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
Integer linear programming
Shortest path LP: what are the constraints?

Path $s \rightarrow t$ is connected:

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

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

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

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

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

Lucky in this case.
“Network planning” problems often hard
LPs versus ILPs

LPs are solvable in polynomial time

• Not always nice polynomials, e.g., $O(n^{12})$

• Targeted algorithms are often better

The general class of ILPs is NP-Hard

Often tractable for reasonable problem sizes

• CPLEX, Gurobi (just specify variables to be integers)

Never try to prove $X$ is hard by writing an ILP for $X$

• Many “easy” problems can be written as ILPs!
This is no good, since for $1,000$ fragments (typical experiment) this would need far longer than the age of the universe. Why?

- $1000! \approx 4 \cdot 10^{2567}$ permutations
  - source: Wolfram Alpha
- say we need $1000$ operations per permutation (summing overlaps)
- if we have a computer that does $1$ billion ($= 10^9$) operations/sec
- it can handle $1$ million ($10^6$) permutations per second
- So it will take $4 \cdot 10^{2567} / 10^6 = 4 \cdot 10^{2561}$ seconds

Age of the universe:
- about $14$ billion years $\approx 4 \cdot 10^{17}$ seconds

We can see that a million ($10^6$) or a billion ($10^9$) times faster computer wouldn't help much, either. Nor would faster handling of the permutations.

"I can’t find an efficient algorithm, I guess I’m just too dumb."

This is no good, since for 1,000 fragments (typical experiment) this would need far longer than the age of the universe. Why?

- $1000! \approx 4 \cdot 10^{2567}$ permutations
- source: Wolfram Alpha
- say we need 1000 operations per permutation (summing overlaps)
- if we have a computer that does 1 billion (= $10^9$) operations/sec
- it can handle 1 million ($10^6$) permutations per second
- So it will take $4 \cdot 10^{2567} / 10^6 = 4 \cdot 10^{2561}$ seconds
- Age of the universe: about 14 billion years $\approx 4 \cdot 10^{17}$ seconds
- We can see that a million ($10^6$) or a billion ($10^9$) times faster computer wouldn't help much, either. Nor would faster handling of the permutations.

(If you try to prove X is hard by writing an ILP for X)

“No, the fact is: you are too dumb.”

“I can't find an efficient algorithm, but neither can all these famous people.”

(If you try to prove X is hard by writing an ILP for X)
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
Exploiting randomness for networking
You run a popular network application

Requests hit “load balancer”

Any server can handle any request

How to distribute requests?

Goal:

Keep response time uniformly low

Uniform load at servers
Round-robin?

Sequential distribution
First request to S1
Second to S2
...
What if requests follow a pattern?
Round-robin?

Sequential distribution
First request to S1
Second to S2
...
What if requests follow a pattern?
Multiple load balancers?
Send to the least loaded server?

Query / track server load
Send request to least loaded server
Overhead of querying / tracking
Send to the least loaded server?

Query / track server load
Send request to least loaded server

Overhead of querying / tracking
Multiple load balancers?
Randomization to the rescue?

Pick a server uniformly at random

How often does imbalance occur?

“Balls into bins” problem
$m$ balls into $n$ bins

Each ball is put into a bin chosen uniformly at random
**m balls into n bins**

Q1: Probability that ball $i$ and $j$ land in the same bin?

Hint: what’s the probability $i$ and $j$ both go to bin $b$?

A: $1/n$
**Q2:** What is the expected number of collisions?

**Hint:** \( X_{ij} = 1 \) if i and j collide. \( E \left[ \sum X_{ij} \right] = \sum P \left[ X_{ij} = 1 \right] \).

**A:** \( \frac{1}{n} \binom{m}{2} \)
**m balls into n bins**

Q3: How many balls will cause this expectation to exceed 1?

A: Given any n, we can solve $mC_2 / n > 1$

$n = 365$ means $m = 28$ — “Birthday paradox”

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
</table>
$n$ balls into $n$ bins

Q4: What is the expected maximum load on any bin?

A: $O\left(\frac{\ln n}{\ln \ln n}\right)$

How do we improve this?
Power of two choices

Instead of picking one random bin, pick two.

Then query their load, and use the less loaded one.
Power of two choices

Instead of picking one random bin, pick two.

Then query their load, and use the less loaded one.

From $O\left(\frac{\ln n}{\ln \ln n}\right)$ to $O\left(\frac{\ln \ln n}{\ln 2}\right)$
Power of $k$ choices?

