Combinatorial Optimization + Fitness Landscapes

Dr. Stephan Steigele

Vorlesung im SS 2008

Bioinf / Universität Leipzig

Recombination issues

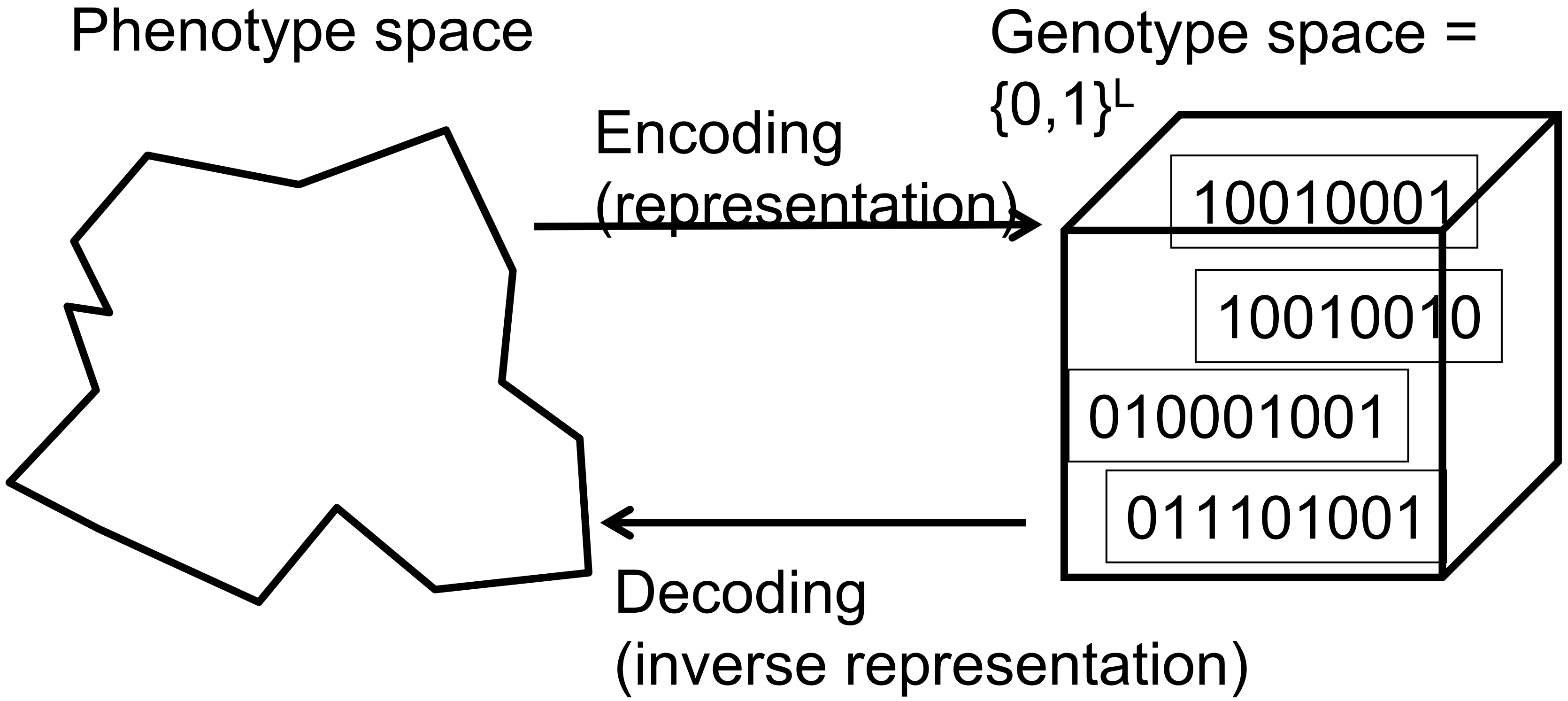
Genetic algorithms

- Holland's original GA is now known as the simple genetic algorithm (SGA)
- Other GAs use different:
  - Representations
  - Mutations
  - Crossovers
  - Selection mechanisms

SGA technical summary tableau

Representation	Binary strings
Recombination	N-point or uniform
Mutation	Bitwise bit-flipping with fixed probability
Parent selection	Fitness-Proportionate
Survivor selection	All children replace parents
Speciality	Emphasis on crossover

Representation


Phenotype space
Genotype space = $\{0,1\}^L$
Encoding (representation)
10010001
10010010
010001001
011101001
Decoding (inverse representation)

SGA reproduction cycle

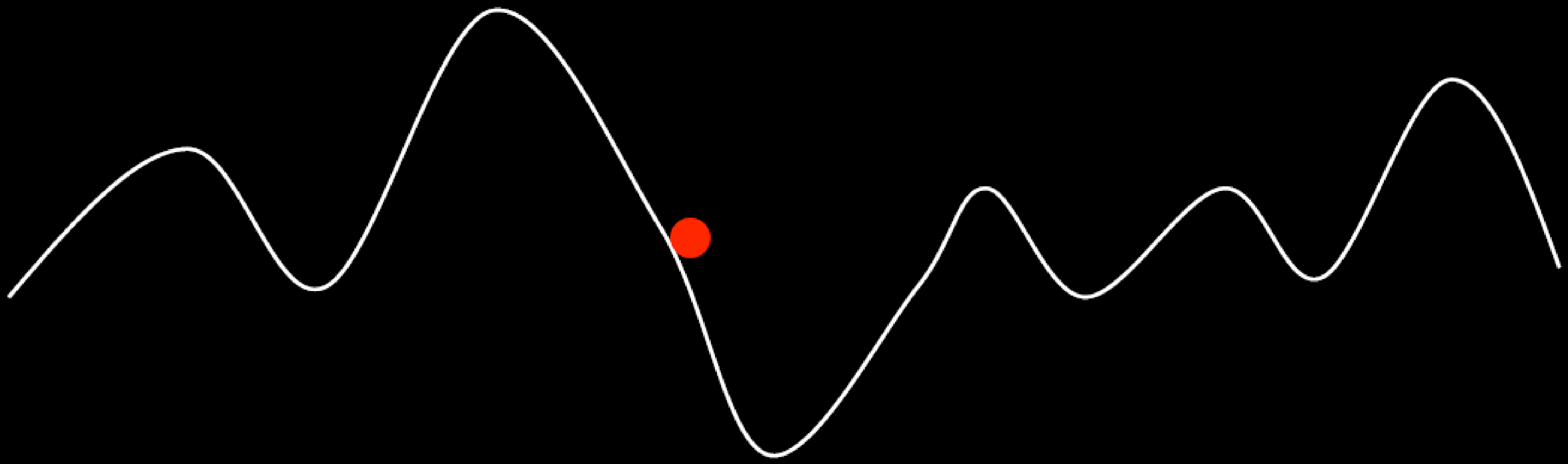
1. Select parents for the mating pool
   (size of mating pool = population size)
2. Shuffle the mating pool
3. For each consecutive pair apply crossover with probability p_c , otherwise copy parents
4. For each offspring apply mutation (bit-flip with probability p_m independently for each bit)
5. Replace the whole population with the resulting offspring

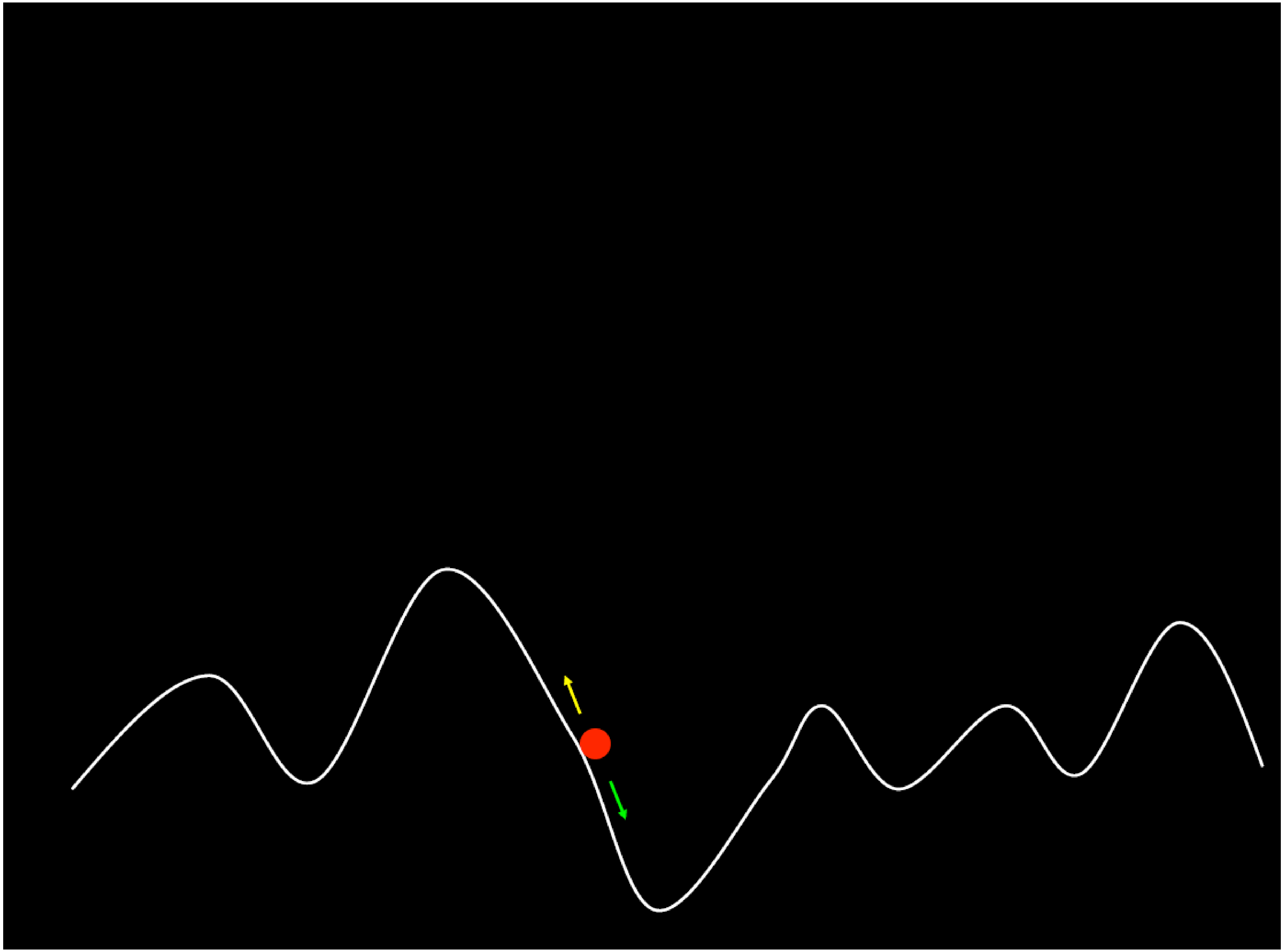
The Real World Isn't (Always) All-Or-Nothing

