Foundations of Artificial Intelligence

CS472/3 — Spring 1999

Lecture #5

Bart Selman

Today's Lecture

Local Search Methods

(informed search, concl.)

Readings: R&N, Chapter 4.

Informed Methods: Heuristic Search

Informed Methods use problem-specific knowledge.

Heuristic search is an attempt to search the most promising paths first.

Relies on an *evaluation function* — indicates the desirability of expanding a node.

Prime examples

Greedy Search minimize estimated cost to reach the goal, i.e., expand the node "closest" to the goal.

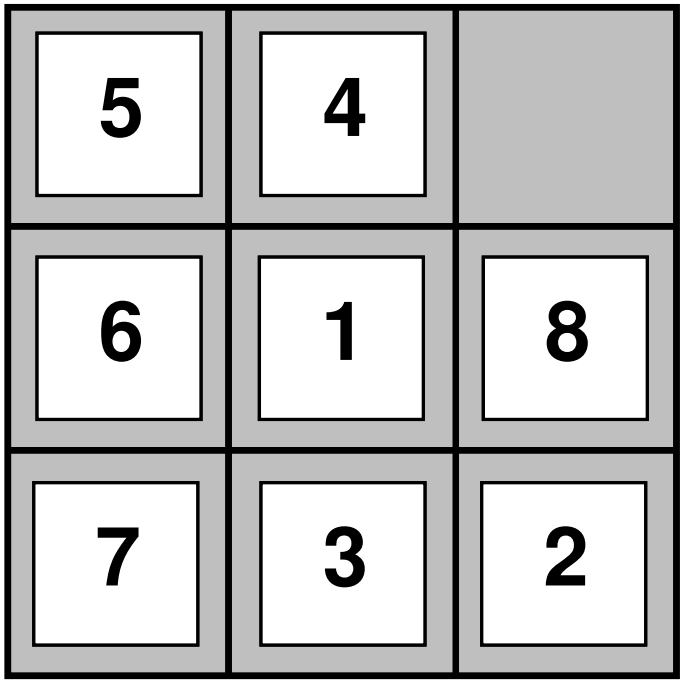
A* minimize total estimated path cost to reach the goal, i.e., expand the node on the "least-cost" solution path to the goal.

Observations A*

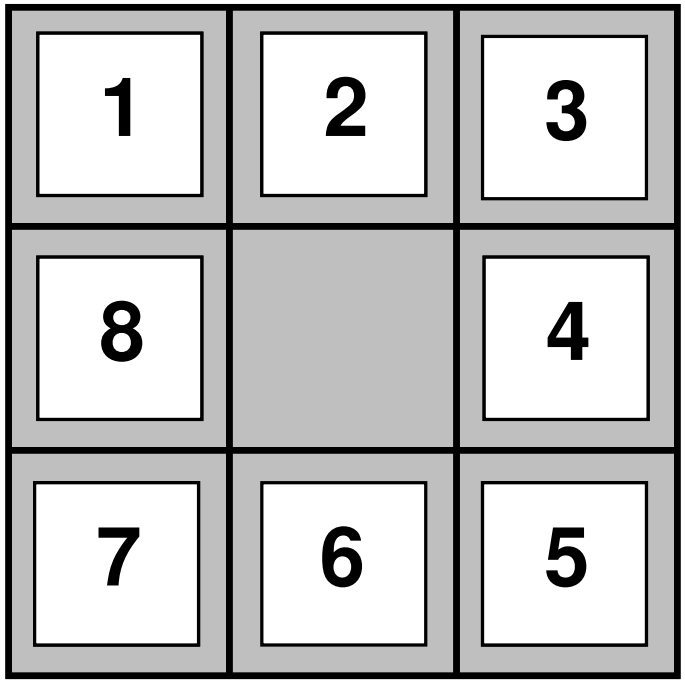
A* is **optimally efficient**: given the information in h,
no other optimal search method can expand fewer nodes.

Non-trivial and quite remarkable!

Heuristic Functions: Example



Start State



Goal State

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Inventing Heuristics

Automatically

A tile can move from sq A to sq B if

A is adjacent to B and B is blank.

(a) A tile can move from sq A to sq B if A is adjacent to B.

(b) A tile can move from sq A to sq B if B is blank.

(c) A tile can move from sq A to sq B.

If all admissible, combine them by taking the *max*.

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?

