Lecture 9: Search 8

Victor R. Lesser
CMPSCI 683
Fall 2010

ANNOUNCEMENTS

- REMEMBER LECTURE ON TUESDAY!
- EXAM ON OCTOBER 18
  - OPEN BOOK
  - ALL MATERIAL COVERED IN LECTURES
  - REQUIRED READINGS
- WILL MOST PROBABLY NOT COVER MATERIAL ON PLANNING

Today's Lecture

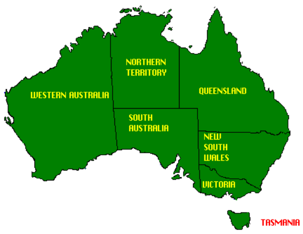
- Another Form of Local Search
  - Repair/Debugging in Constraint Satisfaction Problems
    - GSAT
- A Systematic Approach to Constraint Satisfaction Problems
  - Simple Backtracking Search

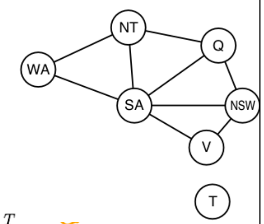
Constraint Satisfaction Problems (CSP)

- A set of **variables** $X_1 \ldots X_n$, and a set of **constraints** $C_1 \ldots C_m$. Each variable X_i has a **domain** D_i of possible **values**.
- A **solution** to a CSP: a complete assignment to all variables that satisfies all the constraints.
- Representation of constraints as predicates.
- Visualizing a CSP as a **constraint graph**.

Example: Map coloring

Constraint graph: nodes are variables, arcs show constraints





Variables WA, NT, Q, NSW, V, SA, T

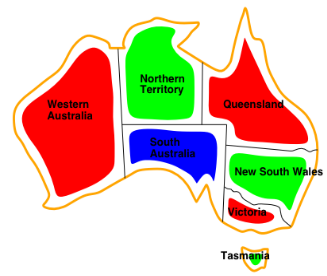
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

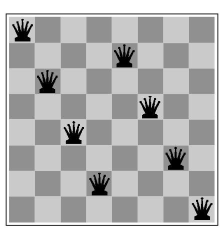
A Valid Map Assignment



Solutions are assignments satisfying all constraints, e.g.,

$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Example 3: N queens

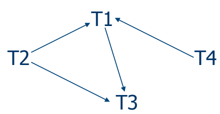


• What are the variables? domains? constraints?

8 queens

- 8 variables X_i, i = 1 to 8; for each column
- Domain for each variable {1,2,…,8}
- Constraints are:
  - $X_i \neq X_j$ for all j = 1 to 8, j≠I; not on same row
  - $|X_i - X_j| \neq |i - j|$ for all j = 1 to 8, j≠I; not on diagonal
  - Note that all constraints involve 2 variables
- Generate-and-test with no redundancies requires "only" N^N combinations…

Task scheduling



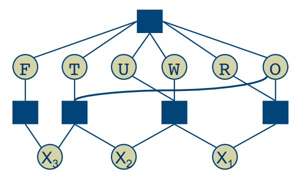
T1 must be done during T3
T2 must be achieved before T1 starts
T2 must overlap with T3
T4 must start after T1 is complete

- What are the variables? domains? constraints?

Non-Binary Constraints

T W O
+ T W O
F O U R

- $O + O = R + 10 \cdot X1$
- $X_1 + W + W = U + 10 \cdot X_2$
- $X_2 + T + T = O + 10 \cdot X_3$
- $X_3 = F$
- *alldiff*(F,T,U,W,R,O)
- Between0–9(F,T,U,W,R,O)
- Between0–1 (X_1,X_2,X_3)



3 or more variables constraints

Constraint optimization

- Representing *preferences* versus absolute constraints.
  - Weighted by constraints violated/satisfied
- Constraint optimization is generally more complicated.
- Can also be solved using local search techniques.
- Hard to find optimal solutions.

