AI- Search, Metaheuristics, Machine Learning, and Deep Learning

Department of Electrical and Computer Engineering
University of Waterloo

Intelligence

One may think of intelligence as the ability to acquire and apply knowledge and skills, which ultimately reflects on our consequent ability to solve complex problems, and/or make decisions that we estimate to produce desirable outcomes and/or those that prevent undesirable outcomes.

Intelligence

- It is fallacious to think of the situations we encounter in our life to be such that making good decisions is a matter of picking up proper decisions and/or solutions from a look-up table; like a simple one-to-one mapping.
- Things and matters in our life are dynamic and uncertain. Our view of them tend to be the result of our limited sensing and perception, gut-feeling, and what others tell us. With all this, we humans, and this applies to our other contemporaries, managed to survive and got better at it as time passes (despite our stumbling at times).
- Thanks to three things we do well: We are good at learning (in acquiring knowledge and we are good at passing our learning to others), we are good at adapting, and we are good in amplifying our ability and efficiency through cooperation.

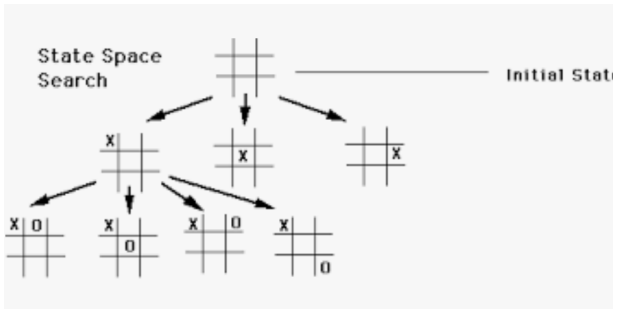
COMPUTING: INTELLIGENT VS CONVENTIONAL

First, we started by us doing the learning and using what we learn to program the computers to do the tasks on our behalf. This is conventional computing.

The question we wish to pose at this juncture, as computers have become super fast: if learning, adaptation and cooperation are that powerful, can we clone them into the things we make- i.e., computers? That is, can we clone our intelligence into the computing systems and devices we make?

Solution/State Space

- We hypothesized the existence of a solution to a problem of interest.
- The solution is reachable by making moves or taking actions in some state space.
- The state space that is made of connected states. Each state could be either the solution we are looking for (or goal state), or a step in our way towards another state that is the solution state, if a solution exits.
- A state is connected to another state, if and only if, it is possible to generate one state from the other, as dictated by the nature of the problem (call it the rules of the game) or the solution process.
- The connection/edge between any two states could represent an action, a choice, a solution component, etc.
- Often, such connectivity is modeled such that the interpretation of the relationship between the two states is embedded in the structure, like in the case of the tree representation, in which a child node (or state) is implicitly considered as a possible next state of the parent state.



State space Search

Blind or uninformed Search

- We studied exhaustive solution searching techniques that examine every state they encounter in their sweeping or their construction process of the state space.
- They know nothing about where to concentrate their search effort. They are blind.
- The value these techniques offer revolves around the way they organize their state-space sweeping/construction process.
- Some of these organize their sweeping process such that whenever they are in a state on the search path, they check every immediate next state connected to this current state. We refer to them as BFS techniques (Breadth First Search). They are driven by completeness more than efficiency. They are horizontally optimistic!
- Some organize their state-space sweeping/construction process in a depth oriented manner, that is grandchildren before children. We refer to this category as DFS (Depth First Search). Again, driven by completeness, but vertically optimistic.
- Our choice of one over the other is largely memory driven.

The Value of Heuristics?

- Sometimes, as we start our search and move from one state to the next on our search path, we look around to discover or learn hints that can help us orient towards the goal state. We referred to these hints as heuristics.
- This presents an interesting development since the search algorithm is no longer just a set of organized steps to examine the space, but rather can make use of external information that can help it optimize its search.
- Two factors decide the goodness of the search:
- 1) how good is the hint (heuristic): i.e, admissible, consistent, dominant;
- 2) and how smart is the search strategy: optimal, complete, and at what cost.

What heuristics can do is providing the search algorithm with problem specific information that may help it concentrate its search effort towards the goal state.

The Value of Metaheuristics?

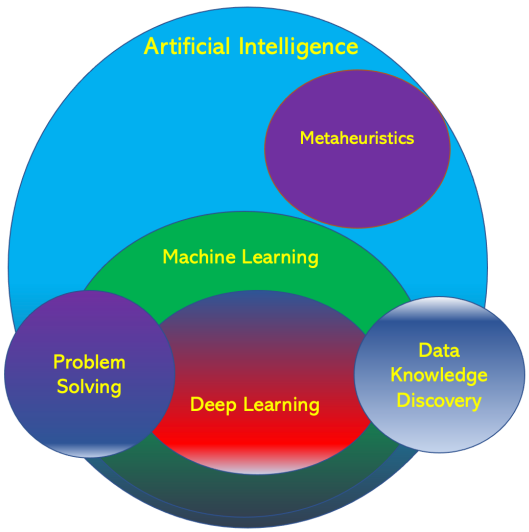
- In Metaheuristics, however, we move up to a higher level where the algorithm is able to exploit more strategic hints to change the way it goes about the problem.
- Think of it to as a higher mechanism designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to a problem.
- Metaheuristics represent more general approximation algorithms applicable to a large variety of optimization problems. Their design principles such that they can be tailored to solve any optimization problem.
- Metaheuristics solve instances of problems that are believed to be hard in general, by exploring the usually large solution search space of these instances.
- These algorithms achieve this by reducing the effective size of the space and by exploring that space efficiently.

The Value of Metaheuristics?

Metaheuristics serve three main purposes: solving problems faster, solving large problems, and forming the base for robust algorithms. Moreover, they are flexible, simple to design and implement.

Heuristics, Metaheuristics, AI and ML: how are they connected?

- One can think of Heuristics and Metaheuristics as computational intelligence paradigms. They possess some smartness in approaching problems in a pragmatic manner. Furthermore, they are often used as building blocks of artificially intelligent systems. Thus, they are considered as a subspace or a branch of AI.
- Machine learning, which is also a branch of Artificial Intelligence (AI) and computer science, focuses on the use of **data** and algorithms to imitate the way humans learn, and gradually achieve better performance qualities in what they do.
- Deep learning: A subset of machine learning, where algorithms are created and operated similar to machine learning, but there are many layers of these algorithms, each providing a different interpretation of the data it feeds.



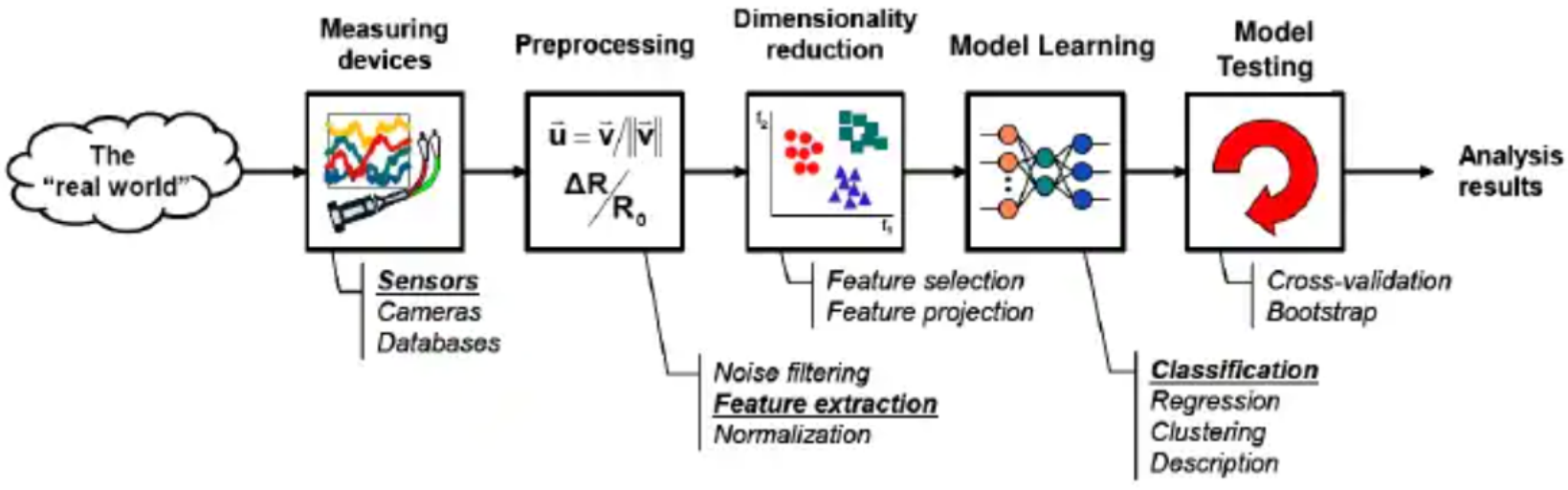
AI Paradigms

ML and ES

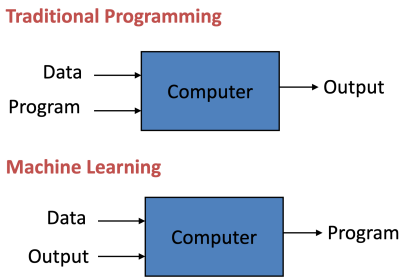
The concept of learning in a ML system
- Learning = Improving with experience at some task
- Improve over task T,
- With respect to performance measure, P
- Based on experience, E.

Motivating Example Learning to Filter Spam
Spam - is all email the user does not want to receive and has not asked to receive
T : Identify Spam Emails
P % of spam emails that were filtered % (non-spam) emails that were incorrectly filtered-out
E : a database of emails that were labelled by users

THE LEARNING PROCESS



Machine Learning and Optimization



- Machine learning is concerned with using an algorithm to learn and generalize from historical data in order to come up with a model that governs that data to the extent it can perform predictions on new data.
- This problem can be stated as approximating a function that maps examples of inputs to examples of outputs. Approximating a function can in turn be stated as function optimization.
- This is where a machine learning algorithm defines a parameterized mapping function (e.g. a weighted sum of inputs)
- An optimization algorithm is used to find the values of the parameters (e.g. model coefficients) that minimize the error of the function when used to map inputs to outputs.
- Thus, this implies that each time we fit a machine learning algorithm on a training dataset, we are solving an optimization problem.