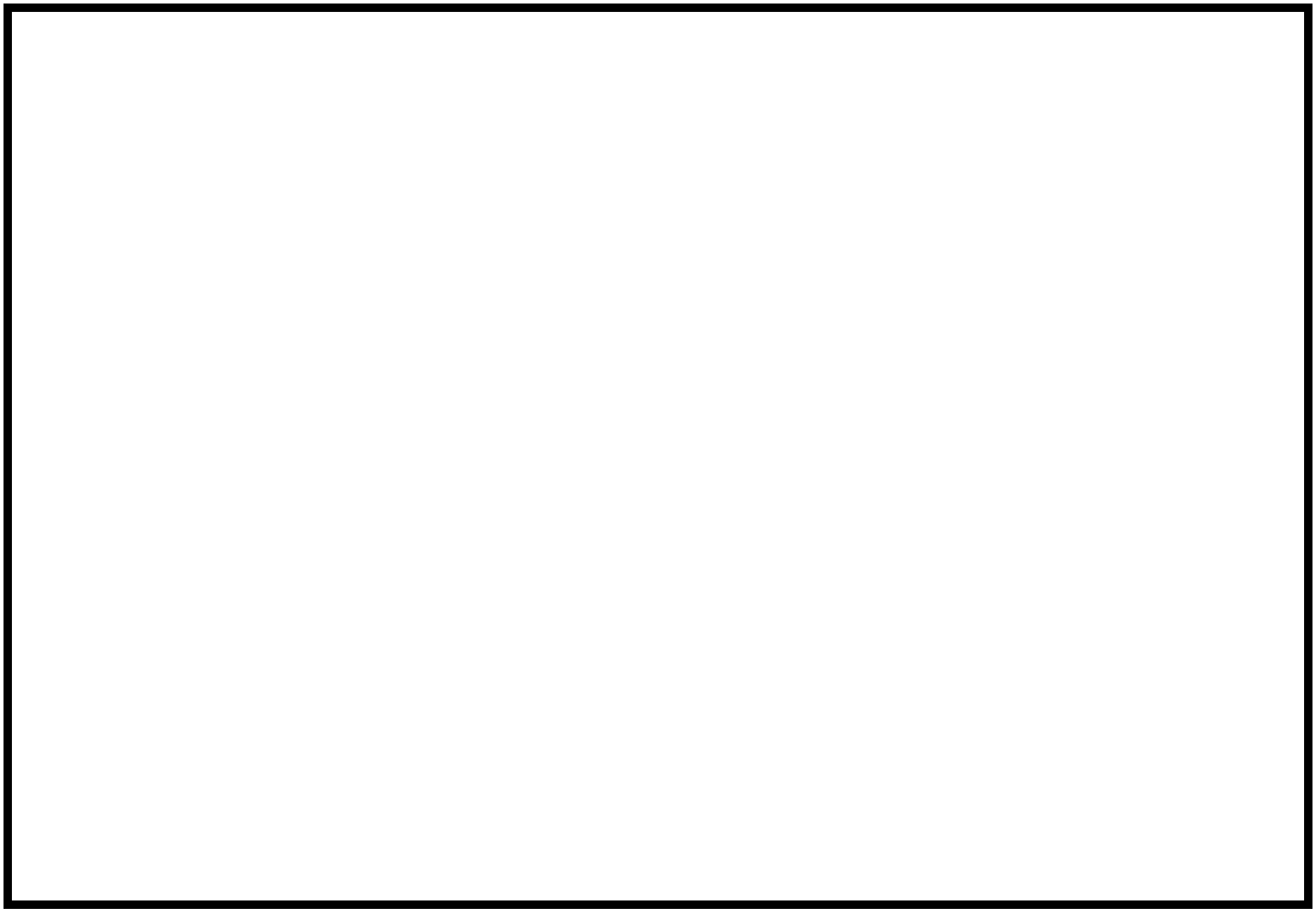
A completely different kind of method is called for:

