# Lecture 6 of 42

## Informed Search:
## Local (Hill-Climbing, Beam) vs.
## Global (Simulated Annealing, Genetic)

**William H. Hsu**
**Department of Computing and Information Sciences, KSU**

**KSOL course page:** http://snipurl.com/v9v3
**Course web site:** http://www.kddresearch.org/Courses/CIS730
**Instructor home page:** http://www.cis.ksu.edu/~bhsu

**Reading for Next Class:**

Sections 5.1 – 5.3, p. 137 – 151, Russell & Norvig 2nd edition
Instructions for writing project plans, submitting homework

## Lecture Outline

- Reading for Next Class: Sections 5.1 – 5.3, R&N 2$^{e}$
- Today: Chapter 4 concluded
  - Properties of search algorithms, heuristics
  - Local search (hill-climbing, Beam) vs. nonlocal search
  - Problems in heuristic search: plateaux, "foothills", ridges
  - Escaping from local optima
  - Wide world of global optimization: genetic algorithms, simulated annealing
- Next class: start of Chapter 5 (Constraints)
  - State space search: graph vs. constraint representations
  - Constraint Satisfaction Problems (CSP)
- Next Week: Constraints and Games
  - Lecture 7: CSP algorithms (Chapter 5 concluded)
  - Lecture 8: Intro to Game Tree Search (Chapter 6)

## Acknowledgements

Stuart Russell

Professor of Computer Science
Chair, Department of Electrical Engineering and Computer Sciences
Smith-Zadeh Professor in Engineering
Computer Science Division
387 Soda Hall
University of California
Berkeley, CA 94720-1776

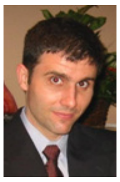
© 2004-2005

Russell, S. J.
University of California, Berkeley
http://www.eecs.berkeley.edu/~russell/

Peter Norvig
Director of Research Google

Norvig, P.
http://norvig.com/

Slides from:
http://aima.eecs.berkeley.edu



PRISM
Pattern Recognition and Intelligent Sensor Machines Texas A&M University

Ricardo Gutierrez-Osuna
Associate Professor

Research interests: Pattern recognition, intelligent sensors, machine olfaction, speech-driven facial animation, biological cybernetics, mobile robotics, machine learning.

© 2005 R. Gutierrez-Osuna
Texas A&M University
http://research.cs.tamu.edu/prism/

## Monotonicity (Consistency) & Pathmax: Review

A heuristic is consistent if

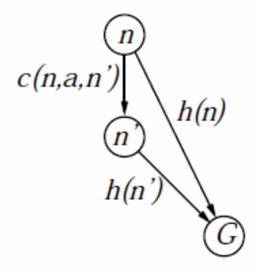
$$h(n) \leq c(n,a,n') + h(n')$$

**Monotone Restriction (a triangle inequality)**

If $h$ is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n,a,n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

I.e., $f(n)$ is nondecreasing along any path.



**Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.**

**If *h* not consistent, use Pathmax to make it consistent:**

$$h'(P) \leftarrow \max(h(P), h'(N) - c(N,P))$$

**Wikipedia (2009). *Consistent Heuristic.***
**http://en.wikipedia.org/wiki/Consistent_heuristic**

## Hill-Climbing aka Gradient Descent

- **function** *Hill-Climbing* (*problem*) **returns** solution state
  - inputs: *problem*: specification of problem (structure or class)
  - static: *current*, *next*: search nodes
  - *current* ← *Make-Node* (*problem.Initial-State*)
  - loop do
    - ⇨ *next* ← a highest-valued successor of *current*
    - ⇨ if *next.value*() < *current.value*() then return *current*
    - ⇨ *current* ← *next* // make transition
  - end
- Steepest Ascent Hill-Climbing
  - *aka* gradient ascent (descent)
  - Analogy: finding "tangent plane to objective surface"
  - Implementations
    - ⇨ Finding derivative of (differentiable) *f* with respect to parameters
    - ⇨ Example: error backpropagation in artificial neural networks (later)
- Discussion: Difference Between Hill-Climbing, Best-First?

