## 5.1 Local Search

Now we're going to talk about an algorithmic technique known as *local search*. A local search algorithm operates in a way which should be familiar to those of you with some optimization background: we start with some feasible solution (which may be very far from optimal), and then we make "local" improvements that maintain feasibility until we get stuck at some "local optimum". We then try to show that wherever we get stuck, that solution must be pretty close to optimal.

Local search is a very natural algorithmic technique, and is often combined (in practice) as a post-processing step with other algorithms (e.g., use a greedy algorithm to build up something, then use local search to try to make it better). As a theoretical technique, it has been useful for a variety of problems, but is traditionally difficult to analyze. There has been some interesting recent progress, though, so there is some renewed interest. The next two lectures we're going to study some of the most famous examples of where we can prove it works well.

## 5.2 Max-Cut

- Input: a graph $G = (V, E)$, where $V$ is the set of vertices and $E$ is the set of edges.
- Feasible solution: a set $S \subseteq V$. Here $S$ is also called a cut of $G$.
- Objective: Maximize $|E(S, \bar{S})|$. Here $E(S, \bar{S}) = \{\{u, v\} : \{u, v\} \in E, u \in S, v \in \bar{S}\}$.

Unlike Min-Cut, this problem is NP-hard. Consider the following natural local search algorithm.

**Algorithm 1** Local Search Algorithm for Max-Cut

**Input**: A graph $G = (V, E)$
**Output**: A set $S \subseteq V$

```
Initialize S arbitrarily
while ∃u ∈ V with more edges to the same side than across do
    move u to the other side
end while
return S
```

This algorithm is actually relatively simple to analyze.

**Theorem 5.2.1** *The Local Search algorithm for Max-Cut runs in polynomial time.*

**Proof:** Clearly each iteration takes polynomial time, so the question is how many iterations there are (this is typical in local search algorithms). We can bound this through some very simple observations.

1. Initially $|E(S, \bar{S})| \geq 0$,
2. In every iteration, $|E(S, \bar{S})|$ increases by at least 1, and
3. $|E(S, \bar{S})|$ is always at most $m \leq n^2$.

These facts clearly imply that the number of iterations is at most $m \leq n^2$, and thus the total running time is polynomial. ■

To prove the next theorem about the approximation ratio, we will do something which is very common when analyzing local search algorithms. We will not reason directly about the solution $S$ which the algorithm outputs, but will instead reason about an *arbitrary* local optimum. Since the algorithm outputs a local optimum, any bound we can prove which holds for all local optimum will hold for the output of the algorithm.

**Theorem 5.2.2** *The local search algorithm is a 2-approximation.*

**Proof:** We say that $S$ is a *local optimum* if there is no improving step: for all $u \in V$, there are more $\{u, v\}$ edges across the cut than connecting the same side. Note that the algorithm outputs a local optimum by definition.

Suppose $S$ is a local optimum. Let $d_{across}(u)$ denote the number of edges incident on $u$ that cross the cut. The since $S$ is a local optimum, we know that

$$|E(S, \bar{S})| = \frac{1}{2} \sum_{u \in V} (d_{across}(u)) \geq \frac{1}{2} \sum_{u \in V} \frac{1}{2} d(u) = \frac{1}{4} \sum_{u \in V} d(u) = \frac{1}{4} \cdot 2m = m/2,$$

where $m = |E|$. We know $OPT \leq m$, and hence the algorithm is a 2-approximation. ■

Note: this is *not* the best-known approximation for Max-Cut. When we talk about semidefinite relaxations, we'll see the best-known algorithm. But this is a good warmup for and example of local search.

## 5.3 Weighted Max-Cut

Weighted Max Cut is the same as Max-Cut, but there is a weight $w_e$ for each edge $e$ and the objective is the max weight cut. Formally

**Input**: Graph $G = (V, E)$, weight function $w : E \to \mathbb{N}$.

**Feasible solution**: $S \subseteq V$.

**Objective**: $\max \sum_{e \in E(S, \bar{S})} w(e)$.

Let $W = \sum_{e \in E} w(e)$. Notice that if we try to use the same algorithm ("vanilla" local search) and analyze the running time as we did before, we get that it is $O(W)$. But $W$ is not necessarily polynomial in the size of the input, since it takes only $O(\log W)$ bits to represent each weight. It turns out that this is fundamental to the algorithm. We can't just change the analysis, we need to change the algorithm itself.

