Today:

- NP-Completeness wrapup
- Brief insights into approximation algs

COSC 581, Algorithms

April 24, 2014

Many of these slides are adapted from several online sources

Announcement

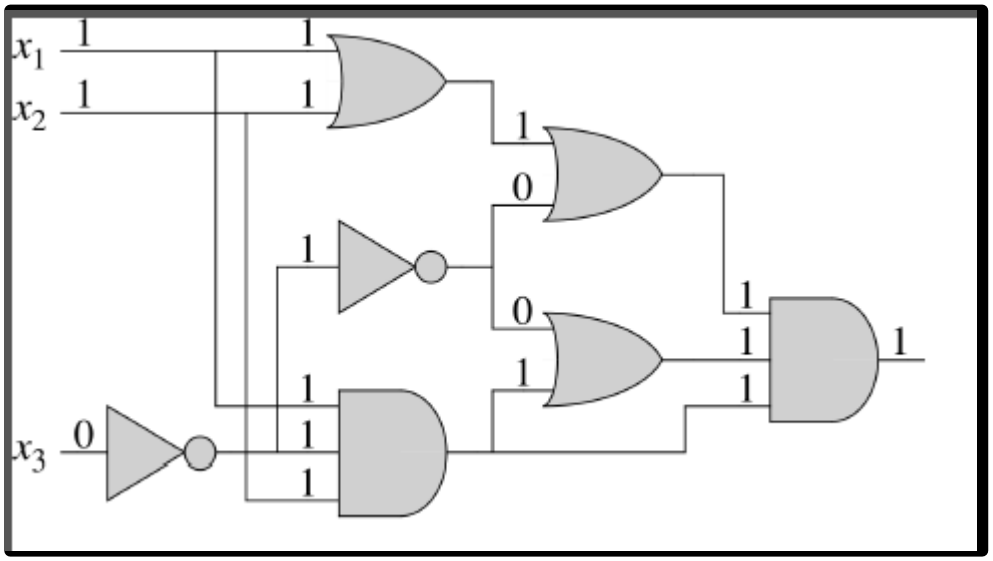
- Final Exam:
  - Wednesday, April 30, 10:15AM – 12:15PM
  - This room (MK 525)
  - Exam Content:
    - 75% of exam will be on multithreaded algorithms, linear programming, NP-completeness
    - 25% of exam will be comprehensive

Reading Assignments

- Today's class:
  - Chapter 34.3
  - Chapter 35.1 (quick look)

Need to show a "First" NP-complete problem: Circuit Satisfiability

- *Circuit-SAT problem:* Given a Boolean combinational circuit, determine if there is a satisfying assignment to inputs such that the circuit's output is 1.



Example circuit with satisfying assignment

Implication: In the area of computer-aided hardware optimization, if a subcircuit always produces 0, then the subcircuit can be replaced by a simpler subcircuit that omits all gates and just outputs a 0.

Circuit-satisfiability problem is NP-complete

- We can show that CIRCUIT-SAT $\in$ NP.
- Proof: CIRCUIT-SAT is poly-time verifiable.
  - Given: (an encoding of) a CIRCUIT-SAT problem C and a certificate, which is an assignment of boolean values to (all) wires in C.
  - Verification algorithm: check each gate and output wire of C:
    - If for every gate, the computed output value matches the value of the output wire given in the certificate and the output of the whole circuit is 1, then the algorithm outputs 1, otherwise 0.
  - This algorithm is executed in linear (i.e., polynomial) time.

Circuit-satisfiability problem is NP-complete (cont.)

- For the "first" NP-complete problem, we must show:
  - A problem B (i.e., Circuit-Sat) is **NP-complete** if:

    (1) Circuit-Sat $\in$ **NP** //We just showed this

    (2) X $\leq_p$ Circuit-Sat, for all X $\in$ **NP**

- How to do (2)?
  - Construct a poly-time algorithm *f* that maps every problem instance *x* in X to a circuit C=*f*(*x*) such that the answer to *x* is YES if and only if C$\in$CIRCUIT-SAT (i.e., C is satisfiable).

Circuit-satisfiability problem is NP-hard (cont.)