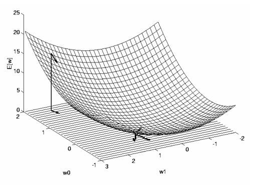
## Iterative Improvement: Framework

- Intuitive Idea
  - "Single-point search frontier"
    - ⇨ Expand one node at a time
    - ⇨ Place children at head of queue
    - ⇨ *Sort only this sublist, by f*
  - Result – direct convergence in direction of steepest:
    - ⇨ Ascent (in criterion)
    - ⇨ Descent (in error)
  - Common property: proceed toward goal *from search locus (or loci)*
- Variations
  - Local (steepest ascent hill-climbing) versus global (simulated annealing or SA)
  - Deterministic *versus* Monte-Carlo
  - Single-point *versus* multi-point
    - ⇨ Maintain frontier
    - ⇨ Systematic search (cf. OPEN / CLOSED lists): parallel SA
    - ⇨ Search with recombination: genetic algorithm

## Hill-Climbing [1]: An Iterative Improvement Algorithm

- **function** ***Hill-Climbing*** **(*problem*) returns solution state**
  - **inputs:** ***problem*: specification of problem (structure or class)**
  - **static:** ***current*, *next*: search nodes**
  - ***current* ← *Make-Node* (*problem.Initial-State*)**
  - **loop do**
    - ⇨ ***next* ← a highest-valued successor of *current***
    - ⇨ **if *next.value*() < *current.value*() then return *current***
    - ⇨ ***current* ← *next*   // make transition**
  - **end**
- **Steepest Ascent Hill-Climbing**
  - ***aka* gradient ascent (descent)**
  - **Analogy: finding "tangent plane to objective surface"**
  - **Implementations**
    - ⇨ **Finding derivative of (differentiable) *f* with respect to parameters**
    - ⇨ **Example: error backpropagation in artificial neural networks (later)**
- **Discussion: Difference Between Hill-Climbing, Best-First?**

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$



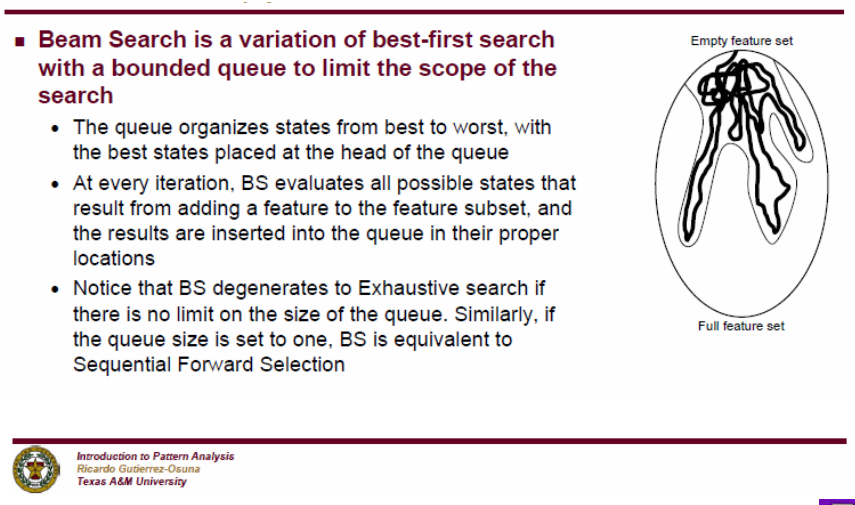
## Hill-Climbing [2]: Restriction of Best-First Search

