Title: Informed Search Methods

Required reading: AIMA, Chapter 4. LWH: Chapter 13 and 14.

Introduction to Artificial Intelligence

CSCE 476-876, Spring 2003

URL: `www.cse.unl.edu/~choueiry/S03-476-876`

Berthe Y. Choueiry (Shu-we-ri)

`choueiry@cse.unl.edu, (402)472-5444`

Outline

- Categorization of search techniques
- Ordered search (search with an evaluation function)
- Best-first search:
  (1) Greedy search (2) A^*
- Admissible heuristic functions:
  how to compare them?
  how to generate them?
  how to combine them?
- Iterative improvement search:
  (1) Hill-climbing (2) Simulated annealing

Types of Search (I)

1- Uninformed vs. informed

2- Systematic/constructive vs. iterative improvement

Uninformed :

use only information available in problem definition,

<u>no idea about distance to goal</u>

→ can be incredibly ineffective in practice

Heuristic :

exploits some knowledge of the domain

also useful for solving optimization problems

Types of Search (II)

Systematic, exhaustive, constructive search:

a <u>partial solution</u> is incrementally extended into <u>global solution</u>

Partial solution =

sequence of transitions between states

Global solution =

Solution from the initial state to the goal state

Examples: $\begin{cases} \text{Uninformed} \\ \text{Informed (heuristic): Greedy search, A}^* \end{cases}$

→ Returns the path; solution = path

Types of Search (III)

Iterative improvement:

A state is gradually modified and evaluated until reaching an (acceptable) optimum

→ We don't care about the path, we care about 'quality' of state
→ Returns a state; a solution = good quality state
→ Necessarily an informed search

Examples (informed):
- Hill climbing
- Simulated Annealing (physics), Taboo search
- Genetic algorithms (biology)

Ordered search

• Strategies for systematic search are generated by choosing which node from the fringe to expand first

• The node to expand is chosen by an **evaluation function**, expressing 'desirability' ⟶ **ordered search**

• When nodes in queue are sorted according to their decreasing values by the evaluation function ⟶ **best-first search**

• Warning: 'best' is actually 'seemingly-best' given the evaluation function. Not always best (otherwise, we could march directly to the goal!)

Search using an evaluation function

- Example: uniform-cost search!

  What is the evaluation function?

  Evaluates cost from .... to ....?

- How about the cost **to** the goal?

  $h(n)$ = estimated cost of the cheapest path from the state at node n to a goal state

  $h(n)$ would help focusing search

Best-first search

1. Greedy search chooses the node n closest to the goal such as $h(n)$ is minimal

2. A* search chooses the least-cost solution

   solution cost $f(n) \begin{cases} g(n)\text{: cost from root to a given node } n \\ + \\ h(n)\text{: cost from the node } n \text{ to the goal node} \end{cases}$

   such as $f(n) = g(n) + h(n)$ is minimal

Greedy search (I)

→ First expand the node whose state 'closest' to the goal!

→ Minimize $h(n)$

function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

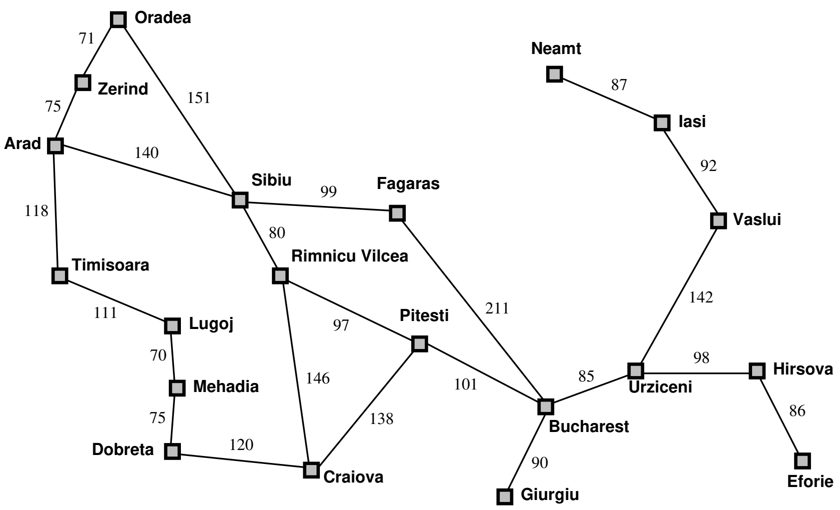
→ Usually, cost of reaching a goal may be estimated,
not determined exactly

→ If state at n is goal, $h(n)=$?