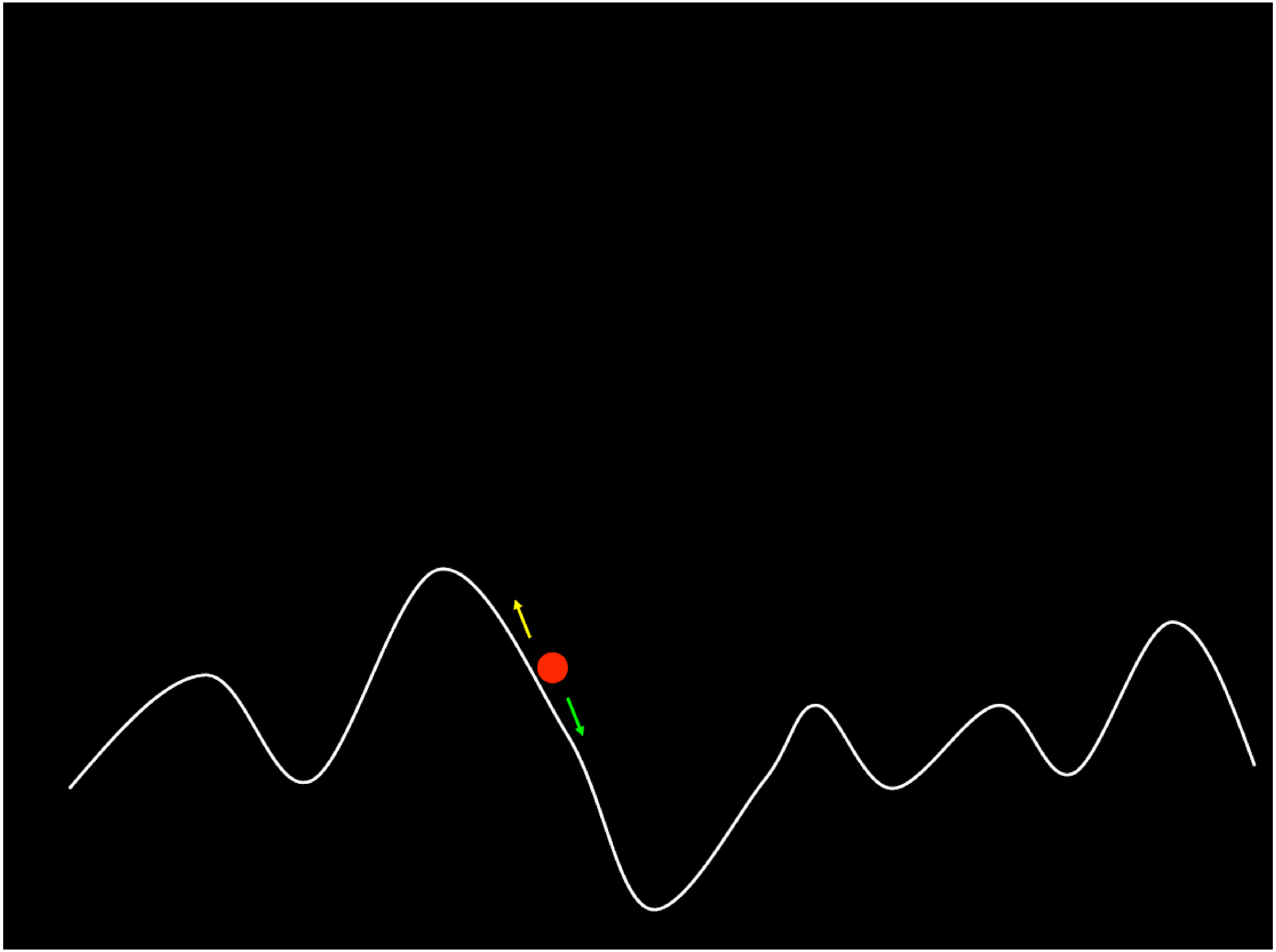
- Lots of real-world choices (profession, location, life partner) involve gradient values (from worst to best)
- Often there are many "locally optimal" solutions - not the absolute best, but good enough
- Then the search for a local optimum looks a lot like hiking a range hills....

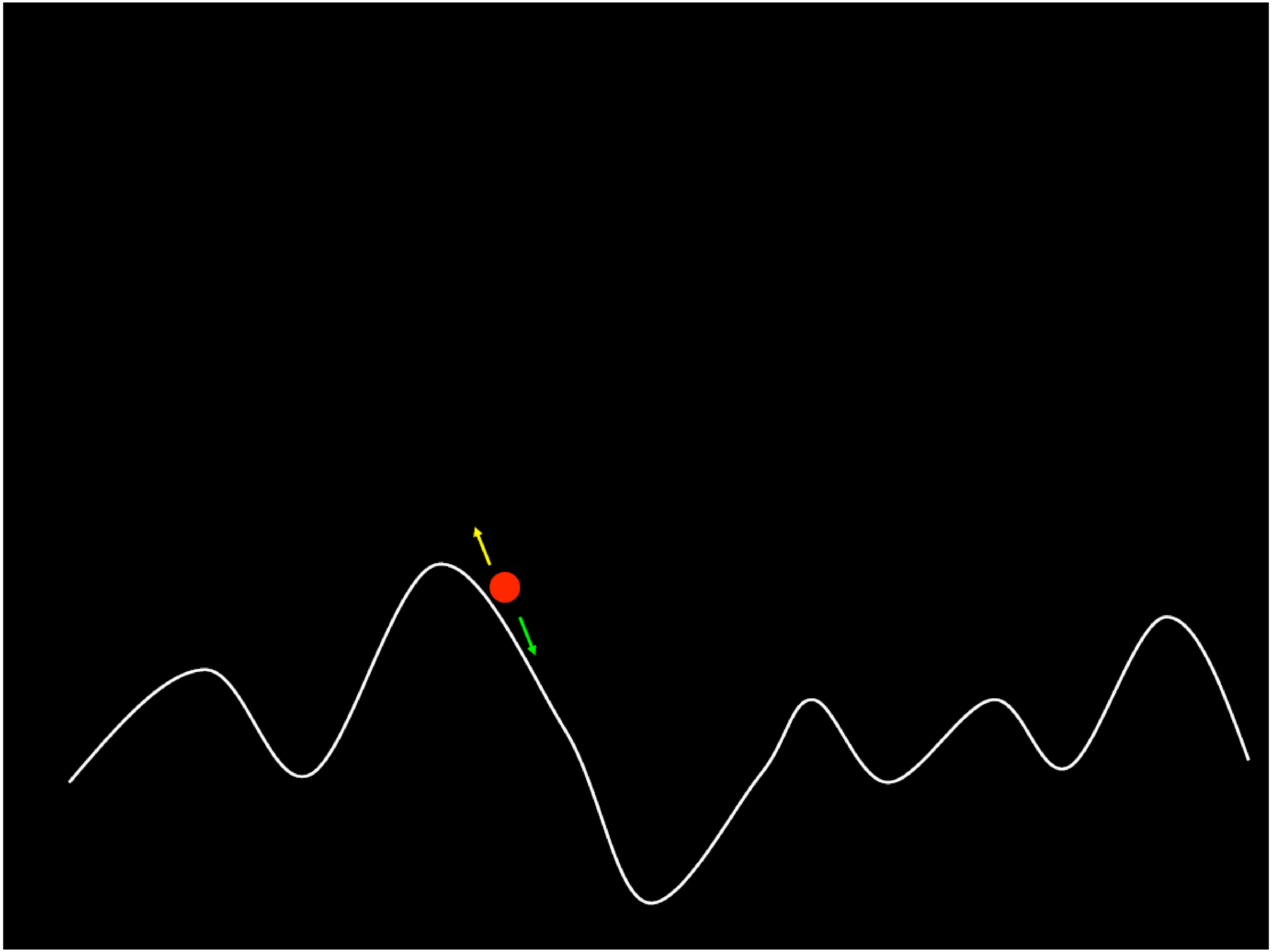
Search as Hill-Climbing

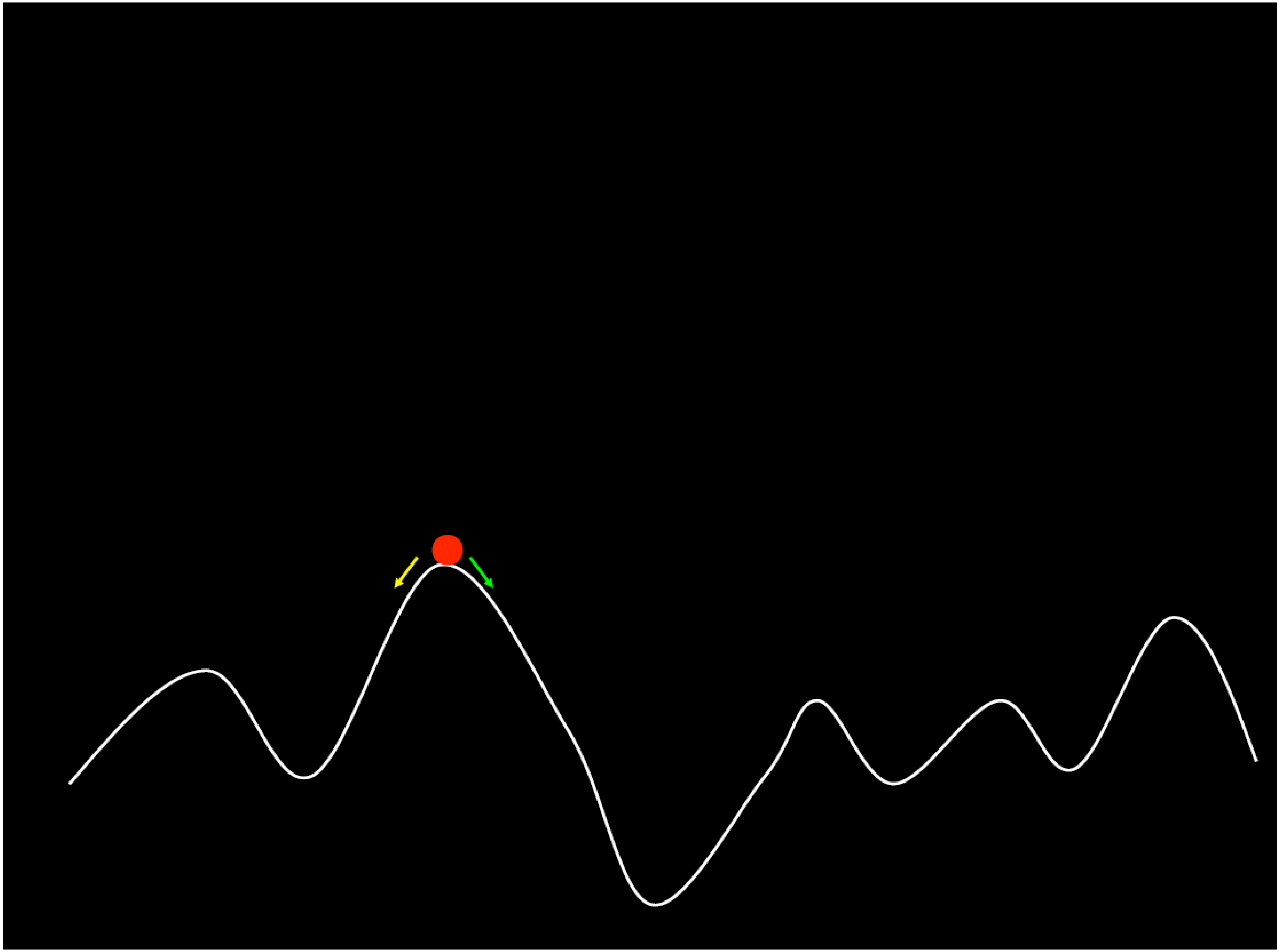
1. Start at some random position
2. Explore your neighborhood to see which direction takes you higher – gives you a *fitter* (better) solution
3. As long as your fitness is increasing, keep exploring; otherwise, stop.