- **Discussion: How is Hill-Climbing a Restriction of Best-First?**
- **Answer: Dropped Condition**
  - **Best first: sort by *h* or *f over current frontier***
    - ⇨ **Compare: insert each element of expanded node into queue, in order**
    - ⇨ **Result: greedy search (*h*) or A/A* (*f*)**
  - **Hill climbing: sort by *h* or *f within child list of current node***
    - ⇨ **Compare: local bucket sort**
    - ⇨ **Discussion (important): *Does it matter whether we include g?***
- **Impact of Modification on Algorithm**
  - **Search time complexity decreases**
  - **Comparison with A/A* (Best-First using *f*)**
    - ⇨ ***Still optimal?*   No**
    - ⇨ ***Still complete?*   Yes**
  - **Variations on hill-climbing (later): momentum, random restarts**

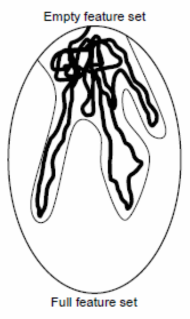
## Beam Search [1]: "Parallel" Hill-Climbing

- **Idea**
  - **Teams of climbers**
    - ⇨ **Communicating by radio**
    - ⇨ **Frontier is only *w teams wide* (*w* ≡ beam width)**
    - ⇨ **Expand cf. best-first but take best *w* only per layer**
  - **Synchronous search: push frontier out to uniform depth from start node**
- **Algorithm Details**
  - **How do we order OPEN (priority queue) by *h*?**
  - **How do we maintain CLOSED?**
- **Question**
  - **What behavior does beam search with *w* = 1 exhibit?**
  - **Hint: only one "team", can't split up!**
  - **Answer: equivalent to hill-climbing**
- **Other Properties, Design Issues**
  - **Another analogy: flashlight *beam* with adjustable radius (hence name)**
  - **What should *w* be? How will this affect solution quality?**

## Beam Search [2]: Restriction of Best-First Search



- **Beam Search is a variation of best-first search with a bounded queue to limit the scope of the search**
  - The queue organizes states from best to worst, with the best states placed at the head of the queue
  - At every iteration, BS evaluates all possible states that result from adding a feature to the feature subset, and the results are inserted into the queue in their proper locations
  - Notice that BS degenerates to Exhaustive search if there is no limit on the size of the queue. Similarly, if the queue size is set to one, BS is equivalent to Sequential Forward Selection

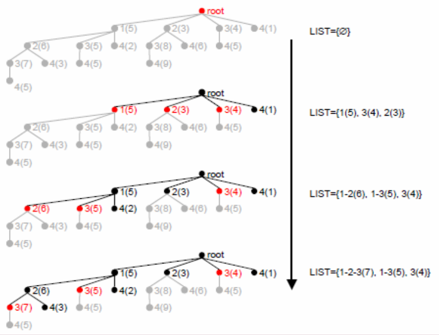


*Introduction to Pattern Analysis*
*Ricardo Gutierrez-Osuna*
*Texas A&M University*

**© 2005 R. Gutierrez-Osuna, Texas A&M University** **http://tr.im/yCaX**

## Beam Search [2]: Restriction of Best-First Search

- The example below illustrates BS for a 4-dimensional search space and a queue of size 3
  - BS cannot guarantee that the optimal subset is found:
    - In the example, the optimal is 2-3-4 (J=9), which is never explored
    - However, with the proper queue size, Beam Search can avoid getting trapped in local minimal by preserving solutions from varying regions in the search space



*Introduction to Pattern Analysis*
*Ricardo Gutierrez-Osuna*
*Texas A&M University*