→ How to choose $h(n)$? Problem specific! Heuristic!

Greedy search (II) Romania

$$h_{\mathtt{SLD}}(n) = \text{straight-line distance between } n \text{ and goal location}$$

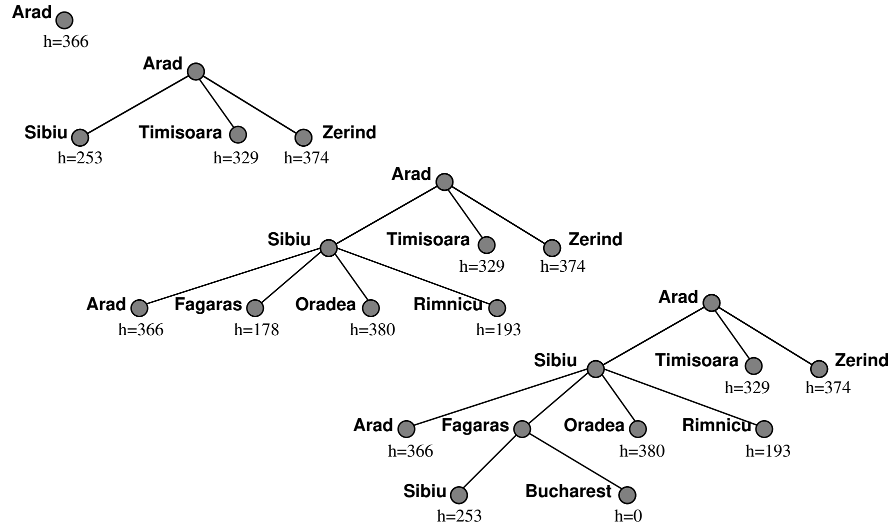


Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search (III) Romania

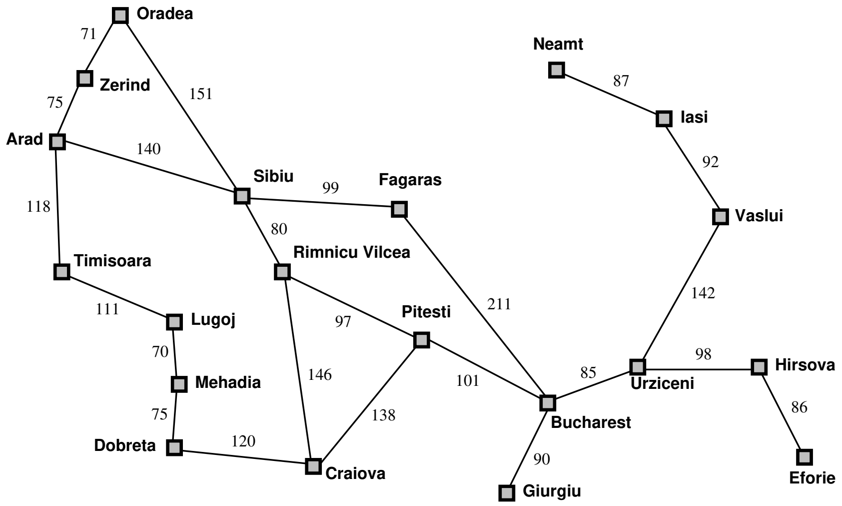
Trip from Arad to Bucharest



... Greedy search!   quick, but not optimal!

