Features And Constraints

Arnav Gupta

October 18, 2024

Contents

1 Constraint Satisfaction Problems

A CSP has:

- a set of variables
- a domain for each variable

- a set of constraints or evaluation function

Two kinds of problems:

1. **Satisfiability** Problems: find an assignment that satisfies constraints (hard constraints)
2. **Optimization** Problems: find an assignment that optimizes the evaluation function (soft constraints)

A solution to a CSP is an assignment to the variables that satisfies all constraints (model of constraints).

CSPs can be represented as graph searching problems in two ways:

- **complete** assignment
  - nodes: assignment of value to all variables
  - neighbours: change one variable value
- **partial** assignment
  - nodes: assignment to first $k-1$ variables
  - neighbours: assignment to variable k

Search spaces can get large, so branching factors can be big. Still, path to goal is not important, only the goal is. No predefined starting node.

CSPs can be solved with:

- generate and test
- backtracking
- consistency
- hill-climbing
- randomized including local search

1.1 Representations

Primal representations consider unary and binary constraints. **Dual** representations consider N-ary constraints.

1.2 CSP Solutions

Generate and Test: go through all combos and check each one

1.2.1 Backtracking

Prune large portions of the state space:

1. order all variables
2. evaluate constraints in the order as soon as they are grounded

Efficiency depends on the order of variables, which can be hard to optimize. Best to push failures as high as possible, since this cuts large branches of the tree early.

1.2.2 Consistency

Look for inconsistencies through a graphical representation. Specifically, give each domain a value until all constraints are satisfied.

1. Consistency Networks

   Domain constraint: a unary constraint on values in a domain, written $\langle X, c(X) \rangle$.

   A node in a consistency networks (CNs) is **domain consistent** if no domain value violates any domain constraint. A CN is domain consistent if all nodes are domain consistent.

   Arc $\langle X, c(X, Y) \rangle$ is a **constraint** on X. An arc $\langle X, c(X, Y) \rangle$ is **arc consistent** if $\forall X \in D_X$, there is some $Y \in D_Y$ such that $c(X, Y)$ is satisfied. A CN is arc consistent if all arcs are arc consistent.

   A set of variables $\{X_1, X_2, X_3, \ldots, X_N\}$ is **path consistent** if all arcs and domains are consistent.

2 AC-3

Makes a CN arc consistent and domain consistent with a **To-Do Arcs** (TDA) queue that has all inconsistent arcs.

Algorithm:

1. make all domains domain consistent

2. put all arcs $\langle Z, c(Z, \_)\rangle$ in TDA
3. repeat the following until TDA is empty
   (a) select and remove an arc $\langle X, c(X, Y)\rangle$ from TDA
   (b) remove all values of the domain of X that don't have a value in the domain of Y that satisfies the constraint $c(X, Y)$
   (c) if any were removed, add all arcs $\langle Z, c'(Z, X)\rangle$ to TDA $\forall Z \neq Y$

AC-3 is guaranteed to terminate with one of:

- every domain is empty: no solution
- every domain has a single value: solution
- some domain has more than one value: split into two, run AC-3 recursively on each half

AC-3 has time complexity $O(cd^3)$ where c is the number of binary constraints and d is the maximum size of each domain.

3 Variable Elimination

Eliminate the variables one by one, passing their constraints to their neighbours. When a single variable remains, if it has no variables, the CN is **inconsistent**. Variables are eliminated according to some elimination ordering.

Algorithm:

- if there is only 1 variable, return the intersection of the unary constraints that contain it
- select a variable X
  - join the constraints in which X appears, forming constraint R
  - project R onto its variables other than X: call this R_2
  - place R_2 between all variables that were connected to X
  - remove X
  - recursively solve the simplified problem
  - return R joined with the recursive solution

4 Local Search

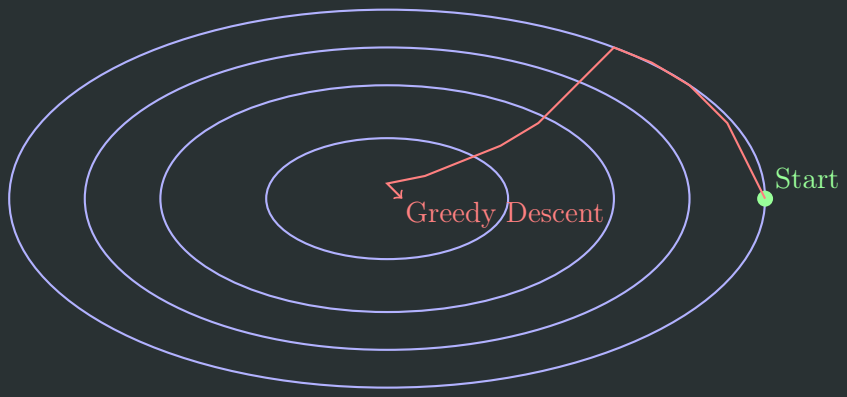
Steps:

- maintain an assignment to each variable
- at each step, select a neighbour of the current assignment
- terminate when a satisfying assignment is found, or return the best assignment found

The aim is to find an assignment with no unsatisfied constraints (no conflicts). So, the heuristic to be minimized is the number of conflicts.

4.1 Greedy Descent

Goal is to find the variable-value pair that minimizes the number of conflicts at each step.



Variants:

- select variable in most conflicts and value that minimizes number of conflicts
- select variable in any conflict and value that minimizes number of conflicts
- select variable at random and value that minimizes number of conflicts
- select variable and value at random

Problems can arise with finding:

- local minima rather than global minima

- plateaus where heuristics are uninformative
- ridges: local minima where looking ahead some number of steps may help

Randomized Greedy Descent allows for random steps (not just downward) and random restart (reassign random values to variables).

In high dimensions, search spaces can have flat canyons that are hard to optimize.

4.2 Stochastic Local Search

Mix of greedy descent (move to lowest neighbour), random walk (take random steps), and random restart.

4.2.1 Simulated Annealing

Steps:

- pick a variable and new value at random
- if it improves, adopt the new value
- otherwise, adopt it probabilistically depending on temperature T
  - with current assignment n and proposed assignment n', move to n' with probability $e^{-(h(n')-h(n))/T}$
- temperature can be reduced

High temperature $\rightarrow$ higher probability of accepting a worse change.

4.2.2 Tabu List

To prevent cycling, maintain a **tabu list** of k last assignments and do not allow assignment already on tabu list.

Can be implemented more efficiently than a list of complete assignments, but still expensive.

4.2.3 Parallel Search

Total assignment called an **individual**:

- maintain a population of k individuals instead of one

- at each stage, update each individual
- whenever an individual is a solution, it can be reported
- similar to using k restarts but uses k times the minimum number of steps

4.2.4 Beam Search

Like parallel search, but choose the k best out of all neighbours.

1. Stochastic Beam Search Probabilistically choose the k individuals at the next generation, using the heuristic: $e^{-h(n)/T}$.

   Maintains diversity and heuristic reflects fitness.

4.2.5 Genetic Algorithms

Pairs of individuals combine to create offspring:

- randomly choose pairs of individuals where the fittest are most likely to be chosen
- for each pair, perform a **cross-over**: form offspring taking parts of the parents
- mutate some values

4.3 Comparing Stochastic Algorithms

Runtime Distribution: plot runtime and proportion of runs that are solved in that runtime