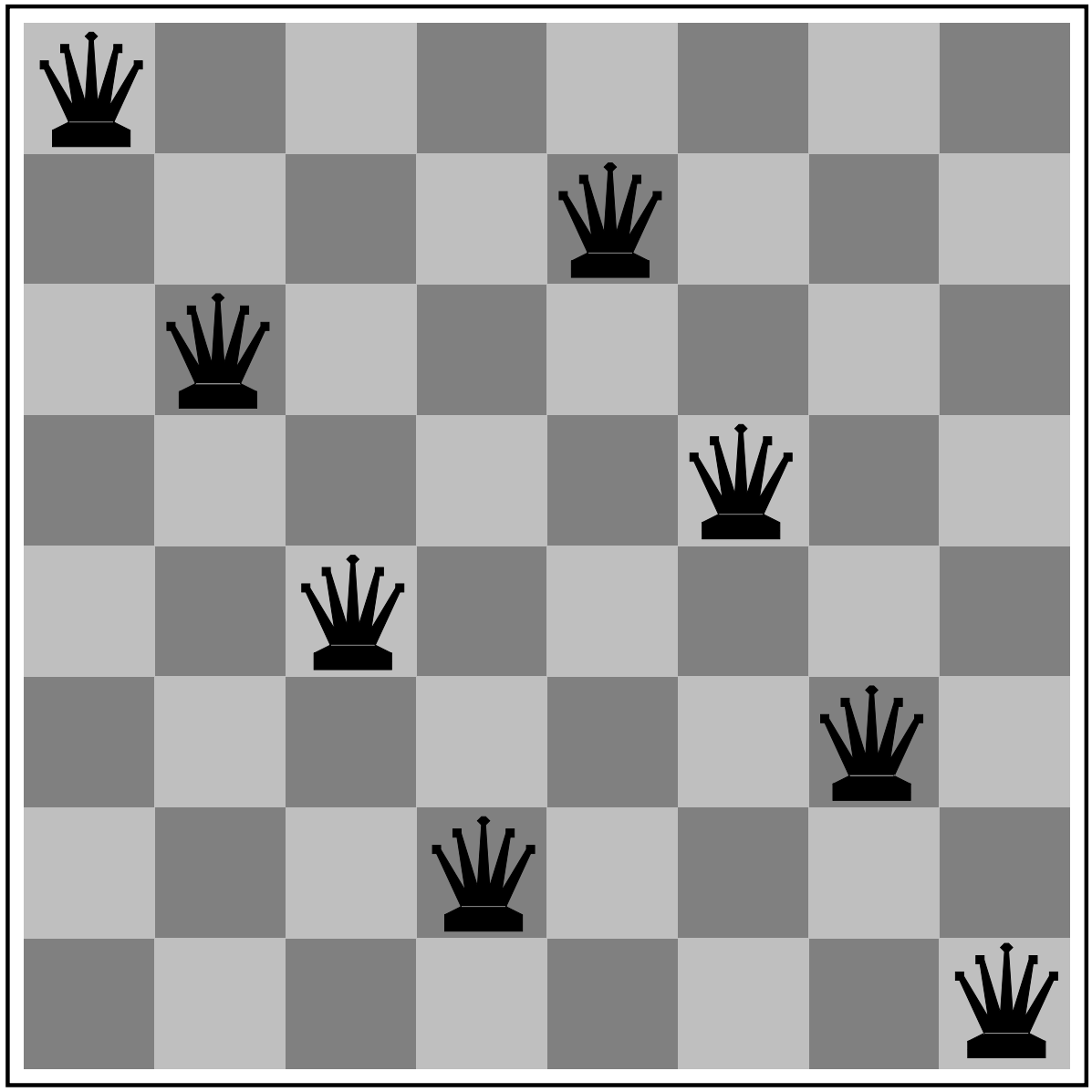
Local Search Methods or
Iterative Improvement Methods

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.





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?

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).