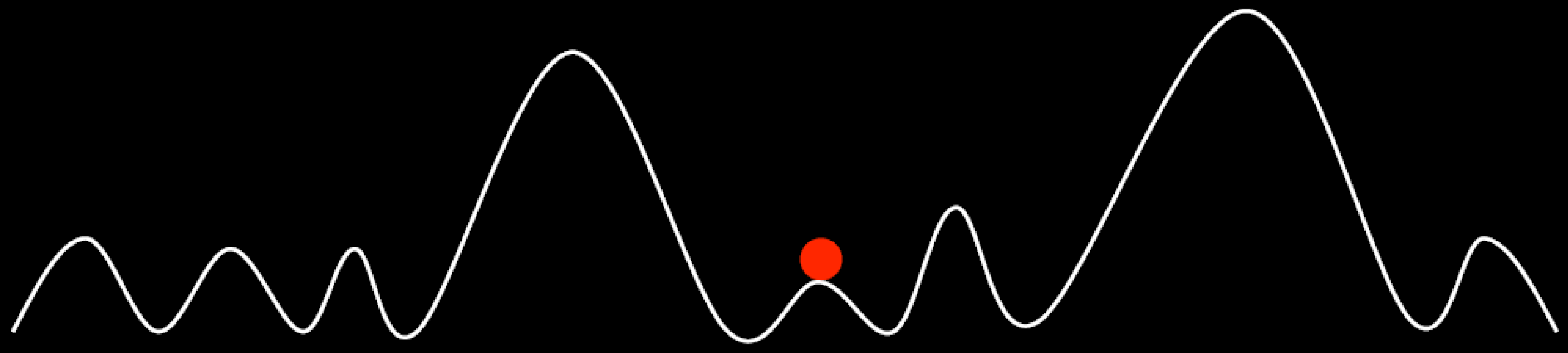




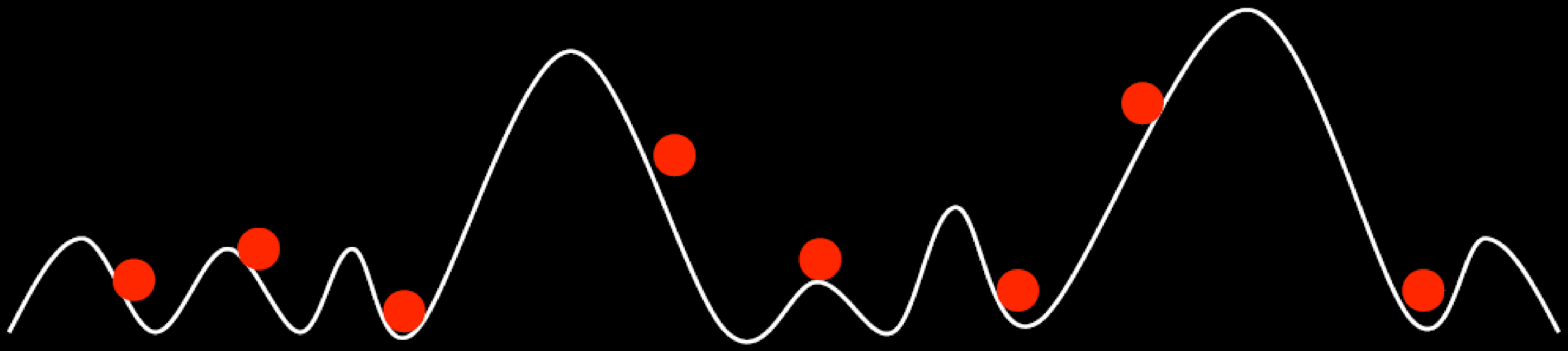




The Problem with Hill-Climbing

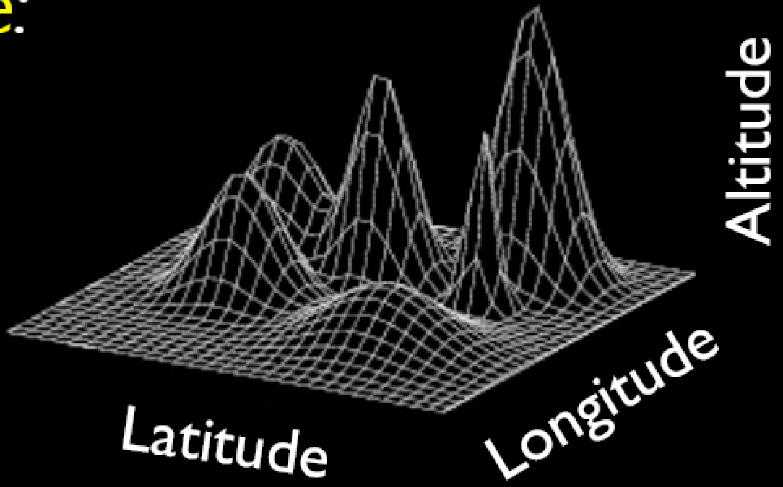


The Genetic Algorithm Insight: A *Population* of Candidate Solutions



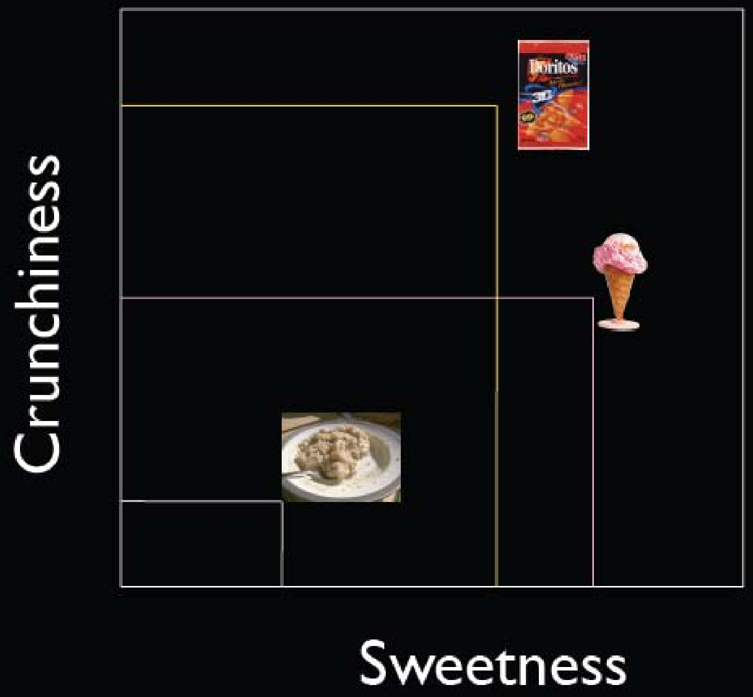
Issue #1: Choice Isn't One-Dimensional

Fitness Landscape:



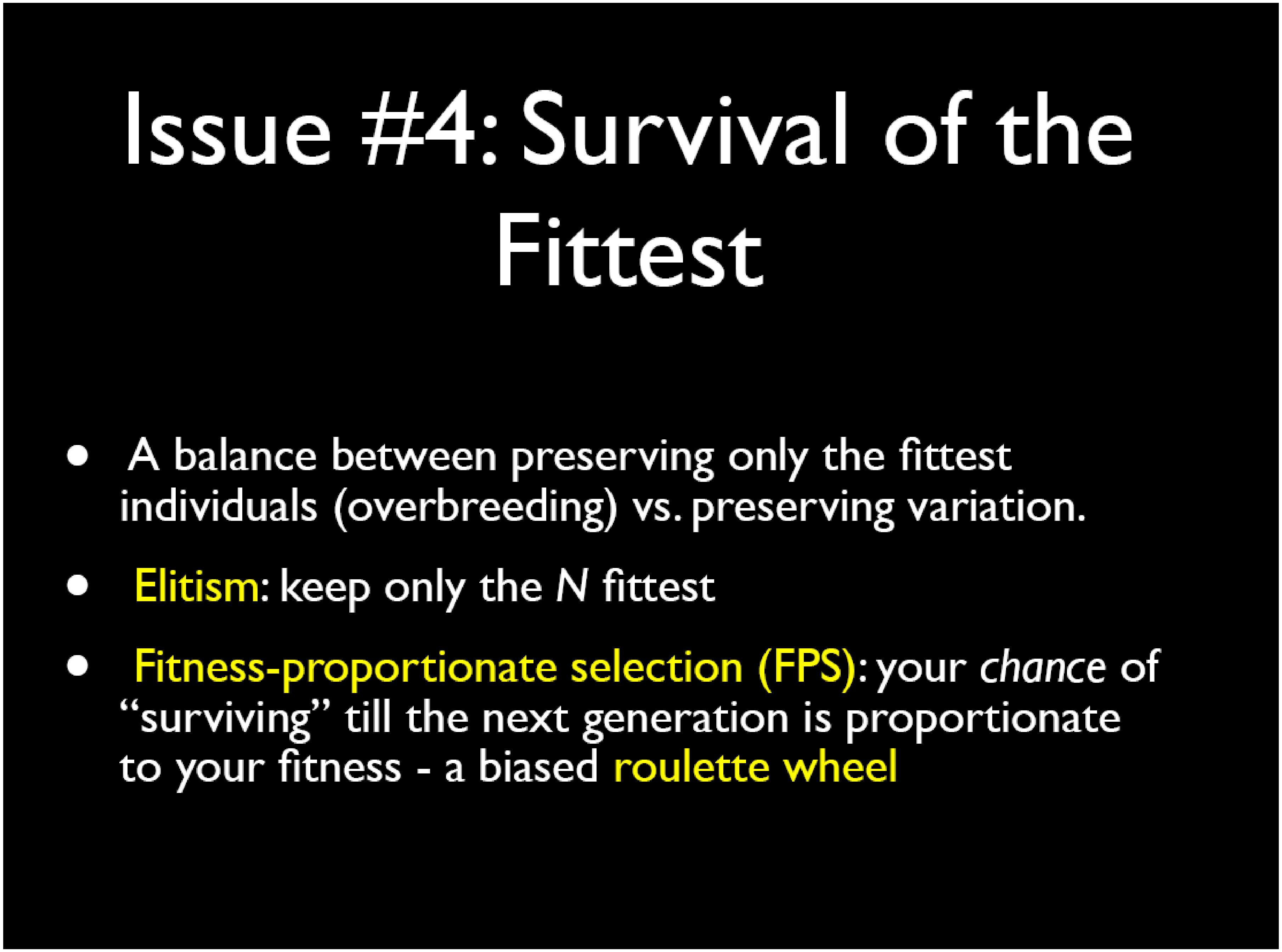
Issue #2: Fitness Isn't One-Dimensional

Multi-Objective Fitness:



Issue #3: Exploring the Landscape

- Introduce a little variation into each member of the population
- "Explore a little bit in each direction" a.k.a. mutation
- Combine components of existing solutions to (we hope) get a better one: recombination a.k.a. crossover a.k.a. sex
- A good representation of our problem will allow us to exploit these operations; a bad one will make that very difficult (as in all AI).

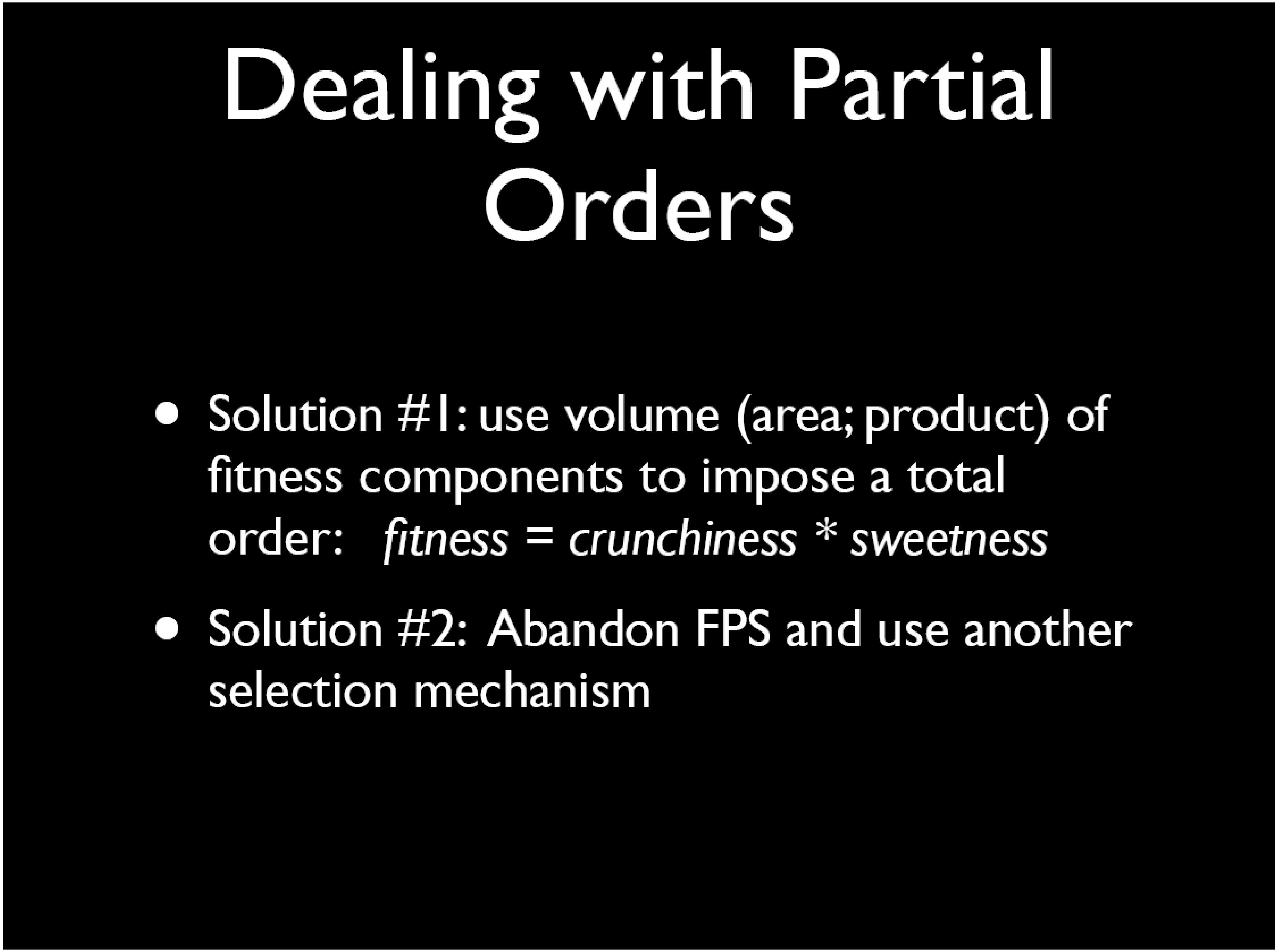


Issue #4: Survival of the Fittest

- A balance between preserving only the fittest individuals (overbreeding) vs. preserving variation.
- Elitism: keep only the *N* fittest
- Fitness-proportionate selection (FPS): your *chance* of “surviving” till the next generation is proportionate to your fitness - a biased roulette wheel

