Uninformed search strategies

Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$

b: maximum branching factor of the search tree
d: depth of the optimal solution
m: maximum length of any path in the state space
C*: cost of optimal solution

All search strategies

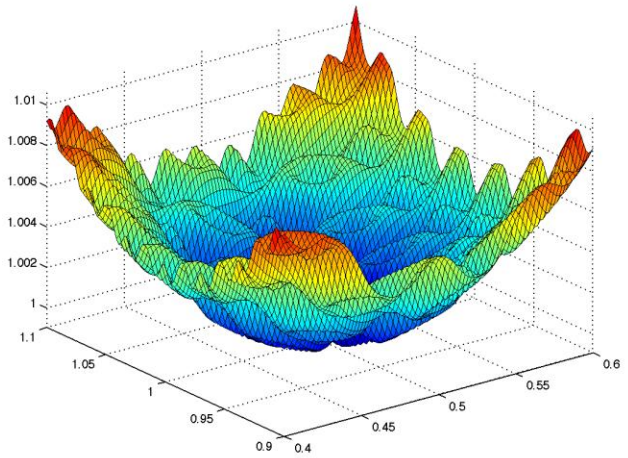
Algorithm	Complete?	Optimal?	Time complexity	Space complexity
BFS	Yes	If all step costs are equal	$O(b^d)$	$O(b^d)$
UCS	Yes	Yes	Number of nodes with $g(n) \leq C^*$	
DFS	No	No	$O(b^m)$	$O(bm)$
IDS	Yes	If all step costs are equal	$O(b^d)$	$O(bd)$
Greedy	No	No	Worst case: $O(b^m)$ Best case: $O(bd)$	
A*	Yes	Yes	Number of nodes with $g(n)+h(n) \leq C^*$	

Why Local Search?

- Blind search and Informed search Algorithms run systematically.
- To achieve the goal, one or more previously explored paths toward the solution need to be stored to find the optimal solution.
- For many problems, the path to the goal is irrelevant. For example, in N-Queens problem, we don’t need to care about the final configuration of the queens as well as in which order the queens are added.

Local search algorithms

- Some types of search problems can be formulated in terms of **optimization**
  - We don't have a start state, don't care about the path to a solution
  - We have an **objective function** that tells us about the quality of a possible solution, and we want to find a good solution by minimizing or maximizing the value of this function



Local search and optimization

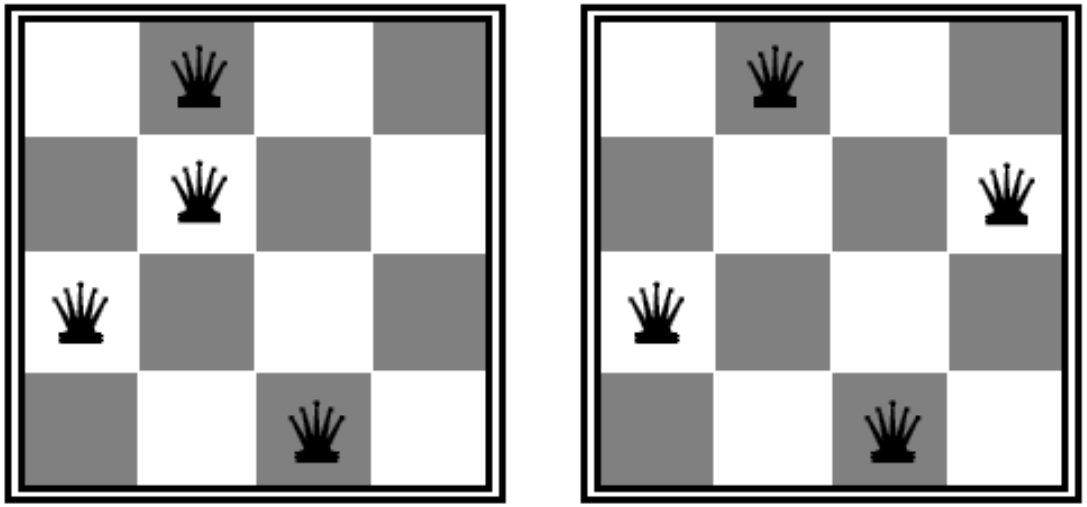
- Local search:
  - Use single current state and move to neighboring states.
- Idea: start with an initial guess at a solution and incrementally improve it until it is one
- Advantages:
  - Use very little memory
  - Find often *reasonable* solutions in large or infinite state spaces.
- Useful for pure optimization problems.
  - Find or approximate best state according to some *objective function*
  - *Optimal if the space to be searched is convex*

Example: Traveling salesman problem

- Find the shortest tour connecting a given set of cities
- **State space:** all possible tours
- **Objective function:** length of tour

Example: *n*-queens problem

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- **State space:** all possible *n*-queen configurations
- What's the **objective function**?
  - Number of pairwise conflicts

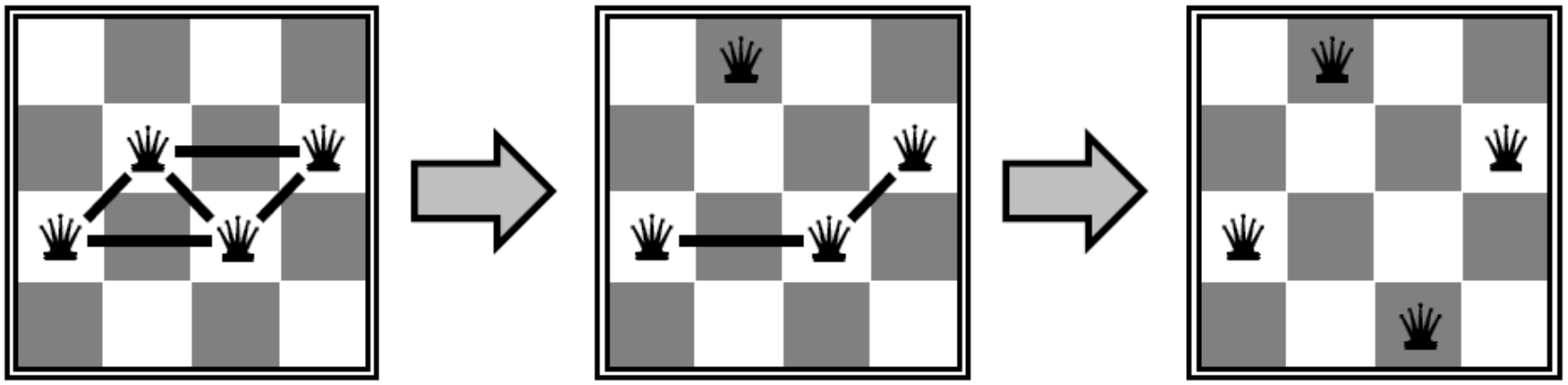


Hill-climbing (greedy) search

- Idea: keep a single “current” state and try to locally improve it
- “Like climbing mount Everest in thick fog with amnesia”

Example: *n*-queens problem

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- **State space:** all possible *n*-queen configurations
- **Objective function:** number of pairwise conflicts
- What's a possible local improvement strategy?
  - Move one queen within its column to reduce conflicts



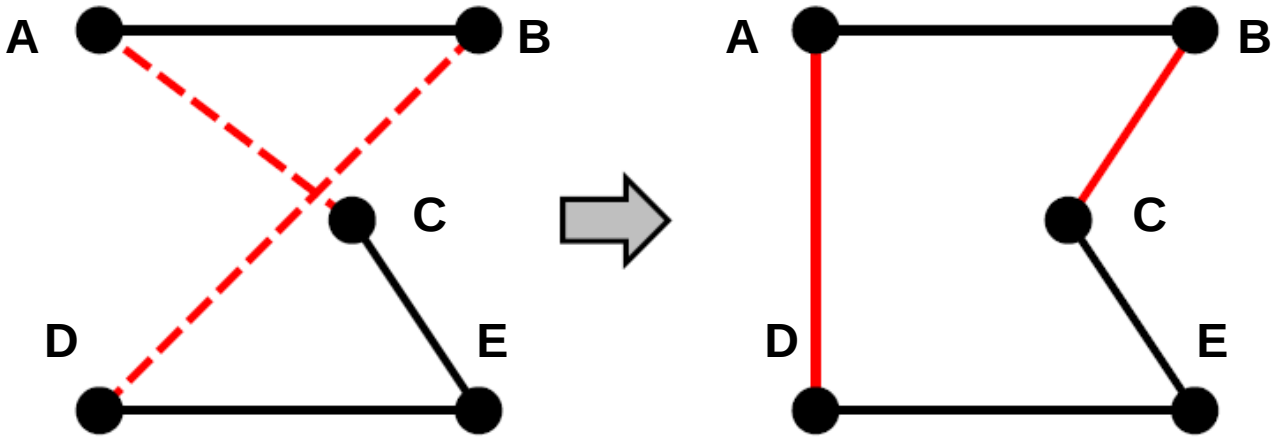
Example: *n*-queens problem

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- **State space:** all possible *n*-queen configurations
- **Objective function:** number of pairwise conflicts
- What's a possible local improvement strategy?
  - Move one queen within its column to reduce conflicts

h = 17

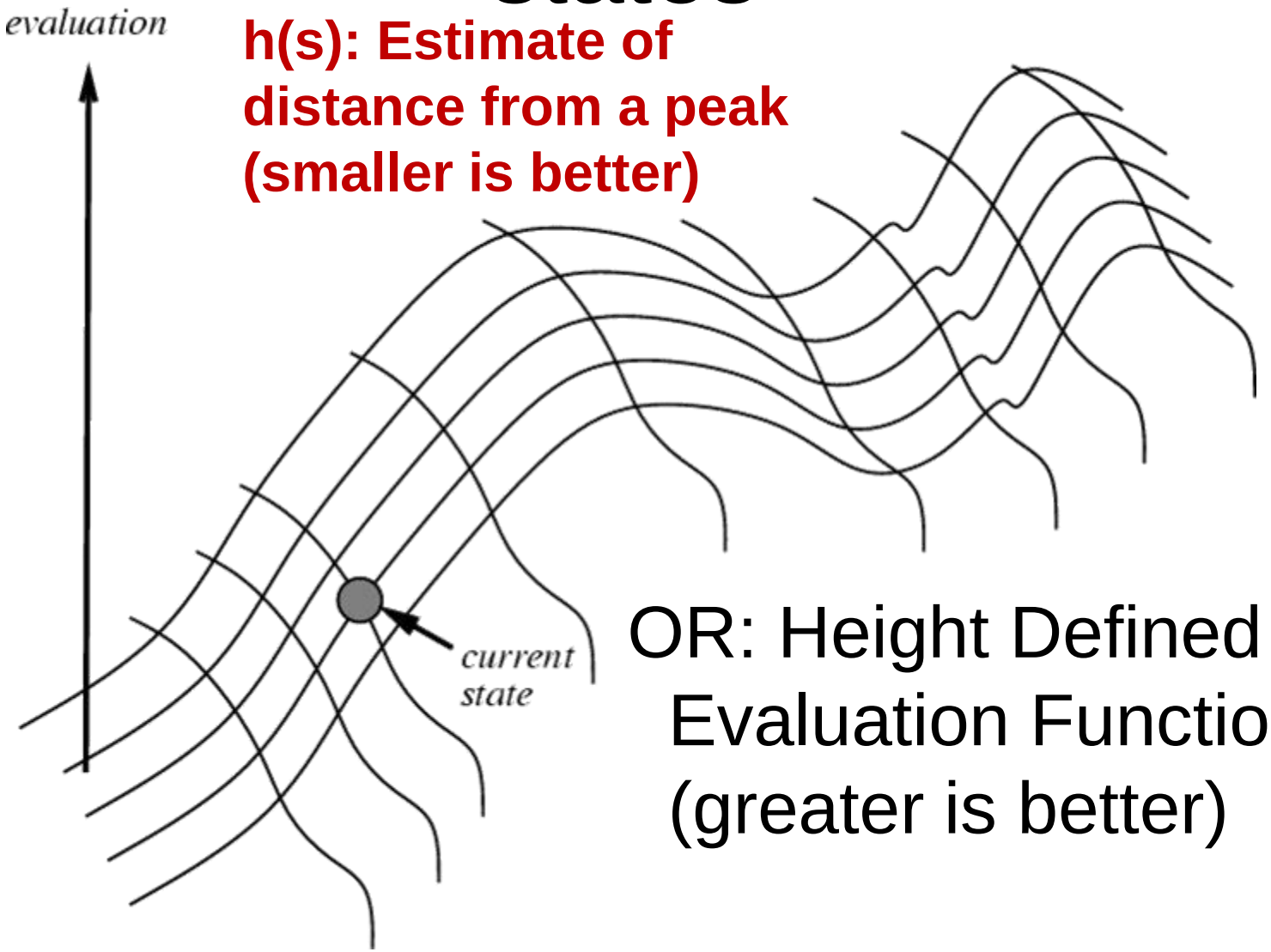
Example: Traveling Salesman Problem

- Find the shortest tour connecting n cities
- **State space:** all possible tours
- **Objective function:** length of tour
- What's a possible local improvement strategy?
  - Start with any complete tour, perform pairwise exchanges



ABDEC **ABCED**

Hill climbing on a surface of states



h(s): Estimate of distance from a peak (smaller is better)

OR: Height Defined by Evaluation Function (greater is better)

Hill-climbing search

I. While (∃ uphill points):
   - Move in the direction of increasing evaluation function ***f***

II. Let $s_{next} = \arg\max_{s} f(s)$, s a successor state to the current state n

   - If ***f(n)*** < ***f(s)*** then move to ***s***
   - Otherwise halt at ***n***

- Properties:
  - Terminates when a peak is reached.
  - Does not look ahead of the immediate neighbors of the current state.
  - Chooses randomly among the set of best successors, if there is more than one.
  - Doesn't *backtrack*, since it doesn't remember where it's been
- a.k.a. *greedy local search*

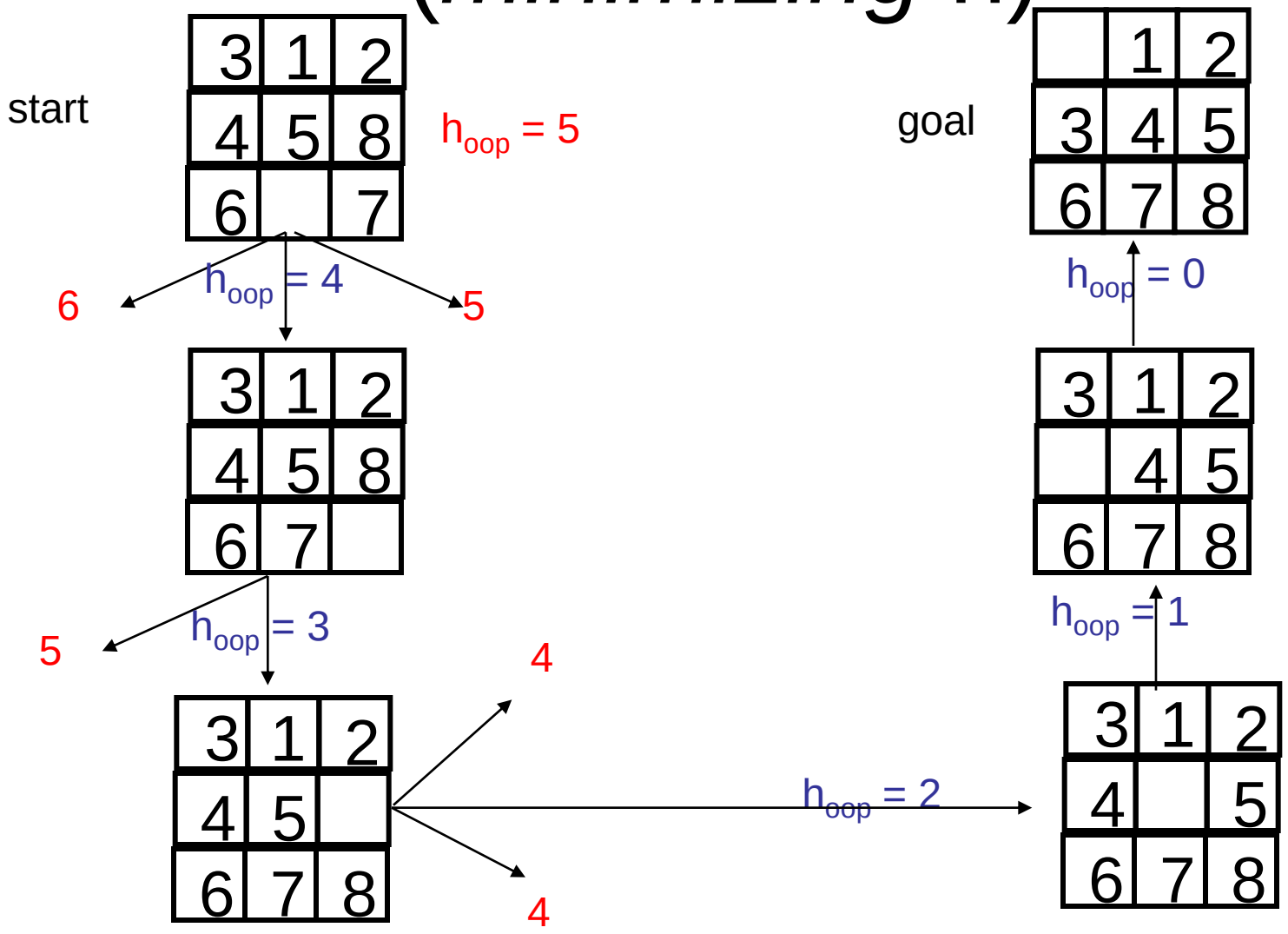
"Like climbing E fog with amnesia"

