Meeting 8: Searching for Constraint Satisfaction
Reading: AIAMA 6.3-6.4

Today’s Schedule:

- Backtracking Search
- Ordering Heuristics
- Local Search
function AC-3(csp) returns false if an inconsistency is found and true otherwise
inputs: csp, a binary CSP with components (X, D, C)
local variables: queue, a queue of arcs, initially all the arcs in csp

while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
        if size of D_i = 0 then return false
        for each X_k in X_i.NEIGHBORS - {X_j} do
            add (X_k, X_i) to queue
    return true

function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
revised ← false
for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
        delete x from D_i
        revised ← true
return revised

Figure 6.3 The arc-consistency algorithm AC-3. After applying AC-3, either every variable is consistent or the decoder indicates a failure.
Exercise

A CSP:
\[ X : \{X_1, X_2, X_3\} \]
\[ D : X_1 \in \{A, B, C\}, X_2 \in \{A, C, E\}, X_3 \in \{E, B, C, A\} \]
\[ C : \{X_1 = X_2 \neq X_3\} \]

Apply AC-3 to check for an inconsistency.
Searching on CSPs

When inference alone cannot solve a CSP by pruning all domains to a single value, there are two basic approaches to searching for solutions:

- partial assignment
- complete assignment
Another example where AC-3 does not help

Consider the following puzzle, similar to a crossword

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

Variables: 1A, 1D, 3A, 2D

Domains:
- for 1A aa,bb
- for 1D ac,bd
- for 3A cc,dd
- for 2D ad,bc

Constraints: first(1A) == first(1D), etc.
The partial assignment tree

choose 1A

choose 1D

conflict!

conflict!
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment then
            add \{var = value\} to assignment
            inferences ← INERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result ← BACKTRACK(assignment, csp)
                if result ≠ failure then
                    return result
            remove \{var = value\} and inferences from assignment
    return failure

Figure 6.5 A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter ???. By varying the functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, we can implement the general-purpose heuristics discussed in the text. The function INERENCE can optionally be used to impose arc-, path-, or k-consistency, as desired. If a value choice leads to failure (noticed either by INERENCE or by BACKTRACK), then value assignments (including those made by INERENCE) are removed from the current assignment and a new value is tried.
Consider the following CSP.

\[ X = X_1, X_2, X_3 \]

\[ D = X_i \in a \ldots z \]

\[ C = (X_1, X_2, X_3) \in \text{Set of three letter lower-case English words} \]

Manually perform backtracking search. Select unassigned variables left-to-right, select values is alphabetical order, no inference.
Selecting Unassigned Variables

The order in which variable are selected, SELECT-UNASSIGNED-VARIABLE, is important in many CSPs. Two common heuristics:

- minimum remaining values (MRV) - choose the variable with the smallest domain remaining
- degree heuristic - choose variable with largest number of unassigned neighbors in the constraint graph

These can be used together, sort by MRV with degree as the tie-breaker
Simple example

A CSP:
$X : \{X_1, X_2, X_3\}$
$D : X_1 \in \{A, B, C\}$, $X_2 \in \{A, C\}$, $X_3 \in \{E, B, C, A\}$
$C : \{X_1 = X_2 \neq X_3\}$

What variable would be chosen first using the MRV heuristic?
Ordering Domain Values

The function ORDER-DOMAIN-VALUES determines the order that domain values are considered. A (sometimes) useful heuristic is

- least-constraining-value (LCV), the value that rules out the fewest choices
LCV for our 3 letter word search
1st level domain size using bash

for L in {a..z} do
  echo -n $L "::"; grep "^$L..$" /usr/share/dict/words | wc -l
done

► a has 89
► b has 55
► ...
► s has 80
► ...
► q has 2
► ...
► x has 0
► y has 38
► z has 15

So a would be chosen as the first value to consider for the first letter using the least-constraining-value heuristic
Inference can be used to prune inconsistent values from the domain during the search. A simple example is forward-checking when X is assigned

for each neighbor Y of X
    delete inconsistent values from the domain of Y

Another is maintaining arc consistency (MAC)

arcs = {}
when X is assigned
    for each unassigned neighbor Y of X
        arcs.append(Y,X)
apply AC-3 using arcs as the initial queue

This recursively propagates the assignment to X
**Min-Conflicts**

```plaintext
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
inputs: csp, a constraint satisfaction problem
        max_steps, the number of steps allowed before giving up

current ← an initial complete assignment for csp
for i = 1 to max_steps do
    if current is a solution for csp then return current
    var ← a randomly chosen conflicted variable from csp.VARIABLES
    value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
    set var = value in current
return failure
```

**Figure 6.8** The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.
Warmup

What is the essential difference between the searches in sections 6.3 and 6.4?
This concludes Part I of the course.

What we have learned thus far

- How to define problems as state space search
  - Puzzles
  - Games
  - CSPs
- How to search intelligently on those spaces for solutions
  - Heuristic Search
  - MiniMax and $\alpha - \beta$ Search
  - Backtracking Search
Next Actions

- Reading on Propositional Logic, AIAMA 7.1-7.4
- Take warmup before noon on Tuesday 2/17.

Reminders:
- PS 1 due by 8am Monday.

Announcements:
- Quiz 1 will be on Tuesday 2/24