

Course 395: Machine Learning – Lectures

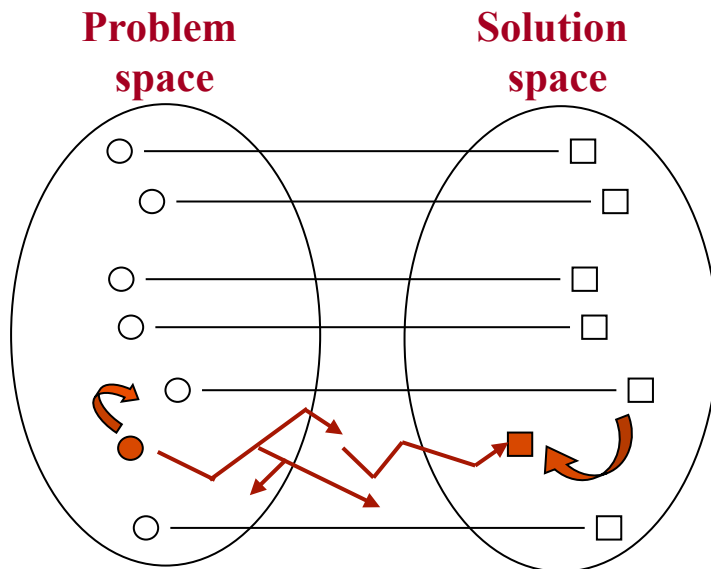
- Lecture 1-2: Concept Learning (*M. Pantic*)
- Lecture 3-4: Decision Trees & CBC Intro (*M. Pantic & S. Petridis*)
- Lecture 5-6: Evaluating Hypotheses (*S. Petridis*)
- Lecture 7-8: Artificial Neural Networks I (*S. Petridis*)
- Lecture 9-10: Artificial Neural Networks II (*S. Petridis*)
- Lecture 11-12: Artificial Neural Networks III (*S. Petridis*)
- Lecture 13-14: : Instance Based Learning & Genetic Algorithms (*M. Pantic*)

Instance Based Learning – Lecture Overview

- Lazy learning
- K-Nearest Neighbour learning
- Locally weighted regression
- Advantages and disadvantages of lazy learning
- Genetic algorithms
 - Representation: Chromosome, Population
 - Evolution: Selection, Inheritance, Crossover and Mutation
 - Beam Search


Eager vs. Lazy Learning

- Eager learning methods construct general, explicit description of the target function based on the provided training examples.
- Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.



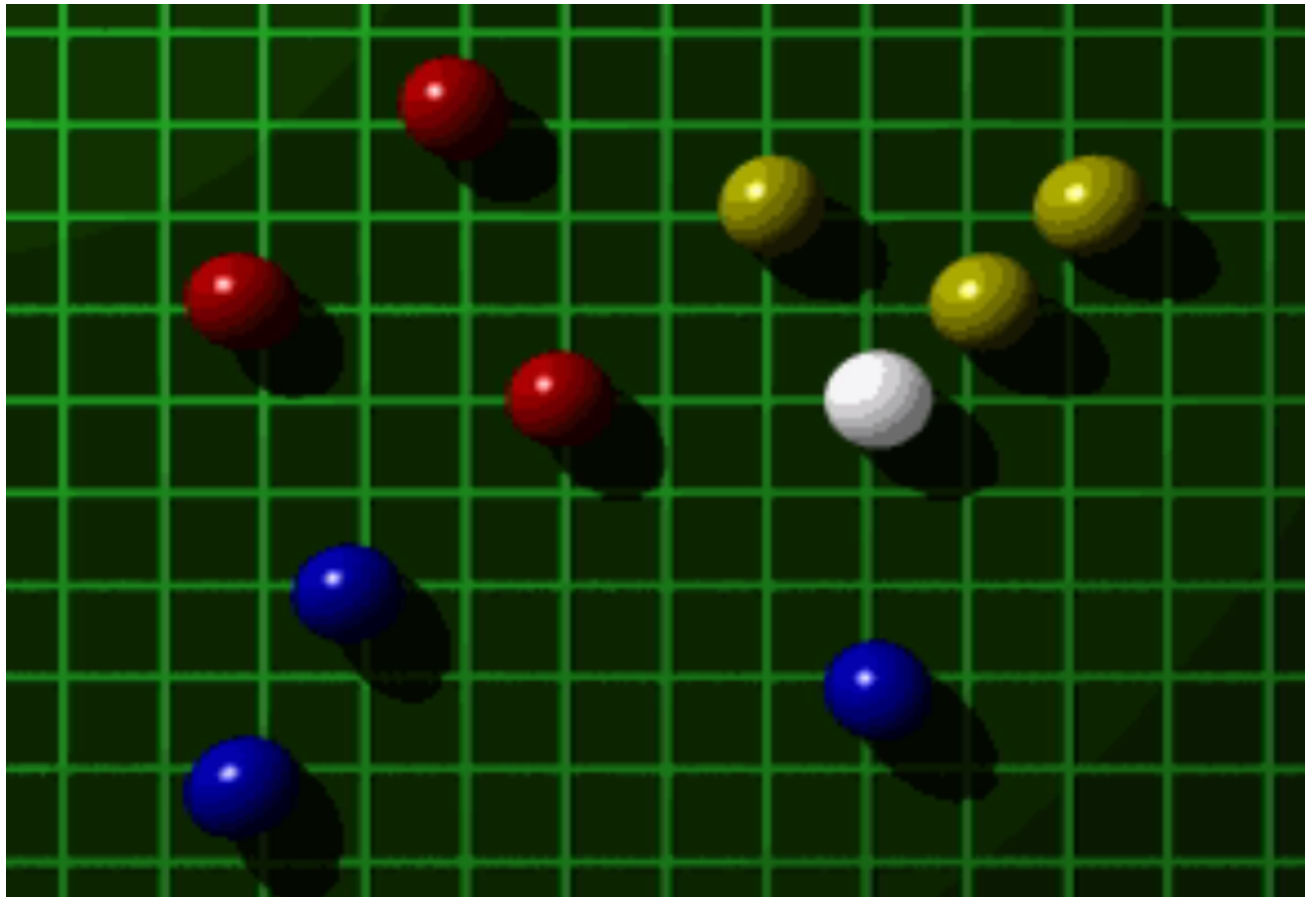
1. Search the memory for similar instances
2. Retrieve the related solutions
- 3. Adapt the solutions to the current instance**
4. Assign the value of the target function estimated for the current instance