Example

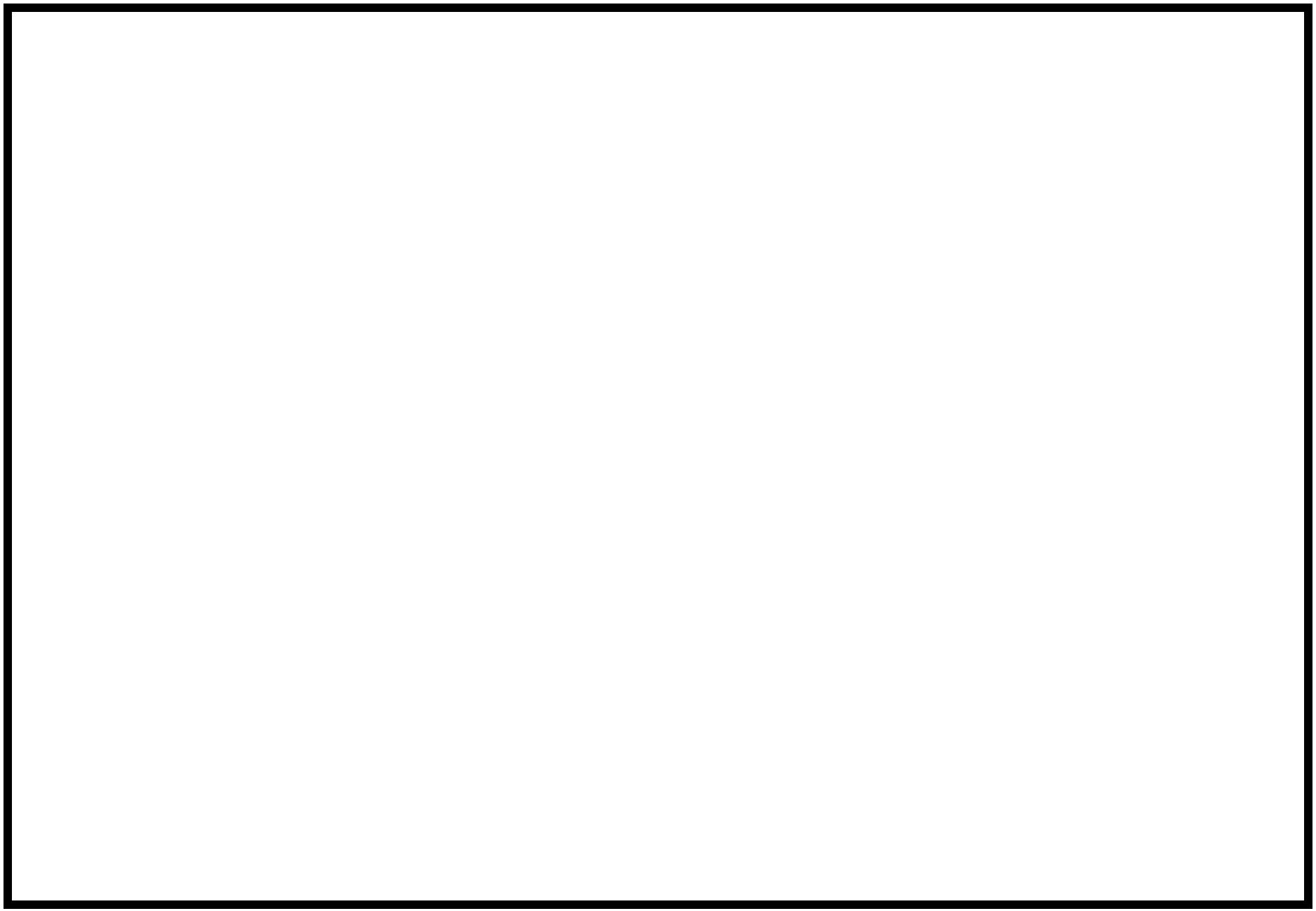
Graph coloring.