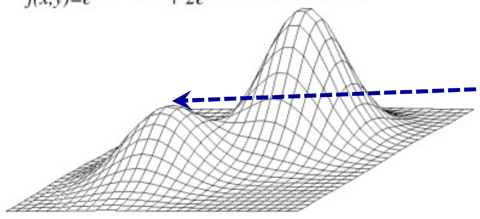
http://tr.im/yCaX

## Problem Situations in Search

- **Optimization-Based Problem Solving as Function Maximization**
- **Foothills *aka* Local Optima**
  - ***aka* relative minima (of error), relative maxima (of criterion)**
  - **Qualitative description**
    - ⇨ **All applicable operators produce suboptimal results (i.e., neighbors)**
    - ⇨ ***However, solution is not optimal!***
- **Lack of Gradient *aka* Plateaus (Plateaux)**
  - **All neighbors indistinguishable according to evaluation function *f***
  - **Related problem: jump discontinuities in function space**
- **Single-Step Traps *aka* Ridges**
  - **Inability to move along steepest gradient**
  - **Sensitivity to operators (need to combine or synthesize)**

## Problem Situation 1: Foothills (Local Optima) – Examples

$f(x,y)=e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2+(y-1.7)^2)}$



Local optima

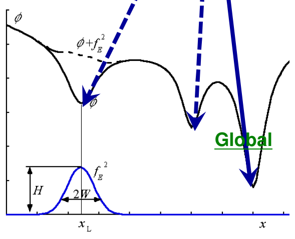
Foothill

Local minima

Wikipedia (2009). *Hill Climbing.*
http://en.wikipedia.org/wiki/Hill_climbing#Problems

Not global

Global



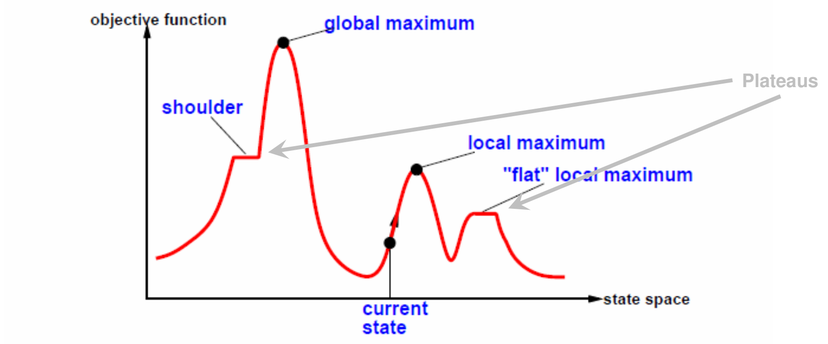
Isshiki, M. (2000).
*Beginner's Guide to Global Explorer.*
http://tr.im/yCt3

## Problem Situation 1: Foothills (Local Optima) – Definition

- Local Optima *aka* Local Trap States
- Problem Definition
  - Point reached by hill-climbing may be maximal but not maximum
  - Maximal
    - ⇨ Definition: *not dominated by any neighboring point* (with respect to criterion measure)
    - ⇨ In this partial ordering, maxima are incomparable
  - Maximum
    - ⇨ Definition: *dominates all neighboring points* (*wrt* criterion measure)
    - ⇨ Different partial ordering imposed: "*z* value"
- Ramifications
  - Steepest ascent hill-climbing will become trapped (*why?*)
  - Need some way to break out of trap state
    - ⇨ Accept transition (i.e., search move) to dominated neighbor
    - ⇨ Start over: random restarts

## Problem Situation 2: Plateaus (Lack of Gradient) - Examples

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

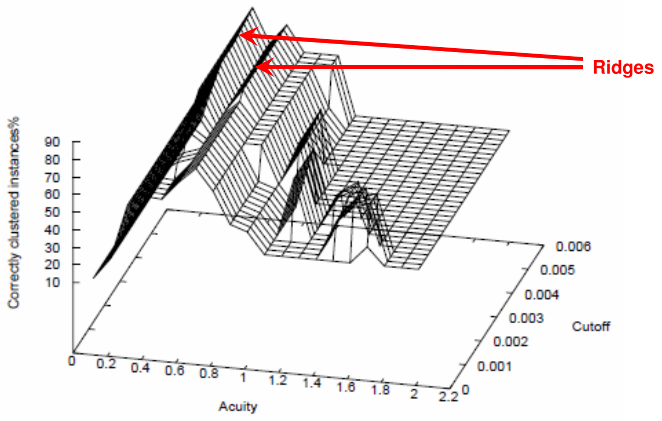
Random sideways moves 😊escape from shoulders ☹loop on flat maxima

Based on slide © 2004 S. Russell & P. Norvig. Reused with permission.

