Today’s Lecture

- Another Form of Local Search
  - Repair/Debugging in Constraint Satisfaction Problems
    - GSAT

- A Systematic Approach to Constraint Satisfaction Problems
  - Simple Backtracking Search

Constraint Satisfaction Problems (CSP)

- A set of variables $X_1...X_n$, and a set of constraints $C_1...C_m$. Each variable $X_i$ has a domain $D_i$ of possible values.
- A solution to a CSP: a complete assignment to all variables that satisfies all the constraints.
- Representation of constraints as predicates.
- Visualizing a CSP as a constraint graph.
Example: Map coloring

<table>
<thead>
<tr>
<th>Variables</th>
<th>Domain</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA, NT, Q, NSW, V, SA, T</td>
<td>{red, green, blue}</td>
<td>adjacent regions must have different colors</td>
</tr>
</tbody>
</table>

Example 3: N queens

- What are the variables? domains? constraints?

A Valid Map Assignment

Solutions are assignments satisfying all constraints, e.g.,
{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green}

8 queens

- 8 variables X_i, i = 1 to 8; for each column
- Domain for each variable {1,2,3,4,5,6,7,8}
- Constraints are:
  - X_i ≠ X_j for all j = 1 to 8, j ≠ i; not on the same row
  - |X_i - X_j| ≠ |i - j| for all j = 1 to 8, j ≠ i; not on diagonal
- Note that all constraints involve 2 variables
- Generate-and-test with no redundancies requires “only” N! combinations…
**Task scheduling**

- T1 must be done during T3
- T2 must be achieved before T1 starts
- T2 must overlap with T3
- T4 must start after T1 is complete

**Non-Binary Constraints**

- T1 + T4 = O
- T2 + W = U
- T3 + T = O
- T4 = F

- alldiff(F, T, U, W, R, O)
- Between0–9(F, T, U, W, R, O)
- Between0–1(X1, X2, X3)

**Constraint optimization**

- Representing *preferences* versus absolute constraints.
  - Weighted by constraints violated/satisfied
  - Constraint optimization is generally more complicated.
  - Can also be solved using local search techniques.
  - Hard to find optimal solutions.

**Local search for CSPs: Heuristic Repair**

- Start state is some assignment of values to variables that may violate some constraints.
- Create a complete but inconsistent assignment
- Successor state: change value of one variable.
- Use heuristic repair methods to reduce the number of conflicts (iterative improvement).
  - The min-conflicts heuristic: choose a value for a variable that minimizes the number of remaining conflicts.
  - Hill climbing on the number of violated constraints
- Repair constraint violations until a consistent assignment is achieved.
- Can solve the million-queens problem in an average of 50 steps!
Heuristic Repair Algorithm

function MIN-CONFLICTS(exp, max-steps) returns a solution or failure
inputs: exp, a constraint satisfaction problem
max-steps, the number of steps allowed before giving up
local variables: current, a complete assignment
var, a variable
value, a value for a variable

current ← an initial complete assignment for exp
for i = 1 to max-steps do
    var ← a randomly chosen, conflicted variable from VARIABLES[exp]
    value ← the value v for var that minimizes CONFLICTS(var, v, current, exp)
    set var=value in current
    if current is a solution for exp then return current
end
return failure

N-Queens Heuristic Repair

- Pre-processing phase to generate initial assignment
  - Greedy algorithm that iterates through rows placing each queen on the column where it conflicts with the fewest previously placed queens

- Repair phase
  - Select (randomly) a queen in a specific row that is in conflict and moves it to the column (within the same row) where it conflicts with the fewest other queens

Example of min-conflicts: N-Queens Problem

A two-step solution of an 8-queens problem. The number of remaining conflicts for each new position of the selected queen is shown. Algorithm moves the queen to the min-conflict square, breaking ties randomly.

SAT- Satisfiability Problem

Given a propositional sentence, determine if it is satisfiable, and if it is, show which propositions have to be true to make the sentence true. 3SAT is the problem of finding a satisfying truth assignment for a sentence in a special format

*Why are we interested in this representational framework?*
**Definition of 3SAT**

- A **literal** is a proposition symbol or its negation (e.g., $P$ or $\neg P$).
- A **clause** is a disjunction of literals; a 3-clause is a disjunction of exactly 3 literals (e.g., $P \lor Q \lor \neg R$).
- A sentence in **CNF** or conjunctive normal form is a conjunction of clauses; a 3-CNF sentence is a conjunction of 3-clauses.

For example,

\[(P \lor Q \lor \neg S) \land (\neg P \lor Q \lor R) \land (\neg P \lor \neg R \lor \neg S) \land (P \lor \neg S \lor T)\]

Is a 3-CNF sentence with four clauses and five proposition symbols.