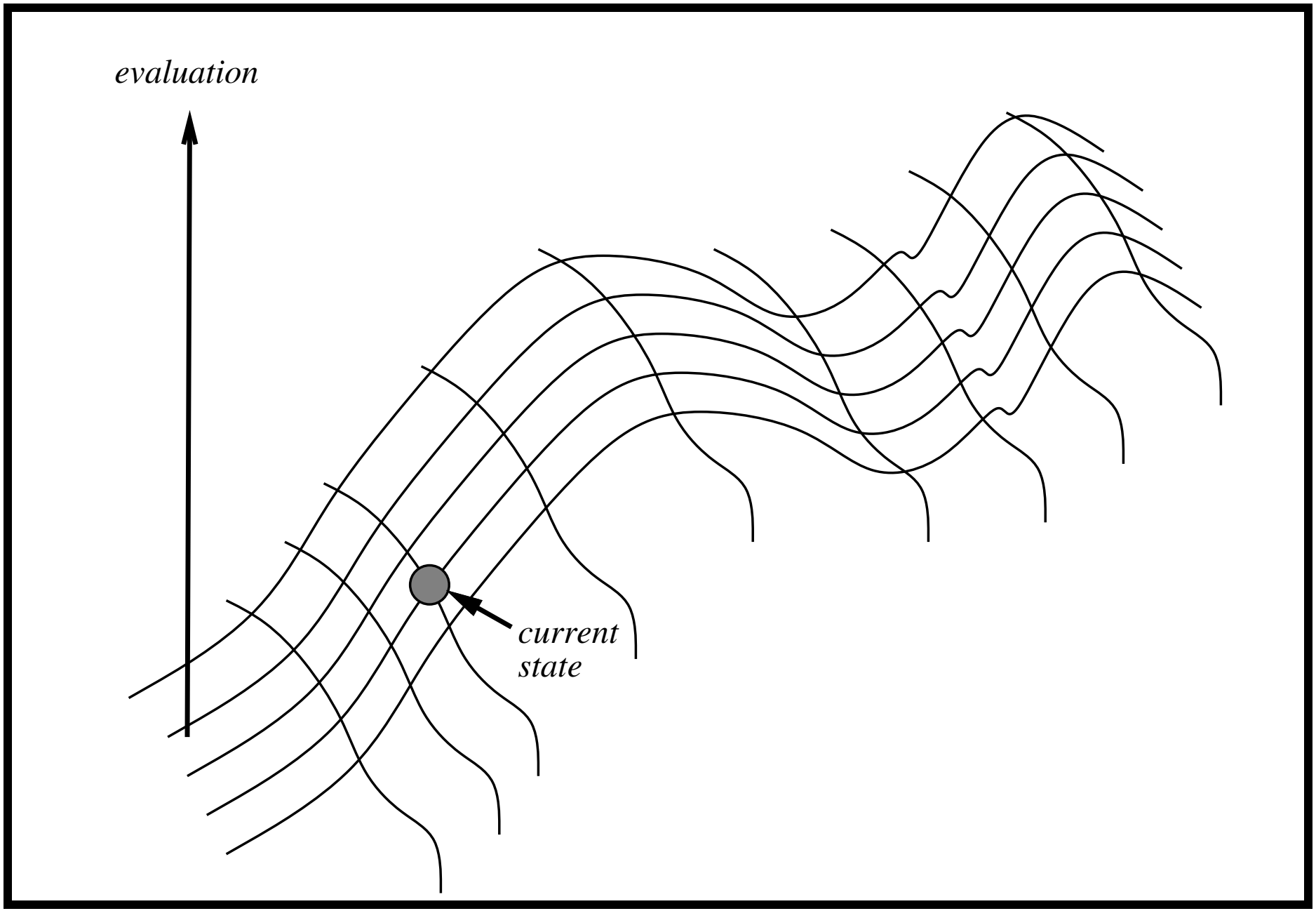
a) start with random coloring of nodes

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

c) repeat b)

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




evaluation
current
state

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

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".

Special kind of CSP.

Applications:

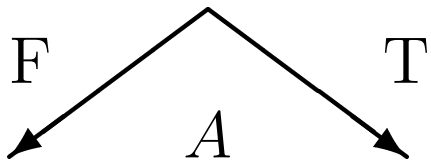
- planning and scheduling
- circuit diagnosis and synthesis
- deductive reasoning
- software testing
- etc.

Satisfiability Testing

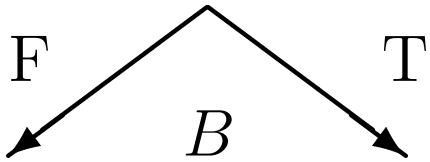
Best-known method: Davis-Putnam Procedure (1960)

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

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



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



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

$\neg C$

$\times$

$\times$

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.

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 $e^{-\delta/T}$ ("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.

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)
- 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	*	*

- 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 better than DP.*

Local Search Obstacles

- Local maxima (minima)

- Plateaus

1. local maximum: rate of change is negative in all directions even though the global maximum hasn't been reached. (maze problem — may have to move AWAY from goal to find the (best) solution)
2. plateaus: all of the local steps look the same (in 8-puzzle, maybe none of the available moves changes the number of tiles out of place)

Approaches

1. Random restarts
2. Simulated annealing
3. Mixed-in random walk
4. Tabu search (prevent repeated states)
5. Genetic algorithms
6. Genetic programming

Local Search — Summary

Surprisingly efficient search technique

- Wide range of applications
- Formal properties elusive
- Intuitive explanation:
  - Search spaces are too large for systematic search anyway ...
  - Area will most likely continue to thrive
  - Often best evailable with lack of global info.
  - To some extent how evolution build us ....