Algorithm Design & Analysis (CSE222)

Lecture-23

Recap

- Scaling Max-Flow
  - In the residual graph, choose the s-t path with high bottleneck.
  - Running time: $O(E^2 \cdot \log_2 C)$

- Shortest Path
  - In the residual graph, choose the shortest (#edges) path s-t.
  - Running time: $O(E^2 \cdot V)$

Outline

- Network Flow: Applications

- Computational Intractability

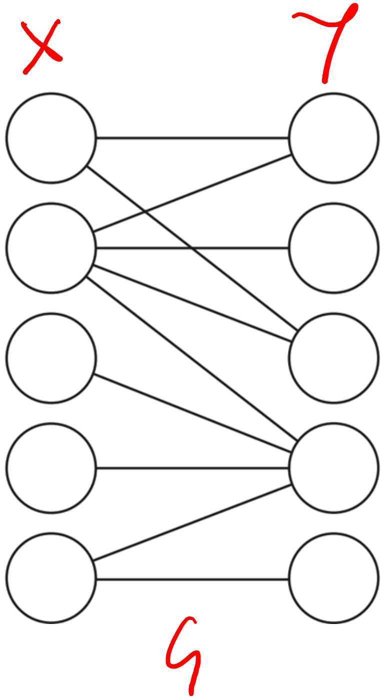
Bipartite Matching

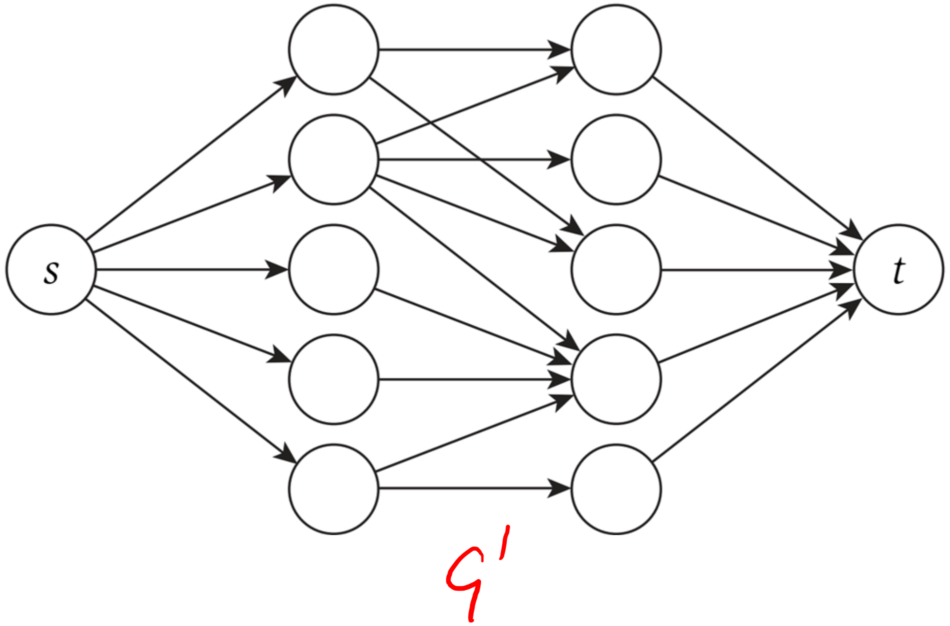
Let G = (V, E) be an undirected graph such that V = X U Y and for every e ∈ E, one end point is in X and another is in Y. A matching M ⊆ E such that each node appears in at most one edge in M.

Problem: Compute maximum matching of G.

Given an instance of bipartite matching create an instance of maximum flow.

- Give directions.
- Add s and t.
- Assign capacity 1.