## Problem Situation 2: Plateaus (Lack of Gradient) - Definition

- Zero Gradient Neighborhoods *aka* Plateaus / Plateaux
- Problem Definition
  - Function space may contain points whose neighbors are *indistinguishable* (*wrt* criterion measure)
  - Effect: "flat" search landscape
  - Discussion
    - ⇨ *When does this happen in practice?*
    - ⇨ *Specifically, for what kind of heuristics might this happen?*
- Ramifications
  - Steepest ascent hill-climbing will become trapped (*why?*)
  - Need some way to break out of zero gradient
    - ⇨ Accept transition (i.e., search move) to random neighbor
    - ⇨ Random restarts
    - ⇨ *Take bigger steps* (later, in planning)

## Problem Situation 3: Ridges (Single-Step Traps) – Examples



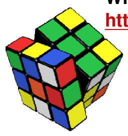
## Problem Situation 3: Ridges (Single-Step Traps) – Definition

- Single-Step Traps *aka* Ridges
- Problem Definition
  - Function space may contain points such that single move in any "direction" leads to suboptimal neighbor
  - Effect
    - ⇨ There exists steepest gradient to goal
    - ⇨ None of allowed steps moves along that gradient
    - ⇨ Thin "knife edge" in search landscape, hard to navigate
    - ⇨ Discussion (important): *When does this occur in practice?*
  - *NB*: ridges can lead to local optima, too
- Ramifications
  - Steepest ascent hill-climbing will become trapped (*why?*)
  - Need some way to break out of ridge-walking
    - ⇨ Formulate composite transition (multi-dimension step) – *how*?
    - ⇨ Accept multi-step transition (at least one to worse state) – *how*?
    - ⇨ Random restarts

## Solution Approach 1: Macros – Intuition

- Intuitive Idea: Take More than One Step in Moving along Ridge
- Analogy: Tacking in Sailing
  - Need to move against wind direction
  - Have to compose move from multiple small steps
    - ⇨ *Combined move*: in (or more toward) direction of steepest gradient
    - ⇨ Another view: *decompose* problem into self-contained *subproblems*
- Multi-Step Trajectories: Macro Operators
  - Macros: (inductively) generalize from 2 to > 2 steps
  - Example: Rubik's Cube
    - ⇨ Can solve 3 x 3 x 3 cube by solving, interchanging 2 x 2 x 2 cubies
    - ⇨ *Knowledge* used to formulate subcube (cubie) as macro operator
  - *Treat operator as single step* (multiple primitive steps)
- Discussion: Issues
  - *How can we be sure macro is atomic? What are pre-, postconditions?*
  - *What is good granularity (size of basic step) for macro in our problem?*

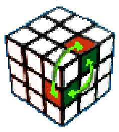
## Solution Approach 1: Macros – Step Size Adjustment



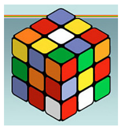
Wikipedia (2009). *Rubik's Cube.*
http://en.wikipedia.org/wiki/Rubik%27s_Cube

© 2007 Harris, D., Lee, J., & Mao, T.
http://tr.im/yCX7

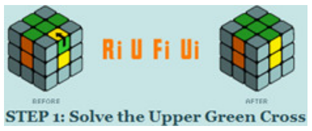
Swapping edge cubelets:

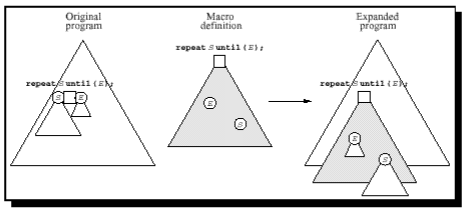


Rubik's Cube
© 1980 Rubik, E.
® Seven Towns, Ltd.



