Midterm 2 on April 4th at 8PM (MATH 175)
Practice Midterm Released Soon
3x5 Index Card (Double Sided)

Midterm 2
- **When?**
  - April 4th from 8PM to 10PM (2 hours)
- **Where?**
  - MATH 175
- **What can I bring?**
  - 3x5 inch index card with your notes (double sided)
  - No electronics (phones, computers, calculators etc…)
  - Minimal Coverage of Topics Covered Today
  - PSPACE
  - Dealing with NP-Complete Problems

Midterm 2
- **When?**
  - April 4th from 8PM to 10PM (2 hours)
- **Where?**
  - MATH 175
- **What material should I study?**
  - The midterm will cover recent topics more heavily
  - Network Flow
    - Max-Flow Min-Cut, Augmenting Paths, etc..
    - Ford Fulkerson, Dinic’s Algorithm etc..
    - Applications of Network Flow (e.g., Maximum Bipartite Matching)
  - Linear Programming
  - NP-Completeness
    - Polynomial time reductions, P, NP, NP-Hard, NP-Complete, coNP
  - PSPACE (only basic questions)

9.1 PSPACE

Geography Game

**Geography.** Alice names capital city c of country she is in. Bob names a capital city c’ that starts with the letter on which c ends. Alice and Bob repeat this game until one player is unable to continue. Does Alice have a forced win?

*Ex.* Budapest → Tokyo → Ottawa → Ankara → Amsterdam → Moscow → Washington → Nairobi → ...

**Geography on graphs.** Given a directed graph G = (V, E) and a start node s, two players alternate turns by following, if possible, an edge out of the current node to an unvisited node.

**Decision Problem.** Can first player (Alice) guarantee to make the last legal move?

**Remark.** Some problems (especially involving 2-player games and AI) defy classification according to P, EXPTIME, NP, and NP-Complete.

Chapter 9
PSPACE: A Class of Problems Beyond NP

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.
PSPACE

Decision problems solvable in polynomial time.

PSPACE Decision problems solvable in polynomial space.

Observation. \( P \subseteq PSPACE \).

* poly-time algorithm can consume only polynomial space

---

PSPACE-Complete

Decision problems solvable in polynomial space.

PSPACE-Complete Problem Y is PSPACE-complete if (i) Y is in PSPACE and (ii) for every problem X in PSPACE, \( X \leq_p Y \).

PSPACE-complete problems

- Competitive facility location.
- Natural generalizations of games.
- Othello, Hex, Geography, Rush-Hour, Instant Insanity
- Shanghai, go-moku, Sokoban
- Given a memory restricted Turing machine, does it terminate in at most k steps?
- Do two regular expressions describe different languages?
- Many more.

---

9.5 PSPACE-Complete

Binary counter. Count from 0 to \( 2^n - 1 \) in binary.

Algorithm. Use \( n \) bit odometer.

Claim. 3-SAT is in PSPACE.

Pf.

- Enumerate all \( 2^n \) possible truth assignments using counter.
- Check each assignment to see if it satisfies all clauses.

Theorem. \( NP \subseteq PSPACE \).

Pf. Consider arbitrary problem Y in NP.

- Since \( Y \leq_p 3\text{-SAT} \), there exists algorithm that solves Y in poly-time plus polynomial number of calls to 3-SAT black box.
- Can implement black box in poly-space.

---

Back to NP

This is all you need to know about PSPACE for Midterm 2.

We will return to discuss more advanced topic in the future (e.g., proving that decision problems are NP-Complete).

---

Traveling Salesperson Problem

TSP. Given a set of \( n \) cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

All 13,509 cities in US with a population of at least 500.

Reference: http://www.tsp.gatech.edu
Traveling Salesperson Problem

**TSP.** Given a set of n cities and a pairwise distance function \( d(u, v) \), is there a tour of length \( \leq D \)?

**Optimal TSP tour**
Reference: [http://www.tsp.gatech.edu](http://www.tsp.gatech.edu)

**HAM-CYCLE:** Given a graph \( G = (V, E) \), does there exist a simple cycle that contains every node in \( V \)?

**Claim.** HAM-CYCLE \( \leq_p \) TSP.

**Pf.**
- Given instance \( G = (V, E) \) of HAM-CYCLE, create \( n \) cities with distance function
- TSP instance has tour of length \( \leq n \) iff \( G \) is Hamiltonian.

**Remark.** TSP instance in reduction satisfies \( \Delta \)-inequality.

\[
d(u, v) = \begin{cases} 1 & \text{if } (u, v) \in E \\ 2 & \text{if } (u, v) \notin E \end{cases}
\]
Coping With NP-Completeness

Q.  Suppose I need to solve an NP-complete problem. What should I do?
A.  Theory says you're unlikely to find poly-time algorithm.

