Analysis of Max Cut Approximation Algorithms

1 Problem Definition

Given undirected graph $G = (V, E)$ with $|V| = n$, $|E| = m$, find partition $(S, V \setminus S)$ maximizing:

$$\text{cut}(S) = |\{(u, v) \in E : u \in S, v \in V \setminus S\}|$$

2 Algorithm 1: Randomized Algorithm

2.1 Implementation

```
std::vector<bool> partition(vertices, false);
std::srand(std::time(nullptr));
for (int i = 0; i < vertices; ++i) {
    partition[i] = std::rand() % 2;
}
```

2.2 Analysis

For each edge $(u, v) \in E$:

$$\mathbb{P}[\text{edge cut}] = \mathbb{P}[x_u \neq x_v] = \frac{1}{2}$$

By linearity of expectation:

$$\mathbb{E}[\text{cut size}] = \sum_{(u,v)\in E} \frac{1}{2} = \frac{|E|}{2}$$

Since $OPT \leq |E|$:

$$\mathbb{E}[\text{cut size}] \geq \frac{1}{2} OPT$$

Time: $O(n+m)$.

3 Algorithm 2: Derandomized Algorithm

3.1 Implementation

```
for (int v = 0; v < vertices; ++v) {
    int cutIfTrue = 0, cutIfFalse = 0;
    for (int neighbor : adjacency[v]) {
        if (neighbor < v) {
            if (partition[neighbor]) cutIfTrue++;
            else cutIfFalse++;
        }
    }
    partition[v] = (cutIfTrue >= cutIfFalse);
}
```

3.2 Analysis

Let X_i = assignment of vertex i. Define:

$$F_i(x_1, \ldots, x_i) = \mathbb{E}[\text{cut} | X_1 = x_1, \ldots, X_i = x_i]$$

Algorithm maintains invariant:

$$F_i(x_1, \ldots, x_i) \geq \frac{OPT}{2}$$

At step i, choose x_i maximizing conditional expectation:

$$x_i = \arg\max_{b \in \{0,1\}} F_i(x_1, \ldots, x_{i-1}, b)$$

Since $\max\{a, b\} \geq \frac{a+b}{2}$:

$$F_{i-1}(x_1, \ldots, x_{i-1}) = \frac{F_i(x_1, \ldots, x_{i-1}, 0) + F_i(x_1, \ldots, x_{i-1}, 1)}{2}$$

Thus chosen x_i satisfies $F_i \geq F_{i-1} \geq \frac{OPT}{2}$.

Time: $O(n+m)$.

4 Algorithm 3: Greedy Algorithm

4.1 Implementation

```
std::priority_queue<int, std::vector<int>, decltype(cmp)> pq(
    cmp);
while (!pq.empty()) {
    int v = pq.top(); pq.pop();
    if (inSet[v]) continue;
    partition[v] = true;
    inSet[v] = true;
    for (int neighbor : adjacency[v]) {
        if (!inSet[neighbor]) {
            difference[neighbor]--;
            pq.push(neighbor);
        }
    }
}
```

4.2 Analysis

Initialize difference$[v] = \deg(v)$. When adding vertex v to S:

- For each neighbor $u \notin S$: edge (v, u) becomes cut
- For each neighbor $u \in S$: edge (v, u) becomes uncut

Net gain when adding v:

$$\Delta = |\{u \notin S : (v, u) \in E\}| - |\{u \in S : (v, u) \in E\}|$$

Algorithm always chooses vertex with $\Delta \geq 0$.
Claim: Final cut has at least $|E|/2$ edges.
Proof: Each edge contributes:

- $+1$ when first endpoint added (if other not in S)
- -1 when second endpoint added (if first in S)
- 0 if both endpoints never added to S

Since $\Delta \geq 0$ at each step, total $\sum \Delta \geq 0$, so cut edges $\geq$ uncut edges.
Time: $O(m \log n)$ with priority queue.

5 Algorithm 4: Multi-start Random Sampling

5.1 Implementation

```
for (int iter = 0; iter < iterations; ++iter) {
    std::vector<double> randomVector(vertices);
    for (int i = 0; i < vertices; ++i) {
        randomVector[i] = dist(gen);
    }
    for (int i = 0; i < vertices; ++i) {
        partition[i] = (randomVector[i] >= 0);
    }
    // Keep best over iterations
}
```

5.2 Analysis

Run Algorithm 1 T times, take maximum cut size.

Let X_i = cut size in iteration i. $\mathbb{E}[X_i] \geq \frac{OPT}{2}$.

Let $X_{\max} = \max\{X_1, \ldots, X_T\}$. For any $\epsilon > 0$:

$$\mathbb{P}[X_i < (1-\epsilon)\frac{OPT}{2}] \leq \text{some bound}$$

By independence:

$$\mathbb{P}[X_{\max} < (1-\epsilon)\frac{OPT}{2}] \leq \left(\mathbb{P}[X_1 < (1-\epsilon)\frac{OPT}{2}]\right)^T$$

Thus probability of poor solution decreases exponentially with T.

Time: $O(T(n+m))$.