Course 395: Machine Learning – Lectures

- Lecture 1-2: Concept Learning (*M. Pantic*)
- Lecture 3-4: Decision Trees & CBC Intro (*M. Pantic*)
- Lecture 5-6: Artificial Neural Networks (*THs*)
- Lecture 7-8: Instance Based Learning (*M. Pantic*)
- ➢ Lecture 9-10: Genetic Algorithms (*M. Pantic*)
- Lecture 11-12: Evaluating Hypotheses (*THs*)
- Lecture 13-14: Guest Lectures on ML Applications
- Lecture 15-16: Inductive Logic Programming (*S. Muggleton*)
- Lecture 17-18: Inductive Logic Programming (*S. Muggleton*)

Genetic Algorithms – Lecture Overview

- Genetic Algorithms
  - Genetic Representation: Chromosome and Population
  - Evolution: Selection of the Fittest
  - Evolution: Inheritance, Crossover and Mutation
  - Beam Search
- Genetic Algorithms: Remarks

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*:

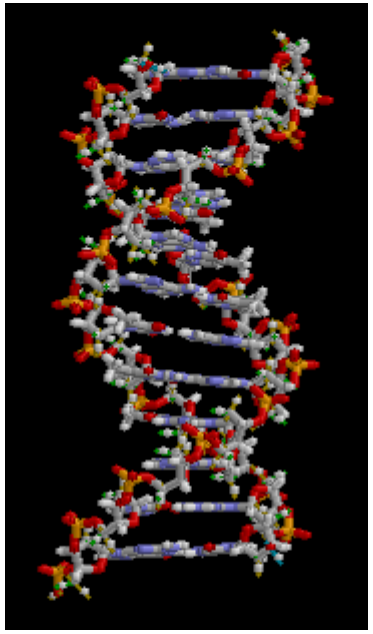
```
1. Choose initial population.
2. Evaluate the individual fitness of individuals in the population.
3. Repeat:
        Select fittest individuals to reproduce;
        Breed new generation through crossover and mutation; give birth to offspring;
        Evaluate the individual fitness of offspring;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

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*).

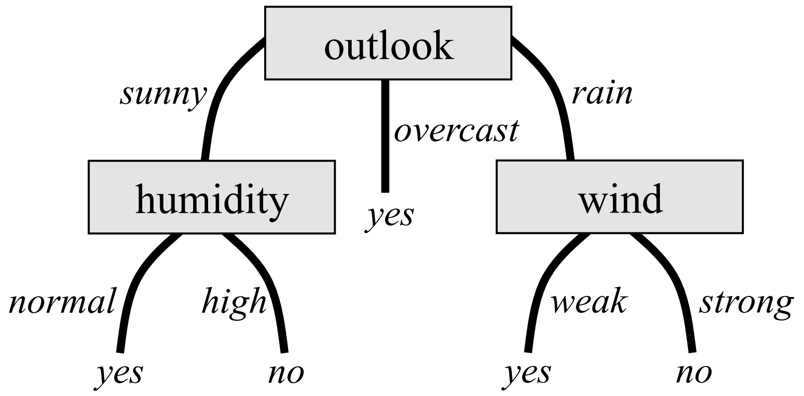
- Chromosomes define traits ranging from hair colour to disease predisposition.
  Genetic disorders like Down Syndrome result from gain or loss of a chromosome.

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,…, 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*

outlook	*humidity*	*wind*	*Play Tennis*
100	10	11	10
011	11	10	10

≡ ***chromosome design***

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.
```

- *Fitness function* is defined as an objective function that quantifies the optimality of a solution (chromosome) to the target problem.

- The actual definition of a fitness function is problem dependent.

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):
  Find max $x \in [0, 255]$ such that $\{\forall y \neq x;\ x, y, x^2, y^2 \in [0, 255] \mid y^2 < x^2\}$.

  if $x^2 \in [0,255] \rightarrow (x\uparrow \rightarrow f(x)\uparrow)$

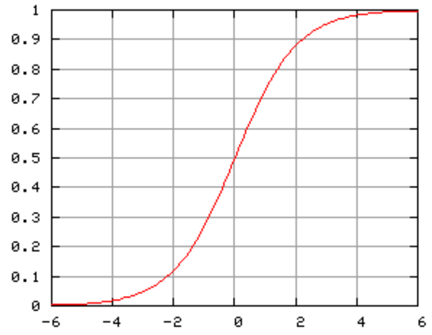
- Fitness Function *f* definition (example 2):
  Given the cities A, B, C, D, E, and F, solve the travelling salesman problem.

  length(x)↓ → f(x)↑

- 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.

