Solving Constraint Programs using Backtrack Search and Forward Checking

Slides draw upon material from:
6.034 notes, by Tomas Lozano Perez
AIMA, by Stuart Russell & Peter Norvig
Constraint Processing, by Rina Dechter

Brian C. Williams
16.410-13
September 27th, 2010

Assignments

• Remember:

• Reading:
  • Today: [AIMA] Ch. 6.2-5; Constraint Satisfaction.
  • Wednesday: Operator-based Planning [AIMA] Ch. 10 “Graph Plan,” by Blum & Furst, posted on Stellar.

• To Learn More: Constraint Processing, by Rina Dechter
  – Ch. 5: General Search Strategies: Look-Ahead
  – Ch. 6: General Search Strategies: Look-Back
  – Ch. 7: Stochastic Greedy Local Search
Constraint Problems are Everywhere

Constraint Satisfaction Problems (CSP)

Input: A Constraint Satisfaction Problem is a triple $<V,D,C>$, where:

- $V$ is a set of variables $V_i$
- $D$ is a set of variable domains,
  - The domain of variable $V_i$ is denoted $D_i$
- $C =$ is a set of constraints on assignments to $V$
  - Each constraint $C_i = <S_i,R_i>$ specifies allowed variable assignments.
  - $S_i$ the constraint’s scope, is a subset of variables $V$.
  - $R_i$ the constraint’s relation, is a set of assignments to $S_i$.

Output: A full assignment to $V$, from elements of $V$’s domain, such that all constraints in $C$ are satisfied.
Constraint Modeling (Programming) Languages

**Features** Declarative specification of the problem that separates the formulation and the search strategy.

**Example**: Constraint Model of the Sudoku Puzzle in Number Jack (http://4c110.ucc.ie/numberjack/home)

```python
matrix = Matrix(N*N,N*N,1,N*N)
sudoku = Model( 
    [AllDiff(row) for row in matrix.row],
    [AllDiff(col) for col in matrix.col],
    [AllDiff(matrix[x:x+N, y:y+N].flat) 
      for x in range(0,N*N,N) 
      for y in range(0,N*N,N)] )
```

Constraint Problems are Everywhere

© Source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/fairuse.
What is the Complexity of AC-1?

AC-1(CSP)
Input: A network of constraints CSP = <X, D, C>.
Output: CSP', the largest arc-consistent subset of CSP.
1. repeat
2. for every $c_{ij} \in C$, 
3. Revise($x_i$, $x_j$) 
4. Revise($x_j$, $x_i$) 
5. endfor 
6. until no domain is changed.

Assume:
• There are $n$ variables.
• Domains are of size at most $k$.
• There are $e$ binary constraints.
What is the Complexity of AC-1?

Assume:
• There are $n$ variables.
• Domains are of size at most $k$.
• There are $e$ binary constraints.

Which is the correct complexity?
1. $O(k^2)$
2. $O(enk^2)$
3. $O(enk^3)$
4. $O(nek)$

---

Revise: A directed arc consistency procedure

Revise $(x_i, x_j)$

Input: Variables $x_i$ and $x_j$ with domains $D_i$ and $D_j$ and constraint relation $R_{ij}$.

Output: pruned $D_i$, such that $x_i$ is directed arc-consistent relative to $x_j$.

1. for each $a_i \in D_i$ 
2. if there is no $a_j \in D_j$ such that $<a_i, a_j> \in R_{ij}$ * $O(k)$
3. then delete $a_i$ from $D_i$.
4. endif
5. endfor

Complexity of Revise?
$= O(k^2)$

where $k = \max_i |D_i|$
Full Arc-Consistency via AC-1

AC-1(CSP)
Input: A network of constraints CSP = <X, D, C>.
Output: CSP', the largest arc-consistent subset of CSP.
1. repeat
2. for every c_{ij} \in C, \quad O(2e*revise)
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. endfor
6. until no domain is changed.

Complexity of AC-1?
= O(nk*e*revise)\quad \text{where} \quad k = \max_i |D_i| 
= O(enk^3) 
\text{\quad n = |X|, e = |C|}

What is the Complexity of Constraint Propagation using AC-3?