Hill-climbing (greedy) search

- Initialize *current* to starting state
- Loop:
  - Let *next* = highest-valued successor of *current*
  - If value(*next*) < value(*current*) return *current*
  - Else let *current* = *next*

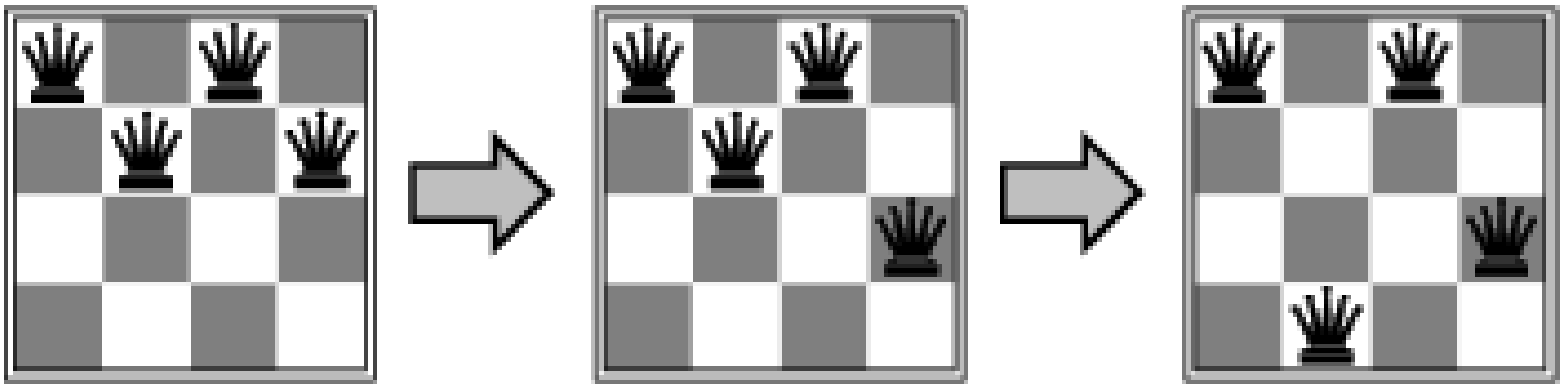
- Variants: choose first better successor, randomly choose among better successors

Hill climbing example I (*minimizing* h)



Hill-climbing Example: *n*-queens

- *n*-queens problem: Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal
- *Good heuristic:* *h* = number of pairs of queens that are attacking each other

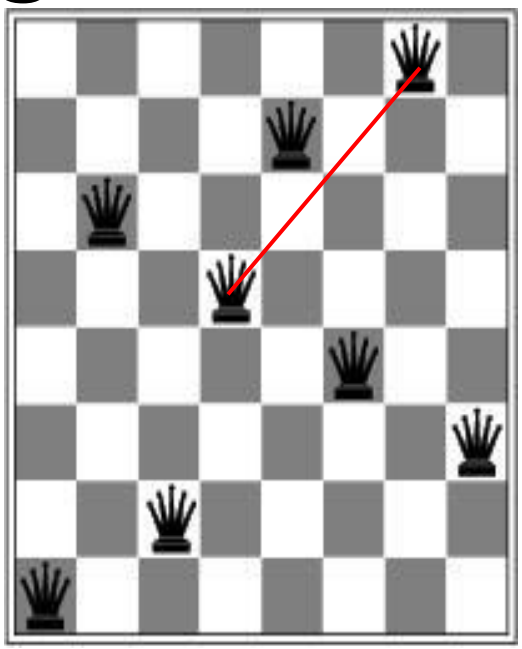


h=5 *h=3* *h=1*

(for illustration)

Hill-climbing example: 8-queens

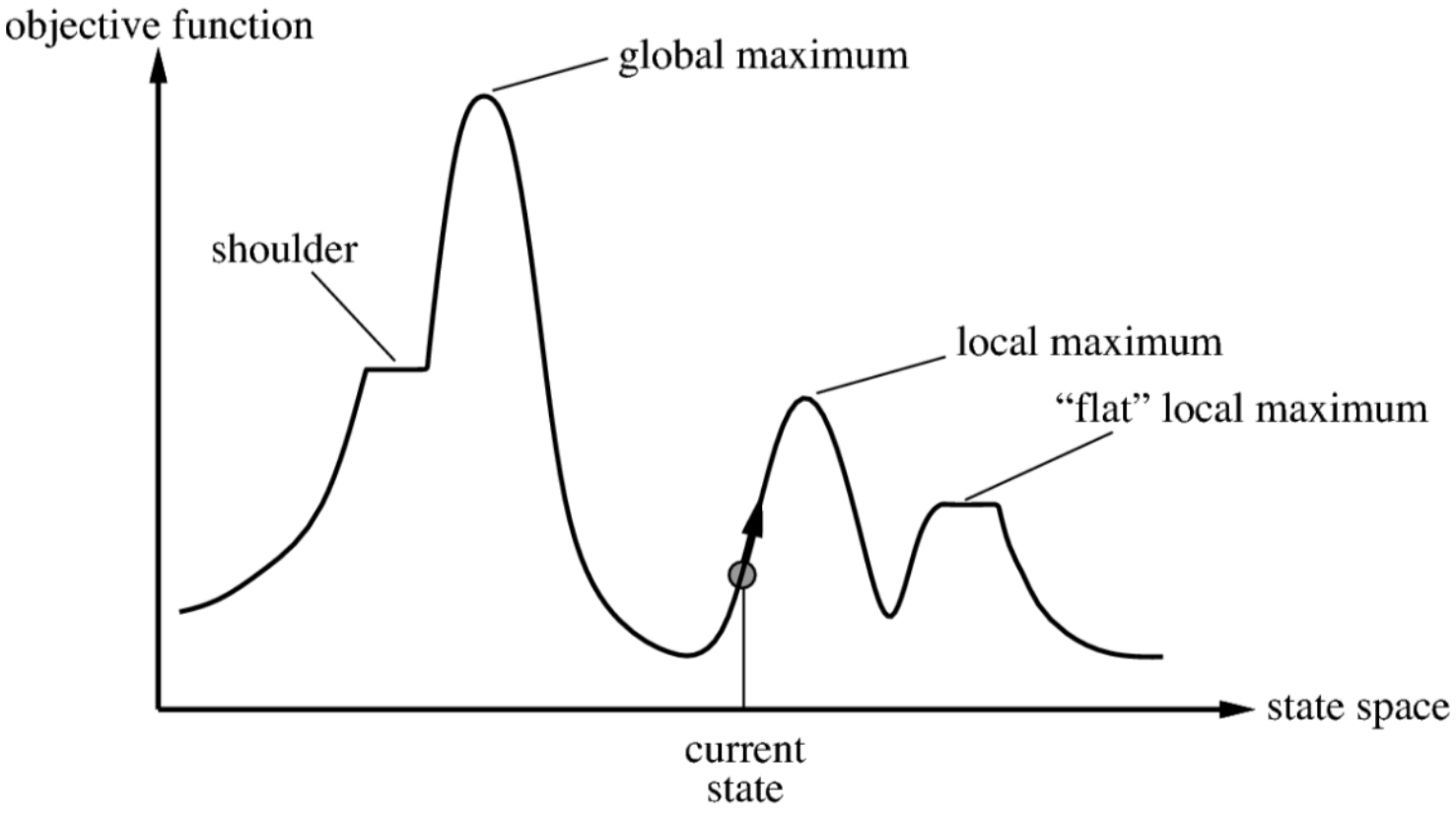
A state with *h*=17 and the *h*-value for each possible successor



A local minimum of *h* in the 8-queens state space (*h*=1).

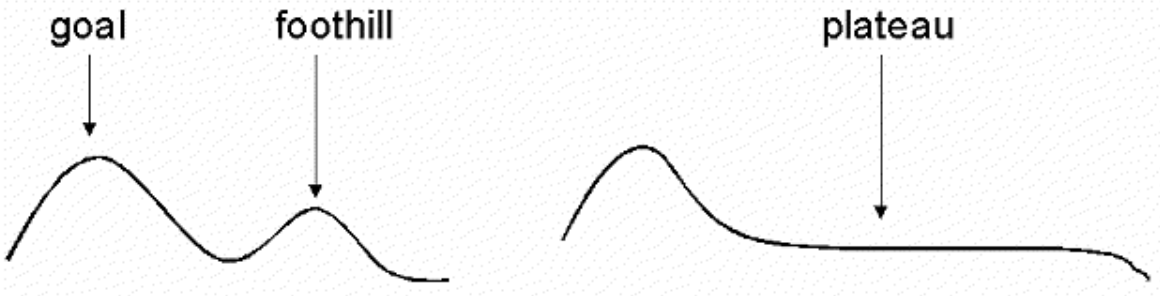
h = number of pairs of queens that are attacking each other

Search Space features

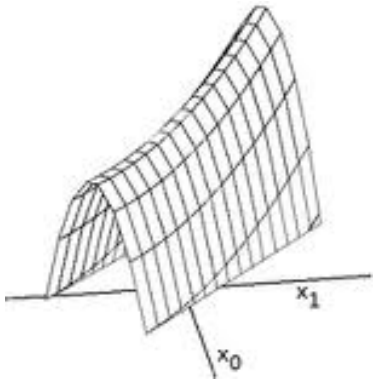


Drawbacks of hill climbing

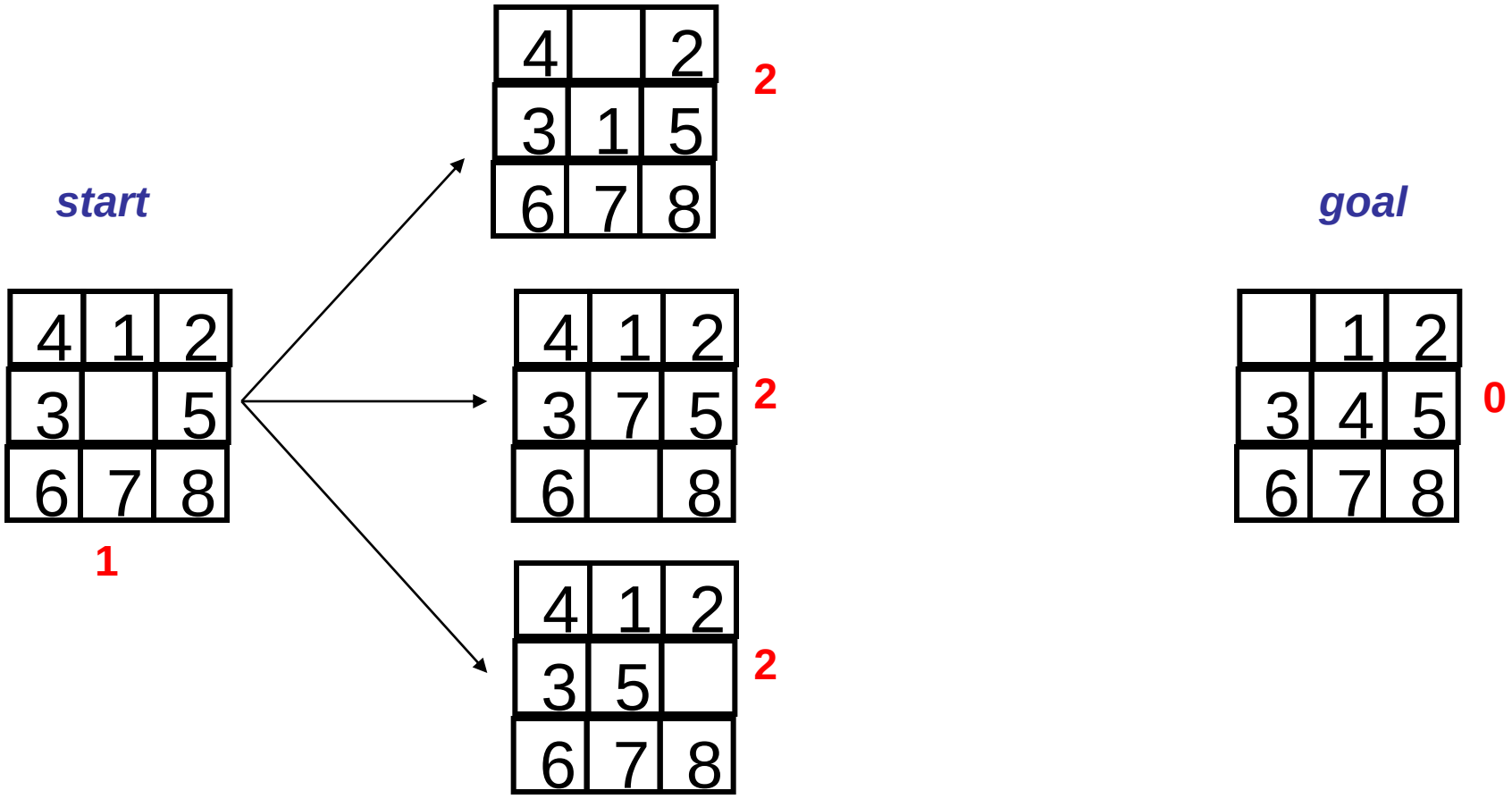
- **Local Maxima:** peaks that aren't the highest point in the space
- **Plateaus:** the space has a broad flat region that gives the search algorithm no direction (random walk)



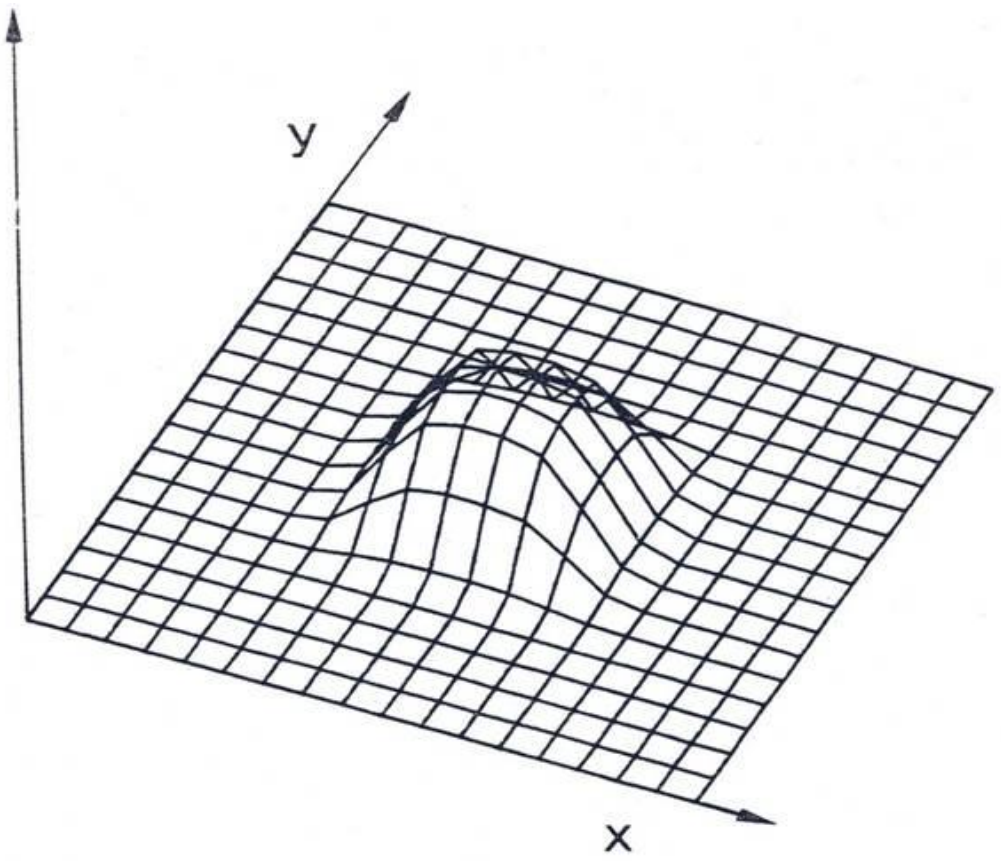
- **Ridges:** dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.



