CS 580: Algorithm Design and Analysis

Jeremiah Blocki
Purdue University
Spring 2019

Homework 5. Due Tonight at 11:59 PM (on Gradescope)

Midterm 2. April 3 @ 8PM (EE 170)
Practice Midterm Released
3x5 Index Card (Double Sided)
Schedule Change: *Class canceled on April 4 (We will have class on April 25th)*

Midterm 2

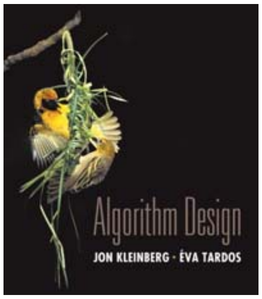
- When?
  - April 3rd from 8PM to 10PM (2 hours)
- Where?
  - EE 170
- What can I bring?
  - 3x5 inch index card with your notes (double sided)
  - No electronics (phones, computers, calculators etc…)
- What material should I study?
  - The midterm will cover recent topics more heavily
    - Network 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

2

Midterm 2

- When?
  - April 3rd from 8PM to 10PM (2 hours)
- Where?
  - EE 170
- 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)

3



Extending the Limits of Tractability

Slides by Kevin Wayne.

4

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.

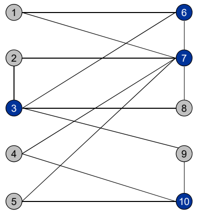
This lecture. Solve some special cases of NP-complete problems that arise in practice.

5

10.1 Finding Small Vertex Covers

Vertex Cover

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. $O(k\,n^{k+1})$.

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

Goal. Limit exponential dependency on k, e.g., to $O(2^k\,k\,n)$.

Ex. n = 1,000, k = 10.
Brute. $k\,n^{k+1} = 10^{34}$ ⇒ infeasible.
Better. $2^k\,k\,n = 10^7$ ⇒ 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.

delete v and all incident edges

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.

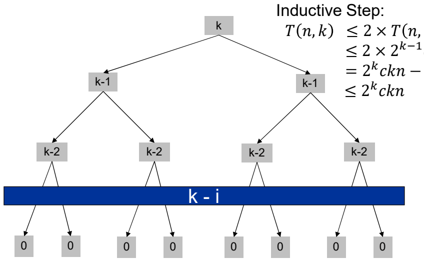
- Correctness follows from previous two claims.
- There are ≤ 2^{k+1} nodes in the recursion tree; each invocation takes O(kn) time. ▪

Finding Small Vertex Covers: Recursion Tree

$$T(n,k) \le \begin{cases} c & \text{if } k = 0 \\ cn & \text{if } k = 1 \\ 2T(n,k-1) + ckn & \text{if } k > 1 \end{cases} \quad \Rightarrow \quad T(n,k) \le 2^k c k n$$

Inductive Step:

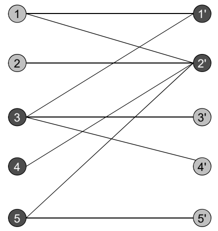
$$\begin{aligned} T(n,k) &\le 2 \times T(n,k-1) + ckn \\ &\le 2 \times 2^{k-1} c(k-1)n + ckn \\ &= 2^k ckn - 2^k cn + ckn \\ &\le 2^k ckn \end{aligned}$$



Vertex Cover in Bipartite Graphs

Vertex Cover

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

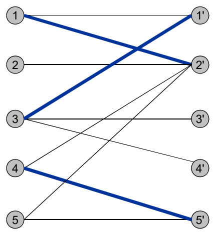


S = { 3, 4, 5, 1', 2' }
|S| = 5

Vertex Cover

Weak duality. Let M be a matching, and let S be a vertex cover. Then, |M| ≤ |S|.

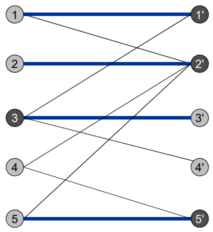
Pf. Each vertex can cover at most one edge in any matching.



