Today’s Class

• Constraint Satisfaction Problems
  • A.K.A., Constraint Processing / CSP paradigm
• Algorithms for CSPs
• Search Terminology

Search Vocabulary

• Constraint satisfaction: a problem-solving paradigm
• Constraint programming, constraint satisfaction problems (CSPs), constraint logic programming…
• Algorithms for CSPs
  • Backtracking (systematic search)
  • Constraint propagation (k-consistency)
  • Variable and value ordering heuristics
  • Backjumping and dependency-directed backtracking

Overview

• Constraint satisfaction: a problem-solving paradigm
• Constraint programming, constraint satisfaction problems (CSPs), constraint logic programming…
• Algorithms for CSPs
  • Backtracking (systematic search)
  • Constraint propagation (k-consistency)
  • Variable and value ordering heuristics
  • Backjumping and dependency-directed backtracking

Constraint Satisfaction

• Constraint /kan'strænt/ (noun):
  • Something that limits or restricts someone or something.¹
  • Control that limits or restricts someone’s actions or behavior.¹
  • A relation … between the values of one or more mathematical variables (e.g., x>3 is a constraint on x).²
  • Assigns values to variables so that all constraints are true.²

General Idea
• View a problem as a set of variables
• To which we have to assign values
• That satisfy a number of (problem-specific) constraints

Bookkeeping

• HW 2 out: Search
  • Due 10/3, 11:59pm
• Jumping Frogs
• Ricochet Robots

Constraint satisfaction

• A relation … between the values of one or more mathematical variables (e.g., x>3 is a constraint on x).
• Assigns values to variables so that all constraints are true.

– http://foldoc.org/constraint

¹ Merriam-Webster online.
² The Free Online Computing Dictionary.
Slightly Less Informal Definition of CSP

- **CSP** = Constraint Satisfaction Problem
- **Given:**
  1. A finite set of **variables**
  2. Each with a **domain** of possible value (often finite)
  3. A set of **constraints** that limit the values the variables can take on
- **Solution:** an assignment of a value to each variable such that the constraints are all satisfied.

CSP Applications

- Decide if a solution exists
- Find some solution
- Find all solutions
- Find the “best solution”
  - According to some metric (objective function)
  - Does that mean “optimal”?

Informal Example: Map Coloring

- Color a map, such that:
  - Using three colors (red, green, blue)
  - No two adjacent regions have the same color

Map Coloring II

- Variables: A, B, C, D, E
- Domains: RGB = {red, green, blue}
- Constraints:
  - A≠B, A ≠ C, A ≠ E, A ≠ D, B ≠ C, C ≠ D, D ≠ E
  - One solution: A=red, B=green, C=blue, D=green, E=blue

Slightly Less Informal

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test: any function over states
  - Successor function can be anything
- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables \( X_i \) with values from a domain \( D \)
  - Sometimes \( D \) depends on \( i \)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Example: N-Queens (1)

- Formulation 1:
  - Variables: \( X_{ij} \)
  - Domains: \{0, 1\}
  - Constraints:
    \[
    \forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}
    \forall i, j, k \ (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}
    \forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\}
    \forall i, j, k \ (X_{ij}, X_{i+k,j-k}) \in \{(0, 0), (0, 1), (1, 0)\}
    \sum_{i,j} X_{ij} = N
    \]
Example: N-Queens (2)

- Formulation 2:
  - Variables: \( Q_k \)
  - Domains: \{1, 2, 3, \ldots \ N\}
  - Constraints:
    - Implicit: \( \forall i, j \text{ non-threatening}(Q_i, Q_j) \)
    - Explicit: \( (Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots \} \)

Example: SATisfiability

- Given a set of propositions containing variables, find an assignment of the variables to \{false, true\} that satisfies them.
- For example, the clauses:
  - \((A \lor B \lor \neg C) \land (\neg A \lor D)\)
  - (equivalent to \((C \rightarrow A) \lor (B \land D \rightarrow A)\))
are satisfied by
  - A = false
  - B = true
  - C = false
  - D = false

Real-World Problems

- Scheduling
- Temporal reasoning
- Building design
- Planning
- Optimization/satisfaction
- Vision
- Graph layout
- Network management
- Natural language processing
- Molecular biology/genomics
- VLSI design

Formal Definition: Constraint Network (CN)

A constraint network (CN) consists of
- A set of variables \( X \equiv \{x_1, x_2, \ldots x_j\}\)
- Each with an associated domain of values \( \{d_1, d_2, \ldots d_j\}\)
- The domains are typically finite
- A set of constraints \( \{c_1, c_2, \ldots c_m\} \) where
  - Each constraint defines a predicate which is a relation over a particular subset of \( X \).
  - E.g., \( c_i \) involves variables \( \{X_{i1}, X_{i2}, \ldots X_{ik}\} \) and defines the relation \( R_i \subseteq D_{i1} \times D_{i2} \times \ldots D_{ik} \)
- **Unary** constraint: only involves one variable
- **Binary** constraint: only involves two variables

Formal Definition of a CN (cont.)

