1 Maximum Cut Problem

Problem formulation

For a graph $G = (V, E)$ we use $n = |V|$ an $m = |E|$. Given a subset $S \subseteq V$ we denote by $\delta(S) = \{(i, j) \in E \mid i \in S,\ j \notin S\} \subseteq E$ the so called *boundary* of S. The *maximum cut problem* consists in finding a subset $S \subseteq V$ such that $|\delta(S)|$ is as large as possible:

$$mc(G) \;=\; \max_{S \subseteq V} |\delta(S)|.$$

We say that a subset $F \subseteq E$ of the edges of G is a *cut*, if $F = \delta(S)$ for a subset $S \subseteq V$.

Given a weight function $w : E \to \mathbb{R}_+$, the problem if finding a cut $F \subseteq E$ such that $w(F) = \sum_{e \in F} w(e)$ is as large as possible is called the *weighted maximum cut problem.* For $e = (i, j) \in E$ we will also use the notation $w_{i,j} = w(e)$. An equivalent way of writing the weighted maximum cut problem is

$$mc_w(G) \;=\; \max_{S \subseteq V} \sum_{\substack{(i,j) \in E \\ i \in S,\ j \notin S}} w_{i,j}.$$

Note also that the weighted maximum cut problem can always be viewed as a problem over the complete graph on vertex set V, simply by extending the weight function:

$$\hat{w}_{i,j} = \begin{cases} w_{i,j} & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

and writing

$$mc_w(G) \;=\; \max_{S \subseteq V} \sum_{\substack{i \in S \\ j \in V \setminus S}} \hat{w}_{i,j}.$$

In the sequel we write about the maximum cut problem (MAXCUT in short), however most results and algorithms can naturally be extended to the weighted case (WMAXCUT in short).

Applications

The (weighted) MAXCUT problem has numerous applications in data analysis (clustering), VLSI design (via planning), physics (Ising problem), floor planning (process partitioning), etc. See in class.

Vertex Formulation

In this section we focus on results and algorithms that view a cut as the boundary of a vertex set S. In this setting the decision variables are associated with the vertices of the graph and the questions they represent are about the membership relation $i \in S$ or $i \notin S$ for $i \in V$.

Lemma 1 *For every graph $G = (V, E)$ with $m = |E|$ we have*

(i) $\frac{m}{2} \leq mc(G) \leq m$;

(ii) finding a subset $S \subseteq V$ *with* $\frac{m}{2} \leq |\delta(S)| \leq mc(G)$ *can be done in polynomial time (2-approximation);*

(iii) $mc(G) = m$ *iff* G *has no odd cycles iff* G *is bipartite.*

Proof: Note first that it is enough to prove the above statements for connected graphs.

For (i) let us consider random subsets $\mathbf{S} \subseteq V$ with probabilities $Prob(i \in \mathbf{S}) = \frac{1}{2}$ for all $i \in V$. Then for an edge $(i, j) \in E$ we have $Prob((i, j) \in \delta(\mathbf{S})) = \frac{1}{2}$. Consequently, $Exp[|\delta(\mathbf{S})|] = \frac{m}{2}$. Since $mc(G) \geq Exp[|\delta(\mathbf{S})|]$, the claim follows. See **randomized rounding** in next paragraph.

For (ii) note first that if $f_G(x) = \sum_{(i,j)\in E}(x_i(1 - x_j) + (1 - x_i)x_j)$, then we have $mc(G) = \max_{x\in\{0,1\}^n} f_g(x)$ and $Exp[|\delta(\mathbf{S})|] = f_G(\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})$. Since f_G is multilinear, we can "round up" from $(\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})$ to a binary vector $\tilde{x} \in \{0, 1\}^n$ such that $f_G(\tilde{x}) \geq Exp[|\delta(\mathbf{S})|] = f_G(\frac{1}{2}, \frac{1}{2}, ..., \frac{1}{2})$. (See details in class – **random cuts**.)