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

Vertex Cover: König-Egerváry Theorem

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



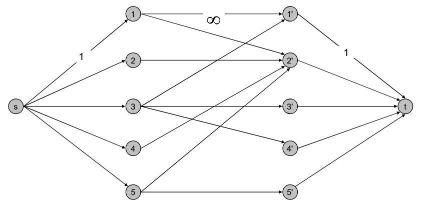
S* = { 3, 1', 2', 5'}
|S*| = 4

M* = 1-1', 2-2', 3-3', 5-5'
|M*| = 4

Vertex Cover: Proof of König-Egerváry Theorem

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

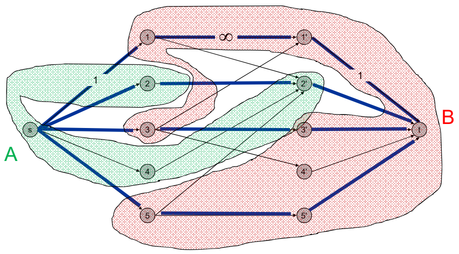
- Suffices to find matching M and cover S such that |M| = |S|.
- Formulate max flow problem as for bipartite matching.
- Let M be max cardinality matching and let (A, B) be min cut.



Vertex Cover: Proof of König-Egerváry Theorem

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

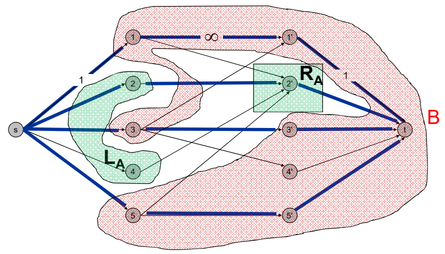
- Suffices to find matching M and cover S such that |M| = |S|.
- Formulate max flow problem as for bipartite matching.
- Let M be max cardinality matching and let (A, B) be min cut.



Vertex Cover: Proof of König-Egerváry Theorem

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

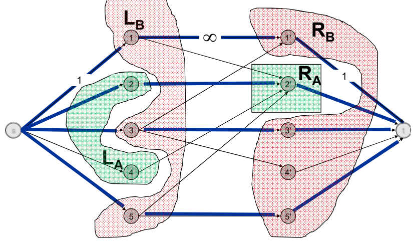
- Suffices to find matching M and cover S such that |M| = |S|.
- Formulate max flow problem as for bipartite matching.
- Let M be max cardinality matching and let (A, B) be min cut.



Vertex Cover: Proof of König-Egerváry Theorem

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

- Suffices 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.

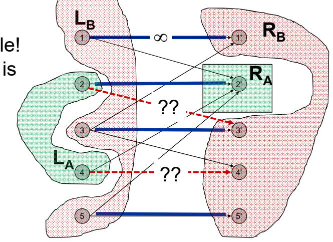


Vertex Cover: Proof of König-Egerváry Theorem

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

- Suffices 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.

Impossible!
(Min-Cut is
finite)

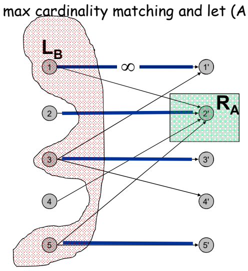


Vertex Cover: Proof of König-Egerváry Theorem

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

- Suffices 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.

$R_A \cup L_B$
vertex cover
of size |M|=4



Vertex Cover: Proof of König-Egerváry Theorem

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

- Suffices 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 $L_A = L \cap A$, $L_B = L \cap B$, $R_A = R \cap A$, $R_B = R \cap B$.

