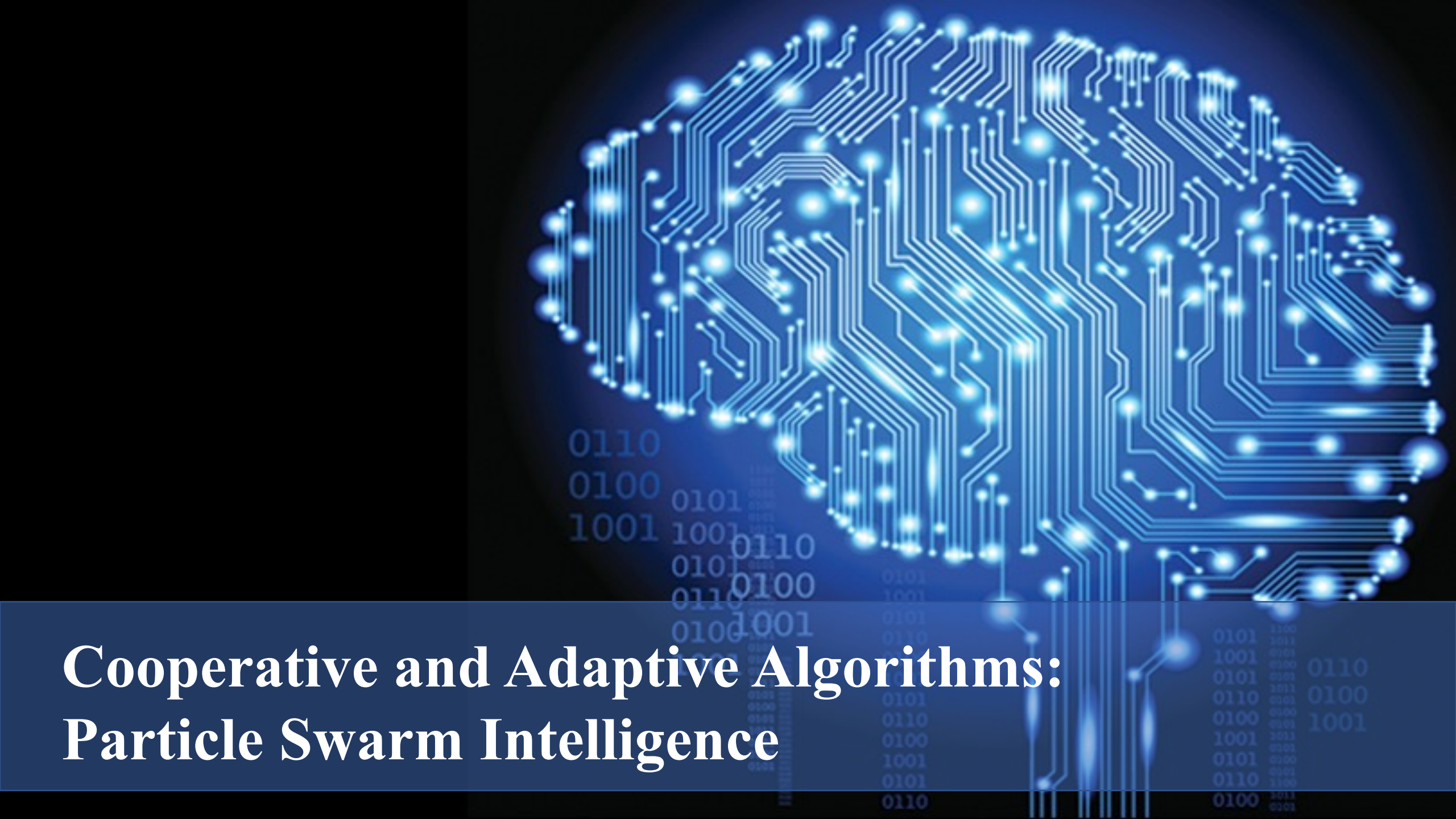


Cooperative and Adaptive Algorithms: Particle Swarm Intelligence

PSO Function Minimization Example

In this section, the PSO algorithm is explained in details to show how the particles are moved and also to show the influence of each parameter on the PSO algorithm. Assume the PSO algorithm has five particles (p^i, i = 1,…, 5), and the initial positions of the particles are presented in Table 1. The velocities of all particles are initialized to be zeros. Moreover, the initial best solutions of all particles are set to 1000 as shown in Table 1. In this example, De Jong function (see Equation 3) was used as a fitness function.

$$\min F(x,y) = x^2 + y^2 \tag{3}$$

where x and y are the dimensions of the problem. Figure 2 shows the surface and contour plot of the De Jong function. As shown, the function is a strictly convex function1. Moreover, the optimal solution is zero and it is found at the origin. Moreover, the lower and upper boundaries of both x and y dimensions were -1 and 1, respectively. Moreover, in PSO algorithm, the inertia (w) was 0.3, and the values of the cognitive and social constants were as follows, $C_1 = 2$ and $C_2 = 2$.

TABLE 1: Initial positions, velocity, and best positions of all particles.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	1	1	0	0	1000	-	-	2
P_2	-1	1	0	0	1000	-	-	2
P_3	0.5	-0.5	0	0	1000	-	-	0.5
P_4	1	-1	0	0	1000	-	-	2
P_5	0.25	0.25	0	0	1000	-	-	0.125

In this example, PSO iteratively searches for the optimal solution, and in each iteration the movement for each particle was calculated including its position, velocity, and fitness function as follows:

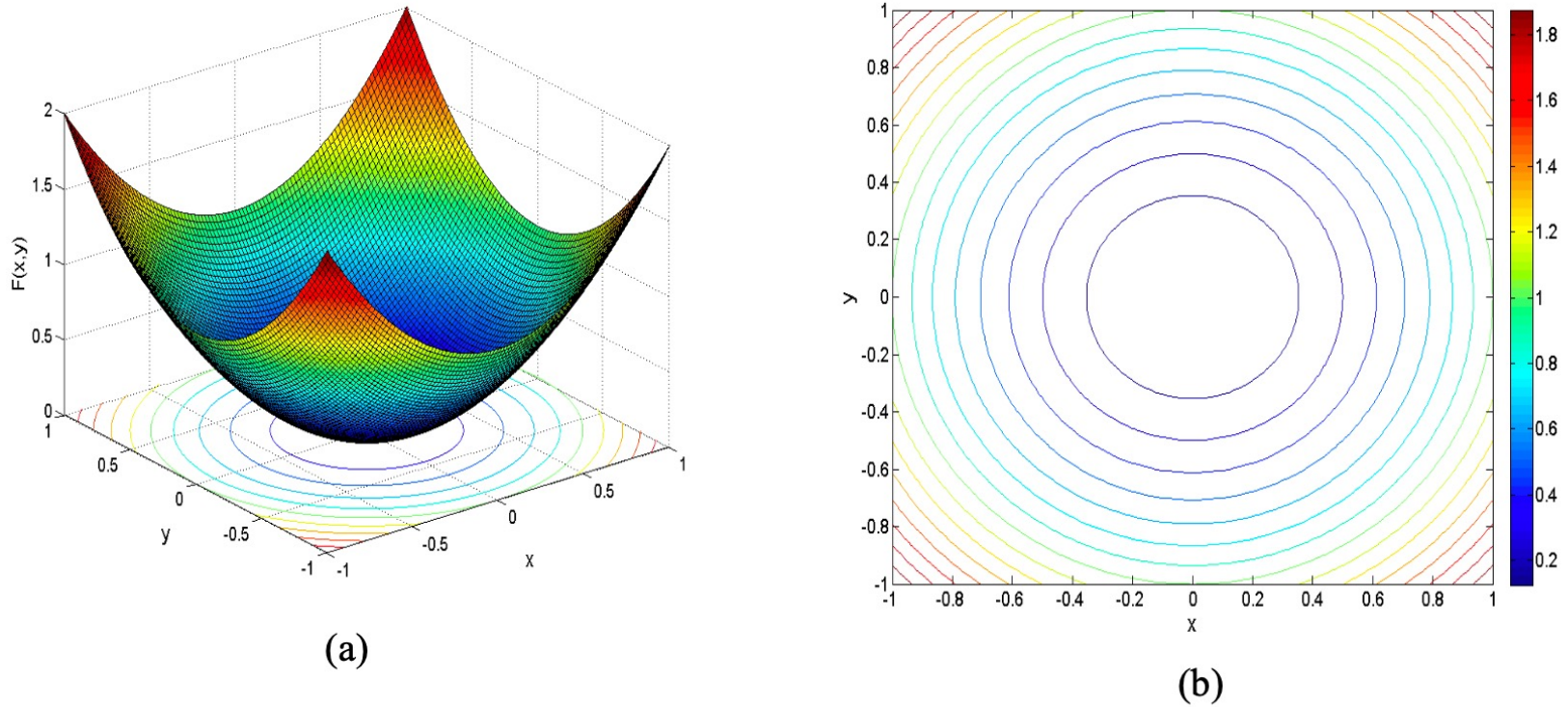


Figure 2: The surface and contour plot of De Jong function in Equation 3, (a) Surface (b) Contour plot.

First Iteration: The first step in this iteration was to calculate the fitness value for all particles, and if the fitness value of any particle ($F(p^i)$) was lower than the corresponding previous best position (p^i), then save the position of this particle. As shown from Tables 1, the first particle was located at (1, 1); hence, the fitness value is ~~12 + 12 = 2~~. Similarly, the fitness values of all particles were calculated as in Table 1. As shown, the fitness values of all particles were lower than the current best solutions; hence, the values of p^i were then updated with the best positions as shown in Table 2, and the best solutions were also updated. Moreover, the fifth particle achieved the best solution, i.e. minimum fitness value; G = (0.25, 0.25). As shown in Table 1, the initial velocities of all particles were zero; thus, the particles will not move in this iteration.

1^2+1^2=2

TABLE 2: The positions, velocity and best positions of all particles after the first iteration.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	1	1	-0.75	-0.75	2	1	1	2
P_2	-1	1	1.25 Text	-0.75	2	-1	1	2
P_3	0.5	-0.5	-0.25	0.75	0.5	0.5	-0.5	0.5
P_4	1	-1	-0.75	1.25	2	1	-1	2
P_5	0.25	0.25	0	0	0.125	0.25	0.25	0.125

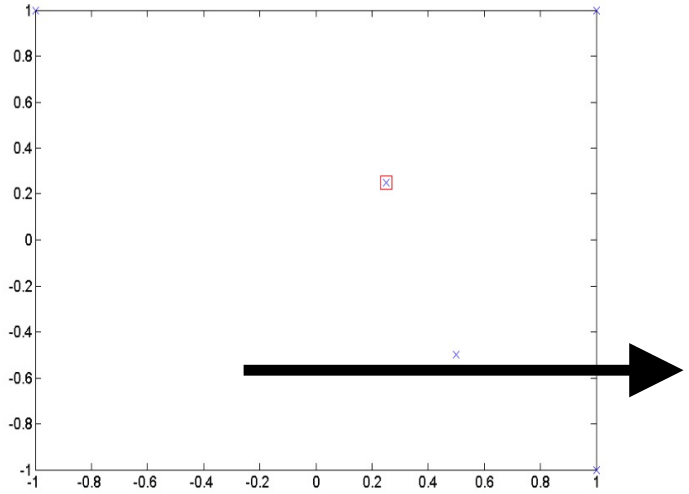
TABLE 3: The positions, velocity and best positions of all particles after the second iteration.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	0.25	0.25	-0.3750	-0.3750	2	1	1	0.125
P_2	0.25	0.25	0.6250	-0.3750	2	-1	1	0.125
P_3	0.25	0.25	-0.1250	0.3750	0.5	0.5	-0.5	0.125
P_4	0.25	0.25	-0.3750	0.6250	2	1	-1	0.125
P_5	0.25	0.25	0	0	0.125	0.25	0.25	0.125

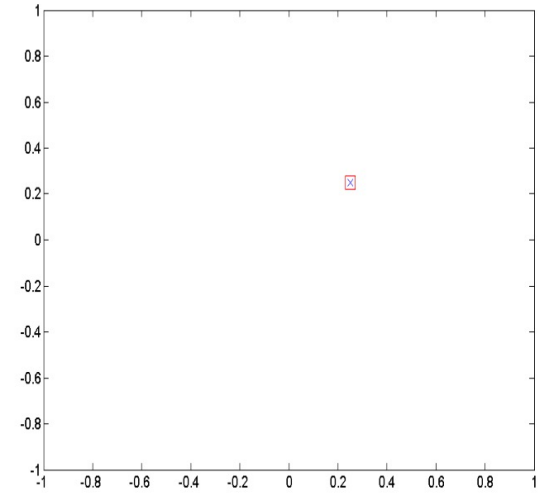
The PSO velocity updating equation

The velocity of each particle was calculated as in Equation (2). In this experiment, assume the two random numbers r_1 and r_2 were equal to 0.5. Since the initial velocity of all particles was zero as shown in Table 1 and the previous best positions and the current positions were equal; thus, the first two terms of

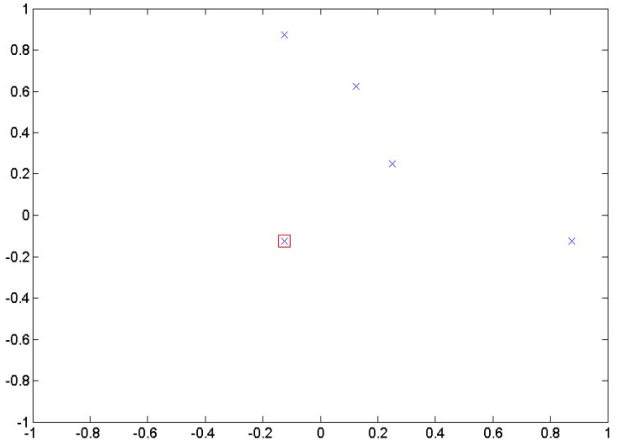
the velocity as shown in Equation (2) were equal to zero. Hence, the velocity in this iteration depends only on the global best position. The velocity of the first particle was calculated as follows, v^i (t+1) = $C_2 r_2$ (G - x^i (t)) = 2 * 0.5 * ((0.25 - 1), (0.25-1)) = (0.75, 0.75), and similarly the velocity of all particles were calculated and the values of all velocities are shown in Table 2. As shown in Table 2, the velocity of the fifth particle was (0, 0) because the fifth particle is the global best solution; hence, it remained at the same position at this iteration. Figure 3 shows the positions of all particles in this iteration. Moreover, Figure 4 shows how the best solution converged to the optimal solution during iterations.



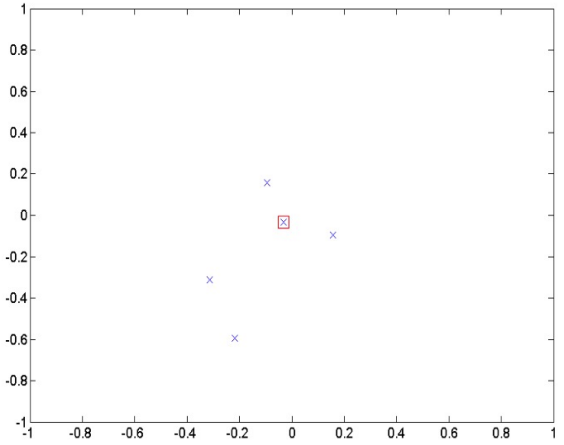
(a) First Iteration



(b) Second Iteration



(c) Third Iteration



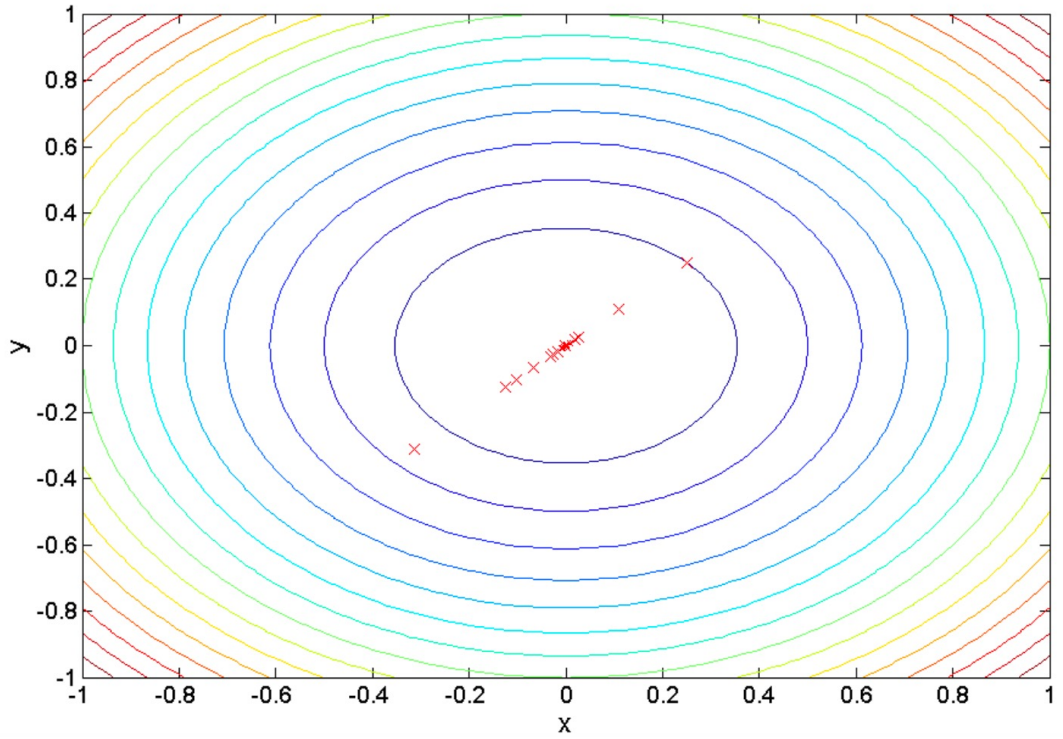
(d) Fourth Iteration

Second Iteration: In this iteration, the particles were moved to the new locations inside the search space using the positions and velocity that were calculated in the first iteration (see Section 4.1.1). The new positions of all particles are listed in Table 3. The velocity of all particles are then calculated (see Table 3) to move the particles in the next iteration. Moreover, the fitness values of all particles were calculated.

TABLE 4: The positions, velocity and best positions of all particles after the third iteration.

Particle No.	Initial Positions		Velocity		Best Solution	Best Position		Fitness Value
	x	y	x	y		x	y	
P_1	-0.1250	-0.1250	-0.1875	-0.1875	0.125	0.25	0.25	0.0313
P_2	0.8750	-0.1250	-1.3125	0.1875	0.125	0.25	0.25	0.7813
P_3	0.1250	0.6250	-0.1875	-0.9375	0.125	0.25	0.25	0.4063
P_4	-0.1250	0.8750	0.1875	-1.3125	0.125	0.25	0.25	0.7813
P_5	0.2500	0.2500	-0.3750	-0.3750	0.125	0.25	0.25	0.1250

Third Iteration: In this iteration, the particles continue moving towards the optimal solution. At the beginning, the particles were moved to the new positions and their fitness values were calculated as in Table 4. As shown, the first particle became much closer to the optimal solution than the other particles and its fitness value was 0.0313. As shown in this iteration, the velocities of all particles were not zero; in other words, the particles will be moved in the next iterations.

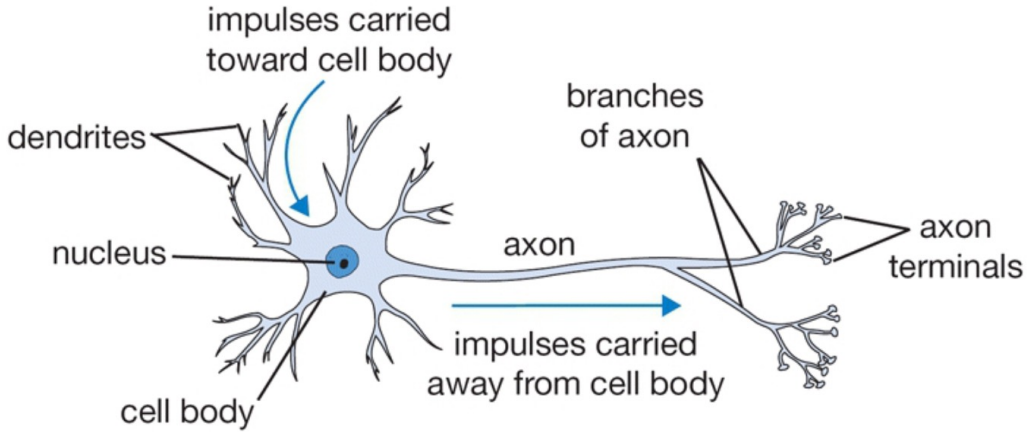


Particle Swarm Optimization

Neural Network Training

Biological motivation and connections

The basic computational unit of the brain is a **neuron**. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14} — 10^{15} **synapses**. The diagram below shows a cartoon drawing of a biological neuron (left) and a common mathematical model (right).