For (iii) note that no odd cycle $C \subseteq E$ can have $C \subseteq \delta(S)$ for any subset $S \subseteq V$. Therefore, if G contains an odd cycle, then we must have $mc(G) < m$. Furthermore, if G is connected and has no odd cycles, then the following procedure will assign all vertices uniquely to S or $\bar{S} = V \setminus S$: assume $V = \{1, 2, ..., n\}$; start with assigning 1 to S; then assign all neighbors of S to $\bar{S}$; then assign all neighbors of $\bar{S}$ to S; then assign all neighbors of S to $\bar{S}$; ... Consequently, all edges will belong to $\delta(S)$ in this case, that is $mc(G) = |\delta(S)| = m$. □

Local optima

Recall that for subsets $A, B \subseteq V$ we have $A \triangle B = (A \setminus B) \cup (B \setminus A)$.

A subset $S \subseteq V$ is called a *local optimum* for the MAXCUT problem if $|\delta(S \triangle \{i\})| \leq |\delta(S)|$ for all vertices $i \in V$.

Lemma 2 *For every graph $G = (V, E)$ we have*

(iv) LocalOpt(G) *stops in* $O(m^2)$ *time and outputs a locally optimal subset* $S \subseteq V$.

(v) For every locally optimal subset $S \subseteq V$ *we have* $|\delta(S)| \geq \frac{m}{2}$.

Proof: It is easy to see that the inside steps of a WHILE loop can be executed in $O(m)$ time. Since we execute WHILE loops as long as $m \geq |\delta(Q)| > |\delta(S)|$, we cannot have more than m such iterations. Finally, the WHILE loop stops only if $|\delta(Q \triangle \{i\})| \leq |\delta(Q)|$ for all vertices $i \in V$, which is equivalent to say that S is a local optimum.

LocalOpt(G)

Require: a graph $G = (V, E)$.
Ensure: a cut $F = \delta(S)$ for a subset $S \subseteq V$ that is a local optimum.

```
Initialize S = ∅ and ∅ ≠ Q ⊆ V.
while |δ(Q)| > |δ(S)| do
  S ← Q
  for i ∈ V do
    if |δ(Q△{i})| > |δ(Q)| then
      Q ← Q△{i}.
    end if
  end for
end while
```

For (v) note that local optimality means that for every vertex $i \in V$ we have at least half of the incident edges in $\delta(S)$, which implies the claim. □

Greedy algorithm

Greedy(G)

Require: a graph $G = (V, E)$.
Ensure: a cut $F = \delta(S)$ for a subset $S \subseteq V$.

```
Initialize S = ∅ and S̄ = ∅, V = {1, 2, ..., n}
for i = 1, ..., n do
  if |δ(S ∪ {i})| > |δ(S)| then
    S ← S ∪ {i}
  else
    S̄ ← S̄ ∪ {i}
  end if
end for
```

Lemma 3 *Algorithm* Greedy(G) *outputs a cut* $|\delta(S)| \geq \frac{m}{2}$ *in time* $O(n^2)$.

Proof: Clearly, after building an incidence structure in $O(m) = O(n^2)$ time, each main iteration of the algorithm can be executed in $O(n)$ time. Since we have n main iterations, the complexity claim follows.

Let us say that vertex $i \in V$ is completing an edge (i, j) if $j < i$. Let us denote by c_i the number of edges that is completed by vertex i, $i \in V$. Since every edge has a unique vertex that completes it, we have $\sum_{i \in V} c_i = m$. On the other hand, due to our choice in the algorithm, we have that at least half of the completed c_i edges belong to the cut, when we process vertex i. Thus, we have at the end $|\delta(S)| \geq \frac{m}{2}$. □

Edge Formulation

In this section we focus on models that describe a cut as a set of edges $F \subseteq E$. The principal decision variables are y_e, $e \in E$ with the meaning $y_e = 1$ iff $e \in F$. Of course, it is not trivial to describe the conditions that make sure that the resulting set F will be a cut.

