Foundations of Artificial Intelligence

CS472/3

Lecture #8

Bart Selman

Today's Lecture

Local search.

Readings: R&N, Section 4.4.

A Different Approach: Local Search

So far, we have considered methods that systematically explore the full search space, possibly using **principled** pruning (CSP / A* etc.).

The current best such algorithms (IDA* / SMA*) can handle search spaces of up to 10^{100} states / around 500 binary valued variables.

(These are "ballpark " figures only!)

Slide CS472–3

What if we have 10,000 or 100,000 variables / search spaces of up to $10^{30,000}$ states?

A completely different kind of method is called for:

Local Search Methods or
Iterative Improvement Methods

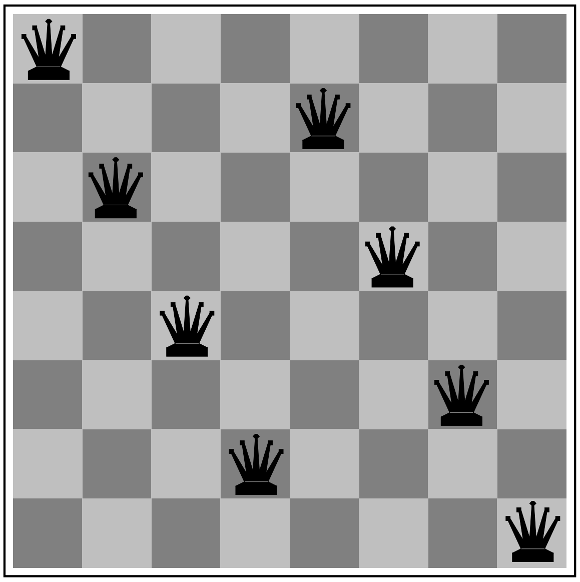
Slide CS472–4

Local Search Methods

Applicable when we're interested in the Goal State —
not in how to get there.

E.g. N-Queens, VLSI layout, or map coloring.

Slide CS472–5



Slide CS472–6

Local Search Methods

Approach quite general: any NP-complete problem / CSP.

Other examples: planning, scheduling, TSP, graph coloring, and time-tabling.

In fact, many (most?) large Operations Research problems are solved using local search. E.g. Delta airlines. (near-optimal!)

Q. What's the "competitor" in OR?

Slide CS472–7

Key idea (suprisingly simply):

```
a) Select (random) initial state
     (initial guess at solution)

b) Make local modification to try
        to improve current state.

c) repeat b) till goal state found
        (or out of time).
```

Slide CS472–8

Example

Graph coloring.

a) start with random coloring of nodes

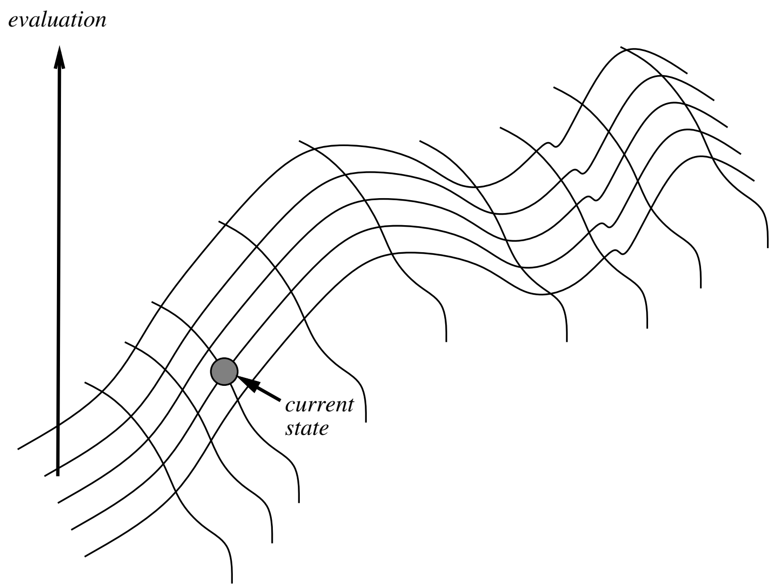
b) change the coloring of one of the nodes to reduce conflicts

c) repeat b)

Slide CS472–9

```
#1  #2   #3  #4  #5 || # conflicts
 R   G    B   R   B ||   1
```

Slide CS472–10



Local Search / Hillclimbing

```
function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
          next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
      next ← a highest-valued successor of current
      if VALUE[next] < VALUE[current] then return current
      current ← next
  end
```

Notes

"successor" normally called "neighbor"

You search in the neighborhood of current state.

