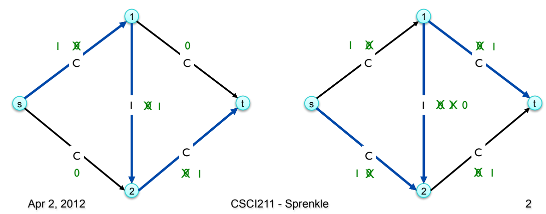
Objectives

- Network Flow Apps
  - Capacity Scaling
- Computational intractability

Review: Ford-Fulkerson: Exponential Number of Augmentations

- Is generic Ford-Fulkerson algorithm polynomial in input size?
  - No. If max capacity is C, then algorithm can take C iterations.

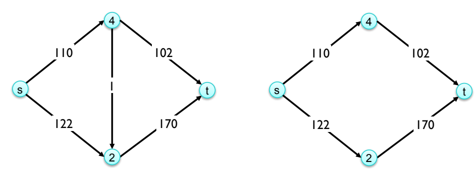


Review: Choosing Good Augmenting Paths

- Use care when selecting augmenting paths
  - Some choices lead to exponential algorithms
  - Clever choices lead to polynomial algorithms
  - If capacities are irrational, algorithm not guaranteed to terminate!
- **Goal: choose augmenting paths so that:**
  - Can find augmenting paths efficiently
  - Few iterations
- [Edmonds-Karp 1972, Dinitz 1970] Choose augmenting paths with:
  - Max bottleneck capacity
  - Fewest number of edges
  - *Sufficiently large bottleneck capacity*

Review: Intuition for Capacity Scaling

- Choosing path with highest bottleneck capacity increases flow by max possible amount.
  - Don't worry about finding *exact* highest bottleneck path
  - Maintain scaling parameter Δ
  - Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only edges with capacity at least Δ



G_f $G_f(100)$

Review: Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:
       while there exists augmenting path P in Gf(Δ) :
           f = augment(f, c, P)
           update Gf(Δ)
       Δ = Δ / 2

   return f
```

- Why does this work?
- What is its running time?

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c)
   foreach e ∈ E,  f(e) = 0
   Δ = greatest power of 2 less than or equal to C
   Gf = residual graph
   Gf(Δ) = Δ-residual graph

   while Δ ≥ 1:     O(log C)
       while there exists augmenting path P in Gf(Δ) :
           f = augment(f, c, P)
           update Gf(Δ)
       Δ = Δ / 2

   return f
```

Capacity Scaling: Correctness

- Assumption. All edge capacities are integers between 1 and C.
- Integrality invariant. All flow and residual capacity values are integral.
- Correctness. If the algorithm terminates, then *f* is a max flow.
- Pf.
  - By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$
  - Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Capacity Scaling: Running Time

- Lemma 1. The outer while loop repeats $O(\log_2 C)$ times.
- Proof. Initially $\Delta \leq C$. Δ decreases by a factor of 2 each iteration. ▪

Capacity Scaling: Running Time

What happens to the flow's value at each iteration of the loop?

- Lemma 2. Let f be the flow at the end of a Δ-scaling phase. Then value of the maximum flow is at most $v(f) + m\,\Delta$.

Proof and further analysis in the book

Objectives

- Oh, the places you've been!
- Oh, the places you'll go!

Now, everything comes down to expert knowledge of **algorithms** and **data structures**. If you don't speak fluent **O-notation**, you may have trouble getting your next job at the technology companies in the forefront.
— Larry Freeman

Algorithm Design Patterns

- What are some approaches to solving problems?
- How do they compare in terms of difficulty?

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow

Course Objectives: Given a problem…

You'll recognize when to try an approach
- AND, when to bail out and try something different

Know the steps to solve the problem using the approach
- e.g., breaking it into subproblems, sorting possibilities in some order

Know how to ***analyze*** the run time of the solution
- e.g., solving recurrence relation

Algorithm Design Patterns

- Greedy
- Divide-and-conquer
- Dynamic programming
- Duality/network flow
- Reductions – Chapter 8
- Local search – Chapter 12
- Randomization – Chapter 13

What Was Our Goal In Finding a Solution?

Polynomial Time → Efficient

POLYNOMIAL-TIME REDUCTIONS

Classify Problems According to Computational Requirements

Fundamental Question:
Which problems will we be able to solve in practice?

Classify Problems According to Computational Requirements

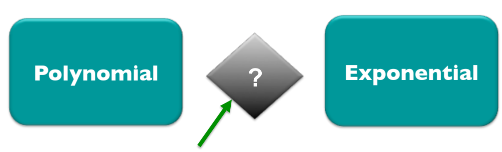
Which problems will we be able to solve in practice?

- Working definition. [Cobham 1964, Edmonds 1965, Rabin 1966] Those with polynomial-time algorithms.

Yes	Probably no
Shortest path	Longest path
Matching	3D-matching
Min cut	Max cut
2-SAT	3-SAT
Planar 4-color	Planar 3-color
Bipartite vertex cover	Vertex cover
Primality testing	Factoring

Classify Problems

Classify problems according to those that can be solved in polynomial-time and those that cannot.



Frustrating news: *Many* problems have defied classification.
Chapter 8. Show that problems are "computationally equivalent" and appear to be manifestations of one *really hard* problem.

Examples:
- Given a Turing machine, does it halt in at most *k* steps?
- Given a board position in an n-by-n generalization of chess, can black guarantee a win?

