# Experiments in Developing Discrete Particle Swarm Optimisers

Martin Gate

May 14, 2019

## 1 Introduction

### 1.1 Continuous Case model

Particle Swarm Optimisers (PSO) introduced in [KE95] have been developed to optimise problems with continuous tuning parameters. The PSO consists of a finite collection (called a swarm) of particles with current state (often called *Parameters*) $x_t^i$ personal best state $p_t^i$ and velocity $V_t^i$; where $i$ is the particle index and $t$ is the iteration index. These three properties of each particle are updated at every iteration using

$$V_{t+1}^i = \mu\left(x_t^i, V_t^i, N_t^i\right) \tag{1.1}$$

$$x_{t+1}^i = \xi\left(x_t^i, V_t^i\right) \tag{1.2}$$

$$p_{t+1}^i = \begin{cases} p_t^i & \text{if } x_{t+1}^i \notin \Omega \text{ or } F(p_t^i) < F(x_{t+1}^i) \\ x_t^i & \text{otherwise} \end{cases} \tag{1.3}$$

where $F(x)$ is the cost of the state $x$; $N_t^i$ is the collection of personal best states used by the $i$th particle to update its velocity from iteration $t$ to $t+1$: $\Omega$ is a feasible subset of states usually characterised as satisfying a collection of constraints .

The PSO has several variants and the above description is a general summery where for continuous parameters the state is a $d$ dimensional vector space of real numbers $\mathbb{R}^d$. Typically the velocity is made up from random multiples of the vector diference between $x_t^i$ personal bests and members of$N_t^i$ giving the swarms directions to go to look for improved $F(x)$ values. Also $\mu$ and $\xi$ are not true functions since they depend on hidden random variables that are crucial for ensuring the ability to find a low cost near optimal personal best after sufficient iteration, but are otherwise processes that depend on the variables given as their arguments.

The current work looks at combinatorial optimisation problems (COP) using discrete tuning parameters in the form of binary strings (BS) for particle state with big integer cost functions. Such a PSO will be called a Set PSO ( abbreviated to SPSO) to emphasise that it interprets BS as sub sets to represent the parameters of SPSO.

# 2 Set PSO

## 2.1 Using Big Integers for Cost-function

The restriction to big integer is strictly not required since the cost function is used only to compare the cost between solutions so at a later stage one may extend to floating point or rational cost functions with extended resolution.However, Most COPs are manageable if the number of possible values is kept reasonably small by restricting the range of integers.Big integers are used to include some of the difficult cases although the chance of finding global best solutions to these problems is minimal. The algorithms are expected to however produce good useful approximations to these hard problems. for the moment we adopt the principle that real number cost functions can be converted into an approximate big integer by multiplying by a suitable large constant to achieve the desired resolution after rounding to the nearest integer. I would like to suggest that main area of application is *machine learning where one is interested in good approximate minimisation solutions of difficult problems*.

## 2.2 The Importance of Satisfying Constraint on Solutions

COP problems such as the Travelling Sales Person(TSP) have constraints defining $\Omega$ that must be satisfied if to be of any use so one must continuously ensure the candidate solution is in $\Omega$ before evaluating its cost. Often feasible points $\Omega$ do not form an open subset of the space of states so a raw update of $x_t^i$ will not give a feasible state most of the time. To cope with this $\xi$ computes a raw update $\breve{x}_{t+1}^i$ which it then attempts to convert into a feasible one before being returned as a update. Sometimes this may not be possible and the returned $x_t^i$ is not in $\Omega$. As indicated in the update equation this results in no change to the personal best thus ensuring that personal bests are updated to feasible states.

## 2.3 Generalised Link Between COP and SPSO

The link between COP and SPSO is forged by using binary strings to represent candidate algorithms which are obtained by decoding the string. I use the term algorithm to represent any method for finding approximate solutions to a COP problem including just providing an approximate solution with no use of an algorithm. I think decoding to an algorithm rather than just a solution gives the PSO significantly more expressive power to find approximate solutions. The structure of candidate solutions may not be the same as binary strings so several binary strings may map to the same solution. The resulting solution is then evaluated to give its cost indicating how well the algorithm fits data in form of a (possibly large) integer; this could include some cost representing the complexity of the used algorithm; lower values being preferred to higher values. This procedure gives a mapping from a binary string to a integer as a cost of the binary string. The SPSO is used to learn a low cost global solution by using a swarm of proposed solutions to iteratively search for a low cost one.

For the moment we restrict the SPSO state to be at most $N$ bits. Each binary string $x$ can thus be regarded as a subset of integers in the range $1 \cdots N$

where $j \in x$ when $x[j] = 1$. To this extent each non zero bit represents the inclusion of some element with a feature and as such encodings that favour this interpretation is expected to do better using the following algorithms.