- Instantiations
  - An **instantiation** of a subset of variables \( S \) is an assignment of a value in its domain to each variable in \( S \)
  - An instantiation is **legal** if it does not violate any constraints
- A **solution** is an instantiation of all of the variables in the network

Typical Tasks for CSP

- Solutions:
  - Does a solution exist?
  - Find one solution
  - Find all solutions
  - Given a partial instantiation, do any of the above
- Transform the CN into an equivalent CN that is easier to solve.
**Binary CSP**

- **Binary CSP**: all constraints are binary or unary
- Can convert a non-binary CSP → binary CSP by:
  - Introducing additional variables
  - Dual graph construction: one variable for each constraint; one binary constraint for each pair of original constraints that share variables
- Can represent a binary CSP as a **constraint graph** with:
  - A node for each variable
  - An arc between two nodes iff there is a constraint involving the two variables
  - Unary constraint appears as a self-referential arc

**Example: Sudoku**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Running Example: Sudoku**

- Variables and their domains
  - $v_{ij}$ is the value in the $j$th cell of the $i$th row
  - $D_{ij} = \{1, 2, 3, 4\}$
- Blocks:
  - $B_1 = \{11, 12, 21, 22\}, \ldots, B_4 = \{33, 34, 43, 44\}$
- Constraints (implicit/intensional)
  - $C_R: \forall i, \bigcup_j v_{ij} = D$ (every value appears in every row)
  - $C_C: \forall j, \bigcup_i v_{ij} = D$ (every value appears in every column)
  - $C_B: \forall k, \bigcup_{ij \in B_k} v_{ij} = D$ (every value appears in every block)
- Alternative representation: pairwise inequality constraints
  - $P: \forall i, j \neq j', v_{ij} \neq v_{ij'}$ (no value appears twice in any row)
  - $F: \forall j, \bigcap_i v_{ij} \neq v_{ij'}$ (no value appears twice in any column)
  - $P': \forall i, j \neq j', k \in B_k, v_{ij} \neq v_{ij'} \neq v_{ij''}$ (no value appears twice in any block)
- Advantage of the second representation: all binary constraints!

**Sudoku Constraint Network**

**Solving Constraint Problems**

- Systematic search
  - Generate and test
  - Backtracking
- Constraint propagation (consistency)
- Variable ordering heuristics
- Value ordering heuristics
- Backjumping and dependency-directed backtracking
Generate and Test: Sudoku

- Try each possible combination until you find one that works:

```
1 3 1
3 1 7
7 1 4
```

- Doesn't check constraints until all variables have been instantiated
- Very inefficient way to explore the space of possibilities (4^7 for this trivial Sudoku puzzle, most illegal)

Systematic Search: Backtracking

(a.k.a. depth-first search!)

- Consider the variables in some order
- Pick an unassigned variable and give it a provisional value such that it is consistent with all of the constraints
- If no such assignment can be made, we've reached a dead end and need to backtrack to the previous variable
- Continue this process until:
  - A solution is found, or
  - We backtrack to the initial variable and have exhausted all possible values

Problems with Backtracking

- Thrashing: keep repeating same failed variable assignments
- Inefficiency: can spend time exploring areas of search space that aren't likely to succeed

Consistency

- Node consistency
  - Node X is node-consistent if every value in the X's domain is consistent with X's unary constraints
  - A graph is node-consistent if all nodes are node-consistent
- Arc consistency
  - Arc (X, Y) is arc-consistent if, for every value x of X, there is a value y for Y that satisfies the constraint represented by the arc.
  - A graph is arc-consistent if all arcs are arc-consistent.
- To create arc consistency, we perform constraint propagation: that is, we repeatedly reduce the domain of each variable to be consistent with its arcs

Constraint Propagation: Sudoku

```
2 2
3 2
```

```
2 2
2 2
```

```
2 2
2 2
```

Example: The Waltz Algorithm

- Waltz algorithm is for interpreting line drawings of solid polyhedra
- Look at all intersections
- Adjacent intersections impose constraints on each other
Waltz on Simple Scenes

- Assume all objects:
  - Have no shadows or cracks
  - Three-faced vertices
  - "General position": no junctions change with small movements of the eye.