Eager vs. Lazy Learning

- Eager learning methods construct general, explicit description of the target function based on the provided training examples.
 - Lazy learning methods simply store the data and generalizing beyond these data is postponed until an explicit request is made.
 - Lazy learning methods can construct a different approximation to the target function for each encountered query instance.
 - Eager learning methods use the same approximation to the target function, which must be learned based on training examples and before input queries are observed.
-  Lazy learning is very suitable for complex and incomplete problem domains, where a complex target function can be represented by a collection of less complex local approximations.

k -Nearest Neighbour Learning

The main idea behind k -NN learning is so-called *majority voting*.



k-Nearest Neighbour Learning

- Given the target function $V: X \rightarrow C$ and a set of n already observed instances (x_i, c_j) , where $x_i \in X, i = [1..n], c_j \in C, j = [1..m], V(x_i) = c_j$, k -NN algorithm will decide the class of the new query instance x_q based on its k nearest neighbours (previously observed instances) $x_r, r = [1..k]$, in the following way:

$$V(x_q) \leftarrow c_l \in C \leftrightarrow (\forall j \neq l) \sum_r E(c_l, V(x_r)) > \sum_r E(c_j, V(x_r)) \text{ where}$$

$$E(a, b) = 1 \text{ if } a = b \text{ and } E(a, b) = 0 \text{ if } a \neq b$$

- The nearest neighbours of a query instance x_q are usually defined in terms of standard Euclidean distance:

$$d_e(x_i, x_q) = \sqrt{\sum_g (a_g(x_i) - a_g(x_q))^2}$$

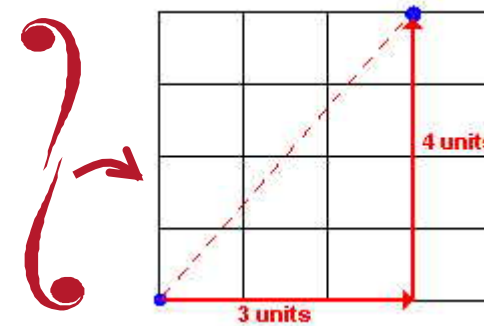
where the instances $x_i, x_q \in X$ are described with a set of $g = [1..p]$ arguments a_g

k-Nearest Neighbour Learning

- Distance between two instances $x_i, x_q \in X$, described with a set of $g = [1..p]$ arguments a_g , can be calculated as:

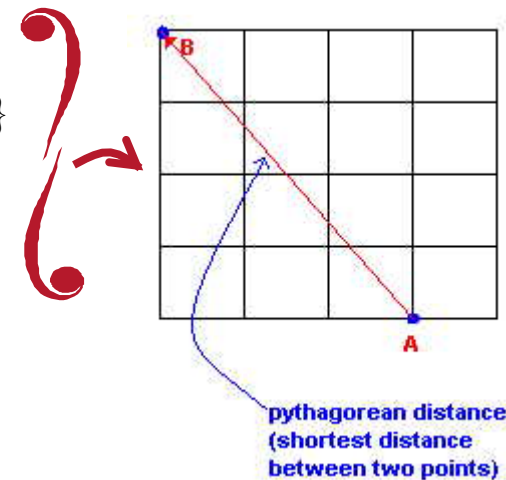
- City-block (Manhattan) distance (L1-norm):

$$d_e(x_i, x_q) = \sum_g |a_g(x_i) - a_g(x_q)|$$



- Euclidean distance (L2-norm):