Local search for CSPs: Heuristic Repair

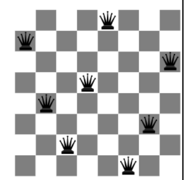
- Start state is some assignment of values to variables that may violate some constraints.
  - Create a complete but inconsistent assignment
- Successor state: change value of one variable.
- Use **heuristic repair** methods to reduce the number of conflicts (iterative improvement).
  - **The** min-conflicts heuristic**: choose a value for a variable that minimizes the number of remaining conflicts.**
  - **Hill climbing on the number of violated constraints**
- Repair constraint violations until a consistent assignment is achieved.
- Can solve the *million*-queens problem in an average of 50 steps!

Heuristic Repair Algorithm

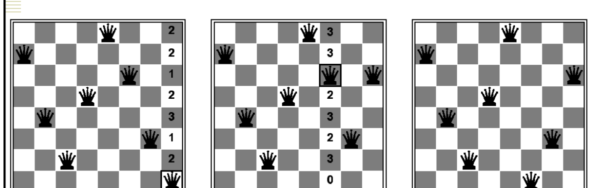
```
function MIN-CONFLICTS(csp, max-steps) returns a solution or failure
   inputs: csp, a constraint satisfaction problem
           max-steps, the number of steps allowed before giving up
   local variables: current, a complete assignment
                    var, a variable
                    value, a value for a variable

   current ← an initial complete assignment for csp
   for i = 1 to max-steps do
      var ← a randomly chosen, conflicted variable from VARIABLES[csp]
      value ← the value v for var that minimizes CONFLICTS(var, v, current, csp)
      set var=value in current
      if current is a solution for csp then return current
   end
   return failure
```


N-Queens Heuristic Repair

- Pre-processing phase to generate initial assignment
  - Greedy algorithm that iterates through rows placing each queen on the column where it conflicts with the fewest previously placed queens
- Repair phase
  - Select (randomly) a queen in a specific row that is in conflict and moves it to the column (within the same row) where it conflicts with the fewest other queens



Example of min-conflicts: N-Queens Problem



A two-step solution of an 8-queens problem. The number of remaining conflicts for each new position of the selected queen is shown. Algorithm moves the queen to the min-conflict square, breaking ties randomly.

SAT- Satisfiability Problem

Given a propositional sentence, determine if it is satisfiable, and if it is, show which propositions have to be true to make the sentence true. 3SAT is the problem of finding a satisfying truth assignment for a sentence in a special format

Why are we interested in this representational framework?

Definition of 3SAT

- A **literal** is a proposition symbol or its negation (e.g., *P* or ¬ *P).*
- A **clause** is a disjunction of literals; a 3-clause is a disjunction of exactly 3 literals (e.g., $P \vee Q \vee \neg R$).
- A sentence in CNF or **conjunctive normal form** is a conjunction of clauses; a 3-CNF sentence is a conjunction of 3-clauses.
- For example,

$(P \vee Q \vee \neg S) \wedge (\neg P \vee Q \vee R) \wedge (\neg P \vee \neg R \vee \neg S) \wedge (P \vee \neg S \vee T)$

Is a 3-CNF sentence with four clauses and five proposition symbols.

Mapping 3-Queens into 3SAT

At least 1 has a Q — not exactly 2 have Q's — not all 3 have Q's

$(Q_{1,1} \vee Q_{1,2} \vee Q_{1,3})$ $\wedge$ $(Q_{1,1} \vee \neg Q_{1,2} \vee \neg Q_{1,3})$

$\wedge$ $(\neg Q_{1,1} \vee Q_{1,2} \vee \neg Q_{1,3})$

$\wedge$ $(\neg Q_{1,1} \vee \neg Q_{1,2} \vee Q_{1,3})$ $\wedge$ $(\neg Q_{1,1} \vee \neg Q_{1,2} \vee \neg Q_{1,3})$

Do the same for each row, the same for each column, the same for each diagonal, and'ing them all together.

