COS 226 Lecture 21: Network Flow

Classical problem-solving model (1940s)

OPERATIONS RESEARCH

Modern implementations benefit from
- Graph algorithm technology
- PQ and data structure design

Researchers still seek efficient algorithms
- many variations
- many practical applications

Optimal solutions still not known

NETWORK: weighted digraph

Abstraction for material FLOWING through the edges
- interpret edge weights as CAPACITIES

Ex: oil flowing in pipes
Ex: commodities flowing on roads and rails
Ex: bits flowing in Internet

SOURCE: node where all material originates
SINK: node where all material goes

MAXFLOW PROBLEM: assign flows to edges that
- equalize inflow and outflow at every vertex
- maximize total flow through the network

Flow network example

AUGMENTING PATH: source-sink path for increasing flow

Easy case:
- ADD flow to each edge on the path
Ex: 0-1-3-5, then 0-2-4-5

More complicated case:
- REMOVE flow from one or more edges
Ex: 0-2-3-1-4-5
Ford-Fulkerson algorithm

**GENERIC method for solving maxflow problems**

- Start with 0 flow everywhere
- **REPEAT** until no augmenting paths are left
  - Increase the flow along ANY augmenting path

**Problem 0:**
- Does this process lead to the maximum flow?

**Problem 1:** fill in unspecified details
- How do we find an augmenting path?

**Problem 2:**
- Cost can be proportional to max capacity

---

Maxflow-mincut theorem

**CUT:** set of edges separating source from sink

**THM:** maxflow is equivalent to mincut
**Proof:** [see text]

**THM:** Ford-Fulkerson method gives maximum flow
**Proof sketch:**
- If there is no augmenting path,
  - identify the first full forward or empty backward edge on every path
- That set of edges defines a min cut

**AUGMENTING-PATH ALG:** specific method for finding a path

**Design goals:**
- find paths quickly
- use as few iterations as possible

---

Edmonds-Karp algorithms

**Idea 1:** use BFS to find augmenting path
**Idea 2:** find path that increases the flow

**BOTH** easy to implement with standard PFS (!)

**RESIDUAL NETWORK**
- For each edge in original network
  - Flow \( x \) in edge \( u \rightarrow v \) with capacity \( c \)
  - Define two edges in residual network
    - **FORWARD** edge: flow \( c-x \) in edge \( u \rightarrow v \)
    - **BACKWARD** edge: flow \( -x \) in edge \( v \rightarrow u \)

**Easy implicit implementation:**
- \#define \( Q (u \rightarrow cap < 0 ? -u \rightarrow flow : u \rightarrow cap - u \rightarrow flow) \)

Graph search in residual network finds augmenting path
Residual networks

Network flow implementation

- **Tricky code for sparse graphs**
  - TWO edge representations with links to each other
  - st array has links to edge representations

```c
void GRAPHmaxflow(Graph G, int s, int t)
{
  int x, d;
  link st[maxV];
  while ((d = GRAPHpfs(G, s, t, st)) != 0)
  for (x = t; x != s; x = st[x]->dup->v)
    st[x]->flow += d; st[x]->dup->flow -= d;
}
```

To make GRAPHsearch find shortest aug path
#define P G->V - cnt

To make GRAPHsearch find max capacity aug path

Max capacity augmenting paths example

Path capacities decrease

Fewer iterations, lower cost per iteration
Shortest augmenting paths (larger example)

Max capacity augmenting paths (larger example)

Analysis of network flow algorithms

THM: ANY FF alg takes \(O(V^E M)\) time
Proof:
- mincut capacity less than \(VM\)
- aug path increases flow through cut by at least 1
- graph search takes \(O(E)\) time

THM: Shortest aug-path alg takes \(O(VE^2)\) time
Proof:
- aug paths increase in length
- at most \(E\) paths for each of \(V\) lengths
- total of at most \(VE\) aug paths
- graph search takes \(O(E)\) time

THM: Max-capacity aug-path alg takes \(O(E^2 \log V \log M)\) time
Proof: [see text]

Network-flow algorithms

Best known worst-case running times
- 1970 \(V^2 E\)
- 1977 \(V^2 E^{(1/2)}\)
- 1978 \(V^3\)
- 1978 \(V^{(5/3)} E^{(2/3)}\)
- 1980 \(VE \log V\)
- 1986 \(VE \log(V^{2/E})\)

Generally NOT relevant in practice
- most improvements are for dense graphs (rare in practice)
- worst-case bounds are overly pessimistic
- simple (but not dumb) algorithms may be preferred in practice

Sparse graphs
- shortest: \(O(V^3)\)
- max capacity: \(O(V^2 \log V \log M)\)

But research is justified:
- simple \(O(E)\) algorithm could still exist!
**Matching**

MATCHING: set of edges with no vertex included twice

MAXIMUM MATCHING: no matching contains more edges

BIPARTITE GRAPH
- two sets of vertices
- all edges connect vertex in one set to vertex in the other

BIPARTITE MATCHING: maximum matching in bipartite graph

What does matching have to do with maxflow??
- bipartite matching REDUCES to maxflow
- we can use maxflow to solve it!

**Bipartite matching example**

Job Placement
- companies make job offers
- students have job choices

BIPARTITE MATCHING
- can we fill every job?
- can we employ every student?

Equivalent: Find maximal subset with no dups in

1A 1B 1C 2A 2B 2E 3C 3D 3E 4A 4B 5D 5E 5F 6C 6E 6F

**Bipartite matching reduction to maxflow**

Standard reduction (see lecture 20)
- given an instance of bipartite matching
- transform it to a maxflow problem
- solve the maxflow problem
- transform maxflow solution to bipartite matching solution

Transformation:
- keep all edges and vertices
- add SOURCE connected to all vertices in one set
- add SINK connected to all nodes of second type
- set all capacities to 1

full edges in maxflow solution give matching solution

NOTE: maxflow easier in unit-capacity networks
Bipartite matching reduction example

SOLUTION: 1-A 2-F 3-C 4-B 5-D 6-E
Alice-Adobe Bob-Yahoo Carol-HP Dave-Apple Eliza-IBM Frank-Sun

Maxflow problem-solving model

Many practical problems reduce to maxflow problems
- merchandise distribution
- matching
- scheduling
- communications networks

Maxflow algorithms provide effective solutions

NEXT STEP: add OPTIMIZATION
- multiple maxflows, in general
- which one is best??

MINCOST FLOW
- generalizes maxflow and shortest paths
- large number of practical applications
- challenge to develop efficient alg/implementation
[stay tuned]