Instead of picking one random bin, pick $k$.

Then query their load, and use the least loaded one.

From $O\left(\frac{\ln n}{\ln \ln n}\right)$ to $O\left(\frac{\ln \ln n}{\ln 2^k}\right)$
Power of $k$ choices?

Only approximates here, without constants from the big-O
Sometimes fully random is not good

Your application is interactive

User requests randomized to servers

Interaction 1 with server a

Interaction 2 with server b

b: hi, how can I help you?

Randomize over sessions?
Hash functions to the rescue

server = Hash (user-id) % n

Hash function is assumed uniform
standard hash() are good enough
Preserves session continuity!
Applied to network path selection
Basis of common traffic balancing!

Applied to network path selection

“ECMP”

Equal-cost multiple path

\( h(\text{src-IP}, \text{dest-IP}, \text{src-port}, \text{dest-port}, \text{TCP}) \)

Caution: \( h() \) is a deterministic function
What if a server fails?

server = Hash (user-id) % (n - 1)

Will re-assign most sessions!

How to prevent this?

“Consistent hashing”
Consistent hashing

Hash the server IDs also to the same space!

Map each session to the “successor” server
Consistent hashing
Consistent hashing
Consistent hashing
Consistent hashing

Give servers multiple IDs to improve failover
Consistent hashing: two readings

• Section 1 here: http://theory.stanford.edu/~tim/s17/l/11.pdf [Roughgarden & Valiant]

• In action at Vimeo: https://medium.com/vimeo-engineering-blog/improving-load-balancing-with-a-new-consistent-hashing-algorithm-9f1bd75709ed
Membership testing & counting
I want to …

… check if an object is in my CDN cache

… check if a TLS certificate has been revoked

… check if I have seen this packet before (loop detection)

… count how many packets of a flow I have forwarded
Hash tables?

Actually store all the entries
A yes/no version of a hash table

How to handle collisions?
Ignore them!

False positives?
Reduce rate and live with it
OK for many applications!

Instead of one $h()$ use several
Bloom filters

$n$ elements to represent
$m$ bits of memory in table $T$

$k$ hash functions $h_1, h_2, \ldots, h_k$

hash range \{1, 2, \ldots, m\}
Bloom filters

Q: After n insertions, what is the probability that $T[i] = 0$?

A: 

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

$n$ elements to represent $m$ bits of memory in table $T$

$k$ hash functions $h_1, h_2, \ldots, h_k$

hash range $\{1, 2, \ldots, m\}$

[ Broder & Mitzenmacher: Network Applications of Bloom Filters ]
Bloom filters

Q: What’s the probability of a match being a false positive?
A: \( \left(1 - e^{-kn/m}\right)^k \) (approximately)
Bloom filters

Q: What value of $k$ minimizes this false positive probability?

A: $k \approx \frac{m}{n} \ln 2$

$n$ elements to represent
$m$ bits of memory in table $T$
$k$ hash functions $h_1, h_2, \ldots, h_k$
hash range $\{1, 2, \ldots, m\}$
Q: What is this minimized false positive probability?

A: \( 2^{-\frac{m}{n} \ln 2} \)
Bloom filters: false positive rate

Under 1% with 10 bits / element
Example application: cache filtering

Cache on second request

Avoid “one-hit wonders”

[Adapted from Broder & Mitzenmacher: Network Applications of Bloom Filters]

[Adapted from Maggs & Sitaraman: Algorithmic Nuggets in Content Delivery]
Example application: cache filtering

Cache on second request
Avoid “one-hit wonders”

Log objects requested in a bloom filter:

Check filter: Yes → cache it. No → don’t cache, but log.

Q: How to deal with bloom filter filling up over time?
Example application: cache filtering

![Graph showing byte hit rate over time with cache filtering enabled]

- **Byte hit rate (%)**
  - 70.00
  - 72.00
  - 74.00
  - 76.00
  - 78.00
  - 80.00
  - 82.00
  - 84.00
  - 86.00
  - 88.00
  - 90.00

- **Date**
  - 17-Feb
  - 27-Feb
  - 9-Mar
  - 19-Mar
  - 29-Mar
  - 8-Apr
  - 18-Apr
  - 28-Apr
  - 8-May
  - 18-May
  - 28-May

**Cache filtering enabled**
4.3 Empirical Benefits

Figure 5: On a typical CDN server cluster serving over one billion objects per day, each server has eight hard disks. The workload served by these servers is processed by Ming Dong Feng using a cluster of about 47 projects.