Must sacrifice one of three desired features.
- Solve problem to optimality.
- Solve problem in polynomial time.
- Solve arbitrary instances of the problem.

This lecture: Solve some special cases of NP-complete problems that arise in practice.

10.1 Finding Small Vertex Covers

VERTEX COVER: Given a graph G = (V, E) and an integer k, is there a subset of vertices S ⊆ V such that |S| ≤ k, and for each edge (u, v) either u ∈ S, or v ∈ S, or both.

k = 4
S = {3, 6, 7, 10}

Finding Small Vertex Covers

Q. What if k is small?

Brute force. O(kn^k).
- Try all C(n, k) = O(n^k) subsets of size k.
- Takes O(kn) time to check whether a subset is a vertex cover.

Goal: Limit exponential dependency on k, e.g., to O(2^k kn).

Ex. n = 1,000, k = 10.
Brute. k^n = 10^40 ⇒ infeasible.
Better. 2^k kn = 10^10 ⇒ feasible.

Remark. If k is a constant, algorithm is poly-time; if k is a small constant, then it's also practical.

Finding Small Vertex Covers

Claim. Let u-v be an edge of G. G has a vertex cover of size ≤ k iff at least one of G - {u} and G - {v} has a vertex cover of size ≤ k - 1.

Pf. ⇒
- Suppose G has a vertex cover S of size ≤ k.
  - S contains either u or v (or both). Assume it contains u.
  - S - {u} is a vertex cover of G - {u}.

Pf. ⇐
- Suppose S is a vertex cover of G - {u} of size ≤ k - 1.
  - Then S ∪ {u} is a vertex cover of G.

Claim. If G has a vertex cover of size k, it has ≤ k(n-1) edges.

Pf. Each vertex covers at most n-1 edges.

Finding Small Vertex Covers: Algorithm

Claim. The following algorithm determines if G has a vertex cover of size ≤ k in O(2^k kn) time.

boolean Vertex-Cover(G, k) {
  if (G contains no edges) return true
  if (G contains ≥ kn edges) return false
  let (u, v) be any edge of G
  a = Vertex-Cover(G - {u}, k-1)
  b = Vertex-Cover(G - {v}, k-1)
  return a or b
}

Pf.
- Correctness follows from previous two claims.
- There are ≤ 2^k nodes in the recursion tree; each invocation takes O(kn) time.
Finding Small Vertex Covers: Recursion Tree

\[
T(n, k) = \begin{cases} 
  c & \text{if } k = 0 \\
  cm & \text{if } k = 1 \\
  2T(n, k-1)+ck & \text{if } k > 1 
\end{cases} \Rightarrow T(n, k) \leq 2^c n k
\]

Independent Set on Trees

**Independent set on trees.** Given a tree, find a maximum cardinality subset of nodes such that no two share an edge.

**Fact.** A tree on at least two nodes has at least two leaf nodes.

**Key observation.** If \( v \) is a leaf, there exists a maximum size independent set containing \( v \).

**Pf.** (exchange argument)
- Consider a max cardinality independent set \( S \).
- If \( v = 5 \), we’re done.
- If \( u \in S \) and \( v \notin S \), then \( S \cup \{ v \} \) is independent \( \Rightarrow S \) not maximum.
- If \( u \in S \) and \( v \notin S \), then \( S \cup \{ v \} \setminus \{ u \} \) is independent.

Independent Set on Trees: Greedy Algorithm

**Theorem.** The following greedy algorithm finds a maximum cardinality independent set in forests (and hence trees).

\[
\text{Independent-Set-In-A-Forest}(F) \{ \\
S \leftarrow \emptyset \\
\text{while } (F \text{ has at least one edge) } \{ \\
\text{Let } e = (u, v) \text{ be an edge such that } v \text{ is a leaf} \\
\text{Add } v \text{ to } S \\
\text{Delete from } F \text{ nodes } u \text{ and } v, \text{ and all edges incident to them.} \\
\} \\
\text{return } S \\
\}
\]

**Pf.** Correctness follows from the previous key observation.

**Remark.** Can implement in \( O(n) \) time by considering nodes in postorder.

Weighted Independent Set on Trees

**Weighted independent set on trees.** Given a tree and node weights \( w_v > 0 \), find an independent set \( S \) that maximizes \( \sum v \in S w_v \).

**Observation.** If \(( u, v )\) is an edge such that \( v \) is a leaf node, then either \( \text{OPT} \) includes \( u \), or it includes all leaf nodes incident to \( u \).

