Searching by Constraint & Searching by Evolution

CSPP 56553
Artificial Intelligence
January 21, 2004
Agenda

• Constraint Propagation: Motivation
• Constraint Propagation Example
  – Waltz line labeling
• Constraint Propagation Mechanisms
  – Arc consistency
  – CSP as search
    • Forward-checking
    • Back-jumping
  – Iterative refinement: min-conflict
• Summary
Leveraging Representation

• General search problems encode task-specific knowledge
  – Successor states, goal tests, state structure
  – “black box” wrt to search algorithm
• Constraint satisfaction fixes representation
  – Variables, values, constraints
  – Allows more efficient, structure-specific search
Constraint Satisfaction Problems

• Very general: Model wide range of tasks
• Key components:
  – Variables: Take on a value
  – Domains: Values that can be assigned to vars
    • Finite, Infinite, Real; Discrete vs Continuous
  – Constraints: Restrictions on assignments
    • Unary, Binary, N-ary
• Constraints are HARD
  – Not preferences: Must be observed
    • E.g. Can’t schedule two classes: same room, same time
Constraint Satisfaction Problem

- Graph/Map Coloring: Label a graph such that no two adjacent vertexes same color
  - Variables: Vertexes
  - Domain: Colors
  - Constraints: If E(a,b), then C(a) \(!=\) C(b)
Question

What are some other problems that can be framed as constraint satisfaction?
Constraint Satisfaction Problem

- “N-Queens”:
  - Place $N$ queens on an $N \times N$ chessboard such that none attacks another
  - Variables: Queens (1/column)
  - Domain: Rows
  - Constraints: Not same row, column, or diagonal
N-queens
Constraint Satisfaction Problem

• Range of tasks:
  – Coloring, Resource Allocation, Satisfiability
  – Varying complexity: E.g. 3-SAT NP-complete
    • Complexity: Property of problem NOT CSP

• Basic Structure:
  – Variables: Graph nodes, Classes, Boolean vars
  – Domains: Colors, Time slots, Truth values
  – Constraints: No two adjacent nodes with same color,
    • No two classes in same time, consistent, satisfying ass't
Problem Characteristics

• Values:
  – Finite? Infinite? Real?
    • Discrete vs Continuous

• Constraints
  – Unary? Binary? N-ary?
    • Note: all higher order constraints can be reduced to binary
Representational Advantages

- **Simple goal test:**
  - Complete, consistent assignment
    - Complete: All variables have a value
    - Consistent: No constraints violates

- **Maximum depth?**
  - Number of variables

- **Search order?**
  - Commutative, reduces branching
  - Strategy: Assign value to one variable at a time
Constraint Satisfaction Questions

- Can we rule out possible search paths based on current values and constraints?
- How should we pick the next variable to assign?
- Which value should we assign next?
- Can we exploit problem structure for efficiency?
Constraint Propagation Method

• For each variable,
  – Get all values for that variable
  – Remove all values that conflict with ALL adjacent
    • A: For each neighbor, remove all values that conflict with ALL adjacent
      – Repeat A as long as some label set is reduced
Constraint Propagation Example

- **Image interpretation:**
  - From a 2-D line drawing, automatically determine for each line, if it forms a concave, convex or boundary surface
  - Segment image into objects

- **Simplifying assumptions:**
  - World of polyhedra
    - No shadows, no cracks
CSP Example: Line Labeling

• 3 Line Labels:
  – Boundary line: regions do not abut: >
    • Arrow direction: right hand side walk
  – Interior line: regions do abut
    • Concave edge line: _
    • Convex edge line: +

• Junction labels:
  – Where line labels meet
CSP Example: Line Labeling

• Simplifying (initial) restrictions:
  – No shadows, cracks
  – Three-faced vertexes
  – General position:
    • small changes in view -> no change in junction type
• $4^2$ labels for 2 line junction: L
• $4^3$; 3-line junction: Fork, Arrow, T
• Total: 208 junction labelings
CSP Example: Line Labeling

