Discrete Particle Swarm Optimization

Illustrated by the Traveling Salesman Problem

Maurice Clerc (Maurice.Clerc@WriteMe.com)

http://www.mauriceclerc.net

29 February 2000

Abstract

The classical Particle Swarm Optimization is a powerful method to find the minimum of a numerical function, on a continuous definition domain. As some binary versions have already successfully been used, it seems quite natural to try to define a more general frame for a discrete PSO. In order to better understand both the power and the limits of this approach, we examine in details how it can be used to solve the well known Traveling Salesman Problem, which is in principle very "bad" for this kind of optimization heuristic. Results show Discrete PSO is certainly not as powerful as some specific algorithms, but, on the other hand, it can easily be modified for any discrete/combinatorial problem for which we have no good specialized algorithm.

Résumé

L'Optimisation par essaim particulaire est une méthode efficace pour trouver le minimum d'une fonction numérique définie sur un domaine continu. Certaines versions binaires ayant été déjà utilisées avec succès, il semble assez naturel d'essayer de définir un cadre plus général pour une OEP discrète. Afin de mieux comprendre à la fois l'efficacité et les limites de cette approche, on examine en détail comment elle peut être appliquée au bien connu Problème du voyageur de commerce, qui est, a priori, très "mauvais" pour ce genre d'heuristique d'optimisation. Les résultats montrent que l'OEP discrète n'est sûrement pas aussi performante que certains algorithmes spécifiques, mais, d'un autre côté, elle peut facilement être adaptée à tout problème discret/combinatoire pour lequel on ne disposerait pas de bon algorithme spécialisé.

A few words about "classical" PSO

The basic principles in "classical" PSO are very simple. A set of moving particles (the swarm) is initially "thrown" inside the search space. Each particle has the following features:

- It has a position and a velocity
- It knows its position, and the objective function value for this position
- It knows its neighbours, best previous position and objective function value (variant: current position and objective function value)
- It remembers its best previous position

From now on, to put b) and c) in a common frame, we consider that the "neighbourhood" of a particle includes this particle itself.

At each time step, the behaviour of a given particle is a compromise between three possible choices:

- To follow its own way
- To go towards its best previous position
- To go towards the best neighbour's best previous position, or towards the best neighbour (variant)

This compromise is formalized by the following equations:

$$\begin{cases} v_{t+1} = c_1 v_t + c_2\left(p_{i,t} - x_t\right) + c_3\left(p_{g,t} - x_t\right) \\ x_{t+1} = x_t + v_{t+1} \end{cases}$$

Equ. 1. Basic equations in "classical" PSO

with

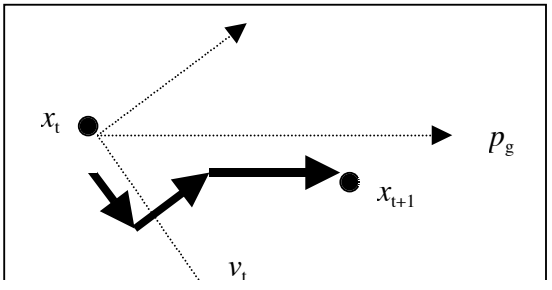
v_t := velocity at time step t

x_t := position at time step t

$p_{i,t}$:= best previous position, at time step t

$p_{g,t}$:= best neighbour' s previous best, at time step t,
(or best neighbour)

c_1, c_2, c_3 := social/cognitive confidence coefficients



The three social/cognitive coefficients respectively quantify:

- how much the particle trusts *itself now*
- how much it trusts its *experience*
- how much it trusts its *neighbours*

Social/cognitive coefficients are usually randomly chosen, at each time step, in given intervals.

Of course, many works have been done to study and to generalize this method (Eberhart and Kennedy 1995; Kennedy and Eberhart 1995; Kennedy 1997; Angeline 1998; Kennedy and Spears 1998; Shi and Eberhart 1998; Shi and Eberhart 1998). In particular, I use below a "nohope/rehope" technique as defined in (Clerc 1999), and the convergence criterion proved in (Clerc and Kennedy under review).

Also, there are many ways to define a "neighbourhood" (Kennedy 1999), but we can distinguish two classes:

- "physical" neighbourhood, which takes distances into account. In practice, distances are recomputed at each time step, which is quite costly, but some clustering techniques need this information.
- "social" neighbourhood, which just takes "relationships" into account. In practice, for each particle, its neighbourhood is defined as a list of particles at the very beginning, and does not change. Note that, when the process converges, a social neighbourhood becomes a physical one.

Discrete PSO

So, what do we really need for using PSO ?

- a search space of positions/states $S = \{s_i\}$
- a cost/objective function *f* on *S*, into a set of values $S \xrightarrow{f} C = \{c_i\}$, whose minimums are on the solution states.
- an order on *C*, or, more generally, a semi-order, so that for every pair of elements of C (c_i, c_j), we can say we have either $c_i < c_j$ or $c_i \geq c_j$.
- if we want to use a physical neighbourhood, we also need a distance *d* in the search space.

