Finish CSPs

CS156

Chris Pollett

Oct 1, 2014

Outline

- Sudoku Example
- Backtracking Search For CSPs

Introduction

- We were talking about constraint satisfaction problems.
- These consisted of a set of variables, a set of domains for these variables, and a set of constraints (pairs of variables and relations) that these variables should satisfy.
- A solution to such a problem was an assignment of values from the domains to each of the variables, such that these values satisfy all the constraints.
- We formulated map-coloring, job-shop scheduling and the n-queens problems as CSPs.
- We looked at different common kinds of constraint people have considered.
- We also looked at local consistency checks such as node consistency, arc consistency, n-consistency which can be done with reasonably fast algorithms to do constraint propagation to narrow down the search space that a search algorithm might need to consider in finding solutions.
- Today, we begin by looking at sudoku formulated as a CSP and see how to do constraint propagation for this particular case.

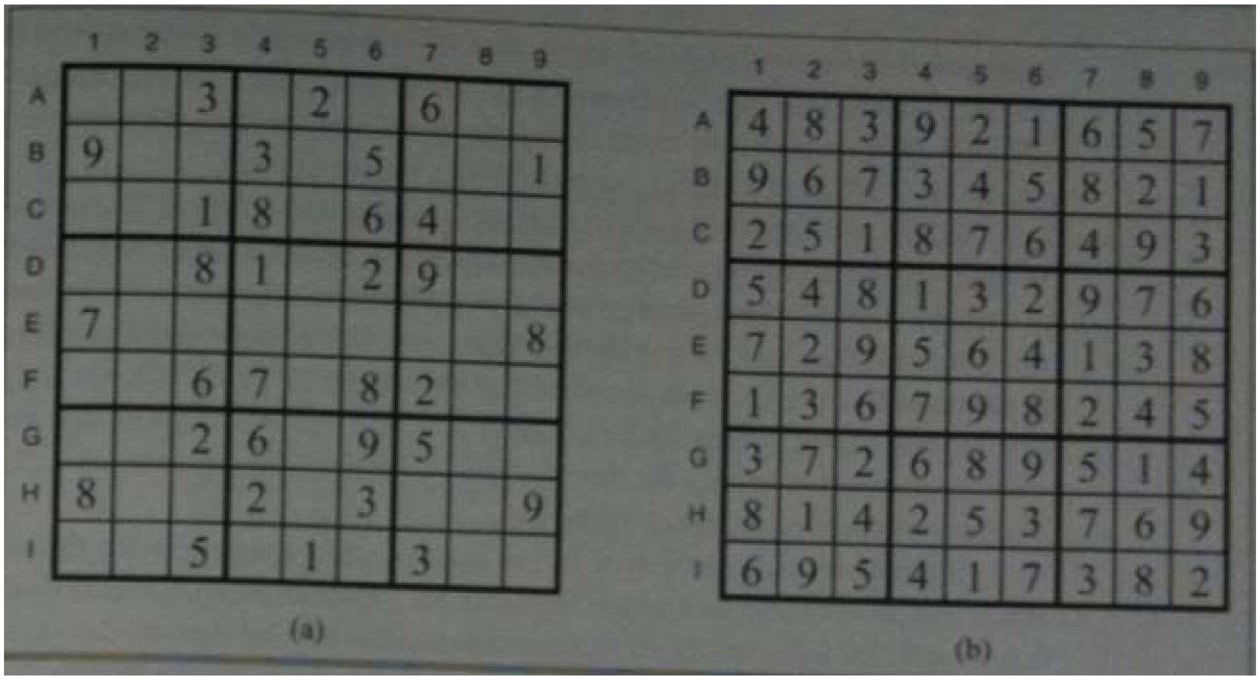
Sudoku

- A Sukodu board is a 9x9 board, 81 squares, some of which are initially filled with values from 1to 9.
- The goal is to fill in all the remaining squares such that no digit appears twice in any row, column, or 3x3 box.
- A row, column, or box is called a **unit**.
- A CSP solver can typical solve any such problem in under a 1/10 of a second.

Formulating Sudoku as a CSP

- A Sudoku puzzle can be considered a CSP with 81 variables, one for each square.
- We label there $A1$,...$A9$ for the top row to $I1$, ..$I9$ for the bottom row.
- The empty squares have domain $\{1, 2, ..., 9\}$.
- The pre-filled squares have a domain consisting of a single value.
- In addition, there are 27 different Alldiff constraints, use for each unit:
  $Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$
  $Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)$
  ...
  $Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$
  $Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)$
  ...
  $Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$ (top left 3x3 square)
  ...

