Computer Networks: Algorithms in networking

Ankit Singla
Algorithms in networking

#0 Three example algorithmic problems in networking

#1 Linear programming: a powerful, generic tool

#2 Exploiting randomness for networking

#3 Distributed decision–making
Linear programming
Clue Words

Clue words can help you figure out how to solve problems.

Here's the problem.

In one amusement park, 21 of the rides are water rides and 16 are not water rides. How many rides are there combined?

**Addition**

Problems that combine values, such as rides, are addition problems.

In this problem the clue word "combined" tells you to add the water rides and the non-water rides.

Some other clue words that tell you a problem might use addition are: add, sum, total, plus, more, together, increase, and both.
Many problems can be expressed as LPs

#1 Traffic engineering: managing traffic on your network

#2 Matchings: inside a circuit switch

#3 Shortest paths: finding friends and lower latency

Note: better algorithms are known for each of these!
Max-flow as a linear program
Max-flow as a linear program

**Objective:**
Maximize flow from s to t

**Constraint:**
Obey edge capacities, $c_{uv}$
Max-flow LP: what are the variables?

Variables:

Flow $s \rightarrow t : f$

Flow on edge $u \rightarrow v : f_{uv}$
Max-flow LP: what are the variables?

Variables:
- Flow $s \rightarrow t : f$
- Flow on edge $u \rightarrow v : f_{uv}$

Objective:
- maximize $f$
Max-flow LP: what are the constraints?

Capacity: $f_{uv} \leq c_{uv}$

Conservation: in-flow = out-flow

\[ \sum_{u \to v} f_{uv} = \sum_{w \to u} f_{wu} \quad \text{(except at s,t)} \]

\[ \sum_{s \to v} f_{sv} = f \]

\[ \sum_{v \to t} f_{vt} = f \]

$f_{uv} \geq 0$
Max-flow LP

Variables:
- Flow s \rightarrow t: f
- Flow on edge u \rightarrow v: f_{uv}

Capacity: \( f_{uv} \leq c_{uv} \)

Conservation: in-flow = out-flow
- \( \sum_{u \rightarrow v} f_{uv} = \sum_{w \rightarrow u} f_{wu} \) (except at s, t)
- \( \sum_{s \rightarrow v} f_{sv} = f \)
- \( \sum_{v \rightarrow t} f_{vt} = f \)
- \( f_{uv} \geq 0 \)

Objective:
- maximize \( f \)

Much slower than Edmonds-Karp!
MCF: multiple commodities
MCF: multiple commodities

Variables:

Flows: \( f_x \)

Flow \( f_x \) on edge \( u \to v \): \( f_{x,uv} \)

Objective:

maximize \( \sum_x f_x \) [ or min\(_x f_x \) ]

Simplification: each source \( x \) only sends to one destination \( d(x) \)
MCF: multiple commodities

Capacity: $\sum_x f_{x,uv} \leq c_{uv}$

Conservation: in-flow = out-flow

$\sum_{u \to v} f_{x,uv} = \sum_{w \to u} f_{x,wu}$ [$! x, d(x)$]  
Flows don’t mix!

$\sum_{x \to v} f_{x,xv} = f_x$

$\sum_{v \to d(x)} f_{x,vd(x)} = f_x$

$f_{x,uv} \geq 0$

Simplification: each source $x$ only sends to one destination $d(x)$
OK, let’s try shortest path
Shortest path as a linear program

Objective:
Minimize s-t path length
(given edge lengths $w_{uv}$)

Constraint:
“Path” = connected edges
Shortest path LP: what are the variables?

Variables:
Is edge $u \rightarrow v$ on SP? $x_{uv}$

Objective:
minimize $\sum_{u \rightarrow v} x_{uv} w_{uv}$
Shortest path LP: what are the constraints?

Path $s \rightarrow t$ is connected:

\[
\sum_{u \rightarrow v} x_{uv} - \sum_{v \rightarrow w} x_{vw} = 0 \quad [! s, t]
\]

\[
= 1 \quad [u = s]
\]

\[
= -1 \quad [u = t]
\]

Also: $x_{uv} \in \{0, 1\}$?

$x_{uv} \in [0, 1]$ is enough!

Lucky in this case.
Define $d_u$ as $s \rightarrow u$ distance

\[ d_s = 0 \]

for each $u \rightarrow v$: $d_v \leq d_u + w_{uv}$

\[ d_c \leq 2 \]

\[ d_g \leq d_c + 8 \]

\[ d_g \leq d_a + 2 \]

\[ \ldots \]
Define $d_u$ as $s\rightarrow u$ distance

$d_s = 0$

for each $u\rightarrow v$: $d_v \leq d_u + w_{uv}$

maximize $d_t$

What happens if I minimize?
This has a neat physical interpretation!

Now pull S and T apart as far as possible
This has a neat physical interpretation!

SP algorithms and protocols with Prof. Perrig!
Linear programming: a nice, generic tool

Efficient solvers exist, so your task is simply writing LPs

- CPLEX, Gurobi

Framing “word problems” is non-trivial

- Even correctness can be difficult to see
- Some formulations are faster than others
- Think of it as another programming language

Theory for LPs, efficient algorithms not in this course