# Local Search for a Globally Optimal Solution

Russell and Norvig Chapter 4

## Limitations of hill climbing

- Can we find a globally optimal solution?
- Can't guarantee that. A good alternative: an approach that will give that "with high probability".
- Need a more thorough exploration of the state space

## Simulated Annealing

- Physical systems are good at finding minimum energy configurations: a physical system cooled down to absolute zero will settle into its minimum energy configuration.
- Mimic the process using a probabilistic process

## Simulated Annealing

- Simulated annealing [Kirkpatrick, Gelatt, Vecchi, 1983].
  - T large ⇒ probability of accepting an uphill move is large.
  - T small ⇒ uphill moves are almost never accepted.
  - Idea: turn knob to control T.
  - Cooling schedule: T = T(i) at iteration i.
- Physical analogy.
  - Take a molten solid and freeze it very abruptly, we do not expect to get a perfect crystal.
  - Annealing: cool material gradually from high temperature, allowing it to reach equilibrium at succession of intermediate lower temperatures.
  - Algorithm will find optimal solution with high probability if using a sufficiently slow cooling schedule.

4

## Example

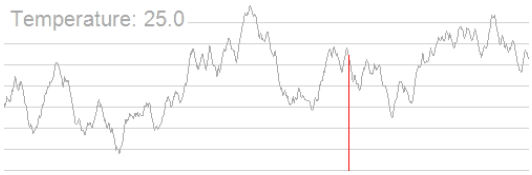



Image from http://en.wikipedia.org/wiki/Simulated_annealing

## Simulated Annealing

```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
           schedule, a mapping from time to temperature

    current ← MAKE-NODE(INITIAL-STATE[problem])
    for t ← 1 to ∞ do
        T ← schedule[t]
        if T = 0 then return current.
        next ← a randomly selected successor of current
        ΔE ← current.VALUE - next.VALUE
        if ΔE > 0 then current ← next
        else current ← next with probability e^(ΔE/T)
```

Temperature controls the probability of increasing steps.

## Properties of Simulated Annealing

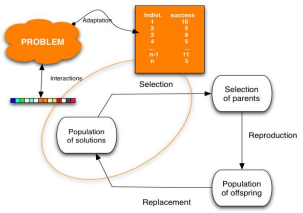
- As the number of moves at a given temperature goes to infinity, the probability of a state becomes proportional to exp(-*E*/*T*) (Boltzman distribution)
- If temperature is lowered slowly enough - global optimum will be found with high probability. A lot of research into what makes a good cooling schedule.
- Widely used in a variety of applications (VLSI layout, airline scheduling, etc.)

## Beam Search

- Variant of hill climbing:
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any successor is optimal → done
  - Else **select *k* best** from successors and repeat.
- Major difference with random-restart search:
  - Information is shared among *k* search threads.
- Can suffer from lack of diversity.

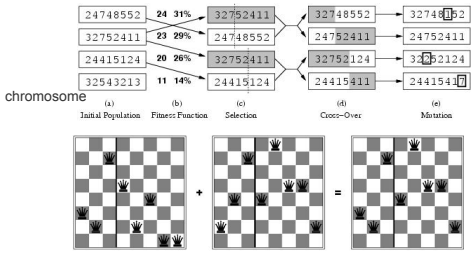
## Genetic algorithms

- Keep a population of solutions that undergo recombination and mutation




## Genetic Algorithms

The state is the genetic material that makes an individual




## Genetic Algorithms

**function** GENETIC_ALGORITHM(*population*, FITNESS-FN) **return** an individual
  **input:** *population*, a set of individuals
    FITNESS-FN, a function quantifying the quality of an individual
  **repeat**
    *new_population* ← empty set
    **for** i =1 **to** SIZE(*population*) **do**
      *x* ← RANDOM_SELECTION(*population*, FITNESS_FN)
      *y* ← RANDOM_SELECTION(*population*, FITNESS_FN)
      *child* ← REPRODUCE(*x*,*y*)
      MUTATE(*child*)
      add *child* to *new_population*
    *population* ← *new_population*
  **until** some individual is fit enough or enough time has elapsed
  **return** the best individual

## Solving TSP

- Represent a tour as a permutation $(i_1,\ldots,i_n)$ of {1,2, …,n}
- Fitness of a solution: negative of the cost of the tour
- Initialization: using either some heuristic, or a random set of permutations
- Need to define crossover and mutation operations.

## Crossover

- **Order** crossover: choose a subsequence of a tour from one parent and preserve the relative order of the cities from the other.

  Example:

  $p_1$ = (1 2 3 | 5 4 6 7 | 8 9)
  $p_2$ = (4 5 2 | 1 8 7 6 | 9 3)
  $c_1$ = (x x x | 5 4 6 7 | x x)
  $c_2$ = (x x x | 1 8 7 6 | x x)

  The tour in $p_2$, starting from its second cut point, is $9 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 8 \rightarrow 7 \rightarrow 6$. Remove the cities already in $c_1$, obtaining the partial tour $9 \rightarrow 3 \rightarrow 2 \rightarrow 1 \rightarrow 8$. Insert this partial tour after the second cut point of $c_1$, resulting in $c_1$ = (2 1 8 | 5 4 6 7 | 9 3 ).

## Crossover (2)

- **Partially Mapped** (PMX) crossover: choose a subsequence of a tour from one parent and preserve the order and position of as many cities as possible from the other parent.

## PMX crossover

$p_1$ = (1 2 3 | 4 5 6 7 | 8 9)
$p_2$ = (4 5 2 | 1 8 7 6 | 9 3)
$c_1$ = (x x x | 4 5 6 7 | x x)
$c_2$ = (x x x | 1 8 7 6 | x x)

Swap defines a mapping:

$1 \leftrightarrow 4,\ 8 \leftrightarrow 5,\ 7 \leftrightarrow 6,\ 6 \leftrightarrow 7.$

The easy ones:

$c_1$ = (x 2 3 | 1 8 7 6 | x 9)
$c_2$ = (x x 2 | 4 5 6 7 | 9 3)

For the rest, use the mapping:

$c_1$ = (4 2 3 | 1 8 7 6 | 5 9)
$c_2$ = (1 8 2 | 4 5 6 7 | 9 3).

## Mutation

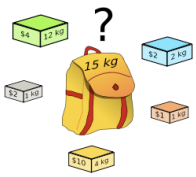
Can use a 2-opt operation:

Select two points along the permutation, cut it at these points and re-insert the reversed string.

Example:

(1 2 | 3 4 5 6 | 7 8 9) → (1 2 | 6 5 4 3 | 7 8 9)

## The knapsack problem




## Local search: summary

- Why I like local search algorithms
  - Easy to implement
  - Widely applicable
  - Provide good results

## Online search

- So far, we have assumed deterministic actions and fully-known environments
  - Permits off-line search
- Consider a new problem:
  - A robot is placed in the middle of a maze
  - The task is to find the exit
  - Actions are deterministic but the environment is unknown

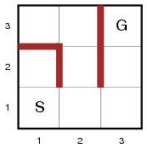
Questions: Can the robot do A* search?

## Online agents

Difference from offline agents: An online agent can only expand the node it is physically in.

Therefore agent needs to work locally: Online DFS, IDS.

Possible only when actions are reversible.




## Simple online search

- Let us assume that actions are reversible and deterministic
  - Like the robot maze (with no holes, slides, etc.)
- Then we can perform local search with an evaluation heuristic:
  - For all actions in state s:
    Try the action, leading to state s'
    Evaluate s'
    Reverse the action, returning to state s
  - Select best action, going to state s'
  - If s' is a goal, return; else repeat loop.