Greedy search (IV) Problems

From Iasi to Fagaras? { False starts: Neamt is a dead-end
Looping



Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search (V) Properties

→ Like depth-first, tends to follow a single path to the goal

→ Like depth-first $\begin{cases} \text{Not complete} \\ \text{Not optimal} \end{cases}$

→ Time complexity: $O(b^m)$, m maximum depth

→ Space complexity: $O(b^m)$ retains all nodes in memory

→ Good h function (considerably) reduces space and time
but h functions are problem dependent :—(

Hmm...

Greedy search minimizes estimated cost to goal $h(n)$
→ cuts search cost considerably
→ but not optimal, not complete

Uniform-cost search minimizes cost of the path so far $g(n)$
→ is optimal and complete
→ but can be wasteful of resources

New-Best-First search minimizes $f(n) = g(n) + h(n)$
→ combines greedy and uniform-cost searches
$f(n)$ = estimated cost of cheapest solution via n
→ Provably: complete and optimal, if $h(n)$ is admissible

A* Search (I)

$$\left.\begin{array}{l}\text{Best-first search}\\ h(n)\ \text{admissible}\end{array}\right\} \longrightarrow \text{A}^*\ \text{search, provably} \left\{\begin{array}{l}\text{optimal}\\ \text{complete}\end{array}\right.$$

Admissible heuristic

An admissible heuristic is a heuristic that never overestimates the cost to reach the goal

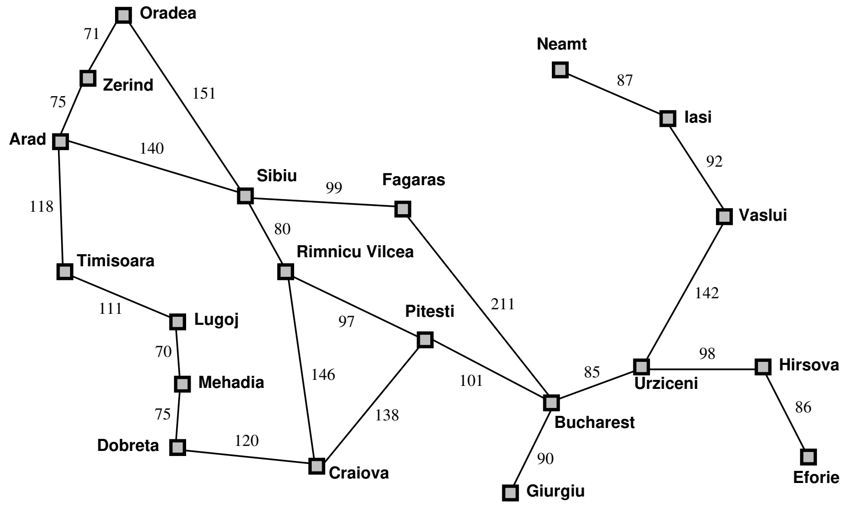
→ is optimistic

→ thinks the cost of solving is less than it actually is

$$\text{Example:}\left\{\begin{array}{l}\text{travel: straight line distance}\\ \text{I can finish college in 3 years}\\ \text{We can fly to Mars by 2003}\end{array}\right.$$

If h is admissible,
$f(n)$ never overestimates the actual cost of the best solution through n.

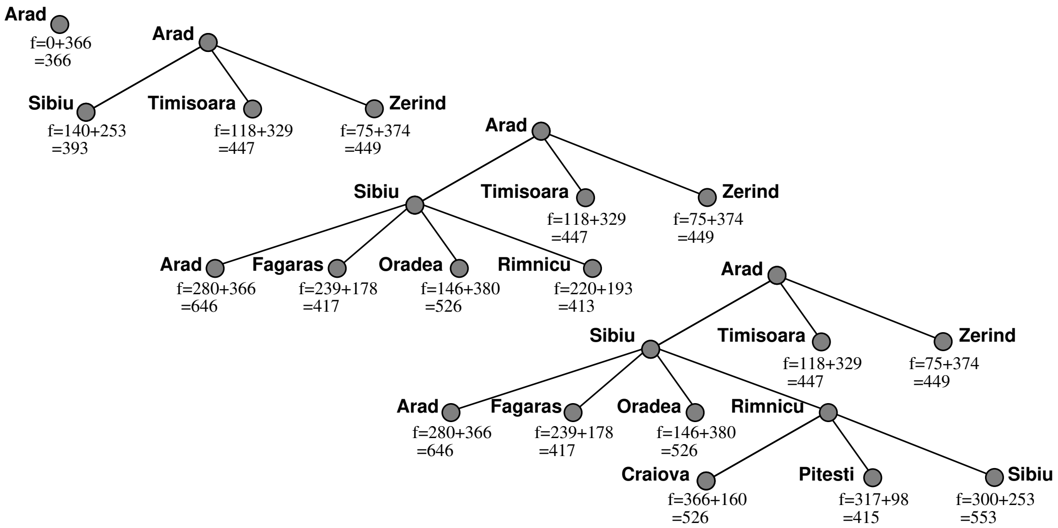
A* Search (II) Romania: from Arad to Bucharest



