CSP Search Concluded: Arc Consistency (AC-3)
Intro to Games and
Game Tree Search

William H. Hsu
Department of Computing and Information Sciences, KSU

KSOL course page: http://snipurl.com/v9v3
Course web site: http://www.kddresearch.org/Courses/CIS730
Instructor home page: http://www.cis.ksu.edu/~bhsu

Reading for Next Class:
Sections 6.4 – 6.8, p. 171 – 185, Russell & Norvig 2nd edition

Outside references:
CSP examples, M. Hauskrecht (U. Pittsburgh) – http://tr.im/zdG6
Notes on CSP, R. Barták (Charles U., Prague) – http://tr.im/zdGE

Lecture Outline

- Reading for Next Class: 6.4 – 6.8 (p. 171 – 185), R&N 2e
- Last Class: Sections 5.1 – 5.3 on Constraint Satisfaction Problems
  - CSPs: definition, examples
  - Heuristics for variable selection, value selection
  - Two algorithms: backtracking search, “one-step” forward checking
- Today: Rest of CSP, 5.4-5.5, p. 151-158; Games Intro, 6.1-6.3, p. 161-174
  - Third algorithm: constraint propagation by arc consistency (AC-3)
  - Scaling up to NP-hard problems
- This Week: CSP and Game Tree Search
  - Rudiments of game theory
  - Zero-sum games vs. cooperative games
  - Perfect information vs. imperfect information
  - Minimax
  - Alpha-beta (α - β) pruning
  - Randomness and expectiminimax
- Next: From Heuristics to General Knowledge Representation
Farmer, Fox, Goose, & Grain

State Space: Review

F = Farmer  X = fox  G = Goose
N = grain  ~ = River

Adapted from slide © 2008 B. R. Maxim, Univ. of Michigan – Dearborn
CIS 479/579 Artificial Intelligence  http://tr.im/zdhV
**CSPs: Review**

Standard search problem: state is a “black box”—any old data structure that supports goal test, eval, successor

CSP:
state is defined by variables $X_i$ with values from domain $D$,
goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Simple example of a formal representation language
Allows useful general-purpose algorithms with more power than standard search algorithms

---

**Map Coloring Example: Review**

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}\}
Algorithm 1 — Backtracking Search: Review

function BACKTRACKING-SEARCH(csp) returns solution/failure
return RECURSIVE-BACKTRACKING([], csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
if assignment is complete then return assignment
var — SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
  if value is consistent with assignment given CONSTRAINTS[csp] then
    add {var = value} to assignment
    result — RECURSIVE-BACKTRACKING(assignment, csp)
    if result ≠ failure then return result
  remove {var = value} from assignment
return failure

Backtracking Example: Review
**Variable and Value Selection:**

**Review**
- **MRV**
  - Minimum remaining values (MRV): choose the variable with the fewest legal values.
  - Tie-breaker among MRV variables.
  - Degree heuristic: choose the variable with the most constraints on remaining variables.

**LCV**
- Value selection (for a given variable).
- Given a variable, choose the least Constraining value.
- The one that rules out the fewest values in the remaining variables.

Combining these heuristics makes 1000 queens feasible.

Based on slides © 2004 S. Russell & P. Norvig. Reused with permission.

---

**Value Propagation:**

**Constraint Prop Without Lookahead**

- **Constraint propagation**
  - Value propagation. Infers:
    - equations from the set of equations defining the partial assignment, and a constraint

No equations/disequations are inferred

No equations/disequations are inferred

© 2005 M. Hauskrecht, University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence
http://www.cs.pitt.edu/~milos/courses/cs2710/
Algorithm 2 — Forward Checking: Review

Idea: Keep track of remaining legal values for unassigned variables. Terminate search when any variable has no legal values.

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally.

Forward Checking
With “One-Step” Constraint Prop

- Constraint propagation

  Forward checking: Infers:
  - disequations from a set of equations defining the partial assignment, and a constraint
  - Equations through the exhaustion of alternatives

Invalid assignment

Based on slides © 2004 S. Russell & P. Norvig. Reused with permission.

© 2005 M. Hauskrecht, University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence
http://www.cs.pitt.edu/~milos/courses/cs2710/
Algorithm 3 — Arc Consistency [1]

Simplest form of propagation makes each arc consistent

\[ X \rightarrow Y \text{ is consistent iff} \]

for every value \( x \) of \( X \) there is some allowed \( y \)

If \( X \) loses a value, neighbors of \( X \) need to be rechecked

Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor or after each assignment

Algorithm 3 — Arc Consistency [2]