Assume:
• There are n variables.
• Domains are of size at most k.
• There are e binary constraints.
Which is the correct complexity?
1. O(k^2)
2. O(ek^2)
3. O(ek^3)
4. O(ek)
Full Arc-Consistency via AC-3

AC-3(CSP)

**Input:** A network of constraints CSP = <X, D, C>.

**Output:** CSP’, the largest arc-consistent subset of CSP.

1. for every \( c_{ij} \in C \),
2. \( \text{queue} \leftarrow \text{queue} \cup \{<x_i, x_j>, <x_j, x_i>\} \)
3. endfor
4. while \( \text{queue} \neq {} \)
5. select and delete arc \( <x_i, x_j> \) from \( \text{queue} \)
6. \( \text{Revise}(x_i, x_j) \)
7. if \( \text{Revise}(x_i, x_j) \) caused a change in \( D_i \),
8. \( \text{then} \) \( \text{queue} \leftarrow \text{queue} \cup \{<x_k, x_i> \mid k \neq i, k \neq j\} \)
9. endif
10. endwhile

Complexity of AC-3?

\[ = O(e+ek^2) = O(ek^3) \]

where \( k = \max_i |D_i|, n = |X|, e = |C| \)

---

Is arc consistency sound and complete?

An arc consistent solution selects a value for every variable from its arc consistent domain.

**Soundness:** All solutions to the CSP are arc consistent solutions?

- Yes
- No

**Completeness:** All arc-consistent solutions are solutions to the CSP?

- Yes
- No

Brian Williams, Fall 10
Incomplete: Arc consistency doesn’t rule out all infeasible solutions

Graph Coloring

- Arc consistent, but no solutions.
- Arc consistent, but 2 solutions, not 8.

To Solve CSPs We Combine

1. Arc consistency (via constraint propagation)
   - Eliminates values that are shown locally to not be a part of any solution.
2. Search
   - Explores consequences of committing to particular assignments.

Methods That Incorporate Search:
- Standard Search
- Back Track Search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair (IR)
- Conflict-directed Back Jumping (CBJ)
Solving CSPs using Generic Search

- **State**
  - Partial assignment to variables, made thus far.

- **Initial State**
  - No assignment.

- **Operator**
  - Creates new assignment \( (X_i = v_{ij}) \)
    - Select any unassigned variable \( X_i \)
    - Select any one of its domain values \( v_{ij} \)
  - Child extends parent assignments with new.

- **Goal Test**
  - All variables are assigned.
  - All constraints are satisfied.

- **Branching factor?**
  - Sum of domain size of all variables \( O(|v|^*|d|) \).

- **Performance?**
  - Exponential in the branching factor \( O((|v|^*|d|)^{|v|}) \).

Search Performance on N Queens

- Standard Search
- Backtracking
- A handful of queens
Solving CSPs with Standard Search

Standard Search:
- Children select any value for any variable \([O(|v|^*|d|)]\).
- Test complete assignments for consistency against CSP.

Observations:
1. The order in which variables are assigned does not change the solution.
   - Many paths denote the same solution,
     - \((|v|!)\), expand only one path (i.e., use one variable ordering).
2. We can identify a dead end before we assign all variables.
   - Extensions to inconsistent partial assignments are always inconsistent.
     - Check consistency after each assignment.

Back Track Search (BT)

1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.

Preselect order of variables to assign
Assign designated variable
**Back Track Search (BT)**

1. Expand assignments of **one variable** at each step.
2. Pursue **depth first**.
3. Check **consistency** after each expansion, and backup.

---

**Procedure Backtracking(<X,D,C>)**

**Input:** A constraint network \( R = <X, D, C> \)

**Output:** A solution, or notification that the network is inconsistent.

\[
\begin{align*}
i &\leftarrow 1; \quad a_i = \emptyset; & \text{Initialize variable counter, assignments,} \\
D_i' &\leftarrow D_i; & \text{Copy domain of first variable.} \\
\text{while } 1 \leq i \leq n&: & \\
\text{instantiate } x_i &\leftarrow \text{Select-Value();} & \text{Add to assignments } a_i \\
\text{if } x_i \text{ is null} & & \text{No value was returned,} \\
\quad &\leftarrow i - 1; & \text{then backtrack} \\
\text{else} & & \text{else step forward and} \\
\quad i &\leftarrow i + 1; & \text{copy domain of next variable} \\
D_i' &\leftarrow D_i; & \\
\text{end while} & & \\
\text{if } i = 0 & & \text{return "inconsistent"} \\
\text{else} & & \\
\quad \text{return } a_i , \text{ the instantiated values of } \{x_i, \ldots, x_n\} \\
\text{end procedure} & & \\
\end{align*}
\]
Procedure Select-Value()

