Artificial Intelligence I

CS473 — Autumn 2001

Lectures # 9 – 10

Today: Local Search

R&N Sec. 4.4, Sec. 6.4, Exer. 6.15, Sec. 20.8

Next: Knowledge Representation

R&N Ch. 6, 7, selected parts of Ch. 8, 9, 10

Henry Kautz

A Different Approach

So far, we have considered methods that systematically explore the full search space, possibly using principled pruning (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!)

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 CSE 573–3

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 CSE 573–4

Key idea (suprisingly simple)

(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).

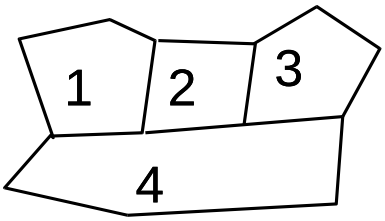
Example

Graph coloring.

(a) start with random coloring of nodes

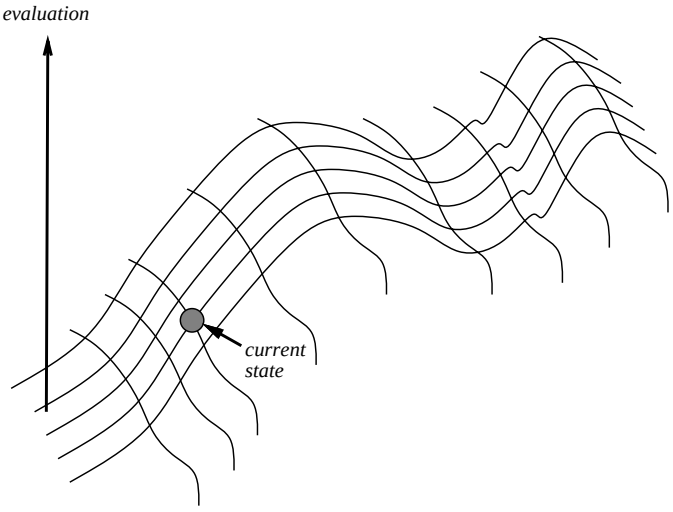
(b) change the coloring of one of the nodes to reduce conflicts

(c) repeat (b)




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

Slide CSE 573–7



Slide CSE 573–8

```
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
```

Slide CSE 573–9

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!

$100,000+$ **variables,** $1,000,000+$ **constraints.**

Slide CSE 573–10

Example

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

e.g., $(A \vee B \vee C) \wedge (\neg B \wedge C \vee D) \wedge (A \vee \neg C \vee D)$

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".

Conjunctive Normal Form (CNF)

Special kind of CSP.

Slide CSE 573–11

Applications:

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

Slide CSE 573–12

Satisfiability Testing

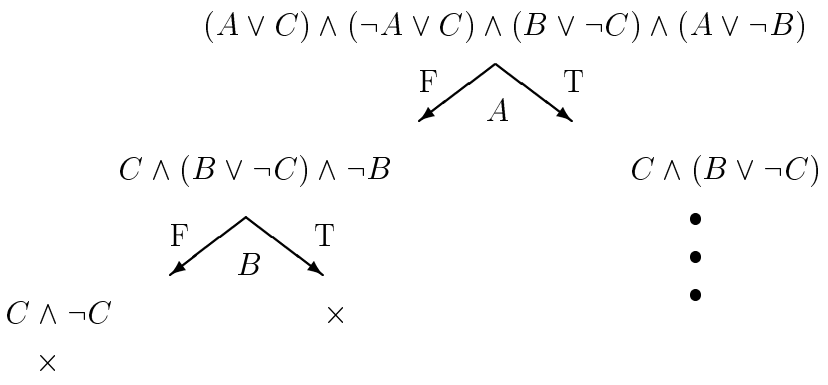
Best-known method: Davis-Putnam Procedure (1960)

Backtrack search (DFS) through the space of
truth assignments

Arc consistency for CNF is called unit-propagation

Q. How would that work?

Slide CSE 573–13



Slide CSE 573–14

To date, Davis-Putnam still the **fastest** sound and
complete method.
Used in **verification, planning...**

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

Consider an *incomplete* local search procedure.

Slide CSE 573–15

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.

Slide CSE 573–16

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)

Slide CSE 573–17

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.

Slide CSE 573–18

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*

Slide CSE 573–19

Search space

Slide CSE 573–20

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

Slide CSE 573–21

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 CSE 573–22

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)

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?**

•Does not work well 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.
  (see also DJP3 2001)

Inside-out: Walksat

1. Pick a **unsatisfied clause** at random.
2. With probability p, flip any variable in the clause;
   with probability $(1-p)$ flip a variable
   in the clause that minimizes number unsat clauses.