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search (III) Romania: from Arad to Bucharest

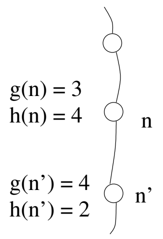


A* Search (IV) Observation: pathmax

Monotonicity: f-cost never decreases
This holds for almost all admissible heuristics

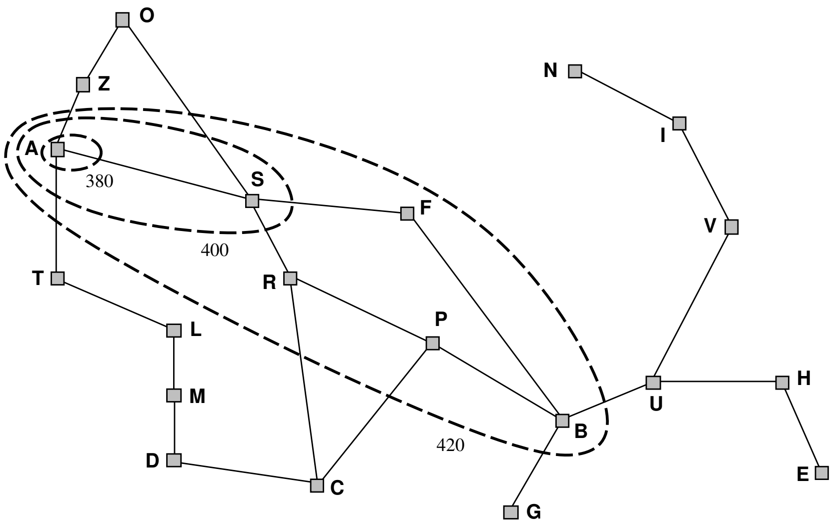
If f is not monotonic, use **pathmax** equation

$$f(n') = max(f(n), g(n') + h(n'))$$



A* never decreasesalong any path out from root
$\longrightarrow$ we can conceptually draw contours in the search space

A* Search (V) Romania: from Arad to Bucharest



$\mathbf{A^*}$ **Search** (VI) Observations

We define f^*: cost of optimal solution

$$\text{A}^* \text{ search} \begin{cases} \text{expands all nodes with } f(n) < f^* \\ h \text{ admissible, thinks the cost of solving} < \text{it actually is} \\ \\ \text{may expand nodes on the `goal-contour' } f(n) = f^* \\ \text{before selecting goal} \end{cases}$$

A* is optimally efficient, optimal, complete

$\mathbf{A^*}$ **Search** (VII) is optimally efficient

.. for any given evaluation function: no other algorithms that finds the optimal solution is guaranteed to expend fewer nodes than A*

Interpretation (proof not presented): Any algorithm that does not expand all nodes between root and the goal contour risks missing the optimal solution

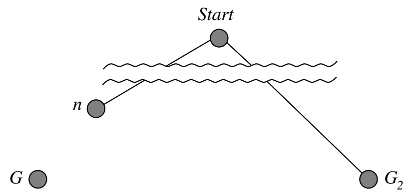
A^* Search (VIII) is optimal

G, G_2 goal states $\Rightarrow g(G) = f(G)$, $f(G_2) = g(G_2)$ $\quad h(G) = h(G_2) = 0$