Output: A value in $D'_i$ consistent with $a_{i-1}$, or null, if none.

\[
\text{while } D'_i \text{ is not empty} \\
\quad \text{select an arbitrary element } a \in D'_i \text{ and remove } a \text{ from } D'_i; \\
\quad \text{if consistent}(a_{i-1}, x_i = a) \\
\quad \quad \text{return } a; \\
\text{end while} \\
\text{return null} \quad \quad \text{no consistent value} \\
\text{end procedure}
\]
Combining Backtracking and Limited Constraint Propagation

Initially: Prune domains using constraint propagation (optional)
Loop:
  • If complete consistent assignment, then return it, Else…
  • Choose unassigned variable.
  • Choose assignment from variable’s pruned domain.
  • Prune (some) domains using Revise (i.e., arc-consistency).
  • If a domain has no remaining elements, then backtrack.

Question: Full propagation is $O(ek^3)$, How much propagation should we do?
Very little (except for big problems)
Forward Checking (FC)
  • Check arc consistency ONLY for arcs that terminate on the new assignment [$O(e k)$ total].

Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

R

V₁ assignments
V₂ assignments
V₃ assignments

Note: No need to check new assignment against previous assignments

1. Perform initial pruning.

Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

R

V₁ assignments
V₂ assignments
V₃ assignments

3. We have a conflict whenever a domain becomes empty.
   • Backtrack

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.
   - Backtrack

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.
   - Backtrack
   - Restore domains

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

1. Perform initial pruning.

3. We have a conflict whenever a domain becomes empty.
   - Backtrack
   - Restore domains

Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.
   - Backtrack
   - Restore domains

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

V₁ assignments
V₂ assignments
V₃ assignments

3. We have a conflict whenever a domain becomes empty.
   - Backtrack
   - Restore domains

1. Perform initial pruning.
Backtracking with Forward Checking (BT-FC)

2. After selecting each assignment, remove any values of neighboring domains that are inconsistent with the new assignment.

3. We have a conflict whenever a domain becomes empty.
   - Backtrack
   - Restore domains

1. Perform initial pruning.
Procedure Backtrack-Forward-Checking(<x,D,C>)

Input: A constraint network R = <X, D, C>
Output: A solution, or notification the network is inconsistent.
Note: Maintains n domain copies D' for resetting, one for each search level i.

D' \leftarrow D \text{ for } 1 \leq i \leq n; \quad \text{(copy all domains)}
i \leftarrow 1; \quad a_i = \emptyset \quad \text{(init variable counter, assignments)}

\text{while } 1 \leq i \leq n \text{ \quad (add to assignments, making } a_i) 

\begin{align*}
\text{instantiate } x_i \leftarrow \text{Select-Value-FC();} & \quad \text{(no value was returned)} \\
\text{if } x_i \text{ is null} & \quad \text{(backtrack)} \\
\quad \text{reset each } D'_k \text{ for } k > i, \text{ to its value before } x_i \text{ was last instantiated;}
\quad i \leftarrow i - 1; & \quad \text{(backtrack)} \\
\text{else} & \\
\quad \text{step forward)} \\
\end{align*}

\text{end while}

\text{if } i = 0 \quad \text{return } "\text{inconsistent}" \quad \text{Constraint Processing,} \\
\text{else} \quad \text{by R. Dechter} \\
\quad \text{return } a_i, \text{ the instantiated values of } \{x_i, \ldots, x_n\} \quad \text{pgs 131-4, 141}

end procedure

Procedure Select-Value-FC()

Output: A value in D'_i consistent with a_{i-1}, or null, if none. \quad O(ek^2)

\text{while } D'_i \text{ is not empty}

\begin{align*}
\text{select an arbitrary element } a \in D'_i \text{ and remove } a \text{ from } D'_i; & \\
\text{for all } k, i < k \leq n & \\
\text{for all values } b \text{ in } D'_k & \\
\quad \text{if not consistent}(a_{i-1}, x_i = a, x_k = b) & \\
\quad \text{remove } b \text{ from } D'_k; & \\
\text{end for} & \\
\text{if } D'_k \text{ is empty} & \quad (x_i = a \text{ leads to a dead-end, don't select } a) \\
\quad \text{reset each } D'_k, i < k \leq n \text{ to its value before } a \text{ was selected;} & \\
\text{else} & \\
\quad \text{return } a; & \\
\text{end while} & \\
\text{return null} & \\
\text{end procedure} & \quad \text{Constraint Processing,} \quad \\
& \quad \text{by R. Dechter} \quad \\
& \quad \text{pgs 131-4, 141}
Search Performance on N Queens

