https://www.cs.ubc.ca/~jordon/teaching/cpsc322/2019w2/

Lecture 10: CSPs - Introduction

January 28th, 2020
https://www.cs.ubc.ca/~jordon/teaching/cpsc322/2019w2/lectures/lecture10.pdf

Variables/Features, domains and Possible Worlds

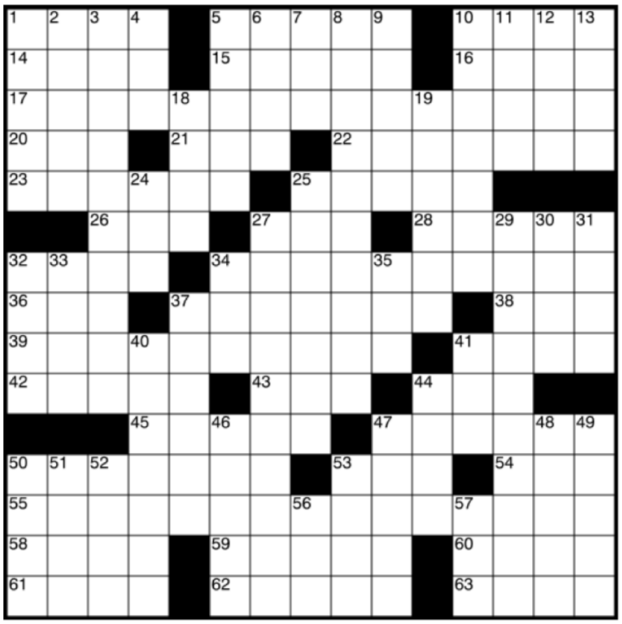
Variables/Features:

- We denote variables using capital letters
- Each variable V has a domain of possible values, denoted $dom(V)$.
- Variables can be of several main kinds:
  - Boolean: $|dom(V)|= 2$
  - Finite: the domain contains a finite number of values
  - Infinite but discrete: the domain is countably infinite
  - Continuous: e.g.,real numbers between 0 and 1

A possible world is a complete assignment of values from domains to variables.

Examples:

→ Crossword puzzle:



  - variables are words that have to be filled in (63 in this case)
  - domains are valid English words of required length
  - possible worlds: all ways of assigning words

→ Crossword 2:

  - variables are cells (individual squares) in the 15x15 grid
  - domains are letters of the alphabet
  - possible worlds: all ways of assigning letters to cells

→ Sudoku

  - variables are empty cells
  - domains are numbers between 1 and 9
  - possible worlds: all ways of assigning numbers to cells

→ n-Queens problem

  - variable: location of a queen on a chess board
    * there are n of them in total, hence the name
  - domains: grid coordinates (n^2)
  - possible worlds: locations of all queens

→ Scheduling problem:
In the scheduling problem we have the following definition of the sets in question:

  - The variables are the different tasks that need to be scheduled (e.g., course in a university; job in a machine shop)
  - The domains are the different combinations of times and locations for each task (e.g., time/room for course; time/machine for job)
  - The possible worlds are a time/location assignment to each task

Constraints

Constraints are restrictions on the values that one or more variables can take:

Def: A unary constraint is a restriction involving only one variable.

Def: A k-ary constraint is a restriction involving the domains of k different variables (However all k-ary constraints can be represented by binary constraints)

Constraints can be specified by:

- Giving a function that returns true when given values for each variable satisfy the constraint
- Giving a list of valid domain values for each variable participating in the constraint

Def: A possible world satisfies a set of constraints if the set of variables involved in each constraint take values that are consistent with that constraint.

Constraints in the above examples:

- Crossword Puzzle:
  Constraints: words have the same letters at points where they intersect
- Crossword 2:
  Constraints: sequences of letters form valid English words
- Sudoku:
  constraints: rows, columns, boxes contain all different numbers
- n-Queens problem
  constraints: no queen can attack another
- Scheduling Problem
  constraints:
  - tasks can't be scheduled in the same location at the same time
  - certain tasks can be scheduled only in certain locations
  - some tasks must come earlier than others; etc.

CSPs

Def: (Constraint Satisfaction Problem)
A constraint satisfaction problem consists of

- a set of variables
- a domain for each variable
- a set of constraints

Def: (model / solution)
A model of a CSP is an assignment of values to variables (i.e. a **possible world**) that satisfies all of the constraints.