Toy Example of a local "maximum"

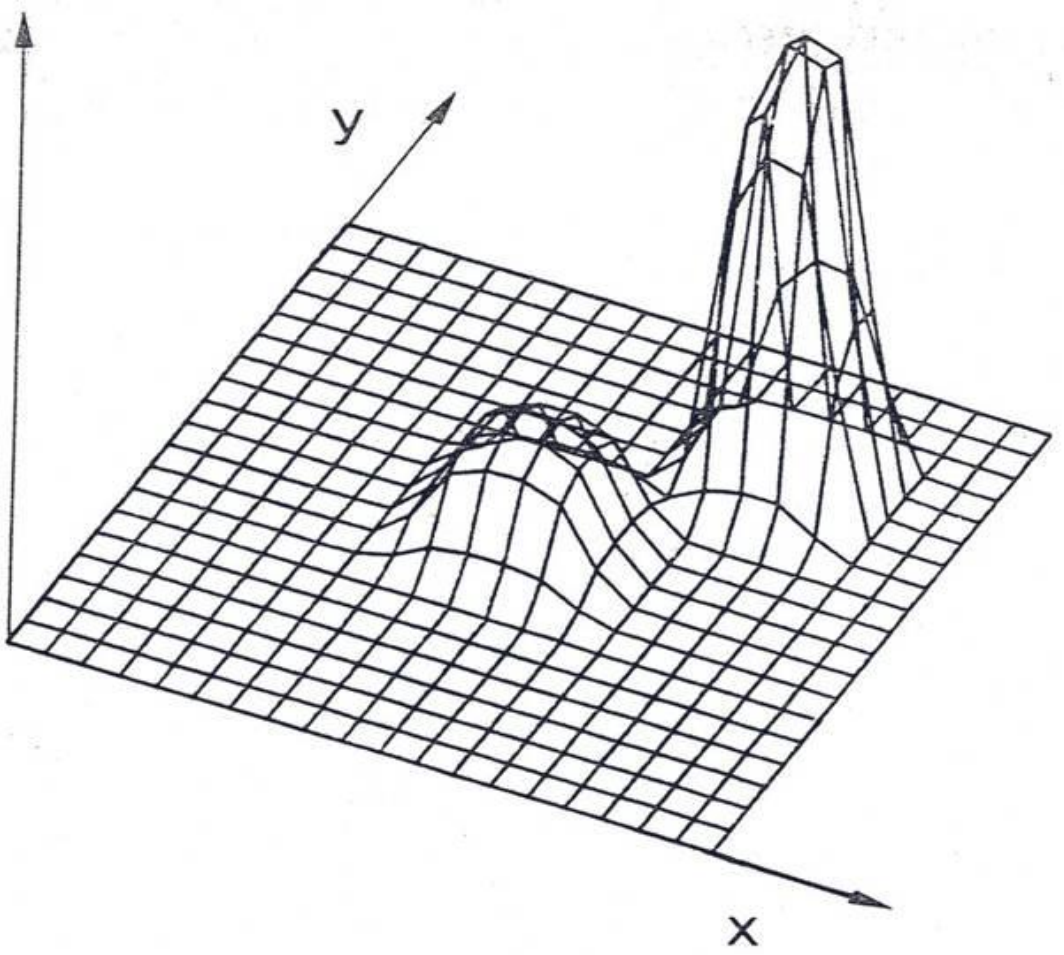


The Shape of an Easy Problem (*Convex*)

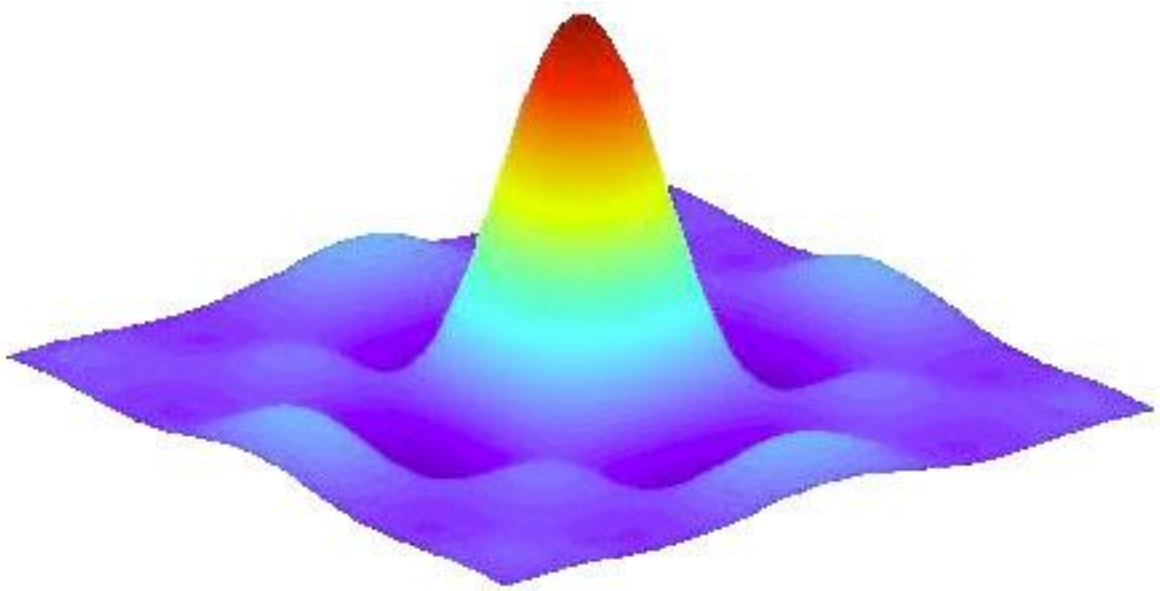


This and next several slides from Goldberg '89

The Shape of a Harder Problem

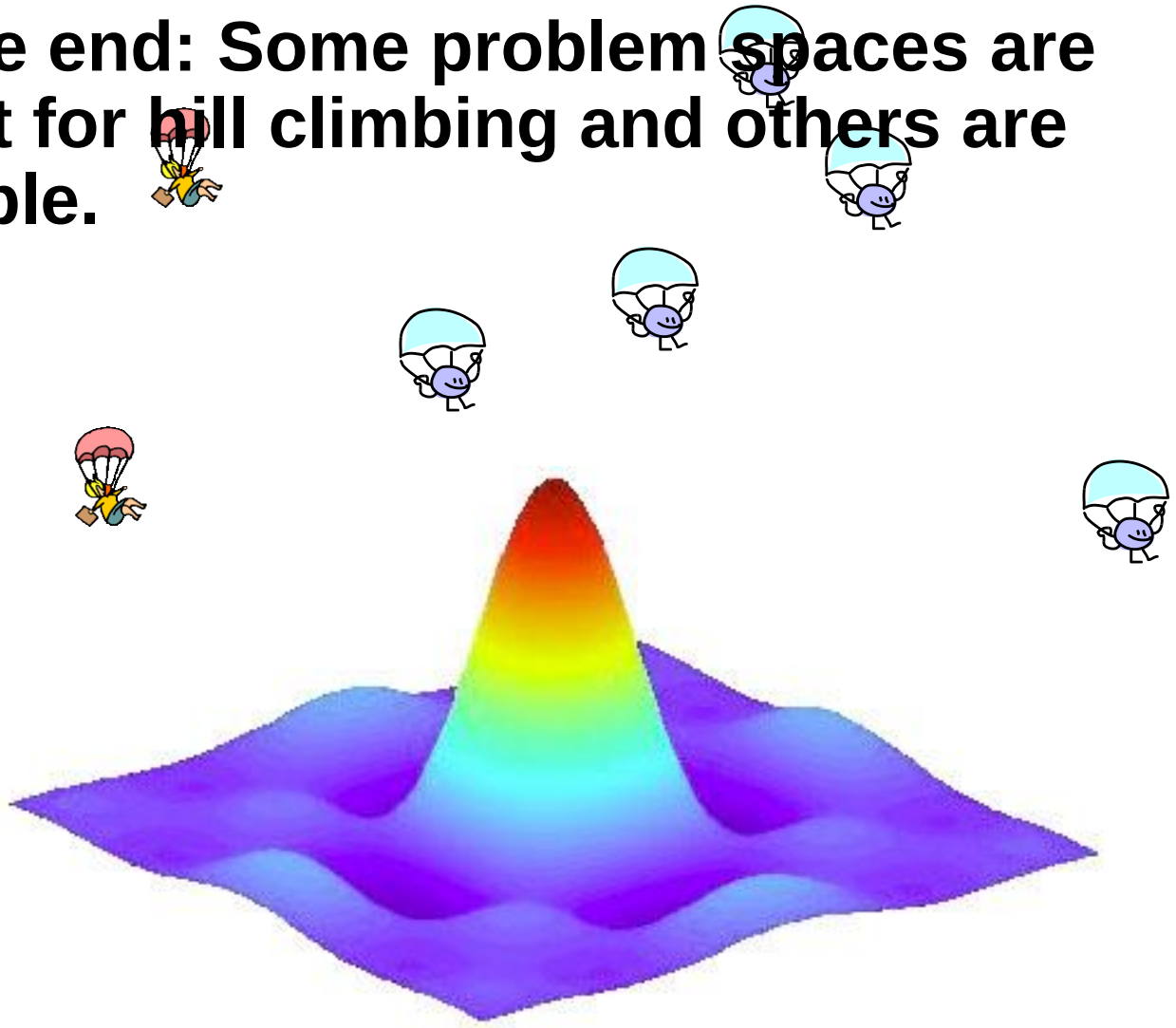


The Shape of a Yet Harder Problem



One Remedy to Drawbacks of Hill Climbing: *Random Restart*

- **In the end: Some problem spaces are great for hill climbing and others are terrible.**



Metaheuristics

- **Problem-Independent methods**
- Meta : in an upper level
- Heuristic : to find
- **Fundamental Properties of Metaheuristics [Blum and Roli 2003]**
  - Metaheuristics are strategies that “guide” the search process.
  - The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
  - Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
  - Metaheuristic algorithms are approximate and usually non-deterministic.

Metaheuristics

Fundamental Properties of Metaheuristics [Blum and Roli 2003]

- Metaheuristics are not problem-specific.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Meta-heuristics are general methods that can be altered to fit the specific problem.

Classification of Metaheuristics

- Single-solution-based :
  - The single-solution-based algorithms work on a single solution e.g. Hill Climbing, Simulated Annealing, Tabu Search
- Population-based :
  - The population-based algorithms work on a population of solutions e.g. Genetic Algorithm, Ant Colony Optimization, Particle Swarm Optimization etc

Advantages of Meta-Heuristics

- Very flexible
- Often global optimizers
- Often robust to problem size, problem instance and random variables
- May be only practical alternative

Disadvantages of Meta-Heuristics

- Often need problem specific information / techniques
- Optimality (convergence) may not be guaranteed
- Lack of theoretic basis
- Different searches may yield different solutions to the same problem (stochastic)
- Stopping criteria
- Multiple search parameters

M**eta-heuristics** inspired by

NATURE

Nature

- As a source of inspiration
- LETS HAVE A LOOK !!

Nature Inspired Algorithms

- Nature provide some of the efficient ways to solve problems
  - Algorithms imitating processes inspired from nature
- **Problems**
  - Aircraft wing design

  - Bullet train

  - Robotic Spy Plane

Bird flocking

Winging at speeds of up to 40 miles per hour, an entire flock of birds can make hairpin turns in an instant. How do they do it?

- Birds do not engage in any behavior that does not bring them a benefit for survival in some way. There are many advantages to flocking, including:
- Foraging
- Protection
- Aerodynamics
- Raising families

Fish Schooling

- The fish synchronizing their swimming so they all move at the same speed and in the same direction, then the fish are said to be **schooling**

- Advantages of schooling:

