CSE 421

Menger’s Theorem, Image Segmentation
NP-Completeness

Shayan Oveis Gharan
Network Connectivity
Network Connectivity

Given a digraph $G = (V, E)$ and two nodes $s$ and $t$, find min number of edges whose removal disconnects $t$ from $s$.

Def. A set of edges $F \subseteq E$ disconnects $t$ from $s$ if all $s$-$t$ paths uses at least one edge in $F$.

Ex: In testing network reliability
Network Connectivity using Min Cut

Thm. [Menger 1927] The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

Pf. \leq
Suppose the removal of F ⊆ E disconnects t from s, and |F| = k. All s-t paths use at least one edge of F. Hence, the number of edge-disjoint paths is at most k.
**Network Connectivity using Min Cut**

**Thm. [Menger 1927]** The max number of edge-disjoint s-t paths is equal to the min number of edges whose removal disconnects t from s.

**Pf. ≥**
Suppose there are k edge disjoint paths from s to t
So, Max flow is k
So, there is a s-t cut (A,B) s.t., \( \text{cap}(A,B) = k \)
Let F be the edges out of A. So, \( |F| = k \).
If we remove F we disconnect t from s.
Image Segmentation
Image Segmentation

Given an image we want to separate foreground from background

• Central problem in image processing.
• Divide image into coherent regions.
Foreground / background segmentation

Label each pixel as foreground/background.

- $V = \text{set of pixels, } E = \text{pairs of neighboring pixels.}$
- $a_i \geq 0$ is likelihood pixel $i$ in foreground.
- $b_i \geq 0$ is likelihood pixel $i$ in background.
- $p_{i,j} \geq 0$ is separation penalty for labeling one of $i$ and $j$ as foreground, and the other as background.

**Goals.**

**Accuracy:** if $a_i > b_i$ in isolation, prefer to label $i$ in foreground.

**Smoothness:** if many neighbors of $i$ are labeled foreground, we should be inclined to label $i$ as foreground.

Find partition $(A, B)$ that maximizes:

$\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E} p_{i,j}$
Difficulties:
- Maximization (as opposed to minimization)
- No source or sink
- Undirected graph

Step 1: Turn into Minimization

Maximizing $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, i \in A, j \in B} p_{i,j}$

Equivalent to minimizing $\sum_{i \in V} a_i + \sum_{j \in V} b_j - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{(i,j) \in E, i \in A, j \in B} p_{i,j}$

Equivalent to minimizing $\sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{(i,j) \in E, i \in A, j \in B} p_{i,j}$
Min cut Formulation (cont’d)

G' = (V', E').

Add s to correspond to foreground;
Add t to correspond to background
Use two anti-parallel edges instead of undirected edge.
Consider min cut \((A, B)\) in \(G'\). \((A = \text{foreground})\)

\[
cap(A, B) = \sum_{j \in B} a_j + \sum_{i \in A} b_i + \sum_{(i,j) \in E} p_{i,j}
\]

Precisely the quantity we want to minimize.
Reductions & NP-Completeness
Computational Complexity

**Goal**: Classify problems according to the amount of computational resources used by the best algorithms that solve them

Here we focus on time complexity

**Recall**: worst-case running time of an algorithm

- **max** # steps algorithm takes on any input of size n
Computational Complexity and Reduction

In most cases, we cannot characterize the true hardness of a computational problem.

So?

We only reduce the number of problems

Want to be able to make statements of the form:

• “If we could solve problem B in polynomial time then we can solve problem A in polynomial time”
• “Problem B is at least as hard as problem A”
Polynomial Time Reduction

Def $A \leq_P B$: if there is an algorithm for problem $A$ using a ‘black box’ (subroutine) that solve problem $B$ s.t.,
- Algorithm uses only a polynomial number of steps
- Makes only a polynomial number of calls to a subroutine for $B$

So,

B is Polynomial time solvable $\implies$ A is Polynomial time solvable

Conversely,

No efficient Algorithm for $A$ $\implies$ No efficient Algorithm for $A$

In words, $B$ is as hard as $A$ (it can be even harder)
Here, we will always use a restricted form of polynomial-time reduction often called Karp or many-to-one reduction.