Usually, *S* is a real space R^D, and *f* a numerical continuous function. But in fact, nothing in Equ. 1 says it must be like that. In particular, *S* may be a finite set of states and *f* a discrete function, and, as soon as you are able to define the following basic mathematical objects and operations, you can use PSO:

- position of a particle
- velocity of a particle
- substraction $(position, position) \xrightarrow{\text{minus}} velocity$ (note time is discrete: time step=1)
- external multiplication $(real\_number, velocity) \xrightarrow{\text{times}} velocity$

- move $(position, velocity) \xrightarrow{\text{move (plus)}} position$

To illustrate this assertion, I have written yet-another-traveling-salesman-algorithm. This choice is intentionally very far from the "usual" use of PSO, so that we can have an idea of the power but also of the limits of this approach. The aim is certainly not to compete with powerful dedicated algorithms like LKH (Helsgaun 1997), but mainly to say something like "If you do not have a specific algorithm for your discrete optimization problem, use PSO: it works".We are in a finite case, so, to anticipate any remark concerning the NFL (No Free Lunch) theorem (Wolpert and Macready 1995), let us say immediately it does not hold here for at least two reasons: the number of possible objective functions is infinite and the algorithm does use some specific information about the chosen objective function by modifying some parameters (and even, optionaly, uses several different objective functions).

Note that a particular discrete (binary) PSO version has already be defined and successfully used (Kennedy and Eberhart 1997). Theoretically speaking, this work is then mainly a generalization.

PSO elements for TSP

Positions and state space

Let $G = \{E_G, V_G\}$ be the valuated graph in which we are looking for Hamiltonian cycles. E_G is the set of edges (nodes) and V_G the set of weighted vertices (arcs). Graph nodes are numbered from 1 to N. Each element of V_G is a triplet $(i, j, w_{i,j}), i \in \{1,\mathtt{K},N\}, j \in \{1,\mathtt{K},N\}, w_{i,j} \in R$. As we are looking for cycles, we can consider just sequences of N+1 nodes, all different, except the last one equal to the first one. Such a sequence is here called a N-cycle and seen as a "position". So the search space is defined as follow: the finite set of all N-cycles.

Objective function

Let us consider a position like $x = (n_1, n_2, \mathtt{K}, n_N, n_{N+1}), n_i \in E_G, n_1 = n_{N+1}$. It is "acceptable" only if all arcs (n_i, n_{i+1}) exist. In the graph, each existing arc as a value. In order to define the "cost" function, a classical way it to just complete the graph, and to create non existent arcs with an arbitraty value $l_{\sup}$ great enough to be sure no solution could contain such a "virtual" arc, for example

$$\begin{cases} l_{\sup} > l_{\max} + (N-1)(l_{\max} - l_{\min}) \\ l_{\max} = MAX(w_{i,j}) \\ l_{\min} = MIN(w_{i,j}) \end{cases}$$

So, each arc (n_i, n_{i+1}) has a value., a real one or a "virtual" one.

Now, on each position, a possible objective function can simply be defined by

$$f(x) = \sum_{i=1}^{N-1} w_{n_i, n_{i+1}}$$

This objective function has a finite number of values and its global minimum is indeed on the best solution.

Velocity

We want to define an operator *v* which, when applied to a position during one time step, gives another position. So, here, it is a permutation of *N* elements, that is to say a list of transpositions. The length of this list is $\|v\|$. A *velocity* is then defined by

$$v = \left(\left(i_k, j_k\right)\right), i_k \in \{1,\mathtt{L}\ ,N\}, j_k \in \{1,\mathtt{K}\ ,N\}, k \uparrow_1^{\|v\|}$$

or, in short

$$v = \left(\left(i_k, j_k\right)\right), k \uparrow_1^{\|v\|}$$

which means "exchange numbers $\left(i_1, j_1\right)$, then numbers $\left(i_2, j_2\right)$, etc. and at last numbers $\left(i_{\|v\|}, j_{\|v\|}\right)$.

Note that two such lists can be equivalent (same result when applied to any position); we note that by $v_1 \cong v_2$. Example: $\left((1,3),(2,5)\right) \cong \left((2,5),(1,3)\right)$. In fact, in this example, they are not only equivalent, they are opposite (see below): when using velocities to move on the search space, this one is like a "sphere".

A null velocity is a velocity equivalent to $\varnothing$, the empty list.

Opposite of a velocity

$$\begin{aligned} \neg v &= \left(\left(i_k, j_k\right)\right), k \downarrow_1^{\|v\|} \\ &= \left(\left(i_{\|v\|-k+1}, j_{\|v\|-k+1}\right)\right), k \uparrow_1^{\|v\|} \end{aligned}$$

It means "to do the same transpositions as in *v*, but in reverse order". It is easy to verify that we have $\neg\neg v = v$ (and $v \oplus \neg v \cong \varnothing$, see below Addition "velocity plus velocity").