The astute reader may have observed that as more objects are served, the latency of accessing an object from disk decreases. As shown in Figure 7, the average disk latency drops from an average of 90 ms to 60 ms as more objects are served.

In Figure 6, we show the impact of Bloom filters on the byte hit rate. When cache filtering was turned on, the byte hit rate increased from around 74% to 83%. This is because Bloom filtering allows for the caching of objects that are likely to be accessed again, reducing the number of disk writes and increasing the byte hit rate.

Note that during the period when Bloom filtering was turned on, the cache-on-second-hit rule were turned on on March 14th and turned off on April 24th to create before, during, and after measurements. Thus, our figures show multiple measurements per day, enabling us to see both intra- and inter-day variations.

The graph shows the number of disk writes performed by the servers. The disk writes per second are plotted against the date. The x-axis represents the dates from 17-Feb to 28-May, and the y-axis represents the number of disk writes per second. The graph indicates that the number of disk writes per second decreases as the cache filtering enabled.

In Figure 8, the average disk latency drops from an average of 90 ms to 60 ms as more objects are served. This is because Bloom filtering allows for the caching of objects that are likely to be accessed again, reducing the number of disk writes and increasing the byte hit rate.

In addition, the number of disk writes per second is reduced significantly as objects that have been accessed before are cached. This decreases the disk write rate by nearly one-half. To be specific, from 10209 writes per second to 5738 writes per second, a decrease of 44%.

Most CDNs implement cache replacement algorithms such as LRU, which evict less popular objects such as one-hit-wonders when the cache is full. Filtering out the less popular objects such as one-hit-wonders reduces the number of disk writes and increases the byte hit rate.

As shown in Figure 9, the average disk latency drops from an average of 90 ms to 60 ms as more objects are served. This is because Bloom filtering allows for the caching of objects that are likely to be accessed again, reducing the number of disk writes and increasing the byte hit rate.

Note that during the period when Bloom filtering was turned on, the cache-on-second-hit rule were turned on on March 14th and turned off on April 24th to create before, during, and after measurements. Thus, our figures show multiple measurements per day, enabling us to see both intra- and inter-day variations.

The graph shows the number of disk writes performed by the servers. The disk writes per second are plotted against the date. The x-axis represents the dates from 17-Feb to 28-May, and the y-axis represents the number of disk writes per second. The graph indicates that the number of disk writes per second decreases as the cache filtering enabled.
Counting bloom filters

I also want deletions.
And maybe even counts?

Akamai: cache-summaries
Network logging

Addition (deletion) increments (decrements) \( k \) locations
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
Distributed decision-making
Does the weight move up or down?
Picking the best route

On the ‘x’ edges:
latency (load) = load

On other edges:
latency (load) = 1

Total load = 1

What’s the optimal traffic split?

Latency = 1.5 for everyone
Let’s add a “free” path

Let’s start with the 0.5-0.5 split

Move $\epsilon$ from each green to red path

Green: $1 + 0.5 + \epsilon$

Red: $(0.5 + \epsilon) + 0 + (0.5 + \epsilon)$

Red’s better by almost 50%

What if $\epsilon = 0.49999…$
Let’s add a “free” path

What is the selfish equilibrium?
(each driver chooses independently)

What is the optimal traffic split?

2 / 1.5 is the **price of anarchy**

Latency = 2 for everyone
A physical interpretation

- Latency $\propto$ Load
- Fixed latency

- Length $\propto$ Load
- Fixed length
How about in this case?

What is the optimal split now?
What is the selfish equilibrium?
… and the price of anarchy?
How about in this case?

Minimize $1(1 - \alpha) + \alpha^{100}\alpha$

What is the optimal split now?

What is the selfish equilibrium?

…and the price of anarchy?
Braess’s paradox: not merely theoretical

Seoul: removal of road gave a speedup
Stuttgart: new road caused a slowdown
New York: removal of road gave a speedup
Many roads in Boston, London could be removed
…
Is this the joint optimum?
Distributed decision-making: wrap up

Distributed settings are challenging

• Selfish players can cause large sub-optimalities
• Co-operative settings are still non-trivial

Centralized / distributed: long running issue

• Best choice depends on setting
  - The Internet is, for good reasons, distributed
  - Is the Web also distributed?
• Best choice may change over time!
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