\( A \leq_{p}^{1} B \): if and only if there is an algorithm for \( A \) given a black box solving \( B \) that on input \( x \)

- Runs for polynomial time computing an input \( f(x) \) of \( B \)
- Makes one call to the black box for \( B \) for input \( f(x) \)
- Returns the answer that the black box gave

We say that the function \( f(.) \) is the reduction.
Example 1: Indep Set $\leq_p$ Clique

**Indep Set**: Given $G=(V,E)$ and an integer $k$, is there $S \subseteq V$ s.t. $|S| \geq k$ and no two vertices in $S$ are joined by an edge?

**Clique**: Given a graph $G=(V,E)$ and an integer $k$, is there $S \subseteq V$, $|U| \geq k$ s.t., every pair of vertices in $S$ is joined by an edge?

**Claim**: Indep Set $\leq_p$ Clique

**Pf**: Given $G = (V, E)$ and instance of indep Set. Construct a new graph $G' = (V, E')$ where $\{u, v\} \in E'$ if and only if $\{u, v\} \notin E$.

[S is an independent set in G]  [S is an Clique in G']
Example 2: Vertex Cover $\leq_p$ Indep Set

**Vertex Cover:** Given a graph $G=(V,E)$ and an integer $k$, is there a vertex cover of size at most $k$?

**Claim:** For any graph $G = (V, E)$, $S$ is an independent set iff $V - S$ is a vertex cover.

**Pf:**

$=>$ Let $S$ be an independent set of $G$.
Then, $S$ has at most one endpoint of every edge of $G$.
So, $V - S$ has at least one endpoint of every edge of $G$.
So, $V - S$ is a vertex cover.

<=$=$ Suppose $V - S$ is a vertex cover.
Then, there is no edge between vertices of $S$ (otherwise, $V - S$ is not a vertex cover).
So, $S$ is an independent set.
Example 3: Vertex Cover \( \leq_p \) Set Cover

Set Cover: Given a set \( U \), collection of subsets \( S_1, \ldots, S_m \) of \( U \) and an integer \( k \), is there a collection of \( k \) sets that contain all elements of \( U \)?

Claim: Vertex Cover \( \leq_p \) Set Cover

Pf:
Given \( (G = (V, E), k) \) of vertex cover we construct a set cover input \( f(G, k) \)

- \( U = E \)
- For each \( v \in V \) we create a set \( S_v \) of all edges connected to \( v \)

This clearly is a polynomial-time reduction

So, we need to prove it gives the right answer
Example 3: Vertex Cover \( \leq_p \) Set Cover

**Claim**: Vertex Cover \( \leq_p \) Set Cover

**Pf**: Given \((G = (V, E), k)\) of vertex cover we construct a set cover

- \( U = E \)
- For each \( v \in V \) we create a set \( S_v \) of all edges connected to \( v \)

Vertex-Cover \((G, k)\) is yes \( \Rightarrow \) Set-Cover \( f(G, k) \) is yes

If a set \( W \subseteq V \) covers all edges, just choose \( S_v \) for all \( v \in W \), it covers all \( U \).

Set-Cover \( f(G, k) \) is yes \( \Rightarrow \) Vertex-Cover \((G, k)\) is yes

If \((S_{v_1}, ..., S_{v_k})\) covers all \( U \), the set \( \{v_1, ..., v_k\} \) covers all edges of \( G \).
A decision problem is a computational problem where the answer is just yes/no.

Here, we study computational complexity of decision Problems.

Why?
- much simpler to deal with
- Decision version is not harder than Search version, so it is easier to lower bound Decision version
- Less important, usually, you can use decider multiple times to find an answer.