$\wedge$

$(Q_{2,1} \vee Q_{2,2} \vee Q_{2,3})$ $\wedge$ $(Q_{2,1} \vee \neg Q_{2,2} \vee \neg Q_{2,3})$

$\wedge$ $(\neg Q_{2,1} \vee Q_{2,2} \vee \neg Q_{2,3})$ $\wedge$ $(\neg Q_{2,1} \vee \neg Q_{2,2} \vee Q_{2,3})$ $\wedge$ $(\neg Q_{2,1} \vee \neg Q_{2,2} \vee \neg Q_{2,3})$

$\wedge$

$(Q_{1,1} \vee Q_{2,2} \vee Q_{3,3})$ $\wedge$ $(Q_{1,1} \vee \neg Q_{2,2} \vee \neg Q_{3,3})$ $\wedge$ $(\neg Q_{1,1} \vee Q_{2,2} \vee \neg Q_{3,3})$

$\wedge$ $(\neg Q_{1,1} \vee \neg Q_{2,2} \vee Q_{3,3})$ $\wedge$ $(\neg Q_{1,1} \vee \neg Q_{2,2} \vee \neg Q_{3,3})$

⋮

etc.

Converting N-SAT into 3-SAT

$A \vee B \vee C \vee D$

$\equiv$

$(A \vee B \vee E) \wedge (\sim E \vee C \vee D)$

A = T	A = F	A = F
B = F	B = T	B = F
C = F	C = F	C = T
D = F	D = F	D = F
E = F	E = F	E = T

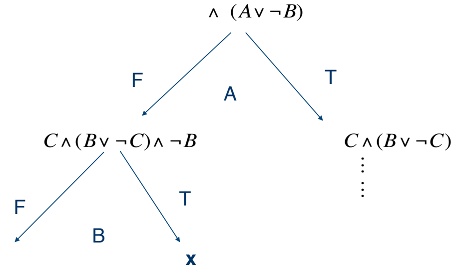
Add in dummy variable E, not interested in its truth value from problem perspective nor does its truth affect satisfiability of original proposition

……

2 - SAT polynomial time but can' t map all problem into 2 - SAT

Davis-Putnam Algorithm (Depth-First Search)

$(A \vee C) \wedge (\neg A \vee C) \wedge (B \vee \neg C)$
$\wedge\ (A \vee \neg B)$



GSAT Algorithm

Problem: Given a formula of the propositional calculus, find an interpretation of the variables under which the formula comes out true, or report that none exists.

procedure GSAT

Input: a set of clauses $\propto$, MAX-FLIPS, and MAX-TRIES

Output: a satisfying truth assignments of $\propto$, if found

```
begin
    for i := 1 to MAX-TRIES ; random restart mechanism
        T := a randomly generated truth assignment
        for j := 1 to MAX-FLIPS
            if T satisfies ∝ then return T
            p :=    a propositional variable such that a change in its truth assignment gives the largest increase in
    total number of clauses of ∝ that are satisfied by T.
            T := T with the truth assignment of p reversed
        end for
    end for
    return "no satisfying assignment found"
end
```


GSAT Performance

formulas		GSAT			DP		
vars	clauses	M-FLIPS	tries	time	choices	depth	time
50	215	250	6.4	0.4s	77	11	1.4s
70	301	350	11.4	0.9s	42	15	15s
100	430	500	42.5	6s	84×10^3	19	2.8m
120	516	600	81.6	14s	0.5×10^6	22	18m
140	602	700	52.6	14s	2.2×10^6	27	4.7h
150	645	1500	100.5	45s	—	—	—
200	860	2000	248.5	2.8m	—	—	—
250	1062	2500	268.6	4.1m	—	—	—
300	1275	6000	231.8	12m	—	—	—
400	1700	8000	440.9	34m	—	—	—
500	2150	10000	995.8	1.6h	—	—	—

GSAT versus Davis-Putnam

(a backtracking style algorithm)

Domain: *n*-queens

formulas			GSAT		
Queens	vars	clauses	flips	tries	time
8	64	736	105	2	0.1s
20	400	12560	319	2	0.9s
30	900	43240	549	1	2.5s
50	2500	203400	1329	1	17s
100	10000	1.6×10^6	5076	1	195s