G optimal goal state $\Rightarrow f^* = f(G)$

G_2 suboptimal $\Rightarrow f(G_2) > f^* = f(G)$ (1)

Suppose n is not chosen for expansion



h admissible $\Rightarrow f^* \geq f(n)$ (2)

Since n was not for expansion $\Rightarrow f(n) \geq f(G_2)$ (3)

(2) + (3) $\Rightarrow f^* \geq f(G_2)$ (4)

(1) and (4) are contradictory $\Rightarrow n$ should be chosen for expansion

A^* Search (IX) is complete

Since A^* search expands all nodes with $f(n) < f^*$, it must eventually reach the goal state unless there are infinitely many

$$\text{nodes } f(n) < f^* \begin{cases} \text{1. } \exists \text{ a node with infinite branching factor} \\ \text{or} \\ \text{2. } \exists \text{ a path with infinite number of nodes along it} \end{cases}$$

$$A^* \text{ is complete if} \begin{cases} \text{on locally finite graphs} \\ \text{and} \\ \exists \delta > 0 \text{ constant, the cost of each operator} > \delta \end{cases}$$

A^* **Search** (X) Complexity

Time :

Exponential in (relative error in h × length of solution path)

... quite bad

Space : must keep all nodes in memory

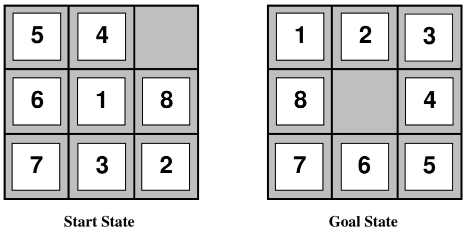
major drawback of A^*

→ Memory Bounded Search in Section 4.3 (not addressed here)

Admissible Heuristic functions (I)

Examples

- Route-finding problems: straight-line distance
- 8-puzzle: $\begin{cases} h_1(n) = \text{number of misplaced tiles} \\ h_2(n) = \text{total Manhattan distance} \end{cases}$



$h_1(S) = ?$

$h_2(S) = ?$

Admissible heuristic functions (II)

Performance

Two criteria to compare admissible heuristic functions:

1. Effective branching factor: b^*
2. Dominance: number of nodes expanded

Effective branching factor b^*

– The heuristic expands N nodes in total

– The solution depth is d

$\longrightarrow$ b^* is the branching factor had the tree been uniform

$$N = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d = \frac{(b^*)^{d+1} - 1}{b^* - 1}$$

– Example: N=52, d=5 $\rightarrow$ $b^* = 1.91$

Dominance

If $h_2(n) \geq h_1(n)$ for all n (both admissible)
then h_2 *dominates* h_1 and is better for search

Typical search costs: nodes expanded

Sol. depth	**IDS**	$\mathbf{A}^*(h_1)$	$\mathbf{A}^*(h_2)$
$d = 14$	3,473,941	539	113
$d = 24$	too many	39,135	1,641

A* expands all nodes $f(n) < f^* \Rightarrow g(n) + h(n) < f^*$
$\Rightarrow h(n) < f^* - g(n)$

If $h_1 \leq h_2$, A* with h_1 will always expand at least as many (if not more) nodes than A* with h_2

$\longrightarrow$ It is always better to use a heuristic function with higher values, as long as it does not overestimate (remains admissible)

Admissible heuristic functions (III)

How to generate admissible heuristics?

$\rightarrow$ Use *exact* solution cost of a relaxed (easier) problem

Steps:
- Consider problem P
- Take a problem P' easier than P
- Find solution to P'
- Use solution of P' as a heuristic for P

• If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*: the shortest solution can be used as a heuristic ($\equiv h_1(n)$)

• If the rules are relaxed so that a tile can move to *any adjacent square*: the shortest solution can be used as a heuristic ($\equiv h_2(n)$)

• For TSP: let path be *any* structure that connects all cities
$\Longrightarrow$ minimum spanning tree heuristic (polynomial)

Admissible heuristic functions (IV)