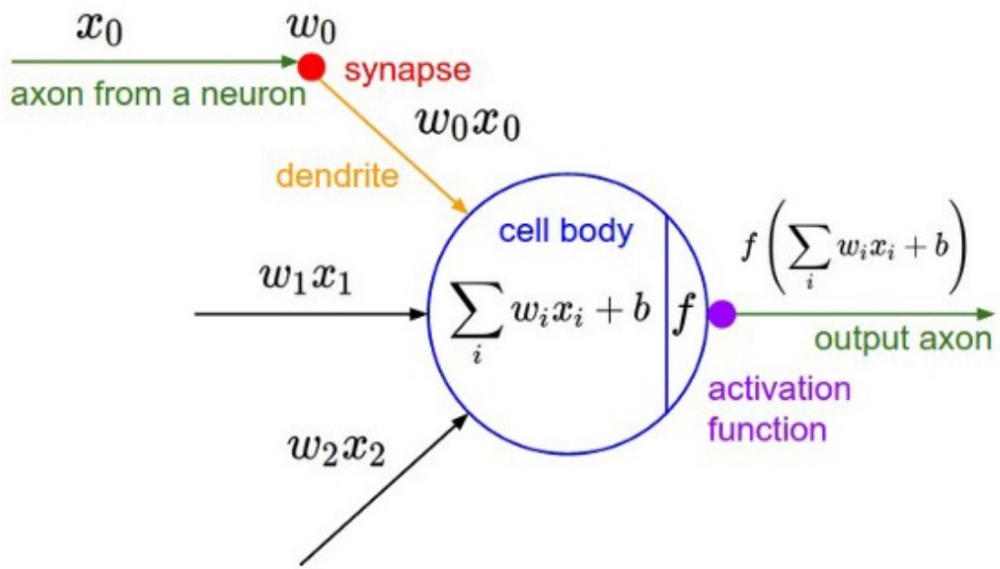
- The basic unit of computation in a neural network is the neuron , often called a node or unit.
- It receives input from some other nodes, or from an external source and computes an output.
- Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies a function to the weighted sum of its inputs.
- The idea is that the synaptic strengths (the weights *w*) are learnable and control the strength of influence and its direction: excitory (positive weight) or inhibitory (negative weight) of one neuron on another.
- In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can *fire*, sending a spike along its axon.

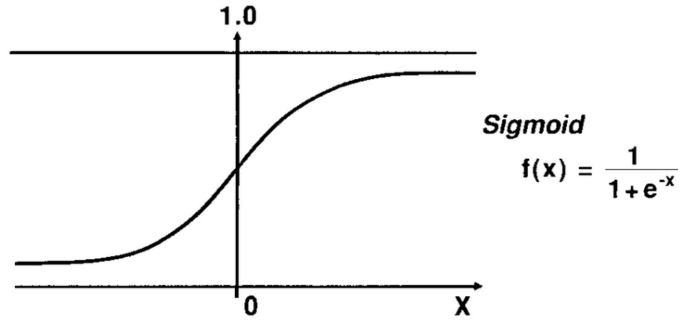
The Computational Model

- In the computational model:

The precise timings of the spikes are ignored,

- Only the frequency of the firing communicates information.
- We model the *firing rate* of the neuron with an **activation function** which represents the frequency of the spikes along the axon
- *The sigmoid function is an example.*





An illustration of the signal processing in a sigmoid function.

Neural Network Architecture

Neural Network Architecture

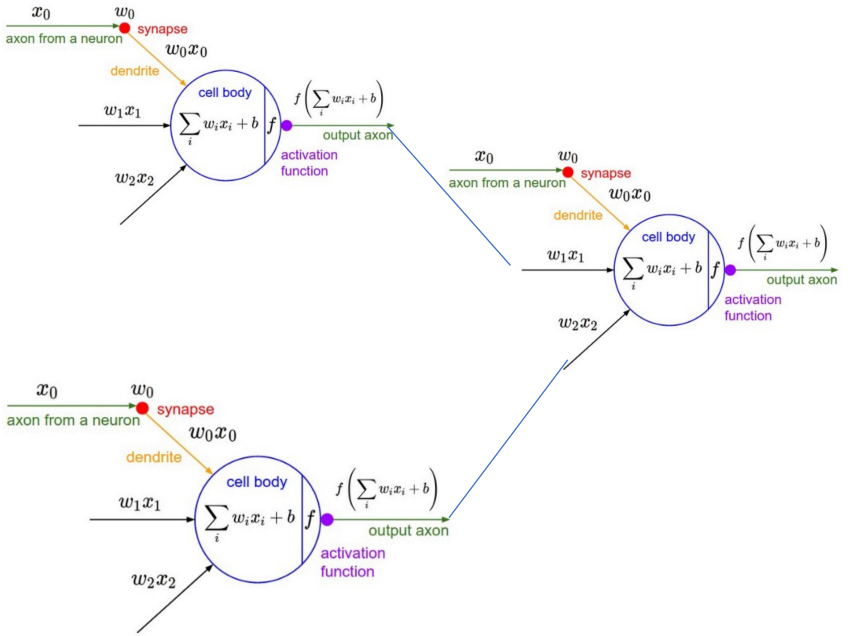
A neural network is made of neurons, biologically the neurons are connected through synapses where information flows (weights for out computational model), when we train a neural network we want the neurons to fire whenever they learn specific patterns from the data, and we model the fire rate using an activation function.

•**Input Nodes (input layer):** No computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called **layer**.

•**Hidden nodes (hidden layer):** In Hidden layers is where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another hidden layer or to the output layer). It is possible to have a neural network without a hidden layer and I'll come later to explain this.

•**Output Nodes (output layer):** Here we finally use an activation function that maps to the desired output format (e.g. class of the object in classification).

•**Connections and weights:** The *network* consists of connections, each connection transferring the output of a neuron i to the input of a neuron *j*. In this sense *i* is the predecessor of *j* and *j* is the successor of *i*, Each connection is assigned a weight *Wij.*

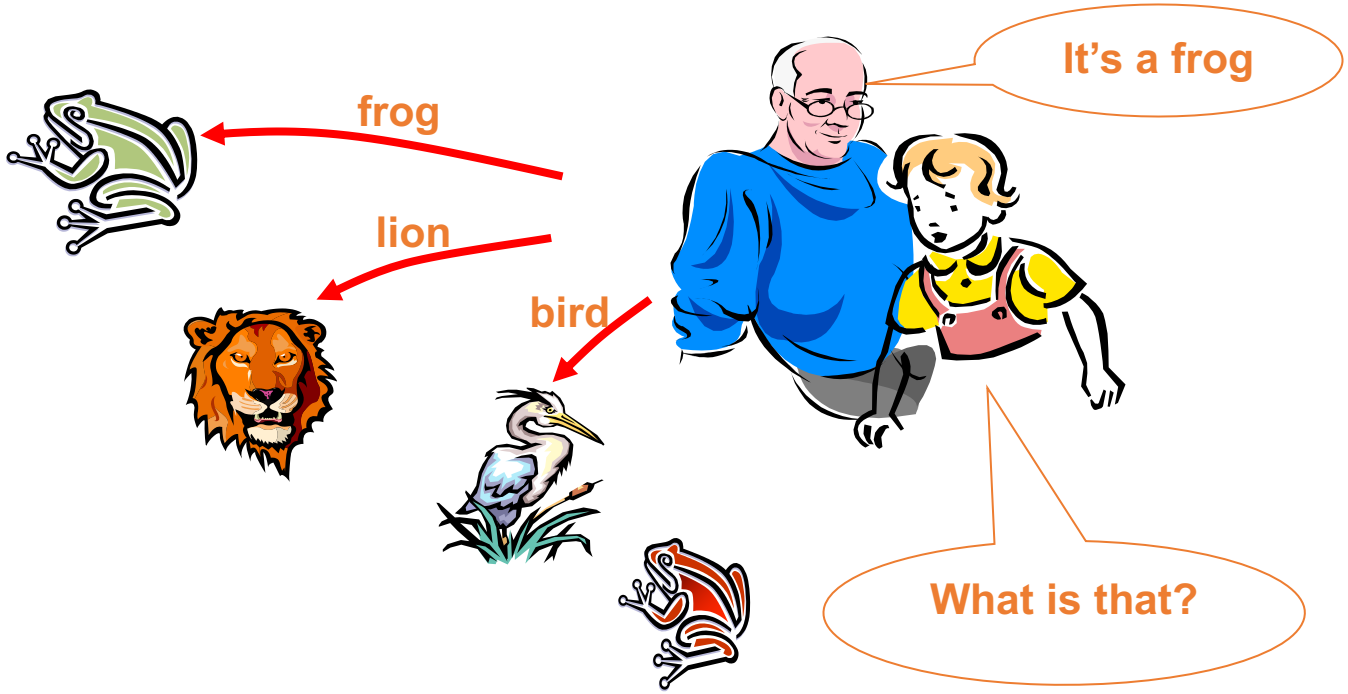


•**Activation function:** the **activation function** of a node defines the output of that node given an input or set of inputs. It is the *nonlinear* activation function that allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

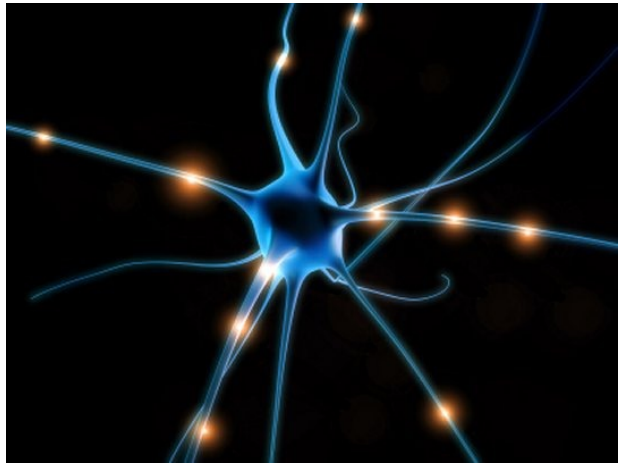
•**Learning rule:** The *learning rule* is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output. This *learning* process typically amounts to modifying the weights and thresholds.

Artificial Neural Network Learning

- Neural Networks (NN) learn the relationship between cause and effect or organize large volume of data into orderly and informative patterns.



Like people, they learn *from experience* (by example)



- **Neural network:** *information processing paradigm inspired by biological nervous systems, such as our brain*
  - Large number of highly interconnected processing elements (*neurons*) working together

Neural Network Training

- Neural Network (NN) performs a mapping from a set of input data to one or more output nodes,
- It's a structured set of nodes that are fully or partially connected,
- Each node is a processing unit that gets stimulated by:
  - An external input,
  - Another node.
- The node produces an output signal that reaches:
  - An output,
  - Another node.
- It's required to find the appropriate set of weights that will lead to a satisfactory network behaviour.

The Multi-Layer Perceptron Model

Artificial neural networks (NNs) are computational models based on the operation of biological neural networks, which constitute the information processing mechanism of the human brain. Their structure is based on the concept of the *artificial neuron*, which resembles biological neurons, as their main processing unit. The artificial neuron constitutes a nonlinear mapping between a set of input and a set of output data. Thus, if the input data are represented as a vector, $Q = (q_1, q_2, \ldots, q_m)^T$, the artificial neuron implements a function:

$$y = F\left(b_i + \sum_{i=1}^{m} w_i q_i\right),$$

where F is the *transfer function*; w_i, $i = 1, 2, \ldots, m$, are the *weights*; and b_i is a *bias*. In order to retain a compact notation, we will henceforth represent the bias, b_i, as a weight, w_0, with an auxiliary constant input, $q_0 = 1$.

The training of the neuron to learn an input-output pair, $\{Q, y\}$, is the procedure of detecting proper weights so that the output y is obtained if Q is presented to the neuron. Obviously, this procedure can be modeled as an error minimization problem:

$$\min_{w_i}\left(y - F\left(\sum_{i=0}^{m} w_i q_i \right) \right)^2 .$$

If more than one input vectors, $Q_1, Q_2, \ldots, Q_K$, are to be learned by the neuron, then the objective function is augmented with a separate square-error term for each input vector:

$$\min_{w_i} \sum_{k=1}^{K} \left(y_k - F\left(\sum_{i=0}^{m} w_i q_{ki} \right) \right)^2 ,$$

where q_{ki} is the i-th component of the k-th input vector, $Q_k = (q_{k1}, q_{k2}, \ldots, q_{km})^T$ (recall that w_0 is the bias of the neuron, with $q_{k0} = 1$ for all k). This kind of training procedure is also known as *batch training*, as all input vectors are presented to the neuron prior to any change of its weights.

Obviously, the dimension of the minimization problem is equal to the number, m, of the weights and biases, which is in direct correlation with the dimension of the input vector. The transfer function, $F(x)$, is selected based on the desired properties that shall be attributed to the model. Although linear transfer functions can be used, nonlinear functions equip the model with enhanced classification capabilities. Thus, nonlinear transfer functions are most commonly used, with *sigmoid* functions defined as:

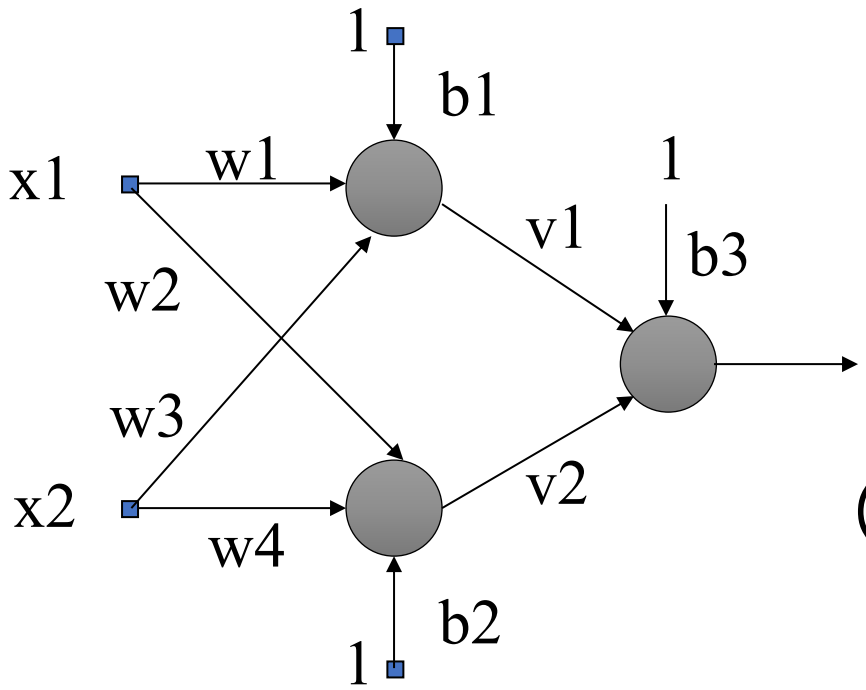
$$F(x,\lambda) = \frac{1}{1+\exp(-\lambda x)},$$

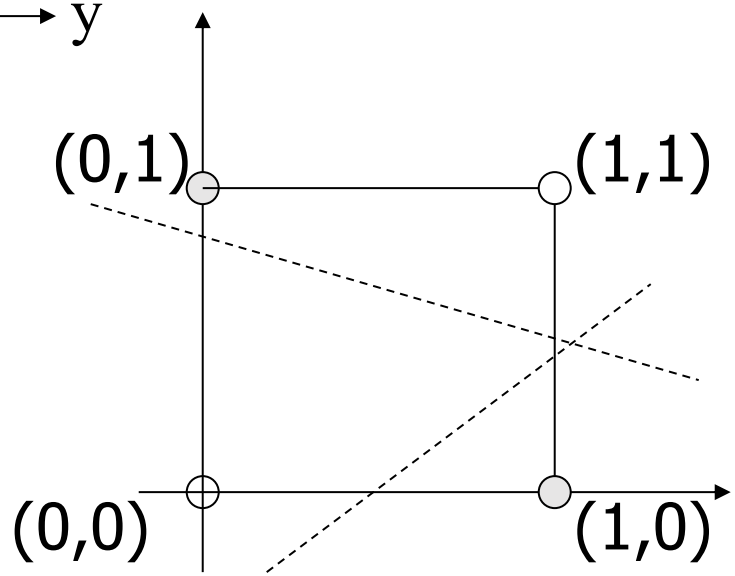
being the most common choice. In addition, alternative transfer functions have been proposed in the literature (Magoulas & Vrahatis, 2006).

Neural Network Training

- PSO proposed for optimizing the weights of an ANN performing the XOR operation.

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0





Neural Network Training

- How is the problem formulated?
  - PSO could be used to optimize the weights, each particle will be a 9-dimensional vector:

$$x = [w_1, w_2, w_3, w_4, b_1, b_2, b_3, v_1, v_2]$$

  - The velocity is a 9-dimensional vector of continuous values,
  - The objective is to minimize the error between the desired and the actual outputs of the network.

- It starts by randomly initializing a group of particles and let them move in the search space until an acceptable error is reached.

Neural Network Training

- PSO was effective as the back-propagation algorithm in training this NN,

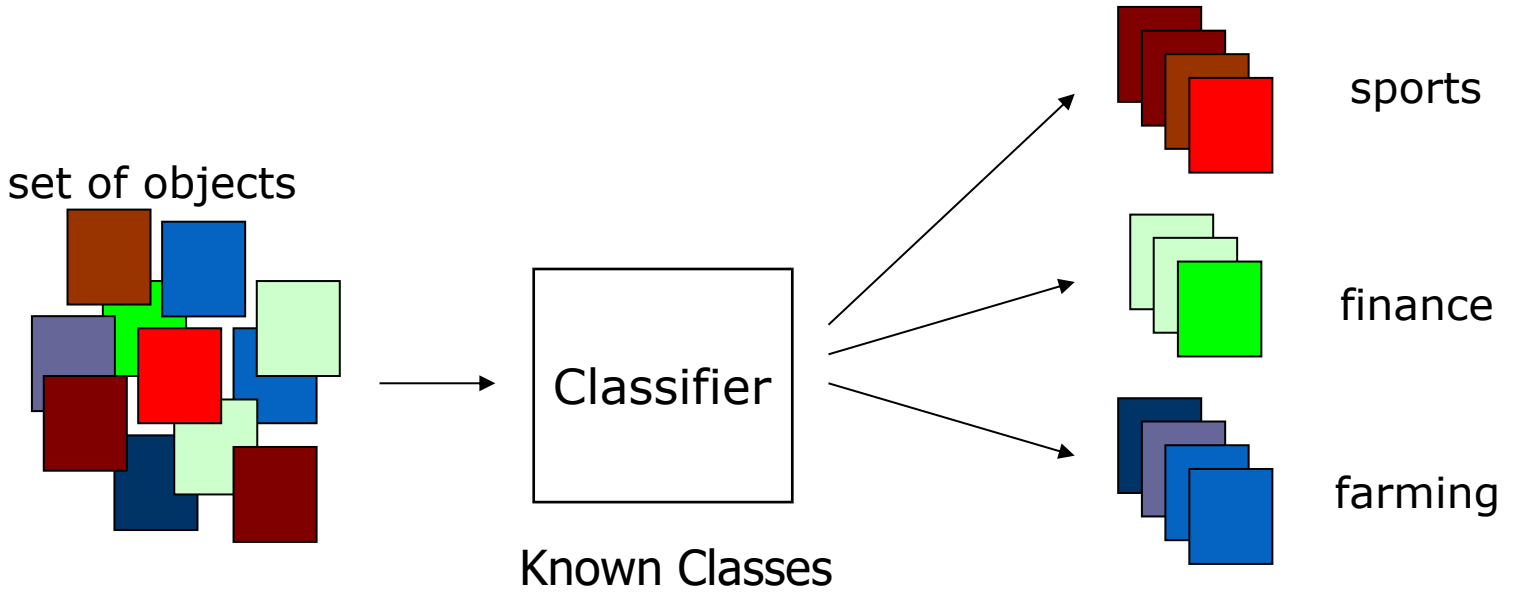
- A swarm of 20 particles was able to train network to an error of < 0.05 in an average of 30.7 iterations.

