# Constraint Satisfaction and Iterative Improvement Search

**Manuela Veloso**
**CMU 15-381, Fall 2000**

See Chapter 4 of Textbook - 4.4.

- CSP definition, examples
- Search improvements
- Hill climbing, simulated annealing, genetic evolution

*These lecture slides should be complemented with the examples discussed and worked out on the blackboard during the lecture.*

## CSP - Definition

Constraint Satisfaction Problem

- A triplet:
- A set of **variables**, $V$.
- A domain of **values**, $D$.
- A set of **constraints**, $C$.
- **Problem:** Assign values to variables without violating the constraints.
- Constraints are usually represented by a function. In simple cases, they can be represented as a set of all the valid assignments to sets of variables.

## Examples

- N-queens
- Map coloring
- Natural language generation
- Assignment problem constrained on rules
- Scheduling

## Job-Schop Scheduling

- Production!
- Several operations to be performed on parts
  - ▷ Polish, drill-hole, paint, weld, cut
- Several resources available
  - ▷ Drilling, polishing, etc machines
- Constraints!
  - ▷ Drill before Polish before Paint
  - ▷ Job1 before Job2
  - ▷ Release dates, due dates, duration of tasks

## Search Algorithms

- Exhaustive forward search
- Forward search with constraint checking and backtracking
- DPLL algorithm

## Unit Propagation

- If all the literals **but one** in a clause have a **false** assignment, then what is "the constraint" about the value that that one unassigned literal may take?
- False $\vee$ False $\vee \ldots \vee A$

## Unit Propagation Algorithm

A constraint satisfaction algorithm

```
Propagate(C,KB)
If C is a uni-clause,
 i.e. all of C's literals are assigned False,
 but one,  and this remaining literal L
 is not assigned, then

 - Assign True to L
 - For each clause C' in KB containing ``not L''

     - Propagate (C', KB - C')
```

## Unit Propagation Examples

**C1:** $A \vee \neg B$

**C2:** $B \vee \neg C$

**C3:** $C \vee \neg A$

**C4:** $A$

**C4':** $\neg B$

## DPLL Procedure

Davis, Putman, Logmann, Loveland 1962

DPLL(KB,A)
Input: A CNF knowledge base KB,
a truth assignment A to propositions in KB
Output: A truth assignment, if KB is satisfiable,
false otherwise.

0. Apply A to KB
1. Propagate(KB,KB)
2. If a clause is violated, return False.
3. If all propositions are in A, return A.
4. Let Q be an unassigned proposition in KB.
5. Return DPLL(KB, A ∪ [Q=False]) or
6. DPLL(KB, A ∪ [Q=True])

## Search Improvements

- Variable Ordering Heuristics
  - ▷ Most constrained variable
  - ▷ Most constraining variable
- Value Ordering Heuristics
  - ▷ Least-constrained value: value that causes the smallest reduction in available values for "neighboring" variables.
- For equally ranked choices, choose **randomly**.
- Extend DPLL with heuristics.
- Generate *schedule* and **refine** it.
- Use a **similar** schedule and **adapt** it.

## Multiple Solutions - Optimization

- CSP addresses "only" the assignment problem: each variable gets one value and the constraints are not violated.
- **Best** assignment?
- Many large scale problems
  - ▷ Travelling salesperson problem - shortest route
  - ▷ Real job shop scheduling - shortest time, least number of resources, minimum cost
  - ▷ HST scheduling - maximize number of observations within time limits
  - ▷ VLSI layout, routing - keep connectivity, reduce space
- Searching ALL assignments is intractable

## Iterative Improvement

- Find "best" solution - not interested in "path" to the solution

1. Start with random configuration
2. Consider several, if not all, **next** configurations
3. Accept some and reject some
4. Restart if no more configurations

## Hill Climbing

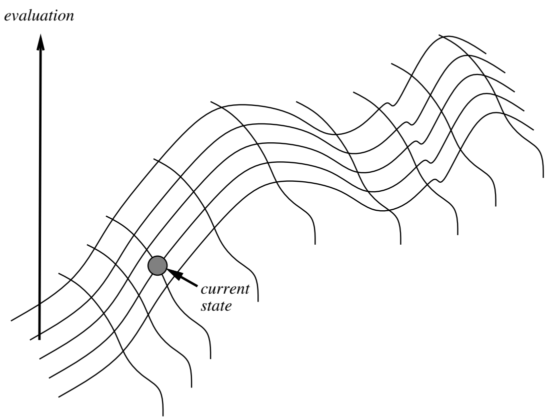
$\boxed{\text{Only } \textbf{move} \text{ to } \textbf{better} \text{ configuration}}$

1. current ← initial configuration
2. loop
3. next ← a highest-valued successor of current
4. if value[next] < value[current] then return current
5. current ← next
6. end

## Hill Climbing Issues

- Easy to implement
- No backtracking, no memory
- Evaluation function
- Number of successors - large? small?
- **Local maximum**
- **Plateaux** - evaluation function is essentially flat
- **Ridges** - steep sloping sides; top of ridge easily reached, but top slopes very gently toward a peak.