## 2.4 Reading Across from the Continuous Case

In the following let *rand* be a random number generator that produces a number in the range $0 \leq rand < 1$ for each use of it.Also let $\neg$ be the logical negation and $\leftarrow$ the assignment of left side to the right side of the arrow.

### 2.4.1 Semantic Reinterpretation of Notation

The notation for the continuous case can be now used where $x_t^i$ is just a subset rather than a vector. The velocity is interpreted as a vector of probabilities where the $j$th component $V_t^i[j]$ is just the probability that $x_t^i[j]$ will flip during the update going from 0 to 1 or *vice versa*.

### 2.4.2 Adding up velocity contributions

The velocity is built up from several contributions, which will be described in detail below, so there is a need for a way of adding up the contributions. the velocity is an array of probabilities each component being regarded as independent so one is interested in how to add probabilities. direct adding of probabilities does not work because this can give values greater than 1; multiplying probabilities just leads to a value smaller than the probabilities being combined. A possible compromise is to combine the probabilities $p_1$ and $p_2$ as follows:

$$p_1 \dot{+} p_2 = p_1 + p_2 - p_1 p_2 \tag{2.1}$$

and for two velocities we define

$$(V_1 \dot{+} V_2)[j] = V_1[j] \dot{+} V_2[j] \qquad \forall j \tag{2.2}$$

this is the method adopted although not the only one that could have been used. For instance taking the maximum value of the probabilities is popular and could be an alternative. the chosen operation (2.1) is called *pseudo-addition* since it has most properties of addition. This can be seen by putting

$$p_i = 1 - e^{-\xi_i} \tag{2.3}$$

and then noting that

$$\begin{aligned} p_1 \dot{+} p_2 &= (1 - e^{-\xi_1}) + (1 - e^{-\xi_2}) - (1 - e^{-\xi_1})(1 - e^{-\xi_2}) && (2.4) \\ &= 1 - e^{-\xi_1} e^{-\xi_2} && (2.5) \\ &= 1 - e^{-(\xi_1 + \xi_2)} && (2.6) \end{aligned}$$

so the mapping $p_1 \mapsto \xi_i$ is a homomorphism between $\dot{+}$ and $+$. In particular the pseudo-sum is commutative and associative.

### 2.4.3 Velocity Components for Directing Towards $N_t^i$

As for continuous case the velocity, as an indicator of the direction to go to find better solutions, is roughly increased along the diference between $x_t^i$ and $p_t^k$ for each $k \in N_t^i$. For the SPSO the diference is represented by the exclusive or of the corresponding subsets giving the difference:

$$del_t^{(i,k)} = x_t^i \oplus p_t^k$$

From this a velocity increment can be produced as $Vdel_t^{(i,k)}$ where

$$r(i,k) \leftarrow \phi\, rand \tag{2.7}$$

$$r_t^{(i,k)} \leftarrow \begin{cases} r(i,k) & \text{if } r(i,k) < 1 \\ 2 - r(i,k) & \text{otherwise} \end{cases} \tag{2.8}$$

$$Vdel_t^{(i,k)}[j] = \begin{cases} r_t^{(i,k)} & \text{if } j \in del_t^{(i,k)} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

the $\phi$ is a heuristic for tuning how aggressively the velocity points towards the target particle k and here takes on a predetermined heuristic value between 0 and 2 (following tradition of continuous case). However, the velocity is limited to be in the range 0 to 1 since it is a probability of flipping, so values above 1 are reflected back. This doubles the density of $Vdel_t^{(i,k)}$ near 1 when $\phi > 1$. The treatment of how to cope with $r(i,k) > 1$ case is not well derived and alternatives such as limiting all such cases to 1 may be better. The velocity increment encourages movement to the target $p_t^k$ by only contributing to the difference between $x_t^i$ and $p_t^k$.

### 2.4.4 Shotgun Blurring of Targets

Typicaly there are a large number of elements (corresponding to dimensions in the continuous case) while the number of particles is kept to just a few say at most 30. This means the use if just the velocity contributions directed to targets will explore only a small subset of changes. In the continuous case this has been solved in [BM14] by adding a noise component to the velocity proportional to distance of particle from corresponding target, which effectively gives a shotgun effect. Further more this simple device makes the PSO converge to a local optimal solution.

For SPSO the distance is the Hamming distance between the particle and target given by

$$d_H(i,k) = |del_t^{(i,k)}| \tag{2.10}$$

where $|z|$ is just the number of elements in $z$.this gives a blurring velocity

$$b = rand(L_{factor} d_H(i,k) + L_{offset})/N \tag{2.11}$$

$$Vblur_t^{(i,k)}[j] = b \ \forall j \tag{2.12}$$

where $L_{factor}$ and $L_{offset}$ are heuristics used to tune the shotgun effect.