Particle Swarm Optimization

Clustering

Classification

- Function that assigns an object to a class
- Infer that “object X is about sports”
- Automatically learn the function from a set of examples
- Group similar objects into classes
- Similarity is high within the class and low between classes



Clustering vs. Classification

Clustering

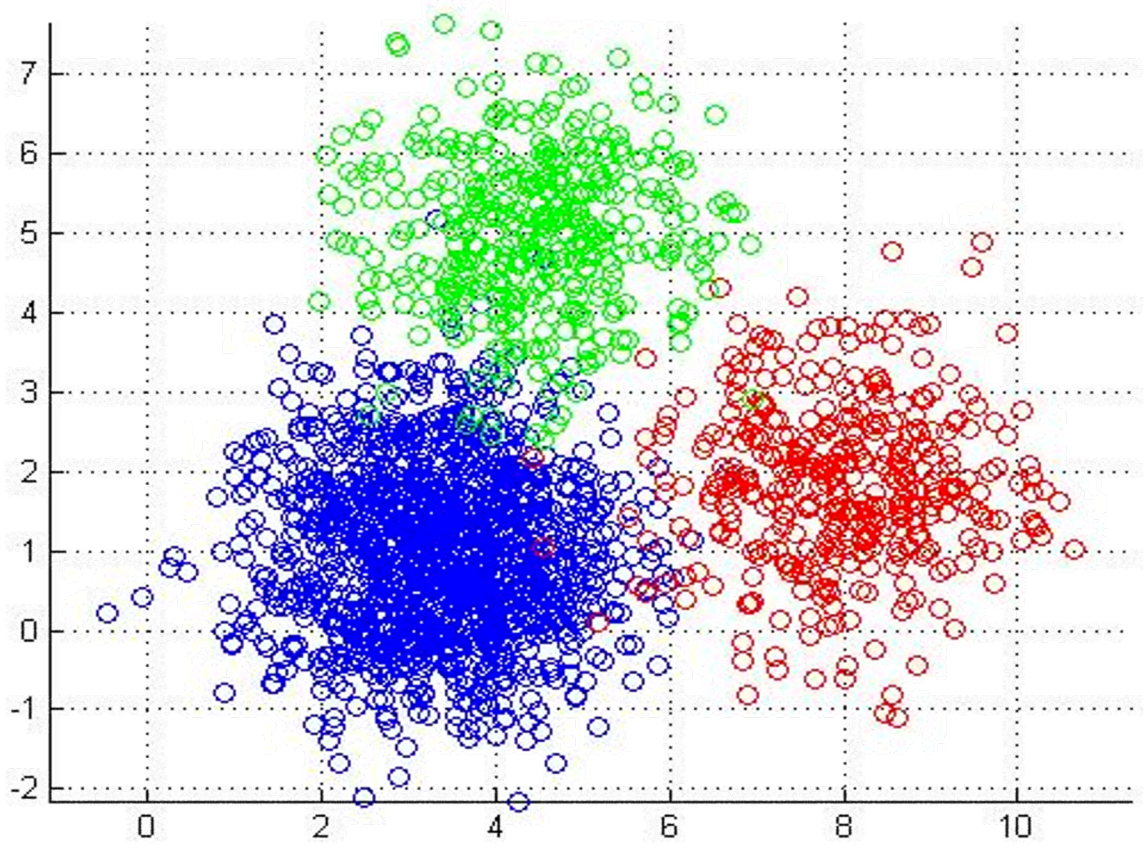
- Unsupervised
- Uses unlabeled data
- Organize patterns w.r.t. an optimization criteria
- Notion of similarity
- Hard to evaluate
- Example: K-means, Fuzzy C-means, Hierarchical

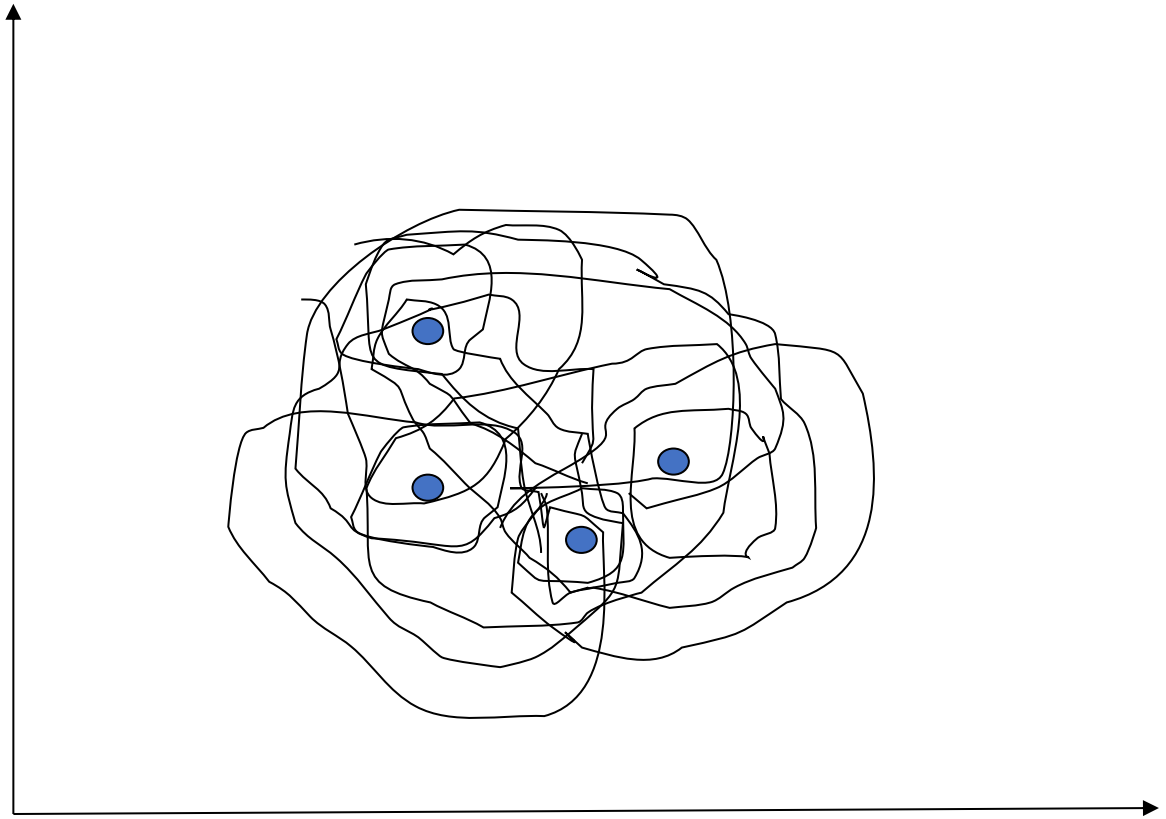
Classification

- Supervised
- Uses labeled data
- Requires training phase
- Domain sensitive
- Easy to evaluate
- Examples: Bayesian, k-NN, Decision Trees

Data Clustering Problem

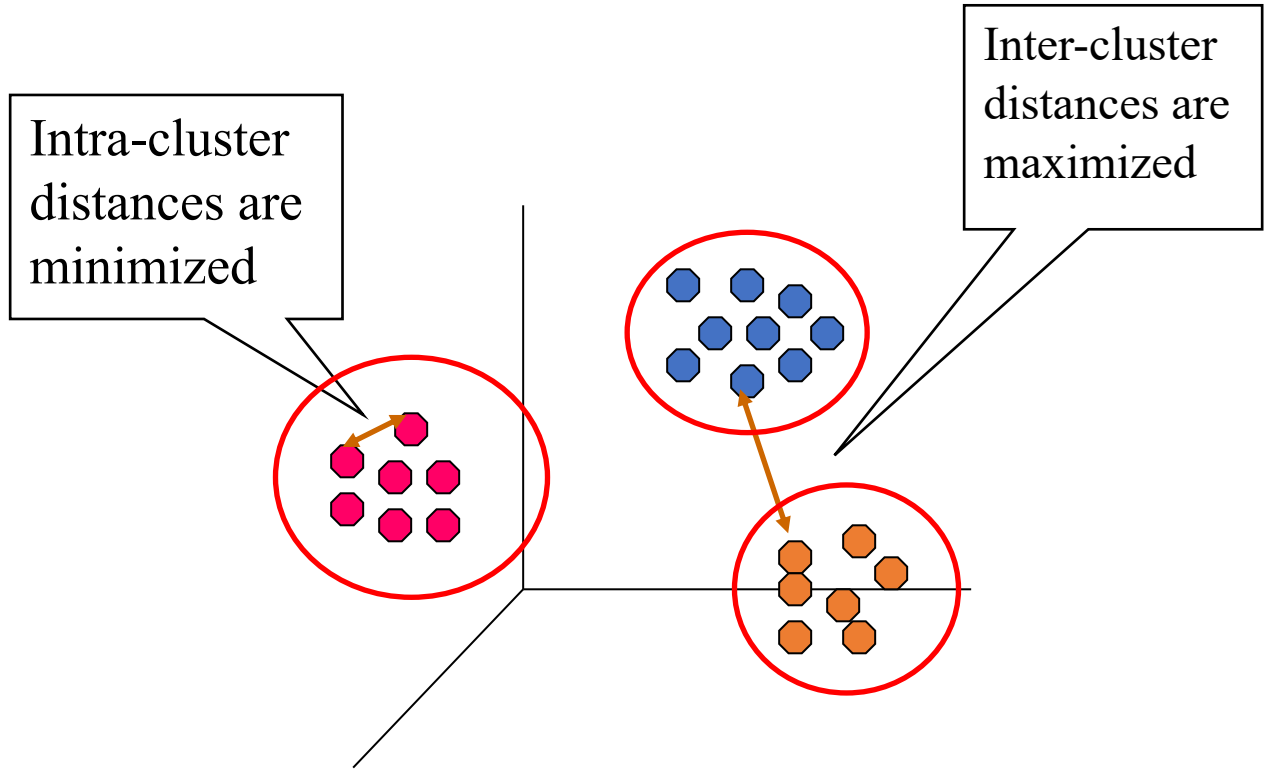
Given *n* objects (each could be a vector of d features), group them in *k* groups (Clusters) in such a way that all objects in a single group have a “natural" relation to one another, and objects not in the same group are somehow different





Applications of data clustering

- Data analysis
- Bioinformatics data
- Image segmentation
- GIS and spatial data
- Data and web mining
- Medical data
- Economic and financial data



K- means clustering Algorithm

Step 1: Initialize k Cluster centers

Repeat

Step 2: Distribute patterns among clusters using similarity measure and satisfying performance index.

Step 3: Compute new cluster centers

until no change in centers.

Apply the algorithm a number of times with different initial conditions

then choose the best solution.

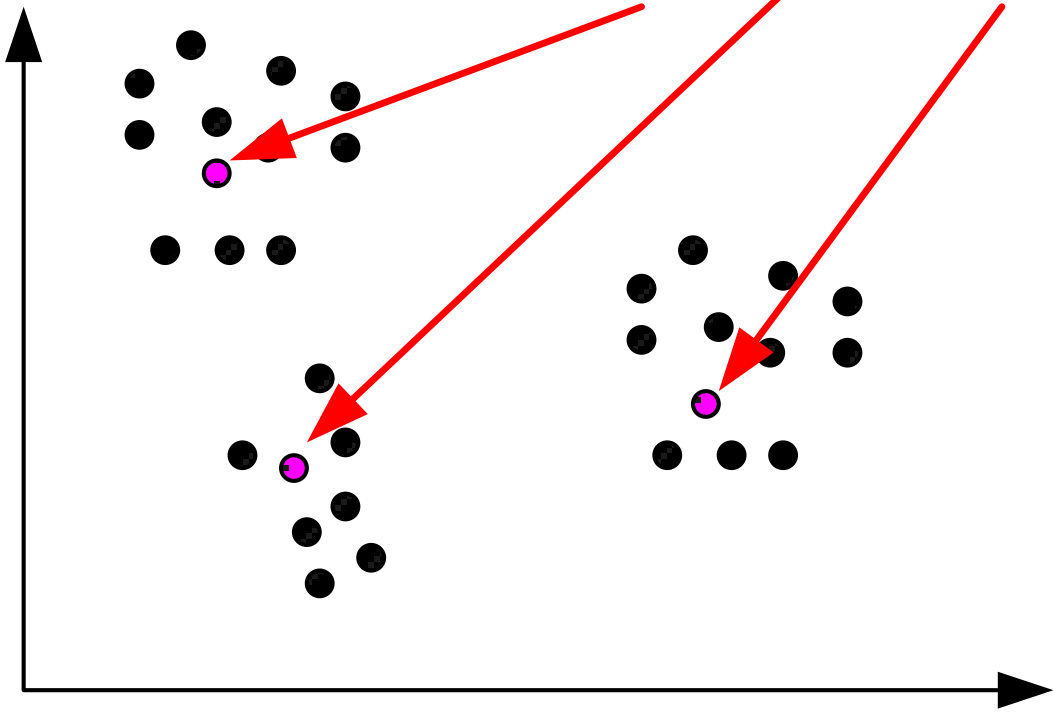
- The similarity measure could be the inverse of the Euclidean distance and a performance index can be the dispersion

$$f(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} \sum_{x_j \in C_i} \left\| x_j - z_i \right\|^2$$

Single Swarm Clustering

Particle representation

$$x^i = (m^{(1)}, m^{(2)}, m^{(3)})^i$$



PSO-Clustering

The PSO Clustering Algorithm

Initialize each particle with K random cluster centers.

for iteration count = 1 to maximum iterations do

for all particle i do

for all pattern X*p* in the dataset do

calculate Euclidean distance of X*p* with all cluster centroids

assign X*p* to the cluster that have nearest centroid to X*p*

end for

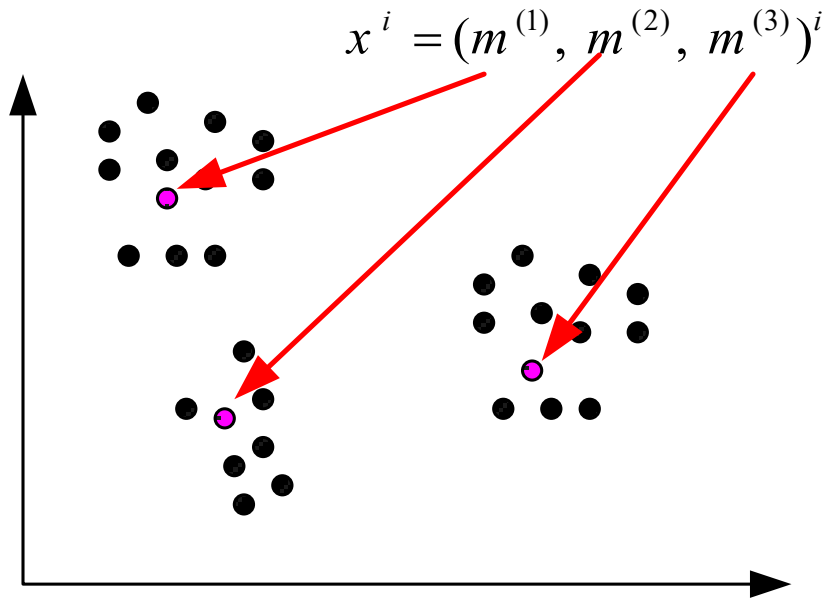
calculate the objective function for the current centers and assignment

end for

find the personal best and global best position of each particle.

Update the cluster centroids according to velocity updating and coordinate updating formula of PSO

end for.



Application to Document Clustering

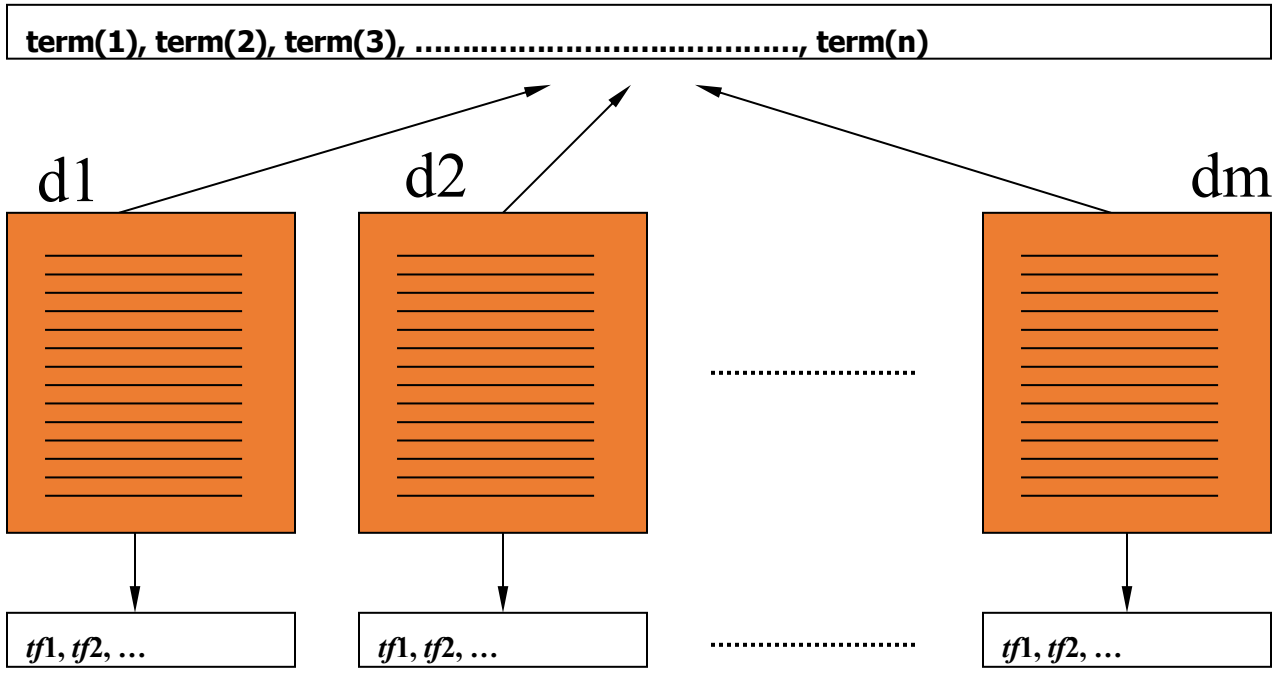
- Problems
  - Overwhelming amount of text information.
  - Search engines return linear ranked list of documents.
  - Documents are of various topics yet interleaved.

- Motivation
  - Uncovering inherent groupings in large document sets.
  - Easy navigation through large document sets.
  - Easy navigation of search engine results.

Document Representation Model

- Most Commonly used: Vector Space Model
- Document Representation: $\mathbf{d} = \{tf_1, tf_2, \ldots, tf_n\}$

  where $tf_i, i = 1, \ldots, n$ is the term frequency



Term Frequency Vectors

Document Representation Model

- Term-Document matrix

	t_1	t_2	t_3	.....	t_n
d_1	0.3	0	0.5	.....	0
d_2	0	0.1	0.4	.....	0.2
:	:	:	:		:
:	:	:	:	.....	:
:	:	:	:		:
d_m	0.1	0	0	.....	0

- **Term Frequency Disadvantage:** A term appearing frequently in all documents has less discriminating power.
- **Document Frequency (DF):** The number of documents in which a certain term appears.
  - The less the DF of a term, the more discriminating power it has.

Similarity Measure

- Cosine Measure (most commonly used) $\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$
- Euclidean Distance $\|x - y\|_2$
- The average distance of documents to the cluster centroid (ADDC)

$$(1/k)\sum_{i=1}^{k}\sum_{j=1}^{n_i}\left\|d_{ij} - m_i\right\|_2 / n_i$$

- n_i is number of documents in cluster i

Hybrid PSO-K-means

1) Start the PSO clustering process until the maximum number of iterations is exceeded
2) Inherit clustering result from PSO as the initial centroid vectors of K-means module.
3) Start K-means process until maximum number of iterations is reached.