Problems with Multi-Objective Fitness

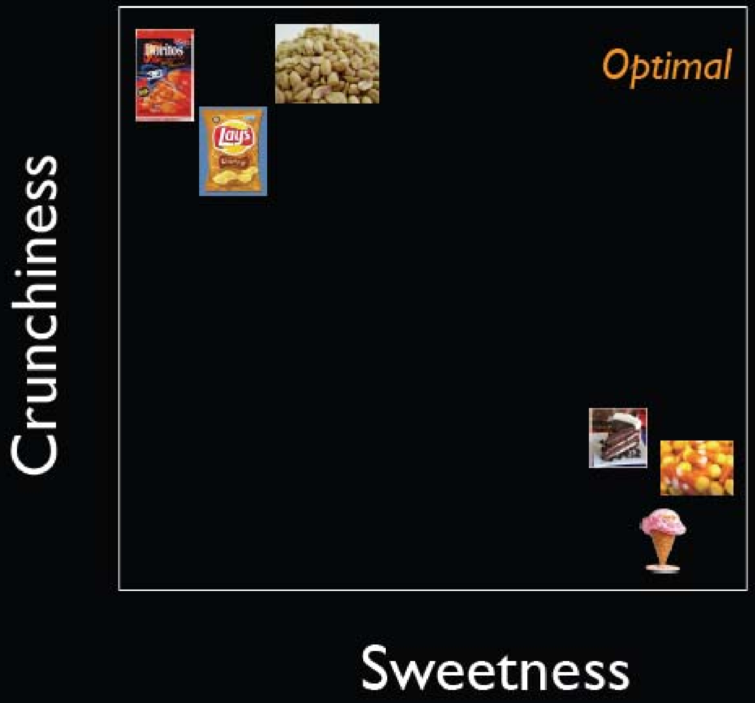
- Fitness-proportionate selection requires a total order on the fitnesses : for each fitness f_1, f_2, either $f_1 < f_2$ or $f_2 < f_1$.
- With multi-objective fitness, we have a partial order: *oatmeal* < *doritos*, *oatmeal* < *ice cream*, but no ordering of *ice cream* and *doritos*.



Dealing with Partial Orders

- Solution #1: use volume (area; product) of fitness components to impose a total order: *fitness* = *crunchiness* * *sweetness*
- Solution #2: Abandon FPS and use another selection mechanism

Imposing a Total Order May Lead to Overspecialization

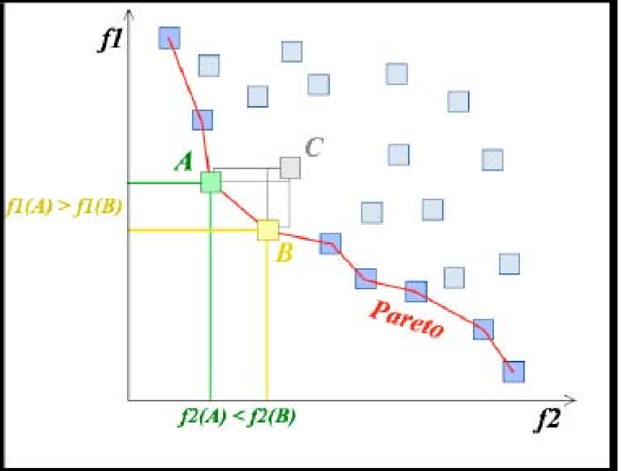


Selection by Nondominated Sorting

- We say that solution *A* dominates solution *B* if every component of A's fitness is greater than the corresponding component of B's fitness.
- This places each solution in a rank, according to how many other solutions dominate it.
- Then we can sort solutions by their rank.

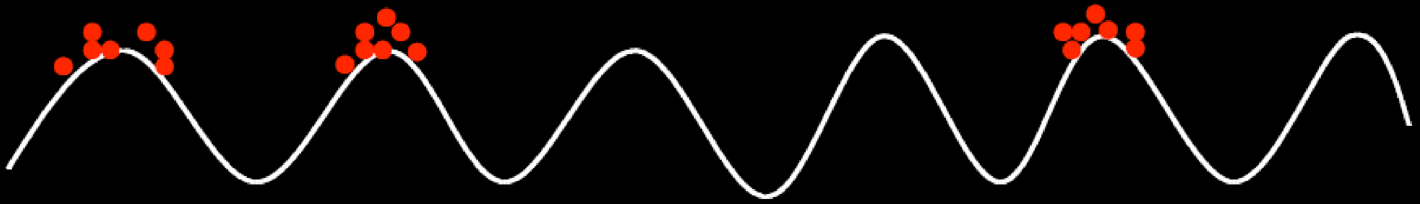
A Little Economics

- **Pareto optimum:** Given a set of alternative allocations of, say, goods or income for a set of individuals, a movement from one allocation to another that can make at least one individual better off without making any other individual worse off is called a Pareto improvement. An allocation is Pareto efficient or Pareto optimal when no further Pareto improvements can be made. [Wikipedia]
- **Pareto front:** set of pareto-optimal allocations – *i.e.*, solutions not dominated by any others
- Similar to...



Here, less is better!

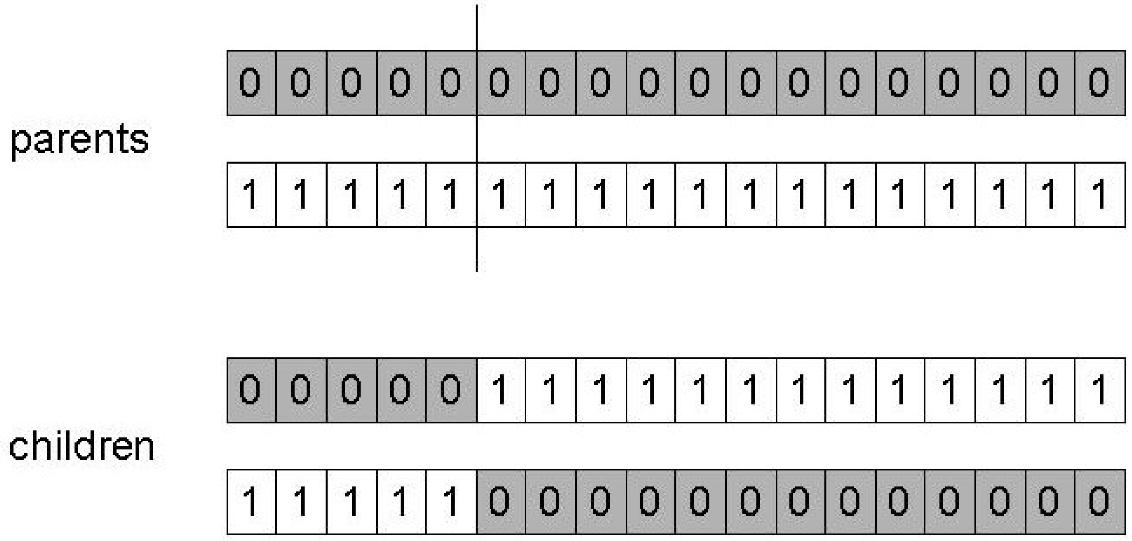
Premature Convergence



- Crowding methods (DeJong 1975; Mahfoud 1992; Goldberg 1996) attempt to slow down convergence by choosing a subset of pop. and using it to replace a the member most similar to it.
- Tournament selection picks a set of individuals at random, and then does selection within that set.
- Fitness sharing (Holland 1975; Goldberg & Richardson 1987) penalizes individuals that are too close together.

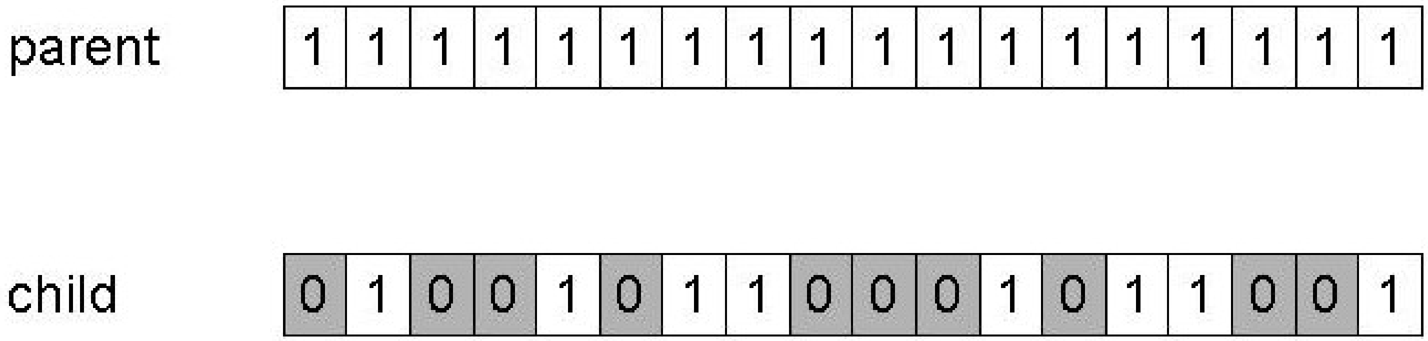
SGA operators: 1-point crossover

- Choose a random point on the two parents
- Split parents at this crossover point
- Create children by exchanging tails



SGA operators: mutation

- Alter each gene independently with a probability p_m
- p_m is called the mutation rate
  - Typically between 1/pop_size and 1/ chromosome_length



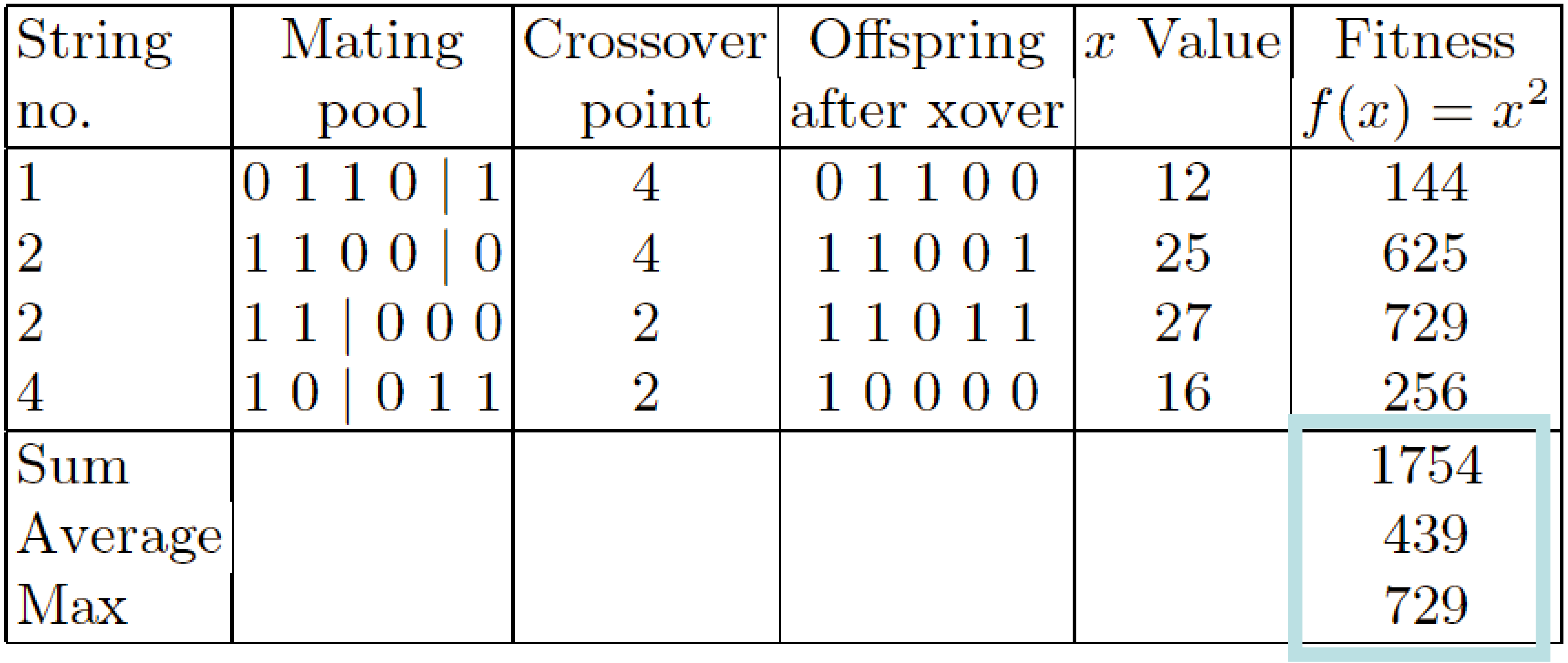
An example after Goldberg '89 (1)