We have a set of admissible heuristics $h_1, h_2, h_3, \ldots, h_m$ but no heuristic that dominates all others, what to do?

$$\longrightarrow h(n) = \texttt{max}(h_1(n), h_2(n), \ldots, h_m(n))$$

$\rightarrow$ Problem:
Cost of computing the heuristic (vs. cost of expanding nodes)

Iterative improvement (a.k.a. local search)

$\longrightarrow$ Sometimes, the 'path' to the goal is irrelevant
only the state description (or its quality) is needed

$$e.g., \text{ a CSP: } \begin{cases} \text{Are all constraints satisfied?} \\ \text{Is a maximum \# of constraints satisfied?} \\ \text{Is a maximum \# of important constraints satisfied?} \end{cases}$$

Iterative improvement search:
gradually modify a sub-optimal state
until reaching a near-optimal state

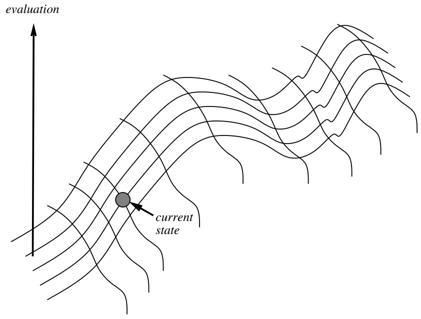
Example: complete-state formulation of N-queens

Intuition

All states are layed up on the surface of a landscape

A state quality/value is its height

Move around and try to reach the highest peak



Two major classes:

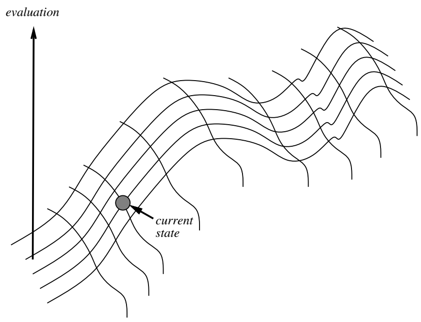
- Hill climbing (a.k.a. gradient ascent/descent)
  → try to make changes to improve current state
- Simulated Annealing (physics)
  → things can temporarily get worse

Hill climbing

Start from any state at random and loop:

Examine all direct neighbors

If a neighbor has higher value then move to it else exit



Problems:

- Local maxima: search halts
- Plateau (flat): random walk
- Ridge: may oscillate from side to side, slow progress

Random-restart hill-climbing

→ When HC halts or no progress is made
re-start from a different (randomly chosen) starting
save best results found so far

→ Repeat random restart
- for a fixed number of iterations, or
- until best results have not been improved for a certain number of iterations

⟶ Optimality (soundness)? Completeness?
⟶ Complexity: space? time?

⟶ In practice, surprisingly good.. (eroding myth)

Simulated annealing (I)

Basic idea: When stuck in a local maximum allow few steps towards less good neighbors to escape the local maximum

Start from any state at random, start count down and loop until time is over:

Pick up a neighbor at <u>random</u>
Set ΔE = value(neighbor) - value(current state)
If ΔE>0 (neighbor is better)
then move to neighbor & restart count down,
else ΔE<0 move to it with probability < 1

$$\text{Transition probability} \simeq e^{\Delta E/T} \begin{cases} \Delta\text{E is negative} \\ \text{T: count-down time} \end{cases}$$

as time passes, less and less likely to make the move towards 'unattractive' neighbors

Simulated annealing (II)

Analogy to physics:

Gradually cooling a liquid until it freezes
If temperature is lowered sufficiently slowly, material will attain lowest-energy configuration (perfect order)

Count down	$\longleftrightarrow$	Temperature
Moves between states	$\longleftrightarrow$	Thermal noise
Global optimum	$\longleftrightarrow$	Lowest-energy configuration

How about decision problems?

Optimization problems		**Decision problems**
Iterative improvement	$\longleftrightarrow$	Iterative repair
State value	$\longleftrightarrow$	Number of constraints violated
Sub-optimal state	$\longleftrightarrow$	Inconsistent state
Optimal state	$\longleftrightarrow$	Consistent state