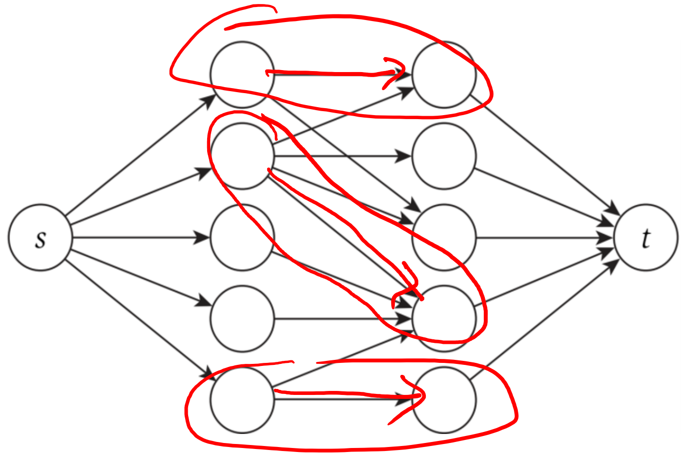


Bipartite Matching

Compute maximum flow.

Let there is a matching of k edges (x_1,y_1), (x_2,y_2),…(x_k,y_k). Each edge (x_i,y_j) will pass 1 unit of ~~flow s-x_i-y_j-t, f(x_i,y_j)~~=1. Since, capacity of each edge is 1, so due to both constraints the total flow is k.

Again, let f' is a flow of value k. Then by integrality property and all capacities being 1, for every edge f(e) is 1 or 0. Let, M' be the set of edges (x,y) such that f(x,y) = 1.



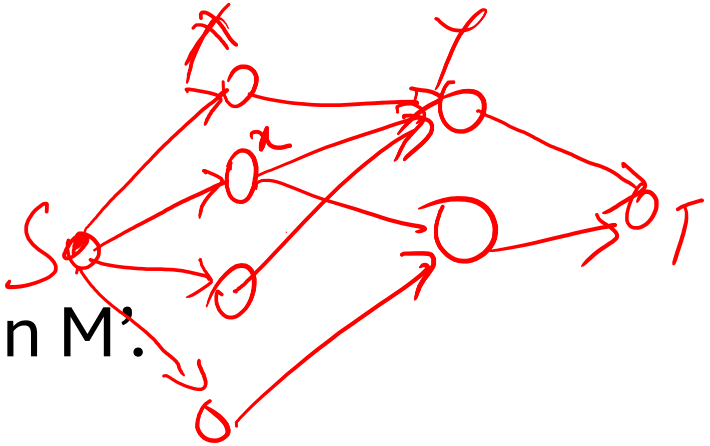
Bipartite Matching

Claim: M’ has k edges.

Proof: Let A = s U X and B = t U Y.

- Value of flow is total flow leaving A minus total flow entering A.
- The first terms is of |M’|, as all edge carries 1 unit of flow.
- The second term is 0 as there is no edge that goes into A in the graph.
- So, M’ contains k edges.

Bipartite Matching



<u>Claim</u>: Each node in X is the tail of at most one edge in M'.

<u>Proof</u>: Prove by contradiction.

- Let x ∈ X be tail of at least two edges in M'.
- As flow is integer valued, so at least two unit of flow leaves x.
- By conservation constraints, at least two unit of flow would have come to x.
- This is impossible as only one edge with capacity 1 comes into x from s.
- So, x is the tail of at most one edge in M'.

<u>Claim</u>: Each node in Y is the head of at most one edge in M'.

Bipartite Matching

Combining all the claims we get,

Claim: The size of maximum matching is equal to the value of maximum flow in G’ and the edges in such a matching G are the ones that carries flow X to Y in G’.

Outline

- Network Flow: Applications

- Computational Intractability

Computational Intractability

- Informal categorization of problems

- Polynomial time algorithms

NP-Completeness

- A large class of problems are equivalent. A polynomial time algorithm to any one of them implies existence of polynomial time algorithm for all of them.
- Exists thousands of such problems.
- All these problems are just one single open problem.

NP-Completeness

- Computationally hard, though we cannot prove it.

- Identifying a problem to be NP-Complete is a good enough reason to stop looking for an efficient algorithm for it.

- Compare relative difficulty of different problems.