- Simple problem: max x^2 over $\{0,1,\ldots,31\}$
- GA approach:
  - Representation: binary code, e.g. 01101 $\leftrightarrow$ 13
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation
- We show one generational cycle done by hand

x^2 example: selection

String no.	Initial population	x Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

X^2 example: crossover



String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 | 1	4	0 1 1 0 0	12	144
2	1 1 0 0 | 0	4	1 1 0 0 1	25	625
2	1 1 | 0 0 0	2	1 1 0 1 1	27	729
4	1 0 | 0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

X^2 example: mutation

String no.	Offspring after xover	Offspring after mutation	x Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	$\boxed{1}$ 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 $\boxed{1}$ 0 0	18	324
Sum				2354
Average				588.5
Max				729

Crossover OR mutation?

- Decade long debate: which one is better / necessary / main-background
- Answer (at least, rather wide agreement):
  - it depends on the problem, but
  - in general, it is good to have both
  - both have another role
  - mutation-only-EA is possible, xover-only-EA would not work

Crossover OR mutation? (cont'd)

Exploration: Discovering promising areas in the search space, i.e. gaining information on the problem

Exploitation: Optimising within a promising area, i.e. using information

There is co-operation AND competition between them

- Crossover is explorative, it makes a *big* jump to an area somewhere "in between" two (parent) areas
- Mutation is exploitative, it creates random *small* diversions, thereby staying near (in the area of) the parent

Crossover OR mutation? (cont'd)

- Only crossover can combine information from two parents
- Only mutation can introduce new information (alleles)
- Crossover does not change the allele frequencies of the population (thought experiment: 50% 0's on first bit in the population, ?% after performing *n* crossovers)
- To hit the optimum you often need a 'lucky' mutation

Royal Road for Genetic Algorithms

- Question is how fitness landscapes are best explored by GAs
- What are the effects of landscape features to GA performance

- Define a set of landscape features
- Construct classes of landscapes that contain these features in varying degrees
- Study these features in detail

Properties of fitness landscapes

- Deception
- Sampling error
- Number of local optima

- Are these all relevant features of fitness landscapes with respect of GA performance??

- Consider the following deceptive order – 3 function
    - For deception to take place order-1 and order-2 schema redirect the cases of higher fitness towards a low fitness individual, where schemas are measured genotypically.
    - Let the global optimum be 111; the global minimum be 000.
    - Lower order schema are now ordered to satisfy the following relationships to achieve deception,

F(0**) > f(1**)	F(00*) > f(11*), f(01*), f(10*)
F(*0*) > f(*1*)	F(0*0) > f(1*1), f(0*1), f(1*0)
F(**0) > f(**1)	F(*00) > f(*11), f(*01), f(*10)

- This might lead to the following specific fitness values for a deceptive function,

Deceptive Function 1	f(000) = 28	f(001) = 26
	f(010) = 22	f(100) = 14
	f(110) = 0	f(011) = 0
	f(101) = 0	f(111) = 30

Again: schema processing

- GAs search implicitly a space of patterns
- Space of patterns could be thought as hyperplanes $\{0,1\}^l$ (**schemas)**
- Schemas are defined of alphabet {0,1,*}
- Schema Therorem states that above average fitness schemas will receive an exponentially increasing number of samples
- Schema theorem doesn’t state how new schemas are discovered

Again: schema processing

- Building block hypothesis states that new schemas are discovered via crossover
- The actual discovery processes are hard to understand

- But first we need to understand more about landscape features of Gas

Landscape features of GA

- **General:** Conflict between the need to explore new regions of the search space vs. the need to exploit the currently most promising directions
- Analysis by landscape features that are more directly connected to the building block hypothesis:

- Degree of hierarchically structured schemas
- Degree of “stepping stones” between low order and high order schemas
- Degree of isolation of fit schemas
- Presence or absence of conflicts among fit schemas

Describing the landscapes

- Fitness functions F:{0,1}^{l} -> R

$$F(x) = \sum_{s \in S} c_s \sigma_s(x)$$

- Construct “royal roads” for the GA to follow to the global optimum
- can be hierarchically clustered
- Should trick bit-wise mutation techniques used e.g. by hill-climbing methods ..

Royal Road function 1

$$
\begin{aligned}
s_1 &= 11111111**; \; c_1 = 8\\
s_2 &= ********11111111**; \; c_2 = 8\\
s_3 &= ****************11111111**; \; c_3 = 8\\
s_4 &= ************************11111111********************************; \; c_4 = 8\\
s_5 &= ********************************11111111************************; \; c_5 = 8\\
s_6 &= **11111111****************; \; c_6 = 8\\
s_7 &= **11111111********; \; c_7 = 8\\
s_8 &= **11111111; \; c_8 = 8\\
s_9 &= 1111111111111111**; \; c_9 = 16\\
s_{10} &= ****************1111111111111111********************************; \; c_{10} = 16\\
s_{11} &= ********************************1111111111111111****************; \; c_{11} = 16\\
s_{12} &= **1111111111111111; \; c_{12} = 16\\
s_{13} &= 11111111111111111111111111111111********************************; \; c_{13} = 32\\
s_{14} &= ********************************11111111111111111111111111111111; \; c_{14} = 32\\
s_{15} &= 1111111111111111111111111111111111111111111111111111111111111111; \; c_{15} = 64
\end{aligned}
$$

Royal Road function 2

Isolated high-fitness regions

$$F(x) = 5\sigma_{**11}(x) - 16\sigma_{*111}(x) + 5\sigma_{11**}(x) - 16\sigma_{111*}(x) + 31\sigma_{1111}(x).$$

With highest value 9 for point X=1111 and average fitness u(s) of five schemas

$$u(**11) = 2$$
$$u(*111) = -1$$
$$u(11**) = 2$$
$$u(111*) = -1$$
$$u(1111) = 9.$$

Royal Road function

- In such a feature, the regions of high fitness are isolated from supporting (lower-order schemas)
- Hill-climbing get stuck in intermediate regions (**11) and (11**)
- Slow at crossing the intervening desserts of lower fitness (*111) and (111*)

- *“partially deceptive functions”*

Royal Road function 3

Multiple conflicting solutions

Consider a function with two equal peaks:

$$F(x) = (x-(1/2))^2$$

Which has two optima, 0 and 1.

There is a risk, that the GA converges on One by exploiting random fluctuations.

There is even a higher risk that crossover produces hybrids from both solutions

Perfomance on Royal Road functions

- All desired schemas are known in advance
- Tracing of individual schemas

- Degree of “regality” of the path to the optimum can be varied
- E.g. change the number of levels of schemas (schemas of order 8,16,32,64 are for levels) to three levels

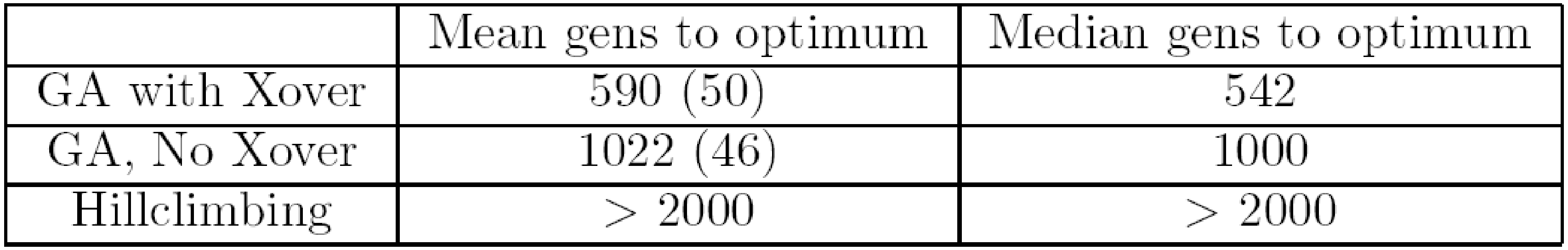
To what extent does crossover help the GA finding highly fit schemas

Bottlenecks ?

- Waiting time for desirable schemas to be discovered
- Role of intermediate levels in the hierarchy

To what extent does crossover help the GA finding highly fit schemas

- Competitive setup of hill-climbing and GAs with and without crossover



	Mean gens to optimum	Median gens to optimum
GA with Xover	590 (50)	542
GA, No Xover	1022 (46)	1000
Hillclimbing	> 2000	> 2000

To what extent does crossover help the GA finding highly fit schemas

There are two stages in the discovery process of a given schema

- The time for the schemas lower-order components to appear in the population
- The time for two instances to cross over in the right way to create the desired schema

The simple GA

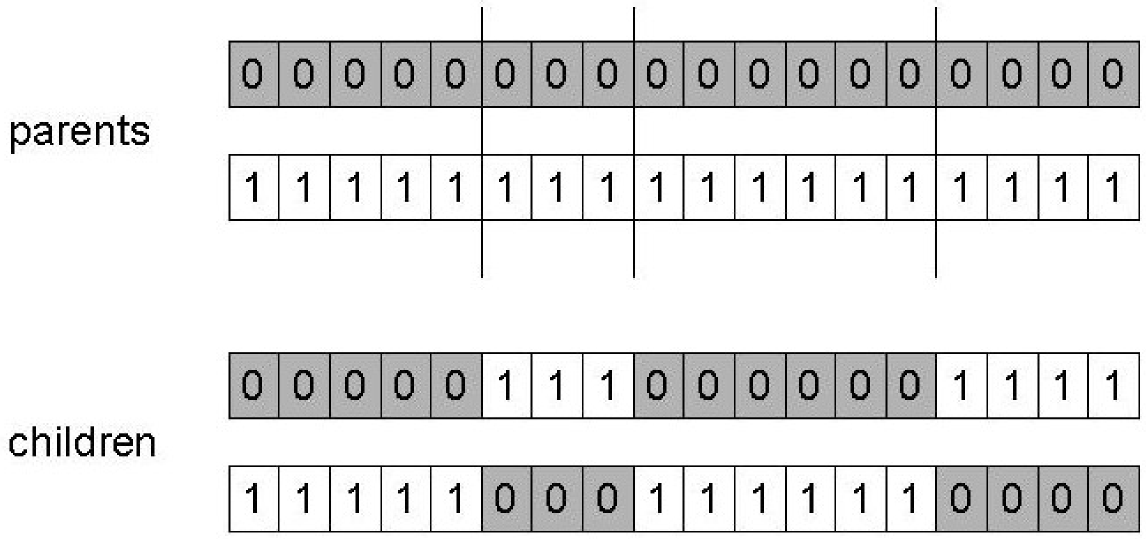
- Has been subject of many (early) studies
  - still often used as benchmark for novel GAs
- Shows many shortcomings, e.g.
  - Representation is too restrictive
  - Mutation & crossovers only applicable for bit-string & integer representations
  - Selection mechanism sensitive for converging populations with close fitness values
  - Generational population model (step 5 in SGA repr. cycle) can be improved with explicit survivor selection

Alternative Crossover Operators

- Performance with 1 Point Crossover depends on the order that variables occur in the representation
  - more likely to keep together genes that are near each other
  - Can never keep together genes from opposite ends of string
  - This is known as *Positional Bias*
  - Can be exploited if we know about the structure of our problem, but this is not usually the case