- Since $X \in NP$, there is a poly-time algorithm A that verifies X.
- Suppose the input length is n.
- Let $T(n)$ denote the worst-case running time.
- Let k be the constant such that $T(n)=O(n^k)$ and the length of the certificate is $O(n^k)$.

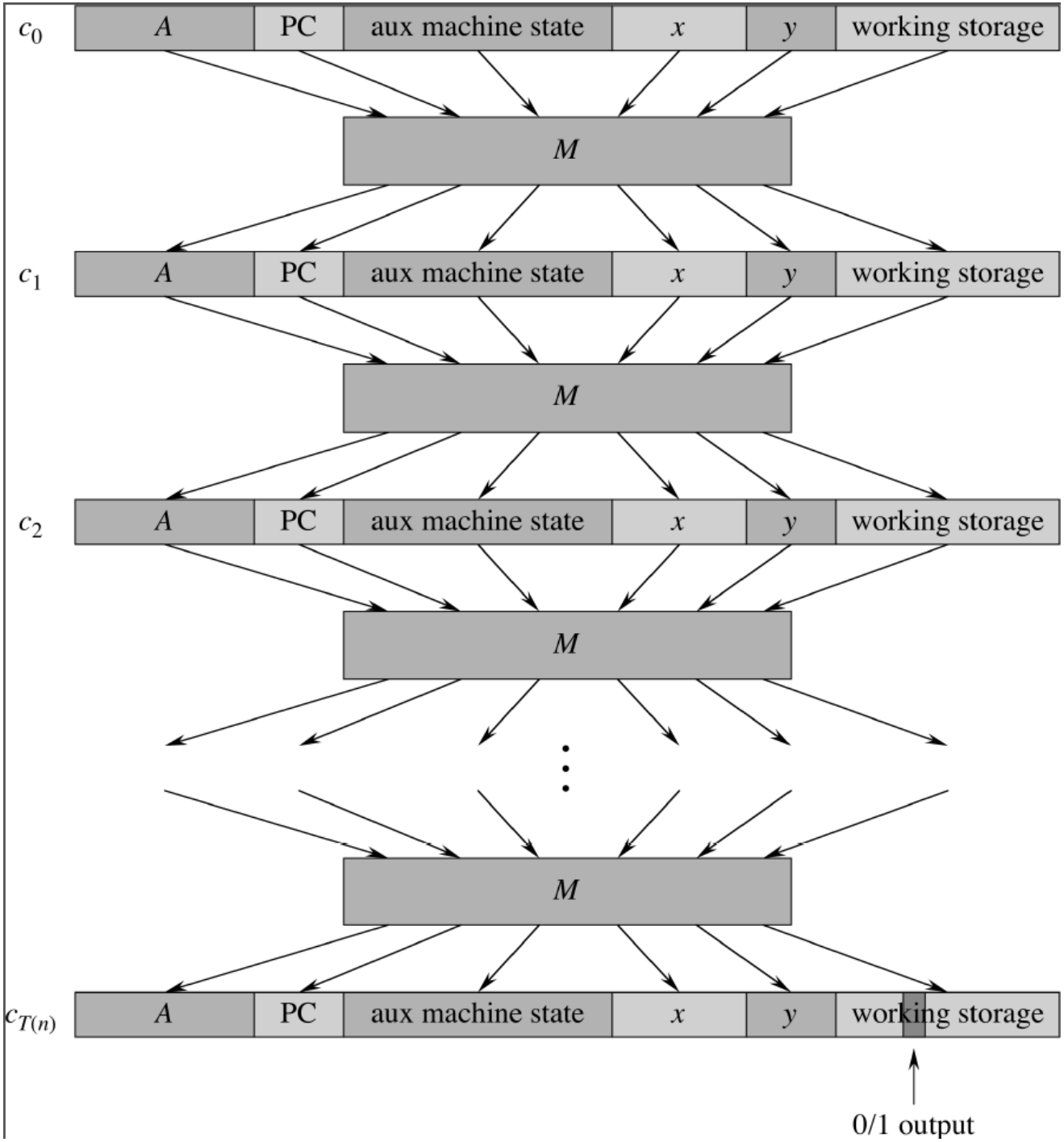
Circuit-satisfiability problem is NP-hard (cont.)

