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
Not always so lucky …

Shortest path LP: what are the constraints?

Path $s \rightarrow t$ is connected:

\[
\sum_{u \rightarrow v} x_{uv} - \sum_{v \rightarrow w} x_{vw} \in \{0, 1\}
\]

Also: $x_{uv} \in \{0, 1\}$ 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
- 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."

"You prove that the reconstruction problem is NP-complete, and you say: So it makes no sense to fire you and get another expert!"


[Image: Computers and Intractability, Garey and Johnson]
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 $4 \cdot 10^{2567} / 10^{9} = 4 \cdot 10^{2558}$ 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.

So you can go to your boss and say:

‘Bad idea, you may get fired!’


Or you could say:

‘Unfortunately, it is very hard to do impossibility proofs. . .’


You prove that the reconstruction problem is NP-complete, and you say:

‘So it makes no sense to fire you and get another expert!’


‘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)

[Image: Computers and Intractability, Garey and Johnson]
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 in put into a bin chosen uniformly at random
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: $\frac{1}{n}$
**$m$** balls into **$n$** bins

**Q2:** What is the expected number of collisions?

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

**A:** $1/n \binom{m}{2}$
Q3: How many balls will cause this expectation to exceed 1?

A: Given any $n$, we can solve $\frac{mC_2}{n} > 1$

$n = 365$ means $m = 28$ — “Birthday paradox”
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 $O\left(\frac{\ln \ln n}{\ln^2 n}\right)$
Power of $k$ choices?

Only big-$O$ functions are sketched here, without constants.
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

$$\text{server} = \text{Hash}(\text{user-id}) \mod n$$

Hash function is assumed uniform

standard hash() are good enough

Preserves session continuity!

Applied to network path selection

[Server image: RRZEicons via Wikimedia]
Basis of common traffic balancing!

Applied to network path selection

“ECMP”

Equal-cost multiple path

h(src-IP, dest-IP, src-port, dest-port, 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: two optional readings

• Section 1 here: http://theory.stanford.edu/~tim/s17/l11.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: \[ \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m} \]
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 $\{1, 2, \ldots, m\}$
Bloom filters

Q: What is this minimized false positive probability?

A: \( 2^{-\frac{m}{n} \ln 2} \)

\( n \) elements to represent

\( m \) bits of memory in table \( T \)

\( k \) hash functions \( h_1, h_2, \ldots, h_k \)

hash range \( \{1, 2, \ldots, m\} \)
Bloom filters: false positive rate

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

Cache on second request

Avoid “one-hit wonders”
Example application: cache filtering

Log objects requested in a bloom filter.

Check filter: Yes $\rightarrow$ cache it. No $\rightarrow$ don’t cache, but log.

Q: How to deal with bloom filter filling up over time?

Cache on second request

Avoid “one-hit wonders”}

Figure 1. An example of a Bloom filter. The filter begins as an array of all 0s. Each item in the set $x_i$ is hashed $k$ times, with each hash yielding a bit location; these bits are set to 1. To check if an element $y$ is in the set, hash it $k$ times and check the corresponding bits. The element $y_1$ cannot be in the set, since all of its bits are 0. The element $y_2$ is either in the set or the filter has yielded a false positive.

The probability of a false positive for an element not in the set, or the false positive rate, can be estimated in a straightforward fashion, given our assumption that hash functions are perfectly random. After all the elements of $S$ are hashed into the Bloom filter, the probability that a specific bit is still 0 is

$$p_I = 1 - \frac{1}{m}^{kn} \approx e^{-kn/m}.$$  

We let $p = e^{-kn/m}$, and note that $p$ is a convenient and very close ($O(1/m)$) approximation for $p_I$. Now, let $\rho$ be the proportion of 0 bits after all the $n$ elements are inserted in the table. The expected value for $\rho$ is of course $E(\rho) = p_I$. Conditioned on $\rho$,

1. Early work considering the performance of Bloom filters with practical hash functions was done by Ramakrishna [Ramakrishna 89]. The question of what hash function to use in practice remains an interesting open question; currently MD5 is a popular choice [Fan et al. 00].

[Adapted from Broder & Mitzenmacher: Network Applications of Bloom Filters]
Example application: cache filtering

![Graph showing byte hit rate (%) over time with cache filtering enabled highlighted](image)

### 4.3 Empirical Benefits

When checking if an object has been accessed in a Bloom filter, we describe a simple experiment conducted in a CDN setting. The experiment demonstrates how cache filtering, specifically the cache-on-second-hit rule, impacts disk I/Os and disk writes.

#### Cache-on-second-hit rule

- **Primary Filter**: An object is added to the primary filter if it has been accessed before.
- **Secondary Filter**: After an object is added to the primary filter, a secondary filter is created with all entries initialized to zero. The probability of false positives increases as more objects are added to the primary filter.
- **Eviction**: When the primary filter reaches a threshold for maximum probability of false positives, it is replaced by the secondary filter, and a new primary filter is initialized.
- **Threshold**: Secondary filters are forgotten when the primary filter is replaced. As the old secondary filter is forgotten, the primary filter, initialized to zero, becomes the primary.

#### Benefits:

- **Reduction in Disk I/Os**: Objects accessed only once leave more disk space for more popular objects. This is achieved by reducing the number of writes to disk from 44% to 25%.
- **Improved Byte Hit Rates**: The aggregate rate of disk writes drops from 18.25% to 11.62%.
- **Optimized Storage**: Not caching objects that are likely to be accessed only once or once-hit-wonders reduces the number of disk writes by nearly half, resulting in a 12.5% increase in byte hit rate.

#### Additional Observations:

- **Cache Usage**: Over three-quarters of the objects were accessed only once, and 40% of access patterns were one-hit-wonders.
- **Optimization Potential**: Implementing a more precise cache filtering technique could significantly reduce the number of writes to disk.

---

The astute reader may have observed that as more objects are served from the cache, the rate of disk writes decreases. This is evident in the graph showing the impact of Bloom filters on disk I/Os. The figure illustrates how the implementation of cache filtering rules can optimize CDN performance, leading to improved byte hit rates and reduced disk I/Os.
Example application: cache filtering

![Graph showing disk writes per second from February 17th to May 28th with cache filtering enabled. The y-axis represents disk writes per second, ranging from 0 to 14,000. The x-axis represents dates from February 17th to May 28th. The graph indicates a decrease in disk writes when cache filtering is enabled.]

- Disk writes per second
- Date
- Cache filtering enabled

Disk writes per second vs Date

- Graph showing disk writes per second from February 17th to May 28th with cache filtering enabled. The y-axis represents disk writes per second, ranging from 0 to 14,000. The x-axis represents dates from February 17th to May 28th. The graph indicates a decrease in disk writes when cache filtering is enabled.

- An arrow labeled "Cache filtering enabled" points to a period of decreased disk writes.
Counting bloom filters

I also want deletions.
And maybe even counts?

Akamai: cache summaries
Network logging

Addition (deletion) increments (decimals) $k$ 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:
\[ \text{latency (load)} = \text{load} \]

On other edges:
\[ \text{latency (load)} = 1 \]

Total load = 1

What’s the optimal traffic split?

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

Total load = 1
What is the optimal traffic split?
What is the selfish equilibrium?

(each driver chooses independently)

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

Latency = 2 for everyone
A physical interpretation
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?

What is the optimal split now?
What is the selfish equilibrium?
… and the price of anarchy?

Minimize \(1(1 - \alpha) + \alpha^{100}\alpha\)
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
...
Hotelling's Law

model speed

ticks: 0

Command Center

NetLogo Code

Model Info
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