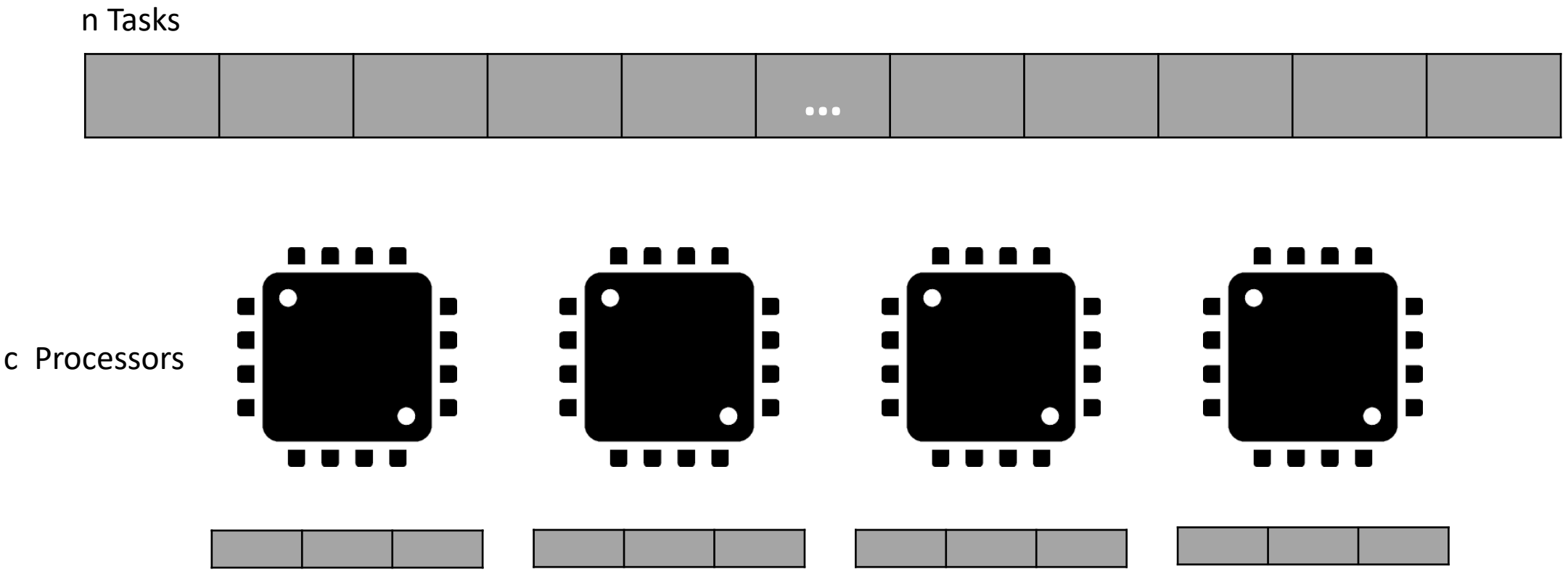
For PSO ***w*** is initially set as 0.72 and the
acceleration coefficient constants ***c1*** and ***c2*** are set as 1.49.
w is reduced by 1% at each generation to ensure good convergence.

Comparison between K-means, PSO, Hybrid PSO

Data Set	Similarity Measure	K-means	PSO-Kmeans	Hybrid PSO
Data Set 1	Euclidean	8.17817	8.11009	6.38039
	Cosine	8.96442	10.41271	8.14551
Data Set 2	Euclidean	7.26175	6.25172	4.51753
	Cosine	8.07653	9.57786	7.21153
Data Set 3	Euclidean	4.59539	4.14896	2.25961
	Cosine	4.97171	5.71146	4.00555
Data Set 4	Euclidean	9.08759	8.62794	6.37872
	Cosine	10.1739	12.8927	9.5379

Set Partitioning

- Set partitioning: a **partition** of a **set** is a grouping of the **set's** elements into non-empty subsets, in such a way that every element is included in one and only one of the subsets.



Set Partitioning

- Number of possible partitions is given by:

$$\mathrm{N}(n,c) = \left(\frac{1}{c!}\right)\sum_{m=1}^{c}(-1)^{c-m}\binom{c}{m}m^{n}$$

- If n = 50 and c = 4, the number is 5.3×10^{38}.

- If n is increased to 100, the number becomes 6.7×10^{58}.

- Enumerating all possible partitions for large problems is practically infeasible.

References

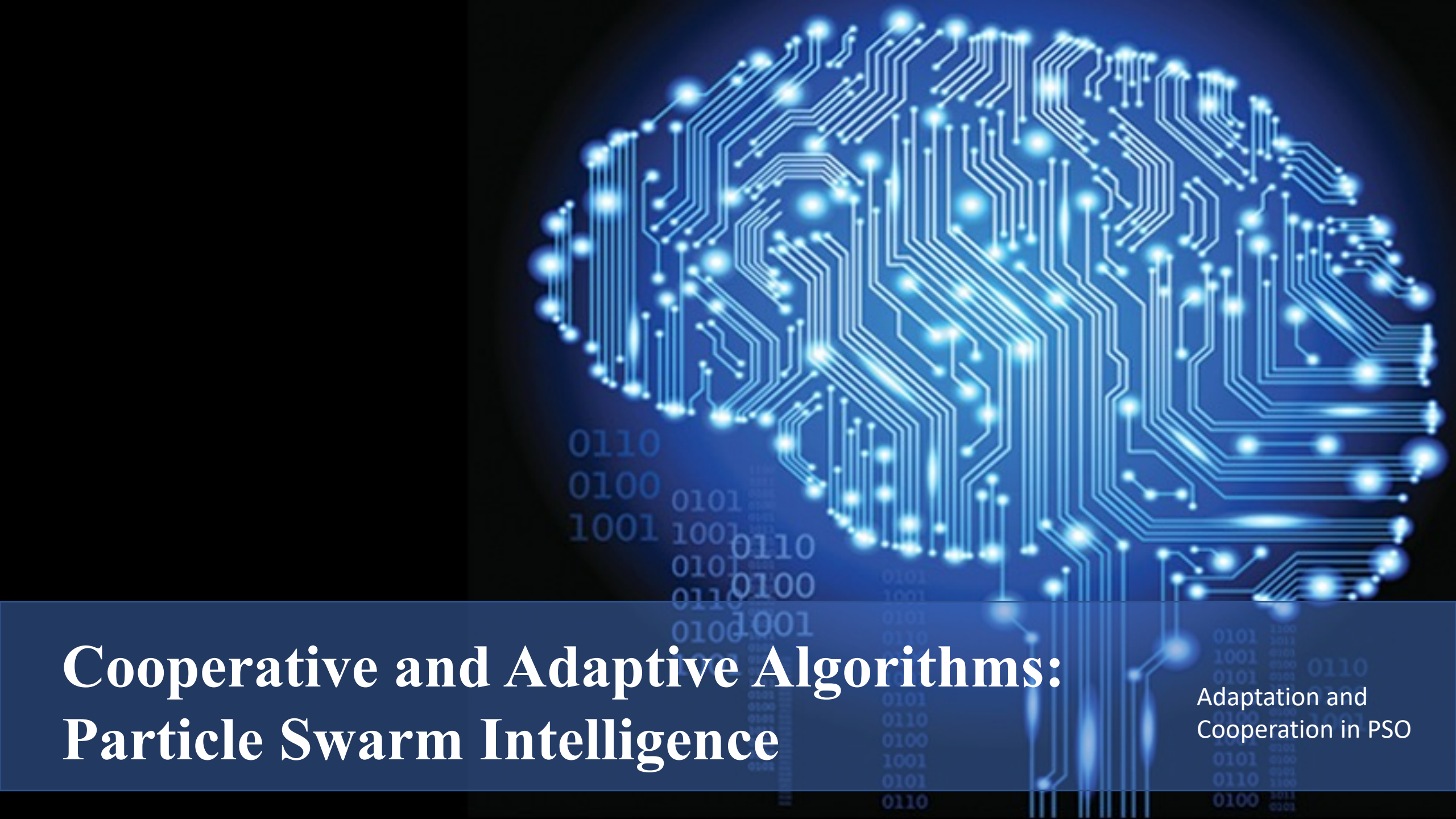
1. J. Kennedy and R. C. Eberhart. “Particle Swarm Optimization”. Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.

2. K. Lei, Y. Qiu and Y. He. “A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer”. Proc. of the 1st International Symposium on Systems and Control in Aerospace and Astronautics ISSCAA, pp. 981-984,2006.

3. Abraham, A., Das, S., & Roy, S. Swarm intelligence algorithms for data clustering. In Soft computing for knowledge discovery and data mining Part IV, pp. 279-313, 2007.

4. X. Cui; J. Gao and T. E. Potok. A flocking based algorithm for document clustering analysis. *Journal of Systems Architecture*, 52(8-9):505–515, August-September 2006

5. A. Ahmadi; F. Karray and M. Kamel. Multiple cooperating swarms for data clustering. In *IEEE Swarm Intelligence Symposium*, pages 206–212, 2007.



Cooperative and Adaptive Algorithms: Particle Swarm Intelligence

Adaptation and Cooperation in PSO

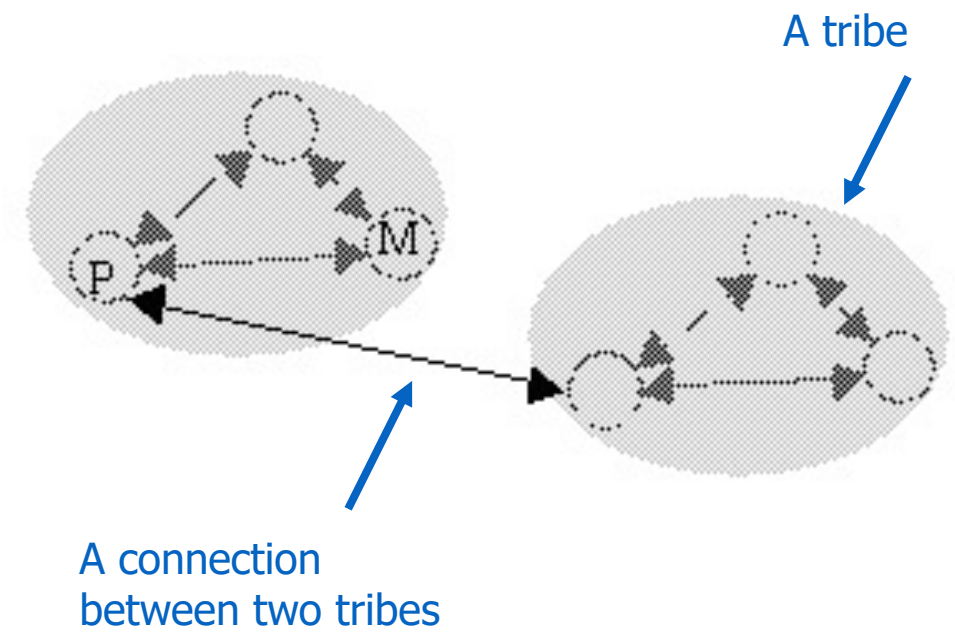
Adaptation

- An adaptive parameter-free PSO, referred to as *TRIBES*, was proposed in [17],
- The purpose was to have the user to call PSO algorithm without the need to set any parameters,
- The main point was to adapt the number particles used in the search.

Adaptation

- A *tribe* refers to a group of connected particles,
- All the tribes should have some type of connection between them to inform one another of their findings,
- This will help in deciding which is the global minimum among all the different solutions that was found by the different tribes.

Adaptation



Adaptation

- Definitions:
  - A *good* particle is a particle that has its pbest improved in the last iteration, otherwise it's *neutral*,
  - Each particle memorizes the last two performance variations, a particle with both variations as improvements is an *excellent* particle,
  - *G* is the number of good particles in a tribe.

Adaptation

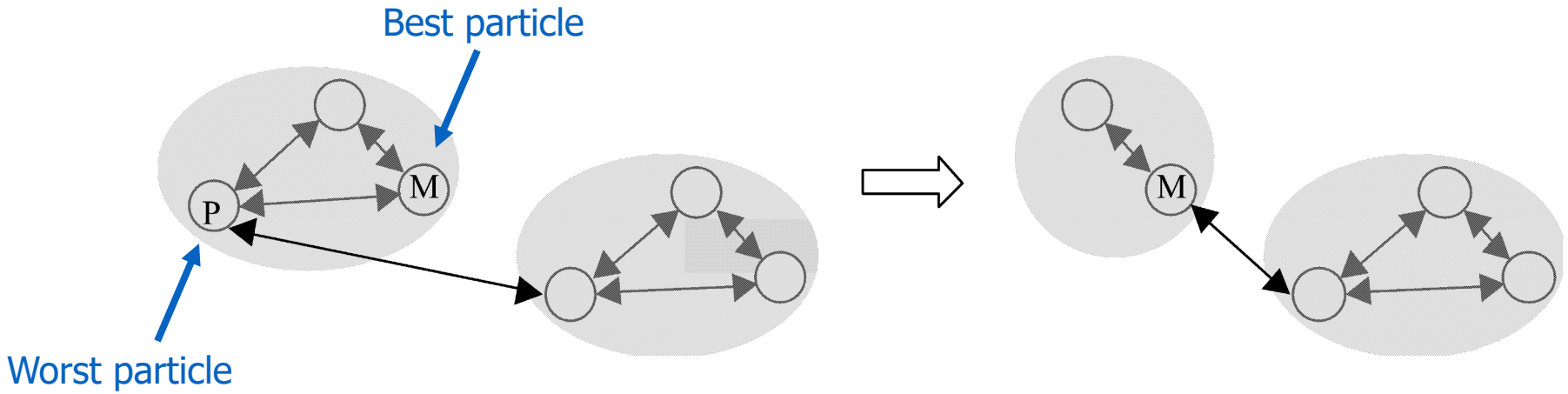
- Definitions:
  - A tribe is marked as good depending on the value of G, if the total number of particles in a tribe is T:

$$r = U(0,1)$$

$$Tribe = \begin{cases} good, & r < \dfrac{G}{T} \\ bad, & otherwise \end{cases}$$

Adaptation

- A good tribe deletes its worst particle to conserve the number of performed function evaluations.



Adaptation

- On the other hand, every bad tribe generates a new random particle simultaneously,
- All the new particles form a new tribe,
- Each particle gets connected to the tribe that generated it through its best particle.

Adaptation

- The idea is to start with a single particle,
- Most likely, this particle won't improve in the first iteration. Hence, it will generate another particle forming another tribe,
- If both don't improve, they will simultaneously generate two other particles forming a third two-particle tribe,
- And the process continues…

Adaptation

- If things go bad, larger and larger tribes will be generated to increase the swarm search power,

- On the other hand, if good solutions start to occur, good tribes will start removing its worst particles reducing the tribes size possibly to complete extinction.

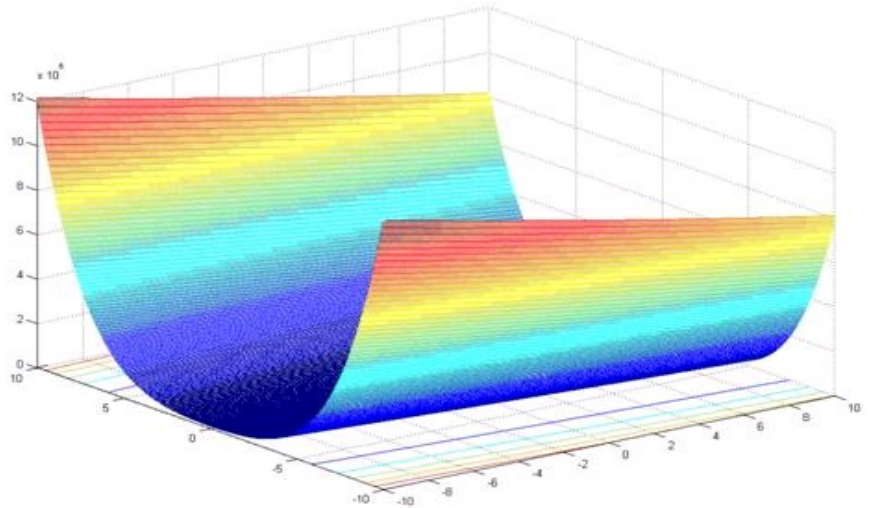
2D Rosenbrock

- Assume that one is constrained with only 40000 function evaluations to perform, what would be the optimal swarm size to use in order to find the best result?

- The question was answered by solving the Rosenbrock function in a dimensionality of 30.

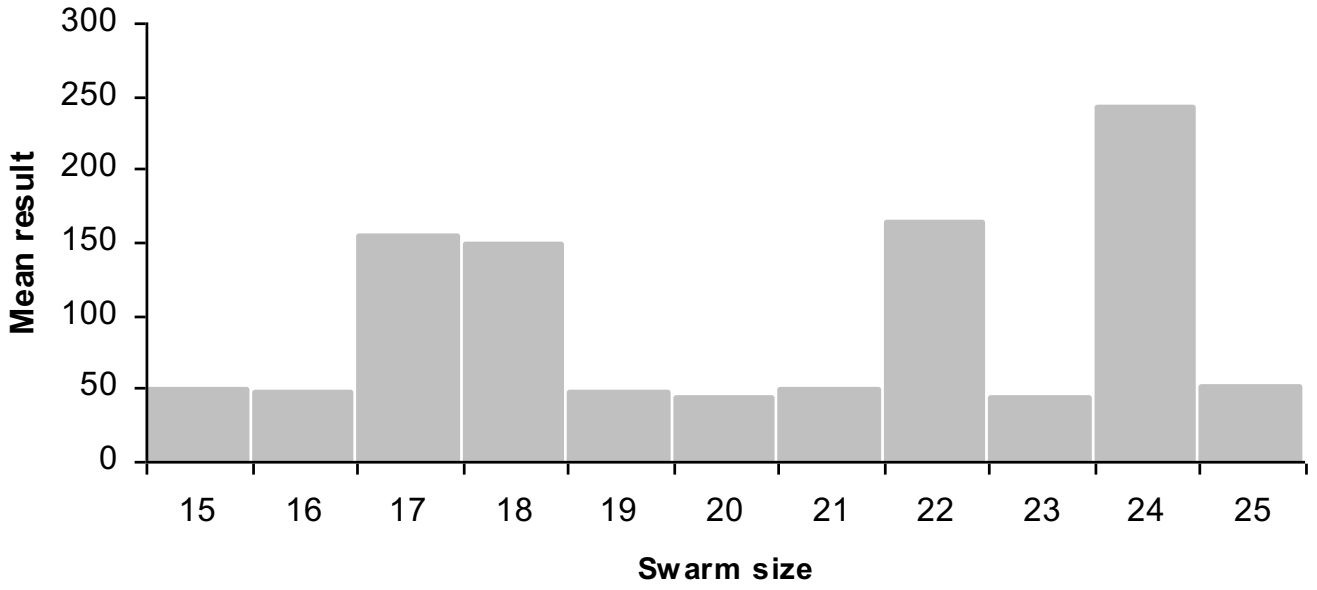
- Rosenbrock:

$$f(x)=\sum_{i=1}^{d-1}\left[(1-x_i)^2+100\left(x_{i+1}-x_i^2\right)^2\right]$$



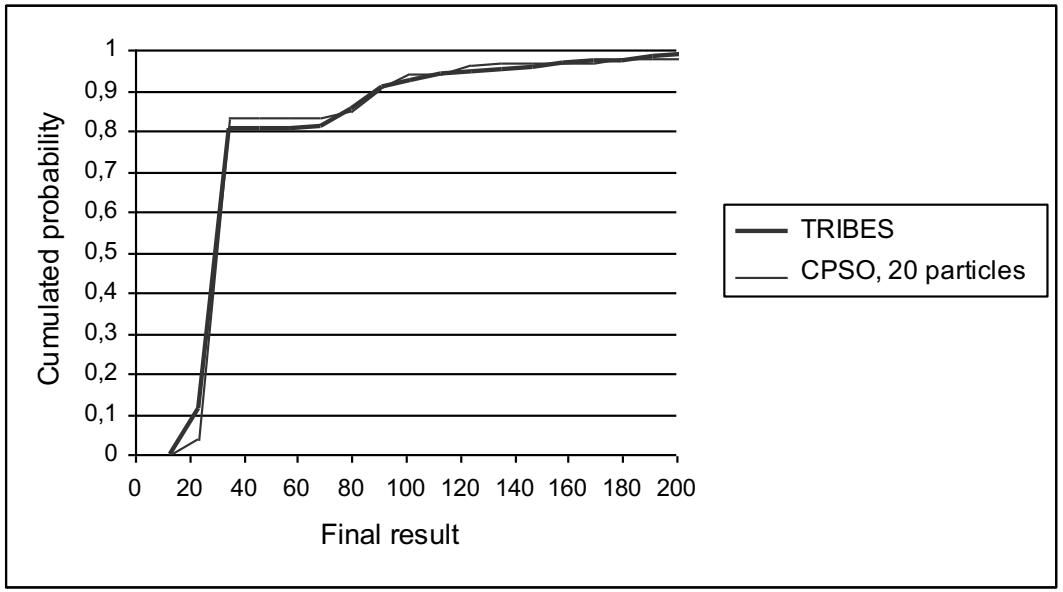
Adaptation

- The results show that using 20 particles is the best. However, there's no clear relation between the number of particles and the performance.