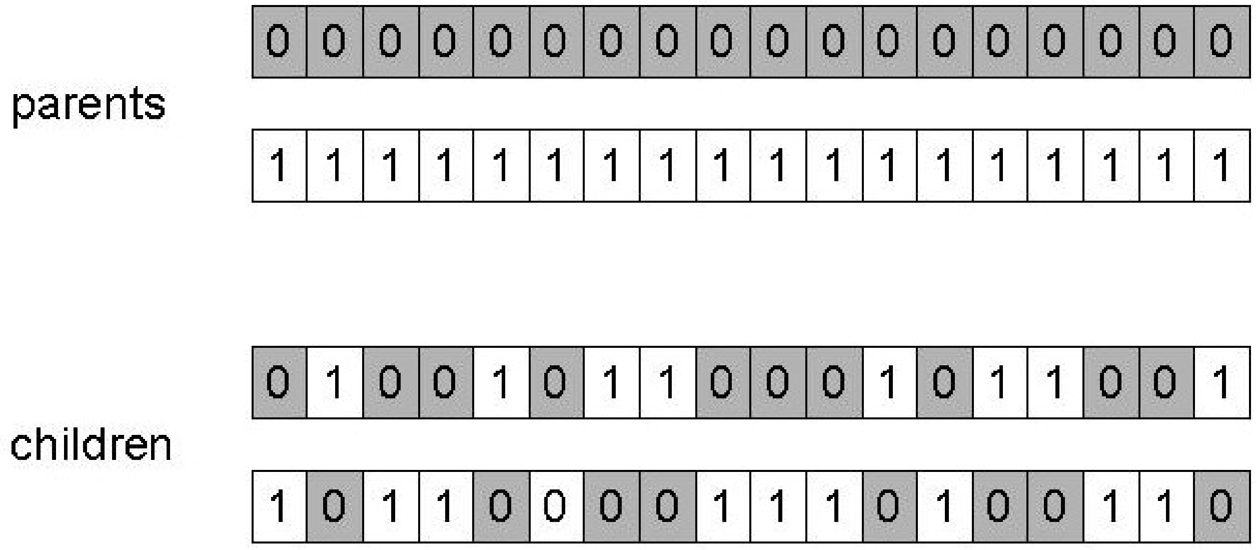
n-point crossover

- Choose n random crossover points
- Split along those points
- Glue parts, alternating between parents
- Generalisation of 1 point (still some positional bias)



Uniform crossover

- Assign 'heads' to one parent, 'tails' to the other
- Flip a coin for each gene of the first child
- Make an inverse copy of the gene for the second child
- Inheritance is independent of position



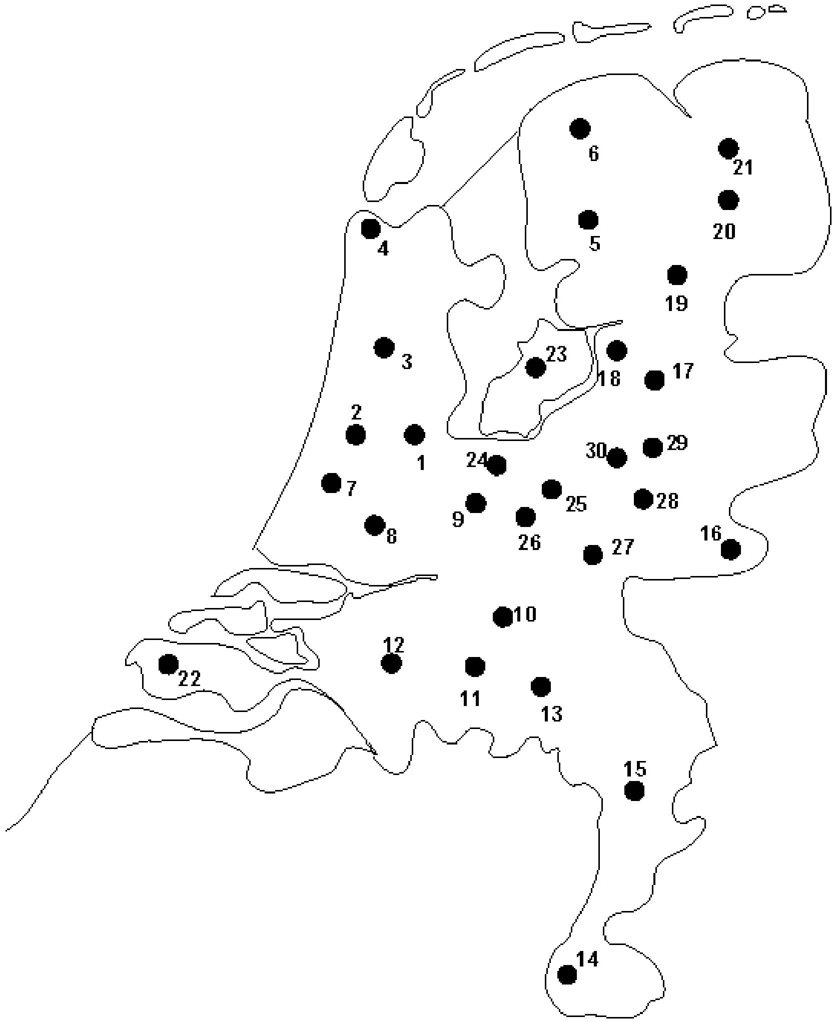
Permutation Representations

- Ordering/sequencing problems form a special type
- Task is (or can be solved by) arranging some objects in a certain order
  - Example: sort algorithm: important thing is which elements occur before others (<u>order</u>)
  - Example: Travelling Salesman Problem (TSP) : important thing is which elements occur next to each other (<u>adjacency</u>)
- These problems are generally expressed as a permutation:
  - if there are *n* variables then the representation is as a list of *n* integers, each of which occurs exactly once

Permutation representation: TSP example

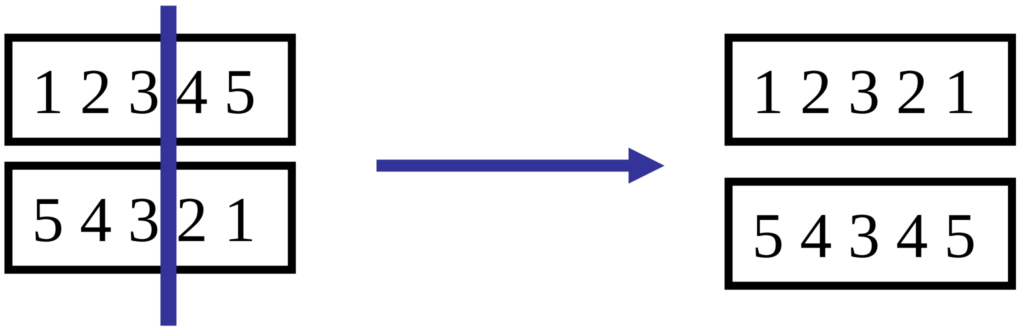
- Problem:
  - Given n cities
  - Find a complete tour with minimal length
- Encoding:
  - Label the cities 1, 2, … , *n*
  - One complete tour is one permutation (e.g. for n =4 [1,2,3,4], [3,4,2,1] are OK)
- Search space is BIG:

  for 30 cities there are 30! ≈ 10^{32} possible tours



Crossover operators for permutations

- "Normal" crossover operators will often lead to inadmissible solutions



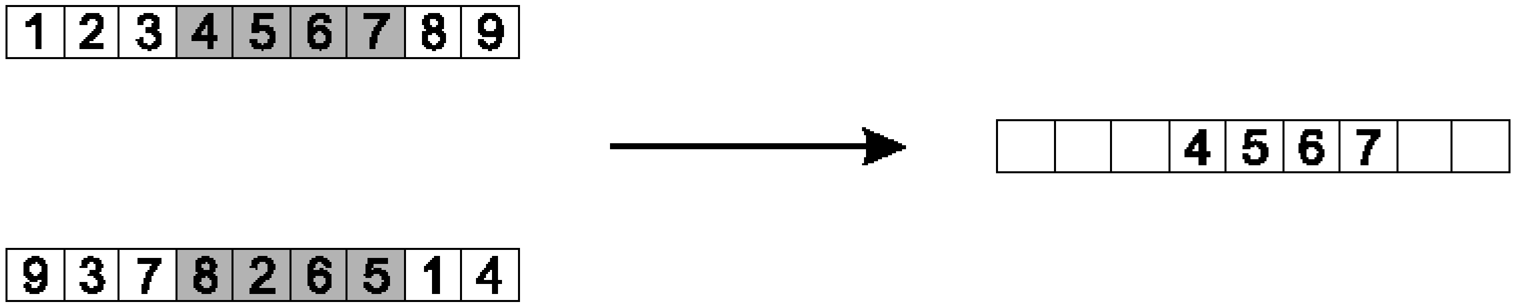
Many specialised operators have been devised which focus on combining order or adjacency information from the two parents

Order 1 crossover

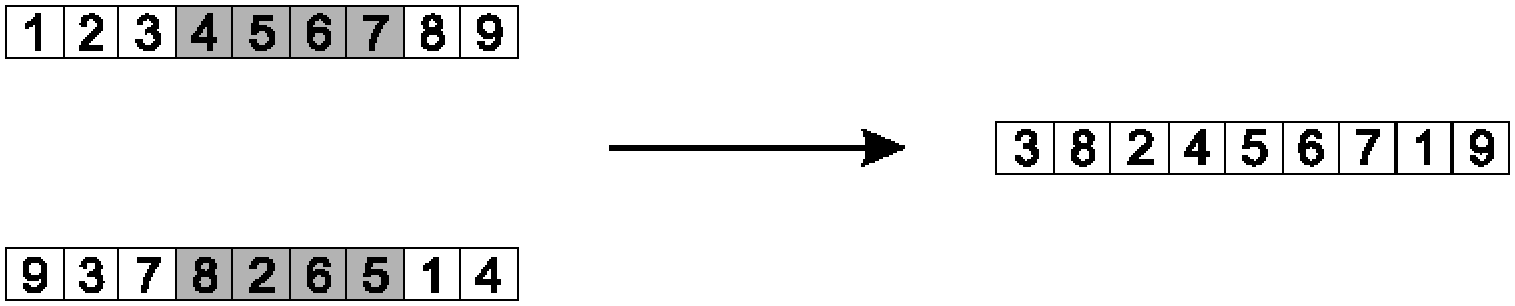
- Idea is to preserve relative order that elements occur
- Informal procedure:
  1. Choose an arbitrary part from the first parent
  2. Copy this part to the first child
  3. Copy the numbers that are not in the first part, to the first child:
     - starting right from cut point of the copied part,
     - using the **order** of the second parent
     - and wrapping around at the end
  4. Analogous for the second child, with parent roles reversed

Order 1 crossover example

- Copy randomly selected set from first parent



- Copy rest from second parent in order 1,9,3,8,2



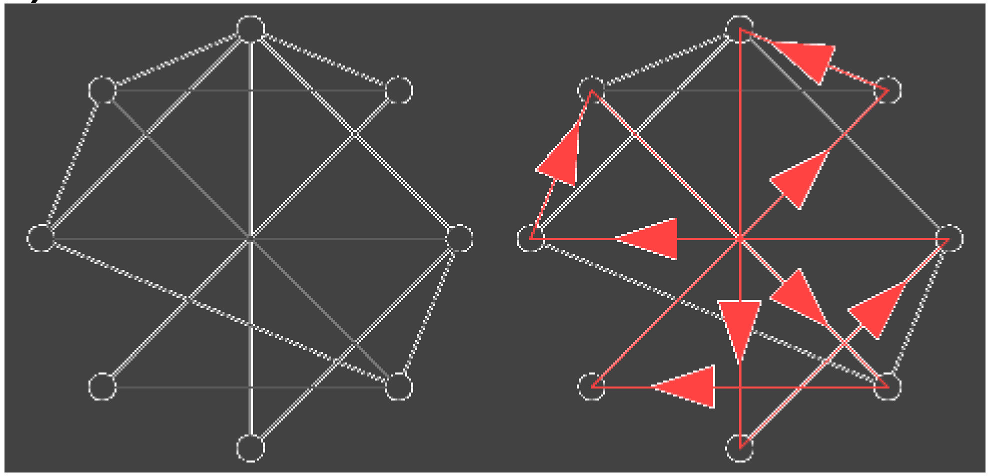
Multiparent recombination

- Recall that we are not constricted by the practicalities of nature
- Noting that mutation uses 1 parent, and “traditional” crossover 2, the extension to *a*>2 is natural to examine
- Been around since 1960s, still rare but studies indicate useful
- Three main types:
  - Based on allele frequencies, e.g., p-sexual voting generalising uniform crossover
  - Based on segmentation and recombination of the parents, e.g., diagonal crossover generalising n-point crossover
  - Based on numerical operations on real-valued alleles, e.g., center of mass crossover, generalising arithmetic recombination operators

Simulated Annealing

TSP

- There are certain optimization problems that become unmanageable using combinatorial methods as the number of objects becomes large.
- A typical example is the traveling salesman problem (TSP).