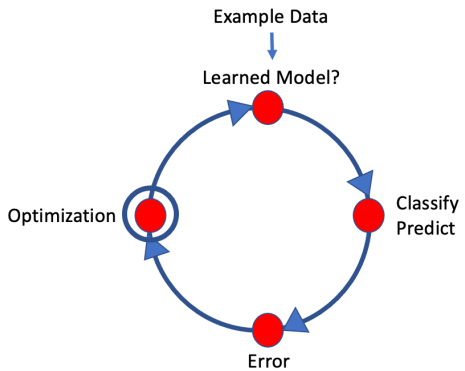
Learning as an Optimization

Given X and new examples of X, $(\hat{X})$, we must map each example onto the expected output value $(\hat{y})$ using our learned function $(\hat{f})$.

$$\hat{y} = \hat{f}(\hat{X}) \tag{1}$$

- The learned mapping will be imperfect. No model is perfect, and some prediction error is expected given the difficulty of the problem, noise in the observed data, and the choice of learning algorithm.
- Mathematically, learning algorithms solve the problem of approximating the mapping function by solving a function optimization problem.
- Specifically, given examples of inputs and outputs, find the set of inputs to the mapping function that results in the minimum loss, minimum cost, or minimum prediction error.
- The more biased or constrained the choice of mapping function, the easier the optimization is to solve.

Machine Learning and Optimization



Metaheuristics: Learning Optimization

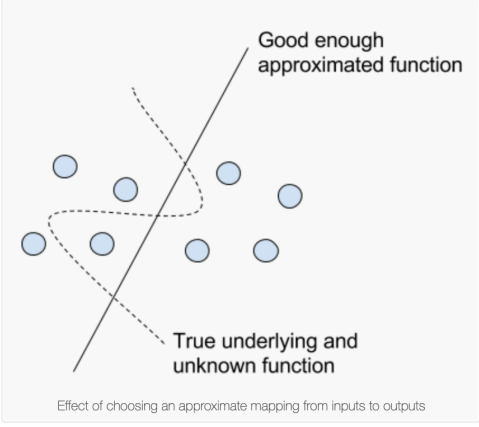
- As we learned in this course, function optimization is the problem of finding the set of inputs to a target objective function that result in the minimum or maximum of the function.
- It can be a challenging problem as the function may have tens, hundreds, thousands, or even millions of inputs, and the structure of the function is unknown, and often non-differentiable and noisy.
- It can be challenging as there is often a limited number of examples from which we can approximate the function, and the structure of the function that is being approximated is often nonlinear, noisy, and may even contain contradictions.

Heuristics, Metaheuristics, AI and ML: how are they connected?

Finding such a model, that is learning the model from the data, can basically be stated as a search for a solution to an optimization problem. And as such, representing a connection between Machine Learning algorithms and their siblings in AI, vis-a-vis,the tools of computational intelligence.

In what follows we will spend some time on the topics of Machine Learning and Deep Learning and shed some light on the utility of search and meatheuristics techniques in designing Machine Learning models.

THE LEARNED MAPPING WILL BE IMPERFECT.



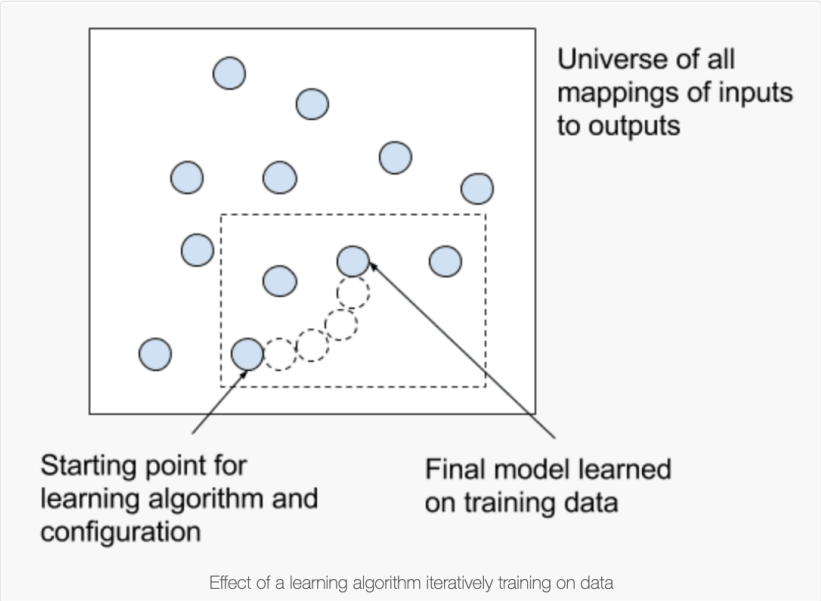
Good enough approximation

The problem of designing and developing a learning system is the problem of learning an approximate of the unknown underlying function that maps the input variables to the output variables.

We have no knowledge of the mapping function, and we hope that data would reveal it.In some cases we can gauge how close the function we learned from the data is to the true underlying function. In some other cases we may never know how close of an approximation the learning system is to the true mapping.

Implications of Machine Learning as Search

- This conceptualization of developing learning systems as a search problem helps to make clear many related concerns in applied machine learning.
- Algorithms that Learn Iteratively: The algorithm used to learn the mapping will impose further constraints, and it, along with the chosen algorithm configuration, will control how the space of possible candidate mappings is navigated as the model is fit (e.g. for machine learning algorithms that learn iteratively).
- Here, we can see that the act of learning from training data by a machine learning algorithm is in effect navigating the space of possible mappings for the learning system, hopefully moving from poor mappings to better mappings (e.g. hill climbing).



Effect of a learning algorithm iteratively training on data

Parameters Optimization of Deep Learning Models using Particle Swarm Optimization

- The goodness of a machine learning systems depends upon appropriately setting its parameters to achieve high-quality results.
- For example, the number of hidden layers and the number of neurons in each layer of a deep-learning network are two key parameters, which influence the performance big time.
- Manual parameter setting and grid search approaches somewhat ease the users? tasks in setting these important parameters. Nonetheless, these two techniques can be very time-consuming.

Parameters Optimization of Deep Learning Models using Particle Swarm Optimization

- Particle swarm optimization (PSO) technique holds great potential to optimize parameter settings and thus saves valuable computational resources during the tuning process of deep learning models.
- As an example of the use of a dataset collected from a Wi-Fi campus network to train deep learning models to predict the number of occupants and their locations.

Parameters Optimization of Deep Learning Models using Particle Swarm Optimization

A PSO metaheuristic can be used to perform parameter selection a deep learning model, here, f PSO, the i^{th} particle's velocity is calculated according to the following:

- Velocity of number of layers

$$\mathrm{V}_{\mathrm{L},i}^{t+1} = \mathrm{w} \cdot \mathrm{V}_{\mathrm{L},i}^{t} + \mathrm{c}_1 \cdot \mathit{rand}. \left(L_i^{\text{best}} - \mathrm{V}_{\mathrm{L},i}^{t} \right) + \mathrm{c}_2 \cdot \mathit{rand}. \left(G^{\text{Lbest}} - \mathrm{v}_{\mathrm{L},i}^{t} \right) \qquad (2)$$

Where V_L is the velocity of the number of hidden layers, L_i^{best} is the particle's best local value of the number of hidden layers, and G^{Lbest} is the best global value of the number of hidden layers.

- Velocity of number of neurons

$$\mathrm{V}_{\mathrm{N},i}^{t+1} = \mathrm{w} \cdot \mathrm{V}_{N,i}^{t} + \mathrm{c}_1 \cdot \mathit{rand}. \left(N_i^{\text{best}} - \mathrm{V}_{\mathrm{N},i}^{t} \right) + \mathrm{c}_2 . \mathit{rand}. \left(G^{\text{Nbest}} - \mathrm{V}_{\mathrm{N},i}^{t} \right) \qquad (3)$$

Where V_N is the velocity of the number of neurons in each hidden layer,
N_i^{best} is the particle's best local value of the number of neurons in each hidden layer,
and G^{Nbest} is the best global value of the number of neurons in each hidden layer.
- Position for number of layers

$$L_i^{t+1} = L_i^t + V_{L,i}^{t+1} \tag{4}$$

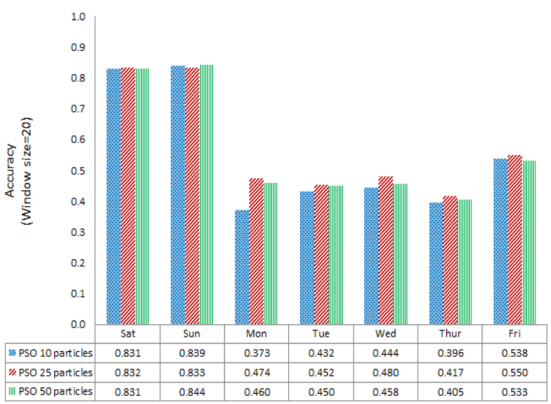
- Position for number of neurons

$$N_i^{t+1} = N_i^t + \mathrm{V}_{\mathrm{N},i}^{t+1} \tag{5}$$

EXPERIMENTAL RESULTS: A SMART BUILDING APPLICATION

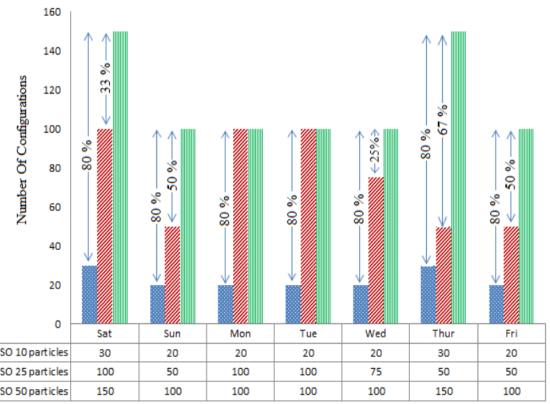
A deep learning model was built based on 6 weeks of Wi-Fi access data collected from 14 buildings of the campus of the University of Houston campus.
The goal is to build a deep learning model that predicts the number of occupants at a given location in 15,30 and 60 minutes from the current time.

