NP-hardness

- Many problems in AI are NP-hard (or worse)
- What does this mean?
- These are some of the hardest problems in CS
- Identifying a problem as NP hard means:
  - You probably shouldn’t waste time trying to find a polynomial time solution
  - If you find a polynomial time solution, either
    - You have a bug
    - Find a place on your shelf for your Turing award
- NP hardness is a major triumph (and failure) for computer science theory

What is the class NP?

- A class of decision problems (Yes/No)
- Solutions can be verified in polynomial time
- Examples:
  - Graph coloring:
  - Sortedness: [1 2 3 4 5 8 7]

What is NP completeness?

- All NP complete problems can be “reduced” to each other in polynomial time
- What is a reduction?
  - Use one problem to solve another
  - A is reduced to B, if we can use B to solve A:
    - A instance \( \rightarrow \) Poly-time transformation \( \rightarrow \) B Solver
    - poly time A solver if B is poly time

Why care about NP-completeness?

- Solving any one NP-complete problem gives you the key to all others
- All NP-complete problems are, in a sense, equivalent
- Insight into solving any one gives you insight into solving a vast array of problems of extraordinary practical and economic significance

Proving NP Completeness

- Want to prove problem C is NP complete
  - Show that C is in NP
  - Find known NP complete problem reducible to C
  - Is graph color NP-complete?
    - Prove that graph coloring is in NP
      - Verify solution in poly time
      - Easy
    - Reduce known NP complete problem to TSPs
      - Much more challenging
      - Reduction to SAT
The First NP Complete Problem  
(Cook 1971)  
- SAT: 
  \((X_1 \lor \overline{X}_1 \lor X_{13}) \land (\overline{X}_2 \lor X_{12} \lor X_{23}) \land \ldots\)  
- Want to find an assignment to all variables that makes this expression evaluate to true  
- NP-complete for clauses of size 3 or greater  
- How would you prove this?

What is NP Hardness?  
- NP hardness is weaker than NP completeness  
- NP hard if an NP complete problem is reducible to it  
- NP completeness = NP hardness + NP membership  
- Consider the problem \#SAT  
  - How many satisfying assignments to:  
    \((X_1 \lor \overline{X}_1 \lor X_{13}) \land (\overline{X}_2 \lor X_{12} \lor X_{23}) \land \ldots\)  
  - Is this in NP?  
  - Is it NP-hard?

#SAT is NP-hard  
- Theorem: #SAT is NP hard  
- Proof:  
  - Reduce SAT to #SAT

NP-Completeness Summary  
- NP-completeness tells us that a problem belongs to class of similar, hard problems.  
- What if you find that a problem is NP hard?  
  - Look for good approximations  
  - Find different measures of complexity  
  - Look for tractable subclasses  
  - Use heuristics

CSPs  
- What is a CSP?  
- One view: Search with special goal criteria  
- CSP definition (general):  
  - Variables \(X_1, \ldots, X_n\)  
  - Variable \(X_i\) has domain \(D_i\)  
  - Constraints: \(C_1, \ldots, C_m\)  
  - Solution: Each variable gets a value from its domain such that no constraints violated  
- CSP examples…  
  - http://4c.ucc.ie/~tw/cspplib/

Our Restricted View  
- Variables \(X_1, \ldots, X_n\)  
- A binary constraint, lists permitted assignments to pairs of variables  
- A binary constraint between binary variables is a table of size 4, listing legal assignments for all 4 combinations.  
- A k-ary constraint lists legal assignments to k variables at a time.  
- How large is a k-ary constraint for binary variables?  

Note: More expressive languages are often used.
CSP Example

Problem: Assign Red, Green and Blue so that no 2 adjacent regions have the same color.

Example Contd.

- Variables: \{WA, NT, Q, SA, NSW, V, T\}
- Domains: \{R, G, B\}
- Constraints:
  For WA – NT:\{(R, G), (R, B), (G, B), (G, R), (B, R), (B, G)\}
- We have a table for each adjacent pair
- Are our constraints binary?
- Can every CSP be viewed as a graph problem?