$$d_e(x_i, x_q) = \sqrt{\sum_g (a_g(x_i) - a_g(x_q))^2}$$

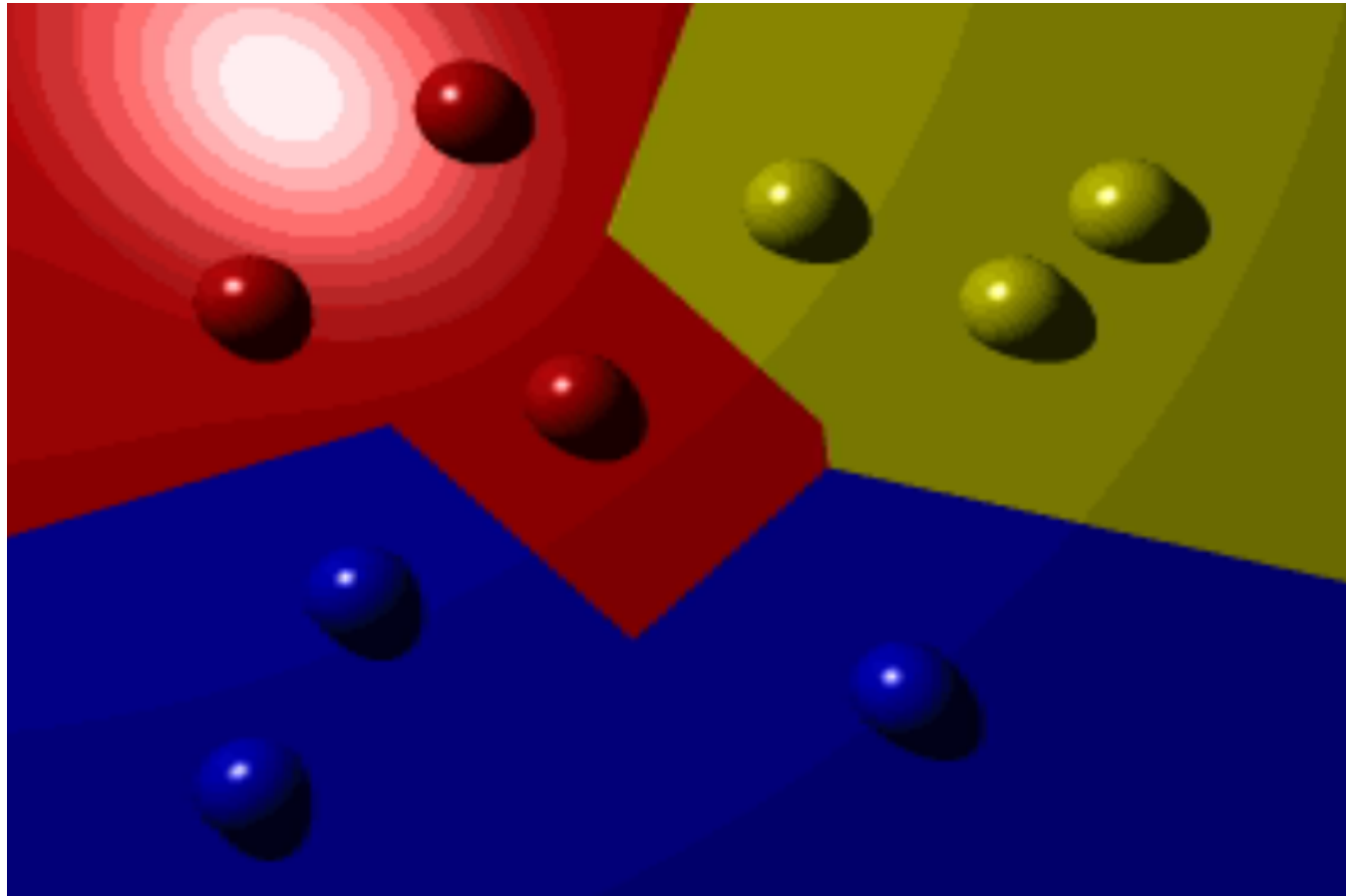


- Chebyshev distance (L-infinity-norm):

$$d_e(x_i, x_q) = \max_g |a_g(x_i) - a_g(x_q)|$$

k -Nearest Neighbour Learning

For $k = 1$, the decision surface is a set of polygons (*Voronoi diagram*), completely defined by previously observed instances (training examples).



k-Nearest Neighbour Learning

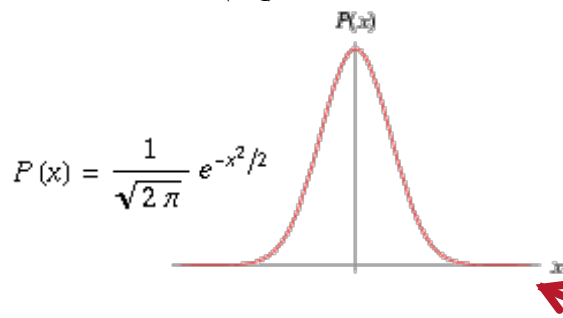
- The nearest neighbours (previously observed instances) $x_r, r = [1..k]$, of a query instance x_q are defined based on a distance $d(x_r, x_q)$ such as the Euclidian distance.
- A refinement of the k -NN algorithm: assign a weight w_r to each neighbour x_r of the query instance x_q based on the distance $d(x_r, x_q)$ such that $(d(x_r, x_q) \downarrow \leftrightarrow w_r \uparrow)$
- **Distance-weighted k -NN algorithm:** Given the target function $V: X \rightarrow C$ and a set of n already observed instances (x_i, c_j) , where $x_i \in X, i = [1..n], c_j \in C, j = [1..m], V(x_i) = c_j$, distance weighted k -NN algorithm will decide the class of the query instance x_q based on its k nearest neighbours $x_r, r = [1..k]$, in the following way:

$$V(x_q) \leftarrow c_l \in C \leftrightarrow (\forall j \neq l) \sum_r w_r \cdot E(c_l, V(x_r)) > \sum_r w_r \cdot E(c_j, V(x_r)) \text{ where}$$