Variants

We may want to solve the following problems using a CSP:

A) determine whether or not a model exists

B) find a model

C) find all of the models

D) count the number of the models

E) find the best model given some model quality

- this is now an optimization problem

F) determine whether some properties of the variables hold in all models

To summarize:

- Need to think of search beyond simple goal driven planning agent.
- We started exploring the first AI Representation and Reasoning framework: **CSPs**

Lecture 11: CSPs - Search and Arc Consistency

January 28th & 30th, 2020
https://www.cs.ubc.ca/~jordon/teaching/cpsc322/2019w2/lectures/lecture11.pdf

Generate-and-Test algorithm

This algorithm focuses on generating all of the possible worlds at a single time, and test them to see if any violate any constraints. The pseudocode is as follows:

```
for a in domA
   for b in domB
      for c in domC
         if (abc) satisfies all constraints:
            return (abc)
return NULL
```

CSPs as Search Problems

We define CSPs the following way:

- **states**: assignments of values to a subset of the variables
- **start** state: the empty assignment (no variables assigned values)
- **neighbours** of a state: nodes in which values are assigned to one additional variable
- **goal state**: a state which assigns a value to each variable, and satisfies all of the constraints

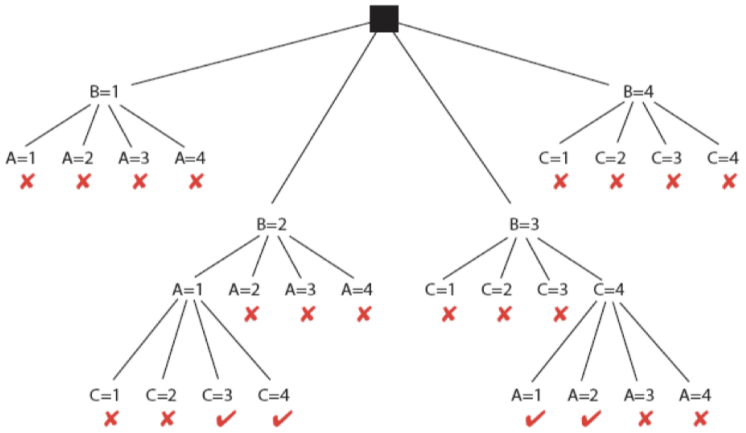
For a CSP problem, some search strategies will work better than others. It is important to notice that for a problem with n variables all of the solutions are at depth n, thus there is *NO* role for a heuristic function.

Also, since the search space is finite, and without cycles, DFS is the best strategy for finding a solution.

We can avoid exploring some sub-trees i.e. prune the DFS Search tree, by implementing the following strategy:

- once we consider a path whose end node violates one or more constraints, we know that **a solution cannot exist below that point**
- thus we should **remove that path** rather than continuing to search

The followign is an example, where the variables set is $\{A, B, C\}$, the domains for each variable is: $\{1, 2, 3, 4\}$ and the constraints are: $\{A < B, B < C\}$:



At this point it is important to note that the algorithm's efficiency will depend on the order in which the nodes are expanded.

Consequently we ask whether it is possible to do better than search.

Consistency

Key Ideas: prune the domains as much as possible before "searching" for a solution.

Def: A variable is domain consistent if no value of its domain is ruled impossible by any unary constraints.

We therefore need a way to deal with constraints that involve more than one variable. For this we define constraint networks.

Constraint Networks

Def: (Constraint Network)
A constraint network is defined by a graph, with:

- one **node** for every **variable**
- one **node** for every **constraint**

and undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint

Arc Consistency

Def: (Arc Consistency)
An arc $\langle X, r(X,Y)\rangle$ is arc consistent if for each value x in $dom(X)$ there is some value y in $dom(Y)$ such that $r(x,y)$ is satisfied.

How can we enforce Arc Consistency?

If an arc $\langle X, r(X,Y)\rangle$ is not arc consistent, all values x in $dom(X)$ for which there is no corresponding value in $dom(Y)$ may be deleted from $dom(X)$ to make the arc $\langle X, r(X,Y)\rangle$ consistent. This removal will never rule out any models/solutions.

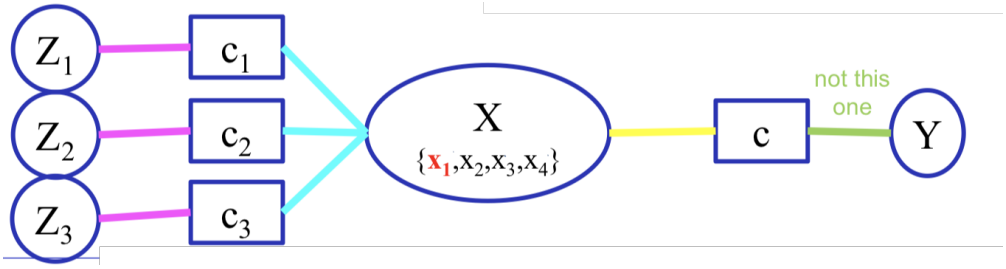
A network is arc consistent if all arcs are consistent.