Move (addition) "position *plus* velocity"

Let *x* be a position and *v* a velocity. The position $x' = x + v$ is found by applying the first transposition of *v* to *p*, then the second one to the result etc.

Example

$$\begin{cases} p = (1,2,3,4,5,1) \\ v = \left((1,2),(2,3)\right) \end{cases}$$

Applying *v* to *x*, we obtain successively

$(2,1,3,4,5,2)$

$(3,1,2,4,5,3)$

Substraction "position *minus* position"

Let x_1 and x_2 be two positions. The difference $x_2 - x_1$ is defined as the velocity *v*, found by a given algorithm, so that applying *v* to x_1 gives x_2. The condition "found by a given algorithm" is necessary, for, as we have seen, two velocities can be equivalent, even when they have the same size. In particular, the algorithm is chosen so that we have

and

$$x_1 = x_2 \Rightarrow v = x_2 - x_1 = \varnothing$$

Addition "velocity *plus* velocity"

Let v_1 and v_2 be two velocities. In order to compute $v_1 \oplus v_2$ we consider the list of transpositions which contains first the ones of v_1, followed by the ones of v_2. Optionnaly, we "contract" it to obtain a smaller equivalent velocity. In particular, this operation is defined so that $v \oplus \neg v = \varnothing$. We have then $\|v_1 \oplus v_2\| \leq \|v_1\| + \|v_2\|$, but we usually do not have $v_1 \oplus v_2 = v_2 \otimes v_1$.

Multiplication "coefficient *times* velocity"

Let *c* be a real coefficient and *v* be a velocity. There are different cases, depending on the value of *c*.

Case *c* = 0

We have $cv = \varnothing$

Case *c* ∈]0,1]

We just "truncate" *v*. Let $\|cv\|$ be the greatest integer smaller than or equal to $c\|v\|$. So we define

$$cv = \left(\left(i_k, j_k\right)\right), k \uparrow_1^{\|cv\|}$$

Case *c* > 1

It means we have $c = k + c', k \in \mathbf{N}^*, c' \in [0,1[$. So we define

$$cv = \underbrace{v \oplus v \oplus \cdots \oplus v}_{k \text{ times}} \oplus c'v$$

Case c < 0

By writing $cv = (-c)\neg v$, we just have to consider one of the previous cases.

Note that we have $v_1 \cong v_2 \Rightarrow cv_1 \cong cv_2$ if *c* is an integer, but it is usually not true in the general case.

Distance between two positions

Let x_1 and x_2 be two positions. The distance between these positions is defined by $d(x_1, x_2) = \|x_2 - x_1\|$. Note it is a "true" distance, for we do have (x_3 is any third position):

$$\|x_2 - x_1\| = \|x_1 - x_2\|$$

$$\|x_2 - x_1\| = 0 \Leftrightarrow x_1 = x_2$$

$$\|x_2 - x_1\| \leq \|x_2 - x_3\| + \|x_3 - x_1\|$$

The algorithm "PSO for TSP". Core and options

Equations

We can now rewrite Equ. 1as follow

$$\begin{cases} v_{t+1} = c_1 v_t \oplus c_2\left(p_{i,t} - x_t\right) \oplus c_3\left(p_{g,t} - x_t\right) \\ x_{t+1} = x_t + v_{t+1} \end{cases}$$

In practice, we consider here that we have $c_3 = c_2$. By defining an intermediate position

$$p_{ig,t} = p_{i,t} + \frac{1}{2}\left(p_{g,t} - p_{i,t}\right)$$

we finally use the following system

$$\boxed{\begin{cases} v_{t+1} = c_1 v_t \oplus c_2'\left(p_{ig,t} - x_t\right) \\ x_{t+1} = x_t + v_{t+1} \end{cases}}$$

Equ. 2. System used for the heuristic "PSO for TSP"

The point is we can then use (Clerc and Kennedy under review) as a guideline to choose "good" coefficients, even if the proofs given in this paper are for differentiable systems. The core of the algorithm simply applies Equ. 2 to each particles of a swarm, at each time step. It is indeed possible to use it just like that, but to avoid to be "sticked" in local minimums, the swarm size has sometimes to be quite big. So, some options are useful, in order to improve performances, in particular the NoHope/ReHope process.

NoHope tests

Criterion 0

If a particle has to move "towards" another one which is at distance 1, either it does not move at all or it goes exactly at the same position than this other one, depending on the social/confidence coefficients. As soon as the swarm size is smaller than $N(N\text{-}1)$, it may arrive that all moves computed according to Equ. 2 are null. In this case there is absolutely no hope to improve the current best solution.

Criterion 1

The NoHope test defined in (Clerc 1999) is "swarm too small". It needs to compute the swarm diameter at each time step, which is costly. But in a discrete case like here, as soon as the distance between two particles tends to become "too small", the two particles become identical (usually first by positions and then by velocities). So, at each time step, a "reduced" swarm is computed, in which all particles are different, which is not very expensive, and the NoHope test becomes "swarm too reduced", say by 50%.