Adaptation

- The function was optimized using TRIBES and it shows that it was able to exactly follow the behaviour of a swarm of 20 particles, 500 runs, 40000 function eval.

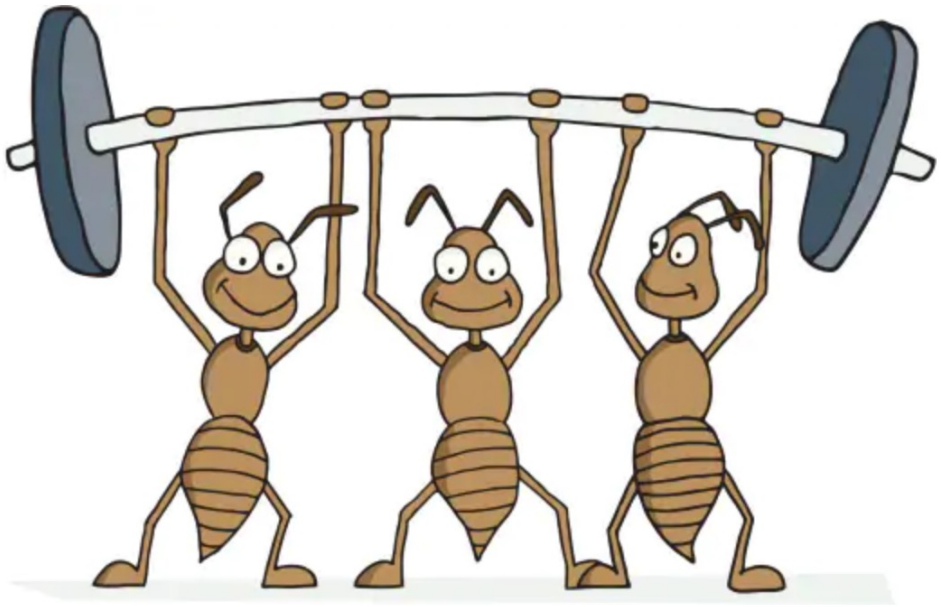


the graph shows that with Classical PSO (20 particles),there is 83% chance of obtaining a value smaller than 40, against 80 % with TRIBES. For small values, TRIBES is clearly better: 11.5 % chance to obtain a value smaller than 25, against 4 % with CPSO

Cooperation

- Different approaches have been proposed to introduce cooperation among different swarms,

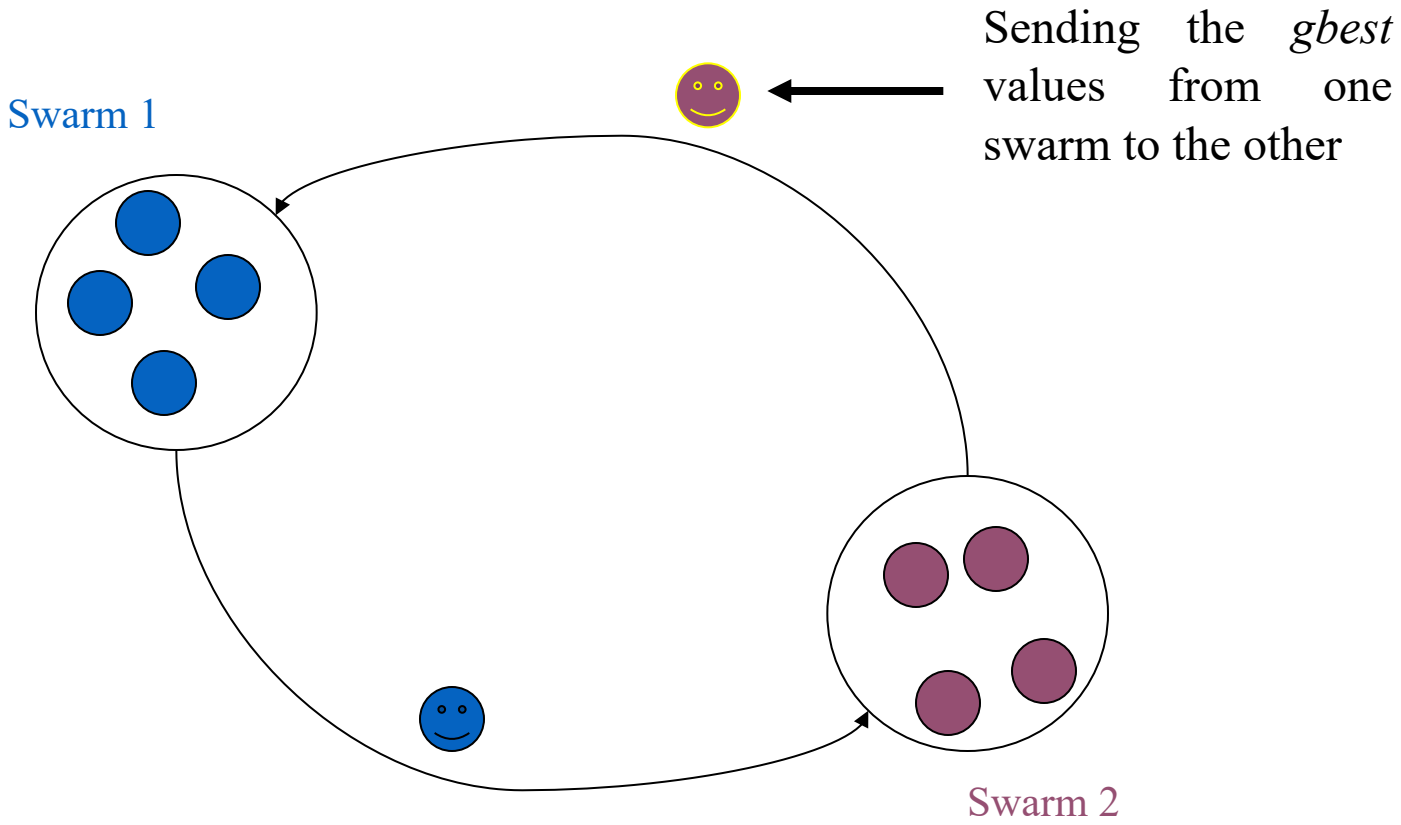
- These approaches include:
  - Concurrent PSO,
  - Cooperative PSO,
  - Hybrid Cooperative PSO.



Cooperation

- One form of multi-swarm algorithms is known as *concurrent-PSO* [13],
- Two different swarms are updated in parallel, both using different algorithms,
- The swarms exchanged their *gbest* values every pre-determined number of iterations,
- Both swarms are to track the better *gbest*.

Cooperation



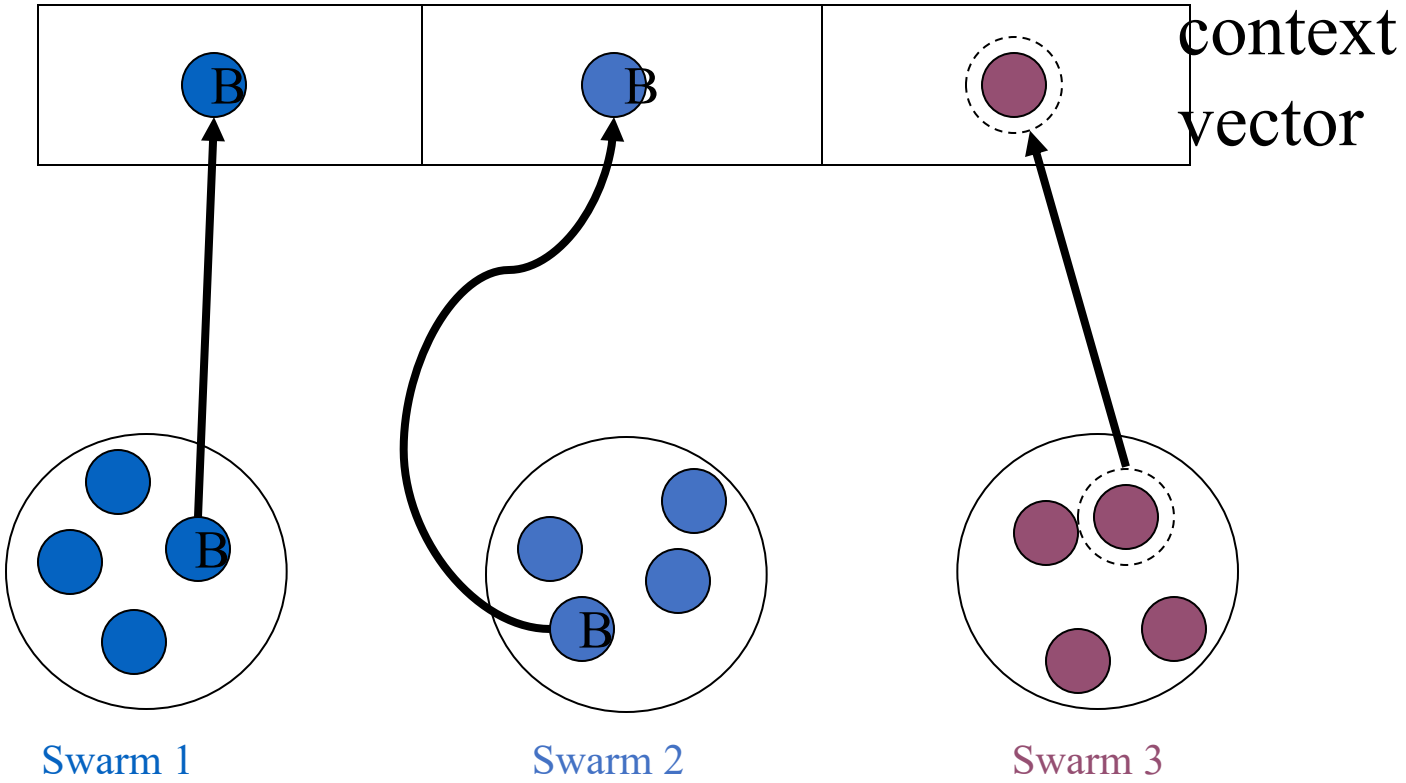
Cooperation

- A method referred to as *Cooperative Particle Swarm Optimizers (CPSO)* was proposed by van den Bergh et. al. in 2004 [14],
- Applies the same concept of the cooperative GA approach,
- Having multiple swarms where the fitness of any particle in any swarm depends on the particles of the other swarms.

Cooperation

- The general idea is to have different swarms optimizing different variables of the problem (different dimensions of the solution),
- The fitness of any particle is determined by its value and the value of the best particles in all the other swarms,
- Performs best if the problem variables are independent.

Cooperation



Cooperation

- A different approach is referred to as *Hybrid Cooperative PSO (CPSO_H)* was also proposed by van den Bergh et. al. in 2004 [14],
- Two swarms were serially updated,
- One swarm using the normal PSO algorithm and the other swarm is using CPSO.

Cooperation

- Each swarm is updated for one iteration only,
- When the PSO swarm get updated, its *gbest* values is sent to the CPSO swarm,
- The CPSO swarm uses the elements of the received *gbest* to update random particles of its sub-swarms.

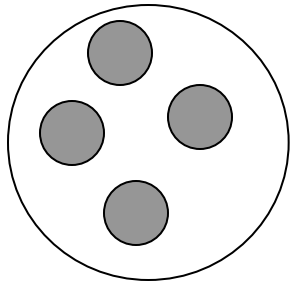
Cooperation

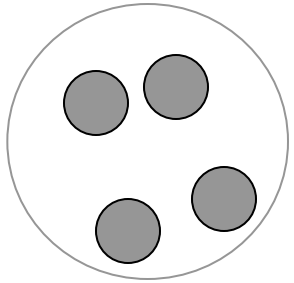
- After CPSO gets updated, it sends its context vector back to the PSO swarm,

- The PSO swarm uses the received context vector to replace a randomly chosen particle.