Lecture 12: CSPs - Arc Consistency & Domain Splitting

January 30th, 2020
https://www.cs.ubc.ca/~jordon/teaching/cpsc322/2019w2/lectures/lecture12.pdf

Arc Consistency Algorithm: high level strategy

The idea is to consider the arcs in turn, making each arc consistent. However some may be revisited. Notable, when we reduce the domain of a variable X to make an arc $\langle X, c\rangle$ arc consistent, we add every arc $\langle Z, c'\rangle$ where c' involves Z and X:



It is not necessary to add other arcs $\langle X, c'\rangle$ where $c \neq c'$, since if an arc $\langle X, c'\rangle$ was consistent before, it will still be arc consistent (in the "for all" we'll just check fewer values)

Arc Consistency Pseudocode

```
TDA <- all arcs in constraint network
while TDA is not empty:
   select arc a from TDA
   if a is not consistent:
      make a consistent
      add arcs to TDA that may now be inconsistent
```

A higher level version of the algorithm is shown below:

```
Procedure GAC(V,dom,C)
   Inputs:
      V: a set of variables
      dom: a function such that dom(X) is the domain of variable X
      C: set of constraints to be satisfied
   Output:
      arc-consistent domains for each variable
   Local
      D_X is a set of values for each variable X
      TDA is a set of arcs
   for each variable X do:
      D_X <- dom(X)
      TDA <- {<X, c> | X in V, c in C and X in scope(c)}
   while TDA is not empty:
      select <X, c> in TDA
      TDA <- TDA.remove(<X, c>)
      ND_X <- {x | x in D_X and there exists y in D_Y s.t. (x,y) satisfies c}
      if (ND_X != D_X) then:
         TDA <- TDA + {<Z, c'> | X in scope(c') and c' != c \
               and Z in scope(c') - {X}}
         D_X <- ND_X
   return {D_X | X is a variable}
```

Arc Consistency Algorithm Complexity

Let's determine Worst-case complexity of this procedure:

- Let the max size of a variable domain be d
- Let the number of variables be n
- Let all constraints be **binary**

In this case we have that:

- The maximum number of binary constraints is: $\frac{n(n-1)}{2}$
- The number times the same arc can be inserted in back into the ToDoArc list is d
- The number of steps involved in checking the consistency of an arc is: d^2

Consequently, the time complexity of the Arc Consistency algorithm is $O(n^2d^3)$

Interpreting Solutions

There are three possible outcomes to the algorithm, and they are interpreted as follows:

- One domain is empty → there is no solution
- Each domain has a unique value → there is a unique solution
- Some domains have more than one value → zero or more solutions:
  - in this case, arc consistency isn't enough to solve the problem: we still need to perform *search*

Domain Splitting

When arc consistency ends some domains have more than one value, which means there may or may not be a solution... We have two options:

A) Apply Depth-First Search with Pruning

B) Split the problem in a number of (eg. two) disjoint cases

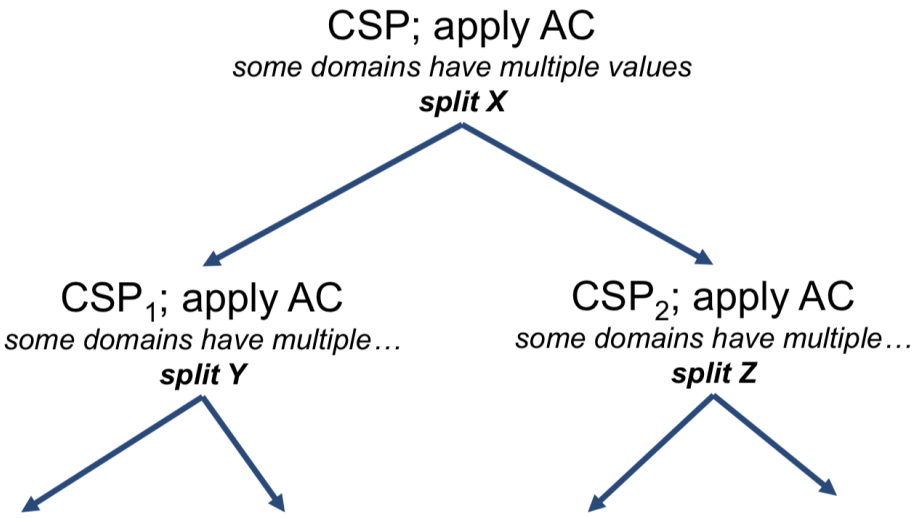
By reducing dom(X) we may be able to run AC again, since we simplify the problem using **arc consistency**:

```
If there is no unique solution (dom(X)>1 for at least one variable):
   Split X
   For all the splits:
      Restart arc consistency on arcs <Z, r(Z,X)>
```

These are the arcs that are possibly inconsistent.

The disadvantage of this is that we are required to keep all of the CSPs around rather than smaller simpler states of DFS.

Searching by Domain splitting



More formally: Arc consistency with domain splitting as another formulation of CSP as search:

```
Start state: run AC on vector of original domains (dom(V_1), ..., dom(V_n))

States: "remaining" domains (D(V_1), ..., D(V_n)) for the vars with D(V_i)\
      in dom(V_i) for each V_i

Successor function: split one of the domains + run arc consistency

Goal state: vector of unary domains that satisfies all constraints

Solution: any goal state
```