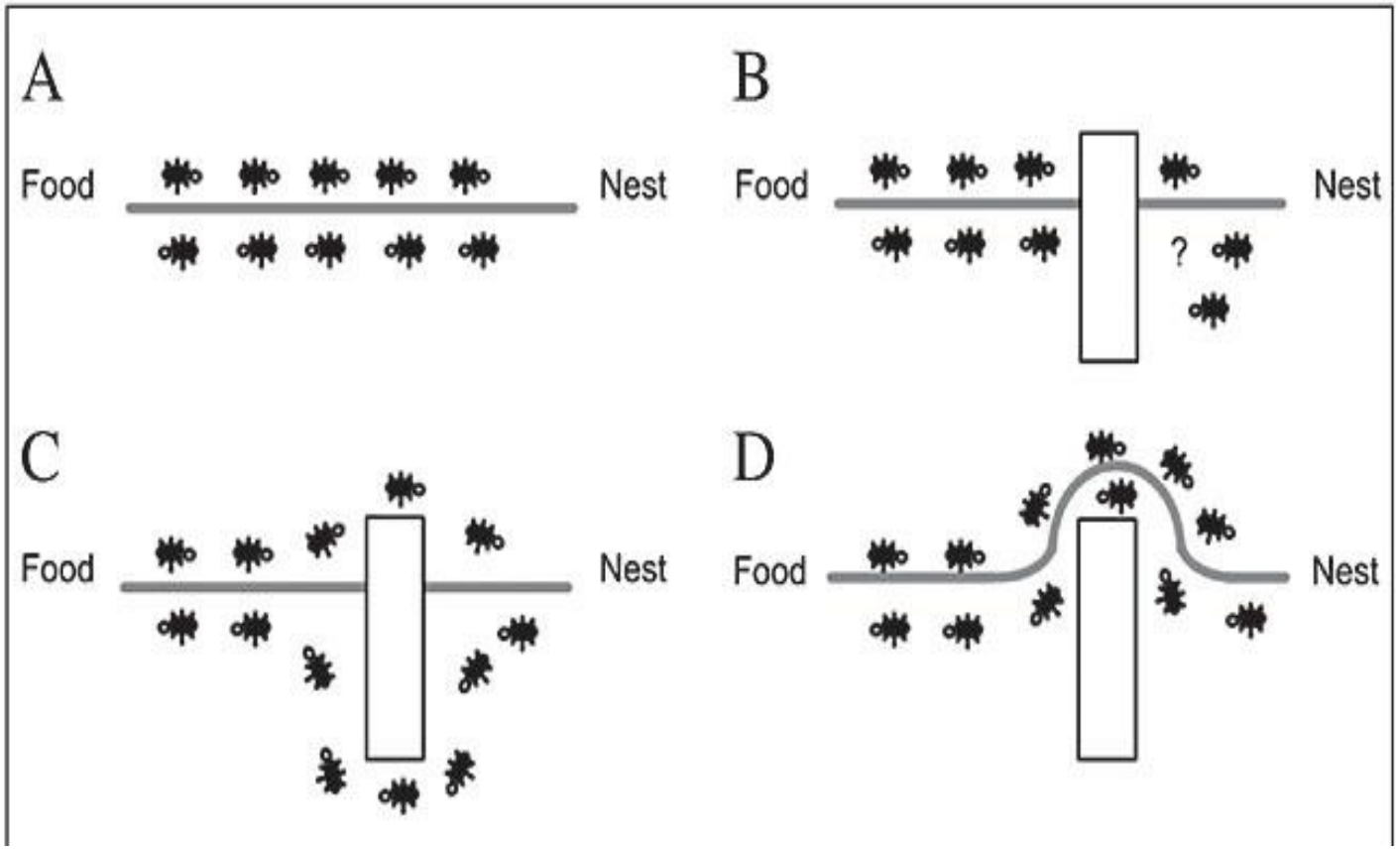
-> defense against predators

-> Foraging

-> higher success in finding a mate

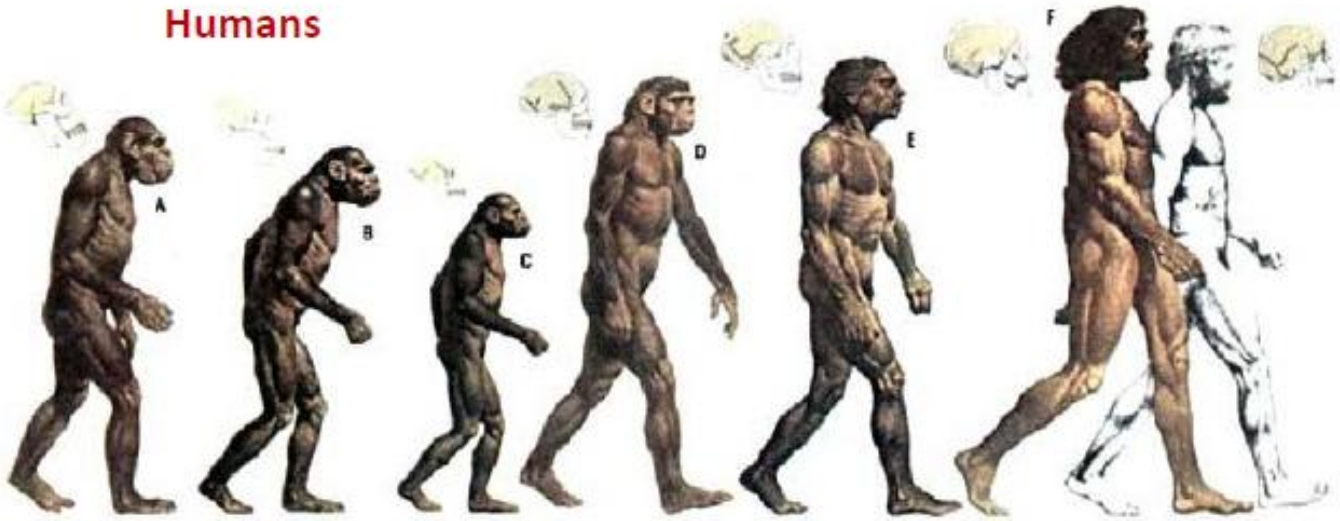
-> increased hydrodynamic efficiency.

Ant colony



Evolution

Humans



Macintosh

Genetics : Survival of the fittest

Aren’t they all showing some optimal behavior??

Thus nature forms a great inspiration for scientists and engineers to optimize real world problems.

Introduction

- The majority of nature-inspired algorithms are based on some successful characteristics of biological system.

- Therefore, the largest fraction of nature inspired algorithms are biology-inspired, or bio-inspired for short.

- Many algorithms have been developed by using inspiration from physical and chemical systems. Some may even be based on music.

Classification Of nature inspired Algorithms

- Thus, we can divide all existing algorithms into four major categories:

1) Swarm intelligence (SI) based
2) Bio-inspired (but not SI-based)
3) Physics/chemistry-based
4) Others.

Bio-inspired but not SI

- SI-based algorithms are a subset of bio-inspired algorithms, while bio-inspired algorithms are a subset of nature-inspired algorithms. That is

- SI-based < bio-inspired < nature-inspired

Bio-inspired but not SI

- Genetic algorithms (evolutionary)
- Differential Evolution (evolutionary)
- Flower pollination algorithm

3. Physics/ Chemistry based

- Many algorithms have been developed by mimicking certain physical and/or chemical laws, including electrical charges, gravity, river systems, etc.

$$\frac{\text{Physics algorithms}}{\text{Chemistry algorithms}} \begin{cases} \notin \text{bio-inspired algorithms} \\ \in \text{nature-inspired algorithms} \end{cases}$$

- Gravitational search
- River formation dynamics
- Water Cycle algorithm
- Galaxy based search algorithm

4. Others

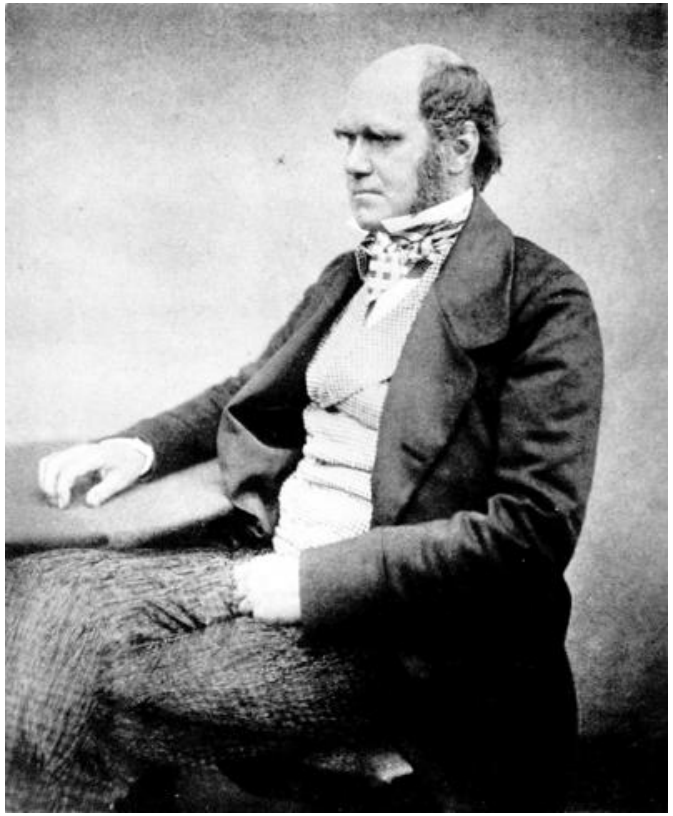
- Some algorithms are not bio-inspired or physics/chemistry-based
- These algorithms have been developed by using various characteristics from different sources, such as social, emotional, etc.
  - Artificial cooperative search
  - Social emotional optimization
  - Backtracking optimization search

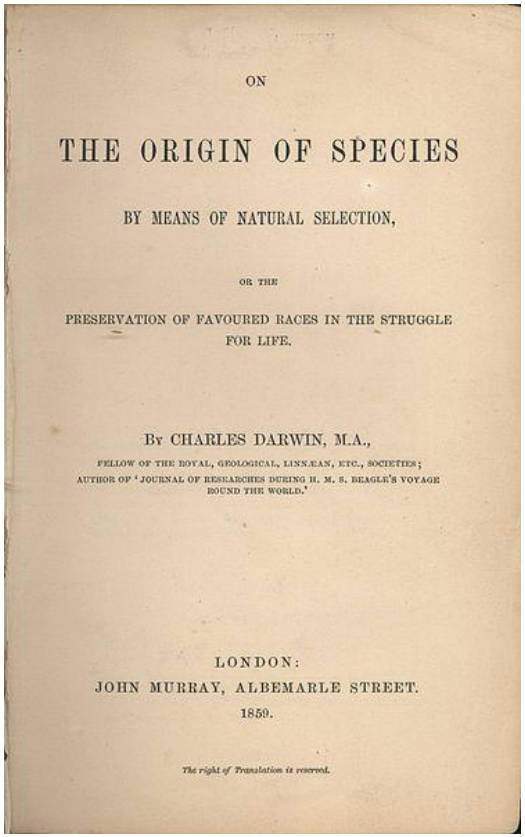
GENETIC ALGORITHMS

DARWIN'S EVOLUTION THEORY

Evolution/Survival of the Fittest

The theory of evolution is the statement that all species on Earth have arisen in this way by evolution from one or more very simple self-reproducing molecules in the primeval soup. I.e. we have evolved via the accumulation of countless advantageous (in context) mutations over countless generations, and species have diversified to occupy environmental niches, as a result of different environments favouring different mutations.





ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

By CHARLES DARWIN, M.A.,

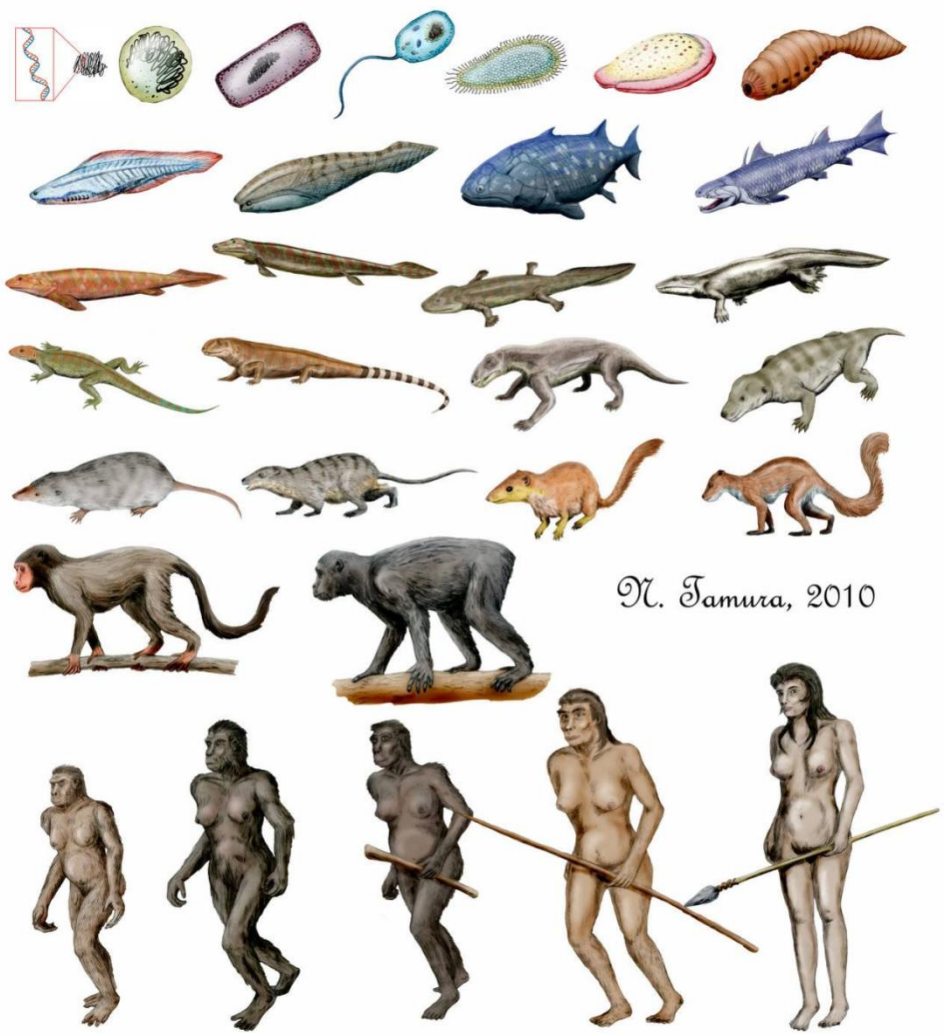
FELLOW OF THE ROYAL, GEOLOGICAL, LINNEAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

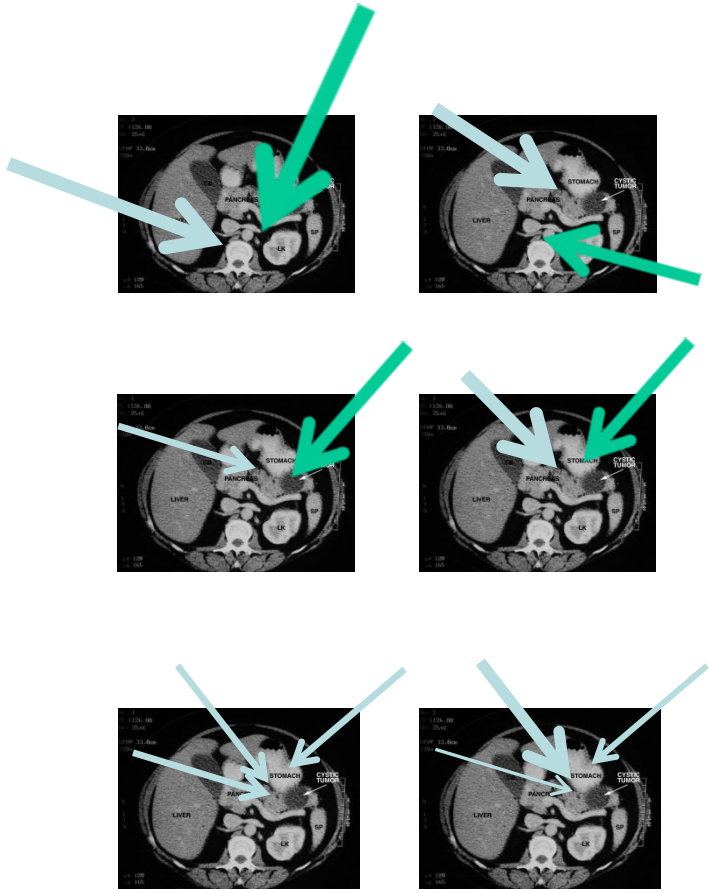
The right of Translation is reserved.