- Very efficient to implement!
  Incrementally compute effect of flipping
- Comparable to GSAT+walk, but faster flips.

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 CSE 573–27

- Time in seconds (SGI Challenge).
- Effectiveness: probability that random initial assignment leads to a solution.
- Complete methods, such as DP, up to 650 variables.
- *Mixed Walk (or Walksat) better than*
  Simul. Ann. better than
  Backtracking (Davis-Putnam).

Slide CSE 573–28

Local Minima

1. **local minimum (maximum):** rate of change is positive (negative) in all directions (maze problem — may have to move **away** from goal to find the (best) solution)
2. **plateaus:** all of the local steps look the same

   True local minima are rare in high dimension space.

   Term often relaxed to include plateaus.

Approaches

1. Simulated annealing (done)
2. Mixed-in random walk (done)
3. Random restarts
4. Tabu search
5. Genetic algorithms / programming

1. **Random restarts:** simply restart at a new random state after a pre-defined number of local steps.
2. **Tabu:** prevent returning quickly to same state.
   Implementation: Keep fixed length queue (**"tabu list"**): add most recent step to queue; drop "oldest" step.
   Never make step that's currently on the tabu list.

Slide CSE 573–31

without tabu:

flip var 1, var 2, var 4, var 2, var 10,
var 11, var 1, var 10, var 3 ...

with tabu (length 5): — possible sequence:

flip var 1, var 2, var 4, var 10, var 11,
var 3, var 7, var 2, var 6 ...

Tabu quite powerful; competitive with simulated annealing and random walk (depends on domain).

Slide CSE 573–32

Genetic algorithm

Approach mimics **evolution**.

See Section 20.8 R&N.

Often presented using rich new vocabulary:

fitness function, population, inviduals, "genes", crossover, mutations, etc.

Still, can be viewed quite directly in terms of **standard local search**.

Note: in some sense, natural evolution performs a local search to improve species: "survival of the fittest", "local search in the gene pool"

What is the evaluation function?

What are some of the special aspect of the evolutionary search process?

(Friedberg 1958; Holland 1975; Koza 1992)

- High degree of **parallelism**
- New individuals ("next state / neighboring states")
  derived from "parents"
- (Greedy) selection next generation:
  Survival of the fittest.

General Idea

Maintain a population of strings (states / individuals / candidate solutions)

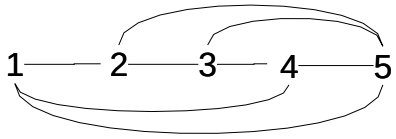
Generate a sequence of generations:

From the current generation, select pairs of strings (based on fitness) to generate new strings, using **crossover**.

Introduce some noise through random **mutations**.

Average and maximum fitness increases over time.

Graph Coloring



	#1	#2	#3	#4	#5	fitness
1	G	B	G	R	B	
2	R	G	G	R	B	
3	R	R	R	G	B	
4	G	G	R	R	B	

generation 1

Slide CSE 573–37

Example Crossover

	"parents"					"children"					
	#1	#2	#3	#4	#5	#1	#2	#3	#4	#5	fitness
1	*G*	*B*	*G*	*R*	*B*	*G*	*B*	*R*	*G*	*B*	
3	*R*	*R*	*R*	*G*	*B*	*R*	*R*	*G*	*R*	*B*	
example mutation:											
	R	*R*	*G*	*R*	*B*	*G*	*R*	*G*	*R*	*B*	

Slide CSE 573–38

Remarks

In practice, several 100 to 1000's of strings.

What if you have only mutations?

What makes this work for graph coloring?

Could this work for SAT testing (GSAT etc.)?

Slide CSE 573–39

Remarks

In practice, several 100 to 1000's of strings.

What if you have only mutations?

Parallel noisy local search

What makes this work for graph coloring?

Locality!

Could this work for SAT testing (GSAT etc.)?

Depends on geometry of "near solutions"

Slide CSE 573–40

Insert genetic-algorithms.ps slides here:
4, 7, 12, 13, 14, 15, 16, 20, 21

Slide CSE 573–41

Perspective

- Jury still out on Genetic Algorithms in general,
  but nature suggests it can be a highly powerful
  mechanism for evolving highly complex systems.

- Schema theory starts to allow analysis,
  but still far from deep understanding
  of formal properties.

Slide CSE 573–42

Local Search — Summary

Surprisingly efficient search method.

- Wide range of applications.
  - any type of optimization / search task
- Formal properties elusive.
  - Why do so many naturally-occuring problems have "near-convex" search spaces?
- Often best method available for **poorly understood** problems — How evolution built us ....