- 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| = cap(A, B)
  - only edges of form (s, u) or (v, t) contribute to cap(A, B)
  - If $u \in L_A \subset A$ then (s,u) contributes 0 to cap(A,B)
  - If $v \in R_B \subset B$ then (v,t) contributes 0 to cap(A,B)
  - $|M| = \text{cap}(A, B) = |L_B| + |R_A| = |S|$. ▪

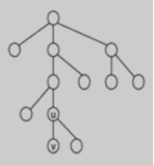
10.2 Solving NP-Hard Problems on Trees

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.

degree = 1



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 $v \in S$, we're done.
- If $u \notin S$ and $v \notin S$, then $S \cup \{v\}$ is independent $\Rightarrow$ S not maximum.
- If $u \in S$ and $v \notin S$, then $S \cup \{v\} - \{u\}$ is independent.
  ▪

Independent Set on Trees: Greedy Algorithm

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

```
Independent-Set-In-A-Forest(F) {
  $S \leftarrow \phi$
  while (F has at least one edge) {
    Let $e = (u, v)$ be an edge such that v is a leaf
    Add v to S
    Delete from F nodes u and v, and all edges
         incident to them.
  }
  return $S \cup V(F)$  //S and all remaining nodes in F
}
```

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 $\Sigma_{v \in S}\, w_v$.

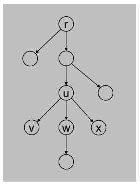
Observation. If (u, v) is an edge such that v is a leaf node, then either OPT includes u, or it includes all leaf nodes incident to u.

Dynamic programming solution. Root tree at some node, say r.

- OPT_{in} (u) = max weight independent set of subtree rooted at u, containing u.
- OPT_{out}(u) = max weight independent set of subtree rooted at u, not containing u.

$$OPT_{in}(u) = w_u + \sum_{v \in \text{children}(u)} OPT_{out}(v)$$

$$OPT_{out}(u) = \sum_{v \in \text{children}(u)} \max\,\{OPT_{in}(v),\ OPT_{out}(v)\}$$



children(u) = { v, w, x }

Weighted Independent Set on Trees: Dynamic Programming Algorithm

Theorem. The dynamic programming algorithm finds a maximum weighted independent set in a tree in O(n) time.

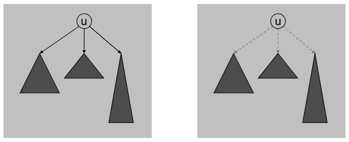
```
Weighted-Independent-Set-In-A-Tree(T) {
   Root the tree at a node r
   foreach (node u of T in postorder) {
      if (u is a leaf) {
         $M_{in}$ [u] = $w_u$
         $M_{out}$[u] = 0
      }
      else {
         $M_{in}$ [u] = $w_u$ + $\Sigma_{v \in children(u)}$ $M_{out}$[v]
         $M_{out}$[u] = $\Sigma_{v \in children(u)}$ max($M_{in}$[v], $M_{out}$[v])
      }
   }
   return max($M_{in}$[r], $M_{out}$[r])
}
```

ensures a node is visited after all its children

Pf. Takes O(n) time since we visit nodes in postorder and examine each edge exactly once. ▪ ← can also find independent set itself (not just value)

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.



see Chapter 10.4, but proceed with caution

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.

Register Allocation

Register Allocation

Register. One of k of high-speed memory locations in computer's CPU.

say 32

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

variables or temporaries

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.

typically infrequently used variables that are not in inner loops

A Useful Property

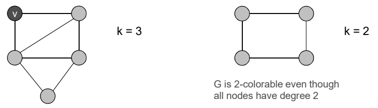
Remark. Register allocation problem is NP-hard.

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

delete v and all incident edges

Pf. Delete node v from G and color G – { v }.