Criterion 2

Another criterion has been added: "swarm too slow", by comparing the velocities of all particles to a threshold, either individually or globally. In one version of the algorithm, this threshold is in fact modified at each time step, according to the best result obtained so far and to the statistical distribution of arc values.

Criterion 3

Another very simple criterion is defined: "no improvement from too many times", but in practice, it appears that criteria 1 and 2 are sufficient.

ReHope process

As soon as there is "no hope", the swarm is re-expanded. The two first methods used here are inspired by the ones described in (Clerc 1999) and (He, Wei et al. 1999).

Lazy DescentMethod (LDM)

Each particle goes back to its previous best position and, from there, moves randomly and slowly ($\|v\| = 1$) and stop *as soon as* it finds a better position or when a maximal number of moves $M_{\max}$ is reached (in the examples below $M_{\max} = N$). If the current swarm is smaller than the initial one, it is completed by a new set of particles randomly chosen.

Deep Descent Method (EDM)

Each particle goes back to its previous best position and, from there, moves slowly ($\|v\| = 1$) *as long as* it finds a better position in at most $M_{\max}$ moves. If the current swarm is smaller than the initial one, it is completed by a new set of particles randomly chosen. You may find, by chance, a solution, but it is more expensive than LDM. Ideally, it should be used only if the combination Equ. 2 + LDM seems to fail.

Local Iterative Levelling (LIL)

This method is more powerful … and more expensive. In practice, it should be used only when we *know* there is a better solution, but the combination Equ. 2 + DDM fails to find it.

The idea is that, as there are an infinity of possible objective functions with the same global minimum, we can locally and temporarily use any of them to guide a particle. For each immediate physical neighbour y (at distance 1) of the particle x, a temporary objective function value $f_t(y)$is computed by using the following algorithm:

LIL ALGORITHM

Find all neighbours at distance 1

Find the best one, $y_{\min}$

Give to y the temporarily objective function value $f_t(y) = \dfrac{f(y_{\min}) + f(x)}{2}$

---

and, according to these temporary evaluations, x moves to its best immediate neighbour. In TSP, such an algorithm is $O(N^2)$. If repeated to often, it increases considerably the total cost of the process. That is why it should be used only if there is really "no hope" (and i f you do want the best solution, for, in practice, the algorithm has usually already found a good one).

Adaptive ReHope Method (ARM)

The three above methods can be automatically used in an adaptive way, according for how long (number of time steps) the best solution has not been improved. An example of strategy is:

Number of time steps without improvement	ReHope type
0-1	No ReHope
2-3	Lazy Descent Method
4	Energetic Descent Method
>4	Local Iterative Levelling

Queens

Instead of using the best neighour/neighbour's previous best of each particle, we can use an extra-particle, which "summarizes" the neighbourhood. This method is a combination of the (unique) queen method defined in (Clerc 1999) and of the multi-clustering method described in (Kennedy 2000). For each neighbourhood, we iteratively build a gravity center and take it as best neighbour (p_g in Equ. 1).

Extra-best particle

In order to speed up the process, the algorithm can also use a special extra-particle, just to keep the best position found so far from the very beginning. It is not absolutely necessary, for this position is also memorized as "previous best" in at least one particle, but it may avoid a whole sequence of iterations between two ReHope processes.

Parallel and sequential versions

The algorithm can run either in (simulated) parallel mode or in sequential mode. In the parallel mode, at each time step, new positions are computed for all particles and then the swarm is globally moved. In sequential mode, each particle is moved at a time on a cycling way. So, in particular, the best neighbour used at time t+1 may be not anymore the same as the best neighbour at time t, even if the iteration is not complete. Note that Equ. 1implicitely supposes a parallel mode, but in practice there is no clear difference in performances on a given sequential machine, and the second method is a bit less expensive.

Examples and results

Parameters choice

The purpose of this paper is mainly to show how PSO can be used, in principle, on a discrete problem. That why we use here mainly "default" values. Of course, further works should study what we could call "optimization of optimization" (in fact, PSO itself may be used to find an optimum in a parameter space).

Social/cognitive coefficients

If nothing else is explicitely mentioned, in all the examples we use $c_1 \in]0,1[$ (typically 0.5 if NoHope/ReHope is used or 0.999 if not) and c_2 randomly chosen at each time step in $[0,2]$. The convergence criterion defined in (Clerc and Kennedy under review) is satisfied. This criterion is proved only for differentiable objective functions , but it is not unreasonable to think it should work here too. At least, in practice, it does.

Swarm size and neighbourhood size

Ideally, the best swarm and neighbourhood sizes are depending on the number of local minimums of the objective function, say $n_{\text{local\_min}}$, so that the swarm can have sub-swarms around each of these minimums. Usually, we simply do not have this information, except the fact that there are at most *N*-1 such minimums. A possible way is to use first a very small swarm, just to have an idea of how looks like the landscape defined by the objective function (see the example below). Also, a statistical estimation of $n_{\text{local\_min}}$ can be made, using the arc values distribution.