- Idea is to represent the computation of A as a sequence of configurations, $c_0, c_1,\ldots,c_i,c_{i+1},\ldots,c_{T(n)}$
- *E*ach c_i can be broken into:
  - (program for A, program counter PC, auxiliary machine state, input x, certificate y, working storage) and
  - c_i is mapped to c_{i+1} by the combinational circuit M implementing the computer hardware.
  - The output of A *(i.e.,* 0 or 1) is written to some designated location in working storage.
  - If the algorithm runs for at most $T(n)$ steps, the output appears as one bit in $c_{T(n)}$.
  - Note: $A(x,y)$=1 or 0.

Circuit-satisfiability problem is NP-hard (cont.)

- The reduction algorithm F constructs a single combinational circuit C as follows:
  - Paste together all $T(n)$ copies of the circuit M.
  - The output of the i^{th} circuit, which produces c_i, is directly fed into the input of the $(i+1)^{st}$ circuit.
  - All items in the initial configuration, except the bits corresponding to certificate y, are wired directly to their known values.
  - The bits corresponding to y are the inputs to C.
  - All the outputs to the circuit are ignored, except the one bit of $c_{T(n)}$ corresponding to the output of A.

Sequence of configurations produced by A when running on input x and certificate y



Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:
  1) F correctly constructs the reduction, i.e., C is satisfiable if and only if there exists a certificate y, such that $A(x,y)=1$.
     - $\Leftarrow$ Suppose there is a certificate y, such that $A(x,y)=1$. Then if we apply the bits of y to the inputs of C, the output of C is the bit of $A(x,y)$, that is $C(y)= A(x,y) =1$, so C is satisfiable.
     - $\Rightarrow$ Suppose C is satisfiable, then there is a y such that $C(y)=1$. So, $A(x,y)=1$.

Circuit-satisfiability problem is NP-hard (cont.)

2) F runs in poly time:
   - Poly space:
     - Size of x is n.
     - Size of A is constant, independent of x.
     - Size of y is $O(n^k)$.
     - Amount of working storage is poly in n since A runs at most $O(n^k)$.
     - M has size poly in length of configuration, which is poly in $O(n^k)$, and hence is poly in n.
     - C consists of at most $O(n^k)$ copies of M, and hence is poly in n.
     - Thus, the C has poly space.
   - The construction of C takes at most $O(n^k)$ steps and each step takes poly time, so F takes poly time to construct C from x.

CIRCUIT-SAT is NP-complete

- In summary
  - CIRCUIT-SAT belongs to NP, verifiable in poly time.
  - CIRCUIT-SAT is NP-hard, since every NP problem can be reduced to CIRCUIT-SAT in poly time.
  - Thus CIRCUIT-SAT is NP-complete.

Coping With NP-Hardness

Brute-force algorithms:

- Develop clever enumeration strategies
- Guaranteed to find optimal solution
- No guarantees on running time

Heuristics:

- Develop intuitive algorithms
- Guaranteed to run in polynomial time
- No guarantees on quality of solution

Approximation algorithms:

- Guaranteed to run in polynomial time.
- Guaranteed to find "high quality" solution, say within 1% of optimum
- Obstacle: need to prove a solution's value is close to optimum, without even knowing what the optimum value is!

Approximation Algorithms

- Suppose we want to find a *minimum* cost solution to some problem (e.g., smallest vertex cover, minimum cost tour of a graph, etc.)
- We define the relative cost of the solution S to be

  $$\text{Cost}(S)/\text{Cost}(S^*)$$

  where S^* is the best solution.
- An algorithm has *ratio r* if it always returns a solution with relative cost at most r.
- An algorithm that has ratio r is called an "r-approximation algorithm"

Approximation Algorithms and Schemes

ρ-approximation algorithm:

- An algorithm A for problem X that runs in polynomial time
- For every problem instance, A outputs a feasible solution within ratio ρ of true optimum for that instance

Polynomial-time approximation scheme (PTAS).

- A family of approximation algorithms $\{A_\varepsilon : \varepsilon > 0\}$ for a problem X
- A_ε is a $(1 + \varepsilon)$ - approximation algorithm for X
- A_ε runs in time polynomial in the input size for a fixed ε

Fully polynomial-time approximation scheme (FPTAS)

- PTAS where A_ε is runs in time polynomial in input size and $1 / \varepsilon$.