- If G – { v } is not k-colorable, then neither is G.
- If G – { v } is k-colorable, then there is at least one remaining color left for v. ▪



G is 2-colorable even though all nodes have degree 2

Chaitin's Algorithm

```
Vertex-Color(G, k) {
   while (G is not empty) {
      Pick a node v with fewer than k neighbors
      Push v on stack
      Delete v and all its incident edges
   }
   while (stack is not empty) {
      Pop next node v from the stack
      Assign v a color different from its neighboring nodes which have already been colored
   }
}
```

say, node with fewest neighbors

Chaitin's Algorithm

Theorem. [Kempe 1879, Chaitin 1982] Chaitin's algorithm produces a k-coloring of any graph with max degree k-1.
Pf. Follows from key fact since each node has fewer than k neighbors.

algorithm succeeds in k-coloring many graphs with max degree ≥ k

Remark. If algorithm never encounters a graph where all nodes have degree ≥ 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.

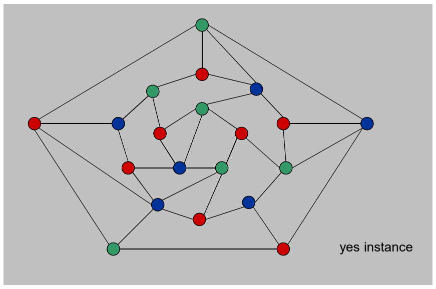
8.7 Graph Coloring

Basic genres.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

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?



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 $\le_P$ k-REGISTER-ALLOCATION for any constant k ≥ 3.

3-Colorability

Claim. 3-SAT $\le_P$ 3-COLOR.

Pf. Given 3-SAT instance Φ, we construct an instance of 3-COLOR that is 3-colorable iff Φ is satisfiable.

Construction.

i. For each literal, create a node.
ii. Create 3 new nodes T, F, B; connect them in a triangle, and connect each literal to B.
iii. Connect each literal to its negation.
iv. For each clause, add gadget of 6 nodes and 13 edges.

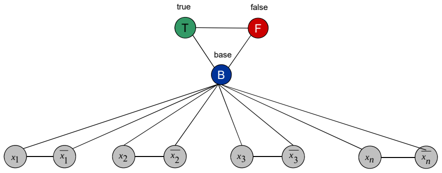
↑ to be described next

3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.



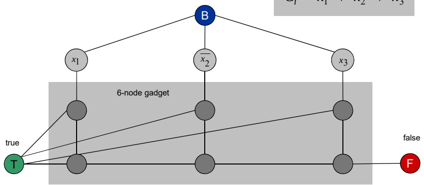
3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

$C_i = x_1 \vee \overline{x_2} \vee x_3$



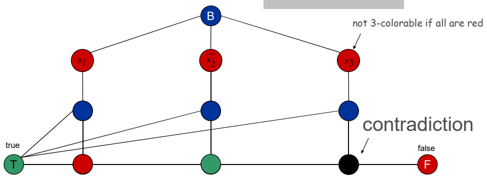
3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Rightarrow$ Suppose graph is 3-colorable.

- Consider assignment that sets all T literals to true.
- (ii) ensures each literal is T or F.
- (iii) ensures a literal and its negation are opposites.
- (iv) ensures at least one literal in each clause is T.

$C_i = x_1 \vee \overline{x_2} \vee x_3$



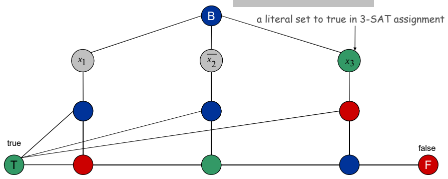
3-Colorability

Claim. Graph is 3-colorable iff Φ is satisfiable.

Pf. $\Leftarrow$ Suppose 3-SAT formula Φ is satisfiable.

- Color all true literals T.
- Color node below green node F, and node below that B.
- Color remaining middle row nodes B.
- Color remaining bottom nodes T or F as forced. ▪

$C_i = x_1 \vee \overline{x_2} \vee x_3$



10.3 Circular Arc Coloring

Wavelength-Division Multiplexing

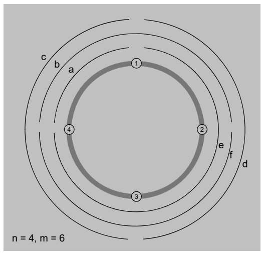
Wavelength-division multiplexing (WDM). Allows m communication streams (arcs) to share a portion of a fiber optic cable, provided they are transmitted using different wavelengths.