Polynomial-Time Reduction

Suppose we could solve Y in polynomial time.
What else could we solve in polynomial time?

Polynomial-Time Reduction

Suppose we could solve Y in polynomial-time.
What else could we solve in polynomial time?

- Reduction. Problem *X* *polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y

For X + **Y**

- Notation: $X \leq_P Y$
  - "X is polynomial-time reducible to Y"
- Conclusion: If Y can be solved in polynomial time and $X \leq_P Y$, then X can be solved in polynomial time.

NP-Complete Problems

- Problems from many different domains whose complexity is unknown
- NP-completeness and proof that all problems are equivalent is **POWERFUL**!
  - All open complexity questions ➔ **ONE** open question!
- What does this mean?
  - "Computationally hard for practical purposes, but we can't prove it"
  - If you find an NP-Complete problem, you can stop looking for an efficient solution
    - Or figure out efficient solution for ALL NP-complete problems

Polynomial-Time Reduction

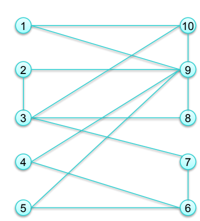
- Purpose. Classify problems according to *relative difficulty*.
- Design algorithms. If $X \leq_P Y$ and Y can be solved in polynomial-time, then X can also be solved in polynomial time.
- Establish intractability. If $X \leq_P Y$ and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial time.
- Establish equivalence. If $X \leq_P Y$ and $Y \leq_P X$, we use notation $X \equiv_P Y$.

Basic Reduction Strategies

- *Reduction by simple equivalence*
- Reduction from special case to general case
- Reduction by encoding with gadgets

Independent Set

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$ and for each edge **at most one** of its endpoints is in *S*?

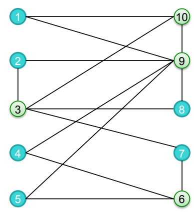


How is this different from the network flow problem?

Ex. Is there an independent set of size ≥ 6?
Ex. Is there an independent set of size ≥ 7?

Independent Set

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$ and for each edge **at most one** of its endpoints is in *S*?

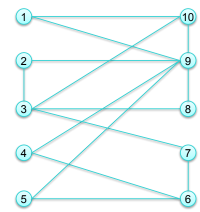


Ex. Is there an independent set of size ≥ 6? Yes
Ex. Is there an independent set of size ≥ 7? No

independent set

Vertex Cover

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$ and for each edge, **at least one** of its endpoints is in S?



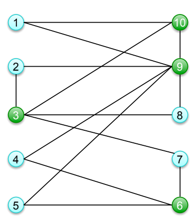
A vertex ***covers*** an edge.

Application: place guards within an art gallery so that all corridors are visible at any time

Ex. Is there a vertex cover of size ≤ 4?
Ex. Is there a vertex cover of size ≤ 3?

Vertex Cover

- Given a graph G = (V, E) and an integer *k*, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$ and for each edge, **at least one** of its endpoints is in S?



Ex. Is there a vertex cover of size ≤ 4? Yes
Ex. Is there a vertex cover of size ≤ 3? No

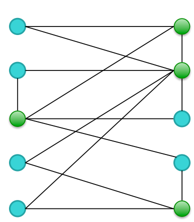
vertex cover

Problem

- Not known if finding Independent Set or Vertex Cover can be solved in polynomial time
- BUT, what can we say about their relative difficulty?

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show *S* is an independent set iff V - S is a vertex cover



independent set

vertex cover

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show S is an independent set iff V - S is a vertex cover
- $\Rightarrow$
  - Let S be an independent set
  - Consider an arbitrary edge (u, v)
  - Since S is an independent set $\Rightarrow u \notin S$ or $v \notin S$ or both $\notin S$ $\Rightarrow u \in V - S$ or $v \in V - S$ or both $\in V - S$
  - Thus, V - S covers (u, v)
    - Every edge has at least one end in V-S
  - V-S is a vertex cover

Vertex Cover and Independent Set

- Claim. VERTEX-COVER $\equiv_P$ INDEPENDENT-SET
- Pf. We show S is an independent set iff V - S is a vertex cover
- $\Leftarrow$
  - Let V - S be any vertex cover
  - Consider two nodes $u \in S$ and $v \in S$
  - Observe that $(u, v) \notin E$ since V - S is a vertex cover
  - Thus, no two nodes in S are joined by an edge $\Rightarrow$ S independent set

Using the Previous Result

- Problem *X* *polynomial reduces to* problem *Y* if arbitrary instances of problem *X* can be solved using:
  - Polynomial number of standard computational steps, ***plus***
  - Polynomial number of calls to **oracle** that solves problem Y
    - Assume have a black box that can solve Y

How do we show polynomial reduction for the independent set and vertex cover?

Summary

- If we have a block box to solve Vertex Cover, can decide whether G has an independent set of size at least *k* by asking the black box whether G has a vertex cover of size at most $n – k$
- If we have a block box to solve Independent Set, can decide whether G has a vertex cover of size at most *k* by asking the block box whether G has an independent set of size at least $n - k$

Planning

- For Friday
  - Problem set – DP notes
- Total problem set points for semester: 201
  - Fill out course evaluations on Sakai
  - If 60% fill out, 1% EC on problem sets
    - Additional 1% for every additional 12.5% who complete
  - Due Monday at midnight