In practice, we usually use here a swarm size S equal to *N*-1 (see below Structured Search Map).

Some considerations, not completely formalized, about the fact that each particle uses three velocities to compute a new one, indicate that a good neighbourhood size should be simply 4 (including the particle itself).

But all this is still "rules of thumb", and this is the main limit of this approach: on a given example, we do not know in advance what are the best parameters and usually have to try different values.

Performance criteria

Too often, we can read in some works something like "to solve the example xxxx from TSPLIB, my program takes 3 seconds on a XYZ machine …". Unfortunately, I do not have a XYZ machine. It is already better when the given information is something like "it needs 7432 tour evaluations to reach a solution". But it is not all the same thing to completely compute the objective function on a *N*-cycle and to re-compute it after having swapped two nodes: the first case is about *N*/4 times more expensive. So we use here two criteria: the number of position evaluations *and* the number of arithmetical/logical operations. It is still not perfect, in particular for the same algorithm can be more or less good written, even for a given language, but at least, if you know your machine is a xxxMips computer, you can estimate how long it would take to run the example.

A toy example as illustration

We use here a very simple graph (17 nodes) from TSPLIB, called br17.atsp (cf. Appendix). The minimal objective function value is 39, and there are several solutions. Although it has just a few nodes, it is designed so that finding one of the best tours is not so easy, so it is nevertheless an interesting example.

How the landscape looks like

We define randomly a small swarm of, say, five particles, where p_{best} is the best one. For each other particle p_i, we now consider the sequence $p_k = p_i + \frac{k}{N-1}\left(p_{best} - p_i\right), k \uparrow_0^{N-1}$

and plot the graph $\left\{\frac{k}{N-1}, \left|f\left(p_{best}\right) - f\left(p_k\right)\right|\right\}$. It give us an idea of the landscape.