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 $O(k^m)$ time by trying all k-colorings.

Goal. $O(f(k)) \cdot poly(m, n)$ on rings.



Wavelength-Division Multiplexing

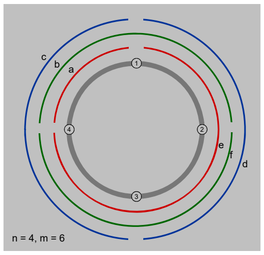
Wavelength-division multiplexing (WDM). Allows m communication streams (arcs) to share a portion of a fiber optic cable, provided they are transmitted using different wavelengths.

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

Bad news. NP-complete, even on rings.

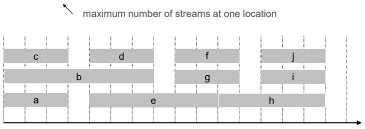
Brute force. Can determine if k colors suffice in $O(k^m)$ time by trying all k-colorings.

Goal. $O(f(k)) \cdot poly(m, n)$ on rings.



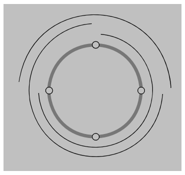
Review: Interval Coloring

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



Circular arc coloring.

- Weak duality: number of colors ≥ depth.
- Strong duality does not hold.

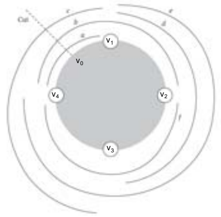


max depth = 2
min colors = 3

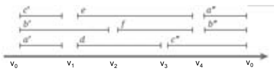
(Almost) Transforming Circular Arc Coloring to Interval Coloring

Circular arc coloring. Given a set of n arcs with depth d ≤ 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.



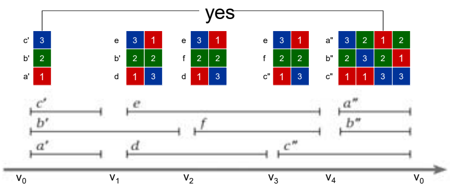
colors of a', b', and c' must correspond to colors of a", b", and c"



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.



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_i, so there are at most k! colorings to consider.

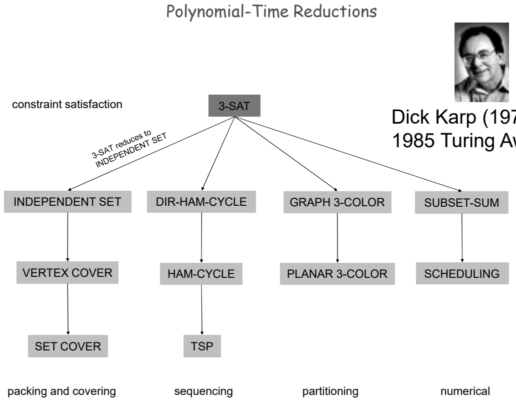
Remark. This algorithm is practical for small values of k (say k = 10) even if the number of nodes n (or paths) is large.

Extra Slides

Extra Slides

8.10 A Partial Taxonomy of Hard Problems

Polynomial-Time Reductions


constraint satisfaction

3-SAT

3-SAT reduces to INDEPENDENT SET

INDEPENDENT SET

DIR-HAM-CYCLE

GRAPH 3-COLOR

SUBSET-SUM

VERTEX COVER

HAM-CYCLE

PLANAR 3-COLOR

SCHEDULING

SET COVER

TSP

packing and covering

sequencing

partitioning

numerical

Dick Karp (1972)
1985 Turing Award

Subset Sum (proof from book)

Construction. Let $X \cup Y \cup Z$ be a instance of 3D-MATCHING with triplet set T. Let n = |X| = |Y| = |Z| and m = |T|.