Evolving

humans

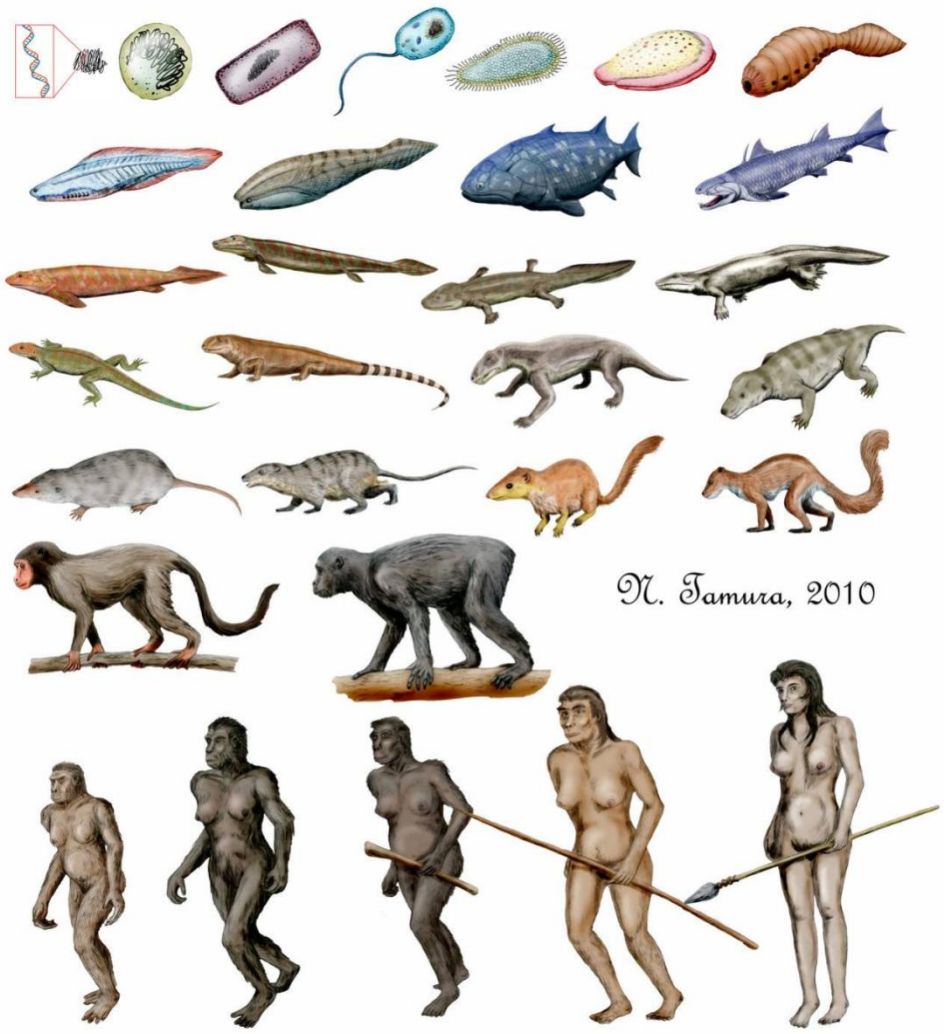


radiotherapy treatment plans

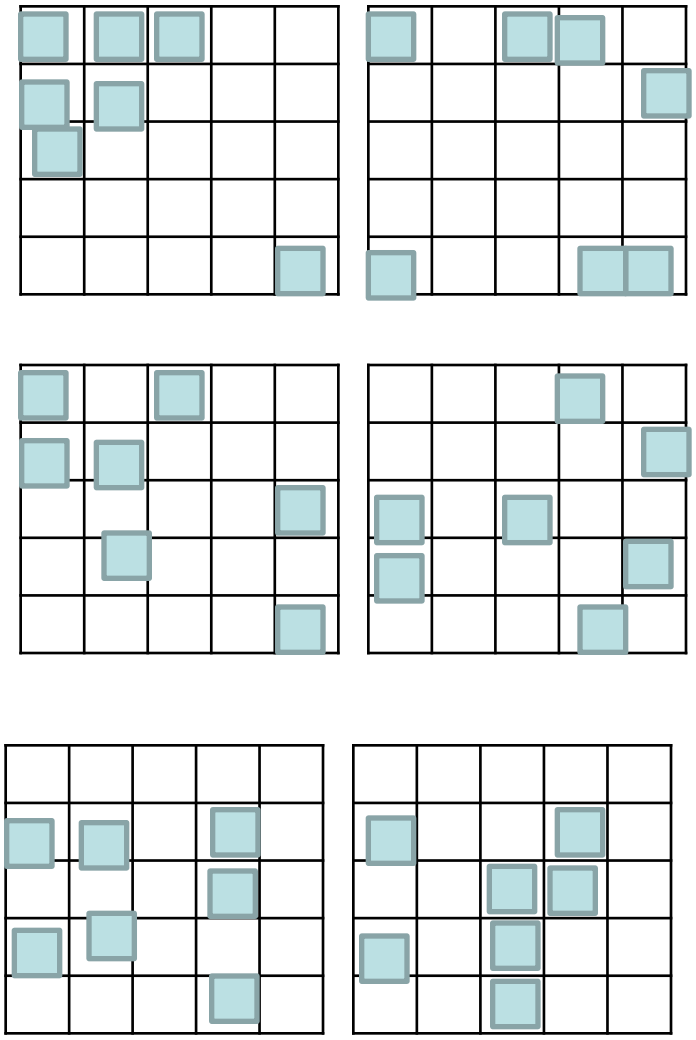


Evolving

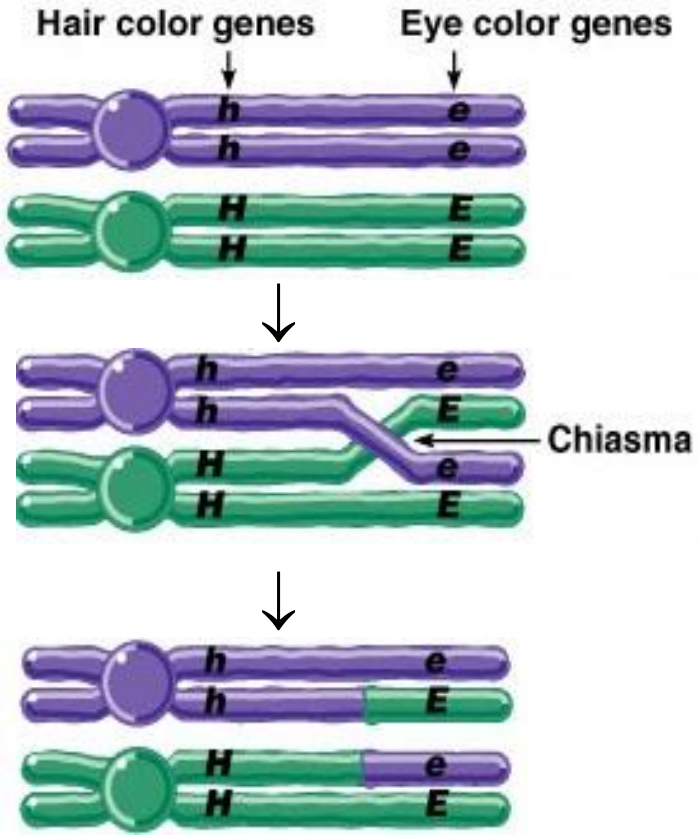
humans



lecture timetables



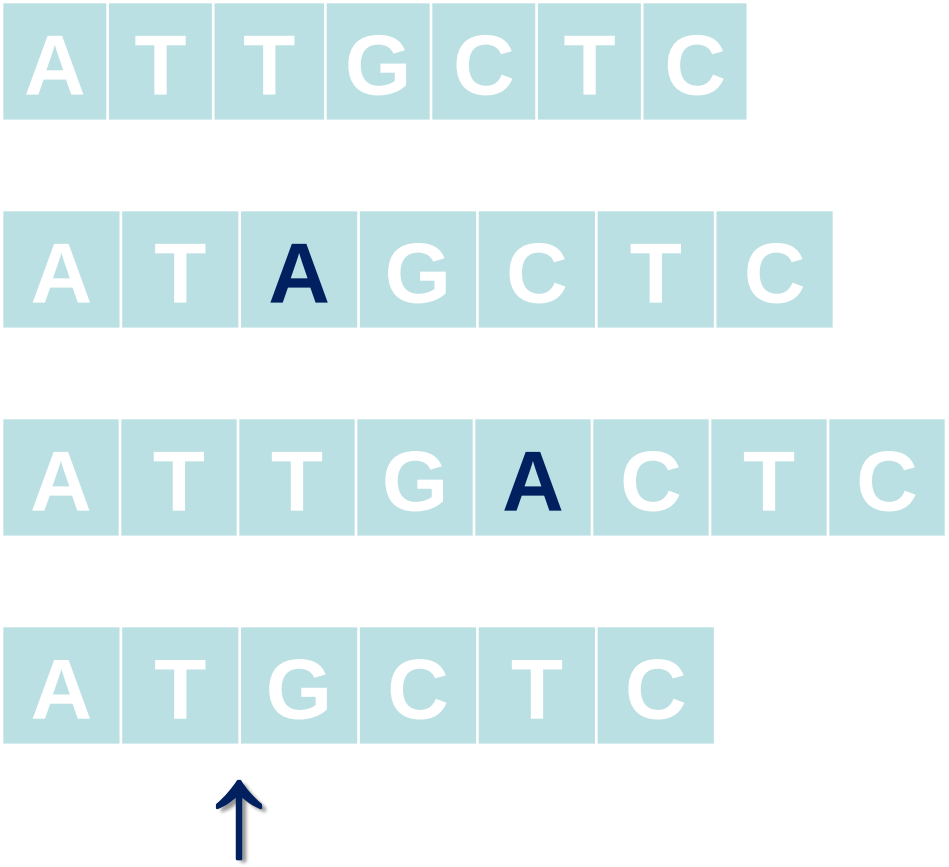
EVOLUTION AT THE GENE LEVEL



Genetic Crossover

- Chromosomes from two different parents
- Chromatids from each overlap at Chiasma
- Recombinant chromosomes are form
- Further passed on to the progeny

EVOLUTION AT THE molecular LEVEL

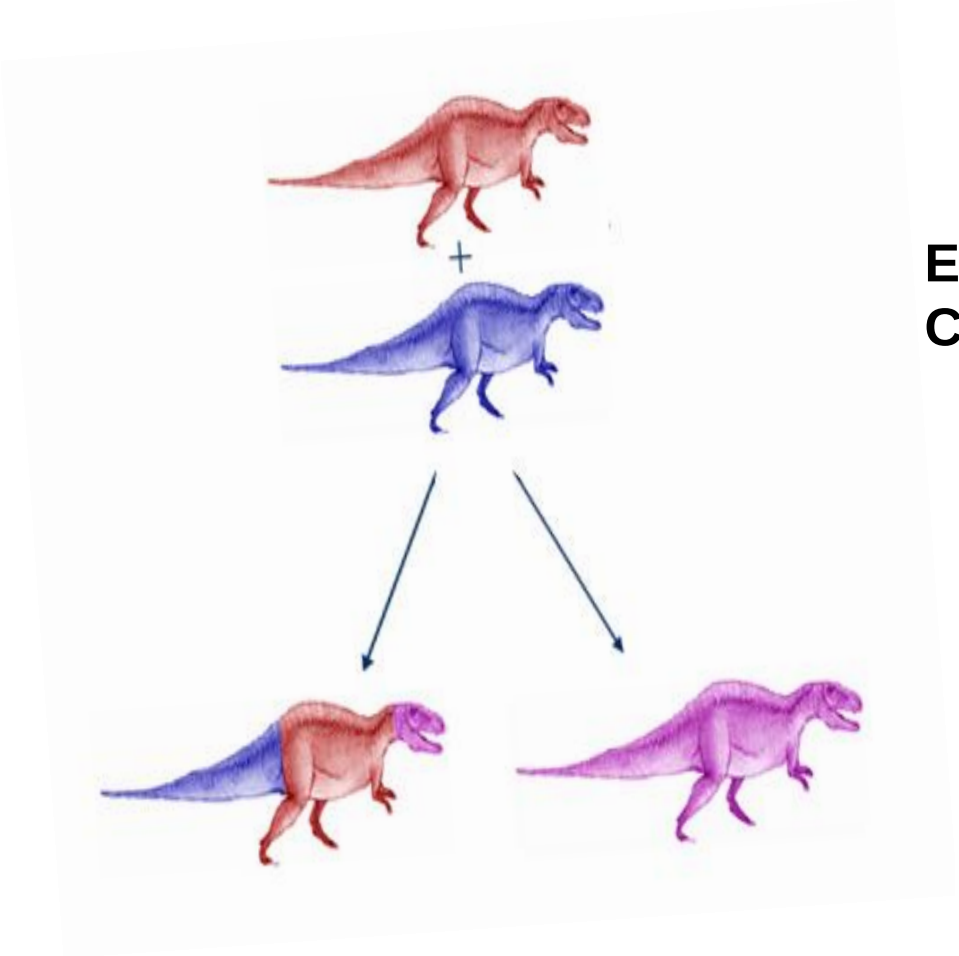


ORIGINAL

SUBSTITUTION

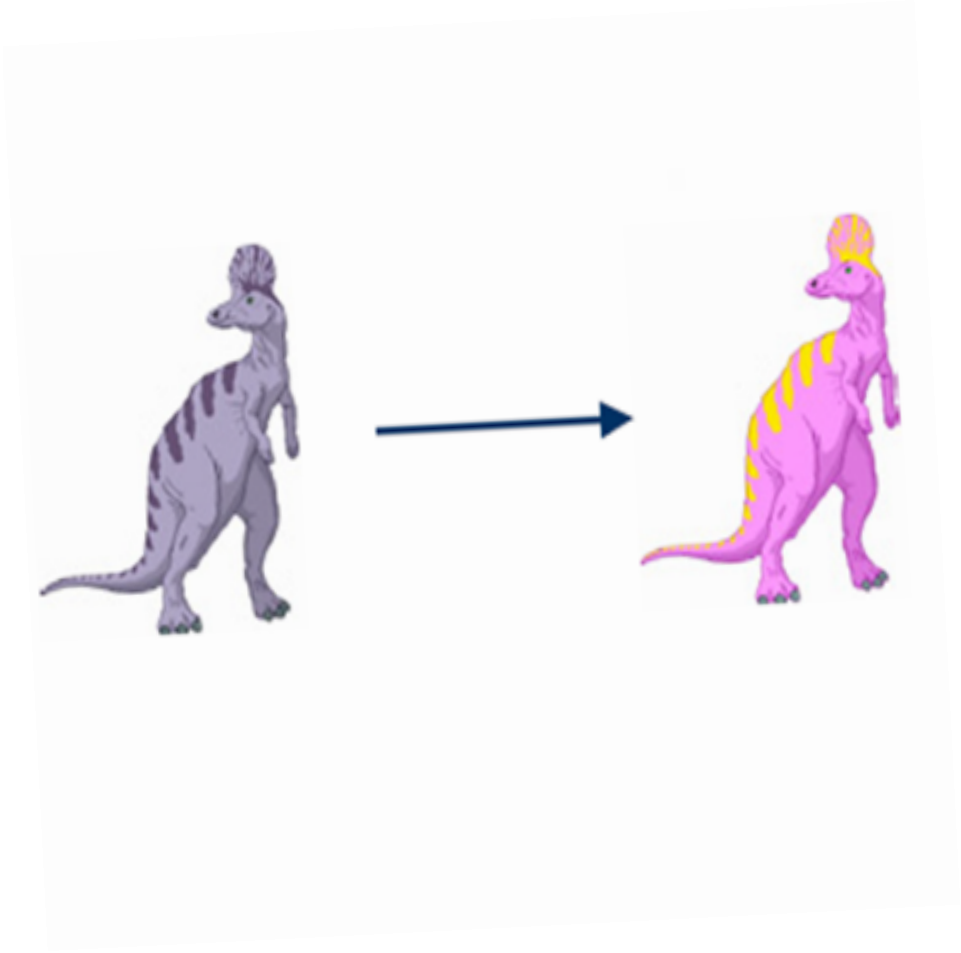
ADDITION

DELETION



EVOLUTION DUE TO GENETIC CROSSOVER

Offsprings have combinations of features inherited from each parent



EVOLUTION DUE TO GENETIC MUTATION

Random changes are observed

Terminologies

- ***Gene***: a solution to problem represented as a set of parameters ,these parameters known as genes.
- ***Chromosome:*** genes joined together to form a string of values called chromosome. Individual - carrier of the genetic information (chromosome). It is characterized by its state in the search space, its fitness (objective function value).
- ***Population :*** pool of individuals which allows the application of genetic operators.
- ***Fitness function :*** The term “fitness function” is often used as a synonym for objective function.
- ***Generation :*** (natural) time unit of the EA, an iteration step of an evolutionary algorithm.

Genetic Algorithms

- An *algorithm* is a set of instructions that is repeated to solve a problem.

- A *genetic algorithm* conceptually follows steps inspired by the biological processes of evolution.

- Genetic Algorithms follow the idea of **SURVIVAL OF THE FITTEST**- Better and better solutions evolve from previous generations until a near optimal solution is obtained.

Genetic Algorithms

- Also known as *evolutionary algorithms*, genetic algorithms demonstrate self organization and adaptation similar to the way that the fittest biological organism survive and reproduce.

- A genetic algorithm is an iterative procedure that represents its candidate solutions as strings of genes called chromosomes.

- **Generally applied to spaces which are too large**

Simple GA

```
{
   initialize population;
   evaluate population;
   while TerminationCriteriaNotSatisfied
   {
   select parents for reproduction;
   perform crossover  and mutation;
   repair();
  evaluate population;
   }
}
```

Every loop called generation

Evolutionary Algorithms

Step 1	*t:= 0*
Step 2	Initialize *P(t)*
Step 3	Evaluate *P(t)*
Step 4	While not terminate do *P'(t)* := variation [*P(t)*]; evaluate [*P'(t)*]; *P(t+1)* := select [*P'(t) U P(t)*]; *t := t + 1;* End

Evolutionary algorithms = Selection + Crossover + Mutation

Simulating Evolution

- We need the following
  1. Representation of an individual
  2. Fitness Function
  3. Reproduction Method
  4. Selection Criteria

1. Representing an Individual

- An individual is data structure representing the “genetic structure” of a possible solution.