**Dynamic programming solution.** Root tree at some node, say \( r \).
- \( \text{OPT}_{in}(u) = \max \text{ weight independent set of subtree rooted at } u, \text{ containing } u \).
- \( \text{OPT}_{out}(u) = \max \text{ weight independent set of subtree rooted at } u, \text{ not containing } u \).

\[
\begin{align*}
\text{OPT}_{in}(v) &= w_v + \sum_{v \in \text{children}(u)} \text{OPT}_{out}(v) \\
\text{OPT}_{out}(u) &= \sum_{v \in \text{children}(u)} \max(\text{OPT}_{in}(v), \text{OPT}_{out}(v))
\end{align*}
\]

10.2 Solving NP-Hard Problems on Trees
Theorem. The dynamic programming algorithm finds a maximum weighted independent set in a tree in O(n) time.

\[
\text{Weighted-Independent-Set-In-A-Tree}(T) \{
\text{Root the tree at node } r \\
\text{foreach (node } u \text{ of } T \text{ in postorder)} \{
\text{if } (u \text{ is a leaf)} \{ \\
M_{\text{in}}[u] = w_u \\
M_{\text{out}}[u] = 0 \\
\}
\text{else } \{ \\
M_{\text{in}}[u] = w_u + \sum_{v \text{ in children}(u)} M_{\text{out}}[v] \\
M_{\text{out}}[u] = \max_{v \text{ in children}(u)} \left( M_{\text{in}}[v], M_{\text{out}}[v] \right) \\
\}\}
\text{\\textbf{ensures a node is visited after all its children can also find independent set itself (not just value)}}
\]

\[\Box]\]

Context

Independent set on trees. This structured special case is tractable because we can find a node that breaks the communication among the subproblems in different subtrees.

Graphs of bounded tree width. Elegant generalization of trees that:
- Captures a rich class of graphs that arise in practice.
- Enables decomposition into independent pieces.

Wavelength-Division Multiplexing

Wavelength-division multiplexing (WDM). Allows m communication streams (arcs) to share a portion of a fiber optic cable, provided they are transmitted using different wavelengths.

Ring topology. Special case is when network is a cycle on n nodes.

Bad news. NP-complete, even on rings.

Brute force. Can determine if k colors suffice in O(k^n) time by trying all k-colorings.

Goal. O(f(k)): poly(m, n) on rings.

10.3 Circular Arc Coloring

Interval coloring. Greedy algorithm finds coloring such that number of colors equals depth of schedule.

Circular arc coloring.
- Weak duality: number of colors \( \geq \) depth.
- Strong duality does not hold.
(Almost) Transforming Circular Arc Coloring to Interval Coloring

Circular arc coloring. Given a set of n arcs with depth \( d \leq k \), can the arcs be colored with \( k \) colors?

Equivalent problem. Cut the network between nodes \( v_1 \) and \( v_n \). The arcs can be colored with \( k \) colors iff the intervals can be colored with \( k \) colors in such a way that "sliced" arcs have the same color.

Extra Slides

Circular Arc Coloring: Dynamic Programming Algorithm

Dynamic programming algorithm.
- Assign distinct color to each interval which begins at cut node \( v_0 \).
- At each node \( v_i \), some intervals may finish, and others may begin.
- Enumerate all \( k \)-colorings of the intervals through \( v_i \) that are consistent with the colorings of the intervals through \( v_{i-1} \).
- The arcs are \( k \)-colorable iff some coloring of intervals ending at cut node \( v_0 \) is consistent with original coloring of the same intervals.

Vertex Cover in Bipartite Graphs

Vertex Cover. Given an undirected graph \( G = (V, E) \), a vertex cover is a subset of vertices \( S \subseteq V \) such that for each edge \((u, v) \in E\), either \( u \in S \) or \( v \in S \) or both.

Circular Arc Coloring: Running Time

Running time. \( O(k! \cdot n) \).
- \( n \) phases of the algorithm.
- Bottleneck in each phase is enumerating all consistent colorings.
- There are at most \( k \) intervals through \( v_0 \), so there are at most \( k! \) colorings to consider.

Remark. This algorithm is practical for small values of \( k \) (say \( k = 10 \)) even if the number of nodes \( n \) (or paths) is large.
Vertex Cover

**Weak duality.** Let $M$ be a matching, and let $S$ be a vertex cover. Then, $|M| \leq |S|$.

**Pf.** Each vertex can cover at most one edge in any matching.

$M = 1-2', 3-1', 4-5'$

$|M| = 3$

$S* = \{3, 1', 2', 5\}$

$|S*| = 4$

$M* = 1-1', 2-2', 3-3', 5-5'$

$|M*| = 4$

König-Egerváry Theorem. In a bipartite graph, the max cardinality of a matching is equal to the min cardinality of a vertex cover.