Effects of consistency checks on Sudoku



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

- Notice we can expand the Alldiff's into $\binom{n}{2}$ binary constraints. For example the first row constraint, becomes $A1 \neq A2$, $A1 \neq A3$, ..., $A1 \neq A9$, $A2 \neq A3$, ...
- So we can apply the AC-3 algorithm on the result.
- Consider $E6$ in the left hand board above. From the constraints in the box, we can remove 2,8, 1, 7 from the domain of $E6$.From the constraints on the column, we can eliminate 5, 6, 2, 8, 9, and 3. This leaves E6 with a domain of {4}. So we know the answer for $E6$.
- Continuing in this fashion with other unfilled squares, it turns out that we can exactly fix all the values in the above puzzle.

- So just running AC-3 on the above solves the puzzle.

Backtracking Search for CSPs

- Unlike the Sudoku example, many CSPs cannot be solved by inference alone -- you also need to search for solutions.
- One possible way to do such a search would be to do a depth-limited search: A state would be a partial assignment, and an action would be adding $var = value$ to the assignment.
- Unfortunately, for a CSP with n variables of domain size d, the branching at the top level is nd, at the second level $(n-1)d$ ... So for n levels the tree has $n!d^n$ leaves, even though there are only d^n possible complete assignments!
- Notice the actions in a CSP are **commutative**. i.e., the order of any given set of actions has no effect on the outcome.
- Therefore, we only need consider a single variable at each node in the search tree.
- So, for example, in the color the map of Australia problem, at the root we might make the choice between $SA = red$, $SA = green$, and $SA = blue$.
- With this restriction, the number of leaves is d^n, as one would expect.
- We call the variant of depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign **backtracking search**.

Backtracking-Search

Here is some pseudo-code for backtracking search:

```
function Backtracking-Search(csp) returns a solution, or failure
    return Backtrack({},csp)

function Backtrack(assignment, csp) returns a solution, 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 := INFERENCE(csp, var, value)
                //this might do AC-3 or the like
            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
```

Making Backtracking Search More Sophisticated

Performance of backtracking can be improved by focusing on:

- Which variable should be assigned next (Select-Unassigned-Variable) and in what order should its values be tried (Order-Domain-Values)?
- What inferences should be performed at each step in the search (Inference)?
- When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

Variable and Value Ordering

- The simplest strategy for Select-Unassigned-Variable is to choose the next assignment unassigned variable in order: $\{X_1, X_2, ...\}$.
- This static variable ordering is often not very efficient. For example, after the assignment WA=red and NT=green there is only one possible value for SA, so it makes sense to assign SA=blue next rather than assign say the variable Q, even though Q might be next in the variable number ordering. This is because, by setting SA = blue we might force the values of further variables.
- This idea can be made into a heuristic. Choose the variable from amongst those that have the fewest "legal" values. This is called the **minimum-remaining-value** (MRV) heuristic. It is also known as the **most-constrained variable** heuristic and the **fail-first** heuristic.
- MRV doesn't help to choose the first region to color in the map of Australia.
- One heuristic one can use for this is the **degree heuristic**: choose the variable that is involved in the largest number of constraints on other unassigned variables. SA is the variable with the highest degree for Australia at 5.
- Using the degree heuristic on Australia, actually solves the problem without having to do backtracking.
- Once the variable has been selected, one then needs to choose the order to examine the values one could assign it.
- This can be done using the **least-constraining-value** heuristic. Pick the value that rules out the fewest choices

for the neighboring variables in the constraint graph.

Interleaving search and Inference

- So far we have seen how AC-3 and other algorithms can infer reductions in the domain of variables before we begin the search.
- It can be even more powerful in the course of search: every time we make a choice of a value for a variable, we have a new opportunity to infer domain reductions.
- **Forward checking** is one way to implement Inference in the backtracking algorithm: Whenever a variable X is assigned we then do arc consistency for it: for each unassigned variable Y that is connected to X by a constraint, delete from Y's domain any value that is inconsistent with the value chosen for C.

Intelligent Backtracking

- Our backtracking algorithm currently has a very simple policy of what to do when a branch of the search fails: back up to the preceding variable and try a different value for it.
- This is called **chronological backtracking**. There are other possible ways to do this which sometimes work better.
- Suppose in the Australia problem, we have the partial assignment {Q=red, NSW = green, V=blue, and T=red}.
- When we try to assign SA, we see that every value violates a constraint.
- Since T was the last assigned, we back up and try to assign a new color for T and then proceed to fail in exactly the same way.
- A more intelligent approach would be to backtrack to a variable that was responsible for making one of the possible values of SA impossible.
- To do this, we can keep track of a set of assignments that are in conflict with some value for SA.
- The set {Q=red, NSW = green, V=blue} is called the **conflict set** for SA.
- The **backjumping** method backtracks to the most recent assignment in the conflict set (so wouldn't consider Tasmania).