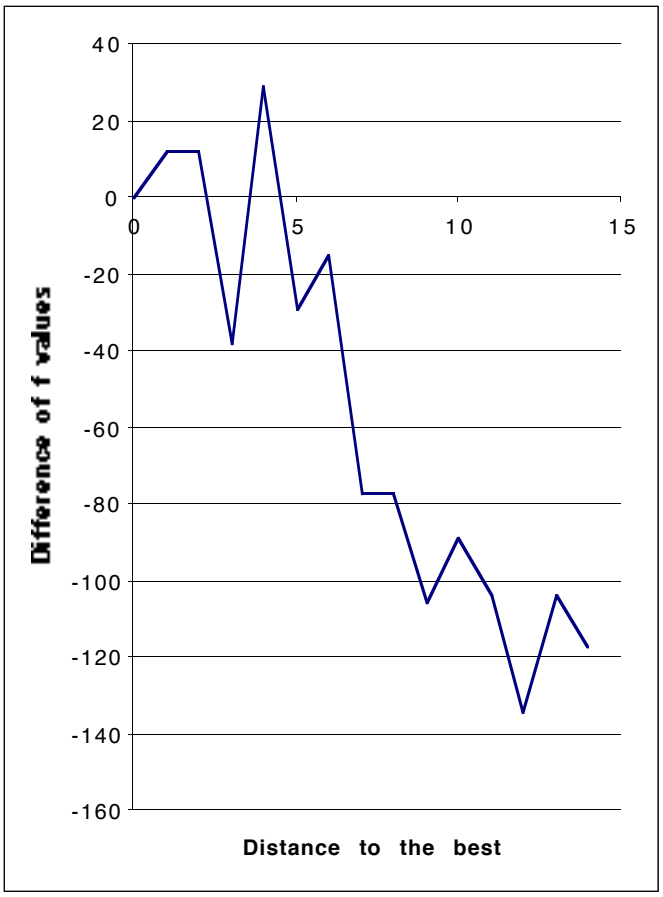


Figure 2. One section of the search space landscape

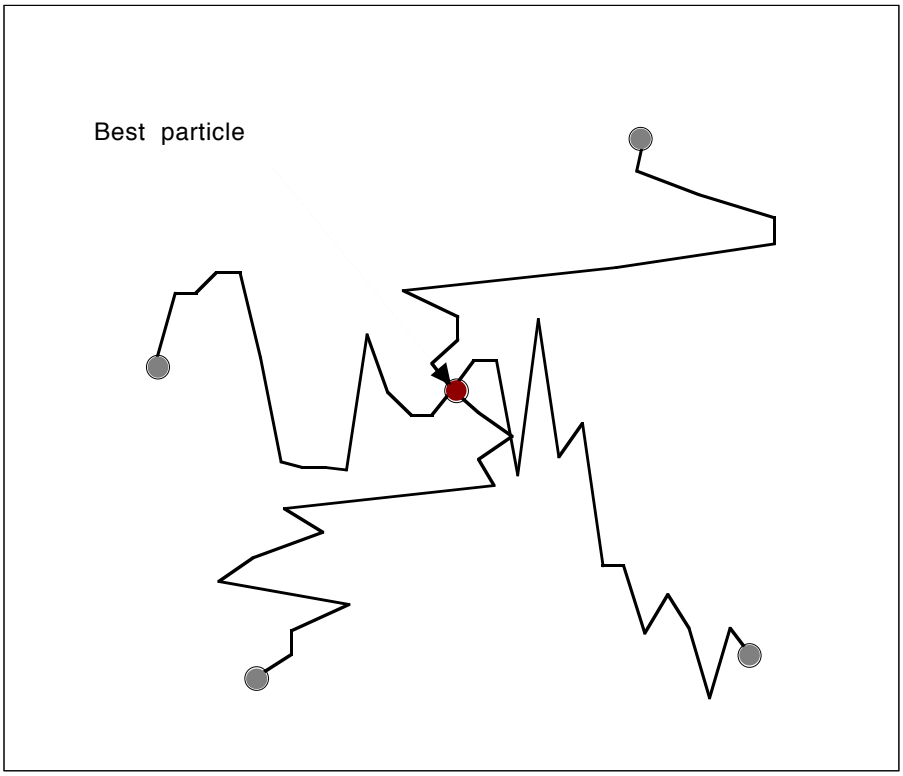


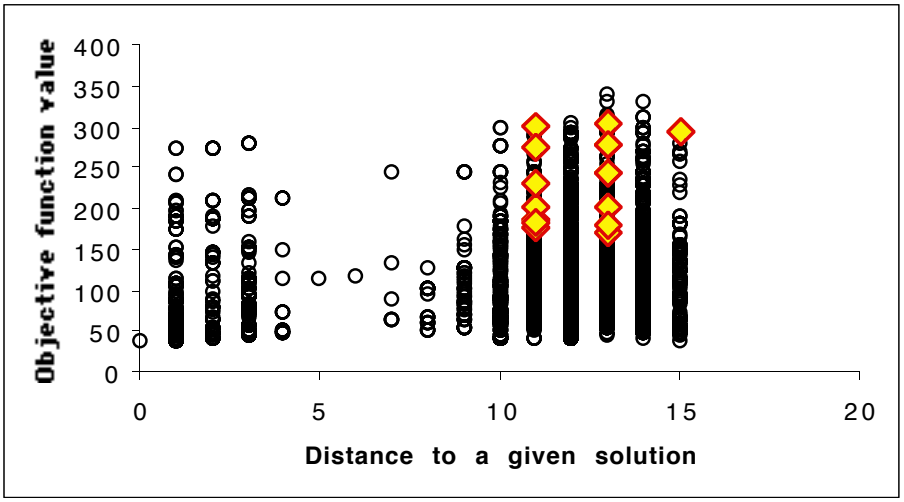
Figure 3. Main sections for a 5-swarm

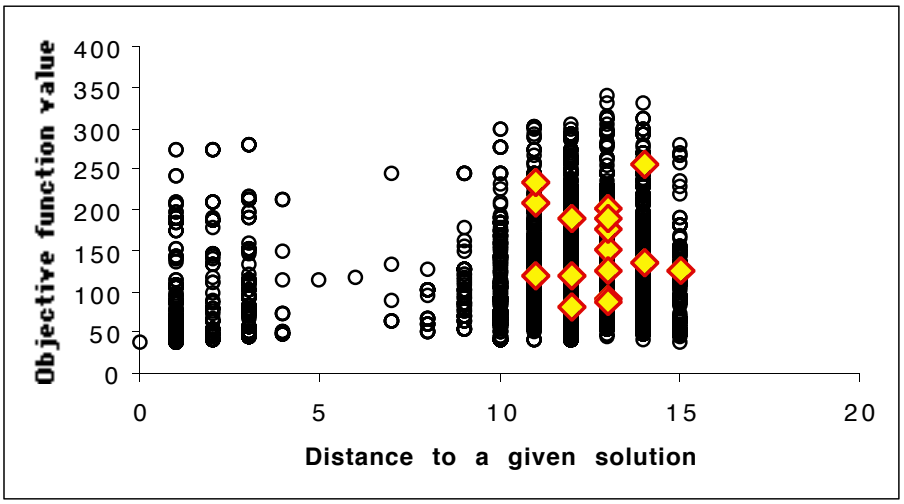
The landscape is clearly quite chaotic, so we surely will have to use quite often the NoHope/ReHope process. Also, Queens option is probably not efficient.