Domain: ***hard*** **random 3CNF formulas, all satisfiable (hard means chosen from a region in which about 50% of problems are unsolvable)**

GSAT Performance *(cont'd)*

- *GSAT Biased Random Walk*
- With probability *p,* follow the standard GSAT scheme,
  - *i.e.*, make the best possible flip.
- With probability 1 - *p*, pick a variable occurring in some unsatisfied clause and flip its truth assignment. (Note: a possible uphill move.)
- GSAT-Walk < Simulated-Annealing < GSAT-Noise < GSAT-Basic

formula		GSAT basic		GSAT walk		GSAT noise		Simul. Ann.	
vars	clauses	time	flips	time	flips	time	flips	time	flips
100	430	.4	7554	.2	2385	.6	9975	.6	4748
200	860	22	284693	4	27654	47	396534	21	106643
400	1700	122	2.6×10^6	7	59744	95	892048	75	552433
600	2550	1471	30×10^6	35	241651	929	7.8×10^6	427	2.7×10^6
800	3400	*	*	286	1.8×10^6	*	*	*	*
1000	4250	*	*	1095	5.8×10^6	*	*	*	*
2000	8480	*	*	3255	23×10^6	*	*	*	*

Comparing noise strategies on hard random 3CNF formulas. (Time in seconds on an SGI Challenge)

3SAT Phase Transition

- Easy -- Sastifiable problems where many solutions
- **Hard -- Sastifiable problems where few solutions**
- Easy -- Few Satisfiable problems

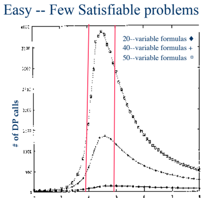


Fig. 1 Solving 3SAT problems.

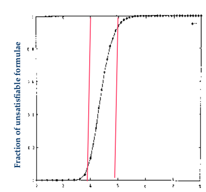


Fig. 2 Fraction of unsatisfiable 3SAT problems.

- Assumes concurrent search in the satisfiable space and the non-satisfiable space (negation of proposition)

A Simplistic Approach to Solving CSPs using Systematic Search

- **Initial state**: the empty assignment
- **Successor function**: a value can be assigned to any variable as long as no constraint is violated.
- **Goal test**: the current assignment is complete.
- **Path cost**: a constant cost for every step. – not relevant

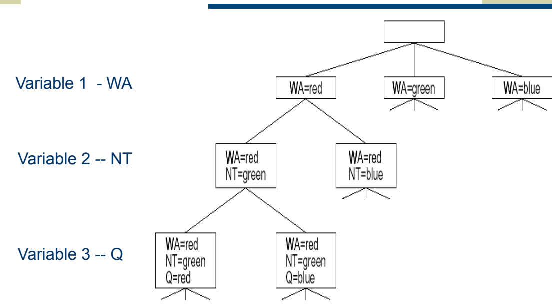
What more is needed?

- Not just a successor function and goal test
- But also a means to propagate the constraints imposed by variables already bound along the path on the potential fringe nodes of that path and an early failure test
- Thus, need explicit representation of constraints and constraint manipulation algorithms

Exploiting Commutativity

- Naïve application of search to CSPs:
  - If use breath first search
  - Branching factor is $n \cdot d$ at the top level, then $(n-1)d$, and so on for n levels (n variables, and d values for each variable).
  - The tree has $n! \cdot d^n$ leaves, even though there are only d^n possible complete assignments!
- Naïve formulation ignores **commutativity** of all CSPs: the order of any given set of actions has no effect on the outcome.
  - [WA=red, NT=green] same as [NT=green, WA=red]
- Solution: **consider a single variable at each depth of the tree.**

Part of the map-coloring search tree



Next Lecture

- Informed-Backtracking Using Min-Conflicts Heuristic
  - Arc Consistency for Pre-processing
  - Intelligent backtracking
  - Reducing the Search by structuring the CSP as a tree search

- Extending the model of simple heuristic search
  - Interacting subproblem perspective