Lots of Approximation Algs

Examples:

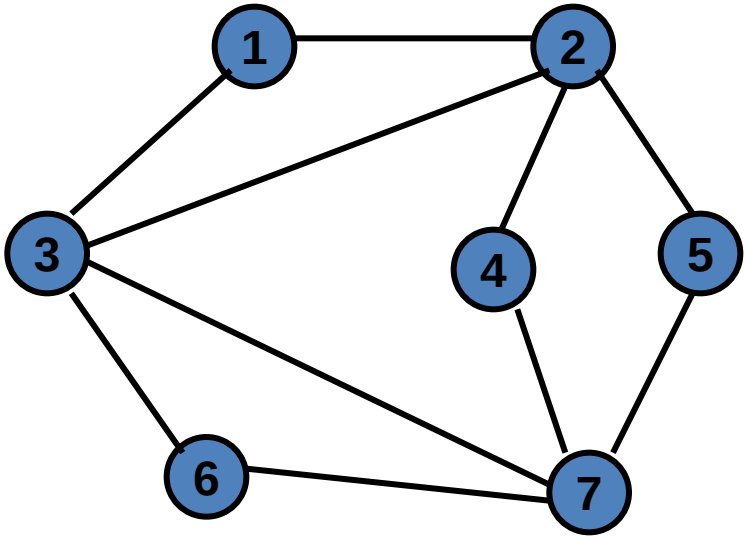
- Vertex Cover (finding a set of vertices that include at least one endpoint of every edge)
- Traveling Salesman (finding a minimum cost tour in an edge-weighted graph)
- Set-covering (finding subset of sets that covers given elements)
- Max-3-CNF Sat (finding max number of clauses that are satisfiable)
- Subset-Sum (finding subset of S that gets as close to t as possible without being greater than t)

Vertex Cover

For graph G = (V, E), a subset S of the vertices is a vertex cover if every edge in E has at least one endpoint in S.

S = {2,3,7} is a vertex cover

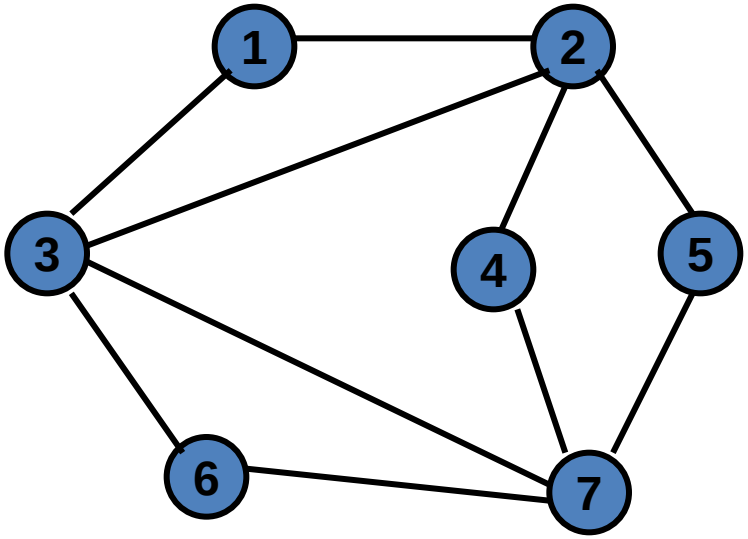
S = {1,2,3,4,5} is not a vertex cover



Minimum Vertex Cover

{2,3,6,7} is a minimum vertex cover for this graph.

