CASE STUDY

Monte Carlo Algorithm for Min Cut

Gambles with correctness. Doesn’t gamble with run-time.

Cut Problems

Max Cut Problem (Ryan O’Donnell’s favorite problem):
Given a connected graph $G = (V, E)$, color the vertices red and blue so that the number of edges with two colors (e = {u, v}) is maximized.
**Cut Problems**

**Max Cut Problem** (Ryan O’Donnell’s favorite problem):
Given a connected graph $G = (V, E)$, find a non-empty subset $S \subset V$ such that number of edges from $S$ to $V - S$ is maximized.

Size of the cut = # edges from $S$ to $V - S$.

Max Cut Problem is **NP-hard**!

**Randomized Approximation for Max Cut**

**Min Cut Problem** (my favorite problem):
Given a connected graph $G = (V, E)$, find a non-empty subset $S \subset V$ such that number of edges from $S$ to $V - S$ is minimized.

Size of the cut = # edges from $S$ to $V - S$.

(how many possible “cuts” are there!)
Randomized Algorithm for Min Cut
(contraction algorithm)

Select an edge randomly:
\{b,d\} selected
Contract that edge.

Size of min-cut: 2

Select an edge randomly:
\{a,d\} selected
Contract that edge.

Size of min-cut: 2
Contraction algorithm for min cut

Example run 1

Select an edge randomly:
\{c, a\} selected
Contract that edge.  (delete self loops)

Size of min-cut: 2

Example run 1

When two vertices remain, you have your cut:
\( S = \{a, b, c, d\} \quad V\setminus S = \{e\} \quad \text{size: 2} \)

Contraction algorithm for min cut

\[
G = G_0 \overset{\text{contract}}{\rightarrow} G_1 \overset{\text{contract}}{\rightarrow} G_2 \overset{\text{contract}}{\rightarrow} \cdots \overset{\text{contract}}{\rightarrow} G_{n-2} \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
\text{\( n \) vertices} \quad \text{\( n \) vertices} \quad \text{\( n \) vertices} \quad \text{\( n \) vertices}
\]

\( n - 2 \) iterations
**Contraction algorithm for min cut**

**Observation:**
For any $i$: A cut in $G_i$ of size $k$ corresponds exactly to a cut in $G$ of size $k$.

**Theorem:**
Let $G = (V, E)$ be a graph with $n$ vertices. The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$.

Should we be impressed?
- The algorithm runs in polynomial time.
- There are exponentially many cuts. ($\approx 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)

**Proof of Theorem**
Let $k$ be the size of a minimum cut.
Which of the following are true (can select more than one):

For $G = G_0$, $k \leq \min_v \deg_G(v)$ ($\forall v, \ k \leq \deg_G(v)$)

For $G = G_0$, $k \geq \min_v \deg_G(v)$

For every $G_i$, $k \leq \min_v \deg_{G_i}(v)$ ($\forall v, \ k \leq \deg_{G_i}(v)$)

For every $G_i$, $k \geq \min_v \deg_{G_i}(v)$

---

**Proof of theorem**

Fix some minimum cut.

$|F| = k$

$|V| = n$

$|E| = m$

**Will show** $\Pr[\text{algorithm outputs } F'] \geq 1/n^2$

**(Note)** $\Pr[\text{success}] \geq \Pr[\text{algorithm outputs } F']$
Proof of theorem

Theorem:
Let $G = (V, E)$ be a graph with $n$ vertices. The probability that the contraction algorithm will output a min-cut is $\geq 1/n^2$.

Should we be impressed?
- The algorithm runs in polynomial time.
- There are exponentially many cuts, ($\approx 2^n$)
- There is a way to boost the probability of success to $1 - \frac{1}{e^n}$ (and still remain in polynomial time)
**Boosting Phase**
(and the world's greatest approximation!)

**Boosting phase**

Run the algorithm \( t \) times using fresh random bits.

\[
\begin{array}{cccccc}
G & G & G & \cdots & G \\
\text{Contraction Algorithm} & \text{Contraction Algorithm} & \text{Contraction Algorithm} & \cdots & \text{Contraction Algorithm} \\
F_1 & F_2 & F_3 & \cdots & F_t \\
\end{array}
\]

Output the minimum among \( F_i \)’s.

larger \( t \) \( \implies \) better success probability

What is the relation between \( t \) and success probability?

Let \( A_i = \) “in the \( i \)’th repetition, we \textbf{don’t} find a min cut.”

\[
\Pr[\text{error}] = \Pr[\text{don’t find a min cut}]
\]

\[
= \Pr[A_1 \cap A_2 \cap \cdots \cap A_t]
\]

\[
= \Pr[A_1] \Pr[A_2] \cdots \Pr[A_t]
\]

\[
= \Pr[A_1]^t \leq \left( 1 - \frac{1}{n^2} \right)^t
\]
Boosting phase

Pr[error] ≤ \left(1 - \frac{1}{n^2}\right)^t

World's most useful inequality: \forall x \in \mathbb{R} : 1 + x ≤ e^x

Let \quad x = -1/n^2

Pr[error] ≤ (1 + x)^t ≤ (e^x)^t = e^{xt} = e^{-t/n^2}

\quad t = n^3 \implies \quad \text{Pr}[error] ≤ e^{-n^3/n^2} = 1/e^n \implies \quad \text{Pr}[success] ≥ 1 - \frac{1}{e^n}

Conclusion for min cut

We have a polynomial-time algorithm that solves the min cut problem with probability \quad 1 - 1/e^n.

\text{Theoretically, not equal to 1. Practically, equal to 1.}
**Important Note**

Boosting is not specific to Min-cut algorithm.

We can boost the success probability of Monte Carlo algorithms via repeated trials.

---

**Final remarks**

Randomness adds an interesting dimension to computation.

Randomized algorithms can be faster and more elegant than their deterministic counterparts.

There are some interesting problems for which:
- there is a poly-time randomized algorithm,
- we can’t find a poly-time deterministic algorithm.