Constraint Graph

Enumerate all Legal combinations Of WA and SA (ignoring other regions)

CSPs as Search

Nodes: Partial Assignments Actions: Make Assignments

Backtracking

- Backtracking is the most obvious (and widely used) method for solving CSPs:
  - Search forward by assigning values to variables
  - If stuck, undo the most recent assignment and try again
  - Repeat until satisfying assignment found or all combinations tried
- Embellishments
  - Methods for picking next variable to assign
    - Most constrained
    - Least constrained
    - Backjumping

NP-Completeness of CSPs

- Are CSPs in NP?
- Are they NP-hard?
- CSPs and graph coloring are equivalent
  - Convert any graph coloring problem to CSP
  - Convert any CSP to graph coloring
- Graph coloring is NP-complete
- CSPs are NP-complete
- End of the story or just the beginning?
**Issues**

- What are good heuristics?
  - Often good to think of this as a local search
  - Focus on choosing actions carefully, instead of pruning nodes carefully
- Can we develop heuristics that apply to the entire class of problems, not just specific instances?
- What’s the best we can hope for?

**Constraint Graphs**

- Constraint graphs are important because they capture the structural relationships between the variables
- **IMPORTANT CONCEPT:**
  - Not all instances of a hard problem class are hard
  - Structural features give insight into hardness
  - Group problems within class by structural features
  - New measure of problem complexity

**Node Consistency**

- Check all nodes for inconsistencies
- For each node, there must exist at least one valid assignment given assignments to neighbors
- Rules out some bad assignments quickly

**Arc Consistency**

- Check all arcs for inconsistencies
- For each value at the start, there must exist a consistent value at the terminus
- Catches many inconsistencies
- Can use to iteratively reduce number of possible assignments to each variable (constraint propagation)

**Generalized Arc Consistency**

- k-consistency
  - Consider sets of k variables
  - For each setting of a k-1 subset
  - Must exist a consistent setting for the kth variable
- Check for more distant influences
- 1-consistency = node consistency
- 2 consistency = arc consistency

**Facts About Arc Consistency**

- What if a graph with n variables is n-consistent?
- What is the worst-case cost of checking n-consistency?
Linear Constraint Structures

Are these easy or hard?

Suppose our chain is arc consistent…

Properties of Chains

Theorem: Arc consistent linear constraint graphs are n consistent.

Properties of Trees

Theorem: Arc consistent constraint trees are n consistent.

Variable Elimination

Domain(NT,SA) = {(blue, green), (blue, red), (green, blue), (green, red), (red, blue), (red, green)}

Eliminate Q

Domain(NT,SA,NSW) = {(blue, green, blue), (blue, red, blue), (red, blue, red), (red, green, red), (green, blue, green), (green, red, green)}

Simplify

Domain(SA, NSW) = {(blue, green, blue), (blue, red, blue), (red, blue, red), (red, green, red), (green, blue, green), (green, red, green)}
Domain(SA, NSW) = 
{(blue, green), (blue, red), 
(green, blue), (green, red), 
(red, blue), (red, green)}

Can identify all settings of SA, V, NSW for which there is guaranteed to be a consistent setting of the remaining variables.

Q: How do we get the settings of the other variables?

Variable Elimination

Var_elim_CSP_solve (vars, constraints)
Q = queue of all variables
i = length(vars)+1
While not(empty(Q))
    X = pop(Q)
    Xi = merge(X, neighbors(X))
    Simplify Xi
    remove_from_Q(Q, neighbors(X))
    add_to_Q(Q, Xi)
    i=i+1

Note: Merge operation can be tricky to implement, depending upon constraint language.

Variable Elimination Issues

• How expensive is this?

• Is it sensitive to elimination ordering?

Variable Elimination Facts

• You can figure out the cost of a particular elimination ordering without actually constructing the tables
• Finding optimal elimination ordering is NP hard
• Good heuristics for finding near optimal orderings
• Another structural complexity measure
• Investment in finding good ordering can be amortized

Variable Elimination Ordering

Is it better to start at the edges and work in, or at the center and work out?

Structural Complexity

• Structural complexity is a somewhat different view of computational complexity: depends upon problem features, not problem class
• For many problems structural complexity is quite manageable
• Idea: Why not convert other NP-hard problems to CSPs and use structural complexity measures, CSP algorithms to solve?

\[ 2^{\text{poly}(k)} \gg 2^k \]
CSP Summary

- CSPs are a specialized language for describing certain types of decision problems
- We can formulate special heuristics and methods for problems that can be described in this language
- In general, CSPs are NP hard
- We can use structural measures of complexity to figure out which ones are really hard