- Genetic structure consists of an alphabet (usually 0,1)

1. Representing an Individual

Binary Encoding

- Most Common – string of bits, 0 or 1.
  Chrom: A = 1 0 11 0 0 1 0 1 1
  Chrom: B = 1 1 1 1 1 1 0 0 0 0

- Gives you many possibilities

- Example Problem: Knapsack problem
- The problem: there are things with given value and size. The knapsack has given capacity. Select things to maximize the values.

- Encoding: Each bit says, if the corresponding thing is in the knapsack

1. Representing an Individual cont..

Permutation Encoding

- Used in “ordering problems”

- Every chromosome is a string of numbers, which represents number is a sequence.

  Chrom A: 1 5 3 2 6 4 7 9 8

  Chrom B: 8 5 6 7 2 3 1 4 9

- Example: Travelling salesman problem
- The problem: cities that must be visited.
- Encoding says order of cities in which salesman will visit.

Representing an Individual cont..

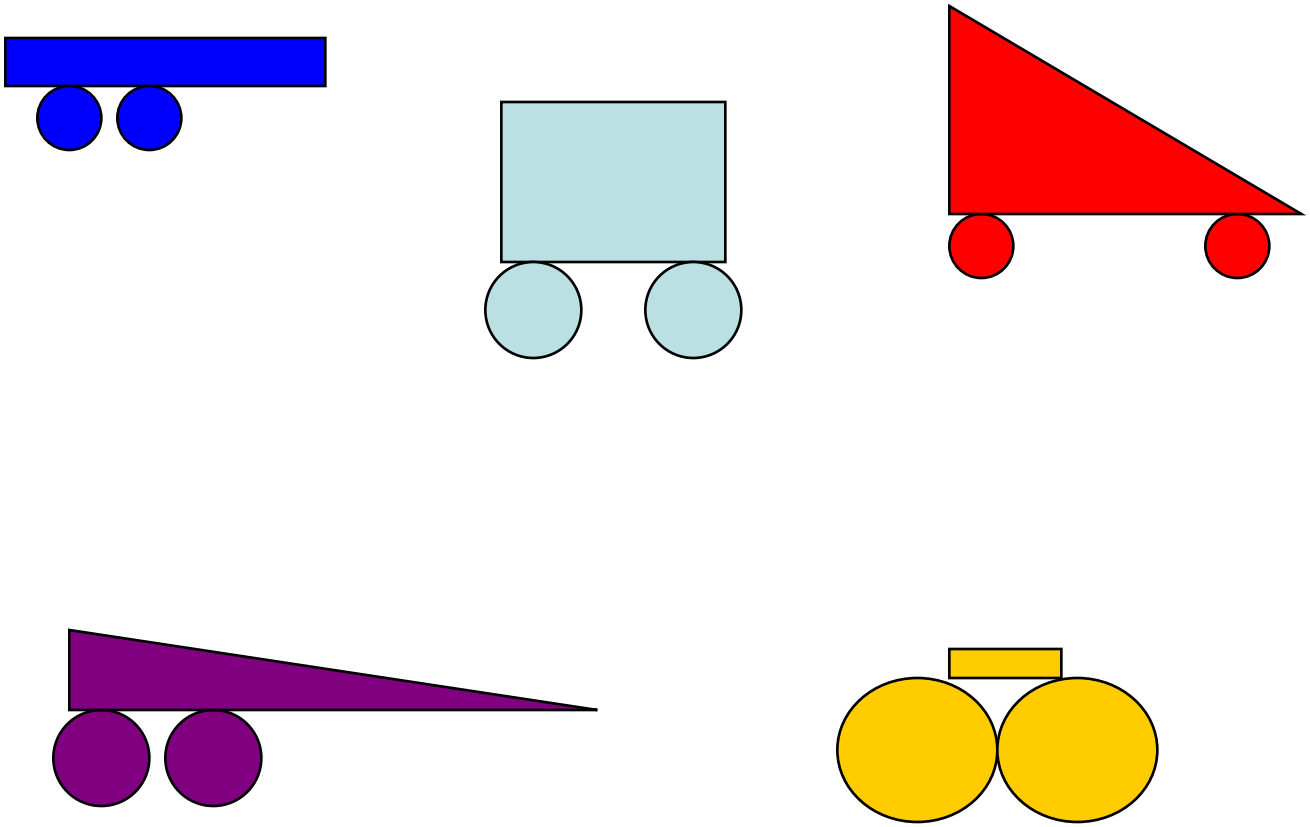
Value Encoding

- Used for complicated values (real numbers) and when binary coding would be difficult

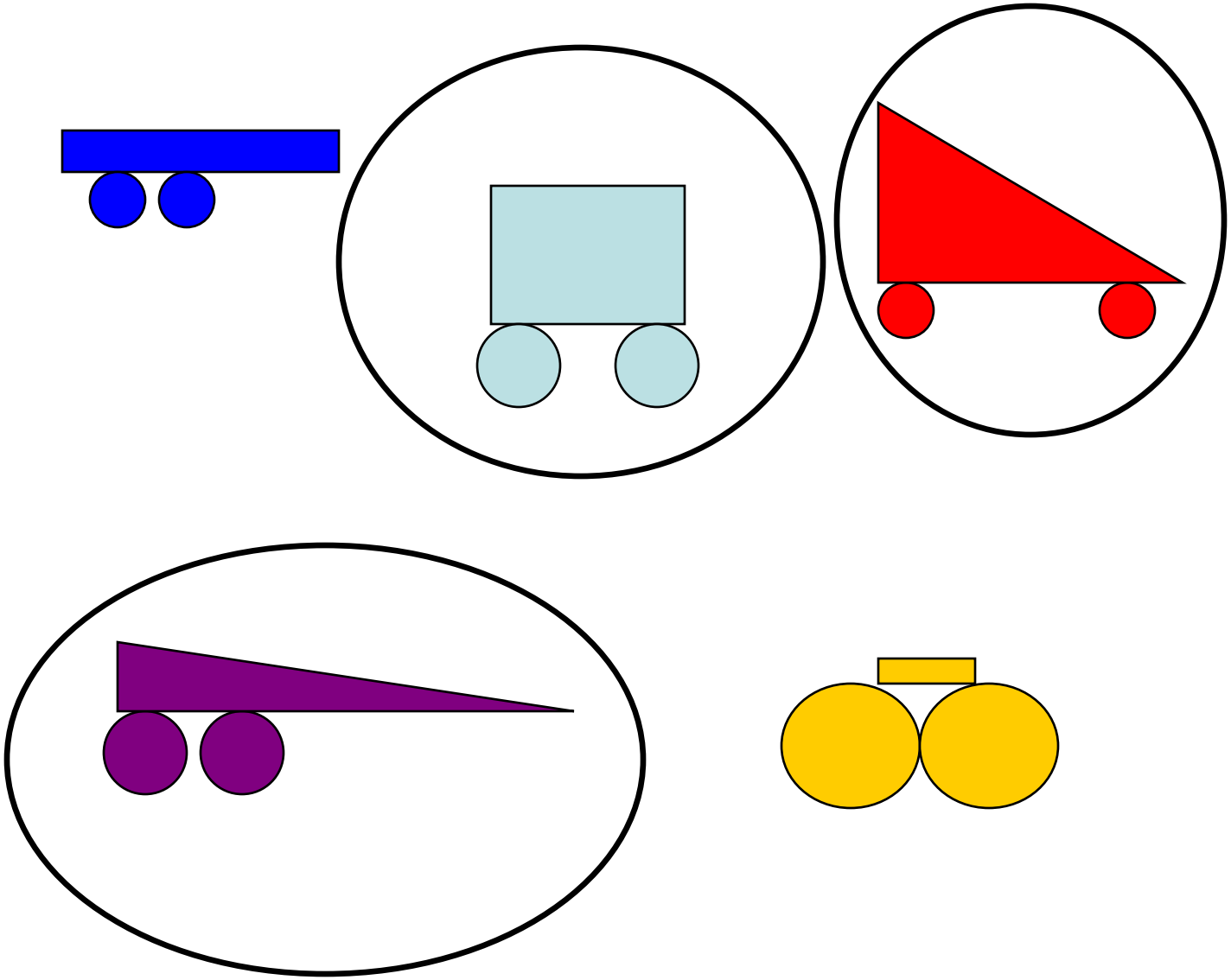
- Each chromosome is a string of some values.
  Chrom A: 1.2323 5.3243 0.4556
  Chrom B: abcdjeifjdhdierjfd
  Chrom C: (back), (back), (right), (forward), (left)

Example: Finding weights for neural nets.
The problem: find weights for network
Encoding: Real values that represent weights