Awareness of the number of occupants in a building at a given time is crucial for many smart building applications including energy efficiency and emergency response services.



	Sat	Sun	Mon	Tue	Wed	Thur	Fri
PSO 10 particles	0.831	0.839	0.373	0.432	0.444	0.396	0.538
PSO 25 particles	0.832	0.833	0.474	0.452	0.480	0.417	0.550
PSO 50 particles	0.831	0.844	0.460	0.450	0.458	0.405	0.533

(a) Accuracy vs. Model



	Sat	Sun	Mon	Tue	Wed	Thur	Fri
PSO 10 particles	30	20	20	20	20	30	20
PSO 25 particles	100	50	100	100	75	50	50
PSO 50 particles	150	100	100	100	100	150	100

(b) Number of different configurations vs. Model

Fig. 2: Comparison between three different swarm sizes (10, 25 and 50).

Applying SA meta-heuristic algorithm for neural network training

Simulated Annealing (SA) is used to improve the performance of Convolution Neural Network (CNN), as an alternative approach for optimal DL. The purpose of using SA is to train CNN is to meet certain accuracy requirement and to make the approximation error accuracy and network complexity indicators minimum.

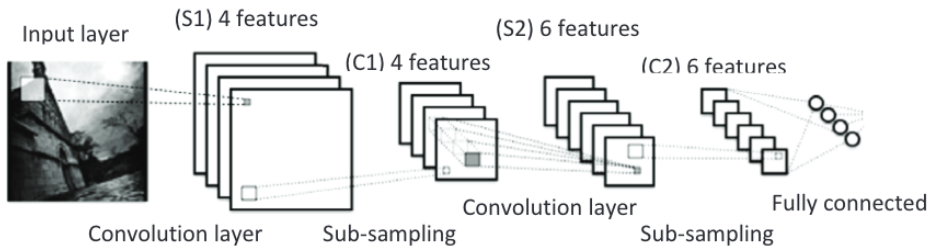
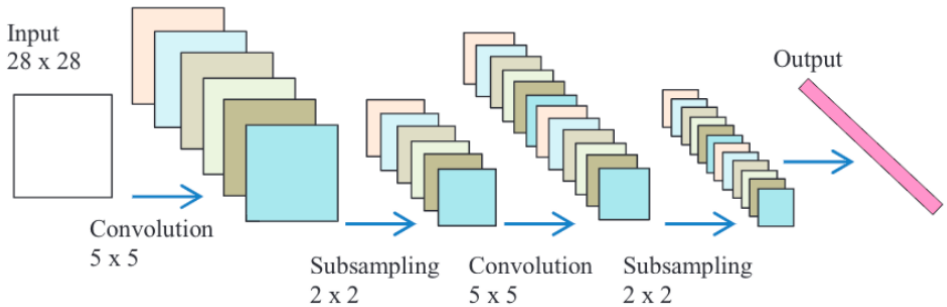


Fig. 2 The architecture of convolution neural network

Applying SA meta-heuristic algorithm on neural network training



The training standard error is used as a fitness function of vector solution.

$$f = \frac{1}{2}\sqrt{\frac{\sum_{i=1}^{R}(o-y)^2}{R}} \tag{6}$$

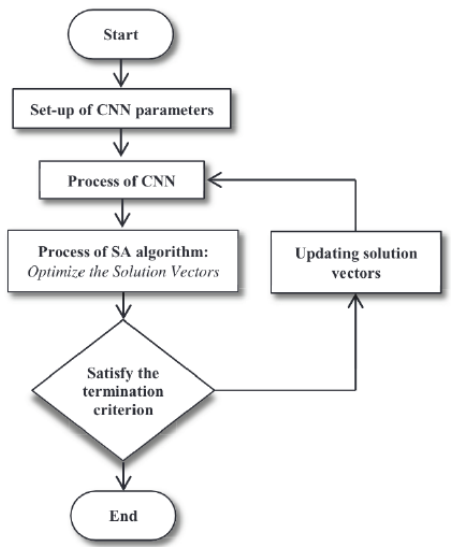
Where o is the desired output, y is the actual output, and R is the number of training samples. The search is terminated if the iterations reach a specified maximum number, or the fitness function is less than a certain threshold.

SA META-HEURISTIC AND DL TRAINING

The CNN algorithm calculates the weights and the bias of the system, and then the results on the last layers are used to compute the loss function.

Selecting a new solution vector from the neighbourhood of the recent solution is attained by adding a randomly generated amount x'. It is updated depending on the objective function $f(x_0 + x') < f(x0)$, or by Boltzmann distribution.

When SA algorithm has achieved the best objective function, the value of weights and the bias are updated for all layers of the system.



SA IN DL OPTIMIZATION

Table 1. Performance comparison of cnn, cnnSA10, cnnSA20 and cnnSA50 in terms of accuracy

Methods	Number of epochs									
	1	2	3	4	5	6	7	8	9	10
cnn	88.87	92.25	93.90	94.81	95.44	96.18	96.59	96.92	97.22	97.27
cnnSA10	89.18	92.38	94.20	95.19	95.81	96.50	96.77	97.04	97.27	97.41
cnnSA20	90.50	92.77	94.68	95.45	96.66	96.87	97.08	97.26	97.30	97.61
cnnSA50	91.10	94.16	95.49	96.20	96.91	96.99	97.33	97.42	97.40	97.71

Demonstration of the performance of implementation CNN and CNN by SA on MNIST dataset are given in Fig. 6. In general, the simulation results show that percentage accuracy or error of CNN by SA is better than the original CNN. The CNN by SA with neighborhood size 50 (cnnSA50) is better than cnnSA10 as well as cnnSA20.

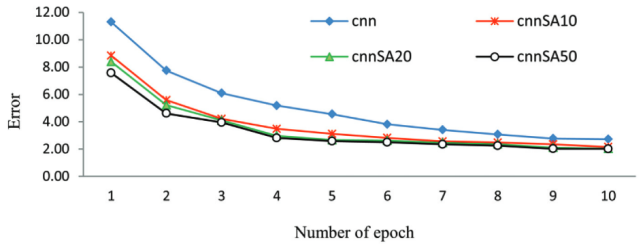


Fig. 6 Simulation results of cnn, cnnSA10, cnnSA20 and cnnSA50

In case of computation time for all methods are shown in Fig. 7. Illustration from this chart shows that the original CNN (cnn) is better that CNN by SA and the time will be increased by increasing the size of the neighborhood on SA.

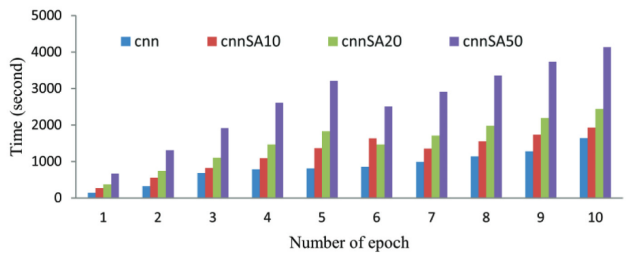
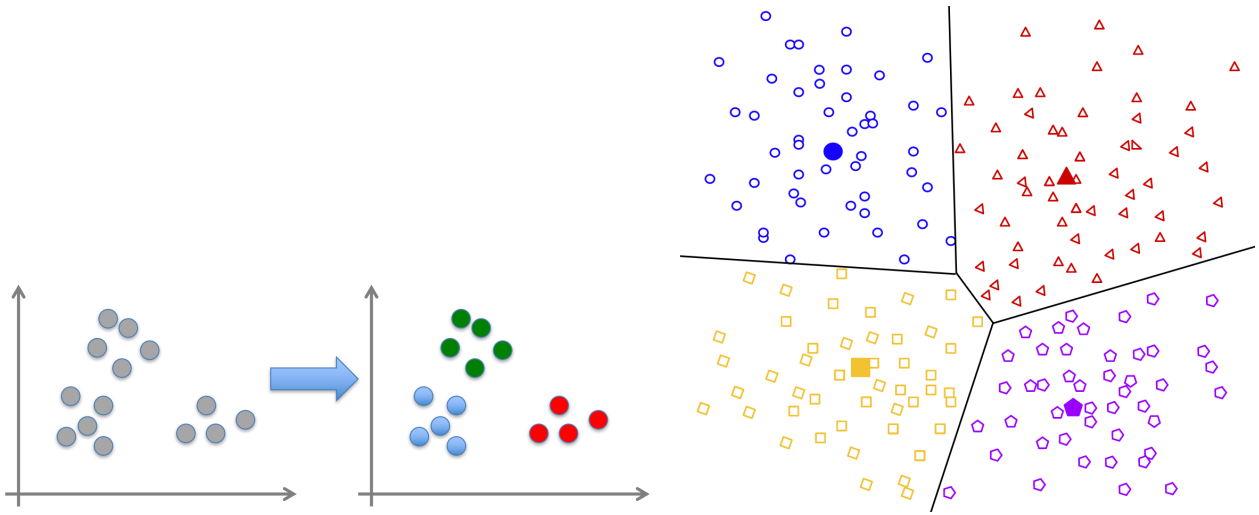


Fig. 7 Computation time for cnn, cnnSA10, cnnSA20 and cnnSA50

Unspervised Learning and ACO



Data Clustering is an important machine learning concept. It is concerned with partitioning a data set into groups based on some measure of similarity computed in some feature space. It is known to be an NP-Complete problem.

- n is the set of objects $\{X_1, X_2, \ldots, X_n\}$
- consists of clustering data space S, and
- partitions K clusters $\{C_1, C_2, \ldots, C_k\}$.