- Let $X = \{ x_1, x_2, x_3 \ x_4 \}$, $Y = \{ y_1, y_2, y_3, y_4 \}$, $Z = \{ z_1, z_2, z_3, z_4 \}$
- For each triplet t= $(x_i, y_j, z_k) \in T$, create an integer w_t with 3n digits that has a 1 in positions i, n+j, and 2n+k.

use base m+1

Claim. 3D-matching iff some subset sums to W = 111,…, 111.

Triplet t_i			x_1	x_2	x_3	x_4	y_1	y_2	y_3	y_4	z_1	z_2	z_3	z_4	w_i
x_1	y_2	z_3	1	0	0	0	0	1	0	0	0	0	1	0	100,001,000,010
x_2	y_4	z_2	0	1	0	0	0	0	0	1	0	1	0	0	10,000,010,100
x_1	y_1	z_1	1	0	0	0	1	0	0	0	1	0	0	0	100,010,001,000
x_2	y_2	z_4	0	1	0	0	0	1	0	0	0	0	0	1	10,001,000,001
x_4	y_3	z_4	0	0	0	1	0	0	1	0	0	0	0	1	100,100,001
x_3	y_1	z_2	0	0	1	0	1	0	0	0	0	1	0	0	1,010,000,100
x_3	y_1	z_3	0	0	1	0	1	0	0	0	0	0	1	0	1,010,000,010
x_3	y_1	z_1	0	0	1	0	1	0	0	0	1	0	0	0	1,010,001,000
x_4	y_4	z_4	0	0	0	1	0	0	0	1	0	0	0	1	100,010,001
															111,111,111,111

Partition

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

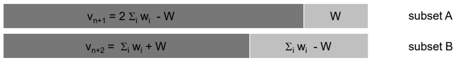
PARTITION. Given natural numbers $v_1, \ldots, v_m$, can they be partitioned into two subsets that add up to the same value?

½ $\Sigma_i \, v_i$

Claim. SUBSET-SUM $\le_P$ PARTITION.

Pf. Let $W, w_1, \ldots, w_n$ be an instance of SUBSET-SUM.

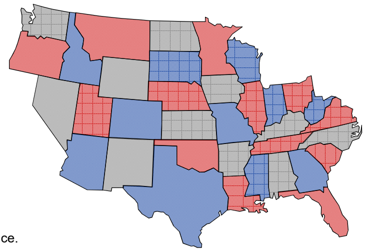
- Create instance of PARTITION with m = n+2 elements.
  - $v_1 = w_1, v_2 = w_2, \ldots, v_n = w_n, \quad v_{n+1} = 2 \, \Sigma_i \, w_i - W, \quad v_{n+2} = \Sigma_i \, w_i + W$
- There exists a subset that sums to W iff there exists a partition since two new elements cannot be in the same partition. ▪


$v_{n+1} = 2 \, \Sigma_i \, w_i - W$
W
subset A
$v_{n+2} = \Sigma_i \, w_i + W$
$\Sigma_i \, w_i - W$
subset B

4 Color Theorem

Planar 3-Colorability

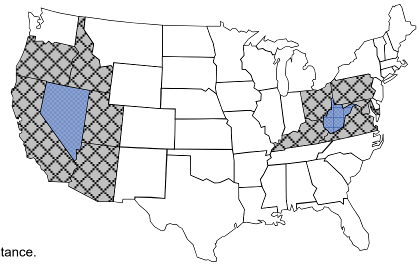
PLANAR-3-COLOR. Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?



YES instance.

Planar 3-Colorability

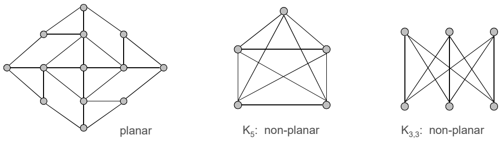
PLANAR-3-COLOR. Given a planar map, can it be colored using 3 colors so that no adjacent regions have the same color?