- Sufficient to find matching $M$ and cover $S$ such that $|M| = |S|$.
- Formulate max flow problem as for bipartite matching.
- Let $M$ be max cardinality matching and let $(A, B)$ be min cut.
- Define $L_A = L \cap A$, $L_B = L \cap B$, $R_B = R \cap B$, $R_A = R \cap A$.

**Claim 1.** $S = L_B \cup R_A$ is a vertex cover.
- Consider $(u, v) \in E$.
- $u \in L_A$, $v \in R_B$ impossible since infinite capacity.
- Thus, either $u \in L_B$ or $v \in R_A$ or both.

**Claim 2.** $|S| = |M|$.
- max-flow min-cut theorem $\Rightarrow |M| = \text{cap}(A, B)$
- only edges of form $(s, u)$ or $(v, t)$ contribute to $\text{cap}(A, B)$
- $|M| = \text{cap}(A, B) = |L_A| + |R_A| = |S|$.

Register Allocation

**Register.** One of $k$ of high-speed memory locations in computer’s CPU.

**Register allocator.** Part of an optimizing compiler that controls which variables are saved in the registers as compiled program executes.

**Interference graph.** Nodes are “live ranges.” Edge $u-v$ if there exists an operation where both $u$ and $v$ are “live” at the same time.

**Observation.** ([Chaitin, 1982]) Can solve register allocation problem iff interference graph is $k$-colorable.

**Spilling.** If graph is not $k$-colorable (or we can’t find a $k$-coloring), we “spill” certain variables to main memory and swap back as needed.
A Useful Property

Remark. Register allocation problem is NP-hard.

Key fact. If a node \( v \) in graph \( G \) has fewer than \( k \) neighbors, \( G \) is \( k \)-colorable iff \( G - \{ v \} \) is \( k \)-colorable.

\[ \text{proof. Delete node } v \text{ from } G \text{ and color } G - \{ v \}. \]

\[ \text{If } G - \{ v \} \text{ is not } k \text{-colorable, then neither is } G. \]

\[ \text{If } G - \{ v \} \text{ is } k \text{-colorable, then there is at least one remaining color left for } v. \]

\[ \square \]

Chaitin’s Algorithm

\[ \text{Chaitin’s Algorithm} \]

\[ \text{Vertex-Color}(G, k) \{ \]

\[ \text{while } (G \text{ is not empty}) \{ \]

\[ \text{Pick a node } v \text{ with fewer than } k \text{ neighbors} \]

\[ \text{Push } v \text{ on stack} \]

\[ \text{Delete } v \text{ and all its incident edges} \]

\[ \text{while } (\text{stack is not empty}) \{ \]

\[ \text{Pop next node } v \text{ from the stack} \]

\[ \text{Assign } v \text{ a color different from its neighboring nodes which have already been colored} \]

\[ \}

3-Colorability

3-COLOR: Given an undirected graph \( G \) does there exists a way to color the nodes red, green, and blue so that no adjacent nodes have the same color?

Chaitin’s Algorithm

Theorem. [Kempe 1879, Chaitin 1982] Chaitin’s algorithm produces a \( k \)-coloring of any graph with max degree \( k \).

\[ \text{proof. Follows from key fact since each node has fewer than } k \text{ neighbors.} \]

Remark. If algorithm never encounters a graph where all nodes have degree \( \geq k \), then it produces a \( k \)-coloring.

Practice. Chaitin’s algorithm (and variants) are extremely effective and widely used in real compilers for register allocation.

Register Allocation

Register allocation. Assign program variables to machine register so that no more than \( k \) registers are used and no two program variables that are needed at the same time are assigned to the same register.

Interference graph. Nodes are program variables names, edge between \( u \) and \( v \) if there exists an operation where both \( u \) and \( v \) are "live" at the same time.

Observation. [Chaitin 1982] Can solve register allocation problem iff interference graph is \( k \)-colorable.

Fact. 3-COLOR \( \leq \_ \) \( k \)-REGISTER-ALLOCATION for any constant \( k \geq 3 \).
3-Colorability

Claim. 3-SAT ≤₃ 3-COLOR.

Pf. Given 3-SAT instance \( \Phi \), we construct an instance of 3-COLOR that is 3-colorable iff \( \Phi \) is satisfiable.