AC-3 Definition

function AC-3(\( \sigma \)) returns the CSP, possibly with reduced domains
inputs: \( \sigma \), a binary CSP with variables \( \{X_1, X_2, \ldots, X_n\} \)
local variables: queue, a queue of arcs, initially all the arcs in \( \sigma \)

while queue is not empty do

\( (X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\text{queue}) \)

if REMOVE-INCONSISTENT-VALUES(\( X_i, X_j \)) then

for each \( X_k \) in NEIGHBORS(\( X_i \)) do

add \( (X_k, X_i) \) to queue

end


function REMOVE-INCONSISTENT-VALUES(\( X_i, X_j \)) returns true if succeeds
removed — false

for each \( x \) in Domain[\( X_i \)] do

if no value \( y \) in Domain[\( X_j \)] allows \( (x,y) \) to satisfy the constraint \( X_i \leftarrow X_j \)

then delete \( x \) from Domain[\( X_i \)]; removed — true

return removed

\( O(n^2d^3) \), can be reduced to \( O(n^2d^3) \) (but detecting all is NP-hard)

Forward Checking
With Full Arc Consistency

- Constraint propagation
  Arc consistency. Infers:
  - disequations from the set of equations and disequations defining the partial assignment, and a constraint
  - equations through the exhaustion of alternatives

© 2005 M. Hauskrecht, University of Pittsburgh
CS 2710 Foundations of Artificial Intelligence
http://www.cs.pitt.edu/~milos/courses/cs2710/

Intro to Games:
Outline

- Games
- Perfect play
  - minimax decisions
  - α-β pruning
- Resource limits and approximate evaluation
- Games of chance
- Games of imperfect information

Games versus Search

“Unpredictable” opponent $\Rightarrow$ solution is a strategy specifying a move for every possible opponent reply

Time limits $\Rightarrow$ unlikely to find goal, must approximate

Plan of attack:

- Computer considers possible lines of play (Babbage, 1846)
- Algorithm for perfect play (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First chess program (Turing, 1951)
- Machine learning to improve evaluation accuracy (Samuel, 1952–57)
- Pruning to allow deeper search (McCarthy, 1956)

Types of Games

<table>
<thead>
<tr>
<th>Deterministic</th>
<th>Chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td>backgammon, monopoly</td>
</tr>
<tr>
<td>battleship, blind tic-tac-toe</td>
<td>bridge, poker, scrabble, nuclear war</td>
</tr>
</tbody>
</table>

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.
**Game Tree:**

2-Player, Deterministic, Turns

---

**Minimax [1]: Example**

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value

= best achievable payoff against best play

E.g., 2-ply game:
**Minimax [2]: Algorithm**

```python
function Minimax-Decision(state) returns an action
    inputs: state, current state in game
    return the a in ACTIONS(state) maximizing Min-Value(Result(a, state))

function Max-Value(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, Min-Value(s))
    return v

function Min-Value(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, Max-Value(s))
    return v
```

**Minimax [3]: Properties**

- **Complete??**
  Yes, if tree is finite (chess has specific rules for this)

- **Optimal??**
  Yes, against an optimal opponent. Otherwise??

- **Time complexity??** $O(b^m)$

- **Space complexity??** $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games
   \[ \Rightarrow \text{exact solution completely infeasible} \]

But do we need to explore every path?
**Figure 6.5 p. 168 R&N 2e**

**What are \( \alpha, \beta \) values here?**

---

**Alpha-Beta (\( \alpha - \beta \)) Pruning [1]: Example**

*Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.*

---

**Alpha-Beta (\( \alpha - \beta \)) Pruning [2]: Algorithm**

*Adapted from slides © 2004 S. Russell & P. Norvig. Reused with permission.*
**Alpha-Beta (α-β) Pruning [3]: Properties**

Pruning does not affect final result.
Good move ordering improves effectiveness of pruning.
With "perfect ordering," time complexity = $O(b^{m/2})$.

- **Depth-limited**
- **Iterative deepening**
- **Memory-bounded**

A simple example of the value of reasoning about which computations are relevant (a form of metareasoning).
Unfortunately, $35^{50}$ is still impossible!

**Can We Do Better?**

**Idea: Adapt Resource-Bounded Heuristic Search Techniques**

- Depth-limited
- Iterative deepening
- Memory-bounded

---

**Static Evaluation Functions**

For chess, typically linear weighted sum of features:

$$Eval(s) = w_1f_1(s) + w_2f_2(s) + \ldots + w_nf_n(s)$$

- e.g., $w_1 = 9$ with
  - $f_1(s) = \text{(number of white queens)} - \text{(number of black queens)}$, etc.
### CSP Techniques
- Variable selection heuristic: Minimum Remaining Values (MRV)
- Value selection heuristic: Least Constraining Value (LCV)
- Constraint satisfaction search algorithms: using variable and value selection

### Detailed CSP Example: 3-Coloring of Planar Graph

### Algorithms
- Value propagation and backtracking
- Forward checking: simple constraint propagation, arc consistency (AC-3)

### Games and Game Theory
- Single-player vs. multi-player vs. two-player
- Cooperative vs. competitive (esp. zero sum)
- Uncertainty
  - Imperfect information vs. perfect information
  - Deterministic vs. games with element of chance

### Game Tree Search
- Minimax, alpha-beta (α - β) pruning
- Static evaluation functions