MSA as an optimization problem

- Sum of Pairs
  - Formally, for column a multiple alignment of *N* sequences with a length M, the total SP score is
- Minimum Entropy
  - The basic idea of the minimum entropy method is to try to minimize the total entropy ME for all columns.

$$SP = \sum_i S(m_i) = \sum_i \sum_j \sum_k s(m_i^j, m_i^k) = \sum_j^N \sum_k^N \sum_i^M s(m_i^j, m_i^k)$$

$$ME = -\sum_i \sum_a p_{ia} \log p_{ia} ; p_{ib} = \frac{c_{ia}}{\sum_b c_{ib}}$$

Method of Steepest Descent

- An algorithm for finding the (nearest local) minimum of a function.
- Steepest descent, also called the gradient descent method, starts at a point p_0 and, as many times as needed, moves from p_i to p_{i+1} by minimizing along the line extending from p_i in the direction of, the local downhill gradient.

Conjugate gradient method

- The conjugate gradient method is an algorithm for finding the nearest local minimum of a function of variables which presupposes that the gradient of the function can be computed. It uses conjugate directions instead of the local gradient for going downhill.
- If the vicinity of the minimum has the shape of a long, narrow valley, the minimum is reached in far fewer steps than would be the case using the method of steepest descent.
- Gradient methods “stuck” in local minima.

Simulated Annealing

- For problems, like TSP, there is a very effective practical algorithm called simulated annealing (thus named because it mimics the process undergone by misplaced atoms in a metal when its heated and then slowly cooled).
- While this technique is unlikely to find the *optimum* solution, it can often find a very good solution, even in the presence of noisy data.

What is simulated annealing?

- **Simulated annealing** (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space.
- Simulated annealing is a generalization of a Monte Carlo method for examining the equations of state and frozen states of n-body systems [Metropolis et al. 1953].

Metropolis

Repetition

Metropolis

- Simulated annealing improves gradient method strategy through the introduction of two tricks. The first is the so-called "Metropolis algorithm" (Metropolis *et al.* 1953), in which some trades that do not improve the score are accepted when they serve to allow the solver to "explore" more of the possible space of solutions. Such "bad" trades are allowed using the criterion that $e^{-\Delta D/T} > R(0,1)$,

Metropolis

- where ΔD is the change of score implied by the trade (negative for a "good" trade; positive for a "bad" trade), T is a "synthetic temperature," and R(0,1) is a random number in the interval [0,1].
- D is called a "cost function," and corresponds to the free energy in the case of annealing a metal.
- If T is large, many "bad" trades are accepted, and a large part of solution space is accessed. Objects to be traded are generally chosen randomly, though more sophisticated techniques can be used.

Metropolis

- The second trick is, again by analogy with annealing of a metal, to lower the "temperature."
- After making many trades and observing that the cost function declines only slowly, one lowers the temperature, and thus limits the size of allowed "bad" trades.
- After lowering the temperature several times to a low value, one may then "quench" the process by accepting only "good" trades in order to find the local minimum of the cost function.
- There are various "annealing schedules" for lowering the temperature, but the results are generally not very sensitive to the details.

What is simulated annealing?

- The concept is based on the manner in which liquids freeze or metals recrystallize in the process of annealing.
- In an annealing process a melt, initially at high temperature and disordered, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium.
- As cooling proceeds, the system becomes more ordered and approaches a "frozen" ground state at T=0.
- If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become quenched forming defects or freezing out in metastable states (ie. trapped in a local minimum energy state).

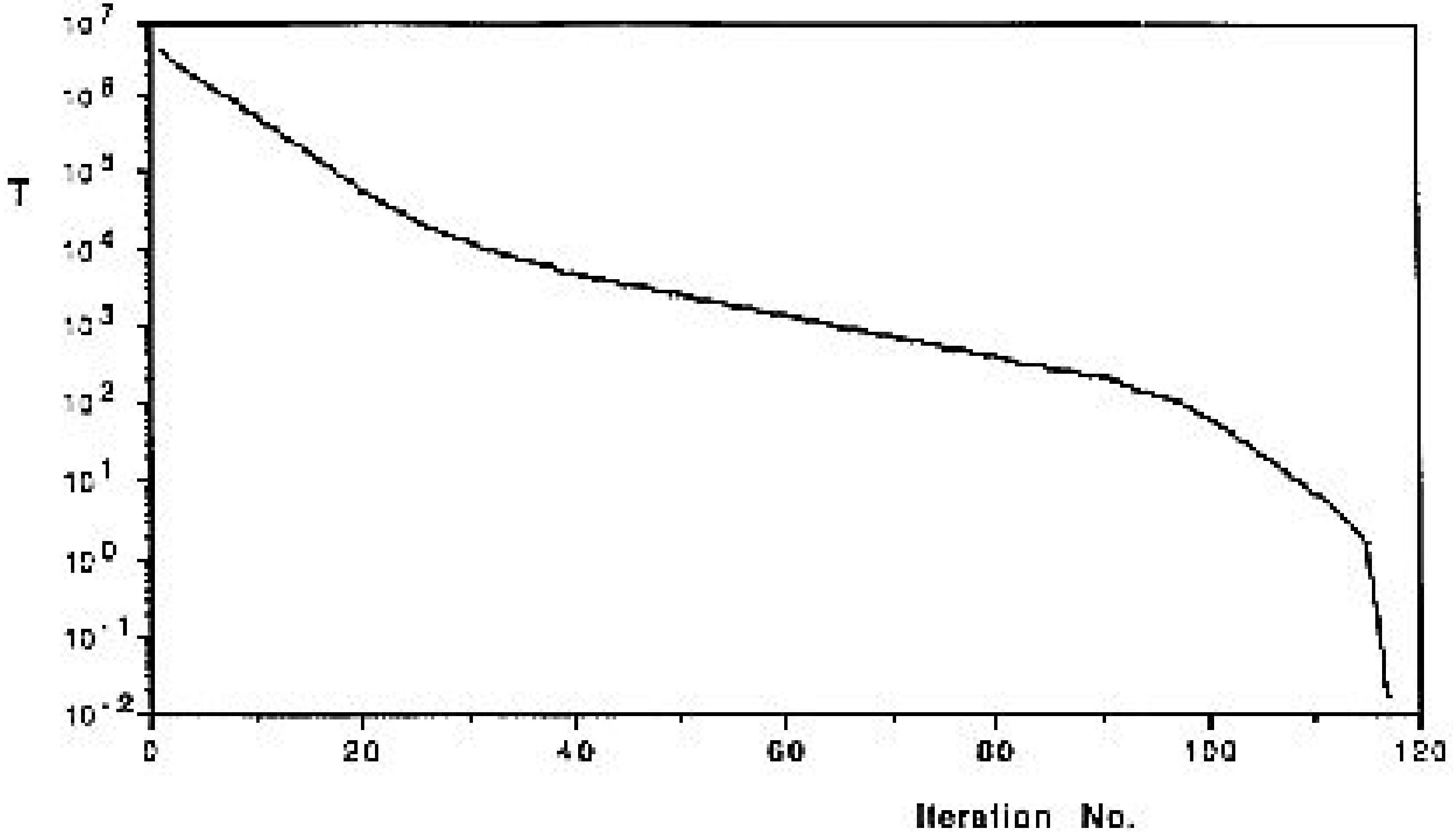
What is simulated annealing?

- Simulated annealing is a technique, which was developed to help solve large combinatorial optimization problems. It is based on probabilistic methods that avoid being stuck at local (non-global) minima. It has proven to be a simple but powerful method for large-scale combinatorial optimization.
- For practical purposes, simulated annealing has solved the famous **traveling salesman** problem: find the shortest of N! paths connecting N cities. Simulated annealing finds a very good approximation to the shortest path out of the huge number of all possible paths.
- Annealing is nature's trick to find extrema in very complicated situations. Simulated annealing mimics on a computer the natural process by which crystal lattices of glass or metal relax when heated.

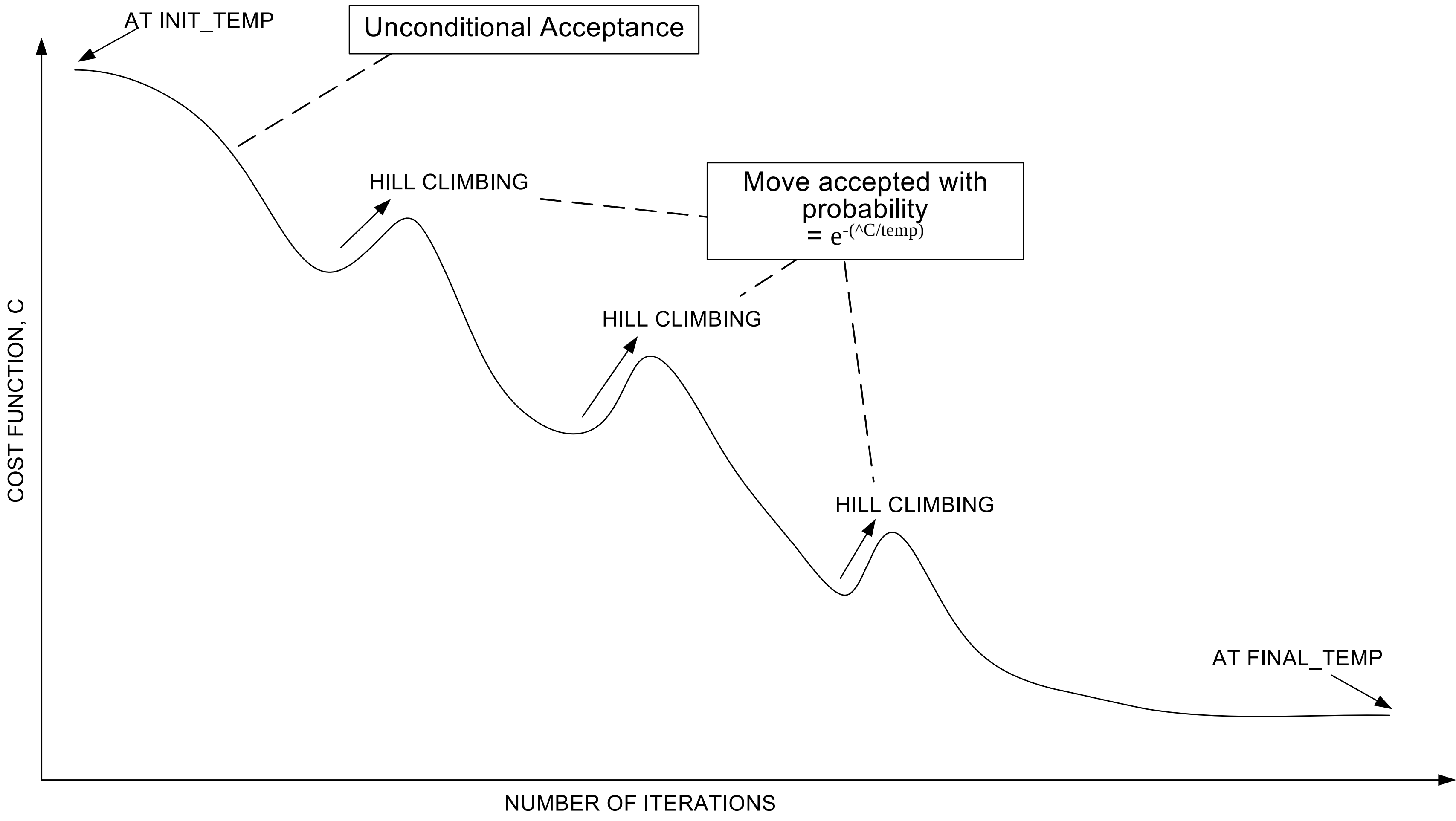
What is simulated annealing?