- Then each line on image is:
  - Boundary line (edge of an object) (→) with right hand of arrow denoting "solid" and left hand denoting "space" or
  - Interior convex edge (+)
  - Interior concave edge (−)

Legal Junctions

- Only certain junctions are physically possible
- How can we formulate a CSP to label an image?
- Variables: vertices
- Domains: junction labels
- Constraints: both ends of a line should have the same label

Line Drawings

- Here are some illegal labelings for this shape:

Example: Map-Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domain: \( D = \{\text{red, green, blue}\} \)
- Constraints: adjacent regions must have different colors
  - WA ≠ NT
  - \( (WA, NT) \notin \{(\text{red, green}), (\text{red, blue}), (\text{green, red})\} \)
- Solutions are assignments satisfying all constraints, e.g.:
  - \( \{WA = \text{red}, NT = \text{green}, Q = \text{red}, NSW = \text{green}, V = \text{red}, SA = \text{blue}, T = \text{green}\} \)

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
  - General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!
Standard Search Formulation

• Standard search formulation of CSPs (incremental)
• Let’s start with a straightforward, dumb approach, then fix it
• States are defined by the values assigned so far
  • Initial state: the empty assignment, {}  
  • Successor function: assign a value to an unassigned variable
  • Goal test: the current assignment is complete and satisfies all constraints

Search Methods

• What does BFS do?
• What does DFS do?

DFS & BFS: not good!

Backtracking Search

• Idea 1: Only consider a single variable at each point
  • Variable assignments are commutative, so fix ordering
  • I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  • Only need to consider assignments to a single variable at each step
  • How many leaves are there now?
• Idea 2: Only allow legal assignments at each point
  • I.e. consider only values which do not conflict previous assignments
  • Might have to do some computation to figure out whether a value is ok
  • “Incremental goal test”

Backtracking Search

• Idea 1: Only consider a single variable at each point
• Idea 2: Only allow legal assignments at each point
• Depth-first search for CSPs with these two improvements is called backtracking search
• Backtrack when there’s no legal assignment for the next variable
• Backtracking search is the basic uninformed algorithm for CSPs
• Can solve n-queens for n = 25

Backtracking Search

function BACKTRACKING-SEARCH(cp) returns solution/failure
return RECURSIVE-BACKTRACKING({}, cp)

function RECURSIVE-BACKTRACKING.assignment, cp returns solution/failure
if assignment is complete then return assignment
else
  select-an-unsatisfied-variable(VARIABLES[cp], assignment, cp)
  for each value in ORDER-DOMAIN-VALUES(assignment[cp], cp) do
    if value is consistent with assignment give CONSTRAINTS[cp] then
      add {var = value} to assignment
      result = RECURSIVE-BACKTRACKING(assignment, cp)
      if result = failure then return result
      remove {var = value} from assignment
    end if
  end for
end if
return failure
Backtracking Example

Backtracking

Good Enough?

Improving Backtracking

• General-purpose ideas give huge gains in speed
• Ordering:
  • Which variable should be assigned next?
  • In what order should its values be tried?
• Filtering: Can we detect inevitable failure early?
• Structure: Can we exploit the problem structure?

Forward Checking

• Idea: Keep track of remaining legal values for unassigned variables (using immediate constraints); terminate when any variable has no legal values

WA  NT  Q  NSW  V  SA  T

Forward Checking
K-consistency

- K-consistency generalizes the notion of arc consistency to sets of more than two variables
  - A graph is K-consistent if, for legal values of any K-1 variables in the graph, and for any Kth variable \( V_k \), there is a legal value for \( V_k \)
- **Strong** K-consistency = J-consistency for all J<=K
- Node consistency = strong 1-consistency
- Arc consistency = strong 2-consistency
- Path consistency = strong 3-consistency

Why Do We Care?