Lemma 4 *A subset $F \subseteq E$ is a cut iff the subgraph (V, F) does not contain an odd cycle iff (V, F) is bipartite.*

Proof: It is immediate to see that if $F = \delta(S)$ for a subset of vertices $S \subseteq V$, then $(V, F) = (S, V \setminus S, F)$ is bipartite, and hence $F = \delta(S)$ cannot contain an odd cycle. For the reverse direction see the proof of (iii) of Lemma 1. $\square$

Let us define an edge set $F \subseteq E$ *independent* if the subgraph (V, F) is bipartite. This defines an independence system. Computing the rank of a subset maybe difficult, but for certain subsets it is easy. For instance, if $C \subseteq E$ is an odd cycle, then $rank(C) = |C| - 1$. It turns out that these rank inequalities are enough to describe the MAXCUT value:

Lemma 5 *Given a graph $G = (V, E)$, we have*

$$\begin{aligned} mc(G) = \quad & \max \sum_{e \in E} y_e \\ & \sum_{e \in C} y_e \qquad \le |C| - 1 \quad C \subseteq E, \quad \textit{odd cycle} \end{aligned}$$

Proof: Clearly, any feasible solution to this problem is free of odd cycles. Thus the claim follows by Lemma 4. $\square$

Lemma 6 *The continuous relaxation of the problem in Lemma 5 has a polynomial separation.*

Proof: Assume $0 \le y_e^* \le 1$, $e \in E$ is an arbitrary feasible solution to the continuous relaxation of the problem. Let us create an auxiliary graph $H = (W, L)$ by setting $W = \{i, i' \mid i \in V\}$ and $L = \{(i, j'), (i', j) \mid (i, j) \in E\}$. Assign length $l(i, j') = l(i', j) = 1 - y_{i,j}^*$ for all $(i, j) \in E$. Let us then denote by SP the shortest $i - i'$ path in H. We claim that if $SP < 1$ then the edges of this shortest path form an odd cycle violated by y^*, while if $SP \ge 1$, then y^* is an optimal solution to the continuous relaxation of the problem. $\square$

Odd subsets formulation

Given a graph $G = (V, E)$ let us consider an arbitrary cycle $C \subseteq E$ and an odd subset $F \subseteq C$. If the binary vector $y \in \{0, 1\}^E$ represents a cut then we must have

$$\sum_{e \in F} y_e + \sum_{e \in C \setminus F} (1 - y_e) \le |C| - 1$$

Consequently, we have

$$
\begin{aligned}
mc(G) = \quad & \max \sum_{e \in E} y_e \\
& \sum_{e \in F} y_e + \sum_{e \in C \setminus F} (1 - y_e) \quad \leq |C| - 1 \quad C \subseteq E, \ \text{ cycle }, \ F \subseteq C, \ \text{odd}
\end{aligned}
$$

Lemma 7 *The continuous relaxation of the above problem has a polynomial separation.*

Proof: Consider an feasible solution $0 \leq y_e^* \leq 1$, $e \in E$. Let us create an auxiliary graph $H = (W, L)$ by defining $W = \{i, i' \mid i \in V\}$ and $L = \{(i,j), (i',j'), (i',j), (i,j') \mid (i,j) \in E\}$. Assign lengths $\ell(i,j') = \ell(i',j) = 1 - y_{i,j}^*$ and $\ell_{i,j} = \ell_{i',j'} = y_{i,j}^*$ for all $(i,j) \in E$. Let us then denote by SP the shortest $i - i'$ path in H. We claim that if $SP < 1$ then the edges of this shortest path form an odd cycle violated by y^*, while if $SP \geq 1$, then y^* is an optimal solution to the continuous relaxation of the problem. □

Planar Graphs

The MAXCUT problem can be solved in polynomial time for certain graphs. This includes the family of planar graphs (Orlova and Dorfman, 1972; Hadlock 1975)

Planar graphs and their duals

See in class.

Lemma 8 *The complement of a maximum cut in a planar graph is a minimum odd cycle cover.*

Proof: See in class. □

Lemma 9 *The minimum of an odd cycle cover in a planar graph corresponds to an optimal chinese postman tour in the dual graph.*

Proof: See in class. □