Sublinear algorithms

6.046 Design and Analysis of Algorithms
MIT
Michael Kapralov
15 May 2014

Algorithms in $P$=runtime polynomial in size of input

Ideally, linear time (polynomial of degree 1)

Example: sorting (time $n \log n$)

Sublinear algorithms:

Do we need to read the entire input?

Can we make runtime polynomial of degree $< 1$?

Sometimes yes! (today’s lecture)

Sublinear algorithms=algorithms that use resources asymptotically smaller than the size of their input

Examples of resources we want to minimize:
Sublinear algorithms=algorithms that use resources asymptotically smaller than the size of their input

Examples of resources we want to minimize:
- time (do not even read the whole input!)
- space (scan through the whole input, but maintain small state)
- communication (distributed algorithms)

Streaming (sublinear space) in previous lecture.

Today: sublinear time algorithms for finding matchings

Matching

Let $G = (P, Q, E)$ be a bipartite graph $|P| = |Q| = n$, $|E| = m$

- $M \subseteq E$ is a matching if no two edges in $M$ share an endpoint
Matching

- Let $G = (P, Q, E)$ be a bipartite graph
  - $|P| = |Q| = n$, $|E| = m$
- $M \subseteq E$ is a matching if no two edges in $M$ share an endpoint

Maximum matching problem

Given a graph $G$, find a matching $M \subseteq E$ of maximum possible cardinality

Maximum matching problem

Can use Hopcroft-Karp to obtain $O(m\sqrt{n})$ runtime

Today: special case of regular bipartite graphs, $O(n\log n)$ runtime!
Maximum matching problem

Can use Hopcroft-Karp to obtain $O(m \sqrt{n})$ runtime

Today: special case of regular bipartite graphs, $O(n \log n)$ runtime!

Applications to
  - task scheduling

Maximum matching problem

Can use Hopcroft-Karp to obtain $O(m \sqrt{n})$ runtime

Today: special case of regular bipartite graphs, $O(n \log n)$ runtime!

Applications to
  - task scheduling (getting coffee at the Forbes cafe)

Background

Let $G = (P, Q, E), |P| = |Q| = n, |E| = m$ be a bipartite graph

- $G$ is $d$-regular if the degree of every vertex is equal to $d$
- Number of edges is $m = nd$

Regular bipartite graphs

- Finding matchings in $O(n \log n)$ expected time
- Edge coloring

Background

- Let $G = (P, Q, E), |P| = |Q| = n, |E| = m$ be a bipartite graph
- $G$ is $d$-regular if the degree of every vertex is equal to $d$
- Number of edges is $m = nd$
- a matching $M$ is perfect if $|M| = n$
- Hall's marriage theorem: every $d$-regular bipartite graph has a perfect matching
Let $G = (P, Q, E), |P| = |Q| = n, |E| = m$ be a bipartite graph

- $G$ is $d$-regular if the degree of every vertex is equal to $d$
- a matching $M$ is perfect if $|M| = n$
- Hall's marriage theorem: every $d$-regular bipartite graph has a perfect matching algorithmic proof!

Complex structures when $d \geq 3$
Sublinear algorithm

Assume that graph is given in adjacency array representation.

Theorem
There exists a randomized $O(n \log n)$ expected time algorithm for finding a perfect matching in a $d$-regular bipartite graph.

Also $O(n \log n)$ time with high probability.

Need $\Omega(n)$ to output matching! Within $O(\log n)$ of output complexity.

Given a partial matching $M$, how can one increase its size?

An augmenting path wrt a partial matching $M$
- starts at an unmatched node on the left
- alternates between unmatched and matched edges
- ends at an unmatched node on the right

Given a partial matching $M$, how can one increase its size?