Cooperation

Swarm 1

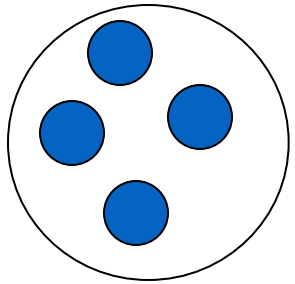


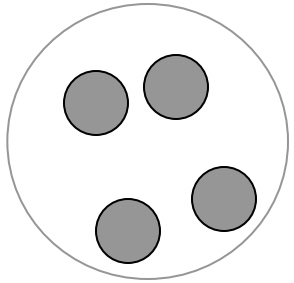


Swarm 2

Cooperation

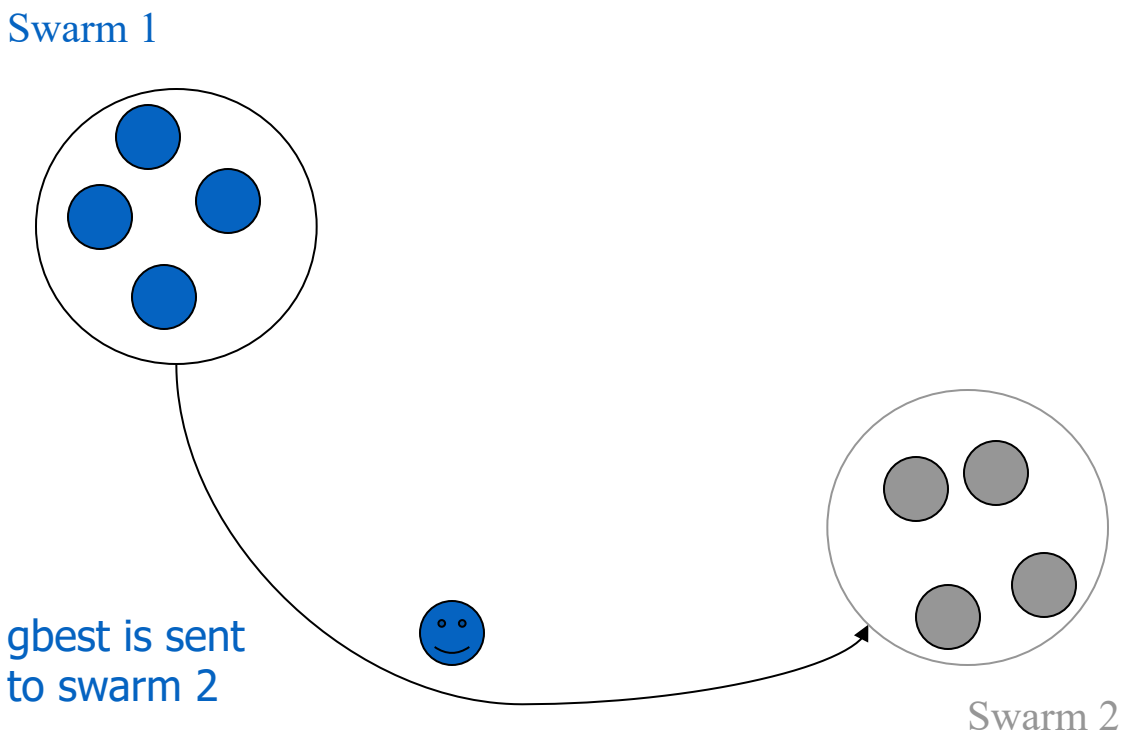
Swarm 1
gets updated



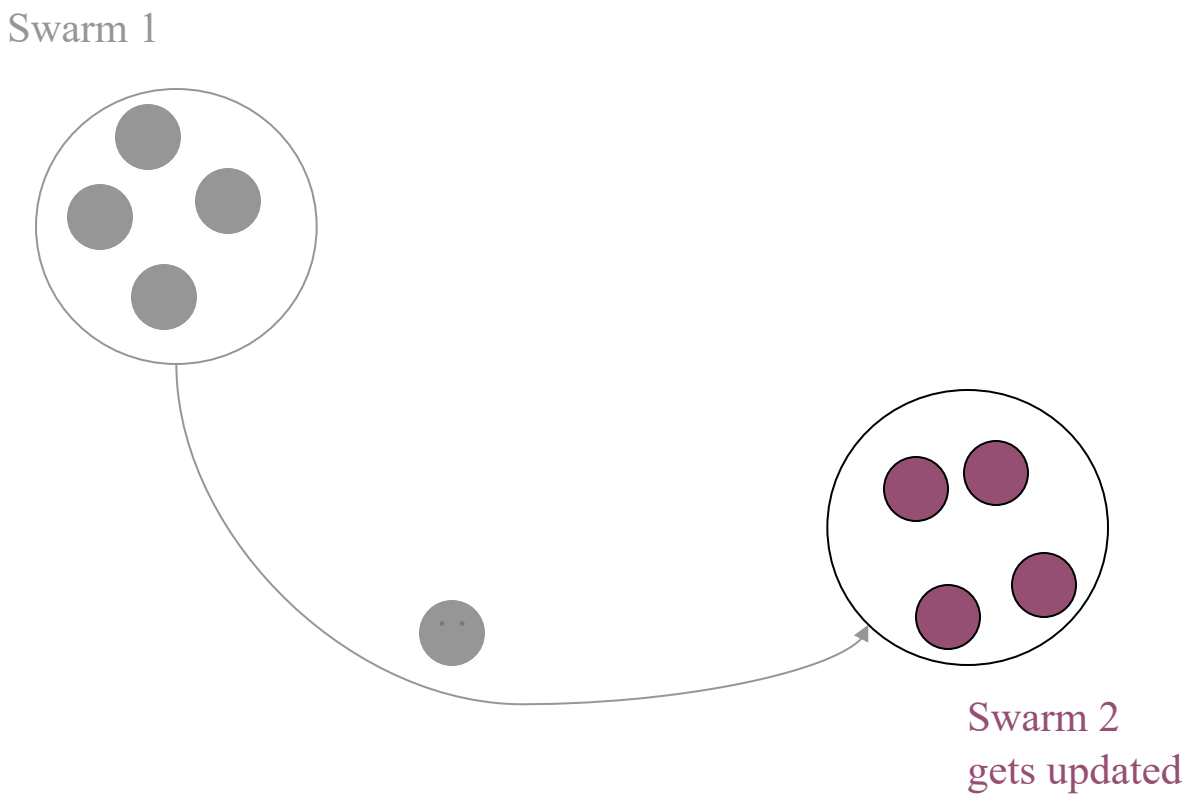


Swarm 2

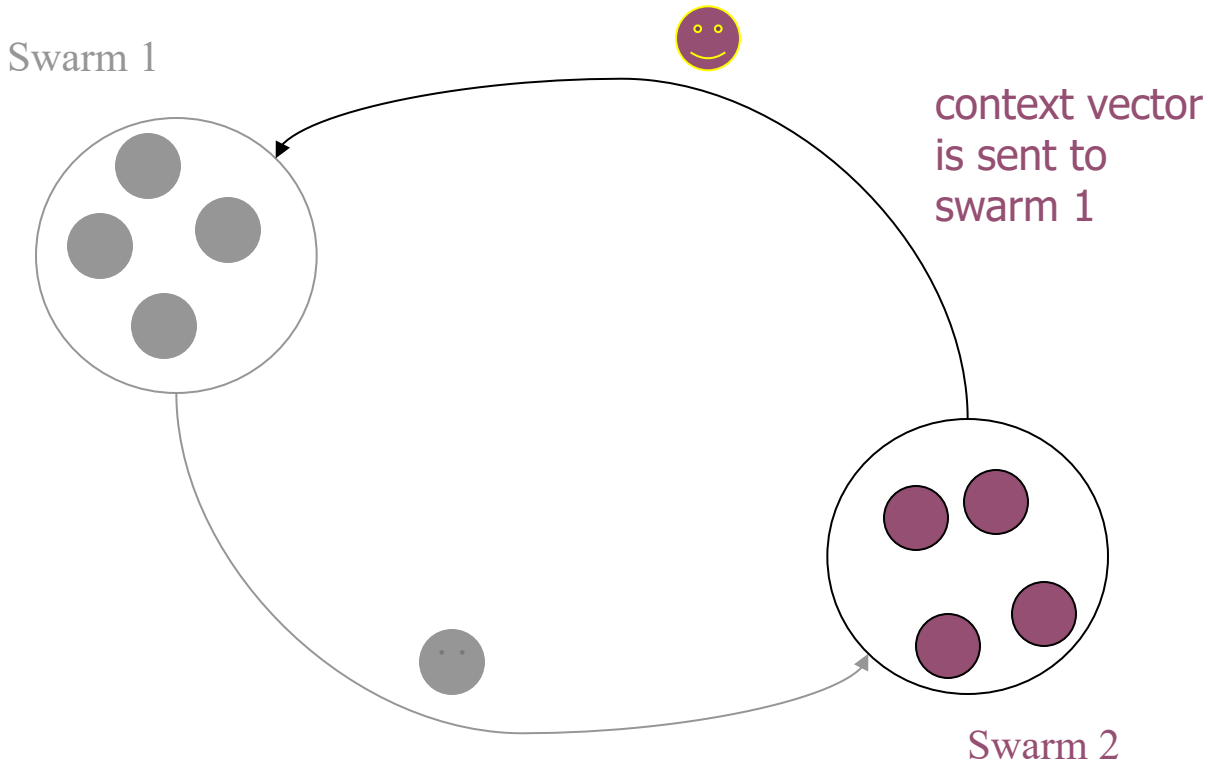
Cooperation



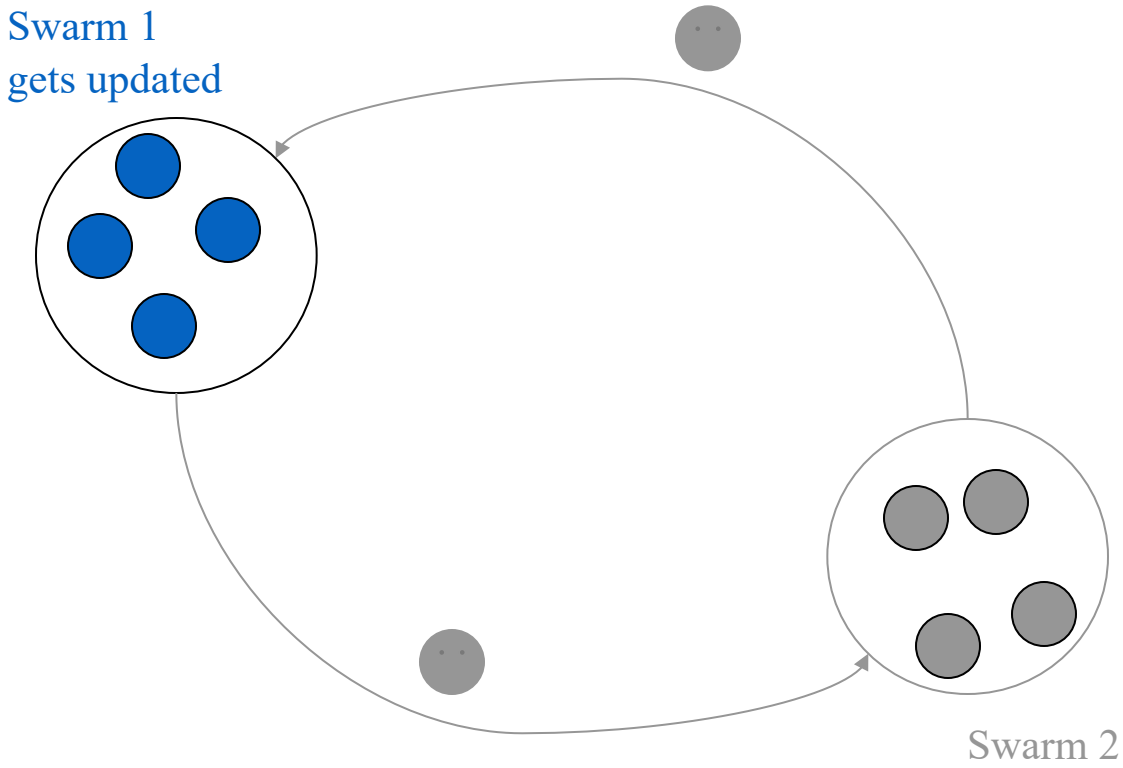
Cooperation



Cooperation



Cooperation



References

1. C. W. Reynolds.”Flocks, Herds and Schools: A Distributed Behavioural Model”. Computer Graphics, Vol. 21, Number 4, pp. 25-34, 1987.
2. C. W. Reynolds. Homepage on Boids. http://www.red3d.com/cwr/boids/
3. M. A. Montes de Oca. “ Particle Swarm Optimization – Introduction ” . Available at: http://iridia.ulb.ac.be/%7Emmontes/slidesCIL/slides.pdf
4. F. Heppner and U. Grenander, “A Stochastic Nonlinear Model for Coordinated Bird Flocks”, In S. Krasner, (ed), The Ubiquity of Chaos. AAAS pub., 1990.
5. J. Kennedy and R. C. Eberhart. “Particle Swarm Optimization”. Proceedings of the IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.
6. R. C. Eberhart and J. Kennedy. “A New Optimizer using Particle Swarm Theory,” Proceedings of the 6th International Symposium on Micro Machine and Human Science, pp. 39–43, 1995.

References

7. I. C. Trelea. ”The Particle Swarm Optimization Algorithm: Convergence Analysis and Parameter Selection”. Information Processing Letter, vol. 85, no. 6, pp. 317-325, 2003.
8. M. Clerc and J. Kennedy. “The Particle Swarm Explosion, Stability and Convergence in a Multi-dimensional Complex Space,” IEEE Transactions on Evolutionary Computation, vol. 6, no. 1, pp. 58–73, 2002.
9. R. C. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence*. PC Tools: Academic, ch. 6, pp. 212–226, 1996.
10. J. Kennedy and R. Mendes. “Neighbourhood Topologies in Fully Informed and best-of-neighbourhood Particle Swarms”. IEEE Transactions on Systems, Man and Cybernetics – Part C. Vol. 36, Issue 4, pp. 515-519, 2006.
11. J.Kennedy and R. C. Eberhart. “A Discrete Binary Version of The Particle Swarm Algorithm”. Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, pp. 4101-4109, 1997.

References

12. M. Clerc, “Discrete Particle Swarm Optimization,” in New Optimization Techniques in Engineering. Springer-Verlag, 2004.

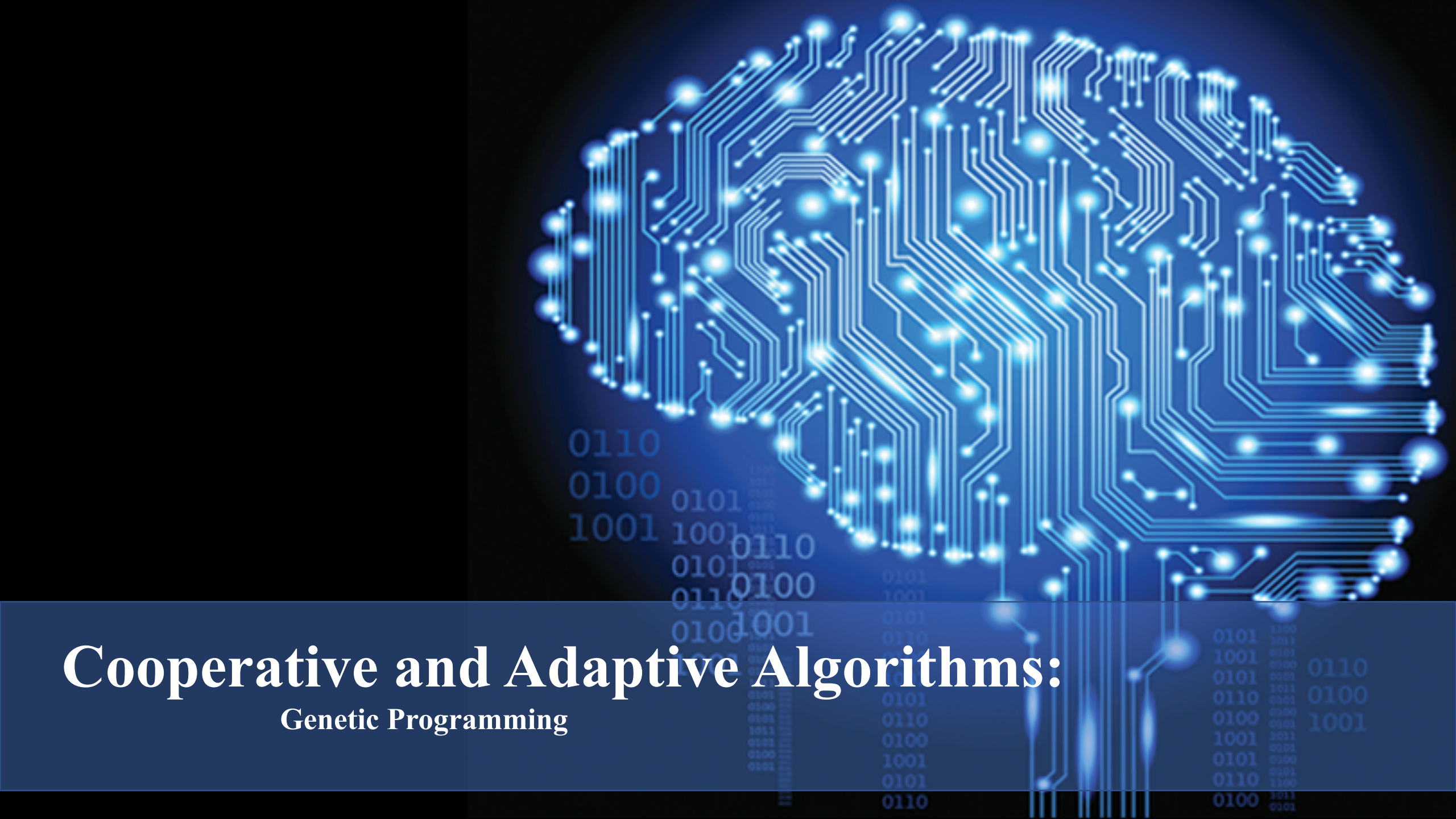
13. S. Baskar and P. N. Suganthan. “A Novel Concurrent Particle Swarm Optimizer”. Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, pp. 792-796, 2004.

14. F. van den Bergh and A. P. Engelbrecht, “A cooperative approach to Particle Swarm Optimization”. IEEE Transactions on Evolutionary Computing, vol. 8, no. 3, pp. 225–239, 2004.

15. W. Pang, K. Wang, C. Zhou, L. Dong, M. Liu, H. Zhang and J. Wang. “Modified Particle Swarm Optimization Based On Space Transformation for Solving Travelling Salesman Problem”. Proceedings of the third international conference on Machine Learning and Cybernetics, pp. 26-29, 2004.

16. K. Lei, Y. Qiu and Y. He. “A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer”. Proc. of the 1st International Symposium on Systems and Control in Aerospace and Astronautics ISSCAA, pp. 981-984,2006.

17. http://clerc.maurice.free.fr/pso/Tribes/Tribes_doc.zip



Cooperative and Adaptive Algorithms:

Genetic Programming

Genetic Programming

GP quick overview

- Developed: USA in the 1990's
- Early names: J. Koza
- Typically applied to:
  - machine learning tasks (prediction, classification…)
- Attributed features:
  - competes with neural nets and alike
  - needs huge populations (thousands)
  - slow
- Special:
  - non-linear chromosomes: trees, graphs
  - mutation possible but not necessary (disputed!)

What is Genetic Programming

Genetic Programming (**GP**) is a technique of evolving programs, starting from a population of unfit (usually random) programs, fit for a particular task by applying operations analogous to natural genetic processes to the population of programs.

It is essentially a heuristic search technique often described as 'hill climbing', i.e. searching for an optimal or at least suitable program among the space of all programs.

Outline of the Genetic Algorithm

- Randomly generate a set of possible solutions to a problem, representing each as a fixed length character string
- Test each possible solution against the problem using a fitness function to evaluate each solution
- Keep the best solutions, and use them to generate new possible solutions
- Repeat the previous two steps until either an acceptable solution is found, or until the algorithm has iterated through a given number of cycles (generations)

Why Genetic Programming?

- “It is difficult, unnatural, and overly restrictive to attempt to represent hierarchies of dynamically varying size and shape with fixed length character strings.”
- “For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program.” [Koza, 1994]
- A parse tree is a good representation of a computer program for Genetic Programming

GP technical summary tableau

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Introductory example: credit scoring

- Bank wants to distinguish good from bad loan applicants
- Model needed that matches historical data

ID	No of children	Salary	Marital status	OK?
ID-1	2	45000	Married	0
ID-2	0	30000	Single	1
ID-3	1	40000	Divorced	1
…				

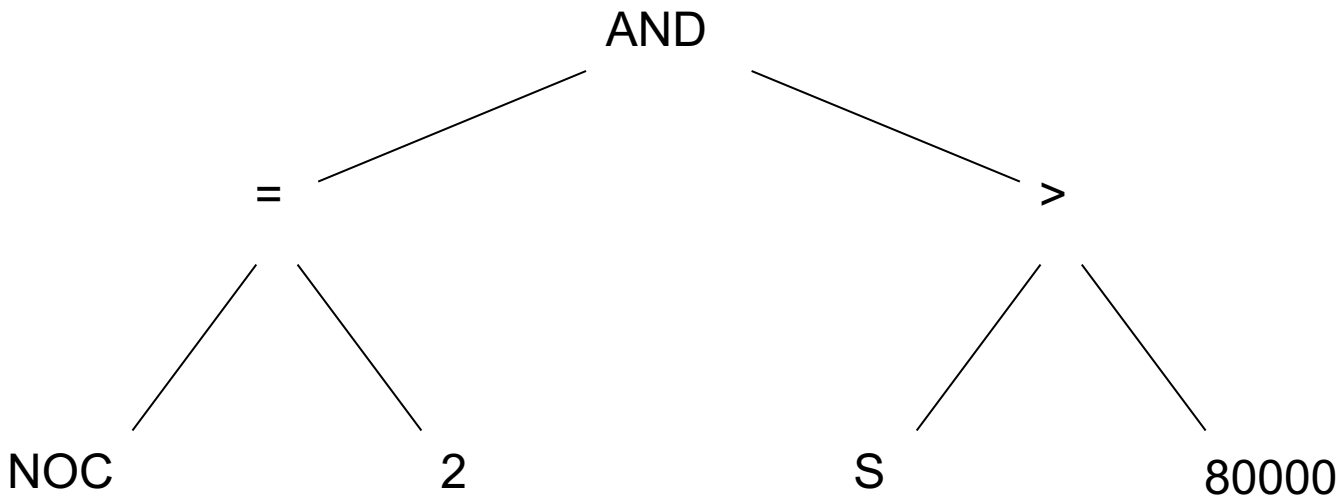
Introductory example: credit scoring

- A possible model:
  - IF (NOC = 2) AND (S > 80000) THEN good ELSE bad
- In general:
  - IF formula THEN good ELSE bad
- Only unknown is the right formula, hence
- Our search space (phenotypes) is the set of formulas
- Natural fitness of a formula: percentage of well classified cases of the model it stands for
- Natural representation of formulas (genotypes) is: parse trees

Introductory example: credit scoring

IF (NOC = 2) AND (S > 80000) THEN good ELSE bad

can be represented by the following tree



Tree based representation

- Trees are a universal form, e.g. consider
-
- Arithmetic formula

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

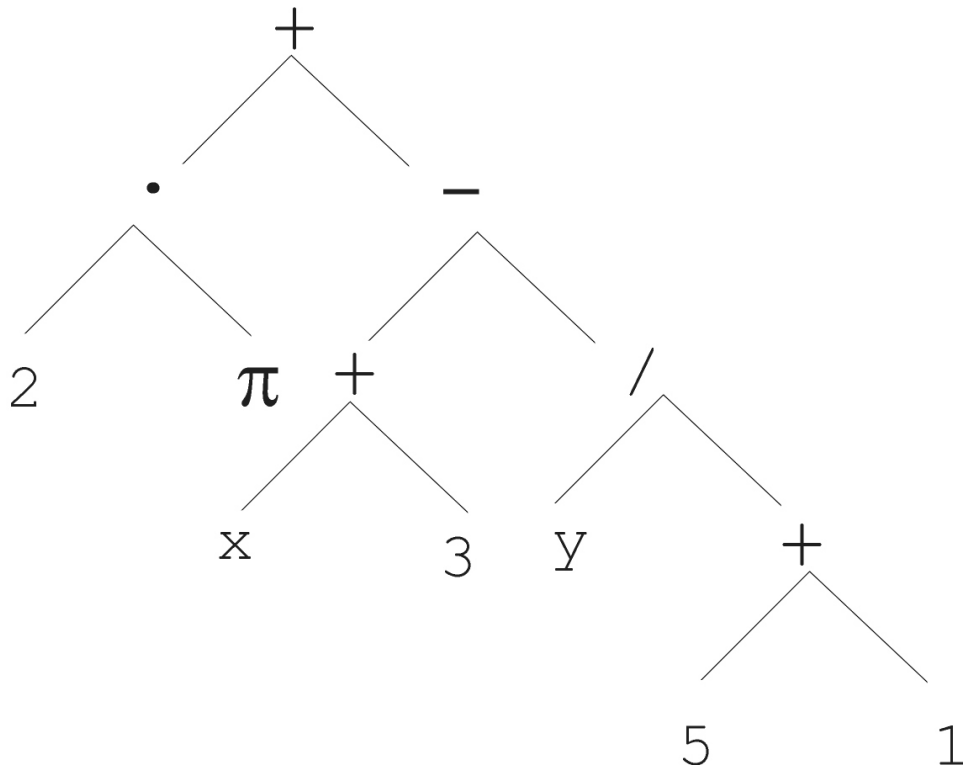
- Logical formula

$$(x \wedge \text{true}) \rightarrow ((\ x \vee y\ ) \vee (z \leftrightarrow (x \wedge y)))$$

- Program

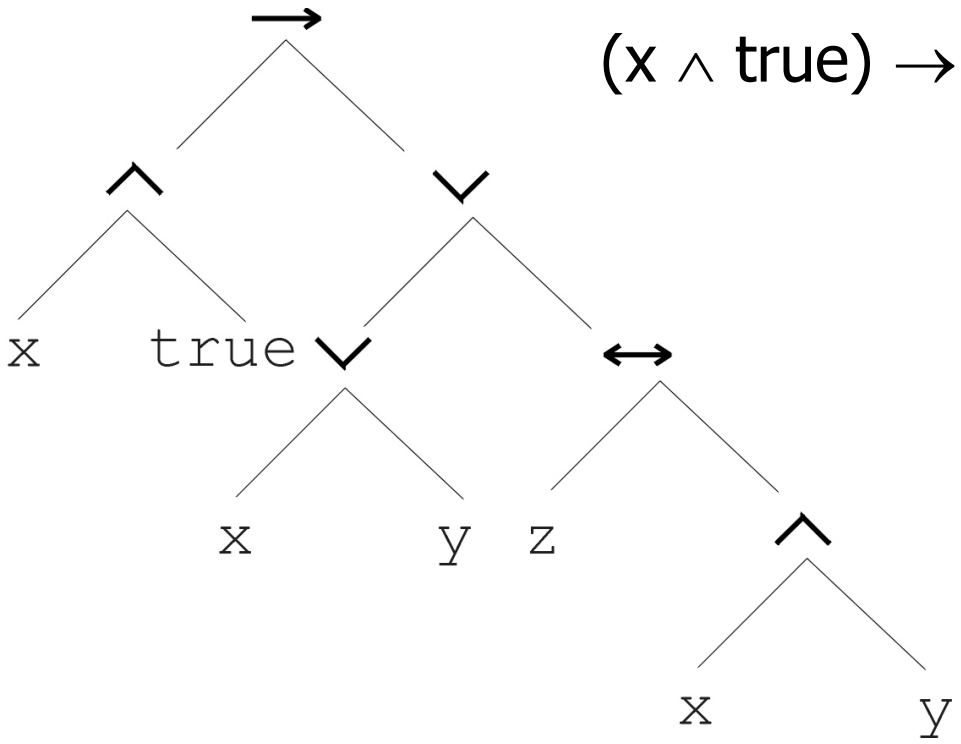
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