- Standard Search
- Backtracking
- BT with Forward Checking
- Dynamic Variable Ordering

- A handful of queens
- About 15 queens
- About 30 queens

BT-FC with dynamic ordering

Traditional backtracking uses a fixed ordering over variables & values.

Typically better to choose ordering dynamically as search proceeds.

- Most Constrained Variable
  When doing forward-checking, pick variable with fewest legal values in domain to assign next.
  ⇒ minimizes branching factor.

- Least Constraining Value
  Choose value that rules out the smallest number of values in variables connected to the chosen variable by constraints.
  ⇒ Leaves most options to finding a satisfying assignment.
Which country should we color next?  \(\rightarrow\) E most-constrained variable (smallest domain).

What color should we pick for it?  \(\rightarrow\) RED least-constraining value (eliminates fewest values from neighboring domains).

**Procedure Dynamic-Var-Forward-Checking(<x,D,C>)**

```
Input: A constraint network \(R = <X, D, C>\)
Output: A solution, or notification the network is inconsistent.

\[D'_i \leftarrow D_i\text{ for } 1 \leq i \leq n;\]
\[i \leftarrow 1;\]
\[s = \min_{1 \leq j \leq n} |D'_j|;\]
\[x_i \leftarrow x_s;\]

while \(1 \leq i \leq n\) do
  instantiate \(x_i \leftarrow \text{Select-Value-FC}()\);  \(\text{Select value (dynamic) and add to assignments, } \mathcal{A}\)
  if \(x_i\) is null do
    reset each \(D'_k\) for \(k > i\), to its value before \(x_i\) was last instantiated;
    \(i \leftarrow i + 1;\)
  else
    if \(i < n\) do
      \(i \leftarrow i + 1;\)
      \(s = \min_{1 \leq j \leq i} |D'_j|;\)
      \(x_i \leftarrow x_s;\)
    else
      \(i \leftarrow i + 1;\)
  end if
end while

if \(i = 0\) then
  return "inconsistent";
else
  return \(\mathcal{A}\), the instantiated values of \(\{x_n, \ldots, x_1\}\)
end procedure
```

Constraint Processing, by R. Dechter
pgs 137-140
Search Performance on N Queens

- Standard Search
- Backtracking
- BT with Forward Checking
- Dynamic Variable Ordering
- Iterative Repair
- Conflict-directed Back Jumping
- A handful of queens
- About 15 queens
- About 30 queens
- About 1,000 queens

Incremental Repair (Min-Conflict Heuristic)

1. Initialize a candidate solution using a “greedy” heuristic.
   - gets the candidate “near” a solution.

2. Select a variable in a conflict and assign it a value that minimizes the number of conflicts (break ties randomly).

The heuristic is used in a local hill-climber (without or with backup).

<table>
<thead>
<tr>
<th>R R R: 3</th>
<th>BRR</th>
<th>GRR</th>
<th>RGR</th>
<th>RRG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Min-Conflict Heuristic

Pure hill climber (w/o backtracking) gets stuck in local minima:

- Add random moves to attempt to get out of minima.
- Add weights on violated constraints and increase weight every cycle the constraint remains violated.

Performance on n-queens.

GSAT: Randomized hill climber used to solve propositional logic SATisfiability problems.

To Solve CSP <X,D,C> We Combine:

1. Reasoning - Arc consistency via constraint propagation
   - Eliminates values that are shown locally to not be a part of any solution.

2. Search
   - Explores consequences of committing to particular assignments.

Methods That Incorporate Search:

- Standard Search
- Back Track Search (BT)
- BT with Forward Checking (FC)
- Dynamic Variable Ordering (DV)
- Iterative Repair (IR)
- Conflict-directed Back Jumping (CBJ)
Next Lecture: Back Jumping

**Backtracking** At dead end, backup to the most recent variable.

**Backjumping** At dead end, backup to the most recent variable that eliminated some value in the domain of the dead end variable.