"How to solve the Rubik's Cube."
© 2009 Daum, N.
http://tr.im/yCSA
(Solution uses 7 macro steps such as the one shown at right.)





© 2001 Schwartzbach, M., *et al.*
Basic Research in Computer Science (BRICS)
Aarhus University, Denmark
http://www.brics.dk/bigwig/refman/macro/

## Solution Approach 2: Global optimization – Intuition

- **Going Global: Adapting Gradient Search**
  - Let search algorithm take some "bad" steps to escape from trap states
  - *Decrease* probability of such steps *gradually* to *prevent return to traps*
- **Analogy: Marble(s) on Rubber Sheet**
  - Goal: move marble(s) into global minimum from any starting position
  - Shake system: hard at first, gradually decreasing vibration
  - Tend to break out of local minima but have less chance of re-entering
- **Analogy: Annealing**
  - Ideas from metallurgy, statistical thermodynamics
  - Cooling molten substance: slow as opposed to rapid (quenching)
  - Goal: maximize material strength of substance (e.g., metal or glass)
- **Multi-Step Trajectories in Global Optimization (GO): Super-Transitions**
- **Discussion: Issues**
  - What does convergence mean?
  - What annealing schedule guarantees convergence?

## Solution Approach 2: Global optimization – Randomization

- **Idea: Apply Global Optimization with Iterative Improvement**
  - Iterative improvement: local transition (primitive step)
  - Global optimization (GO) algorithm
    - ⇨ "Schedules" exploration of landscape
    - ⇨ Selects next state to visit
    - ⇨ Guides search by specifying probability distribution over local transitions
- **Randomized GO: Markov Chain Monte Carlo (MCMC) Family**
  - MCMC algorithms first developed in 1940s (Metropolis)
  - First implemented in 1980s
    - ⇨ "Optimization by simulated annealing" (Kirkpatrick *et al.,* 1983)
    - ⇨ Boltzmann machines (Ackley, Hinton, Sejnowski, 1985)
  - Tremendous amount of research and application since
    - ⇨ Neural, genetic, Bayesian computation
    - ⇨ See: CIS730 Class Resources page

## Solution Approach 2A: GO by Simulated Annealing - Algorithm

Idea: escape local maxima by allowing some "bad" moves
but gradually decrease their size and frequency

```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
    inputs: problem, a problem
            schedule, a mapping from time to "temperature"
    local variables: current, a node
                     next, a node
                     T, a "temperature" controlling prob. of downward steps

    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 ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^{Δ E/T}
```

## Solution Approach 2A: GO by Simulated Annealing - Intuition

- **Simulated Annealing is a stochastic optimization method that derives its name from the annealing process used to re-crystallize metals**
  - During the annealing process in metals, the alloy is cooled down slowly to allow its atoms to reach a configuration of minimum energy (a perfectly regular crystal)
    - If the alloy is annealed too fast, such an organization cannot propagate throughout the material. The result will be a material with regions of regular structure separated by boundaries. These boundaries are potential fault-lines where fractures are most likely to occur when the material is stressed
  - The laws of thermodynamics state that, at temperature T, the probability of an increase in energy ΔE in the system is given by the expression

$$P(\Delta E) = e^{-\frac{\Delta E}{kT}}$$

    - where k is known as Boltzmann's constant
- **The SA algorithm is a straightforward implementation of these ideas**

```
1. Determine an annealing schedule T(i)
2. Create an initial solution Y(0)
3. While T(i)>T_MIN
   3a. Generate a new solution Y(i+1) which is a neighbor of Y(i)
   3b. Compute ΔE= [ J(Y(i)) - J(Y(i+1)) ]
   3b. If ΔE<0
          then
             always accept the move from Y(i) to Y(i+1)
          else
             accept the move with probability P=exp(-ΔE/T(i))
```