$$E(a, b) = 1 \text{ if } a = b, E(a, b) = 0 \text{ if } a \neq b, \text{ and}$$

$$w_r = 1 / (d(x_r, x_q))^2$$

any other measure favouring the votes of nearby neighbours will do (e.g. Gaussian distribution)



k -Nearest Neighbour Learning: Remarks

- By the distance-weighted k -NN algorithm, the value of k is of minor importance as distant examples will have very small weight and will not greatly affect the value of $V(x_q)$.
- If $k = n$, where n is the total number of previously observed instances, we call the algorithm a global method. Otherwise, if $k < n$, the algorithm is called a local method.
- **Advantage** – Distance-weighted k -NN algorithm is robust to noisy training data: it calculates $V(x_q)$ based on a weighted $V(x_r)$ values of *all* k nearest neighbours x_r , effectively smoothing out the impact of isolated noisy training data.
- **Disadvantage** – All k -NN algorithms calculate the distance between instances based on *all* attributes → if there are many irrelevant attributes, instances that belong together may still be distant from one another.
- **Remedy** – weight each attribute differently when calculating the distance between two instances

Locally Weighted Regression

- Locally weighted regression is a most general form of k -NN learning. It constructs an explicit approximation to target function V that fits the training examples in the local neighbourhood of the query instance x_q .
- *Local* – V is approximated based only on the data (neighbours) near x_q .
Weighted – contribution of a datum is weighted by its distance from x_q .
Regression – refers to the problem of approximating a real-valued target function.
- **Locally weighted regression:**
target function: $V: X \rightarrow C$,
target function approximation near x_q : $V'(x_q) = w_0 + w_1 a_1(x_q) + \dots + w_n a_n(x_q)$,
where $x_q \in X$ is described with a set of $g = [1..n]$ attributes a_j
training examples: set of k nearest neighbours x_r , $r = [1..k]$, of the query instance x_q ,
learning problem: learn the most optimal weights w given the set of training examples
learning algorithm (distance-weighted gradient descent training rule):
$$\Delta w_j = \eta \cdot \sum_r \{ \mathcal{K}(d(x_r, x_q)) \cdot (V(x_r) - V'(x_r)) \cdot a_j(x_r) \}$$

↪ function of distance that determines weights of x_r

Instance Based Learning – Practice

- Tom Mitchell's book –chapter 8
- Relevant exercises from chapter 8: 8.1, 8.2, 8.3

Instance Based Learning – Lecture Overview

- Lazy learning
- K-Nearest Neighbour learning
- Locally weighted regression
- Advantages and disadvantages of lazy learning

➤ Genetic algorithms

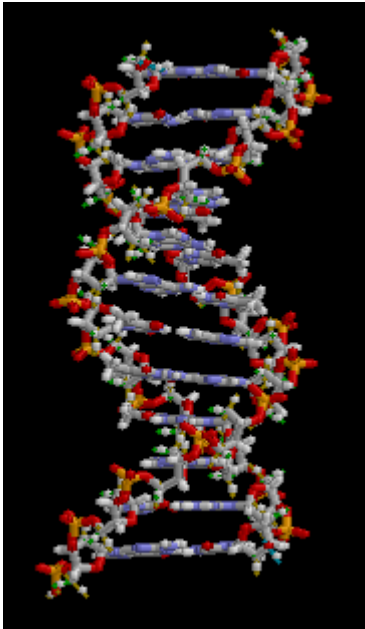
- Representation: Chromosome, Population
- Evolution: Selection, Inheritance, Crossover and Mutation
- Beam Search

Genetic Algorithms

- Genetic Algorithms are a learning method inspired by evolutionary biology.
- Genetic Algorithms are implemented as a computer simulation of the evolution process – a population of candidate solutions (hypotheses) evolves toward better solutions by repeatedly mutating and recombining the best members (hypotheses) of the population.
- *Prototypical Genetic Algorithm:*

1. Choose initial *population*.
2. Evaluate the individual *fitness* of individuals in the population.
3. Repeat:
 - Select fittest individuals to reproduce;
 - Breed new generation through *crossover* and *mutation*; give birth to offspring;
 - Evaluate the individual fitness of offspring;
 - Replace worse ranked part of population with offspring;Until terminating condition.

Evolutionary Biology: Basics



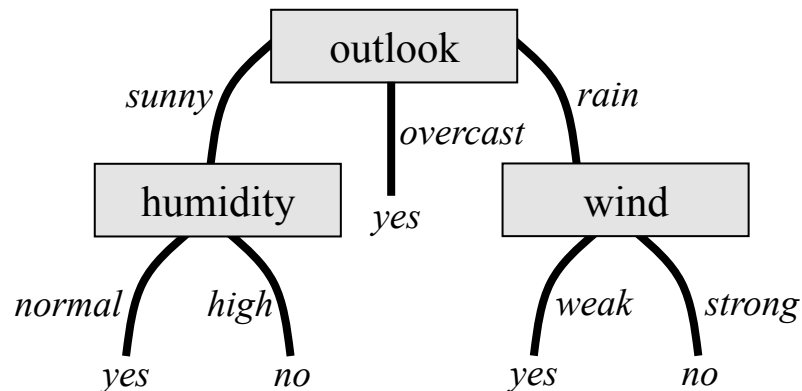
- *Chromosome* is a long, continuous piece of DNA (deoxyribonucleic acid), containing genetic instructions for a biological development of a cell.
- Humans have 46 chromosomes, i.e., 23 pairs of chromosomes. We inherit 23 chromosomes from mother and 23 from father. The matching chromosomes of father and mother can exchange small parts of themselves (*crossover*), creating new chromosomes not inherited from a parent. Each chromosome can undergo changes in its DNA due to copying errors of the genetic material or due to exposure to radiation, chemicals, or viruses (*mutation*).
- Chromosomes define traits ranging from hair colour to disease predisposition. Genetic disorders like Down Syndrome result from gain or loss of a chromosome.