### 2.4.5 Inertia Term

A heuristc $\omega$ borrowed from the continuous case is an inertia term that multiplies the velocity probability by a reducing factor to slow down the progress of a particle and stop the velocity terms from over saturation.

Using the multiplying operation is a crude and simple and may not be the best; the mapping from pseudo-addition to addition given in Equation 2.3 suggests a more complex operation involving raing the probability of not flipping to some power. However, for the moment, simpler operation of multiplying which has the desired reduction of velocity is used here.

### 2.4.6 Combining the Velocity Contributions

these various contributions are combined to give

$$V_{t+1}^{i} = \omega \left( V_t^i \dot{+} \dot{\sum_{k \in N_t^i}} Vblur_t^{(i,k)} \right) \dot{+} \dot{\sum_{k \in N_t^i}} Vdel_t^{(i,k)} \tag{2.13}$$

where multiplying by $\omega$ multiply each component of the velocity by $\omega$. In the continuous case $\omega$ just multiplies $V_t^i$; in the discrete case blurring tends to saturate and mask the update that moves the solution in direction of targets, so $\omega$ is applied to this as well.

### 2.4.7 Updating to the Next Frame Using the Probabilistic Velocity

In the continuous case the raw update is given by adding the velocity component to the current state. For the SPSO case by analogy we use flipping to give a raw update from the velocity, thus put for each component:

$$(V_{t+1}^i[j]), \breve{x}_{t+1}^i[j]) \leftarrow \begin{cases} (0, \neg x_t^i[j]) & \text{if } V_{t+1}^i[j] > rand \\ (V_{t+1}^i[j], x_t^i[j]) & \text{otherwise} \end{cases} \tag{2.14}$$

At this stage the raw state $\breve{x}_{t+1}^i$ may not satisfy the constraint so to keep things general a function `ToConstraint()`[1] supplied by the cost-function interface is applied to attempt to make the raw state satisfy the constraint based on the raw state as a starting hint and the old state,$x_t^i$ to facilitate this attempt.If it fails then revert back to the old state.Thus:

$$\breve{x}_{t+1}^i \leftarrow \quad \text{ToConstraint}(x_t^i, \breve{x}_{t+1}^i) \tag{2.15}$$

$$x_{t+1}^i \leftarrow \quad \begin{cases} \breve{x}_{t+1}^i & \text{if } \breve{x}_{t+1}^i \in \Omega \\ x_t^i & \text{otherwise} \end{cases} \tag{2.16}$$

this gives an update that satisfies the constraints, $\Omega$ where the initalisation of the state $x_0^i$ is chosen to ensure it also satisfies the constraint. this is done by random selection until the application of gives something that satisfies the constraint. Note during this operation the first argument will be the empty set and should be treated as a special case by the cost function.

[1]in the code this function has a diferent interface to minimise the API but is used to acheive the same thing.

## 2.5 Dealing With Noisy Cost Function

In the targeted application of Machine Learning the cost-function $F(x)$ is not a direct mathematical function of $x$ and has a random component so it has to be evaluated several times before comparing the fitness of $x$. In the current context `ToConstraint()` can be used to achieve this by returning a copy of $x_t^i$ for $\check{x}_{t+1}^i$ but stating that constraints have not been satisfied. This has the side effect of not updating $p_{t+1}^i$ as well giving the opportunity to revaluate it. the cost-function has control of updates in this way and can delay update until it is satisfied that a comparison of costs is valid.

## 2.6 grouping Swarm Particles

It is anticipated that the main things that improve the SPSO performance is change in heuristics and the method used to choose the target sets $N_t^i$. Further more it is expected that particles in the swarm will adopt specialist behaviour by belonging to a group of particles. To support this the swarm is partitioned into groups where each group of swarm particles has the same set of targets and heuristics. Groups are each given a name to aid describing a group.To begin with there is one group called "root" which contains all particles. Heuristics are often the same between groups so when forming a new group the heuristics are linked to the "root" group, although this can be modified and in general groups share heuristics to simplify changing a heuristic. Groups are populated by moving swarm particles from other groups thus maintaining a partition of swarm. By default the heuristics are chosen to have useful values and need not be explicitly given.

## References

[BM14] Mohammad Reza Bonyadi and Zbigniew Michalewicz. A locally convergent rotationally invariant particle swarm optimization algorithm. *Swarm Intell,Springer*, pages 1–40, June 2014.

[KE95] J. Kennedy and R. Eberhart. Particle swarm optimization. *International conference on neural net- works, IEEE*, 4(3):1942–1948, July 1995.