An augmenting path wrt a partial matching $M$
- starts at an unmatched node on the left
- alternates between unmatched and matched edges
- ends at an unmatched node on the right
Alternating random walk
Natural randomization: alternate between taking
- a uniformly random unmatched edge from $P$ to $Q$
- the matched edge from $Q$ to $P$, as before

An augmenting path can be obtained by removing possible
loops

\[
\begin{align*}
P & \xrightarrow{\text{uniformly random}} Q \\
Q & \xrightarrow{\text{matched}} P
\end{align*}
\]
Alternating random walk

Natural randomization: alternate between taking
- a uniformly random unmatched edge from \(P\) to \(Q\)
- the matched edge from \(Q\) to \(P\), as before

An augmenting path can be obtained by removing possible loops

Algorithm

Let \(M_0 := \emptyset\). For each \(k = 1, \ldots, n\)
1. Run the alternating random walk wrt \(M_{k-1}\) until it hits an unmatched vertex in \(Q\)
2. Augment along the path obtained from the walk, set \(M_k\) equal to the new matching

Theorem

The algorithm above finds a perfect matching in \(O(n \log n)\) expected time.
Define a graph $H$ such that alternating random walk on $G$ wrt $M$ is a simple random walk on $H$.

Connect source $s$ and sink $t$ with $d$ edges to unmatched nodes.

Contract each pair $(u,v) \in M$ into supernode.

The algorithm can be restated as follows.

Let $M_0 := \emptyset$. For each $k = 1, \ldots, n$
1. Run the simple random walk from $s$ in the matching graph $H(G, M_{k-1})$ until it hits $t$
2. Augment along the path obtained from the walk, set $M_k$ equal to the new matching.

Main lemma

Lemma
Let $M$ be a matching that leaves $2k$ nodes unmatched. The expected time until the simple random walk in the matching graph $H$ starting from $s$ hits $t$ is at most $4 + 2n/k$.

- reduce to bounding return time
- bound stationary distribution of the random walk in $H$.
Main lemma

**Lemma**

Let $M$ be a matching that leaves $2k$ nodes unmatched. The expected time until the simple random walk in the matching graph $H$ starting from $s$ hits $t$ is at most $4 + 2n/k$.

- reduce to bounding return time
- bound stationary distribution of the random walk in $H$

Throwing darts randomly:

- board area is $n$
- target area is $k$

How many throws in expectation?

```
39 / 89
```
Augmenting a matching that leaves $k$ nodes unmatched takes $4 + 2n/k$ expected time.

Target shrinks as matching gets larger!

The expected runtime is at most

$$\sum_{k=1}^{n} \left( 4 + \frac{2n}{n+1-k} \right) = 2n(2+H_n) = 2n(2+\ln n),$$

where $H_n$ is the $n$-th harmonic number.

Constants are small!
Main lemma

Lemma
Let M be a matching that leaves $2k$ nodes unmatched. The expected time until the simple random walk in the matching graph $H$ starting from $s$ hits $t$ is at most $4 + 2n/k$.

Proof of main lemma

Identify $s$ and $t$ in the matching graph $H(G, M_k)$. The resulting graph $H^*(G, M_k)$ is a balanced directed graph.
Proof of main lemma

The resulting graph $H^*(G, M_k)$ is a balanced directed graph.

Fundamental Theorem of Markov Chains

Let $G = (V, E)$ be a

- strongly connected directed graph (i.e. there is a path from every vertex to every vertex)
- with a self-loop on every vertex (to avoid periodicities)

Then there exists a unique stationary distribution.

Lemma

If we add self-loops to every node, then the stationary distribution exists and is unique.

Proof.

There exists a (directed) path from every node to every node, as a consequence of the balanced property.

Proof of main lemma

Hence, for a vertex $u \in V_{H^*}$ the stationary distribution of the simple random walk is

$$\pi(u) = \frac{\text{out}(u)}{\sum_{u' \in V_{H^*}} \text{out}(u')}.$$  

Crucial observation:

expected time for a random walk started at $s$ to reach $t$ in $H(G, M_k)$ equals expected return time to $s$ in $H^*$, i.e. $1/\pi(s)$.

Proof of main lemma

The degree of $s$ in $H^*$ is $dk + 1$ (due to self-loop!). Hence, the expected number of steps before the random walk reaches $t$ is at most

$$\frac{1}{\pi(s)} = \sum_{j \in V(H)} \frac{\text{out}(j)}{dk + 1} \leq \frac{(n-k)d + 2k(d+1) + dk + 1}{dk + 1} \leq 4 + \frac{2n}{k}.$$  

Runtime of the algorithm

Finding an augmenting path with respect to a matching that leaves $k$ nodes unmatched takes at most $2 + 4n/k$ expected time.