  correct(h)↑ → f(h)↑

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



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

t = x *(example 1)*

t = correct(h) *(example 3)*

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, $i = [1..n]$, 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 with rank 1 $\rightarrow Pr(h_i) = p$,
    h_i with rank 2 $\rightarrow Pr(h_i) = p \cdot (1-p)$,
    h_i with rank 3 $\rightarrow Pr(h_i) = p \cdot (1-p)^2$, …

  - ***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.
```

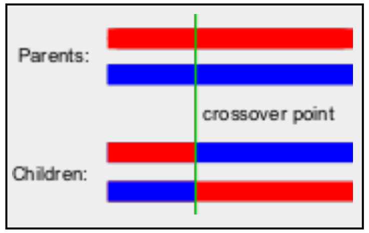
- *Genetic operator* is a process used to maintain genetic diversity, which is necessary for successful evolution.

- Genetic operators: survival of the fittest (*selection*), reproduction (*crossover*), *inheritance*, and *mutation*.

Genetic Algorithms: Genetic Operators

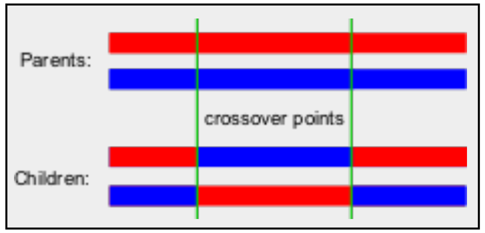
- *Inheritance* is a genetic operator used to propagate problem solving genes of fittest chromosomes to the next generation of evolved solutions to the target problem.

- *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.

  $$1110010\underline{1}011011 \longrightarrow 1110010\underline{0}011011$$

  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*:

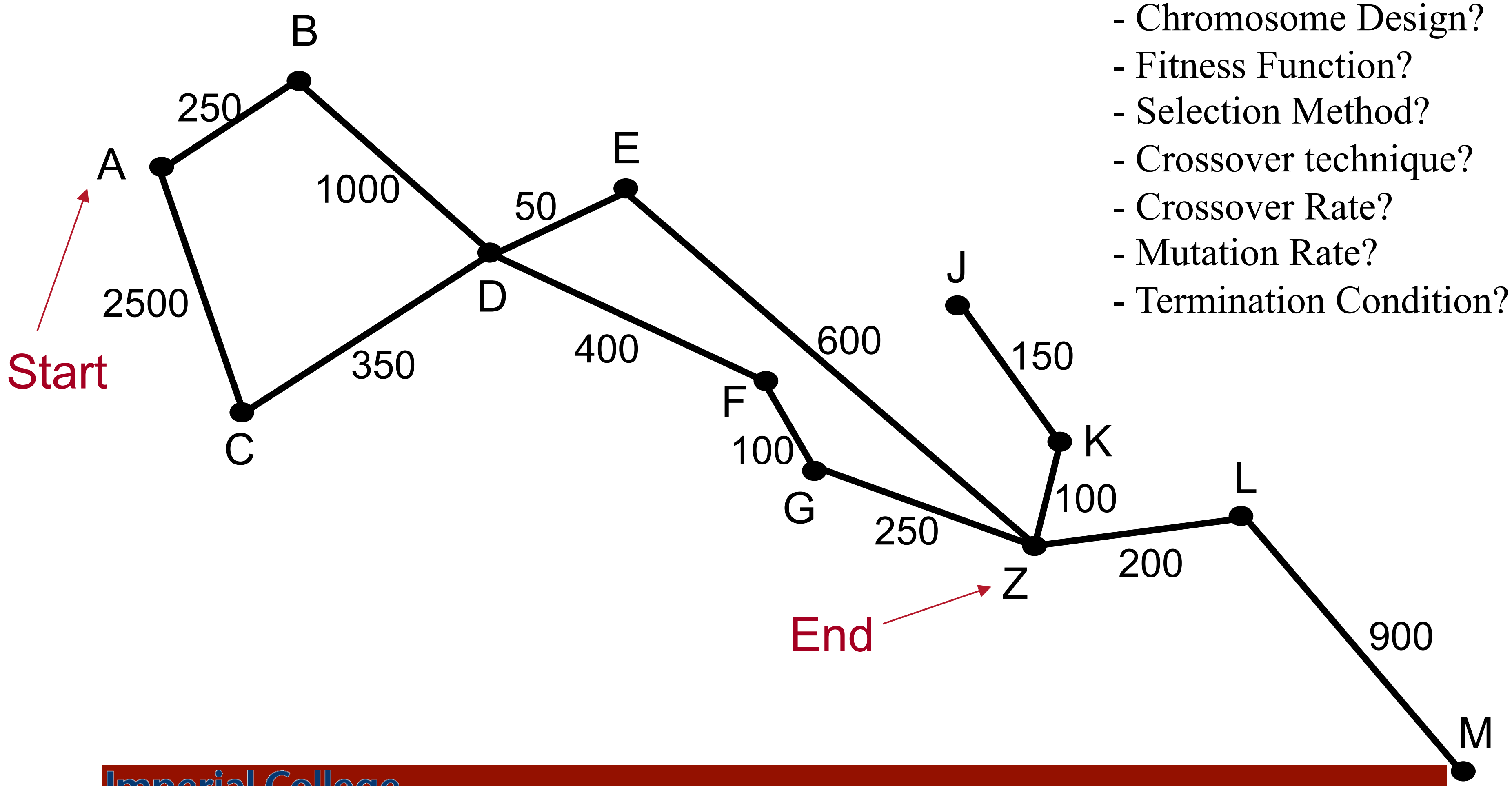
```
1. Choose initial population.
2. Evaluate the individual fitness of individuals in the population.
3. Repeat:
        Select fittest individuals to reproduce;
        Breed new generation through crossover and mutation; give birth to offspring;
        Evaluate the individual fitness of offspring;
        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:

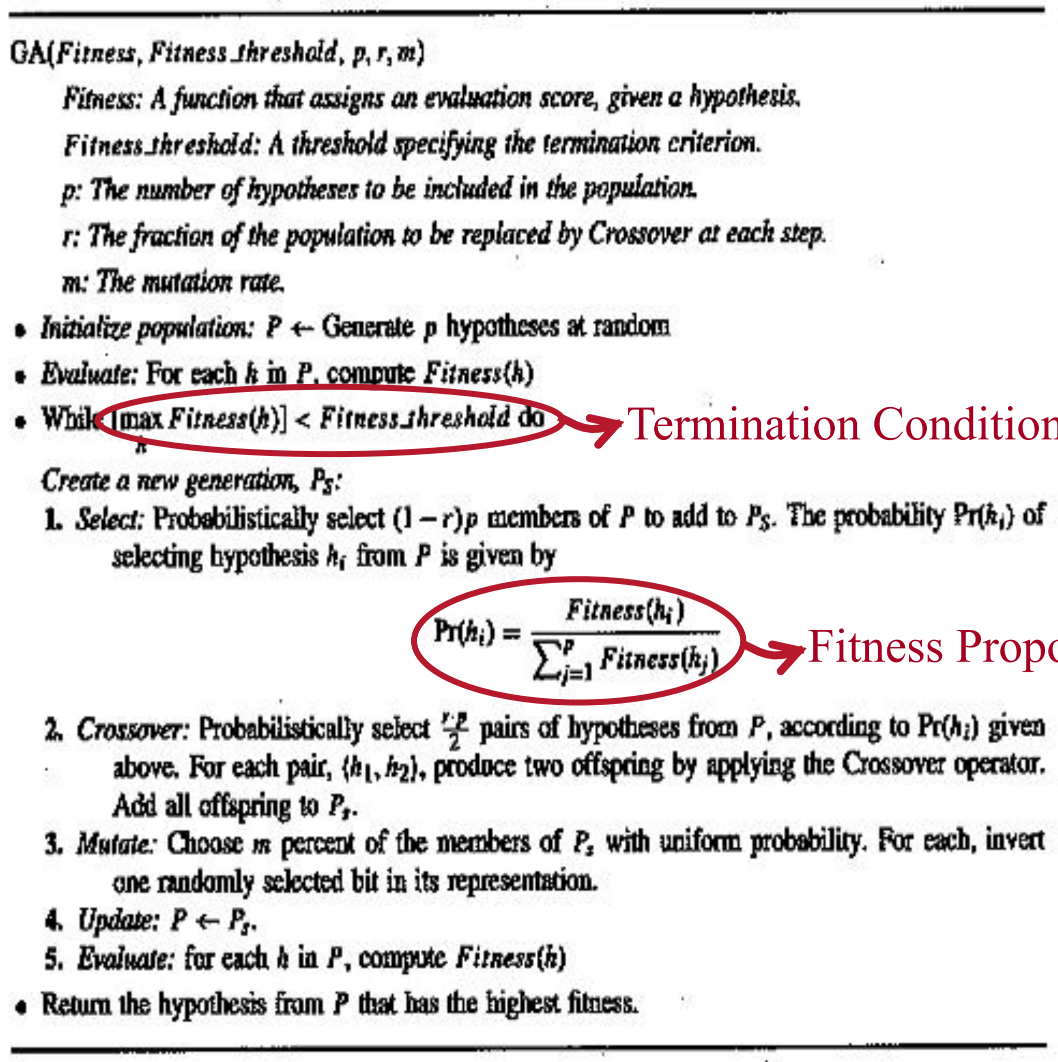
- Chromosome Design:

A	B	C	D	E	F	G	J	K	L	M	Z
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$
1	1	0	1	0	1	1	0	0	0	0	1