---

**Mapping 3-Queens into 3SAT**

At least 1 has a $Q$ and not exactly 2 have $Q$'s and not all 3 have $Q$'s

\[(Q_1 \lor Q_2 \lor Q_3) \land (\neg Q_1 \lor \neg Q_2 \lor \neg Q_3) \land (\neg Q_2 \lor \neg Q_3 \lor \neg Q_1) \land (\neg Q_3 \lor \neg Q_1 \lor \neg Q_2)\]

Do the same for each row, the same for each column, the same for each diagonal, and then all together.

\[(Q_1 \lor Q_2 \lor Q_3) \land (\neg Q_1 \lor \neg Q_2 \lor \neg Q_3) \land (\neg Q_2 \lor \neg Q_3 \lor \neg Q_1) \land (\neg Q_3 \lor \neg Q_1 \lor \neg Q_2) \land \ldots \]

\[(Q_1 \lor \neg Q_2 \lor \neg Q_3) \land (\neg Q_1 \lor Q_2 \lor \neg Q_3) \land (\neg Q_1 \lor \neg Q_2 \lor Q_3) \land (\neg Q_1 \lor \neg Q_2 \lor \neg Q_3)\]

\[\vdots\]

\[\vdots \]

---

**Converting N-SAT into 3-SAT**

\[A \lor B \lor C \lor D\]

\[= (A \lor B \lor E) \land (\neg E \lor C \lor D)\]

| $A$ = $T$ | $A = F$ | $A = F$ |
| $B = F$ | $B = T$ | $B = F$ |
| $C = F$ | $C = F$ | $C = T$ |
| $D = F$ | $D = F$ | $D = F$ |
| $E = F$ | $E = F$ | $E = T$ |

2-SAT polynomial time but can’t map all problem into 2-SAT.

---

**Davis-Putnam Algorithm (Depth-First Search)**

\[(A \lor C) \land (\neg A \lor C) \land (B \lor \neg C) \land (A \lor \neg B)\]

\[C \land (B \lor \neg C) \land \neg B\]

\[C \land (B \lor \neg C) \land \ldots\]
GSAT Algorithm

Problem: Given a formula of the propositional calculus, find an interpretation of the variables under which the formula comes out true, or report that none exists.

procedure GSAT
Input: a set of clauses \( \propto \), MAX-FLIPS, and MAX-TRIES
Output: a satisfying truth assignment of \( \propto \), if found
begin
for \( i = 1 \) to MAX-TRIES; random restart mechanism
T := a randomly generated truth assignment
for \( j = 1 \) to MAX-FLIPS
if \( T \) satisfies \( \propto \) then return \( T \)
\( \Delta \) := a propositional variable such that a change in its truth assignment gives the largest increase in total number of clauses of \( \propto \) that are satisfied by \( T \).
\( T \) := \( T \) with the truth assignment of \( p \) reversed
end for
end for
return "no satisfying assignment found"
end

GSAT Performance

GSAT Biased Random Walk
- With probability \( p \), follow the standard GSAT scheme,
  - i.e., make the best possible flip.
- With probability \( 1 - p \), pick a variable occurring in some unsatisfied clause and flip its truth assignment. (Note: a possible uphill move.)
- GSAT-Walk < Simulated-Annealing < GSAT-Noise < GSAT-Basic

3SAT Phase Transition

- Easy -- Satisfiable problems where many solutions
- Hard -- Satisfiable problems where few solutions
- Easy -- Few Satisfiable problems
- Assumes concurrent search in the satisfiable space and the non-satisfiable space (negation of proposition)
A Simplistic Approach to Solving CSPs using Systematic Search

- **Initial state**: the empty assignment
- **Successor function**: a value can be assigned to any variable as long as no constraint is violated.
- **Goal test**: the current assignment is complete.
- **Path cost**: a constant cost for every step. – not relevant

What more is needed?

- Not just a successor function and goal test
- But also a means to propagate the constraints imposed by variables already bound along the path on the potential fringe nodes of that path and an early failure test
- Thus, need explicit representation of constraints and constraint manipulation algorithms

Exploiting Commutativity

- Naïve application of search to CSPs:
  - If use breadth first search
  - Branching factor is $n \cdot d$ at the top level, then $(n-1) \cdot d$, and so on for $n$ levels ($n$ variables, and $d$ values for each variable).
  - The tree has $n^d$ leaves, even though there are only $d^n$ possible complete assignments!
- Naïve formulation ignores commutativity of all CSPs: the order of any given set of actions has no effect on the outcome.
  - [WA=red, NT=green] same as [NT=green, WA=red]
- Solution: consider a single variable at each depth of the tree.

Part of the map-coloring search tree

Variable 1 - WA

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>red</td>
<td>green</td>
<td>blue</td>
</tr>
</tbody>
</table>

Variable 2 - NT

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>red</td>
<td>blue</td>
</tr>
</tbody>
</table>

Variable 3 - Q

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>red</td>
<td>blue</td>
</tr>
</tbody>
</table>
Next Lecture

- Informed-Backtracking Using Min-Conflicts Heuristic
- Arc Consistency for Pre-processing
- Intelligent backtracking
- Reducing the Search by structuring the CSP as a tree search

- Extending the model of simple heuristic search
- Interacting subproblem perspective