- The molecules of hot glass or metal are free to move about.
- Temperature is an average of the thermal energy in each molecule of an object.
- If the temperature drops quickly, these molecules solidify into a complex structure.
- However, if the temperature drops slowly, they form a highly ordered crystal.
- The molecules of a crystal solidify into a minimal energy state.

Cooling schedule



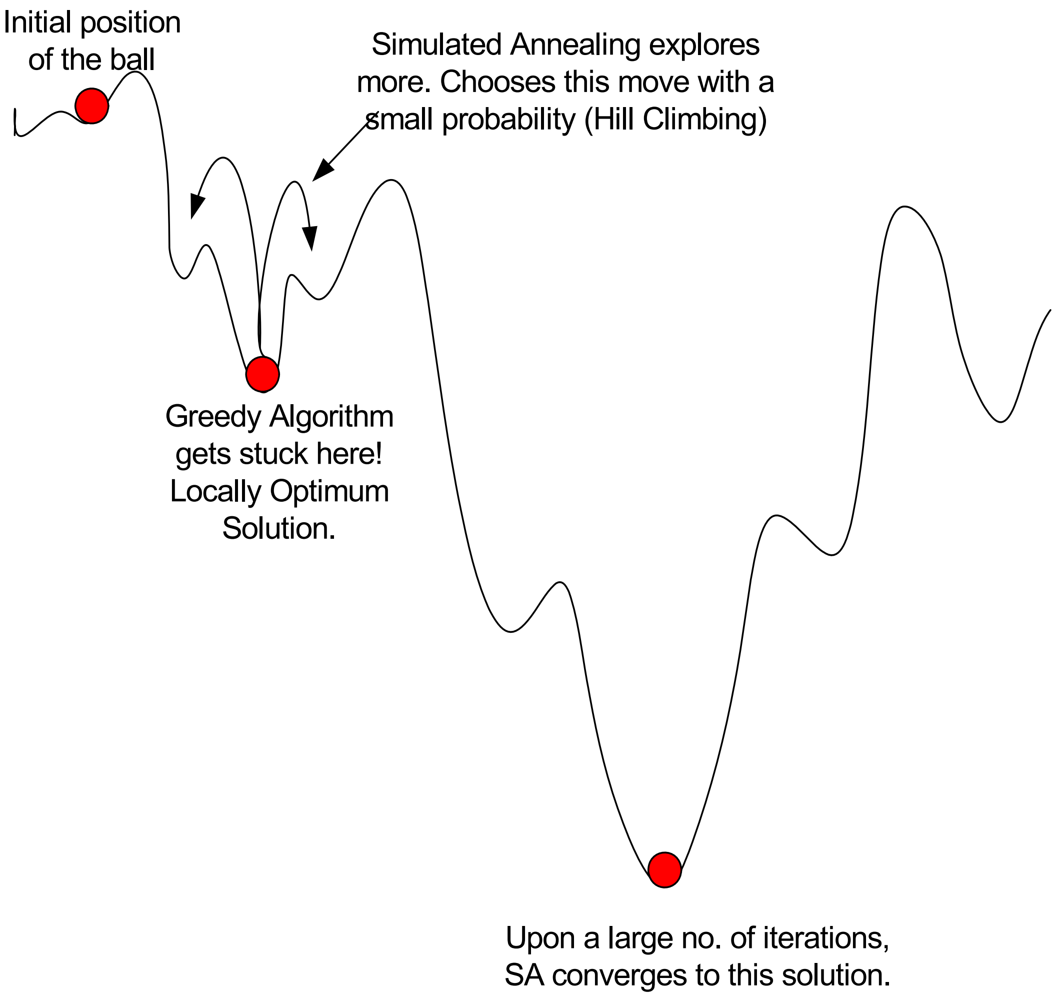
Convergence of simulated annealing



Ball on terrain example – Simulated Annealing vs Greedy Algorithms

The ball is initially placed at a random position on the terrain. From the current position, the ball should be fired such that it can only move one step left or right.What algorithm should we follow for the ball to finally settle at the lowest point on the terrain?

Ball on terrain example – SA vs Greedy Algorithms


Initial position of the ball
Simulated Annealing explores more. Chooses this move with a small probability (Hill Climbing)
Greedy Algorithm gets stuck here! Locally Optimum Solution.
Upon a large no. of iterations, SA converges to this solution.

The algorithm

- In the simulated annealing (SA) method, each point *s* of the search space is compared to a state of some physical system, and the function *E*(*s*) to be minimized is interpreted as the internal energy of the system in that state.
- Therefore the goal is to bring the system, from an arbitrary *initial state*, to a state with the minimum possible energy.

The algorithm

- In the simulated annealing algorithm, an objective function to be minimized is defined.
- In TSP it will be the total path length through a set of points. The distance between each pair of points is equivalent to the "energy" of a molecule.
- Then, "temperature" is the average of these lengths. Starting from an initial point, the algorithm swaps a pair of points and the total "energy" of the path is calculated.
- Any downhill step is accepted and the process repeats. An uphill step may be accepted.
- Thus, the algorithm can escape from local minima.
- This uphill decision is made by the Metropolis criteria.
- As the optimization process proceeds, the algorithm closes in on the global minimum.

The basic iteration

- At each step, the SA heuristic considers some neighbor *s'* of the current state *s*, and probabilistically decides between moving the system to state *s'* or staying put in state *s*.
- The probabilities are chosen so that the system ultimately tends to move to states of lower energy.
- Typically this step is repeated until the system reaches a state which is good enough for the application, or until a given computation budget has been exhausted.

The neighbors of a state

- The neighbors of each state are specified by the user, usually in an application-specific way.
- For example, in the traveling salesman problem, each state is typically defined as a particular *tour* (a permutation of the cities to be visited);
- then one could define two tours to be neighbors if and only if one can be converted to the other by interchanging a pair of adjacent cities.

The neighbors of a state

- The neighbor selection method is particularly critical.
- The method may be modeled as a *search graph* — where the states are vertices, and there is an edge from each state to each of its neighbors.
- Roughly speaking, it must be possible to go from the initial state to a "good enough" state by a relatively short path on this graph, and such a path must be as likely as possible to be followed by the SA iteration.

The neighbors of a state

- In practice, one tries to achieve this criterion by using a search graph where the neighbors of *s* are expected to have about the same energy as *s*.
- It is true that reaching the goal can always be done with only *n*-1 general swaps, while it may take as many as *n*(*n*-1)/2 adjacent swaps.
- However, if one were to apply a random general swap to a fairly good solution, one would almost certainly get a large energy increase;
- whereas swapping two adjacent cities is likely to have a smaller effect on the energy.

The algorithm’s parameters

- Since the algorithm makes very few assumptions regarding the objective function, it is quite robust. The degree of robustness can be adjusted by the user using some important parameters:
  - factor - annealing temperature reduction factor
  - ntemps - number of temperature steps to try
  - nlimit - number of trials at each temperature
  - glimit - number of successful trials (or swaps)

Transition probabilities

- The probability of making the transition from the current state s to a candidate new state s' is a function $P(e, e', T)$ of the energies $e = E(s)$ and $e' = E(s')$ of the two states, and of a global time-varying parameter T called the *temperature*.
- One essential requirement for the transition probability P is that it must be nonzero when $e' > e$, meaning that the system may move to the new state even when it is *worse* (has a higher energy) than the current one.
- It is this feature that prevents the method from becoming stuck in a *local minimum* — a state whose energy is far from being minimum, but is still less than that of any neighbor.

Transition probabilities

- On the other hand, when T goes to zero, the probability $P(e, e', T)$ must tend to zero if $e' > e$, and to a positive value if $e' < e$.
- That way, for sufficiently small values of T, the system will increasingly favor moves that go "downhill" (to lower energy values), and avoid those that go "uphill".
- In particular, when T becomes 0, the procedure will reduce to the greedy algorithm — which makes the move if and only if it goes downhill.

Transition probabilities

- The *P* function is usually chosen so that the probability of accepting a move decreases when the difference $e' - e$ increases — that is, small uphill moves are more likely than large ones.
- However, this requirement is not strictly necessary, provided that the above requirements are met.
- Given these properties, the evolution of the state *s* depends crucially on the temperature *T*.
- Roughly speaking, the evolution of *s* is sensitive only to coarser energy variations when *T* is large, and to finer variations when *T* is small.

The annealing schedule

- Another essential feature of the SA method is that the temperature is gradually reduced as the simulation proceeds.
- Initially, *T* is set to a high value (or infinity), and it is decreased at each step according to some *annealing schedule* — which may be specified by the user, but must end with *T*=0 towards the end of the allotted time budget.
- In this way, the system is expected to wander initially towards a broad region of the search space containing good solutions, ignoring small features of the energy function;
- then drift towards low-energy regions that become narrower and narrower; and finally move downhill according to the steepest descent heuristic.

Pseudo-code

- The following pseudo-code implements the simulated annealing heuristic, as described above, starting from state *s0* and continuing to a maximum of *kmax* steps or until a state with energy *emax* or less is found.
- The call neighbour(*s*) should generate a randomly chosen neighbour of a given state s; the call random() should return a random value in the range [0, 1).
- The annealing schedule is defined by the call temp(r), which should yield the temperature to use, given the fraction *r* of the time budget that has been expended so far.

Pseudo-code

```
• s := s0; e := E(s)                          // Initial state, energy.
• k := 0                                      // Energy evaluation count.
• while k < kmax and e > emax                 // While time remains ANDnot
                                              //        good enough:
• sn := neighbour(s)                          // Pick some neighbor.
• en := E(sn)                                 // Compute its energy.
• if random() < P(e, en, temp(k/kmax)) then
                                               // Should we move to it?
• s := sn; e := en                            // Yes, change state.
• k := k + 1                                  // One more evaluation done

• return s                                    // Return current solution
```

Saving the best solution seen

```
s := s0; e := E(s)                          // Initial state, energy.
sb := s; eb := e                            // Initial "best" solution
k := 0                                      // Energy evaluation count.
while k < kmax and e > emax                 // While time remains & not good enough:
    sn := neighbor(s)                       // Pick some neighbor.
    en := E(sn)                             // Compute its energy.
    if en < eb then                         // Is this a new best?
        sb := sn; eb := en                  // Yes, save it.
    if random() < P(e, en, temp(k/kmax)) then
                                            // Should we move to it?
        s := sn; e := en                    // Yes, change state.
    k := k + 1                              // One more evaluation done
return sb                                   // Return the best solution found.
```

Pseudocode for TSP

```
• initialize temperature
• for i := 1...ntemps do
•     temperature := factor * temperature
•     for j := 1...nlimit do
          try change a shift of a random sequence
•         delta := current_cost - trial_cost
•         if delta < 0 then
•             make the swap permanent
•             increment good_swaps
•         else
•             p := random number in range [0...1]
•             m := exp( - delta / temperature )
•             if p < m then                  // Metropolis criterion
•                 make the swap permanent
•                 increment good_swaps
•             end if
•         end if
•         exit when good_swaps > glimit
•     end for
• end for
```

MSA Example

- The MSA without gaps with a SP-score scheme can be used as an example application of simulated annealing.
- Given:
  - window-size W
  - set of N sequences x: $|x_i|>W$ for $1 \le i \le N$
- In this problem, many sequences should be aligned (a position of the window in each sequence should be found or, in other words, a vector of shifts should be found) to maximize the sum of pairwise scores.

Related methods

- Tabu search (TS) is similar to simulated annealing, in that both traverse the solution space by testing neighbors of the current solution. In order to prevent cycling, the TS algorithm maintains a "tabu list" of solutions already seen, and moves to those solutions are suppressed.
- Stochastic hill climbing (SH) runs many hill-climbing searches from random initial locations.

Tabu search

- tabu search: In this general category of meta-heuristics, the essential idea is to 'forbid' search moves to points already visited in the (usually discrete) search space, at least for the upcoming few steps.
- That is, one can temporarily accept new inferior solutions, in order to avoid paths already investigated.
- This approach can lead to exploring new regions of D, with the goal of finding a solution by 'globalized' search.
- Tabu search has traditionally been applied to combinatorial optimization (e.g., scheduling, routing, traveling salesman) problems.