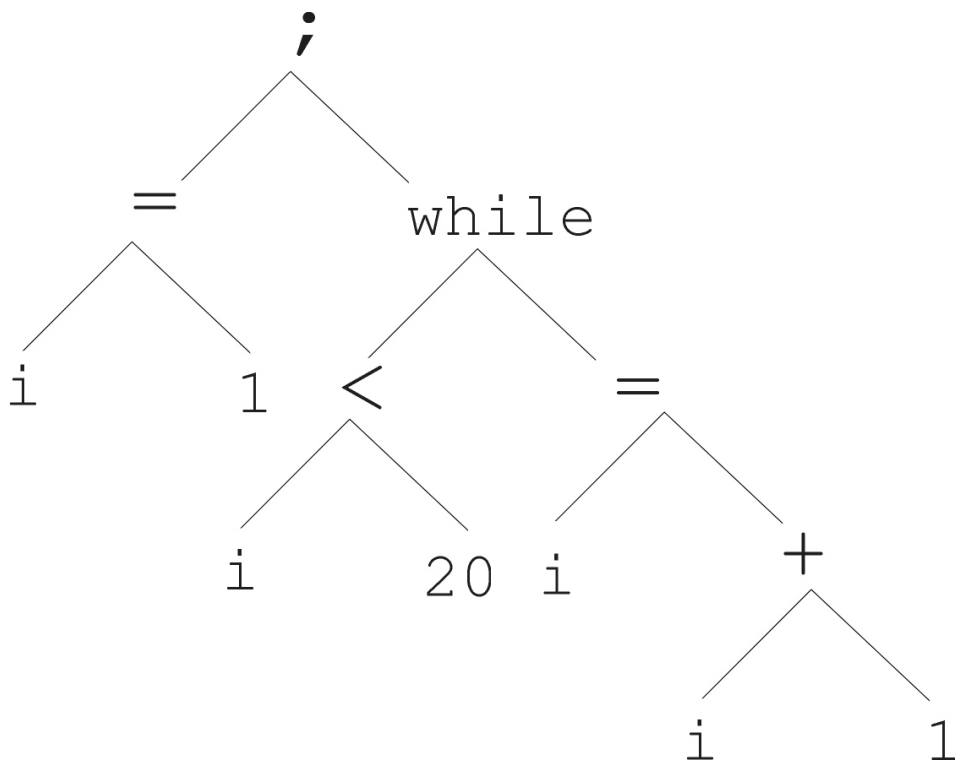
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree based representation



$(x \wedge \text{true}) \rightarrow ((\ x \vee y\ ) \vee (z \leftrightarrow (x \wedge y)))$

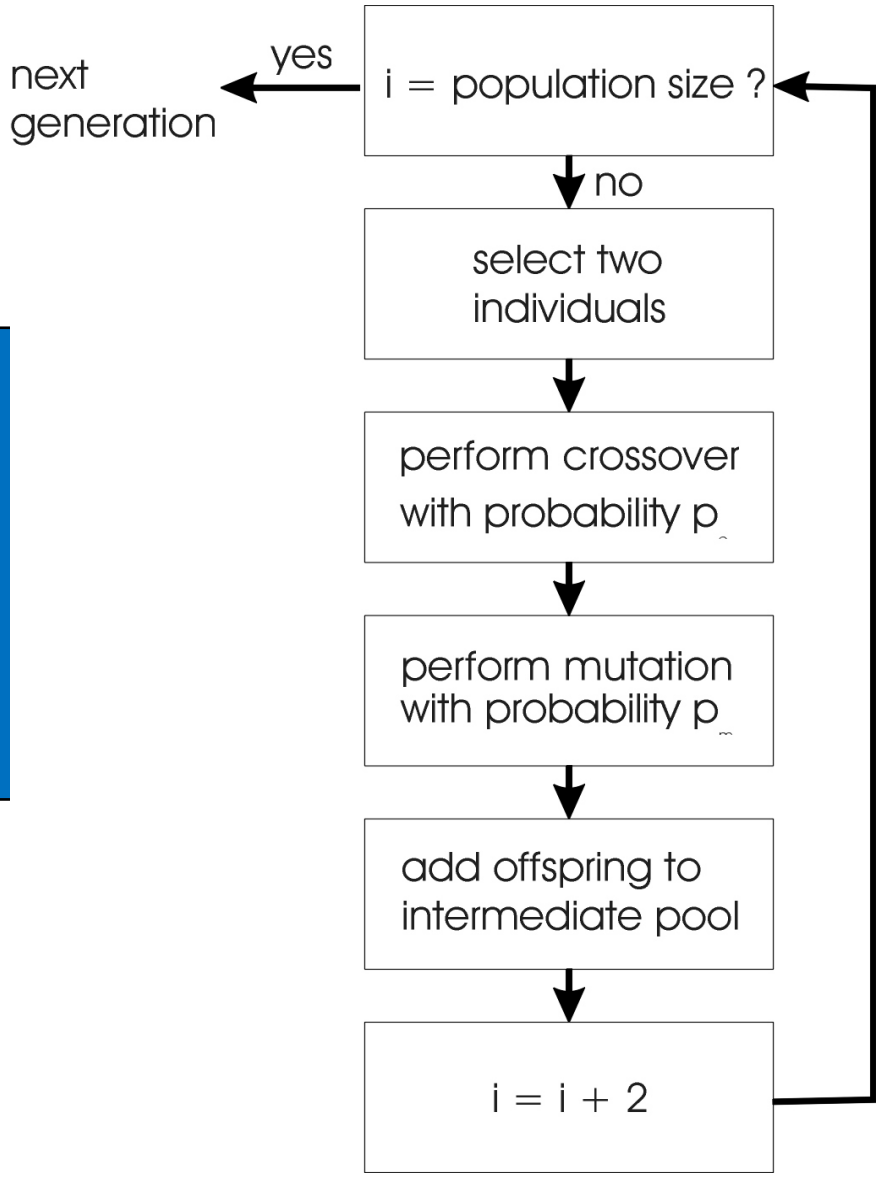
Tree based representation



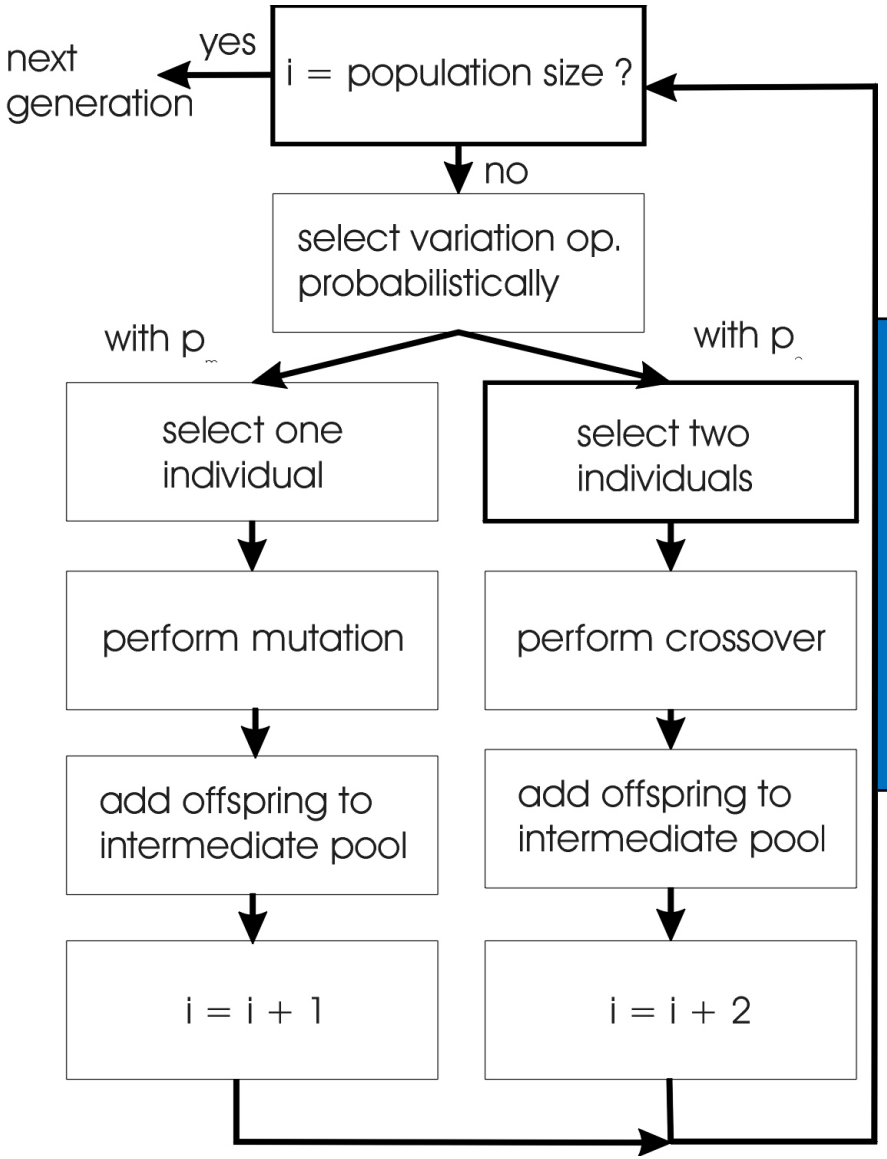

```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation

- In GA, ES, EP chromosomes are linear structures (bit strings, integer string, real-valued vectors, permutations)
- Tree shaped chromosomes are non-linear structures
- In GA, ES, EP the size of the chromosomes is fixed
- Trees in GP may vary in depth and width



GA flowchart



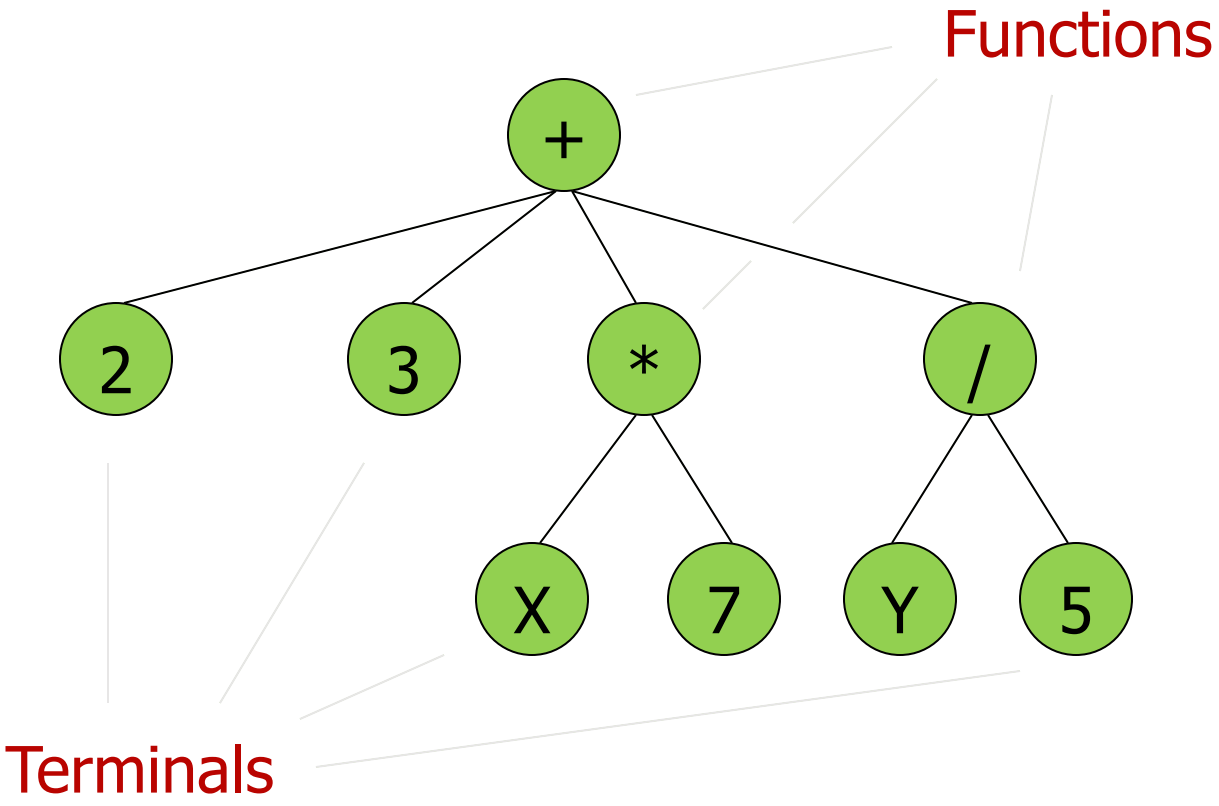
GP flowchart

Tree based representation

- Symbolic expressions can be defined by
  - Terminal set T
  - Function set F (with the arities of function symbols)
- Adopting the following general recursive definition:
  - Every $t \in T$ is a correct expression
  - $f(e_1, \ldots, e_n)$ is a correct expression if $f \in F$, arity(f)=n and $e_1, \ldots, e_n$ are correct expressions
  - There are no other forms of correct expressions
- In general, expressions in GP are not typed (closure property: any $f \in F$ can take any $g \in F$ as argument)

Using Trees To Represent Computer Programs

(+ 2 3 (* X 7) (/ Y 5))



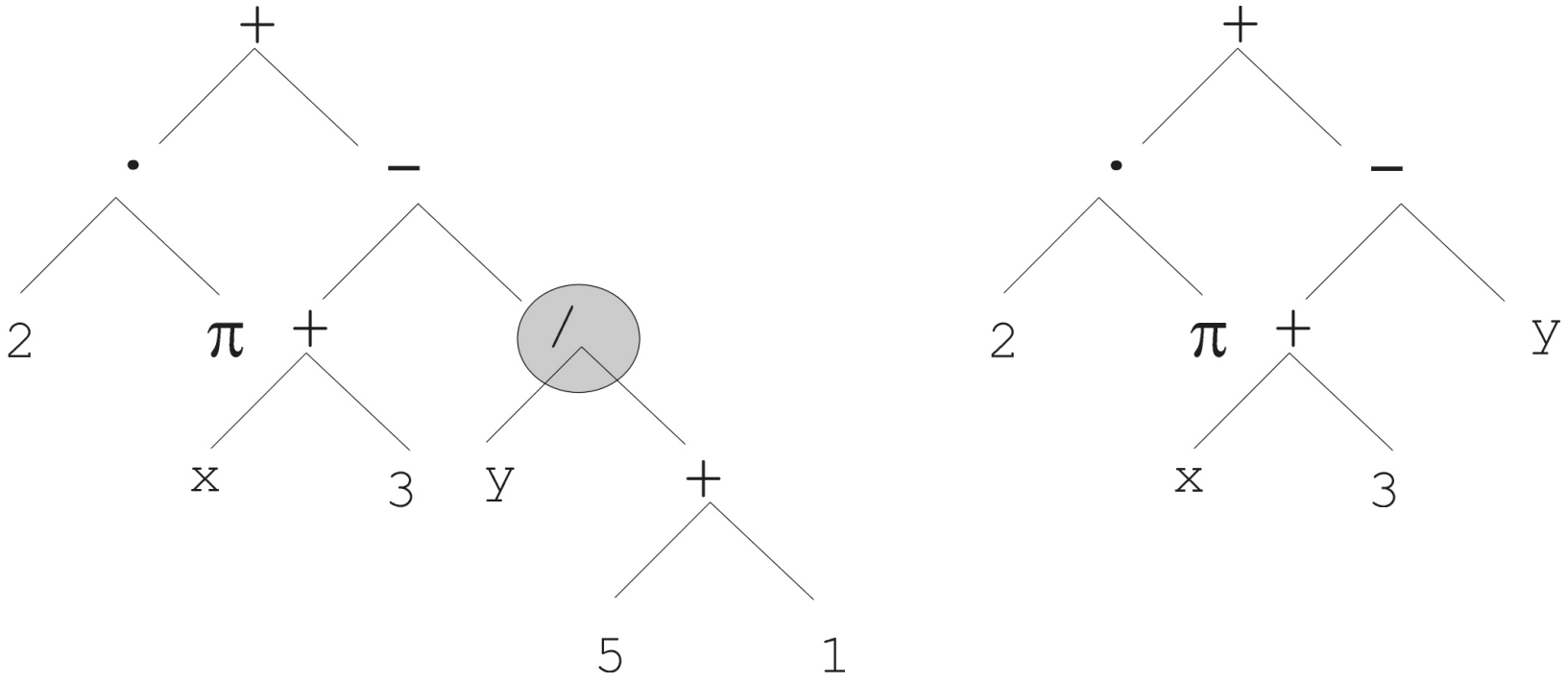
Offspring creation scheme

Compare

- GA scheme using crossover AND mutation sequentially (be it probabilistically)
- GP scheme using crossover OR mutation (chosen probabilistically)

Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree

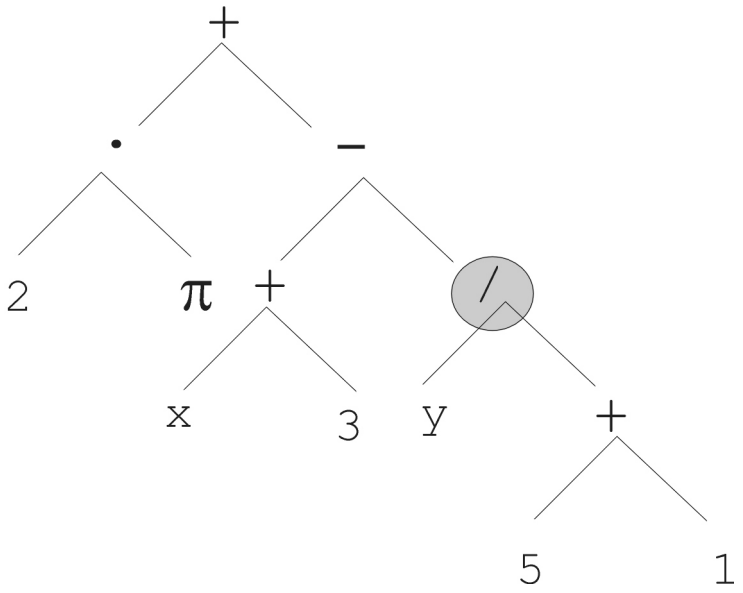


Mutation cont'd

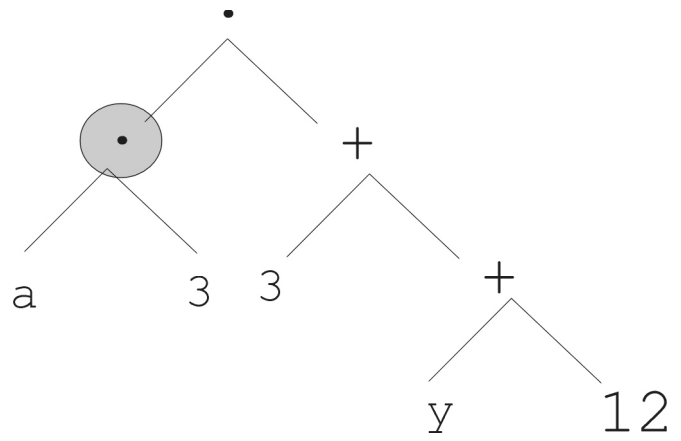
- Mutation has two parameters:
  - Probability p_m to choose mutation vs. recombination
  - Probability to chose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be 0 (Koza'92) or very small, like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

Recombination

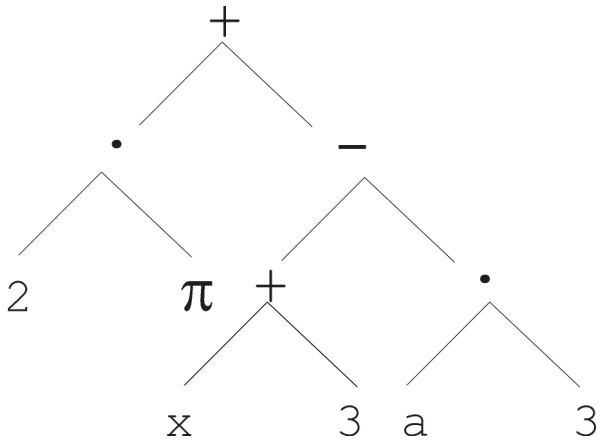
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
  - Probability p_c to choose recombination vs. mutation
  - Probability to chose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents



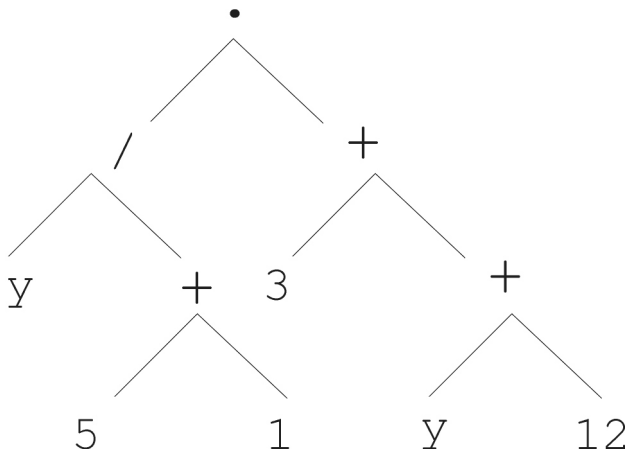
Parent 1



Parent 2



Child 1



Child 2

Selection

- Parent selection typically fitness proportionate
- Over-selection in very large populations
  - rank population by fitness and divide it into two groups:
  - group 1: best x% of population, group 2 other (100-x)%
  - 80% of selection operations chooses from group 1, 20% from group 2

Initialisation

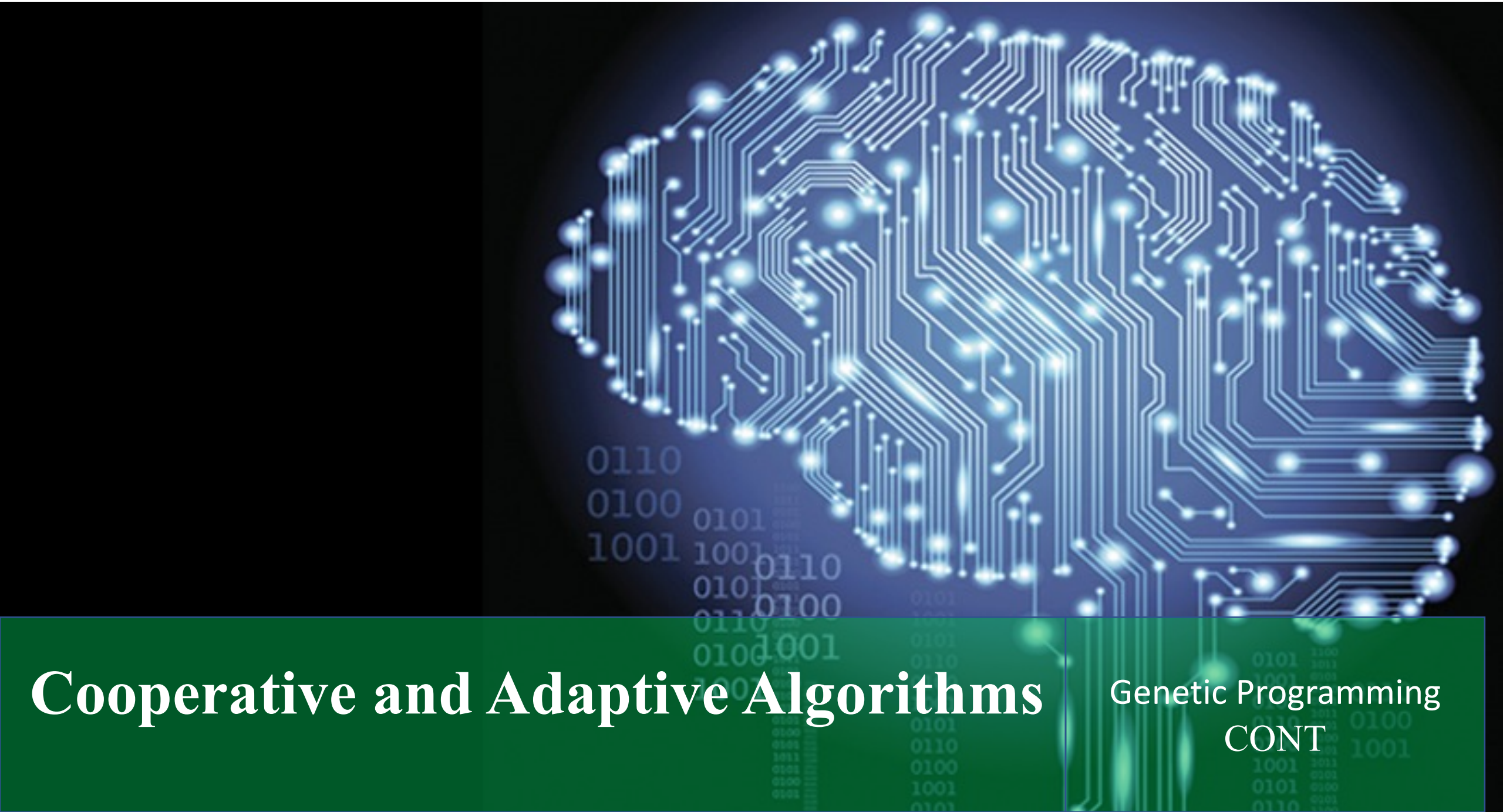
- Maximum initial depth of trees D_{max} is set
- Full method (each branch has depth = D_{max}):
  - nodes at depth $d < D_{max}$ randomly chosen from function set F
  - nodes at depth $d = D_{max}$ randomly chosen from terminal set T
- Grow method (each branch has depth $\leq D_{max}$):
  - nodes at depth $d < D_{max}$ randomly chosen from $F \cup T$
  - nodes at depth $d = D_{max}$ randomly chosen from T
- Common GP initialisation: ramped half-and-half, where grow & full method each deliver half of initial population

Bloat

- Bloat = “survival of the fattest”, i.e., the tree sizes in the population are increasing over time
- Ongoing research and debate about the reasons
- Needs countermeasures, e.g.
  - Prohibiting variation operators that would deliver “too big” children
  - Parsimony pressure: penalty for being oversized

Problems involving “physical” environments

- Trees for data fitting vs. trees (programs) that are “really” executable
- Execution can change the environment → the calculation of fitness
- Example: robot controller
- Fitness calculations mostly by simulation, ranging from expensive to extremely expensive (in time)
- But evolved controllers are often very good

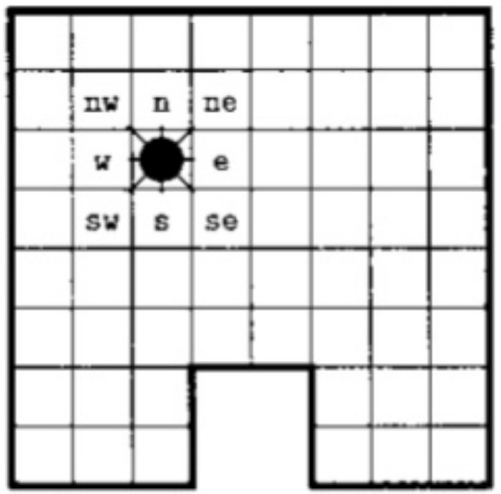


Cooperative and Adaptive Algorithms

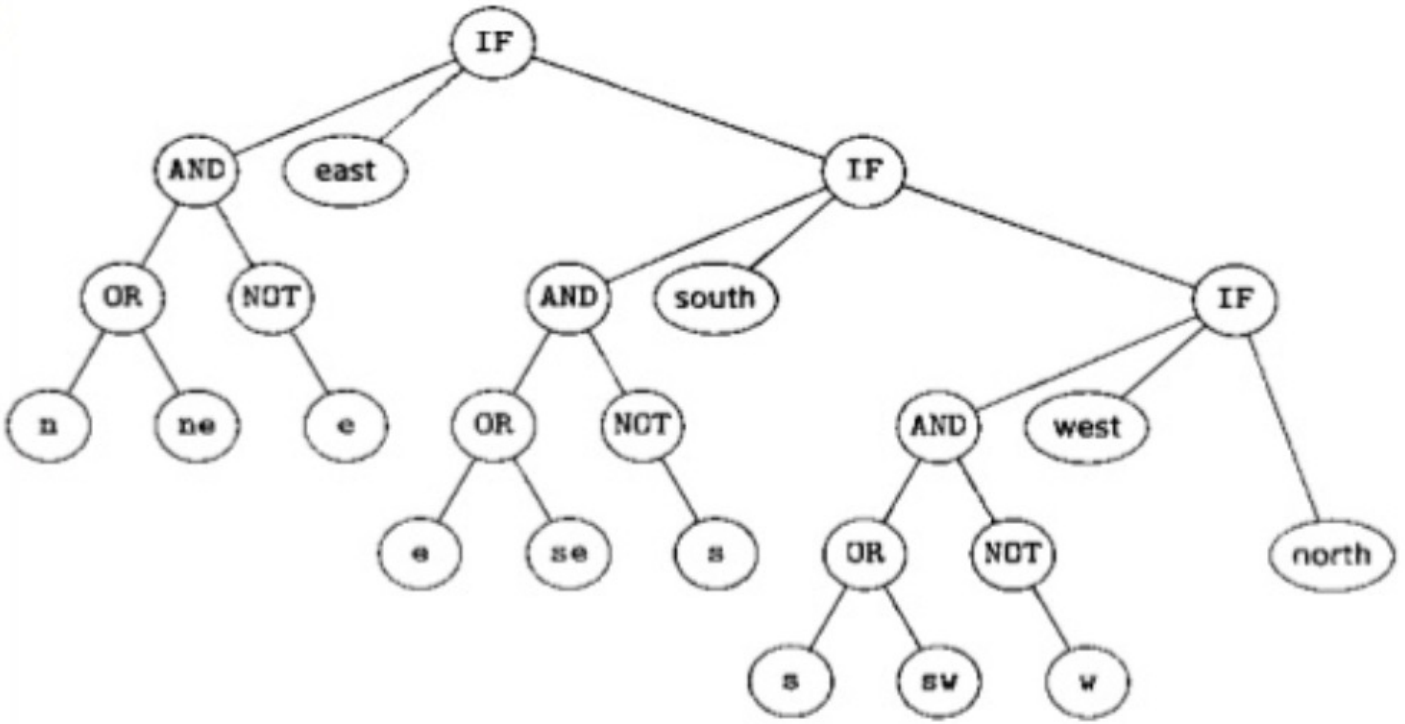
Genetic Programming
CONT

A WALL FOLLOWING ROBOT

- Program Representation in GP
  - Functions
    - AND (x, y) = 0 if x = 0; else y
    - OR (x, y) = 1 if x = 1; else y
    - NOT (x) = 0 if x = 1; else 1
    - IF (x, y, z) = y if x = 1; else z
  - Terminals
    - Actions: move the robot one cell to each direction
      {north, east, south, west}
    - Sensory input: its value is 0 whenever the coressponding cell is free for the robot to occupy; otherwise, 1.
      {n, ne, e, se, s, sw, w, nw}

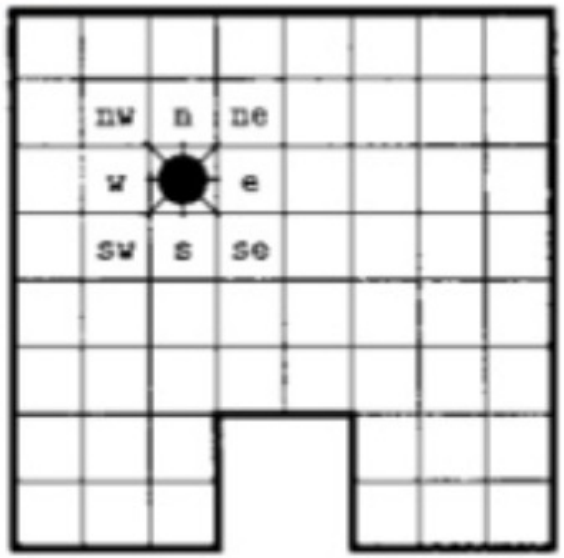


A WALL FOLLOWING PROGRAM



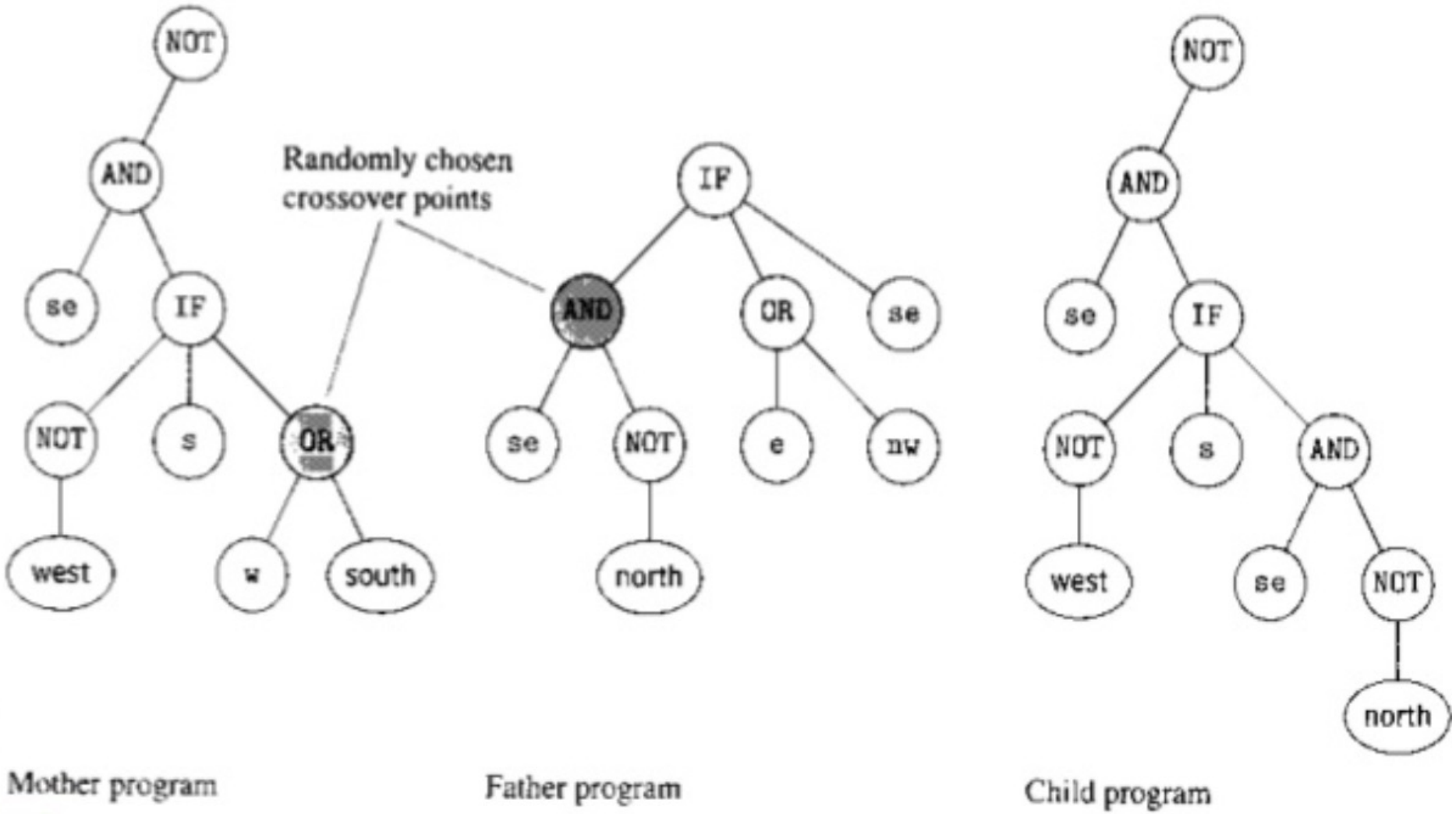