Construction.
\begin{enumerate}
\item For each literal, create a node.
\item Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
\item Connect each literal to its negation.
\item For each clause, add gadget of 6 nodes and 13 edges.
\end{enumerate}

Extra Slides
8.10 A Partial Taxonomy of Hard Problems

Polynomial-Time Reductions

Dick Karp (1972)
1985 Turing Award

SUBSET-SUM. Given natural numbers \( w_1, \ldots, w_n \) and an integer \( W \), is there a subset that adds up to exactly \( W \)?

PARTITION. Given natural numbers \( v_1, \ldots, v_m \), can they be partitioned into two subsets that add up to the same value?

\[ \frac{1}{2} \sum v_i \]

Claim. \( \text{SUBSET-SUM} \leq P \text{ PARTITION} \).

Pf. Let \( W, w_1, \ldots, w_n \) be an instance of \( \text{SUBSET-SUM} \).

- Create instance of \( \text{PARTITION} \) with \( m = n+2 \) elements:
  \[ v_i = \begin{cases} w_i & \text{if } 1 \leq i \leq n \\ 2 & \text{if } i = n+1 \\ \sum w_i - W & \text{if } i = n+2 \end{cases} \]

- There exists a subset that sums to \( W \) if and only if there exists a partition since two new elements cannot be in the same partition.

\[ \begin{array}{c|c}
\text{subset A} & W \\
\hline
\text{subset B} & \sum w_i - W \\
\end{array} \]

4 Color Theorem

Planar 3-Colorability

PLANAR-3-COLOR. Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

YES instance.
Planar 3-Colorability

**PLANAR-3-COLOR.** Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?

**NO instance.**

Planarity

- **Def.** A graph is planar if it can be embedded in the plane in such a way that no two edges cross.
- **Applications:** VLSI circuit design, computer graphics.

Kuratowski’s Theorem. An undirected graph $G$ is non-planar iff it contains a subgraph homeomorphic to $K_5$ or $K_{3,3}$.

Planarity Testing

- **Planarity testing.** [Hopcroft-Tarjan 1974] $O(n)$.
- A simple planar graph can have at most 3n - 6 vertices.

Remark. Many intractable graph problems can be solved in poly-time if the graph is planar; many tractable graph problems can be solved faster if the graph is planar.

Planar Graph 3-Colorability

- **Claim.** PLANAR-3-COLOR ≤ₚ PLANAR-GRAPH-3-COLOR.
- **Pf sketch.** Create a vertex for each region, and an edge between regions that share a nontrivial border.

Planar Graph 3-Colorability

- **Claim.** $W$ is a planar graph such that:
  - In any 3-coloring of $W$, opposite corners have the same color.
  - Any assignment of colors to the corners in which opposite corners have the same color extends to a 3-coloring of $W$.
- **Pf.** Only 3-colorings of $W$ are shown below (or by permuting colors).
Planar Graph 3-Colorability

Claim. \(3\text{-COLOR} \leq \text{PLANAR-GRAPH-3-COLOR}\).

Pf. Given instance of 3-COLOR, draw graph in plane, letting edges cross.
- Replace each edge crossing with planar gadget \(W\).
- In any 3-coloring of \(W\), \(a \neq a'\) and \(b \neq b'\).
- If \(a \neq a'\) and \(b \neq b'\) then can extend to a 3-coloring of \(W\).

Multiple crossings

Gadget \(W\)

Planar k-Colorability

**Planar-2-Color.** Solvable in linear time.

**Planar-3-Color.** \(NP\)-complete.

**Planar-4-Color.** Solvable in \(O(1)\) time.

**Theorem.** [Appel-Haken, 1976] Every planar map is 4-colorable.
- Resolved century-old open problem.
- Used 50 days of computer time to deal with many special cases.
- First major theorem to be proved using computer.

**False intuition.** If \(PLANAR-3\text{-COLOR}\) is hard, then so is \(PLANAR-4\text{-COLOR}\) and \(PLANAR-5\text{-COLOR}\).

Polynomial-Time Detour

**Graph minor theorem.** [Robertson-Seymour 1980s]

Corollary. There exist an \(O(n^3)\) algorithm to determine if a graph can be embedded in the torus in such a way that no two edges cross.

**Pf of theorem.** Tour de force.

**Mind boggling fact 1.** The proof is highly non-constructive!

**Mind boggling fact 2.** The constant of proportionality is enormous!

Unfortunately, for any instance \(G = (V, E)\) that one could fit into the known universe, one would easily prefer \(n^70\) to even \(constant\) time, if that constant had to be one of Robertson and Seymour's. - David Johnson

**Theorem.** There exists an explicit \(O(n)\) algorithm.
**Practice.** LEDA implementation guarantees \(O(n^7)\).