Genetic Algorithms: Chromosome, Population

- *Chromosome* is a set of parameters which defines a candidate solution to the target problem.
- *Population* is a set of chromosomes representing a space of candidate solutions to the target problem.
- A standard representation of a chromosome is an array of bits (e.g., a byte 01001101). *Advantage*: Due to their fixed size, byte-based representations are easily aligned. Other data structures (of a fixed length or not) can be used as well.
- Chromosome design **example 1**: *chromosome for value 15 would be 00001111*
Find $\max x \in [0, 255]$ such that $\{\forall y \neq x; x, y, x^2, y^2 \in [0, 255] \mid y^2 < x^2\}$.
- Chromosome design **example 2**: *chromosomes would be designed as ACBEDF*
Given the cities A, B, C, D, E, and F, solve the travelling salesman problem.

Genetic Algorithms: Chromosome, Population

- Chromosome design (example 3, Mitchell's book, p.59, pp. 252-253):
Find out when the tennis should be played given the weather forecast.



$outlook = \{sunny, overcast, rain\} \rightarrow 100, 010, 001, 011, \dots, 111, 000$

$humidity = \{normal, high\} \rightarrow 10, 01, 11, 00$

$wind = \{weak, strong\} \rightarrow 10, 01, 11, 00$

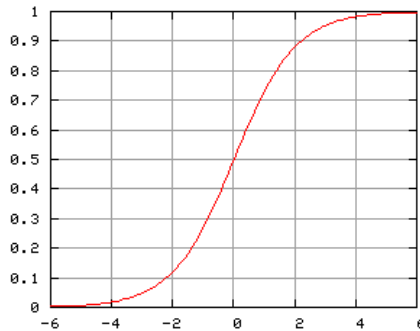
$PlayTennis = \{yes, no\} \rightarrow 10, 01, 11, 00$

<i>outlook</i>	<i>humidity</i>	<i>wind</i>	<i>Play Tennis</i>
100	10	11	10
011	11	10	10

 \equiv *chromosome design*

Genetic Algorithms: Selection of the Fittest

- *Fitness function* is defined as an objective function that quantifies the optimality of a solution (chromosome) to the target problem.
- Fitness Function f definition (example 1): *if $x^2 \in [0, 255] \rightarrow (x \uparrow \rightarrow f(x) \uparrow)$*
Find $\max x \in [0, 255]$ such that $\{\forall y \neq x; x, y, x^2, y^2 \in [0, 255] \mid y^2 < x^2\}$.
- Fitness Function f definition (example 2): *length(x) $\downarrow \rightarrow f(x) \uparrow$*
Given the cities A, B, C, D, E, and F, solve the travelling salesman problem.
- Fitness Function f definition (example 3) (Mitchell's book, p.59, pp. 252-253):
Find out when the tennis should be played given the weather forecast.



$$P(t) = \frac{1}{1 + e^{-t}}$$

$t = x$ (example 1)

$t = \text{correct}(h)$ (example 3)

correct(h) $\uparrow \rightarrow f(h) \uparrow$

where correct(h) is the percentage of all training examples correctly classified by h

Genetic Algorithms: Selection of the Fittest

- Prototypical Genetic Algorithm:

1. Choose initial *population*.
2. Evaluate the individual *fitness* of individuals in the population.
3. Repeat:
 - **Select** fittest individuals to reproduce;
 - Breed new generation through *crossover* and *mutation*; give birth to offspring;
 - Evaluate the individual fitness of offspring;
 - Replace worse ranked part of population with offspring;Until terminating condition.

- *Selection method* is procedure used to rate the fitness of individual solutions and select the fitter ones.

Genetic Algorithms: Selection of the Fittest

- Selection methods:

- ***Fitness Proportionate (Roulette Wheel) Selection:***

Evaluate the fitness f of all individuals h_i , $i, j = [1..n]$, select k of them with the likelihood $Pr(h_i)$:

$$Pr(h_i) = f(h_i) / \sum_j f(h_j).$$

- ***Tournament Selection:***

Choose k individuals to compete, evaluate their fitness, rank them, and assign probability $Pr(h_i)$ to each h_i :

$$h_i \text{ with rank } 1 \rightarrow Pr(h_i) = p,$$

$$h_i \text{ with rank } 2 \rightarrow Pr(h_i) = p \cdot (1-p),$$

$$h_i \text{ with rank } 3 \rightarrow Pr(h_i) = p \cdot (1-p)^2, \dots$$

- ***Rank Selection:***

Evaluate the fitness f of all individuals h_i , $i = [1..n]$, rank them, selects k of them having the largest fitness $f(h_i)$.