When does Hill-Climbing stop?

Current approaches: Often simply keep going!

Use: time limit.

Simple method: very fast, uses minimal memory, surprisingly effective!

$10,000+$ **variables,** $1,000,000+$ **constraints.**

Example

A wide variety of key CS problems can be translated into a propositional logical formalization

e.g., (A ∨ B ∨ C) ∧ (¬B ∧ C ∨ D) ∧ (A ∨ ¬C ∨ D)

and solved by finding a truth assignment to the propositional variables (A, B, C,...) that makes it true, i.e., a *model*.

If a formula has a model, we say that it is "satisfiable".

Special kind of CSP.

Applications:

planning and scheduling
circuit diagnosis and synthesis
deductive reasoning
software testing
etc.

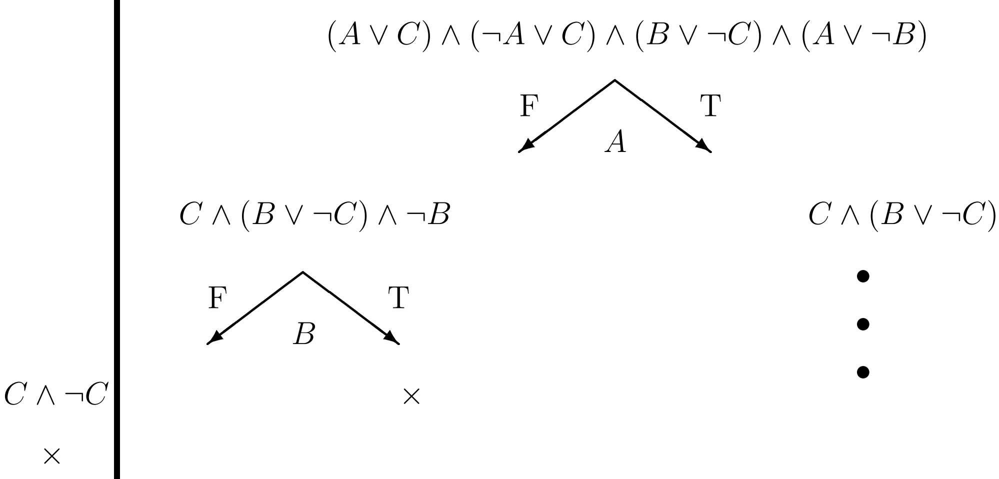
Slide CS472–15

Satisfiability Testing

Best-known method: Davis-Putnam Procedure (1960)

Backtrack search (DFS) through the space of
truth assignments (with unit-propagation).

Slide CS472–16



Slide CS472–17

To date, Davis-Putnam still the *fastest* sound and *complete* method.

However, there are classes of formulas where the procedure scales badly.

Consider an *incomplete* local search procedure.

Slide CS472–18

Greedy Local Search — GSAT

Begin with a random truth assignment (assume CNF).
Flip the value assigned to the variable that
    yields the *greatest number of satisfied clauses*.
    (Note: Flip even if there is no improvement.)
Repeat until a model is found, or have performed
    a specified maximum number of flips.
If a model is still not found, repeat the entire process,
    starting from a different initial random assignment.

A	B	C	$(A \vee C)$	$\wedge$	$(\neg A \vee C)$	$\wedge$	$(B \vee \neg C)$	Score
F	F	F	$\times$		$\surd$		$\surd$	2
F	F	T	$\surd$		$\surd$		$\times$	2
F	T	T	$\surd$		$\surd$		$\surd$	3

(Selman, Levesque, and Mitchell 1992)

How well does it work?