Then the constraints in partition clustering can be described as follows:

Clustering and ACO

$$\min(f)$$

where, f is a cluster validity function. There are many cluster validity functions. However, the most popular used in evolutionary algorithm for solving clustering problem is:

$$f = \frac{\sum_{j=1}^{k} \left[\sum_{\forall x \in C_{ij}} d\left(x_i, \bar{x}_j\right) / \left|C_{ij}\right| \right]}{k} \tag{7}$$

Where, $d\left(x_i, \bar{x}_j\right)$ is distance metrics. In general, it is Euclidean distance:

$$d\left(X_i, \bar{X}_j\right) = \left\| X_i - \bar{X}_j \right| \tag{8}$$

ACO based clustering employs distributed agents, each tasked with discovering a possible partition of the objects in a given data set.

Clustering and ACO

• Information associated with an ant about clustering of objects is accumulated in the pheromone trail matrix which is used by the other ants to construct possible clustering solutions and iteratively improve them.

• When the given maximum number of iterations is met, the best solution found with respect to a given metric represents an optimal or near-optimal partitioning of objects into clusters.

• Assume R ants to form a colony.
Each ant starts with an empty solution string S of length N where each element of the string corresponds to one of the objects.

$$artificial\ ant\ S_i = \overbrace{\boxed{C_1}\boxed{C_2}\boxed{\cdots}\boxed{C_k}}^{N}$$
$$Solution\ \ String$$

Clustering and ACO

• Then the artificial ant uses the pheromone trail matrix (Table I) to allocate each element of String S to a cluster id as follow:
• In the first, the elements of the pheromone trail matrix are initialized to the random values. And then, at each iteration, the pheromone matrix is updated depending on the quality of solutions produced.

N	K			
	1	2	...	k
1	δ_{11}	δ_{12}		δ_{1k}
2	δ_{21}	δ_{22}		δ_{2k}
⋮			...	⋮
n	δ_{n1}	δ_{n2}		δ_{nk}

• The first step of ant constructing a solution is generating random numbers from a uniform distribution in the range between 0 and 1. The numbers generated are equal to the length of the solution string.

Clustering and ACO

• To generate a solution S, the agent selects cluster number for each element of string S by one of the following ways:
• Set a probability threshold, q_0, and $0 < q_0 < 1$. If the elements' corresponding the random numbers are lower than q_0, the elements cluster id chosen with highest pheromone.
• On the contrary, when the corresponding random numbers are higher than the threshold, q_0, one of the K clusters using a stochastic distribution with a probability $(1 - q_0)$, denoted as, p_{ij}. It choose any one of cluster id with a normalized pheromone probability given by

$$p_{ij} = \frac{\delta_{ij}}{\sum_{k=1}^{K} \delta_{ik}}, \quad j = 1, \ldots, K \tag{9}$$

where p_{ij} is the normalized pheromone probability for element i belongs to cluster j.

Clustering and ACO

• Local Search: With a view that local search can help to find good results, the ACO clustering algorithm employs a local search procedures to improve solutions discovered by the artificial ants.

• The algorithm calculates the value of cluster validity function for each ant by using. Then, local search procedure is implemented on top L solutions with highest cluster validity values.

Local Search

Set local search probability threshold p_{ls} in $[0, 1]$. Small changes about the cluster id of elements in $S_k, k = 1, \ldots, L$ has made to get its neighbor. (i) $k = 1$.
(ii) Let S_t be a temporary solution and assign $S_t(i) = S_k(i), i = 1, \ldots, N$
(iii) For each element i of S_t, draw a random number r in $(0, 1)$. If $r \leq p_{ls}$, an integer j in the range $(1, K)$, such that $S_k(i) \neq j$ is randomly selected and let $S_t(i) = j$
(iv) Calculate cluster validity value f_t associated with solution string S_t. If f_t is less than f_k, then $S_k = S_t$ and $f_k = f_t$
(v) $k = k + 1$; if $k \leq L$ go to step (ii), else stop.

Pheromone update

After performing the local search operation, the pheromone matrix is updated. Such a pheromone updating process reflects the usefulness of dynamic information provided by the artificial ants. Formally, pheromone trails are updated by the following rule:

$$\delta_{ij}(t+1) = (1-\rho)\delta_{ij} + \sum_{l=1}^{L} \Delta f_{ij}^{l} \tag{10}$$

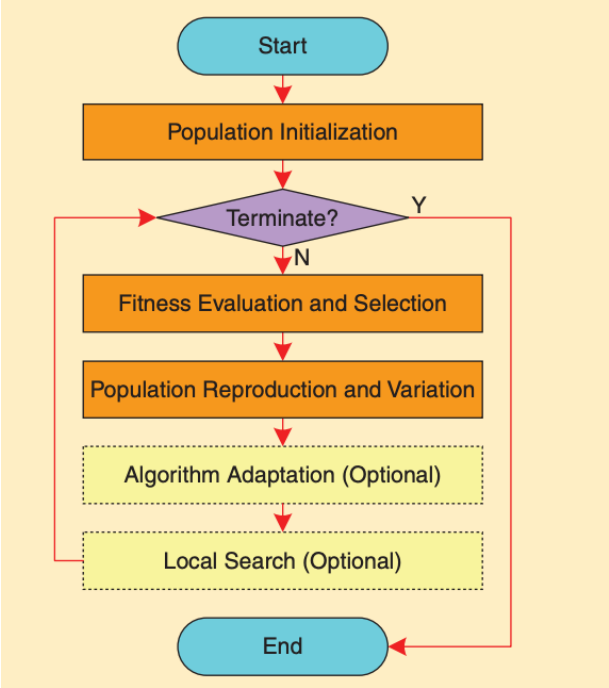
where ρ denotes the persistence of trail that lies between $[0, 1]$ and $(1-\rho)$ the evaporation.

Machine Learning and Evolutionary Strategies

- The developments and applications of ES algorithms have been one of the fastest growing fields in computing science. Moreover, research into enhancing the EC algorithms via machine learning (ML) techniques is witnessed to have played an important role in the literature
- Many attempts have been made to apply variants ES algorithms as design tools of efficient ML techniques.
- In contrast, there have been attempts to employ ML as a tool to help ES achieve smarter performance.

Machine Learning for Smarter Evolutionary Strategies

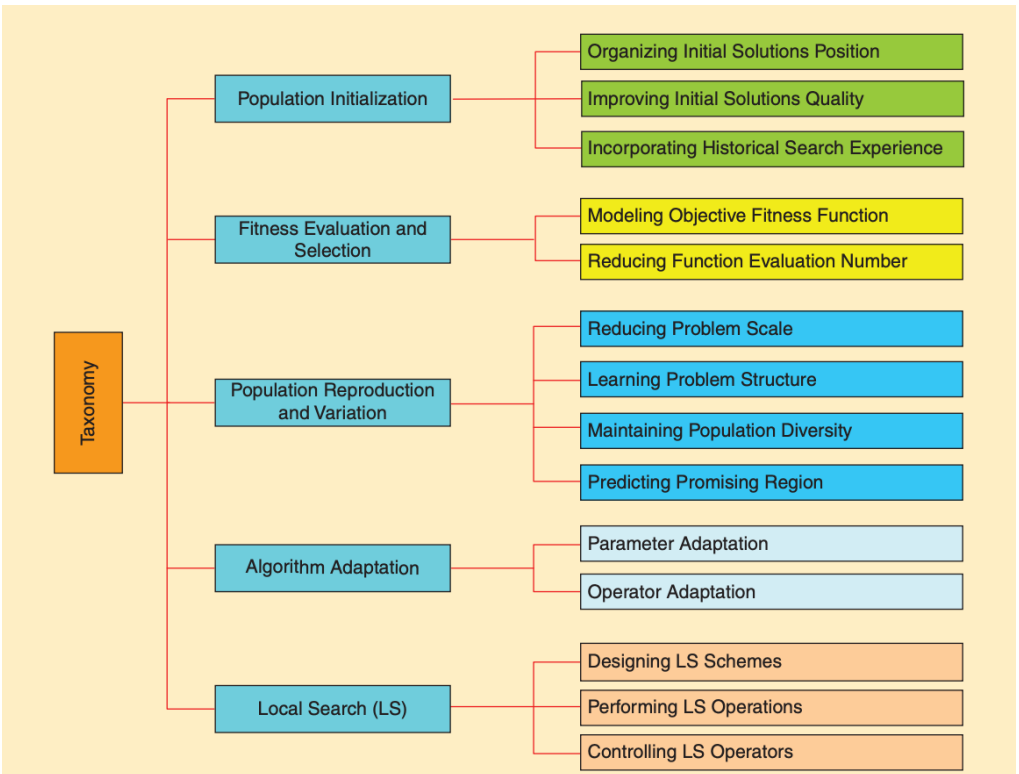
- ES have earned a reputable track record in dealing with large scale optimization or design problems.
- As depicted in the figure, their approach to the solution process involves a sequence of problems, as experienced in Assignment 3.
- The first step is the ?initialization? step. Then the algorithm enters evolutionary iterations with two operational steps, namely,
- "fitness evaluation and selection" and
- "population reproduction and variation". The new population is then evaluated again and the iteration continues until a termination criterion is satisfied.
- ES algorithms sometimes additionally perform adaptation or local search (LS) procedure. ES algorithms with LS are known as Memetic Algorithms (MA).



