Solving CSPs

CS156

Chris Pollett

Sep 29, 2014

Outline

- CSP Variants
- Quiz
- Inferring solutions
- Kinds of consistency checks

Introduction

- Last week, we had started talking about constraint satisfaction problems, (CSPs).
- A CSP consisted of a set of variables, X, a set of domains, D for each of the variables in X, and a set of constraints C where each constraint involved a relation $R\left(X_{i_1}, X_{i_m}\right)$ we would like to hold of our variables. .
- A solution to a CSP was an assignment to each variable X_i in X a value $v_i \in D_i$ such that each constraint is satisfied by this assignment.
- We showed how map coloring, job-shop scheduling, and the 8-queens problems could be formulated as CSPs.
- At the end of last day we began looking at some variants of CSPs, such as allowing the domains of the variables to be infinite, or allowing the domains to be continuous, or allowing the domains to involve linear or nonlinear equalities or inequalities as constraints.
- Before we talk about algorithms to solve CSPs today, we finish looking at a couple more CSP variants.

More Variations on the CSP Formalism

- One way to restrict the kind of CSP we are working with is to place a bound on the number of variables that can occur in a constraint.
- For example, we could restrict ourselves to **unary** constraints, constraints whose relation only take one input, such as the constraint $\langle (SA), SA \neq green \rangle$.
- One step up from this is to allow **binary constraints**, those that take two variables as inputs such as $SA \neq NSW$.
- One can of course keep going allowing so-called **higher-order** constraints such as $BETWEEN(X, Y, Z)$ which might express that the value of Y is between that of X and Z (as integers).
- Just as in programming languages where you can have functions which take a variable number of arguments, we can imagine constraints that take an arbitrary number of arguments.

```
#in python
def print_some_args(*args):
   for arg in args:
       print arg
```

  Such constraints are called **global constraints**. For example, $Alldiff$ is a constraint among two or more arguments saying that all of the arguments are different.
- One can specify the constraint in Sudoku as a bunch of $Alldiff$ constraints.

Preference Constraints

- Recall when we were looking at local search, we also discussed optimization problems -- those problems where we were trying to optimize some objective function.
- One can also imagine having in addition to **absolute constraints**, such as professor can't teach two classes at the same time, **preference constraints**, which indicate preferred values, such as Prof. X prefers not to teach mornings.
- Such preferences can be encoded as costs on individual variable assignments.
- Problems which optimize such cost functions are called **constraint optimization problems**, (COPs).

Quiz

Which of the following is true?

1. k local beam search is another name for simulated annealing.
2. For games like tic-tac-toe it is impossible to come up with a search tree.
3. Opening and ending playbooks for games behave a little like a bidirectional search.

Inference in CSPs

- We now consider algorithms for solving CSPs.
- Like most regular state-space algorithms we have considered so far, CSP algorithms often involve search.
- Unlike these algorithms, CSP algorithms can also involve **inferences** known as **constraint propagation**.
- Constraint propagation involves using the constraints to reduce the number of legal values for a variable, which can in turn reduce the number of legal values for another variable, and so on.
- Constraint propagation may be intertwined with search or it may be done as a pre-processing step before the search starts. Sometimes constraint propagation actually suffices to solve the CSP.
- In coming up with an algorithm to solve a CSP, one might imagine having a partial assignment or set of partial assignments so far. The idea is we want to come up with new partial assignments which set values for more of the variables.
- A solution to the CSP must be a complete (assign all var's), consistent (satisfy all constraints) assignment. So we want to keep the partial assignments we have at least consistent.
- One way to do this is to use the notion of **local consistency**: We treat each variable and its current domain as a node in a graph (in binary rel case, hypergraph if higher-order), and each binary constraint as an arc, and for each node/arc we enforce "consistency notion".
- We next look at different kinds of "consistency notions".

Node Consistency

- We say a single variable is **node consistent** if all the values in the variables in its current domain satisfy the variable's unary constraints. So if we had the constraint $SA \neq green$ we could make SA node consisting by restricting its domain from $\{red, green, blue\}$ to just $\{red, blue\}$.
- We say that a network (a graph) is **node consistent** if every variable in the network is node consistent.
- Node consistency can be done as a pre-processing step, and in so doing we eliminate unary constraints.