First intuition: It will get **stuck** in local minima,
with a few unsatisfied clauses.

Note we are not interested in **almost** satisfying assignments
E.g., a plan with one "magic" step is useless.
Contrast with optimization problems.

Surprise: It often finds **global** minimum!
I.e., finds satisfying assignments.

form.	GSAT			Davis-Putnam		
vars	m.flips	retries	time	choices	depth	time
50	250	6	0.5 *sec*	77	11	1 *sec*
70	350	11	1 *sec*	42	15	15 *sec*
100	500	42	6 *sec*	10^3	19	3 *min*
120	600	82	14 *sec*	10^5	22	18 *min*
140	700	53	14 *sec*	10^6	27	5 *hrs*
150	1500	100	45 *sec*	—	—	—
200	2000	248	3 *min*	—	—	—
300	6000	232	12 *min*	—	—	—
500	10000	996	2 *hrs*	10^{30}	> 100	10^{19} *yrs*

Search space

Improvements to Basic Local Search

Issue: How to move more quickly to successively lower plateaus?

Avoid getting "stuck" / **local minima**.

Idea: introduce uphill moves ("noise") to escape from long plateaus (or true local minima)

Noise strategies:

a) Simulated Annealing

Kirkpatrick *et al.* 1982; Metropolis *et al. 1953*

b) Mixed Random Walk

Selman and Kautz 1993

Simulated Annealing

•Noise model based on statistical mechanics.

•Pick a random variable

If flip improves assignment: do it.

else flip with probability $p = e^{-\delta/T}$ ("upward").

δ number of additional clauses becoming *unsatisfied*

T = "temperature"

Higher temperature = greater likelihood of upward moves.

Slowly decrease T from high temperature to near zero.

What is p for $T \rightarrow \inf$? For $T \rightarrow 0$? For $\delta = 0$?

Slide CS472–25

Sim. Annealing introduced as analogue to a **physical process**

Way to grow crystals.

Kirkpatrick *et al.* 1982; Metropolis *et al. 1953*

Physical analogy remains elusive.

Can prove that with exponential schedule will converge to global optimum.

Difficult to be more precise about convergence rate.

See recent work on rapidly mixing Markov chains.

Key aspect: **upwards moves / sideways moves.**

Expensive, but if you have time can be best.

(hundreds of papers per year / many applications)

Slide CS472–26

Random Walk

Random walk SAT algorithm:

I Pick random truth assignment.

II Repeat until all clauses satisfied:

Flip variable from any unsatisfied clause.

•Solves 2-SAT (2 variables per clause) in O(n^2) flips.

(Papadimitriou 1992) Why limited value?

•Does not work at all for hard k-SAT (k $\geq$ 3).

Mixing Random Walk with Greedy Local Search

•With probability p, **walk**,

i.e., pick a variable in some unsatisfied clause and flip it;

with probability $(1 - p)$ make a **greedy** flip,

i.e., one that makes greatest decrease in number of unsatisfied clauses.

•Value for parameter p determined empirically,

by finding best setting for a problem class.

Experimental Results: Hard Random 3CNF

	GSAT				Simul. Ann.	
	basic		walk			
vars	time	eff.	time	eff.	time	eff.
100	.4	.12	.2	1.0	.6	.88
200	22	.01	4	.97	21	.86
400	122	.02	7	.95	75	.93
600	1471	.01	35	1.0	427	.3
800	*	*	286	.95	*	*
1000	*	*	1095	.85	*	*
2000	*	*	3255	.95	*	*

Slide CS472–29

- Time in seconds (SGI Challenge).
- Effectiveness: prob. that random initial assignment leads to a solution.
- Complete methods, such as DP, up to 400 variables.
- *Mixed Walk better than*
  Simul. Ann. better than
  Basic GSAT better than
  Backtracking (Davis-Putnam).

Slide CS472–30