

1 Online Resource Allocation

Consider the following problem. You are in charge of managing a network, viewed as a graph $G = (V, E)$ with m edges. G could be directed or undirected, or even have parallel edges; it doesn't matter. You then receive a series of requests for connections between various pairs: $(s_1, t_1), (s_2, t_2), \dots$. When the i th request comes in, you have to route it, that is, you have to give it a path from s_i to t_i . Your goal is to even out the load. In particular, what you want to do is keep the maximum congestion over the edges in G as small as possible. Of course the big problem is that you don't know what requests are going to come in the future.

What we will discuss today is a polynomial-time algorithm for this problem that for any sequence of requests guarantees a maximum congestion at most an $O(\log m)$ factor larger than the best possible routing in hindsight (had we been given the requests up front and also had exponential time to solve for the best routing, since even offline this problem is NP-hard). In fact, the best polynomial-time approximation known for this problem in the offline case is a factor $O(\log(m)/\log \log(m))$, so we're not doing much worse than that.

The algorithm actually will solve more general allocation problems too. For example, suppose we have a set E of m resources, and request i comes with a subset $S_i \subseteq E$ and needs one of the resources in S_i . If the resources are machines and the requests are jobs, this is often called the "restricted machines" job scheduling problem. Our goal here is to minimize the maximum load.

In fact, an even more general setting this can handle (which includes both of the above) is that request i comes with a collection S_i of *sets* of resources, and one must allocate to i one of the *sets* in S_i . Can you see how this can model the case of routing paths?

1.1 A lower bound

Before giving an algorithm, let's first see why we $O(\log m)$ is the best we can hope for.

Here is an argument: we'll assume the online algorithm is deterministic to keep this simpler. First, create k "bridges" (where k will be roughly $m/(\log m)$): this is just a matching of k edges directed north-to-south. We'll have one source node s that connects to the north end of all k bridges, and one sink node t that connects to the south end of all k bridges (so $k + 2k$ edges so far). Now we have a source node s_L that connects to the north end of the leftmost $k/2$ bridges and a sink node t_L that connects to the south end of the leftmost $k/2$ bridges. Similarly a source s_R and sink t_R that connect to north and south ends respectively of the rightmost $k/2$ bridges (so $k + 2k + 2k$ edges so far). Now a source s_{LL} and sink t_{LL} that connect to the leftmost $k/4$ of the leftmost $k/2$ bridges, and similarly s_{LR}, t_{LR} that connect to the rightmost $k/4$ of the leftmost $k/2$ bridges, s_{RL}, t_{RL} that connect to the leftmost $k/4$ of the rightmost $k/2$ bridges and s_{RR}, t_{RR} that connect to the rightmost $k/4$ of the rightmost $k/2$ bridges. We keep doing this $\lg k$ levels. In total we have $m = O(k \log k)$ edges, all directed south.

Now, we have k requests from s to t . At this point the average load on bridges is 1, and WLOG say the average load on the left half is \geq the average load on the right half. The adversary now gives

$k/2$ requests from s_L to t_L . So now the average load on the left half of the bridges is 2. Again, the adversary looks to see which half of that half is more highly loaded and gives $k/4$ requests there, bringing the average load there up to 3. This keeps going until finally at the end there is some bridge with load $\lg(k)$. But in hindsight you could have had *all* of the first k requests on the right $k/2$, then *all* of the next $k/2$ requests on whichever $k/4$ the adversary didn't use later, etc., giving a maximum load of 2.

1.2 Two algorithms that don't work

Before giving an algorithm that works, it will be helpful to consider two “strawman” algorithms that don't work. We'll then combine them to create an algorithm that does. Let's go back to thinking about routing paths in graphs, though everything will apply to the more general scenarios.

Strawman #1: route to greedily minimize maximum congestion.

Claim: if the graph is a circle, this can do as badly as $\Omega(n)$ times worst than the optimal routing. Can anyone see why? What if you get $(1, 2)$ twice, then $(2, 3)$ twice, then $(3, 4)$ twice, etc. The algorithm will use up the whole circle on the first two requests, then use it up again on the next two requests, and so on. What should it have done? It should have routed on shortest paths. This leads us to...

Strawman #2: ignore congestion so far, and just route on shortest paths.