- Key observation: Not all 208 realizable
  - How many? 18 physically realizable
CSP Example: Line Labeling

- Label boundaries clockwise
- Label arrow junctions
CSP Example: Line Labeling

- Label boundaries clockwise
CSP Example: Line Labeling

• Label fork junctions with +’s
CSP Example: Line Labeling

- Label arrow junctions with -’s
Waltz’s Line Labeling

- For each junction in image,
  - Get all labels for that junction type
  - Remove all labels that conflict with ALL adjacent
    - A: For each neighbor, remove all labels that conflict with ALL
      - Repeat A as long as some label set is reduced
Waltz Propagation Example
Waltz’s Line Labeling

• Full version:
  – Removes constraints on images
    • Adds special shadow and crack labels
    • Includes all possible junctions
      – Not just 3 face

• Physically realizable junctions
  – Still small percentage of all junction labels
  – $O(n) : n = \# \text{ lines in drawing}$
Constraint Propagation Mechanism

- Arc consistency
  - Arc V1->V2 is *arc consistent* if for all x in D1, there exists y in D2 such that (x,y) is allowed

- Delete disallowed x’s to achieve arc consistency
Arc Consistency Example

- Graph Coloring:
  - Variables: Nodes
  - Domain: Colors (R,G,B)
  - Constraint: If $E(a,b)$, then $C(a) \neq C(b)$

- Initial assignments

```
\begin{align*}
& \text{R,G,B} \\
& \text{R,B} \\
& \text{B} \\
& X \\
& Y \\
& Z
\end{align*}
```
Arc Consistency Example

<table>
<thead>
<tr>
<th>Arc</th>
<th>Rm Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X -&gt; Y</td>
<td>None</td>
</tr>
<tr>
<td>X -&gt; Z</td>
<td>X=B</td>
</tr>
<tr>
<td>Y -&gt; Z</td>
<td>Y=B</td>
</tr>
<tr>
<td>X -&gt; Y</td>
<td>X=R</td>
</tr>
<tr>
<td>X -&gt; Z</td>
<td>None</td>
</tr>
<tr>
<td>Y -&gt; Z</td>
<td>None</td>
</tr>
</tbody>
</table>