Arc Consistency

- One can actually show that any n-ary constraint R in variables X_1, ..., X_n with domains D_1, ...D_n can be written in terms of binary constraints, so it suffices to consider now the only other important case, the binary case. The basic idea is to introduce a new variable Z whose domain is the Cartesian product $D_1 \times ... \times D_n$. We then use constraints $R_i(X_i, Z)$ which express that X_i is the i component of a tuple Z which is in R. So if have $R_1(X_1, Z)$, ..., $R_n(X_n, Z)$ all with the same Z we have forced which tuple in R we are talking about.
- We say a variable in a CSP is **arc-consistent** if every value in its current domain satisfies the variable's binary constraints.
- More formally, X_i is arc consistent with respect to the variable X_j if for every value in the current domain D_i there is some value in the domain D_j that satisfies the binary constraint on the arc (X_i, X_j)
- We say a network is arc-consistent if every variable is arc consistent with respect to every other variable.
- For example, suppose the domain of both X and Y are the single digit natural numbers and we have the constraint $Y = X^2$.
- This might be written explicitly as:
  $\langle (X, Y), \{(0, 0), (1, 1), (2, 4), (3, 9)\}\rangle$
- To make X arc-consistent with Y we reduce its domain to $\{0, 1, 2, 3\}$.
- To make Y arc-consistent with X we reduce its domain to $\{0, 1, 4, 9\}$.

AC-3 Algorithm

- AC-3 is a popular algorithm for arc-consistency.
- Its name derives from the fact that it was the third algorithm considered for arc-consistency in Mackworth 1977.

```
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 arc, 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
                # since X_i's domains change might affect X_k
    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 tl
            delete x from D_i
            revised := true
    return revised
```

- Suppose a CSP has n variables, each with domain size at most d and with c binary constraints.
- Each arc can be inserted (X_k, X_i) in the queue at most d times because X_i has at most d variables to delete.
- Checking the consistency of an arc can be done in $O\left(d^2\right)$ (time to check each pair in domains).
- So the whole algorithm has $O\left(cd^3\right)$ total worst-case time.

Path Consistency

- Arc consistency can help reduce the domain of variables, sometimes finding a solution.
- For some problems though, it won't tell us if there is a solution or not.
- For example, consider the problem of coloring the map of Australia that we talked about on Wednesday.
- If we had two colors $\{red, blue\}$ for each variable, it would be arc-consistent even though the graph is not 2-colorable. (WA, NT, SA all touch each other)
- To further reduce the domain spaces, we need a stronger consistency notion.
- We say a two-variable set $\{X_i, X_j\}$ is **path-consistent** with respect to a third variable X_m if, for every assignment $\{X_i = a, X_j = b\}$ consistent with the constraints $\{X_i, X_j\}$, there is an assignment to X_m that satisfies the constraints on $\{X_i, X_m\}$ and $\{X_m, X_j\}$. Here X_m can be though of as along a path from X_i to X_j.
- Suppose we want to make $\{WA, SA\}$ path consistent with respect to NT.
- We start by enumerating the consistent assignments to the set: $\{WA = red, SA = blue\}$ and $\{WA = blue, SA = red\}$. We can see that with both of these assignment NT can be neither red nor $blue$.
- So we eliminate both assignments, and the graph can't be two colored.
- Mackworth also had a PC-2 algorithm for path consistency.

k-consistency

- You can come up with even stronger forms of constraint propagation.
- A CSP is **k**-consistent if, for any set of $k-1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable.
- 1-consistency corresponds with node consistency.
- 2-consistency corresponds with arc consistency.
- We say a CSP is **strongly k-consistent**, if it is j-consistent for all $j \leq k$.
- Suppose we have a CSP with n nodes and make it strongly n-consistent. We can then solve the problem as follows: First, we choose a value consistent for X_1. (can do since 1 consistent). Since the problem is 2-consistent we can find a value for X_2 which is consistent with X_1, and so on.
- For each variable X_i, we need only search through the d values in the domain to find a value consistent with $X_1, ...X_{i-1}$. So the whole process takes $O\left(n^2 d\right)$ time.
- Unfortunately, one can show that any algorithm for establishing n-consistency takes exponential time and space in the worst case.

Global Constraints

- What kind of consistency check can we do with global constraints?
- One simple inconsistency check we can do with $Alldiff$ works as follows: if m variables are involved in the constraint, and if they have n possible distinct values altogether, and $m > n$, then the constraint cannot be satisfied.
- This leads to the following simple algorithm: First remove any variable in the constraint that has a singleton domain, and delete that variable's value from the domain of the remaining variables. Repeat as long as there are singleton variables. If one ever see the empty domain an inconsistency has been found.