Genetic Algorithms: Genetic Operators

- Prototypical Genetic Algorithm:

1. Choose initial *population*.
2. Evaluate the individual *fitness* of individuals in the population.
3. Repeat:

 Select fittest individuals to reproduce;

 Breed new generation through *crossover* and *mutation*; give birth to offspring;

 Evaluate the individual fitness of offspring;

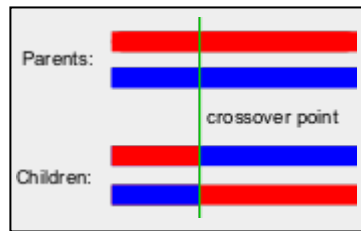
 Replace worse ranked part of population with offspring;

Until terminating condition.

- *Genetic operator* is a process used to maintain genetic diversity, which is necessary for successful evolution.
- Genetic operators: survival of the fittest (*selection*), reproduction (*crossover*), *inheritance*, and *mutation*.

Genetic Algorithms: Genetic Operators

- *Inheritance* is a genetic operator used to propagate problem solving genes of fittest chromosomes to the next generation of evolved solutions to the target problem.
- *Crossover* is a genetic operator used to vary the coding of chromosomes from one generation to the next.



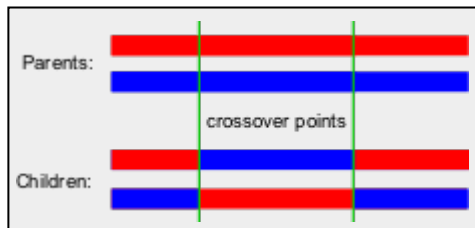
Mask: 111 0000000

➤ ***Single-point Crossover:***

A crossover point on the parent chromosome is selected.

(*Crossover mask* defines the position of the crossover point.)

All data beyond that point is swapped between two parents.



Mask: 111 000000 11111

➤ ***Two-point Crossover:***

Two crossover point on the parent chromosome are selected.

All data between these points is swapped between two parents.

➤ ***Uniform Crossover:***

Crossover mask is generated as a random bit string.

Bits are sampled uniformly from the two parents.

e.g. Mask: 1011001001

Genetic Algorithms: Genetic Operators

- *Inheritance* is a genetic operator used to propagate problem solving genes of fittest chromosomes to the next generation of evolved solutions to the target problem.
- *Crossover* is a genetic operator used to vary the coding of chromosomes from one generation to the next.
 - *Single-point Crossover*
 - *Two-point Crossover*
 - *Uniform Crossover*
- *Mutation* is a genetic operator used to maintain genetic diversity by triggering small random changes in the bits of a chromosome.
Procedure: A single bit is chosen at random and its value is changed.

11100101011011 \longrightarrow 11100100011011

Purpose: Allow the algorithm to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping evolution.

Mutation rate: A percentage of the population to be mutated.

Genetic Algorithms: Terminating Condition

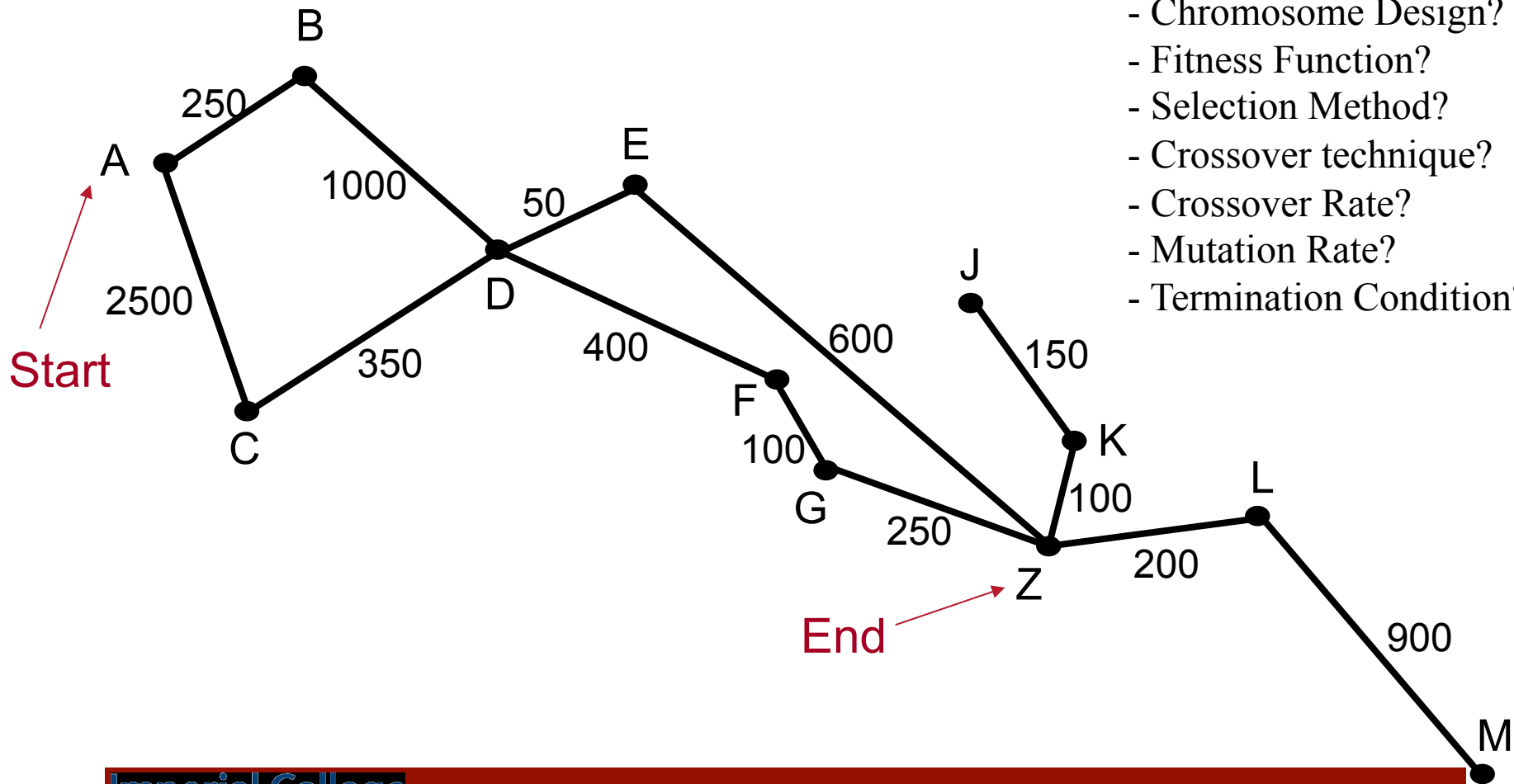
- Prototypical Genetic Algorithm:

1. Choose initial *population*.
2. Evaluate the individual *fitness* of individuals in the population.
3. Repeat:
 - Select fittest individuals to reproduce;
 - Breed new generation through *crossover* and *mutation*; give birth to offspring;
 - Evaluate the individual fitness of offspring;
 - Replace worse ranked part of population with offspring;Until terminating condition

- *Termination Condition* specifies when breeding novel generation populations of chromosomes will cease.
- Termination conditions: solution with a predefined target fitness is found, a certain number of generations is bred, allocated budget (computation time / money) reached, manual inspection, a combination of several conditions.

Genetic Algorithms: Example

Traveling Salesman Problem A → Z:



GA parameters:

- Chromosome Design?
- Fitness Function?
- Selection Method?
- Crossover technique?
- Crossover Rate?
- Mutation Rate?
- Termination Condition?

Genetic Algorithms: Example

GA parameters:

A	B	C	D	E	F	G	J	K	L	M	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
1	1	0	1	0	1	1	0	0	0	0	1

- **Chromosome Design:**
- **Fitness Function f :** based on the length of path h , $f(h) \equiv P(t) = 1 / (1 + e^t)$, $t = \text{length}(h)$
- **Selection Method:** e.g., rank selection method
- **Crossover Technique:** 2-point crossover, crossover mask \leftarrow 100000011111
- **Crossover Rate:** k , usually $\pm 60\%$
- **Mutation Rate:** m , usually very small $\pm 1\%$
- **Termination Condition:** e.g., $\text{length}(h) < 2000 m$

1. Choose initial *population*.
2. Evaluate the *fitness* of individuals in the population.
3. Repeat:
 - Select k best-ranking individuals to reproduce ($k \equiv$ crossover rate);
 - Generate offspring of k individuals through *crossover*; *mutate* $m\%$ of offspring;
 - Evaluate the individual fitness of offspring;
 - Replace k worse ranked part of population with offspring;Until terminating condition.

Genetic Algorithm (Mitchell)

GA(*Fitness*, *Fitness_threshold*, *p*, *r*, *m*)

Fitness: A function that assigns an evaluation score, given a hypothesis.

Fitness_threshold: A threshold specifying the termination criterion.

p: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

- Initialize population: $P \leftarrow$ Generate p hypotheses at random

- Evaluate: For each h in P , compute $Fitness(h)$

- While $(\max_h Fitness(h)) < Fitness_threshold$ do \rightarrow Termination Condition: A predefined target fitness

Create a new generation, P_S :

1. Select: Probabilistically select $(1-r)p$ members of P to add to P_S . The probability $Pr(h_i)$ of selecting hypothesis h_i from P is given by

$$Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^p Fitness(h_j)}$$

\rightarrow Fitness Proportionate Selection Method

2. Crossover: Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P , according to $Pr(h_i)$ given above. For each pair, (h_1, h_2) , produce two offspring by applying the Crossover operator. Add all offspring to P_S .

3. Mutate: Choose m percent of the members of P_S with uniform probability. For each, invert one randomly selected bit in its representation.