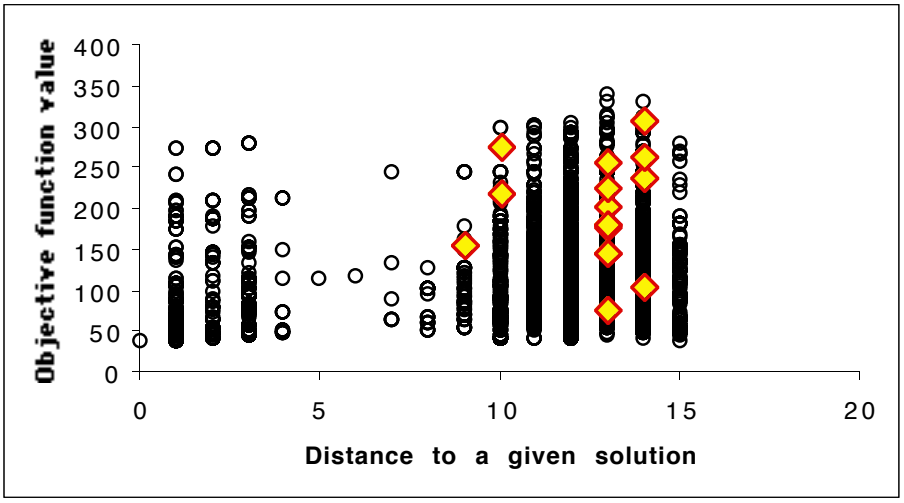
Structured Search Map. How the swarm moves

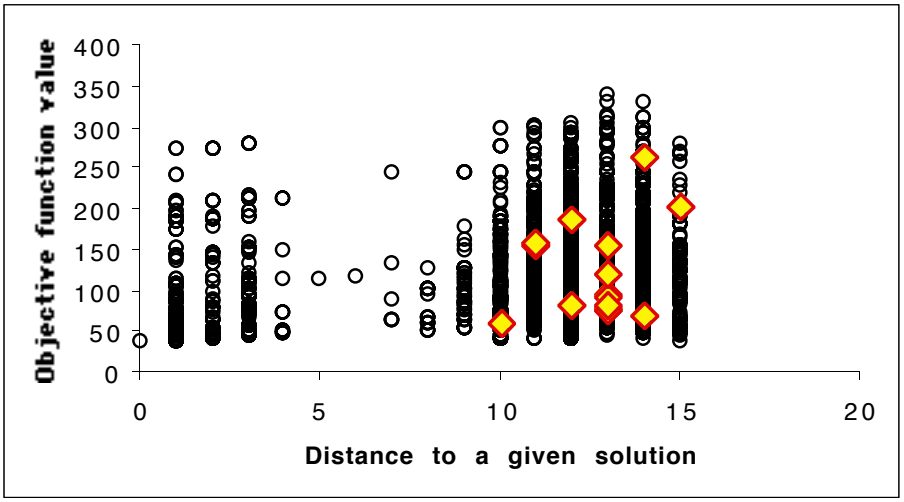
Let us suppose we know a solution x_{sol} . We consider all the positions of the search space, an plot the map $\left(d\left(x_{sol},x\right),f\left(x\right)\right)$. On this map, at each time step, we highlight the positions of the particles, so that we can see how it moves. We have some equivalence classes according to the equivalence relation $x \cong x' \Leftrightarrow d\left(x_{sol},x\right) = d\left(x_{sol},x'\right)$. As we can see, not only is the swarm globally moving towards the solution, but also some particles tend to move towards the minimum (in terms of objective function value) of the equivalence classes. It means that even if it do not find the best solution, it finds at least (and usually quite rapidly) interesting quasi-solutions. It also means that if the swarm is big enough, we may simultaneously find several solutions.

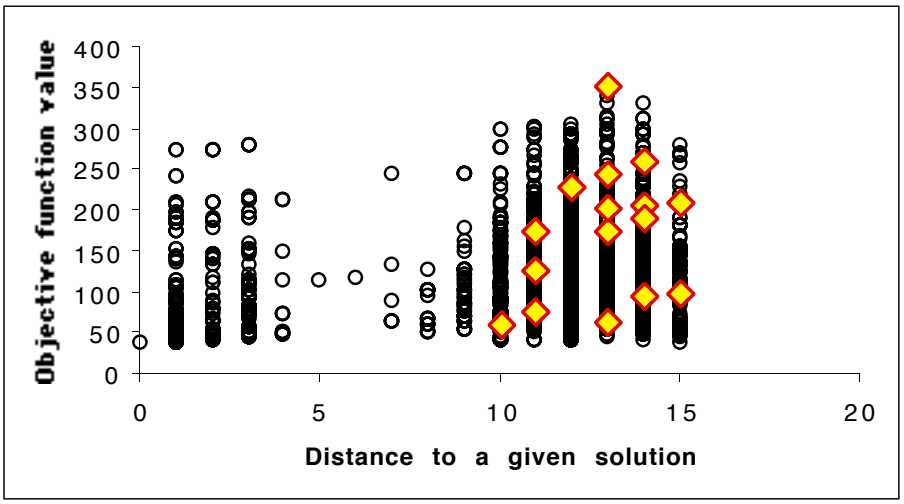
In our example, as the number of positions is quite big ($16! \approx 21^{12}$), we plot only a few selection of them (4700). We use here a swarm size of 16. In Figure 4, we see clearly that the example has been designed to be a bit difficult, for there is a "gap" between distances 4 and 8, but, finally, the swarm finds three solutions in one "standard" run, and some others if we modify the parameters (see Appendix).