Reductions

To prove some problems are computationally difficult we show,

- ‘Problem X is at least as hard as problem Y’

To claim this we reduce problem Y to problem X: If there is a black box that solves instances of X. How to solve any instance of Y in polynomial number of steps and polynomial number of calls to the black box that solves X?

If yes then denote $Y \leq_p X$.
Read as: ‘Y is polynomial time reducible to X’ or ‘X is as hard as Y with respect to polynomial running time’

Reduction

If $Y \leq_p X$, and X is polynomial time solvable then we can replace the black box call to X with the polynomial time algorithm for X. Recall Y was solved in polynomial number of steps and polynomial number of calls to the black box.

Now solution of Y involves polynomial number of steps and polynomial number of calls to a subroutine (algorithm for X) that runs in polynomial time. Hence, solving Y takes polynomial time.

<u>Claim</u>: Let $Y \leq_p X$. If X can be solved in polynomial time then Y can be solved in polynomial time.

Reduction

Y is polynomial time reducible to X: $Y \leq_p X$

Although we use X to solve instances of Y, it does not mean that X is less harder than Y or Y is at least as hard as X.

Problem: (Y) Smallest number in an array of size n.
It takes $O(n)$.
(X) Sort the array in ascending order.
It takes $O(n\log n)$.

$Y \leq_p X$. Solving Y by reducing to X takes $O(n\log n + n) = O(n\log n)$.

Reduction

Claim: Let $Y \leq_p X$. If Y cannot be solved in polynomial time then X cannot be solved in polynomial time.

Proof:

- If we have a problem Y that is known to be hard and $Y \leq_p X$ then the hardness extends to problem X.
- Else we could use X to solve Y.

So, if we can find one hard problem (Y) then we can show that another problem (X) is hard by reducing Y to X.

Reduction Example: Max Bipartite Matching $\leq_p$ Max Network Flow.

Vertex Cover

A vertex cover of a graph G = (V, E) is a set of nodes S such that every edge has at least one endpoints in S.

Cover each edge by choosing at least one of its vertices.

Problem: Given a graph G and a number k, does G contains a vertex cover of size at most k.

Independent Set

An independent set of a graph G = (V, E) is a set of nodes S such that no two nodes are adjacent to each other.

Problem: Given a graph G and a number k, does G contains an independent set of size at least k.

Relation b/w Independent Set and Vertex Cover

Claim: Given a graph G = (V, E), iff S is an independent set of G then V \ S is a vertex cover of G.

Proof: Both independent set and vertex cover are defined using edges.

- Suppose S is an independent set and let e = (u, v) be some edge.
- At most one of u or v can be in S. Hence, at least one of u or v is in V-S.
- So V \ S is a vertex cover.

- Let V \ S is a vertex cover and u, v ∈ S.
- Then there could not be an edge between u and v, else edge would be covered in V \ S.
- So, S is an independent set.

Optimization Vs Decision

Independent Set

Optimization Problem: Compute maximum independent set in given graph G.

Decision Problem: Given a graph G and a number k check if there exists an independent set of size at least k.

Both problems are same: Solving one would solve other.

How solving optimization problem solves decision problem? (trivial)

How solving decision problem solves optimization problem? (non-trivial)

If decision problem is hard then its optimization is also hard.

Independent Set $\leq_p$ Vertex Cover

We need to show any instance of independent set into an instance of vertex cover

$|V| = n$

- Given an instance of independent set {G, k},
- Query vertex cover black box if there is a vertex cover of size ≤ n-k.

- If yes from black box then there is an independent set of size ≥ k
- If no from black box, there is no VC of size ≤ n-k so there is no IS of size ≥ k

Vertex Cover $\leq_p$ Independent Set

Claim: Vertex cover $\leq_p$ Independent set

Proof: To verify if G has a vertex cover of size at most k, we ask independent set black box to check independent set of size at least n-k.

So, both the problems are equivalently difficult.

Reference

Slides

Algorithms Design by Kleinberg & Tardos - Chp 8.1