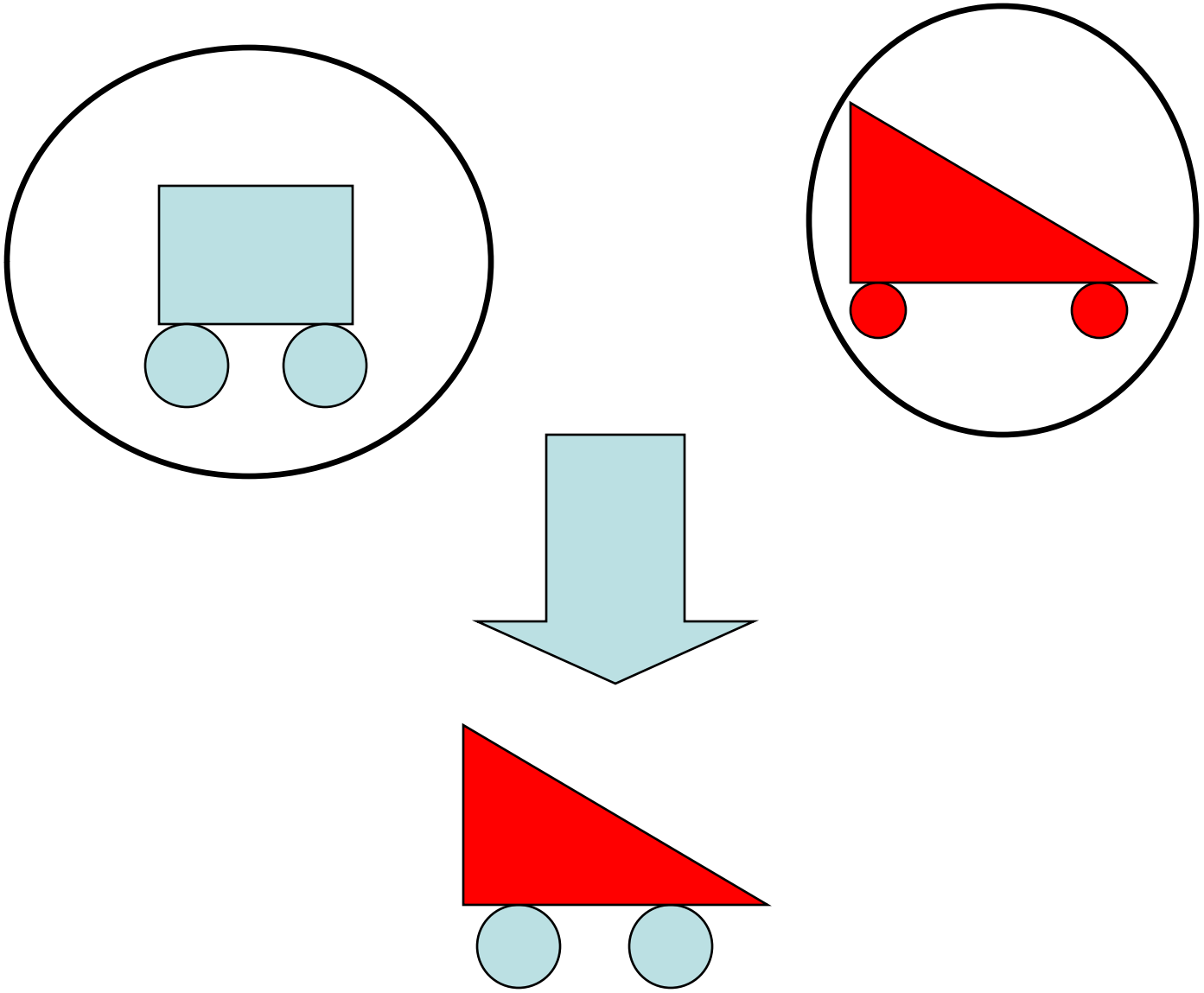
Initial population



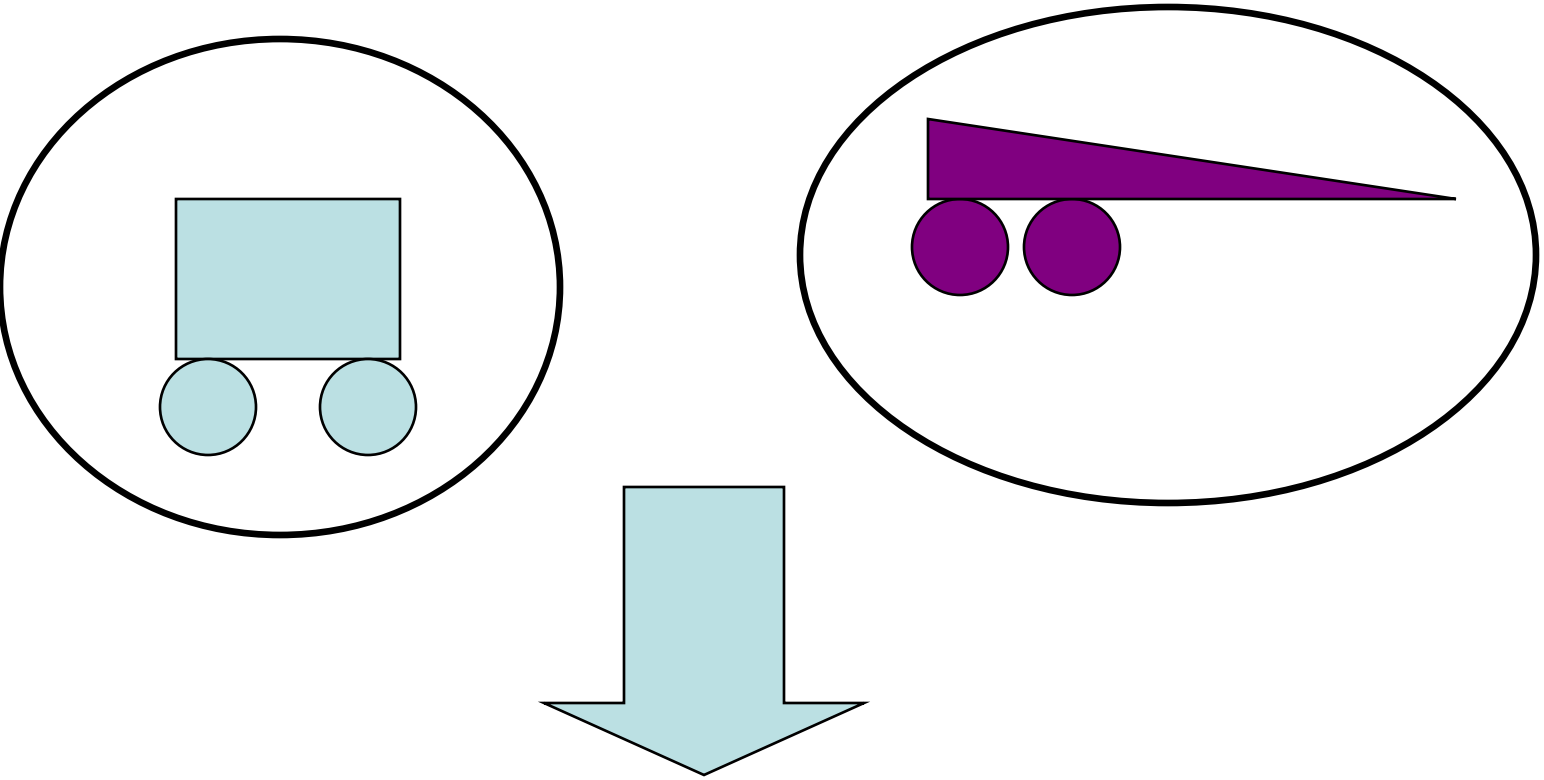
Select



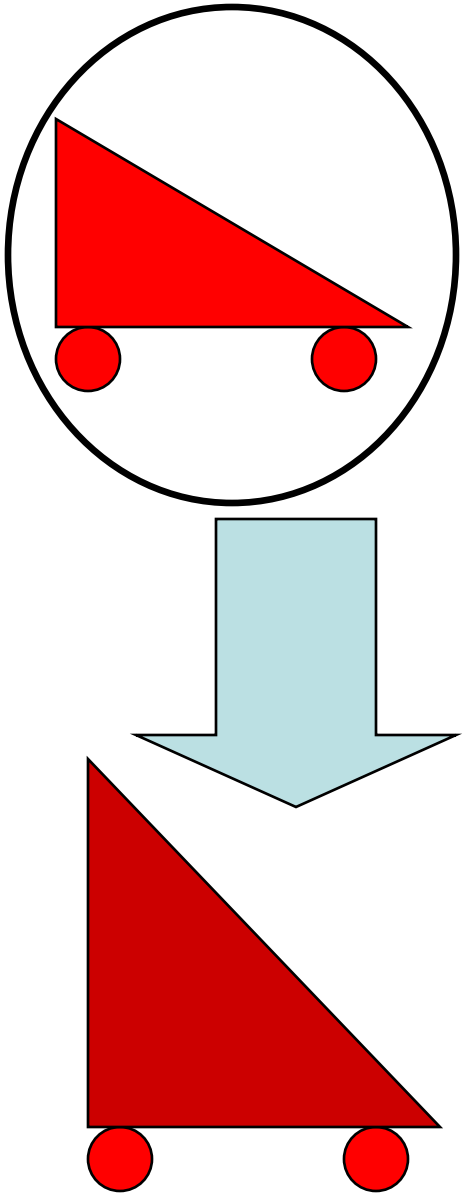
Crossover



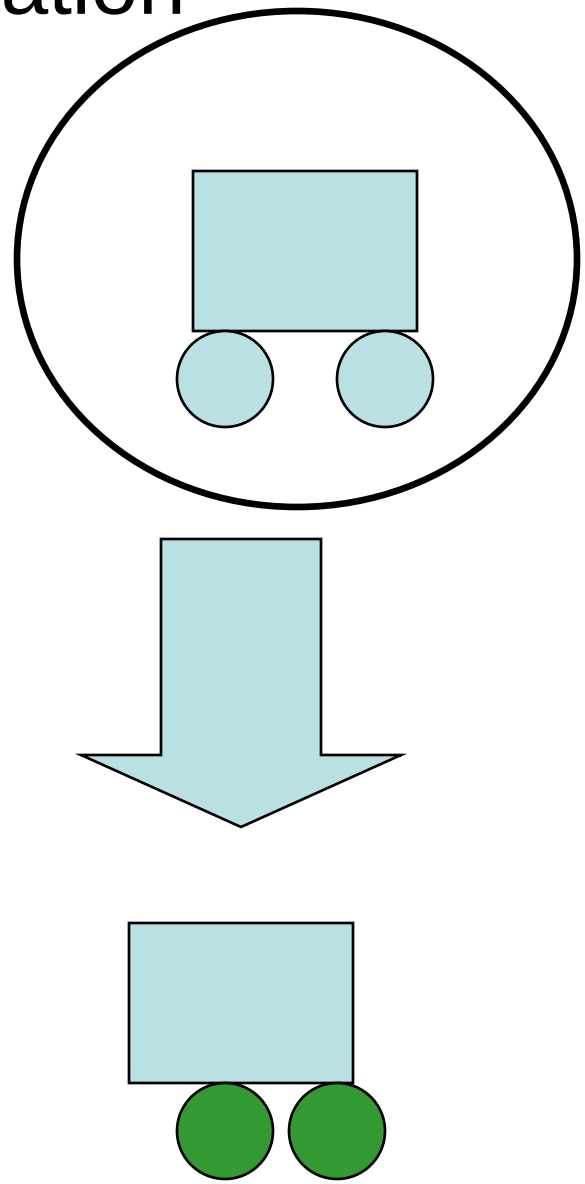
Another Crossover



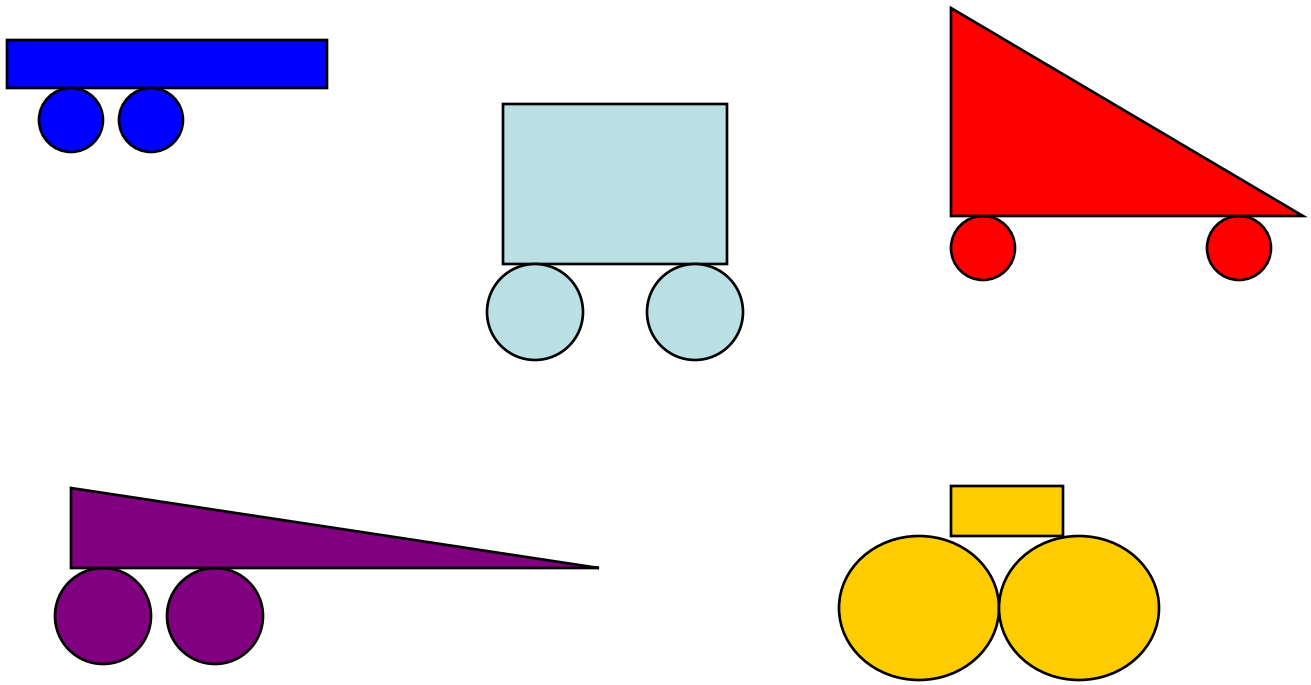
A mutation



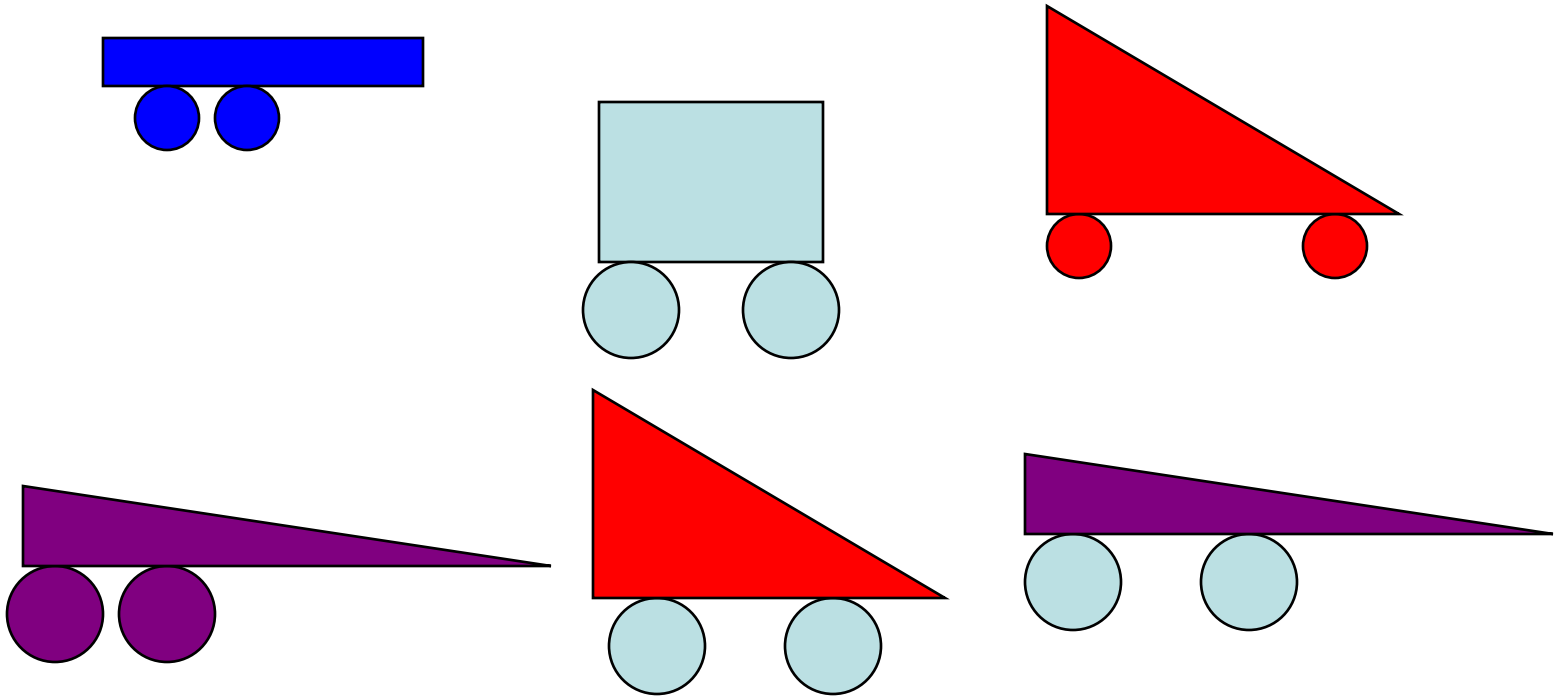
Another Mutation



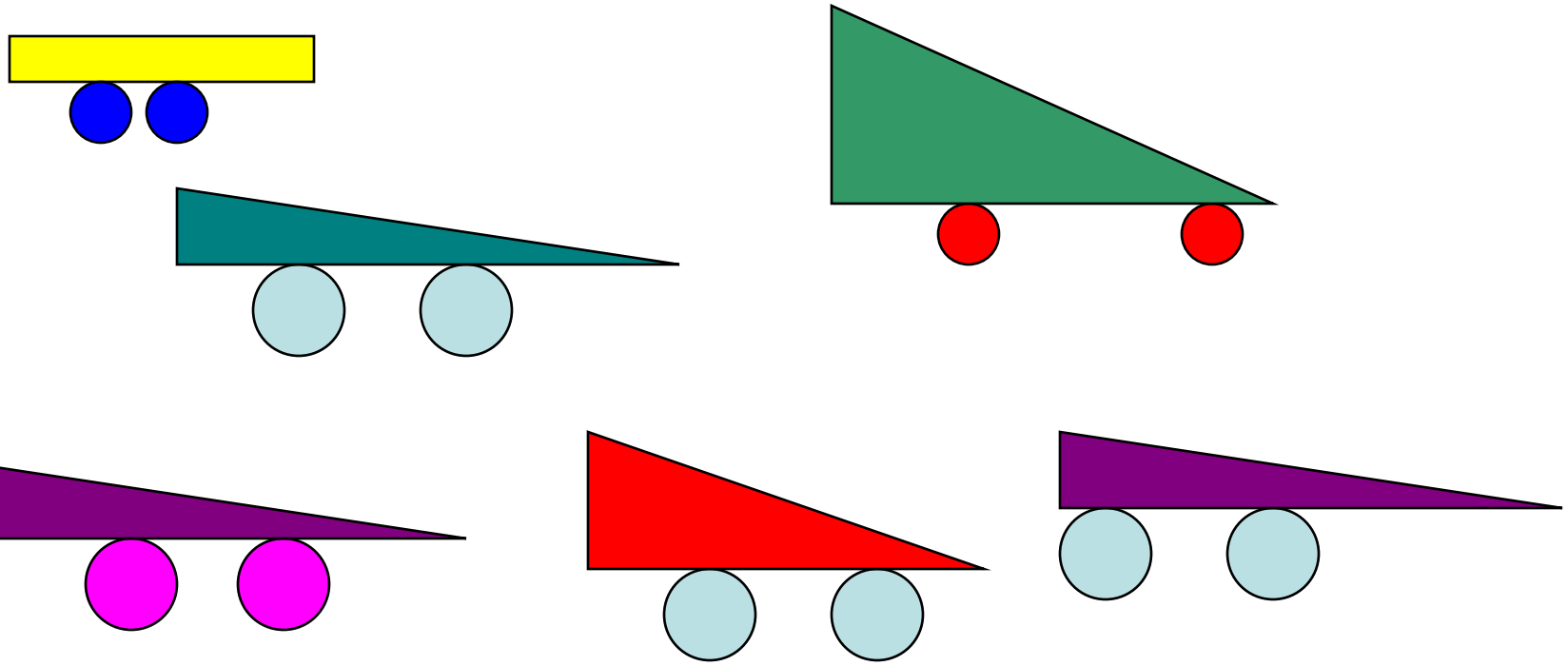
Old population + children



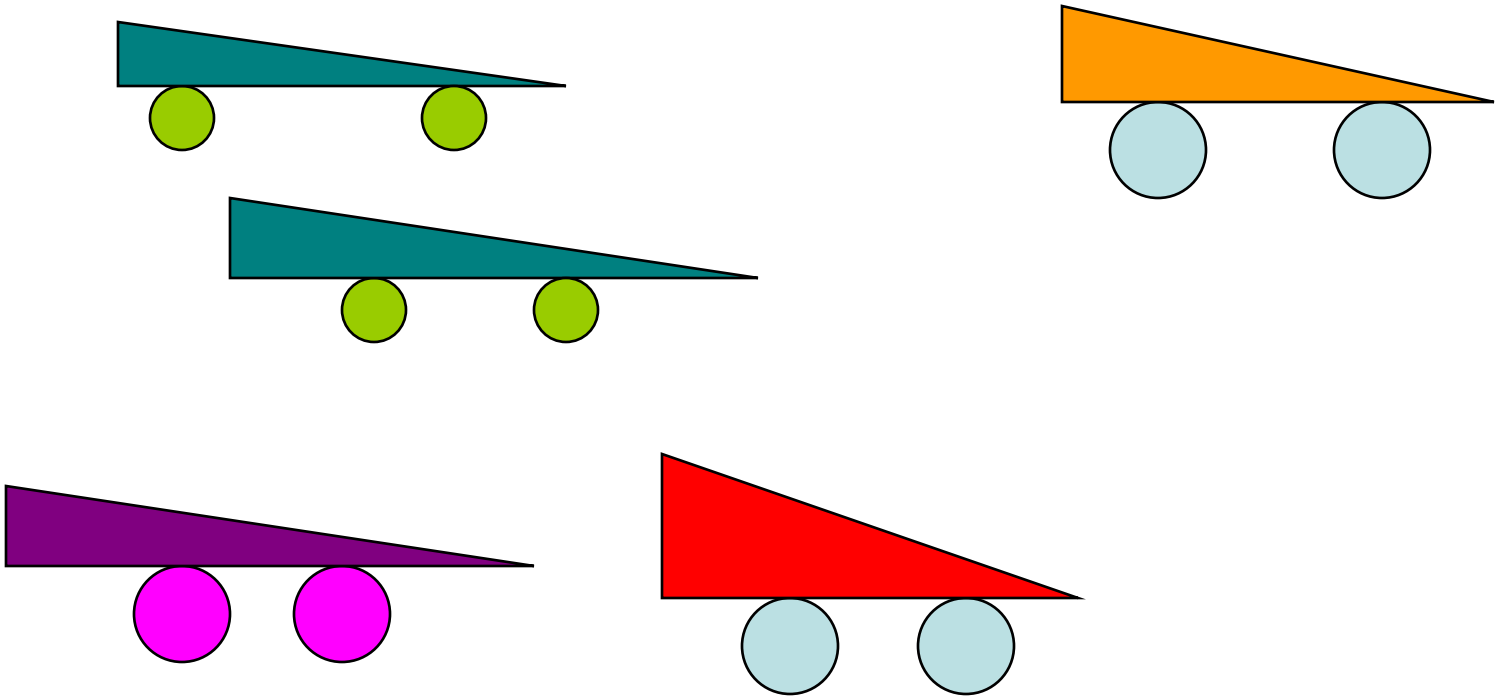
New Population: Generation 2



Generation 3



Generation 4, etc …



GA for solving Numerical Problem

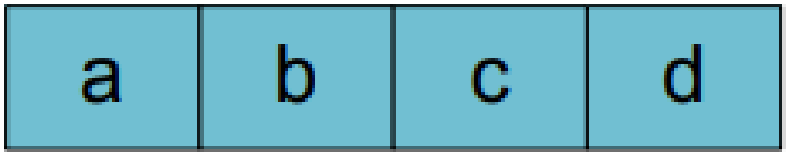
- Suppose there is equality
- $a + 2b + 3c + 4d = 30$,
- Use genetic algorithm to find the value of a, b, c, and d that satisfy the above equation.