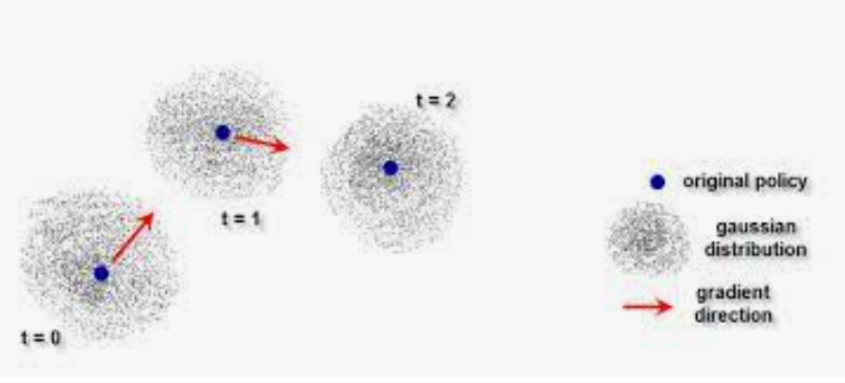
Figure

ML and ES

- The main idea is that the ES algorithm has stored ample data about the search space, problem features, and population information during the iterative search process,
- thus the ML technique is helpful in analyzing these data for enhancing the search performance. In this way, useful information can be extracted to understand the search behavior and to assist with future searches for the global optimum.
- ES algorithms that incorporated ML techniques have demonstrated advantageous wrt convergence speed and solution quality.
- Improvement areas can be identified in the figure: Initiation, Fitness function, Problem context learning, parameter adaptation, etc



ML AND ES



• Organizing Initial Solutions Position using OED (Orthogonal Experiment Design):

. Improving Initial Solutions Quality: As opposed to OED technique which improves initialization by organizing the positions of initial solutions, ML techniques such as ANN generate initial solutions of higher quality helping ES converge in fewer generations.

. Incorporating Historical Search Experience: ML techniques can also help with population initialization by acquiring useful information from historical search experiences

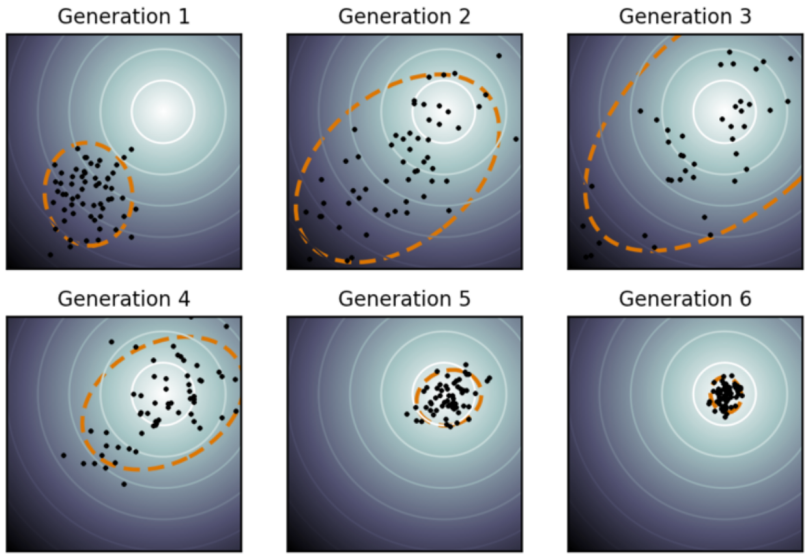
ML and ES

- Modeling Objective Function: In many practical problems, the objective function cannot be expressed in an analytical form of decision variables or the analytical formula can be too costly to evaluate. In this case, an approximate model of the real objective function is needed for evaluating the population. Among various modeling methods, the ANN technique (for single objective function, and for multi-objective function)

• ML for Population Reproduction and Variation: ML techniques have drawn significant interests in reducing the problem scale to generate new individuals effectively and efficiently. The main idea is to cluster the cities into different groups according to their position distribution information. EC algorithms find the optimal path in each group first before they solve the whole problem by regarding each group as a city.

• D. ML for Algorithm Adaptation: Parameter Adaptation. Among various adaptive approaches used in ES algorithms the most popularly adopted approach is the use of statistical methods to analyze search process data for a better understanding of the search process and hence adaptively control the algorithm?s parameters.

Machine Learning and ES



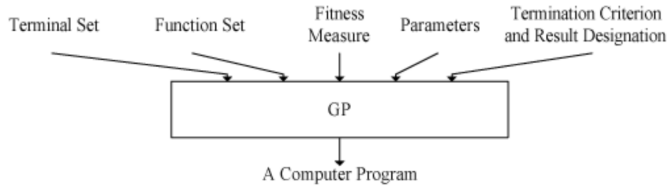
- Predicting Promising Region
- Operator Adaptation: The adaptive control of operators refers to the selection of one or multiple operators from a candidate set of operators adaptively, according to the information of the optimization process. It works in a similar way to parameter adaptation except that the adapted object is the algorithm?s operators instead of parameters.
- ML for Local Search: The LS aims to refine a solution within the solution?s neighborhood. Several researchers have used ML to design suitable LS operators.

Machine Learning and Genetic Programming

• Genetic programming is an evolutionary algorithm that automatically evolves programs, functions, or any other type of symbolic expression, that carries out some type of calculation, to solve a specific task or problem. Generally, these programs are represented as variable length structures such as a syntax tree.

• The most common application of GP, perhaps, is the symbolic regression which attempts to find a mathematical expression that best fits a given set of training data (supervised learning).

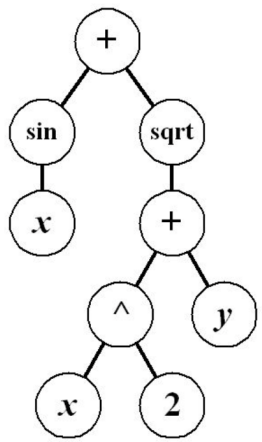
•As we established, ML is about discovering associations, correlations, mappings between things (objects, phenomena, etc). On the other hand, we studied GP as the problem of evolving programs that can demonstrate ability to produce outputs in response to inputs, such that for any particular input the output satisfies certain performance criteria. The program evolves in a supervised manner based on some dataset we believe that captures this association with a reasonable degree of consistency. Thus, GP is also about finding associations or mappings embedded in datasets.



Population of programs represented by trees

Example:

$$\sin(x) + \sqrt{x^2 + y}$$



Machine Learning and Genetic Programming

• Earlier, we studied ML as an optimization problem in the sense of finding the system parameters that minimize certain design objective function. Often we assume a model (like a NN) and try to optimize its parameters (like the weights of the NN).

• With GP, we are not limited to treating the system parameters as unknowns, but also we can treat the functions that tie these parameters together as unknowns, and hence increasing the degrees of freedom of the design process. Thus, it is a search in a space defined by terminals (constants and variables) and functions (that operate on the terminals). These functions could be arithmetic or logical or a mix of both.

GP-based Image Classification

•Terminals: For object classification problems, terminals generally correspond to image features.

• Some conventional approaches to image recognition usually use high level, domain specific features of images as inputs to a learning/classification system, which generally involves a time consuming feature selection and a hand-crafting of feature extraction programs.

• Alternatively, one can use pixel level information, domain independent statistical features (referred to as pixel statistics) as terminals and we expect the GP evolutionary process can automatically select features that are relevant to a particular domain to construct good genetic programs.

GP-based Image Classification

• Functions: In the function set, the four standard arithmetic and a conditional operation was used to form the function set:
Function Set $= \{+, -, *, /, \text{if}\}$

• The $+, -$, and $*$ operators have their usual meanings addition, subtraction and multiplication, while $/$ represents "protected" division which is the usual division operator except that a divide by zero gives a result of zero.
Each of these functions takes two arguments. The *if* function takes three arguments. The first argument, which can be any expression, constituting the condition.

• If the first argument is negative, the *if* function returns its second argument; otherwise, it returns its third argument.

• The *if* function allows a program to contain a different expression in different regions of the feature space, and allows discontinuous

GP-based Image Classification

- Fitness Function: As an example, classification accuracy on the training set of object cutout images can be used as the fitness function.
- The classification accuracy of a genetic program classifier refers to the number of object cutout images that are correctly classified by the genetic program classifier as a proportion of the total number of object images in the training set.
- According to this design, the best fitness is 100%, meaning that all object images have been correctly recognized.
- To calculate the classification accuracy of a genetic program, one needs to determine how to translate the program output into a class label. This is described in section.

3.1 Static Range Selection

Introduced in [5, 6], the static range selection (SRS) method has been used in many approaches to classification problems with three or more classes. In this method, two or more pre-defined thresholds/boundaries are applied to the numeric output value of the genetic program and linearly translated the ranges/regions between these boundaries as different classes. This method is simple because these regions are set by the fixed boundaries at the beginning of evolution and remain constant during evolution.

If there are N classes in a classification task, these classes are sequentially assigned to N regions along the numeric output value space from some negative numbers to positive numbers by N−1 thresholds/boundaries. Class 1 is allocated to the region with all numbers less than the first boundary, class 2 is allocated to all numbers between the first and the second boundaries, and class N to the region with all numbers greater than the last boundary N−1.

Step 1 Initialise the class boundaries as some random values as in the SRS method.

Step 2 Evaluate each genetic program in the population to obtain the program output result value for each training example and the fitness value of the program based on the fitness function.

Step 3 For each class c, calculate the centre of the class according to equation 2:

$$\text{Center}_c = \frac{\sum_{p=1}^{M}\sum_{\mu_c=1}^{L}(W_p \times \text{Result}_{p\mu_c})}{\sum_{p=1}^{M}\sum_{\mu_c=1}^{L} W_p} \quad (2)$$

where M is the number of programs in the population and p is the index, L is the number of total number of training examples for class c and μ_c is the index, $Result_{p\mu_c}$ is the the output value of the pth program on training example μ_c for class c, and W_p is a weight factor which reflect the relative importance or contribution of the program p over all the programs in the population and is calculated by equation 3:

$$W_p = fitness_p + 50\% \quad (3)$$

where $fitness_p$ is the fitness (classification accuracy) of program p on all the examples in the training set.

Step 4 Calculate the boundary between every two classes by taking the middle point of the two adjacent class centres.

Step 5 Classify the training examples based on the class boundaries and calculate the new fitness (classification accuracy) of each genetic program.

GP-based Image Classification

• Parameters and Termination Criteria: The parameter values used in this approach are shown in the table.

