Constraint Satisfaction

Manuela Veloso

15-381 - Fall 2001

CSP definition, examples

Search improvements

Satisfiable knowledge base

Forward search

DPLL algorithm

Hill climbing, simulated annealing, genetic evolution

CSP - Definition

Constraint Satisfaction Problem

- 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

Knowledge in Propositional Logic

- Conjunctive normal form - CNF - a conjunction of **clauses**
  - Examples:
    $KB = (A \vee B \vee C) \wedge \neg C$
    $KB = (A \vee \neg B) \wedge (\neg C \vee D) \wedge A$

- Clause: disjunction of literals
  - $(A \vee \neg B)$

- Literal: proposition or its negation
  - $\neg A, D$

Propositional Satisfiability

- Is the knowledge **satisfiable**?
  - Is there a *truth assignment* to each individual proposition that makes true the knowledge base, i.e., the conjunction of *all* the clauses?
  - If there is such solution, then that's a **model**.

SATISFIABILITY - Find such truth assignment!

Forward Search with Backtrack

- Choose one unassigned proposition at a time.
- Create two search branches: true and false assignments.
- Backtrack as soon as a clause is violated.

C1: $A \vee \neg B$

C2: $B \vee \neg C$

C3: $C \vee \neg A$

Improving Brute-Force Search

- How? Ideas?

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 $\cup$ (Q=False)) or
6. DPLL(KB, A $\cup$ (Q=True))

DPLL Example

C1: $A \vee \neg B$

C2: $B \vee \neg C$

C3: $C \vee \neg A$

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.

Other Search Improvements

- Generate a complete assignment – *schedule* – and **refine** it!

- Find 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. But what about the **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 - 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

Only **move** to **better** configuration

1. current $\leftarrow$ initial configuration
2. loop
3. next $\leftarrow$ a highest-valued successor of current
4. if value(next) $<$ value(current) then return current
5. current $\leftarrow$ 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.

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

- Simulated 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

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*.
  ΔE = 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

- Search for the optimum is based on simulated genetic evolution.
- Survival of the fittest – 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 *explore* 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.

- Another approach: Greedily *exploit* the best result seen so far and do not *explore* any other points.
  - Problem? overlook possible better points

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
  - Unit propagation; DPLL
  - Variable and value ordering heuristics
- Optimization - iterative search
  - Hill climbing
  - Simulated annealing
  - Genetic evolution