1. A strongly N-consistent CSP with N variables can be solved **without backtracking**
2. For any CSP that is strongly K-consistent:
   - If we find an appropriate variable ordering (one with “small enough” branching factor)
   - We can solve the CSP without backtracking

Ordered Constraint Graphs

- Select a variable ordering, \( V_1, \ldots, V_n \)
- **Width of a node** in this OCG is the number of arcs leading to earlier variables:
  \[ w(V_i) = \text{Count} \left( (V_i, V_j) \mid k < i \right) \]
- **Width of the OCG** is the maximum width of any node:
  \[ w(G) = \text{Max} (w(V_i)), 1 <= i <= N \]
- **Width of an unordered CG** is the minimum width of all orderings of that graph (“best you can do”)

Tree-Structured Constraint Graph

- A **constraint tree** rooted at \( V_1 \) satisfies:
  - There exists an ordering \( V_1, \ldots, V_n \) such that every node has zero or one parents (i.e., each node only has constraints with at most one “earlier” node in the ordering)
  - Also known as an ordered constraint graph with width 1
- If this constraint tree is also node- and arc-consistent (a.k.a. strongly 2-consistent), it can be solved **without backtracking**
  - (More generally, if the ordered graph is strongly k-consistent, and has width \( w < k \), then it can be solved without backtracking.)

Proof Sketch for Constraint Trees

- Perform backtracking search in the order that satisfies the constraint tree condition
- Every node, when instantiated, is constrained only by at most one previous node
- Arc consistency tells us that there must be at least one legal instantiation in this case
  - (If there are no legal solutions, the arc consistency procedure will collapse the graph – some node will have no legal instantiations)
- Keep doing this for all \( n \) nodes, and you have a legal solution – without backtracking!

Variable Ordering

- **Intuition**: choose variables that are highly constrained early in the search process; leave easy ones for later
- **Minimum width ordering** (MWO): identify OCG with minimum width
- **Maximum cardinality ordering**: approximation of MWO that’s cheaper to compute: order variables by decreasing cardinality (a.k.a. degree heuristic)
- **Fail first principle** (FFP): choose variable with the fewest values (a.k.a. minimum remaining values (MRV))
  - **Static FFP**: use domain size of variables
  - **Dynamic FFP** (search rearrangement method): At each point in the search, select the variable with the fewest remaining values
Minimum Width

• Or “minimum remaining values” (MRV):
  • Choose the variable with the fewest legal values
  • Why min rather than max?
  • Also called “most constrained variable”
  • “Fail-fast” ordering

Variable Ordering II

• Maximal stable set: find largest set of variables with no constraints between them and save these for last
• Cycle-cutset tree creation: Find a set of variables that, once instantiated, leave a tree of uninstantiated variables; solve these, then solve the tree without backtracking
• Tree decomposition: Construct a tree-structured set of connected subproblems

Value Ordering

• Intuition: Choose values that are the least constrained early on, leaving the most legal values in later variables
• Maximal options method (a.k.a. least-constraining-value heuristic): Choose the value that leaves the most legal values in uninstantiated variables
• Min-conflicts: For iterative repair search (Coming up)
• Symmetry: Introduce symmetry-breaking constraints to constrain search space to “useful” solutions (don’t examine more than one symmetric/isomorphic solution)

Iterative Repair

• Start with an initial complete (but invalid) assignment
• Hill climbing, simulated annealing
• Min-conflicts: Select new values that minimally conflict with the other variables
  • Use in conjunction with hill climbing or simulated annealing or…
• Local maxima strategies
  • Random restart
  • Random walk
  • Tabu search: don’t try recently attempted values

Min-Conflicts Heuristic

• Iterative repair method
  1. Find some “reasonably good” initial solution
  • E.g., in N-queens problem, use greedy search through rows, putting each queen where it conflicts with the smallest number of previously placed queens, breaking ties randomly
  2. Find a variable in constraint violation
  3. Select a new value constraint violations
  • O(N) time and space
  4. Repeat steps 2 and 3 until done

Performance depends on quality and informativeness of initial assignment; inversely related to distance to solution

Challenges

• What if not all constraints can be satisfied?
  • Hard vs. soft constraints
  • Degree of constraint satisfaction
  • Cost of violating constraints
• What if constraints are of different forms?
  • Symbolic constraints
  • Numerical constraints [constraint solving]
  • Temporal constraints
  • Mixed constraints
More Challenges

- What if constraints are represented intensionally?
  - Cost of evaluating constraints (time, memory, resources)
- What if constraints/variables/values change over time?
  - Dynamic constraint networks
  - Temporal constraint networks
  - Constraint repair
- What if you have multiple agents or systems involved in constraint satisfaction?
  - Distributed CSPs
  - Localization techniques