## Hill Climbing Issues




## Handling the Evaluation Function

- Randomized hill climbing
  - ▷ Choose and evaluate random move
  - ▷ Start from a large set of random initial points, do plain hill-climbing, and save best of all the searches
- Simmulated annealing
  - ▷ **Explore:** Move to worse successor “some times.”

## Simulated Annealing

- Escape from local maximum - not extreme random restart.
- Similar to hill-climbing, but:
  - ▷ pick random move, if better node than current then fine.
  - ▷ pick if worse move with some probability.
- Two factors to control *exploration*:
  - ▷ divergence - "how worse is the new state,"
  - ▷ coverage - "how long has the search pursued."

## Simulated Annealing

- Annealing - The process of cooling a material until freezes.
- Value: total energy of the atoms in the material.
- Temperature and schedule: rate at which temperature decreases.
- Probability of jumping to bad move decreases with temperature.

## Simulated Annealing (cont.)

*current* is the initial state.
*schedule* is a mapping from search time to temperature.
for each search time step $t$
  $T = schedule[t]$
  if $T = 0$ RETURN *current*.
  *next* is a random successor of *current*.
  $\Delta E$ = VALUE[*next*] $-$ VALUE[*current*]
  if $\Delta E > 0$ then *current* is *next*.
   else *current* is *next* with probability $e^{\Delta E/T}$.

- As T tends to 0, what does simulated annealing become?

## Search by Evolution

- Based on genetic evolution.
- Search for the optimum is based on simulated evolution.
- Survival of the fittest and natural selection – Darwin, 1859, *On the Origin of Species by Means of Natural Selection.*
- Genetic algorithms: Transform a population of individuals into a new population.
- Conventional operators:
  - ▷ Reproduction
  - ▷ Crossover
  - ▷ Mutation

## How to “Use” the Population?

- How does the GA operations allow the GA to effectively search the search space?
- Testing a “few” random points in the space.
- Acquires estimate of the average fitness of the complete search space.
- How to pursue the search?
  - ▷ Blindly pick new random points.
    - Problem? no use of past experience; non-adaptive; plain random search; find a needle in a haystack; only a small number of points will be tested.

## How to “Use” the Population?

  - ▷ Another approach: Greedily *exploit* the best result seen so far and do not *explore* any other points.
    - Problem? overlook possible better points
- So now we want to still visit some of the points not visited, but give some preference to the best-seen point so far.

## Exploration and Exploitation

- Exploration: Testing a random point now has a “cost.” A new random point has in average the current average fitness estimate. Then the expected loss associated with random exploration is the difference between the best-seen fitness and the average fitness estimate.
- Exploitation: Not testing a new random point has also a cost.

## The Conventional Genetic Algorithm

- $F$ – Fitness function
- $F_t$ – termination fitness
- $G$ – number of generations
- $M$ – size of the population
- $p_r$ – frequency of reproduction
- $p_c$ – frequency of crossover
- $p_m$ – frequency of mutation

## The Conventional Genetic Algorithm

```
GA (population, F, G)
 population <-- get random population
 calculate fitness of each individual
 calculate average fitness of population
 repeat
  parents <-- REPRODUCTION (population,F,pr,M)
  population <-- CROSSOVER (parents,pc)
  population <-- MUTATION (population,pm)
  calculate fitness of each individual
  calculate average fitness of population
 until F(best-individual) >= termination fitness
 return best-individual
```

## Summary

- Constraint satisfaction problems
- Several CSP search algorithms
  - ▷ Exhaustive forward search
  - ▷ Forward search with early backtracking
  - ▷ Constraint propagation
  - ▷ DPLL
  - ▷ Variable and value ordering heuristics
- Optimization - iterative search
  - ▷ Hill climbing
  - ▷ Simulated annealing
  - ▷ Genetic evolution