Objective function

- Formulate the objective function for this problem the objective is minimizing the value of function f(x) where
- Min f(x) = ((a + 2b + 3c + 4d) - 30).
- Number of variables: 4 (a,b,c,d)

GA for solving Numerical Problem cont..

- Structure of Chromosome



- To speed up the computation, we can restrict that the values of variables a, b, c, and d are integers between 0 and 30.

GA for solving Numerical Problem cont..

- **Step 1: Initialization**
- For example we define the number of chromosomes in population are 6, then we generate random value of gene a, b, c, d for 6 chromosomes

Chromosome[1] = [a;b;c;d] = [12;05;23;08]

Chromosome[2] = [a;b;c;d] = [02;21;18;03]

Chromosome[3] = [a;b;c;d] = [10;04;13;14]

Chromosome[4] = [a;b;c;d] = [20;01;10;06]

Chromosome[5] = [a;b;c;d] = [01;04;13;19]

Chromosome[6] = [a;b;c;d] = [20;05;17;01]

GA for solving Numerical Problem cont..

- **Step 2. Evaluation**
- We compute the objective function value for each chromosome produced in initialization step:

F_obj[1] = Abs((12 + 2*05 + 3*23 + 4*08) - 30)
= Abs((12 + 10 + 69 + 32) - 30)
= Abs(123 - 30)
= 93

F_obj[2] = Abs((02 + 2*21 + 3*18 + 4*03) - 30)
= Abs((02 + 42 + 54 + 12) - 30)
= Abs(110 - 30)
= 80

F_obj[3] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
= Abs((10 + 08 + 39 + 56) - 30)
= Abs(113 - 30)
= 83

F_obj[4] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
= Abs((20 + 02 + 30 + 24) - 30)
= Abs(76 - 30)
= 46

F_obj[5] = Abs((01 + 2*04 + 3*13 + 4*19) - 30)
= Abs((01 + 08 + 39 + 76) - 30)
= Abs(124 - 30)
= 94

F_obj[6] = Abs((20 + 2*05 + 3*17 + 4*01) - 30)
= Abs((20 + 10 + 51 + 04) - 30)

GA for solving Numerical Problem cont..

$$\text{Fitness}[1] = 1 / (1+\text{F\_obj}[1]) = 1 / 94 = 0.0106$$

$$\text{Fitness}[2] = 1 / (1+\text{F\_obj}[2]) = 1 / 81 = 0.0123$$

$$\text{Fitness}[3] = 1 / (1+\text{F\_obj}[3]) = 1 / 84 = 0.0119$$

$$\text{Fitness}[4] = 1 / (1+\text{F\_obj}[4]) = 1 / 47 = 0.0213$$

$$\text{Fitness}[5] = 1 / (1+\text{F\_obj}[5]) = 1 / 95 = 0.0105$$

$$\text{Fitness}[6] = 1 / (1+\text{F\_obj}[6]) = 1 / 56 = 0.0179$$

$$\text{Total} = 0.0106 + 0.0123 + 0.0119 + 0.0213 + 0.0105 + 0.0179 = 0.0845$$

- To compute fitness probability we must compute the fitness of each chromosome.
- To avoid divide by zero problem, the value of F_obj is added by 1.

GA for solving Numerical Problem cont..

- **Step 3. Selection**
- The fittest chromosomes have higher probability to be selected for the next generation.
-

The probability for each chromosomes is formulated by: **P**[i] = **Fitness**[i] / **Total**

P[1] = 0.0106 / 0.0845
= 0.1254

P[2] = 0.0123 / 0.0845
= 0.1456

P[3] = 0.0119 / 0.0845
= 0.1408

P[4] = 0.0213 / 0.0845
= 0.2521

P[5] = 0.0105 / 0.0845
= 0.1243

P[6] = 0.0179 / 0.0845
= 0.2118

GA for solving Numerical Problem cont..

- From the probabilities above we can see that Chromosome 4 that has the highest fitness, this chromosome has highest probability to be selected for next generation chromosomes. For the selection process we use roulette wheel, for that we should compute the cumulative probability values:

$$\mathbf{C}[1] = 0.1254$$
$$\mathbf{C}[2] = 0.1254 + 0.1456 = 0.2710$$
$$\mathbf{C}[3] = 0.1254 + 0.1456 + 0.1408 = 0.4118$$
$$\mathbf{C}[4] = 0.1254 + 0.1456 + 0.1408 + 0.2521 = 0.6639$$
$$\mathbf{C}[5] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 = 0.7882$$
$$\mathbf{C}[6] = 0.1254 + 0.1456 + 0.1408 + 0.2521 + 0.1243 + 0.2118 = 1.0$$

GA for solving Numerical Problem cont..

- Having calculated the cumulative probability of selection process using roulette-wheel can be done. The process is to generate random number R in the range 0-1 as follows.

$$\mathbf{R}[1] = 0.201$$
$$\mathbf{R}[2] = 0.284$$
$$\mathbf{R}[3] = 0.099$$
$$\mathbf{R}[4] = 0.822$$
$$\mathbf{R}[5] = 0.398$$
$$\mathbf{R}[6] = 0.501$$

GA for solving Numerical Problem cont..

- If random number R[1] is greater than C[1] and smaller than C[2] then select Chromosome[2] as a chromosome in the new population for next generation:

$$\mathbf{NewChromosome}[1] = \mathbf{Chromosome}[2]$$
$$\mathbf{NewChromosome}[2] = \mathbf{Chromosome}[3]$$
$$\mathbf{NewChromosome}[3] = \mathbf{Chromosome}[1]$$
$$\mathbf{NewChromosome}[4] = \mathbf{Chromosome}[6]$$
$$\mathbf{NewChromosome}[5] = \mathbf{Chromosome}[3]$$
$$\mathbf{NewChromosome}[6] = \mathbf{Chromosome}[4]$$

GA for solving Numerical Problem cont..

- Chromosomes in the population thus became:

Chromosome[1] = [02;21;18;03]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;08]
Chromosome[4] = [20;05;17;01]
Chromosome[5] = [10;04;13;14]
Chromosome[6] = [20;01;10;06]

GA for solving Numerical Problem cont..

- In this example, we use one-cut point, i.e. randomly select a position in the parent chromosome then exchanging sub-chromosome.
- Parent chromosome which will mate is randomly selected and the number of mate Chromosomes is controlled using crossover_rate (ρc) parameters.
- Chromosome k will be selected as a parent if R[k] <ρc.
- Suppose we set that the crossover rate is 25%, then Chromosome number k will be selected for crossover if random generated value for Chromosome k below 0.25.

GA for solving Numerical Problem cont..

$$R[1] = 0.191$$
$$R[2] = 0.259$$
$$R[3] = 0.760$$
$$R[4] = 0.006$$
$$R[5] = 0.159$$
$$R[6] = 0.340$$

The process is as follows:

- First we generate a random number R as the number of population
- For random number R above, parents are Chromosome[1], Chromosome[4] and Chromosome[5] will be selected for crossover.

$$\text{Chromosome}[1] \asymp \text{Chromosome}[4]$$
$$\text{Chromosome}[4] \asymp \text{Chromosome}[5]$$
$$\text{Chromosome}[5] \asymp \text{Chromosome}[1]$$

GA for solving Numerical Problem cont..

- After chromosome selection, the next process is determining the position of the crossover point.
- This is done by generating random numbers between 1 to (length of Chromosome – 1).
- In this case, generated random numbers should be between 1 and 3. After we get the crossover point, parents Chromosome will be cut at crossover point and its genes will be interchanged. For example we generated 3 random number and we get:
- C[1] = 1
- C[2] = 1
- C[3] = 2

GA for solving Numerical Problem cont..

- Then for first crossover, second crossover and third crossover, parent's genes will be cut at gene number 1, gene number 1 and gene number 3 respectively, e.g.

$$\mathbf{Chromosome}[1] = \mathbf{Chromosome}[1] \times \mathbf{Chromosome}[4]$$
$$= [02;21;18;03] \times [20;05;17;01]$$
$$= [02;05;17;01]$$
$$\mathbf{Chromosome}[4] = \mathbf{Chromosome}[4] \times \mathbf{Chromosome}[5]$$
$$= [20;05;17;01] \times [10;04;13;14]$$
$$= [20;04;13;14]$$

GA for solving Numerical Problem cont..

$$\mathbf{Chromosome}[5] = \mathbf{Chromosome}[5] \prec \mathbf{Chromosome}[1]$$
$$= [10;04;13;14] \prec [02;21;18;03]$$
$$= [10;04;18;03]$$

Thus Chromosome population after experiencing a crossover process:

$\mathbf{Chromosome}[1] = [02;05;17;01]$

$\mathbf{Chromosome}[2] = [10;04;13;14]$

$\mathbf{Chromosome}[3] = [12;05;23;08]$

$\mathbf{Chromosome}[4] = [20;04;13;14]$

$\mathbf{Chromosome}[5] = [10;04;18;03]$

$\mathbf{Chromosome}[6] = [20;01;10;06]$

GA for solving Numerical Problem cont..

- Step 5. Mutation
- Number of chromosomes that have mutations in a population is determined by the mutation_rate parameter.
- Mutation process is done by replacing the gene at random position with a new value.
- The process is as follows.
  - First we must calculate the total length of gene in the population.
  - In this case the total length of gen is total_gen = number_of_gen_in_Chromosome * number of population = 4 * 6 = 24
- Mutation process is done by generating a random integer between 1 and total_gen (1 to 24). If generated random number is smaller than mutation_rate(ρm) variable then marked the position of gene in chromosomes.
- Suppose we define ρm 10%, it is expected that 10% (0.1) of total_gen in the population that will be mutated: number of mutations = 0.1 * 24 = 2.4 ≈ 2

GA for solving Numerical Problem cont..

- Suppose generation of random number yield 12 and 18 then the chromosome which have mutation are Chromosome number 3 gene number 4 and Chromosome 5 gene number 2. The value of mutated genes at mutation point is replaced by random number between 0-30. Suppose generated random number are 2 and 5 then Chromosome composition after mutation are:

Chromosome[1] = [02;05;17;01]
Chromosome[2] = [10;04;13;14]
Chromosome[3] = [12;05;23;**02**]
Chromosome[4] = [20;04;13;14]
Chromosome[5] = [10;**05**;18;03]
Chromosome[6] = [20;01;10;06]

GA for solving Numerical Problem cont..

- Finishing mutation process then we have one iteration or one generation of the genetic algorithm. We can now evaluate the objective function after one generation:

Chromosome[1] = [02;05;17;01]
F_obj[1] = Abs((02 + 2*05 + 3*17 + 4*01) - 30)
= Abs((2 + 10 + 51 + 4) - 30)
= Abs(67 - 30)
= 37
Chromosome[2] = [10;04;13;14]
F_obj[2] = Abs((10 + 2*04 + 3*13 + 4*14) - 30)
= Abs((10 + 8 + 33 + 56) - 30)
= Abs(107 - 30)
= 77
Chromosome[3] = [12;05;23;02]
F_obj[3] = Abs((12 + 2*05 + 3*23 + 4*02) - 30)
= Abs((12 + 10 + 69 + 8) - 30)
= Abs(87 - 30)
= 47
Chromosome[4] = [20;04;13;14]
F_obj[4] = Abs((20 + 2*04 + 3*13 + 4*14) - 30)
= Abs((20 + 8 + 39 + 56) - 30)
= Abs(123 - 30)
= 93
Chromosome[5] = [10;05;18;03]
F_obj[5] = Abs((10 + 2*05 + 3*18 + 4*03) - 30)
= Abs((10 + 10 + 54 + 12) - 30)
= Abs(86 - 30)
= 56
Chromosome[6] = [20;01;10;06]
F_obj[6] = Abs((20 + 2*01 + 3*10 + 4*06) - 30)
= Abs((20 + 2 + 30 + 24) - 30)
= Abs(76 - 30)
= 46

GA for solving Numerical Problem cont..

- From the evaluation of new Chromosome we can see that the objective function is decreasing, this means that we have better Chromosome or solution compared with previous Chromosome generation. New Chromosomes for next iteration are:

$$\mathbf{Chromosome}[1] = [02;05;17;01]$$
$$\mathbf{Chromosome}[2] = [10;04;13;14]$$
$$\mathbf{Chromosome}[3] = [12;05;23;02]$$
$$\mathbf{Chromosome}[4] = [20;04;13;14]$$
$$\mathbf{Chromosome}[5] = [10;05;18;03]$$
$$\mathbf{Chromosome}[6] = [20;01;10;06]$$

GA for solving Numerical Problem cont..

- These new Chromosomes will undergo the same process as the previous generation of Chromosomes such as evaluation, selection, crossover and mutation and at the end it produce new generation of Chromosome for the next iteration.
- This process will be repeated until a predetermined number of generations.
- For this example, after running 50 generations, best chromosome is obtained:
- Chromosome = [07; 05; 03; 01]
- This means that: a = 7, b = 5, c = 3, d = 1
- If we use the number in the problem equation: a + 2b + 3c + 4d = 30 7 + (2 * 5) + (3 * 3) + (4 * 1) = 30
- We can see that the value of variable a, b, c and d generated by genetic algorithm can satisfy that equality.

Evolutionary Algorithms

- Selection
  - The survival of the fittest, which means the highest quality chromosomes and/characteristics will stay within the population
  - Motivation is to preserve the best (make multiple copies) and eliminate the worst
- Crossover
  - Create new solutions by considering more than one individual
  - The recombination of two parent chromosomes (solutions) by exchanging part of one chromosome with a corresponding part of another so as to produce offsprings (new solutions)
  - Search for new and hopefully better solutions
- Mutation
  - The change of part of a chromosome (a bit or several bits) to generate new genetic characteristics
  - Keep diversity in the population

Concept of Exploration vs Exploitation

- Exploration
  - Search for promising solutions
  - Generate solutions with enough diversity and far from the current solutions
  - Mutation operators
  - The search is typically on a global scale
- Exploitation
  - Preferring the good solutions
  - Generate new solutions that are better than existing solutions
  - Crossover and Selection operator
  - This process is typically local
- Excessive exploration – Random search.
- Excessive exploitation – Premature convergence

Evolutionary Algorithms

Good evolutionary algorithm