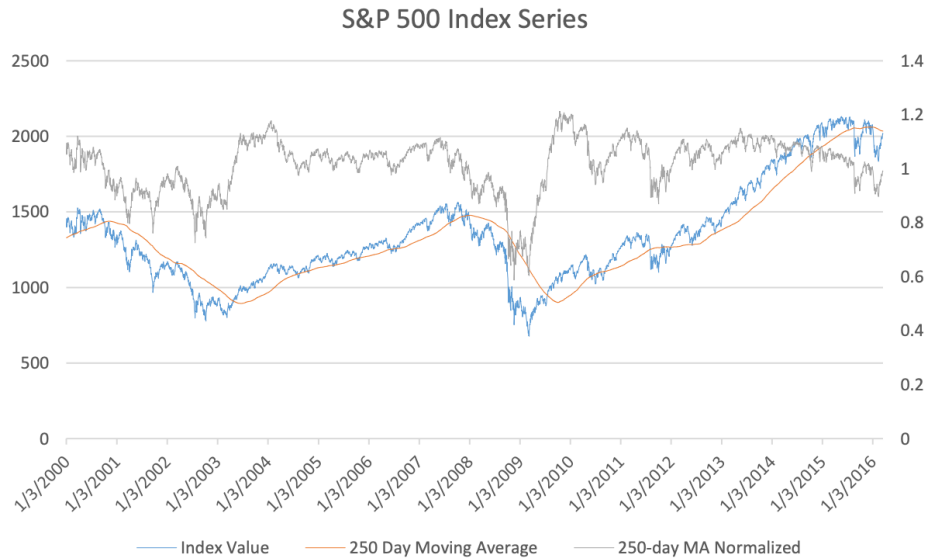
Parameter Kinds	Parameter Names	Shape1	shape2	coin1	coin2	coin3
Search Parameters	population-size	300	300	300	500	500
	initial-max-depth	3	3	3	3	3
	max-depth	5	5	5	6	6
	max-generations	50	50	50	50	50
	object-size	16×16	16×16	70×70	70×70	70×70
Genetic Parameters	reproduction-rate	20%	20%	20%	20%	20%
	cross-rate	50%	50%	50%	50%	50%
	mutation-rate	30%	30%	30%	30%	30%
	cross-term	15%	15%	15%	15%	15%
	cross-func	85%	85%	85%	85%	85%

• The learning/evolutionary process is terminated when one of the following conditions is met:

  • The classification problem has been solved on the training set, that is, all objects of interest in the training set have been correctly classified without any missing objects or false alarms for any class.

  • The number of generations reaches the predefined number, max-generations.

GP-based Market Prediction



- S&P 500 Long-Flat (Invest-don?t invest)
- Ignore transaction costs
- Ignore out of market returns
- Predictors
- S&P 500 Price
- S&P 500 Volume
- 250-day MA normalized

GP-based Market Prediction

Primitive set

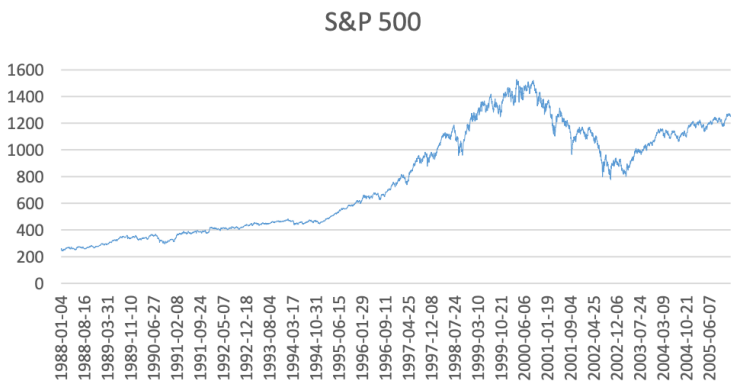
- Functions
  - Add
  - Subtract
  - Multiply
  - Divide
  - Gt
  - Lt
  - And
  - Or
  - Not
  - offsetValue
  - ifElseBoolean
  - movingAverage
  - periodMaximum
  - periodMinimum
  - AbsoluteDifference

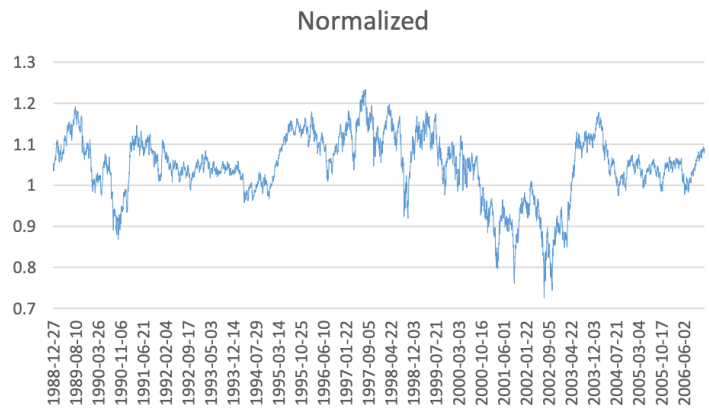
- Terminals
  - randomInteger(low high)
  - randomDouble(low high)
  - True
  - False
  - offsetValue(0)

- Hundreds of other technical indicators (http://www.investopedia.com/active-trading/technical-indicators/)

Experiment- Market Prediction

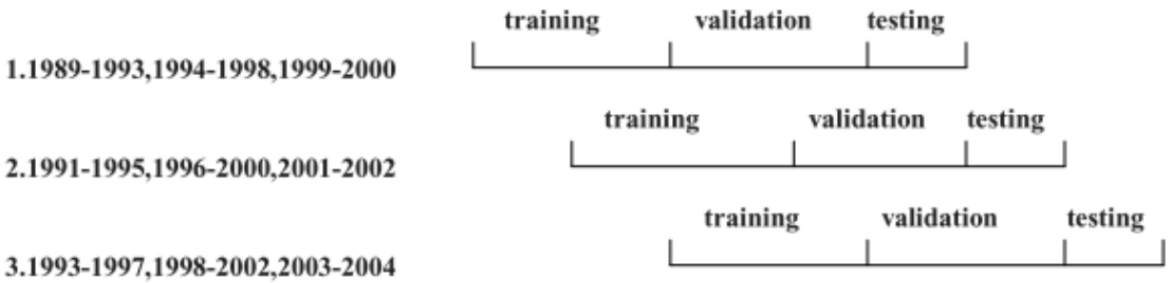
- Investment decisions in S&P 500 Index
- Modeled after (Chen et al., 2008), 1988-2004
- Long-Flat decisions
- Normalized by 250-day moving average
- Fitness = investment gain





Experiment- Market Prediction

- Training-validation-prediction approach (T-V-P)



(Chen et al., 2008, p. 110)

- Training-prediction approach (T-P)
  - Training - 1989-1998
  - Prediction- 1999-2004

Results T-V-P w/Trans Cost

Method	Mean	Std. Dev.	Min	Max	95% CI	# beating benchmark
1999-2000						
Buy & Hold	0.0751					
GP	0.0434	0.0664	-0.1917	0.1197	[0.0250 ... 0.0618]	5/50
ADF	0.0309	0.0798	-0.3054	0.0845	[0.0088 ... 0.0530]	3/50
ADT	0.0510	0.0519	-0.1974	0.1042	[0.0366 ... 0.0654]	5/50
2001-2002						
Buy & Hold	-0.3144					
GP	-0.3693	0.1306	-0.8087	-0.2885	[-0.4055 ... -0.3331]	1/50
ADF	-0.3347	0.0887	-0.7290	-0.1777	[-0.3593 ... -0.3102]	2/50
ADT	-0.3697	0.1390	-0.7450	-0.0134	[-0.4082 ... -0.3312]	1/50
2003-2004						
Buy & Hold	0.3332					
GP	0.2945	0.0497	0.1432	0.3291	[0.2807 ... 0.3083]	0/50
ADF	0.3139	0.0390	0.1170	0.3539	[0.3031 ... 0.3247]	1/50
ADT	0.3247	0.0150	0.2349	0.3522	[0.3205 ... 0.3289]	2/50

(Moskowitz, 2016, p. 119)

Particle Swarm Optimization

Discrete PSO

Discrete PSO

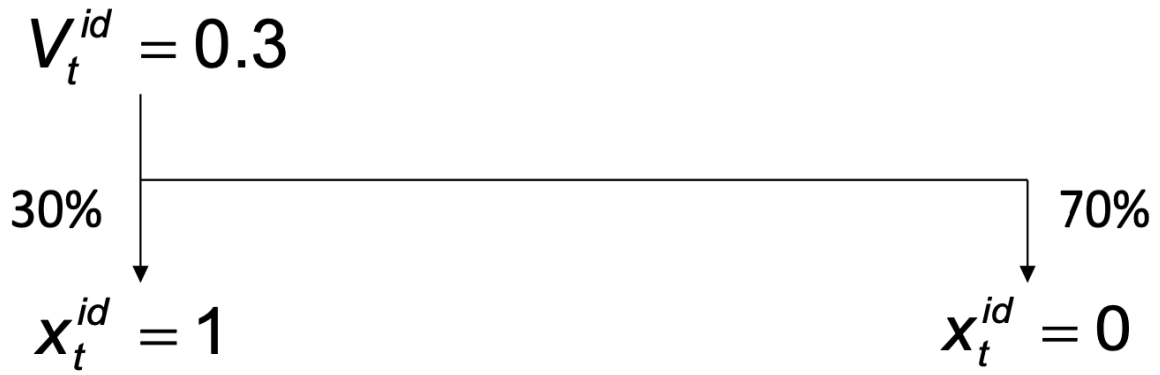
- PSO was originally developed for continuous-valued spaces
- Many problems are defined for discrete valued spaces
- Examples: Feature selection, TSP, Assignment problems, Scheduling,..
- Changes to PSO can be as simple as ***discretization*** of the position vectors or complex as ***redefining of the arithmetic operations*** (addition, multiplication)

Particle Swarm Optimization

Binary PSO

Binary PSO

- A binary PSO version was proposed by Kennedy and Eberhart (below),
- Each particle represents a position in the binary space,
- Each element can take the value of 0 or 1.
- Velocities are defined as ***probabilities*** that one element will be in one state or the other,
- If:



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.