Assignments:
X = G
Y = R
Z = B
Limitations of Arc Consistency

- **Arc consistent:**
  - No legal assignment

- **Arc consistent:**
  - >1 legal assignment

```plaintext
\[
\begin{align*}
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{R,G} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\text{Arc consistent:} & \quad \text{G,B} \\
\end{align*}
\]```
CSP as Search

- Constraint Satisfaction Problem (CSP) is a restricted class of search problem
  - State: Set of variables & assignment info
    - Initial State: No variables assigned
    - Goal state: Set of assignments consistent with constraints
  - Operators: Assignment of value to variable
CSP as Search

- **Depth**: Number of Variables
- **Branching Factor**: $|\text{Domain}|$
- **Leaves**: Complete Assignments

- **Blind search strategy**:
  - Bounded depth $\rightarrow$ Depth-first strategy

- **Backtracking**:
  - Recover from dead ends
  - Find satisfying assignment
Constraint Graph

R, G, B

X

Y

R, B

Z

R, B
CSP as Search

X = R

X = G

X = B

Y = R

Y = B

Z = B

Z = R

Yes!

No!
Backtracking Issues

- CSP as Search
  - Depth-first search
    - Maximum depth = # of variables
  - Continue until (partial/complete) assignment
    - Consistent: return result
    - Violates constraint -> backtrack to previous assignment
  - Wasted effort
    - Explore branches with no possible legal assignment
Backtracking with Forward Checking

• Augment DFS with backtracking
  – Add some constraint propagation

• Propagate constraints
  – Remove assignments which violate constraint
  – Only propagate when domain = 1
    • “Forward checking”
    • Limit constraints to test
Backtracking+Forward Checking
Heuristic CSP

• Improving Backtracking
  – Standard backtracking
    • Back up to last assignment

  – Back-jumping
    • Back up to last assignment that reduced current domain
    • Change assignment that led to dead end
Heuristic backtracking: Back-jumping
Heuristic Backtracking: Backjumping
Heuristic Backtracking: Backjumping

Dead end! Why?
Back-jumping

- Previous assignment reduced domain
  - C3 = B
- Changes to intermediate assignment can’t affect dead end
- Backtrack up to C3
  - Avoid wasted work - alternatives at C4
- In general, forward checking more effective
Heuristic CSP: Dynamic Ordering

• Question: What to explore next

• Current solution:
  – Static: Next in fixed order
    • Lexicographic, leftmost..
  – Random

• Alternative solution:
  – Dynamic: Select “best” in current state
Question

- How would you pick a variable to assign?
- How would you pick a value to assign?
Dynamic Ordering

- Heuristic CSP
  - Most constrained variable:
    - Pick variable with smallest current domain
  - Least constraining value:
    - Pick value that removes fewest values from domains of other variables
    - Improve chance of finding SOME assignment
- Can increase feasible size of CSP problem
Dynamic Ordering
Dynamic Ordering with FC
Incremental Repair

- Start with initial complete assignment
  - Use greedy approach
  - Probably invalid - i.e. violates some constraints
- Incrementally convert to valid solution
  - Use heuristic to replace value that violates
    - “min-conflict” strategy:
      - Change value to result in fewest constraint violations
      - Break ties randomly
      - Incorporate in local or backtracking hill-climber
Incremental Repair

5 conflicts

2 conflicts

0 conflicts
Question

• How would we apply iterative repair to Traveling Salesman Problem?
Min-Conflict Effectiveness

- N-queens: Given initial random assignment, can solve in $\sim O(n)$
  - For $n < 10^7$

- GSAT (satisfiability)
  - Best (near linear in practice) solution uses min-conflict-type hill-climbing strategy
    - Adds randomization to escape local min

- $\sim$Linear seems true for most CSPs
  - Except for some range of ratios of constraints to variables

- Avoids storage of assignment history (for BT)
CSP Complexity

• Worst-case in general:
  – Depth-first search:
    • Depth = # of variables
    • Branching factor: |Domain|
    • $|D|^{n+1}$
  – No loops in constraint graph
  – $O(n|D|^{2})$
Tree-structured CSPs

Create breadth-first ordering
Starting from leaf nodes \([O(n)]\),
    remove all values from parent domain that do not
        participate in some valid pairwise constraint
\([O(|D|^2)])\]

Starting from root node, assign value to variable
Iterative Improvement

- Alternate formulation of CSP
  - Rather than DFS through partial assignments
  - Start with some complete, valid assignment