For each node $v$, let $\delta(v) = \{\{u, v\} \in E\}$ denote the edges incident on it. For a set of edges $E'$, we let $w(E') = \sum_{e \in E'} w(e)$. Finally, for $S \subseteq V$ we let $w(S) = w(E(S, \bar{S}))$ denote the weight of the edges that have one endpoint in $S$ and one endpoint in $\bar{S}$.

Consider the following local search algorithm. Intuitively, this algorithm start off with a "reasonable" solution (rather than an arbitrary one), and then only makes local improvements that help "a lot".

**Algorithm 2** WEIGHTED MAX-CUT LOCAL SEARCH Algorithm

**Input**: Undirected graph $G = (V, E)$.
**Output**: A set $S \subseteq V$.

$S \leftarrow \{v\}$, where $w(\delta(v)) = \max_{u \in V} w(\delta(u))$.
**while** $\exists v \in \bar{S}$ (or $S$) such that $w(S \cup \{v\}) \geq (1 + \frac{\epsilon}{n})w(S)$ (or $w(S \setminus \{v\}) \geq (1 + \frac{\epsilon}{n})w(S)$) **do**
  Add (remove) $v$ if first (second) condition.
**end while**
**return** $S$

**Theorem 5.3.1** *There are at most $O(\frac{1}{\epsilon} \cdot n \log n)$ iterations, and so the total running time is polynomial.*

**Proof:** Let $S_0 = \{v\}, S_1, S_2, \ldots, S_r$ be sets created by the algorithm. Notice that

$$w(S_0) \geq \frac{W}{n},$$

and so by induction we have that

$$w(S_i) \geq \left(1 + \frac{\epsilon}{n}\right) w(S_{i-1}) \geq \left(1 + \frac{\epsilon}{n}\right)^i \frac{W}{n}.$$

Then,

$$\begin{aligned} & w(S_r) \geq \left(1 + \frac{\epsilon}{n}\right)^r \frac{W}{n} && \text{(Note that } w(S_r) \leq W\text{)} \\ \implies & \left(1 + \frac{\epsilon}{n}\right)^r \leq n \\ \implies & r \leq \frac{\log n}{\log\left(1 + \frac{\epsilon}{n}\right)} \leq \frac{\log n}{\frac{\epsilon}{2n}} = \frac{2}{\epsilon} n \log n. \end{aligned}$$

■

**Theorem 5.3.2** *The algorithm is a $(2 + \epsilon)$-approximation.*

**Proof:** One more piece of notation: for a set $T \subseteq V$ and $v \in V$, let $w(\{v\}, S) = \sum_{e=\{v,u\}, u \in S} w_e$.

Let $S$ be the set returned by the algorithm. Then $\forall v \in S$

$$\begin{aligned}
w(S \setminus \{v\}) &\leq \left(1 + \frac{\epsilon}{n}\right) w(S) \\
w(S \setminus \{v\}) - w(S) &\leq \frac{\epsilon}{n} w(S) \\
w(\{v\}, S) - w(\{v\}, \bar{S}) &\leq \frac{\epsilon}{n} w(S) \\
w(\{v\}, \bar{S}) &\geq w(\{v\}, S) - \frac{\epsilon}{n} w(S) \\
2w(\{v\}, \bar{S}) &\geq w(\delta(v)) - \frac{\epsilon}{n} w(S) \\
w(\{v\}, \bar{S}) &\geq \frac{1}{2} w(\delta(v)) - \frac{\epsilon}{2n} w(S).
\end{aligned}$$

Similarly for $v \in \bar{S}$

$$w(\{v\}, S) \geq \frac{1}{2} w(\delta(v)) - \frac{\epsilon}{2n} w(S).$$

Therefore

$$\begin{aligned}
2w(S) = \sum_{v \in S} w(\{v\}, \bar{S}) + \sum_{v \in \bar{S}} w(\{v\}, S) &\geq \sum_{v \in V} \left( \frac{1}{2} w(\delta(v)) - \frac{\epsilon}{2n} w(S) \right) \\
&= W - \frac{\epsilon}{2} w(S)
\end{aligned}$$

which implies

$$w(S) \geq \frac{W}{2 + \frac{\epsilon}{2}}.$$

Since $OPT \leq W$, this completes the proof. ■