1 Introduction

1.1 Min-cut algorithm

We consider a graph G that is connected, undirected multigraph on n vertices. Multigraph means there can be multiple edges between two vertices. A cut is a set of edges, whose removal splits G into two or more components. A min-cut is a cut of minimum cardinality.

The randomized min-cut algorithm works as follows (show example):

1. Pick an edge uniformly at random and merge end-points (a contraction).
2. Remove self-loops if any are introduced
3. Repeat 1-2 until there are only 2 vertices left
4. Return candidate min-cut

Important: contractions does not reduce min-cut size of G, since every cut in an intermediate step is also a step in the original graph.

However, does not always find a min-cut. We will now analyse the probability of finding such a cut:

We define $\Gamma(v)$ to be the neighborhood of v and $d(v)$ its degree. Let C be a min-cut and k be the cardinality. G clearly has $kn/2$ edges (each vertice has k edges and remove duplicates between two pairs). We bound the prob that no edge of C is contracted. We call the event of NOT picking and edge in C at the ith step, $\mathcal{E}_i$ for $1 \leq i \leq n-2$.

Probability of picking the edge in first step is $k/(kn/2) = 2/n$, ie $P[\mathcal{E}_1] \geq 1 - 2/n$. Assuming $\mathcal{E}_1$, the prob in the second step is then $P[\mathcal{E}_1 | \mathcal{E}_2] = 1 - 2/(n-1)$. At step i we therefore have $P[\mathcal{E}_i | \cup_{j=1}^{i-1} \mathcal{E}_j] \geq 1 - 2/(n-i+1)$. The total prob that no edge in C is picked us then

$$P[\cup_{i=1}^{n-2} \mathcal{E}_i] \geq \prod_{i=1}^{n-2} \left(1 - \frac{2}{n-i+1}\right) = \frac{2}{n(n-1)}$$

ie prob is larger than $2/n^2$. If we run it $n^2/2$ times, the prob that a min-cut is not found is at most

$$\left(1 - \frac{2}{n^2}\right)^{n^2/2} < 1/e$$

note, this the the prob of failure after running $n^2/2$ times.

1.2 Las Vegas and Monte Carlo

Las Vegas algorithms are randomized algorithms that always give the correct answer but can differ in runtime from one run to another. RandQS is such an exampple.

Monte Carlo algorithms are randomized algorithms where we aren't guaranteed a correct answer, but we can bound the probability of failure. We can make this failure rate arbitrarily small by running the algorithm several times. The Min-Cut algorithm is an example of this.

For Y/N MC algorithms, we have those with one-sided error and two-sided error.

One-sided: For atleast one of Y or N, there must be a zero prob for error. (e.g. Miller-Rabin primality test, zero prob for failure when outputting *composite*)

Two-sided: Non-zero prob that it errors when outputting Y or N.

1.3 Binary planar partition

We define the binary planar partition, of a set of size n non-intersecting linesegments $S = \{s_1, s_2, \ldots, s_n\}$, as binary tree where each internal node v in the tree is a region $r(v)$ of the plane. The root node of the tree corresponds to the entire plane. Each internal node is then intersected by the line $l(v)$ into two new regions. Given S we then want to find a binary planar partition s.t. each region atmost contains one line segment s_i. Note that a line segment can be split into multiple if intersected by a line $l(v)$ s.t. each part lies within a different region.

Within the field of computer graphics, we can use binary planar partitions to solve the problem of *hidden line elimination.* For some given direction of viewing, we can use the idea of **`painter's algorithm`** which consists of drawing the objects that are furthest behind first and then progressively drawing objects that are nearer. With a BPP, we can recursively traverse the tree as follows. At the root, decide which side of the partition line L_1 is further away than the other and the render the objects in that subtree recursively, after that render the other side recursively.

The time it takes to render everything then obviously depends on the size of the tree, which means we want it to be as small as possible. Since line segments can be split into multiple segments, we cannot be sure of a partition of size $\mathcal{O}(n)$.

The algorithm RandAuto describes a fast randomized approach to constructing a BPP of expected size $\mathcal{O}(n \log n)$. Consider a line segment s. Let $l(s)$ then be the line obtained by extending s infinitely on both sides. From the set S, we can then construct an *autopartition*, which is a BPP, by only using intersecting lines from the set $\{l(s_1), l(s_2), \dots, l(s_n)\}$.

Given the set S, RandAuto then works by first picking a permutation π of $\{1, 2, \dots, n\}$ uniformly at random. Then while any regions contains more than one line segment, we partition the plane with the line $l(s_i)$ where i is the first in the ordering π s.t. s_i cuts that region.

We will now prove that the expected size of the autopartition is $\mathcal{O}(n \log n)$:

Let u and v be two line segments, where we define $index(u, v) = i$ if $l(u)$ intersects $i - 1$ segments before intersecting v. We have $index(u, v) = \infty$ if $l(u)$ doesn't hit at all. We can have $index(u, v) = index(u, w) = i$ since we can extend u in both directions.

$u \dashv v$ is the event that $index(u, v) \neq \infty$, ie. $index(u, v) = i$ for some i. Then $u_1, u_2, \dots, u_{i-1}$ are the segments that are cut before $l(u)$ hits v. $u \dashv v$ only happens if u occurs before any of $\{u_1, \dots, u_{i-1}, v\}$ in the randomly permutation π, which has prob $1/(i + 1)$.

We let $C_{u,v} = [u \dashv v]$ then clearly, $E[C_{u,v}] = P[u \dashv v] \leq 1/(index(u, v) + 1)$. The expected size of the autopartition P_π is then n plus expected intersections due to cuts. This is

$$\begin{aligned}
n + E\left[\sum u \sum_v C_{u,v}\right] &= n + \sum u \sum_v P\left[u \dashv v\right] \\
&\leq n + \sum u \sum_{v \neq u} \frac{1}{index(u, v) + 1} \\
&\leq n + \sum u \sum_{i=1}^{n-1} \frac{2}{i + 1} \\
&\leq n + 2nH_n \\
&= \mathcal{O}(n \log n)
\end{aligned}$$