So, the expected runtime is at most

$$\sum_{k=1}^{n} 4 + \frac{2n}{n+1-k} = 2n(2 + H_n) = O(n \log n),$$  

where $H_n$ is the $n$-th harmonic number.
Runtime of the algorithm

Finding an augmenting path with respect to a matching that leaves $k$ nodes unmatched takes at most $2 + 4n/k$ expected time.

So, the expected runtime is at most

$$\sum_{k=1}^{n} 4 + \frac{2n}{n+1-k} = 2n(2 + H_n) = O(n \log n),$$

where $H_n$ is the $n$-th harmonic number.

The high probability result can be obtained via truncation of the random walk and standard concentration inequalities.

Any regular bipartite graph can be decomposed as a union of edge disjoint perfect matchings!
Any regular bipartite graph can be decomposed as a union of edge disjoint perfect matchings!

Edge coloring

- Regular bipartite graphs
- Finding matchings in $O(n\log n)$ expected time
- Edge coloring

Given graph $G$, assign colors to edges so that no vertex has two incident edges of the same color.
Edge coloring

Given graph $G$, assign colors to edges so that no vertex has two incident edges of the same color.

Equivalently, partition edge set into as few matchings as possible.

Can always color using $d_{\text{max}} + 1$ colors. NP-hard to distinguish between $d_{\text{max}} + 1$ and $d_{\text{max}}$ in general graphs.

---

Edge coloring bipartite multigraphs

We will prove that $d_{\text{max}}$ colors suffice in bipartite graphs!

And give an efficient algorithm for coloring.

---

Why is edge coloring useful?
One machine can perform one job in any given time slot. How many time slots do we need?

Color edges in $d_{\text{max}}$ colors so that every vertex has distinct colors adjacent to it.

One can get a very simple $O(m \log n)$ algorithm for edge coloring:

1. Transform $G$ into a bipartite regular graph with $n'$ vertices and $m' = O(m)$ edges.
2. Take out matchings one by one, taking $O(n' \log n')$ time per matching.
3. Total time $O(d_{\text{max}} n' \log n') = O(m \log n)$.

Transform $G$ into a bipartite regular graph:

1. merge nodes whose degree is at most $d_{\text{max}}/2$, so that at most one such node is left on each side
2. add vertices if needed
3. add $O(m)$ edges to give every vertex degree $d_{\text{max}}$. 
Edge coloring bipartite multigraphs

- Diagram 1
- Diagram 2
- Diagram 3
- Diagram 4
- Diagram 5
- Diagram 6
Transform $G$ into a bipartite regular graph:
1. merge nodes whose degree is at most $d_{\text{max}}/2$, so that at most one such node is left on each side
2. add vertices if needed
3. add $O(m)$ edges to give every vertex degree $d_{\text{max}}$.

Why is number of added edges $O(m)$?

One can get a very simple $O(m \log n)$ algorithm for edge coloring:
1. Transform $G$ into a bipartite regular graph with $n'$ vertices and $m' = O(m)$ edges.
2. Take out matchings one by one, taking $O(n' \log n')$ time per matching.
3. Total time $O(d_{\text{max}} n' \log n') = O(m \log n)$. 

One can get a very simple $O(m \log n)$ algorithm for edge coloring:
1. Transform $G$ into a bipartite regular graph with $n'$ vertices and $m' = O(m)$ edges.
2. Take out matchings one by one, taking $O(n' \log n')$ time per matching.
3. Total time $O(d_{\text{max}} n' \log n') = O(m \log n)$. 

Edge coloring bipartite multigraphs

One can get a very simple $O(m \log n)$ algorithm for edge coloring:

1. Transform $G$ into a bipartite regular graph with $n'$ vertices and $m' = O(m)$ edges.
2. Take out matchings one by one, taking $O(n' \log n')$ time per matching.
3. Total time $O(d_{\max} n' \log n') = O(m \log n)$.

How do we implement sampling?