Claim: you can make this do really badly too. Can anyone see a bad example? What if you see the same pair (s, t) over and over again, and there is one path of length 1 and many disjoint paths of length 2 between them?

1.3 The Awerbuch-Azar-Plotkin algorithm

The actual algorithm will combine aspects of each.

First, to simplify, let's assume final OPT cost (the maximum congestion, or maximum load) is known. If not, we will be able to guess and double, but let's just assume the value of the optimal cost is known to keep things simple. Here is the algorithm.

Algorithm 1 Awerbuch-Azar-Plotkin

1. Give each edge (each resource) a cost of \$1.
 2. When a request arrives, route it on the cheapest path (or in the more general setting, give the request its cheapest set).
 3. Multiply the cost of each resource used by the request that just arrived by a factor $r = (3/2)^{1/OPT}$, and goto 2.
-

This is a version of the “multiplicative weights” algorithm. Notice that if a resource has load ℓ then its current cost is $(3/2)^{\ell/OPT}$. Also, this is an algorithm you could use for pricing resources, allowing the requests to choose for themselves what sets to purchase.

Analysis: Let W be the total sum of all the costs, over all m resources. So, initially $W = m$. If we can show that at the end $W = O(m)$ then we are done. Can you see why? Because for any resource

e , its load $\ell(e)$ must satisfy $(3/2)^{\ell(e)/OPT} \leq W$ which means $\ell(e)/OPT \leq \log_{3/2} W = O(\log m)$.

One useful thing to notice: by choosing the cheapest path (cheapest set) we have chosen the path (set) that minimizes the *increase* in W out of all the options we have. Can you see why?

Let's use t to index time (i.e., requests). Define $\ell_t(e)$ to be the load on resource e after request t , so $\ell_0(e) = 0$ for all e . Define P_t to be the path (or set) chosen by the algorithm for request t and let P_t^* to be the path (or set) chosen by the optimal solution that we are comparing ourselves to. Finally, let's use T to denote the total number of requests.

The total *increase* in W from the start of the process to the end is $W_T - m$. Let's now write this as the sum of increases over the time steps $t = 1, 2, \dots, T$ and use the facts we know.

$$\begin{aligned}
W_T - m &= \sum_t \sum_{e \in P_t} (3/2)^{\ell_{t-1}(e)/OPT} \left((3/2)^{1/OPT} - 1 \right) \\
&\leq \sum_t \sum_{e \in P_t^*} (3/2)^{\ell_{t-1}(e)/OPT} \left((3/2)^{1/OPT} - 1 \right) \\
&\quad [\text{because } P_t \text{ was the path minimizing the increase in } W] \\
&= \sum_e \sum_{t: e \in P_t^*} (3/2)^{\ell_{t-1}(e)/OPT} \left((3/2)^{1/OPT} - 1 \right) \\
&\quad [\text{switching order of summations}] \\
&\leq \sum_e (3/2)^{\ell_T(e)/OPT} \sum_{t: e \in P_t^*} \left((3/2)^{1/OPT} - 1 \right). \\
&\quad [\text{since loads can only increase over time}]
\end{aligned}$$

Now, notice that the function $(3/2)^x - 1$ is convex, and equals 0 when $x = 0$ and equals $1/2$ when $x = 1$. Therefore, for $0 \leq x \leq 1$ we have $(3/2)^x - 1 \leq x/2$ by definition of convexity. So, we have for each resource e :

$$\sum_{t: e \in P_t^*} \left((3/2)^{1/OPT} - 1 \right) \leq \sum_{t: e \in P_t^*} \frac{1}{2 \cdot OPT}.$$

Next, notice that the number of times t such that $e \in P_t^*$ is exactly the load of e in the optimal solution, which in particular is at most OPT , since that represents the maximum load in the optimal solution over all e . So,

$$\sum_{t: e \in P_t^*} \frac{1}{2 \cdot OPT} \leq \frac{1}{2}.$$

Putting all this together we have:

$$\begin{aligned}
W_T - m &\leq \frac{1}{2} \sum_e (3/2)^{\ell_T(e)/OPT} \\
&= \frac{1}{2} W_T. \\
&\quad [\text{by definition of } W_T]
\end{aligned}$$

So, moving terms around we get $\frac{1}{2} W_T \leq m$, so $W_T \leq 2m$ and we are done!