Binary PSO:

- Introduces by kennedy and Eberhart.
- Each individual (particle) has to take a binary decision.

$$P(x_{id}=1)=f(x_{id}^{t-1},v_{id}^{t-1},p_{id},p_{gd}),\quad d\in\{1,\ldots,n\}$$

Previous state

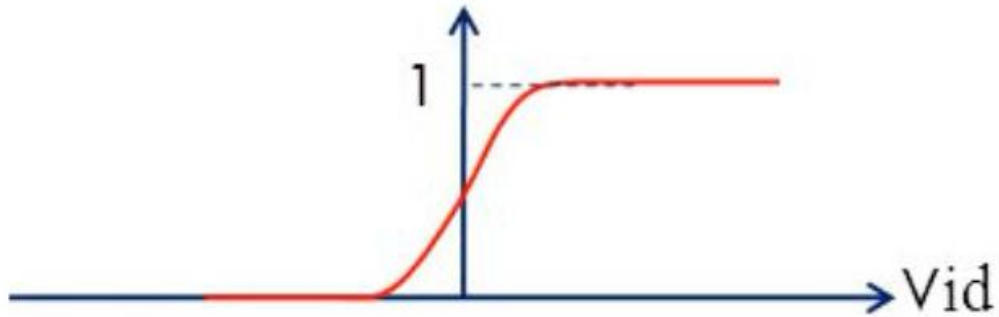
predisposition

- Predisposition is derived based on individual and group performance:

$$v_{id}^{t}=v_{id}^{t-1}+\varphi_1 r_1(p_{id}-x_{id}^{t-1})+\varphi_2 r_2(p_{gd}-x_{id}^{t-1})$$

- v_{id} determines a threshold in the probability function and therefore should be bounded in the range of [0.0, 1.0].

$$sig(v_{id}) = \frac{1}{1+\exp(-v_{id})}$$



- state of the dth position in the string at time t:

$$x_{id}^{t} = \begin{cases} 1 & if \quad \rho_{id} < sig(v_{id}^{t}) \\ 0 & if \quad \rho_{id} \geq sig(v_{id}^{t}) \end{cases}$$

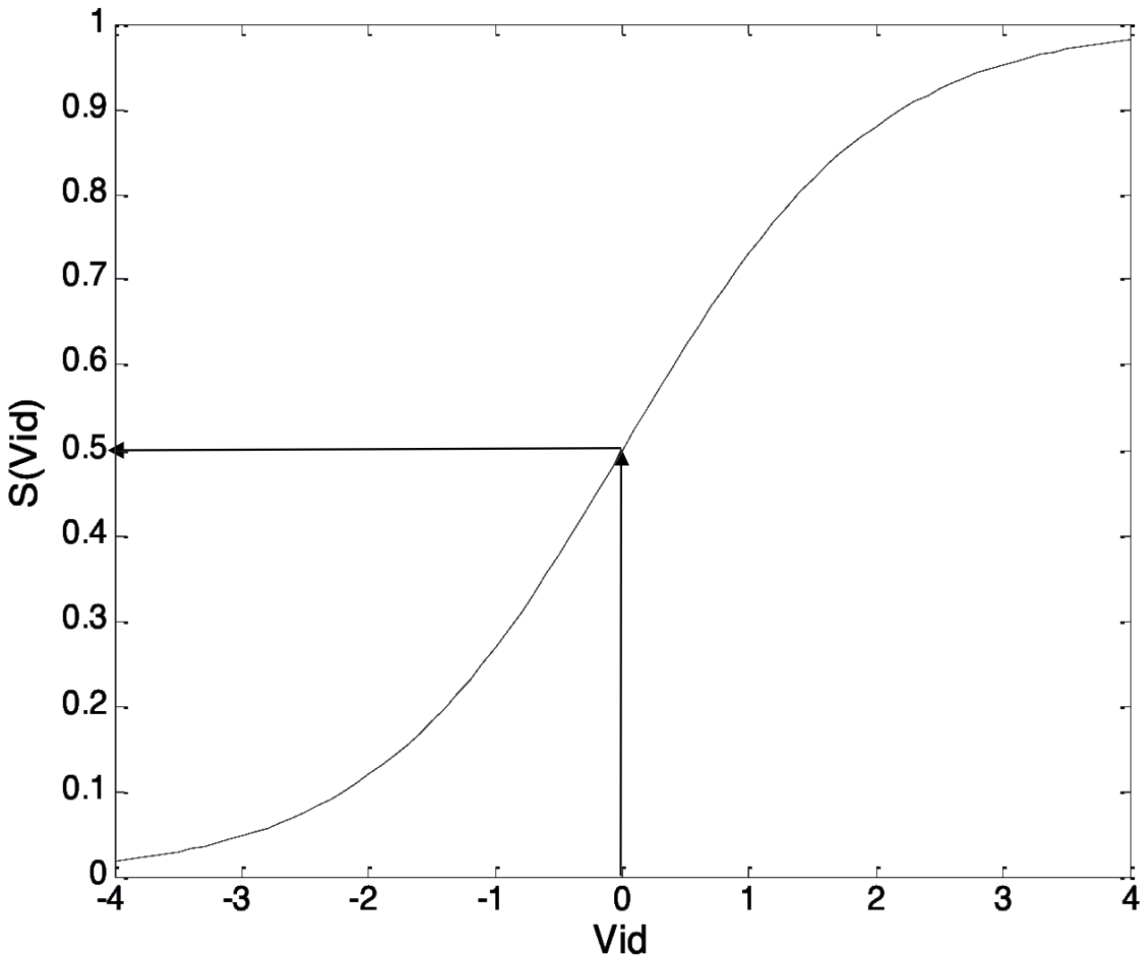
- Where ρ_{id} is a random number with a uniform distribution.

Binary PSO

- This means that:

$$prob\left(x_{t+1}^{id}=1\right)=\begin{cases}0.5 & ,v_{t+1}^{id}=0\\ <0.5 & ,\ v_{t+1}^{id}<0\\ >0.5 & ,\ v_{t+1}^{id}>0\end{cases}$$

- Note that the velocity components could remain as real-valued numbers using the original equation, but fed to the sigmoid function before updating the position vector.



Binary PSO

- A comparison was made between the binary PSO and three version of Gas:
  - Crossover only GA (GA_c),
  - Mutation only GA (GA_m),
  - A regular GA.

- The objective was to optimize different functions that were generated using a ***random function generator***.

- This allows the creation of random binary problems with some specified characteristics:
  - Number of local optima, bit strings
  - Dimension, no of components (bits)
  - Fitness function

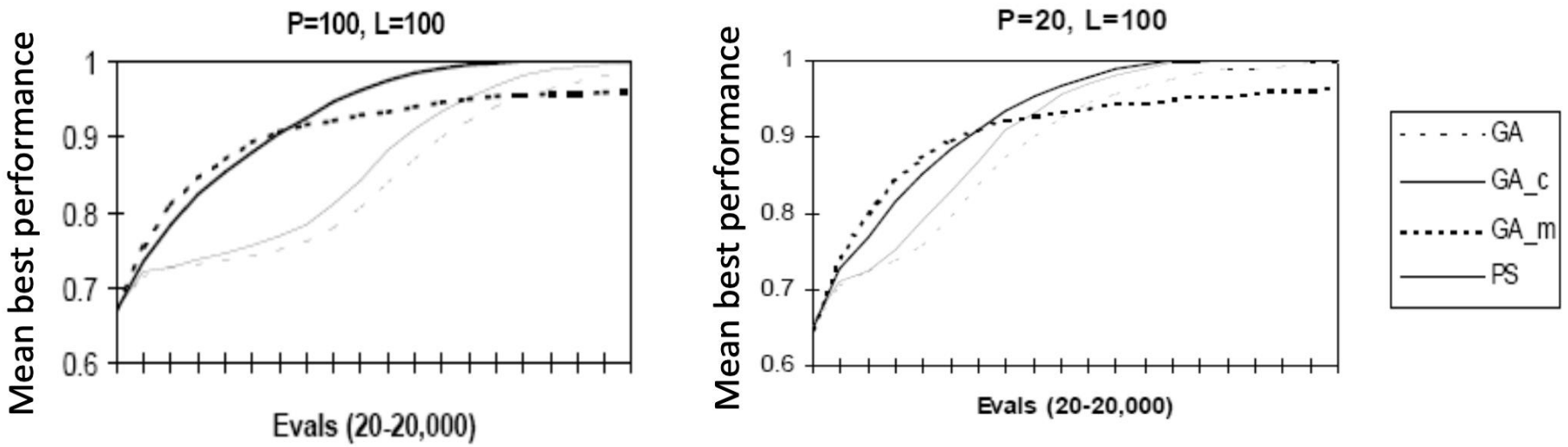
J.Kennedy and R. C. Eberhart. "A Discrete Binary Version of The Particle Swarm Algorithm". Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, pp. 4101-4109, 1997.

Binary PSO

- In that study, the binary PSO was the only algorithm that found the global optimum on every single trial, regardless of problem features.

$$f(c) = \frac{1}{L} \max_{i=1}^{P} \{L - Hamming(c, Peak_i)\}$$

- The experiments show that the binary PSO progressed faster than the other algorithms.



P is the number of peaks and L is the binary vector length

Particle Swarm Optimization

Permutation PSO

Permutation PSO

- Several attempts have been made to apply PSO to permutation problems,
- The difficulty of extending PSO to such problems is that the notions of ***velocity and direction have no natural extensions*** for these problems.
- Clerc applied PSO to solve the TSP,
- The position of a particle was the solution to a problem (permutation of cities),
- The velocity of a particle was defined as the set of swaps to be performed on a particle.