    - Search for optimal assignment wrt some criterion
- Example: Traveling Salesman Problem
  - Minimum length tour through cities, visiting each one once
Iterative Improvement Example

- TSP
  - Start with some valid tour
    - E.g. find greedy solution
  - Make incremental change to tour
    - E.g. hill-climbing - take change that produces greatest improvement
      - Problem: Local minima
      - Solution: Randomize to search other parts of space
  - Other methods: Simulated annealing, Genetic alg’s
Agenda

- **Motivation:**
  - Evolving a solution
- **Genetic Algorithms**
  - Modeling search as evolution
    - Mutation
    - Crossover
    - Survival of the fittest
    - Survival of the most diverse
- **Conclusions**
**Motivation: Evolution**

- Evolution through natural selection
  - Individuals pass on traits to offspring
  - Individuals have different traits
  - Fittest individuals survive to produce more offspring
  - Over time, variation can accumulate
    - Leading to new species
Motivation: Evolution

• Passing on traits to offspring
  – Chromosomes carry genes for different traits
    • Usually chromosomes paired - one from each parent
  – Chromosomes are duplicated before mating
    • Crossover mixes genetic material from chromosomes
  – Each parent produces one single chromosome cell
  – Mating joins cells
  – Mutation: error in duplication -> different gene
Evolution

- Variation: Arises from crossover & mutation
  - Crossover: Produces new gene combinations
  - Mutation: Produces new genes

- Different traits lead to different fitnesses
Simulated Evolution

• Evolving a solution
• Begin with population of individuals
  – Individuals = candidate solutions ~ chromosomes
• Produce offspring with variation
  – Mutation: change features
  – Crossover: exchange features between individuals
• Apply natural selection
  – Select “best” individuals to go on to next generation
• Continue until satisfied with solution
Genetic Algorithms Applications

- Search parameter space for optimal assignment
  - Not guaranteed to find optimal, but can approach
- Classic optimization problems:
  - E.g. Traveling Salesman Problem
- Program design (“Genetic Programming”)
- Aircraft carrier landings
Question

- How can we encode TSP as GA?
- Demo
Genetic Algorithm Example

- **Cookie recipes** (Winston, AI, 1993)
  - As evolving populations
- **Individual = batch of cookies**
  - Quality: 0-9
    - Chromosomes = 2 genes: 1 chromosome each
      - Flour Quantity, Sugar Quantity: 1-9
- **Mutation:**
  - Randomly select Flour/Sugar: +/- 1 [1-9]
- **Crossover:**
  - Split 2 chromosomes & rejoin; keeping both
Mutation & Crossover

Mutation:

2 1
2 2

Crossover:

2 2
1 3
Fitness

- Natural selection: Most fit survive
- Fitness = Probability of survival to next gen
- Question: How do we measure fitness?
  - “Standard method”: Relate fitness to quality

  - \( f_i = q_i / \sum_j q_j \)

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
<th>Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>4</td>
<td>0.4</td>
</tr>
<tr>
<td>3 1</td>
<td>3</td>
<td>0.3</td>
</tr>
<tr>
<td>1 2</td>
<td>2</td>
<td>0.2</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Genetic Algorithms Procedure

- Create an initial population (1 chromosome)
- Mutate 1+ genes in 1+ chromosomes
  - Produce one offspring for each chromosome
- Mate 1+ pairs of chromosomes with crossover
- Add mutated & offspring chromosomes to pop
- Create new population
  - Best + randomly selected (biased by fitness)
GA Design Issues

- Genetic design:
  - Identify sets of features = genes; Constraints?
- Population: How many chromosomes?
  - Too few => inbreeding; Too many=>too slow
- Mutation: How frequent?
  - Too few=>slow change; Too many=> wild
- Crossover: Allowed? How selected?
- Duplicates?
GA Design: Basic Cookie GA

- Genetic design:
  - Identify sets of features: 2 genes: flour+sugar;1-9

- Population: How many chromosomes?
  - 1 initial, 4 max

- Mutation: How frequent?
  - 1 gene randomly selected, randomly mutated

- Crossover: Allowed? No

- Duplicates? No

- Survival: Standard method
## Example

### Generation 0:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

### Generation 1:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### Generation 2:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

### Mutation of 2

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

### Generation 3:

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Basic Cookie GA Results

