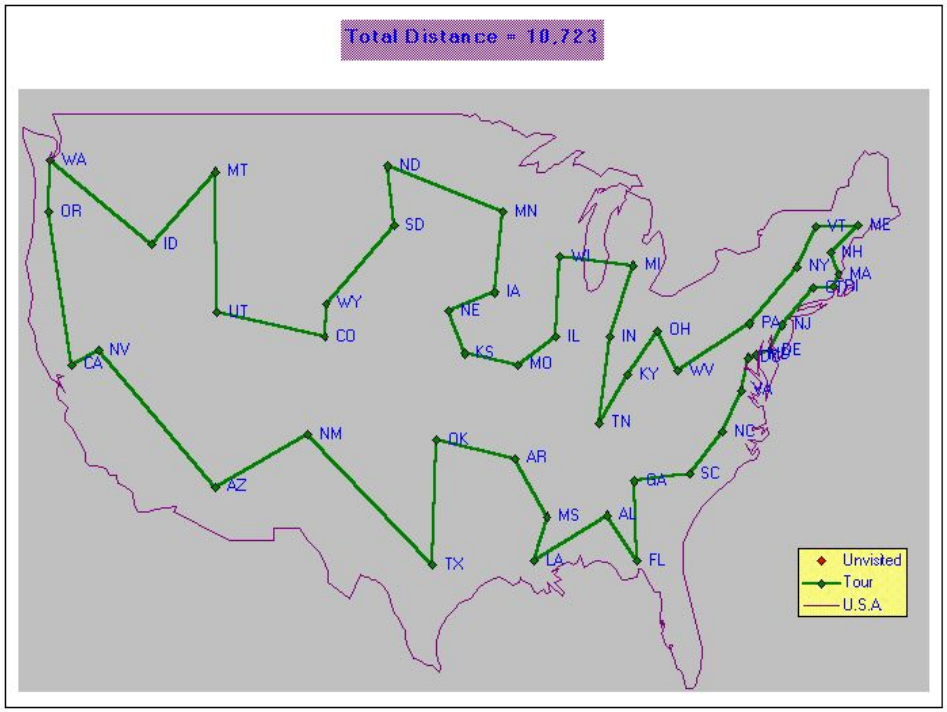


Respecting capacities, find link flows which balance supply and demand among sources and sinks, with minimum total cost.

Applications

- Logistics and shipping
- Earthwork in roadway construction
- Passenger selection in ridesharing

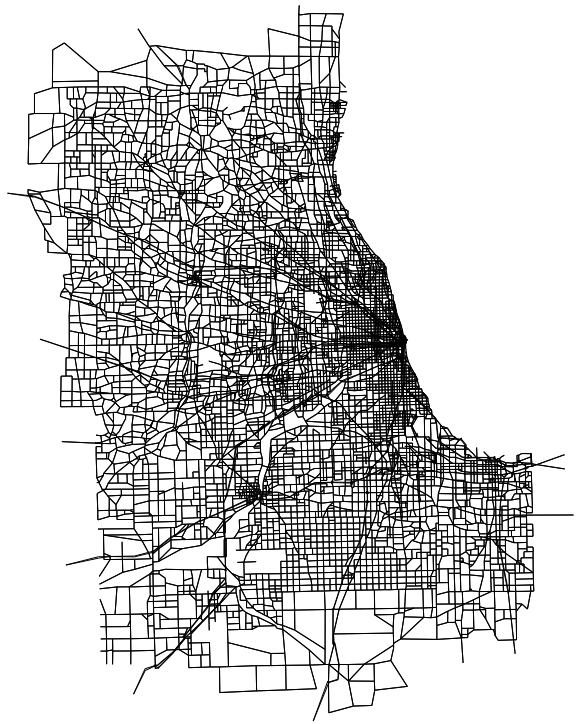
Traveling Salesman Problem



Find a path which passes through all nodes and returns to the origin with minimum total cost.

ALGORITHMS AND COMPLEXITY

Several times in this class, we've alluded to the large size of real-world optimization problems.



Now that we are looking at specific types of problems and algorithms, we can more precisely relate the size of a problem to the time it takes to solve it.

More specifically, we are given both a problem and a solution algorithm, and want to know how long it takes to solve it.

Example. In the bisection method, if the initial interval has a width of 10, how many iterations will it take before the optimal solution is known to within 0.1? Within 0.01?

In network optimization problems, the time is usually expressed in terms of the number of links m and the number of nodes n, although it can depend on other problem features as well.

A few difficulties:

- The number of steps can vary, even for problems of the same size. (Sometimes we are lucky, sometimes we are not.)
- It can be difficult to calculate the exact number of steps required.

The solution to these difficulties is the use of "big O" notation.

The big O notation is used to express the *worst case* behavior as the problem grows asymptotically larger and larger.

Formally, a function $f(x)$ is $O(g(x))$ if $f(x) \leq Cg(x)$ for some constant $g(x)$ and when x is sufficiently large.

Examples.

- $x^2 + 2x + 100$ is $O(x^2)$
- $3(x-2)^3$ is $O(x^3)$
- $3(x-2)^3$ is $O(x^4)$
- $x^{1000} + e^x$ is $O(e^x)$

It is often much easier to calculate the number of steps an algorithm needs in terms of "big O" notation, for a few reasons:

- We don't have to keep track of details, only the most significant terms.
- If something is difficult to calculate, we can work with upper bounds.

Examples.

- If a network has no "parallel" links, m is $O(n^2)$.
- If the number of links adjacent to each node is no more than a constant, m is $O(n)$.

Over the next few weeks, as we see different kinds of network algorithms we will be comparing their big-O complexity as well.

(This is surprisingly important. Even switching from an $O(n^2)$ to an $O(n \log n)$ algorithm can make a huge difference for large problems.)

A "brute force" solution to the shortest path problem is to list off all possible paths between two nodes in a network. What is the complexity of this?

This method is $O(n!)$ — you really don't want to do this. In fact you can solve this problem easily in $O(n^2)$ time, and can reduce this further by being clever.

If the big-O complexity of an algorithm is polynomial in the problem size, it is said to be a *polynomial* algorithm (in P).

Intuitively, problems in P are "easy" while problems not in P are considered "hard."

Interestingly: the shortest path problem is in P, but not the longest path problem or the traveling salesperson problem.

The minimum spanning tree, shortest path, maximum flow, and minimum cost flow problems are also in P.

This is related to one of the biggest open problems in computer science. If you can find a polynomial algorithm for the traveling salesperson problem, you will win a \$1 million prize and almost certainly win the Nobel prize in economics.