We already know that finding a minimum vertex cover is NP-hard



2-approximation to Vertex Cover

APPROX-VERTEX-COVER(G)

C = ∅

E’ = G.E

while E’ ≠ ∅

let (*u*,*v*) be an arbitrary edge of E’

C = C ∪{*u*,*v*}

remove from E’ every edge incident on either *u* or *v*

return C

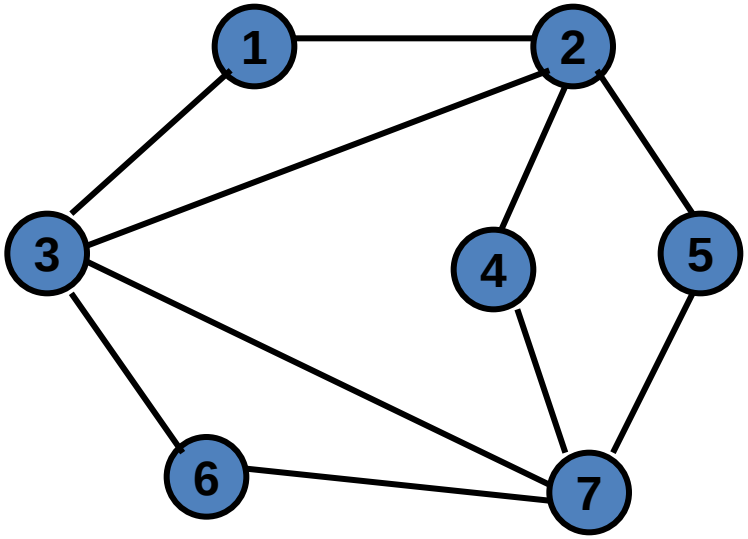
Example: Approx. Vertex Cover

Pick edge (1, 2). C = {1, 2}

Pick edge (3, 6). C = {1, 2, 3, 6}

Pick edge (4, 7). C = {1, 2, 3, 6, 4, 7}

The minimum vertex cover is {2,3,7}, and has 3 vertices.



Proof that APPROX-VC is 2-approx. alg

```
APPROX-VERTEX-COVER(G)
    C = ∅
    E' = G.E
    while E' ≠ ∅
            let (u,v) be an arbitrary edge of E'
            C = C ∪{u,v}
            remove from E' every edge incident
    on either u or v
    return C
```

S is a VC: The set C of vertices returned must be a VC, since the alg. loops until every edge has been covered by some vertex in C.

$|C| \leq 2|C^*|$: Let A be the set of edges that line 4 of algorithm picked. In order to cover the edges in A, any VC must include at least 1 endpoint of each edge in A. No two edges in A share an endpoint. Thus, no 2 edges in A are covered by the same vertex from C*. This implies that $|C^*| \geq |A|$. The size of the VC cover returned is $|C| = 2|A|$. Combining, we get: $|C| = 2|A| \leq 2|C^*|$. Thus, Approx-Vertex-Cover is a 2-approximation alg.

Lots of techniques for Approx. Algs

- The vertex cover approximation gives you an idea of how the approach would work.

Some top journals in algorithms

- *SIAM Journal on Computing*
- *Automatica*
- *Theoretical Computer Science*
- *Journal of Computer and System Sciences*
- *Algorithmica*
- *Journal of Algorithms*
- *Discrete Applied Mathematics*

Some top conferences in algorithms

- ACM Symposium on Theory of Computing
- IEEE Symposium on Foundations of CS
- ACM-SIAM Symposium on Discrete Algorithms
- International Colloquium on Automata, Languages, and Programming
- Symposium on Theoretical Aspects of CS
- Mathematical Foundations of CS

Example titles of current (2013-2014) research in *SIAM Journal of Computing*

- "On a linear program for minimum-weight triangulation"
- "Distributed (δ+1)-coloring in linear (δ) time"
- "New lower bounds for the rank of matrix multiplication"
- "Deterministic algorithms for the Lovasz local lemma"
- "On the insertion time of cuckoo hashing"
- "On the power of randomization in algorithmic mechanism design"
- "Adding one edge to planar graphs makes crossing number and 1-planarity hard"
- "Jaywalking your dog: Computing the Frechet distance with shortcuts"
- "An algebraic theory of complexity for discrete optimization"
- "On the complexity of package recommendation problems"
- "On the NP-hardness of Max-Not-2"
- "Characterizing truthful mluti-armed bandit mechanisms"
- "Approaching optimality for solving SDD linear systems"
- "NP-hardness of approximately solving linear equations over reals"
- "LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs"
- "Set covering with our eyes closed"
- "Computing the girth of a planar graph in linear time"
- "Computing shortest paths and convex pseudodisks"

- See you at the final exam! – Wed., 10:15AM