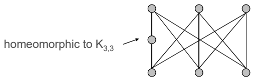
NO instance.

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

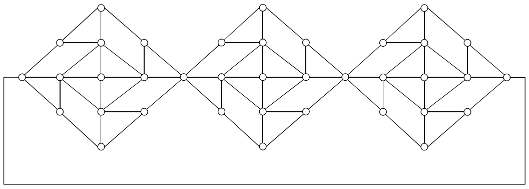
Planarity testing. [Hopcroft-Tarjan 1974] O(n).

simple planar graph can have at most 3n edges

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

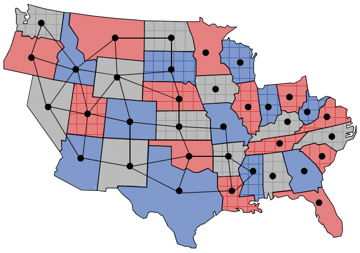
Q. Is this planar graph 3-colorable?



Planar 3-Colorability and Graph 3-Colorability

Claim. PLANAR-3-COLOR $\leq_P$ PLANAR-GRAPH-3-COLOR.

Pf sketch. Create a vertex for each region, and an edge between regions that share a nontrivial border.

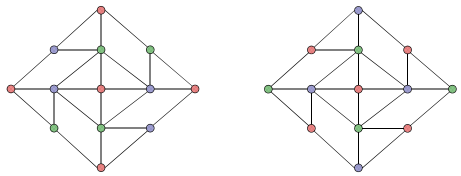


Planar Graph 3-Colorability

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).

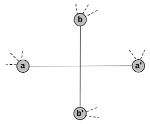


Planar Graph 3-Colorability

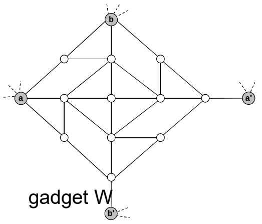
Claim. 3-COLOR $\leq_P$ PLANAR-GRAPH-3-COLOR.

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 ≠ a' and b ≠ b'.
- If a ≠ a' and b ≠ b' then can extend to a 3-coloring of W.



a crossing



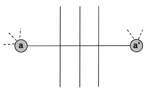
gadget W

Planar Graph 3-Colorability

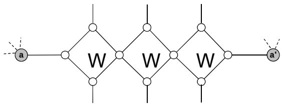
Claim. 3-COLOR $\leq_P$ PLANAR-GRAPH-3-COLOR.

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 ≠ a' and b ≠ b'.
- If a ≠ a' and b ≠ b' then can extend to a 3-coloring of W.



multiple crossings



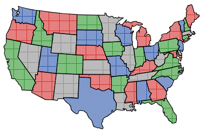
gadget W

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.

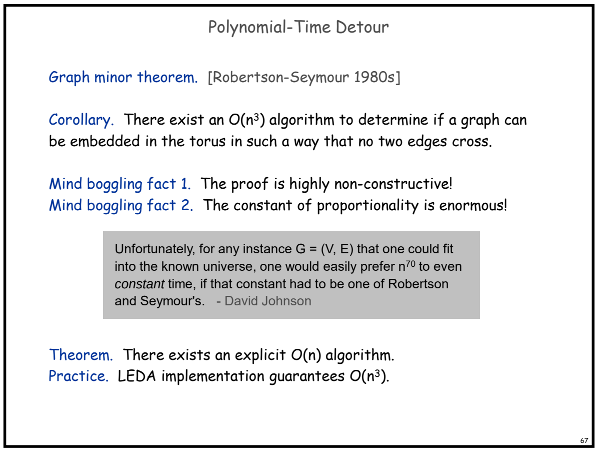
False intuition. If PLANAR-3-COLOR is hard, then so is PLANAR-4-COLOR and PLANAR-5-COLOR.

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.



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.

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 $O(n^3)$.

67