- Results are for 1000 random trials
  - Initial state: 1 1-1, quality 1 chromosome
- On average, reaches max quality (9) in 16 generations
- Best: max quality in 8 generations

Conclusion:
- Low dimensionality search
  - Successful even without crossover
Adding Crossover

- Genetic design:
  - Identify sets of features: 2 genes: flour+sugar; 1-9
- Population: How many chromosomes?
  - 1 initial, 4 max
- Mutation: How frequent?
  - 1 gene randomly selected, randomly mutated
- Crossover: Allowed? Yes, select random mates; cross at middle
- Duplicates? No
- Survival: Standard method
Basic Cookie GA+Crossover Results

- Results are for 1000 random trials
  - Initial state: 1 1-1, quality 1 chromosome
- On average, reaches max quality (9) in 14 generations
- Conclusion:
  - Faster with crossover: combine good in each gene
  - Key: Global max achievable by maximizing each dimension independently - reduce dimensionality
Solving the Moat Problem

- Problem:
  - No single step mutation can reach optimal values using standard fitness (quality=0 => probability=0)

- Solution A:
  - Crossover can combine fit parents in EACH gene

- However, still slow: 155 generations on average
Questions

- How can we avoid the 0 quality problem?
- How can we avoid local maxima?
Rethinking Fitness

• Goal: Explicit bias to best
  – Remove implicit biases based on quality scale

• Solution: Rank method
  – Ignore actual quality values except for ranking
    • Step 1: Rank candidates by quality
    • Step 2: Probability of selecting ith candidate, given that i-1 candidate not selected, is constant p.
      – Step 2b: Last candidate is selected if no other has been
    • Step 3: Select candidates using the probabilities
## Rank Method

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Quality</th>
<th>Rank</th>
<th>Std. Fitness</th>
<th>Rank Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>4</td>
<td>1</td>
<td>0.4</td>
<td>0.667</td>
</tr>
<tr>
<td>1 3</td>
<td>3</td>
<td>2</td>
<td>0.3</td>
<td>0.222</td>
</tr>
<tr>
<td>1 2</td>
<td>2</td>
<td>3</td>
<td>0.2</td>
<td>0.074</td>
</tr>
<tr>
<td>5 2</td>
<td>1</td>
<td>4</td>
<td>0.1</td>
<td>0.025</td>
</tr>
<tr>
<td>7 5</td>
<td>0</td>
<td>5</td>
<td>0.0</td>
<td>0.012</td>
</tr>
</tbody>
</table>

Results: Average over 1000 random runs on Moat problem
- 75 Generations (vs 155 for standard method)
No 0 probability entries: Based on rank not absolute quality
Diversity

• Diversity:
  – Degree to which chromosomes exhibit different genes
  – Rank & Standard methods look only at quality
  – Need diversity: escape local min, variety for crossover
  – “As good to be different as to be fit”
Rank-Space Method

• Combines diversity and quality in fitness
• Diversity measure:
  – Sum of inverse squared distances in genes
    \[ \sum_{i} \frac{1}{d_i^2} \]
• Diversity rank: Avoids inadvertent bias
• Rank-space:
  – Sort on sum of diversity AND quality ranks
  – Best: lower left: high diversity & quality
### Rank-Space Method

**W.r.t. highest ranked 5-1**

<table>
<thead>
<tr>
<th>Chromosome</th>
<th>Q</th>
<th>D</th>
<th>D Rank</th>
<th>Q Rank</th>
<th>Comb Rank</th>
<th>R-S Fitness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 4</td>
<td>4</td>
<td>0.04</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.667</td>
</tr>
<tr>
<td>3 1</td>
<td>3</td>
<td>0.25</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>0.025</td>
</tr>
<tr>
<td>1 2</td>
<td>2</td>
<td>0.059</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>0.222</td>
</tr>
<tr>
<td>1 1</td>
<td>1</td>
<td>0.062</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>0.012</td>
</tr>
<tr>
<td>7 5</td>
<td>0</td>
<td>0.05</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0.074</td>
</tr>
</tbody>
</table>

Diversity rank breaks ties
After select others, sum distances to both
Results: Average (Moat) 15 generations
GA’s and Local Maxima

• Quality metrics only:
  – Susceptible to local max problems

• Quality + Diversity:
  – Can populate all local maxima
    • Including global max
  – Key: Population must be large enough
Genetic Algorithms

• Evolution mechanisms as search technique
  – Produce offspring with variation
    • Mutation, Crossover
  – Select “fittest” to continue to next generation
    • Fitness: Probability of survival
      – Standard: Quality values only
      – Rank: Quality rank only
      – Rank-space: Rank of sum of quality & diversity ranks

• Large population can be robust to local max