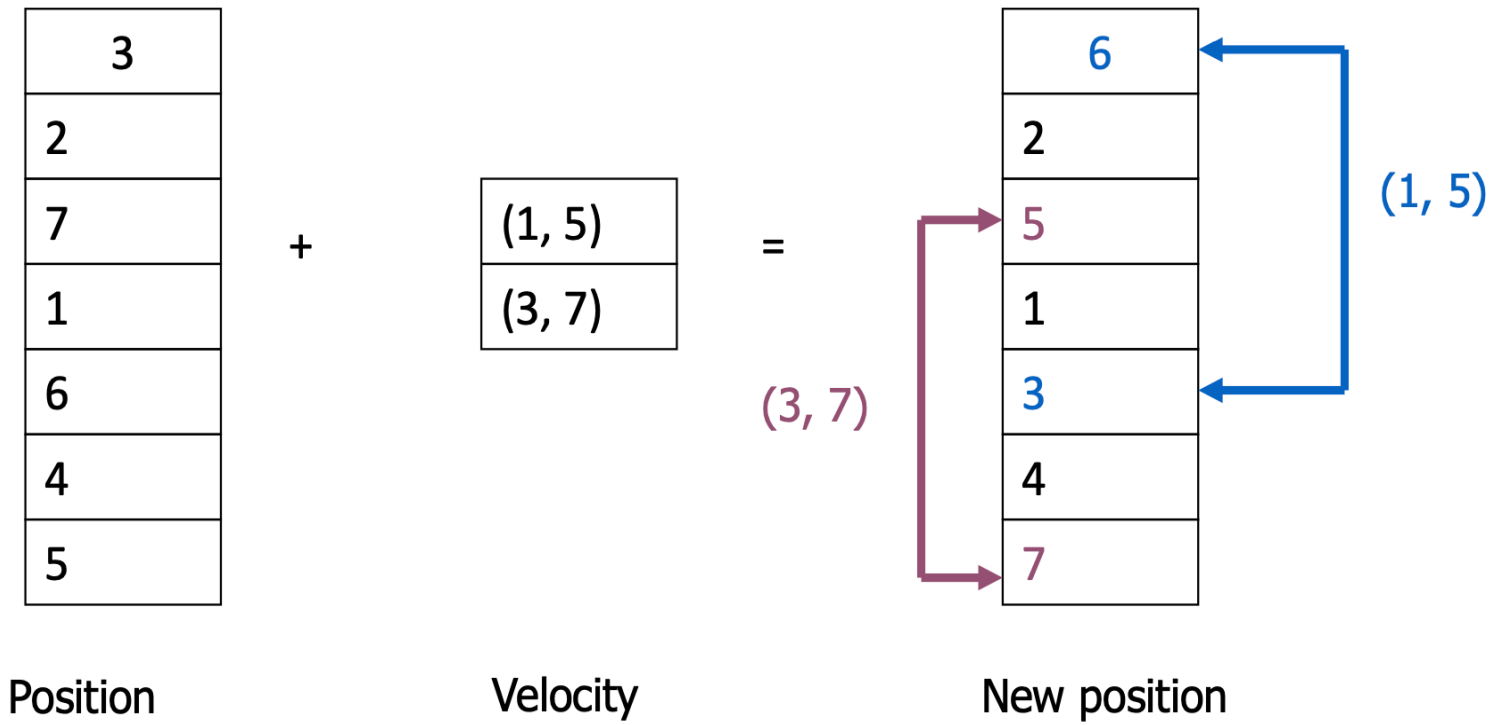
M. Clerc, "Discrete Particle Swarm Optimization," in New Optimization Techniques in Engineering. Springer-Verlag, 2004.

Permutation PSO

- Three operations were re-defined for the new search space:
  - Adding a velocity to a position,
  - Subtracting two positions,
  - Multiplying a velocity by a constant.

Permutation PSO

- **Adding a velocity to a position:** the operation is performed by applying the sequence of swaps defined by the velocity to the position vector.

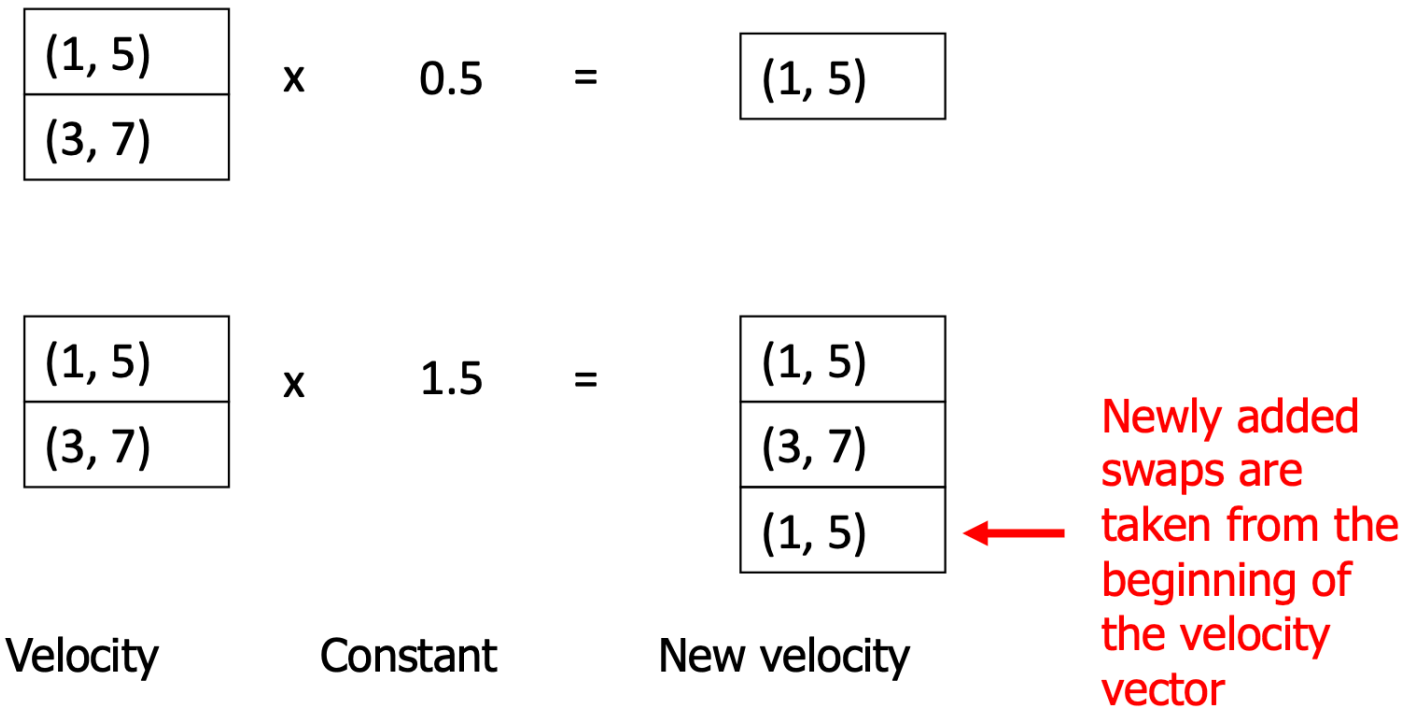


Permutation PSO

- **Subtracting two positions:**
  - Subtracting two positions should produce a velocity,
  - This operation produces the sequence of swaps that could transform one position to the other.

Permutation PSO

- **Multiplying a velocity by a constant:** This operation is performed by changing the length of the velocity vector (number of swaps) according to the constant c.
  - If $c = 0$, the length is set to zero,
  - If $c < 1$, the velocity is truncated,
  - If $c > 1$, the velocity is augmented.



Permutation PSO

- Using 16 particles and a neighbourhood of 4 particles, it finds the optimal solution for problem with size of 17 (br17.tsp) using 7990 tour evaluations.

- Using a ***social neighbourhood*** (close particles in the swarm matrix) is less expensive than using ***physical neighbourhood*** (close in the search space).

- However, applying it to larger problems might be computationally expensive (due to the handling of multiple solutions).

Swarm size	Neighbourhood size	Neighbourhood type	Logical/arithmetic operations
16	4	Social	4.8M
16	4	Physical	6.3M

Permutation PSO

- A different PSO approach was applied to the TSP by Pang et. al.,
- A continuous PSO version was used,
- In order to evaluate the particle fitness, they performed a ***space transformation*** from a particle in the continuous domain to a permutation in the solution space.
  - The used rule known as ***Great Value Priority (GVP),***

Permutation PSO

- **Great Value Priority (GVP):**
  - First, all the elements in the position vector (along with their indices) were sorted to get a sorted list in descending order,
  - The sorted indices were taken as the permutation.
  - The idea was that if x_i had the highest value in the position vector, it means that city number i comes first in the permutation vector,
  - This transformation was carried before calculating the particles fitness.

Permutation PSO

- Example:

$$if \quad x = [0.2, 0.3, 0.1, 0.8]$$

then the sorted list would be:

$$\begin{matrix} 4 & 2 & 1 & 3 \end{matrix}$$
$$x_{sorted} = [0.8, 0.3, 0.2, 0.1]$$

- and the permutation would be:

Tour to be evaluated and its cost is the particles' x fitness → $P = [4, 2, 1, 3]$

Permutation PSO

- The work also experimented with applying local search to the permutation with a certain probability to get a better one,
- The local search was applied by selecting two cities to swap,
- If a better tour is found, in order to go back to the continuous domain, the same swap is applied to the original particle x.

Permutation PSO

- Example:

$$\textit{if} \quad P = [4, 2, 1, 3] \qquad \text{and} \quad X = [0.2,\ 0.3,\ 0.1,\ 0.8]$$

and the local search produced the better tour:

$$P = [2, 4, 1, 3]$$

then the new position would be:

$$x = [0.2, 0.8, 0.1, 0.3]$$

by swapping the same elements

Permutation PSO

- The PSO and PSO_LS both were applied to 4 TSP problems using:
  - 50 particles and 2000 iterations,
  - The position constrained in [-1, 1],
  - The velocity constrained in [-0.1, 0.1],
  - w=1, c_1=c_2=2,
  - A local search probability of 1%,
  - The results are the averages taken over 10 runs.

Permutation PSO

Instance	PSO_TS	PSO_TS_LS	Optimal Solution
burma14	33.6948	**30.8785**	30.8785
eil51	582.501	**459.273**	426
berlin52	8100.105	**7938.041**	7542
eil76	588.712	**564.523**	538