- Fitness Function f: based on the length of path h, $f(h) \equiv P(t) = 1 / (1 + e^t)$, $t = length(h)$
- 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., $length(h) < 2000\ m$

```
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
    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
            $$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{j=1}^{p} Fitness(h_j)}$$
    2. Crossover: Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from $P$, according to $\Pr(h_i)$ given
        above. For each pair, $\langle h_1, h_2 \rangle$, 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.
```

Termination Condition: A predefined target fitness

Fitness Proportionate Selection Method

Genetic Algorithm (Lecture)

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.
```

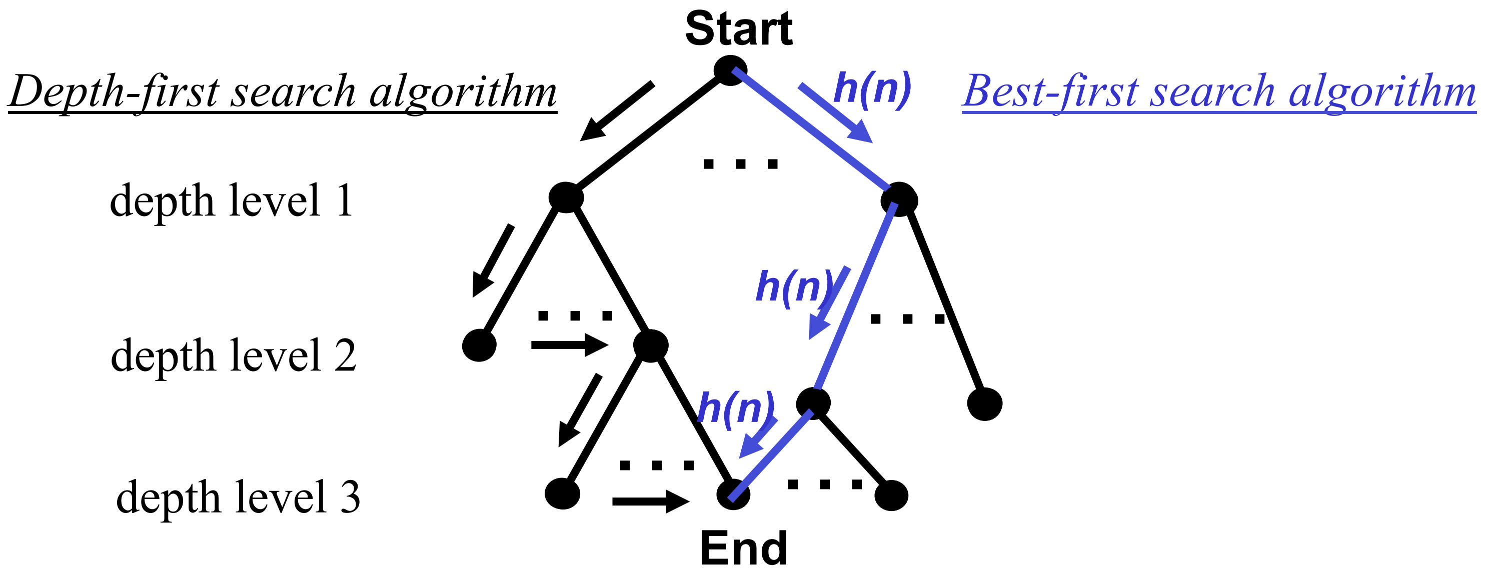
A more general version of GA than the one given in Mitchell's book.

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 .

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*).
  $\Rightarrow \{k \rightarrow n \;\Rightarrow\; \textit{Beam Search} \rightarrow \textit{Best-first Search}\}$

- Beam Search:
  - ❑ completeness
  - ❑ optimality

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*).
  $\Rightarrow \{k \rightarrow n \;\Rightarrow\; \textit{Beam Search} \rightarrow \textit{Best-first Search}\}$
- 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*).
  $\Rightarrow \{k \rightarrow n \;\Rightarrow\;$ *Beam Search* $\rightarrow$ *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 – Exam Questions

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

Course 395: Machine Learning – Lectures

- Lecture 1-2: Concept Learning (*M. Pantic*)
- Lecture 3-4: Decision Trees & CBC Intro (*M. Pantic*)
- Lecture 5-6: Artificial Neural Networks (*S. Zafeiriou*)
- Lecture 7-8: Instance Based Learning (*M. Pantic*)
- Lecture 9-10: Genetic Algorithms (*M. Pantic*)
- ➤ Lecture 11-12: Evaluating Hypotheses (*THs*)
- Lecture 13-14: Guest Lectures on ML Applications
- Lecture 15-16: Inductive Logic Programming (*S. Muggleton*)
- Lecture 17-18: Inductive Logic Programming (*S. Muggleton*)