Empty feature set
Full feature set

## Solution Approach 2A: GO by Simulated Annealing - Properties

At fixed "temperature" $T$, state occupation probability reaches Boltzman distribution

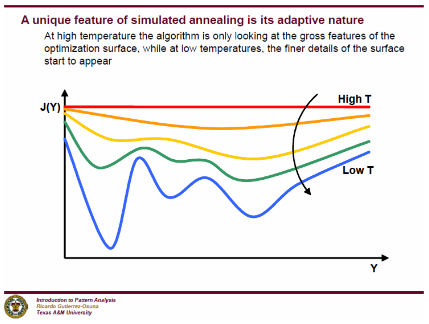
$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

$T$ decreased slowly enough $\Longrightarrow$ always reach best state $x^*$
because $e^{\frac{E(x^*)}{kT}}/e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small $T$

Is this necessarily an interesting guarantee??

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.



**A unique feature of simulated annealing is its adaptive nature**

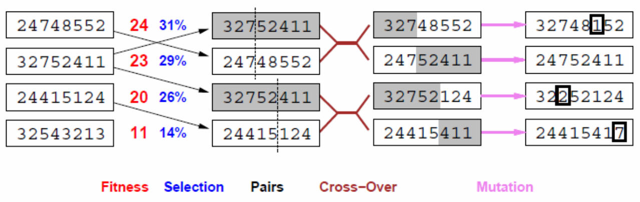
At high temperature the algorithm is only looking at the gross features of the optimization surface, while at low temperatures, the finer details of the surface start to appear

## Solution Approach 2B: GO by Genetic Algorithm

**Algorithm**

```
1. Create an initial random population
2. Evaluate initial population
2. Repeat until convergence (or a number of generations)
   2a. Select the fittest individuals in the population
   2b. Perform crossover on the selected individuals to create offspring
   2c. Perform mutation on the selected individuals
   2d. Create the new population from the old population and the offspring
   2e. Evaluate the new population
```

= stochastic local beam search + generate successors from pairs of states



## Terminology

- Search Frontier: Active Nodes on OPEN List
- Problem Situations in Heuristic Search
  - Foothills: local optima – neighbor states all look worse with respect to h
    - Maxima: hill climbing
    - Minima: gradient descent
  - Plateaus: jump discontinuity – neighbors all look the same wrt h
  - Ridges: single-step trap – neighbors tend to be worse except for narrow path
- Solution Approaches
  - Macro operators
  - Global optimization (GO): simulated annealing (SA), genetic algorithm (GA)
- Iterative Improvement Search Algorithms
  - Hill-climbing: restriction of best-first search to children of current node
  - Beam search
    - Beam width *w*: max number of nodes in search frontier / OPEN
    - Generalization of hill-climbing – sort children of best *w* nodes
  - Simulated annealing: slowly decreasing chance of suboptimal move
  - GA: randomized GO with selection, mutation, reproduction

## Summary Points

- Algorithm A (Arbitrary Heuristic) *vs.* A* (Admissible Heuristic)
- Monotone Restriction (Consistency) and Pathmax
- Local Search: Beam Width *w*
  - Beam search (constant *w* specified as input, “by user”)
  - Hill-climbing (*w* = 1)
- Problems in Heuristic Search: Foothills, Plateaus, Ridges
- Deterministic Global Search: Ordered OPEN List, *w* = ∞
  - Greedy (order by *h*)
  - Uniform cost search (order by *g*: special case of A/A* where *h* = 0)
  - A/A* (order by *f* = *g* + *h*)
- Randomized Global Search
  - Simulated annealing
  - GA: case of genetic and evolutionary computation