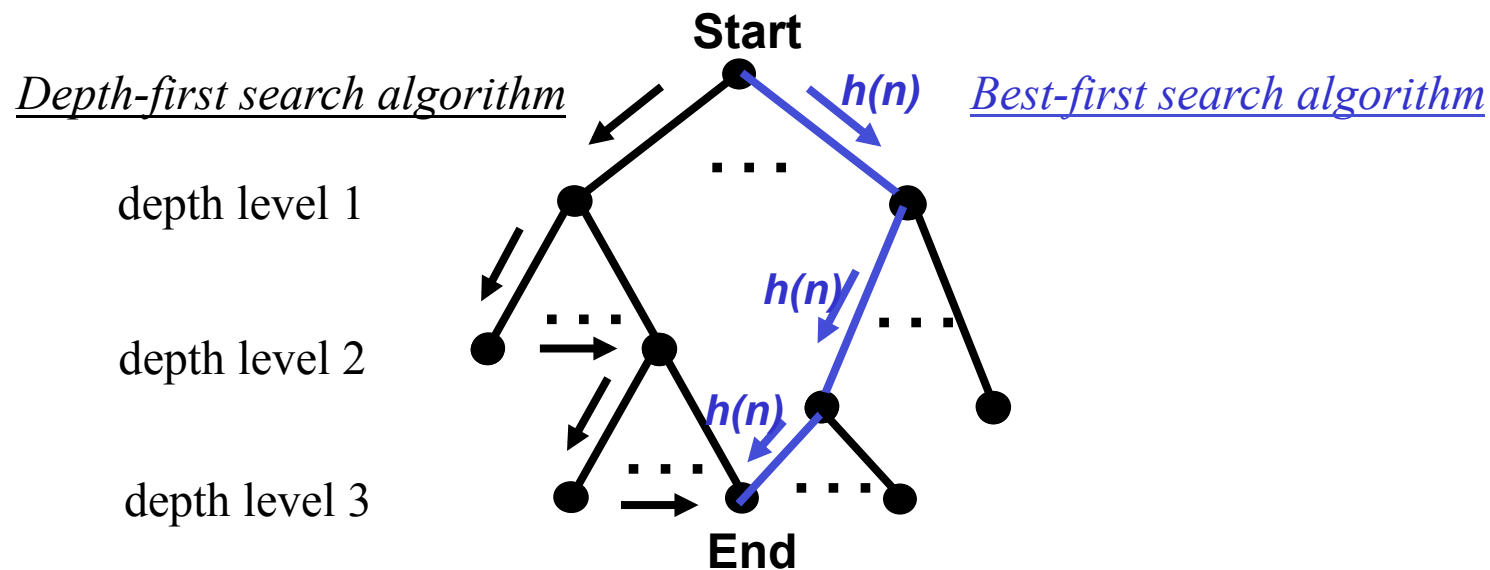
4. Update: $P \leftarrow P_S$.

5. Evaluate: for each h in P , compute $Fitness(h)$

- Return the hypothesis from P that has the highest fitness.

Beam Search

- Genetic Algorithms employ a (randomized) beam search method to seek a maximally fit hypothesis.
- *Beam Search* is a heuristic (informed) search algorithm that is an optimisation of best-first search .
- *Best-first Search* is a search algorithm that optimizes blind (uninformed) search by expanding the most promising node as measured by a specific heuristic function h .



Beam Search

- Genetic Algorithms employ a (randomized) beam search method to seek a maximally fit hypothesis.
- *Beam Search* is a heuristic (informed) search algorithm that is an optimisation of best-first search .
- *Best-first Search* is a search algorithm that optimizes blind (uninformed) search by expanding the most promising node as measured by a specific heuristic function h .
- Best-first Search evaluates at each depth level all n nodes using a heuristic function. Beam Search evaluates at each depth level only k nodes (k is the *beam width*).
 $\Rightarrow \{k \rightarrow n \Rightarrow \text{Beam Search} \rightarrow \text{Best-first Search}\}$
- Beam Search:
 - ❌ completeness (does the search strategy produce always a solution?)
 - ❌ optimality (does the search strategy produce always the best solution?)

Genetic Algorithms: Remarks

- Genetic Algorithms employ a (randomized) beam search method to seek a maximally fit hypothesis.
- *Best-first Search* is a search algorithm that optimizes blind (uninformed) search by expanding the most promising node as measured by a specific heuristic function h .
- Best-first Search evaluates at each depth level all n nodes using a heuristic function. Beam Search evaluates at each depth level only k nodes (k is the *beam width*).
 $\Rightarrow \{k \rightarrow n \Rightarrow \text{Beam Search} \rightarrow \text{Best-first Search}\}$
- Genetic Algorithms (case $k < n$, where k is the number of evaluated hypotheses):
 - ❌ completeness (does the search strategy produce always a solution?)
 - ❌ optimality (does the search strategy produce always the best solution?)
- Genetic Algorithms (case $k = n$, where k is the number of evaluated hypotheses):
 - ❌ completeness (does the search strategy produce always a solution?)
 - ❌ optimality (does the search strategy produce always the best solution?)

Genetic Algorithms: Remarks

- *Genetic Algorithms suffer from so-called crowding.*
Crowding occurs when a highly fit individual quickly reproduces, spreading its genes to a large part of the population.
Remedies: use different fitness functions / selection methods, increase the mutation rate, apply ‘fitness sharing’ to similar individuals, apply ‘cousin-based’ crossover.
- *Genetic Algorithms cannot handle well dynamic problem domains.*
Remedies (if the fitness function is still applicable, otherwise there is no remedy): increase the mutation rate when the solution quality drops, include novel randomly-generated elements into the population.
- *Genetic Algorithms cannot handle well right/wrong classification problems.*
Problem: too restrictive (single-valued) fitness function.
Remedies: ‘fuzzifying’ / ‘pseudo-randomising’ the fitness function
- Genetic Algorithms can often rapidly locate good solutions.
- When the evolving population are computer programs rather than bit strings, we refer to it as to *genetic programming*.

Instance Based Learning – Practice

- Tom Mitchell's book – chapter 9
- Relevant exercises from chapter 9: 9.1, 9.2, 9.3