```
(IF (AND (OR (n) (ne)) (NOT (e)))
    (east)
    (IF (AND (OR (e) (se)) (NOT (s)))
        (south)
        (IF (AND (OR (s) (sw)) (NOT (w)))
            (west)
            (north))))
```

- Experimental Setup
  - **Population** size: 5,000
  - **Fitness measure**: the number of cells next to the wall that are visited during 60 steps
    - Perfect score (320)
      - One Run (32) × 10 randomly chosen starting points
  - **Termination condition**: found perfect solution
  - **Selection**: tournament selection

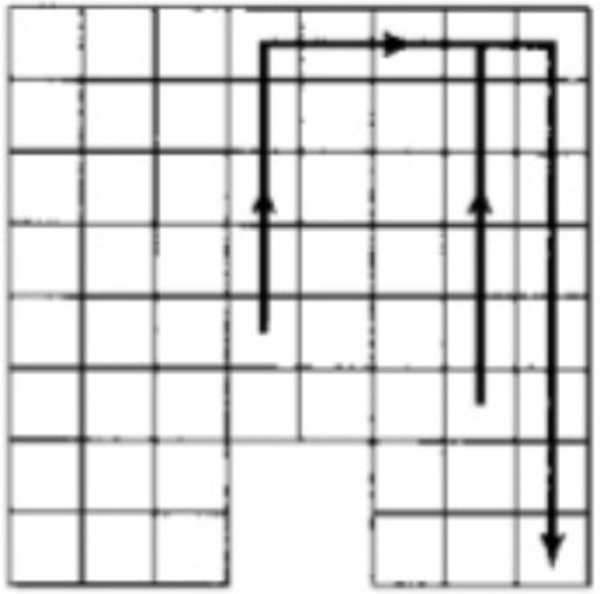


EVOLVING THE ROBOT

- Creating Next Generation
  - 500 programs (10%) are copied directly into next generation.
    - Tournament selection
      - 7 programs are randomly selected from the population 5,000.
      - The most fit of these 7 programs is chosen.
  - 4,500 programs (90%) are generated by crossover.
    - A mother and a father are each chosen by tournament selection.
    - A randomly chosen subtree from the father replaces a randomly selected subtree from the mother.
  - In this example, mutation was not used.

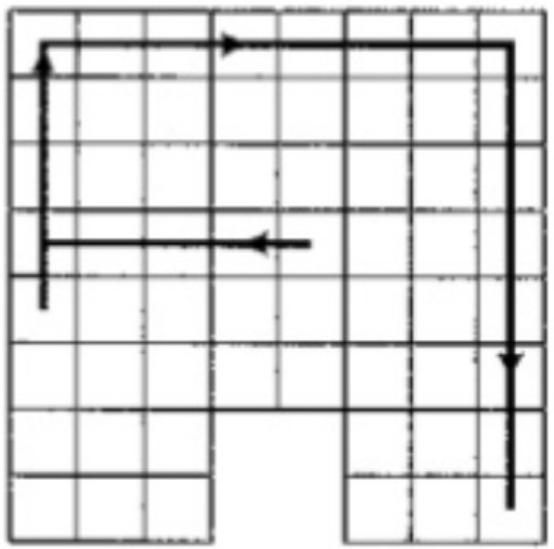

NOT
AND
se
IF
NOT
s
OR
west
w
south
Randomly chosen
crossover points
IF
AND
OR
se
se
NOT
e
nw
north
NOT
AND
se
IF
NOT
s
AND
west
se
NOT
north
Mother program
Father program
Child program

- Generation 2
  - The most fit program (fitness = 117)
    - Smaller than the best one of generation 0, but it does get stuck in the lower-right corner.



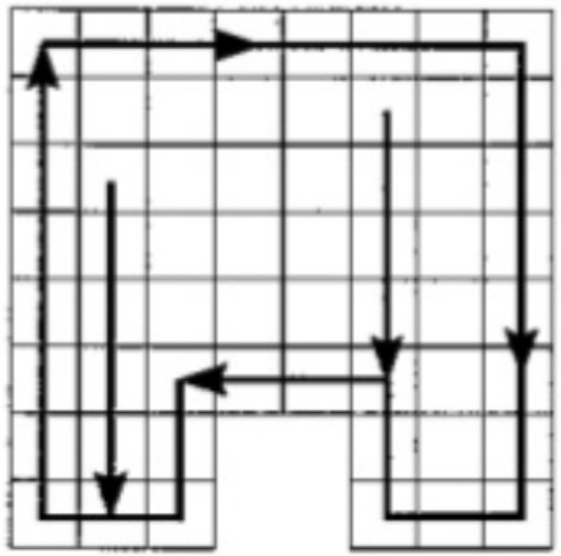
```
(NOT (AND (IF (ne)
              (IF (se)(south)(east))
              (north))
          (NOT (NOT (e)))))
```

- Generation 6
  - The most fit program (fitness = 163)
    - Following the wall perfectly but still gets stuck in the bottom-right corner.



```
(IF (AND (NOT (e))
         (IF (e)(s)(nw)))
    (OR (IF (1)(e)(south))
        (IF (north)(east)(nw)))
    (IF (OR (AND (0)(north))
            (AND (e)(IF (e)
                        (IF (se)(south)(east))
                        (north))))
        (AND (e)
             (NOT (IF (s)(sw)(e))))
        (OR (OR (AND (nw)(east))
                (west))
            (nw))))
```

- Generation 10
  - The most fit program (fitness = 320)
    - Following the wall around clockwise and moves south to the wall if it doesn't start next to it.



```
(IF (IF (IF (se)(0)(ne))
        (OR (se)(east))
        (IF (OR (AND (e)(0))
                (sw))
            (OR (sw)(0))
            (AND (NOT (NOT (AND (s)(se))))
                 (se))))
    (IF (w)
        (OR (north)
            (NOT (NOT (s))))
        (west))
    (NOT (NOT (NOT (AND (IF (NOT (south))
                            (se)
                            (w))
                        (NOT (n)))))))
```

- Fitness Curve
  - Fitness as a function of generation number
    - The progressive (but often small) improvement from generation to generation