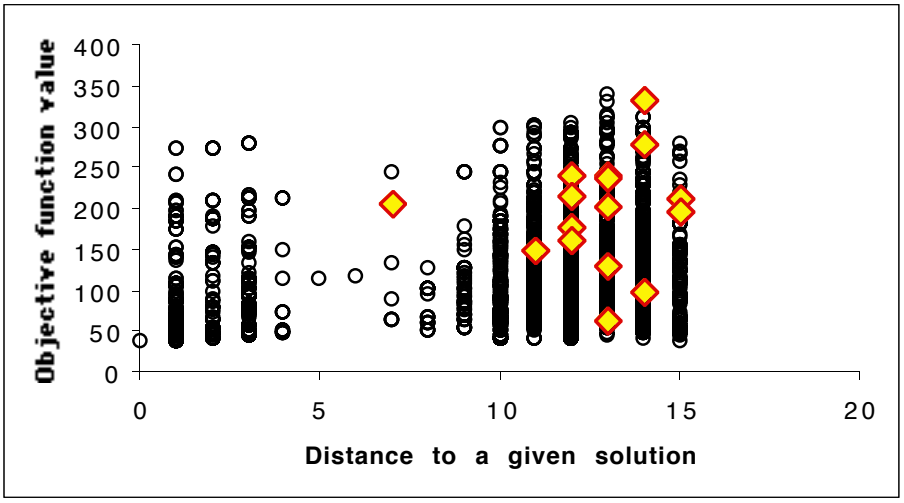

Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution


Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution


Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution


Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution


Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution


Objective function value
400
350
300
250
200
150
100
50
0
0
5
10
15
20
Distance to a given solution

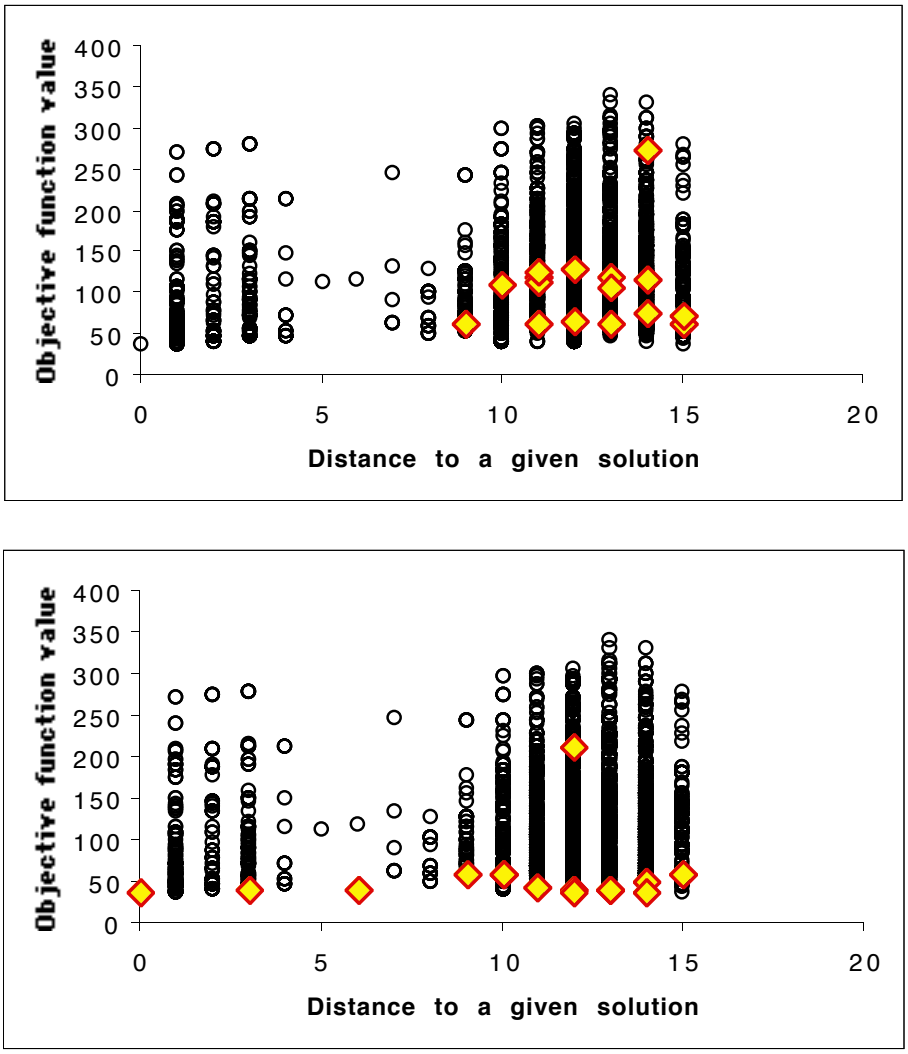


Figure 4. How the swarm moves on the Structured Search Map

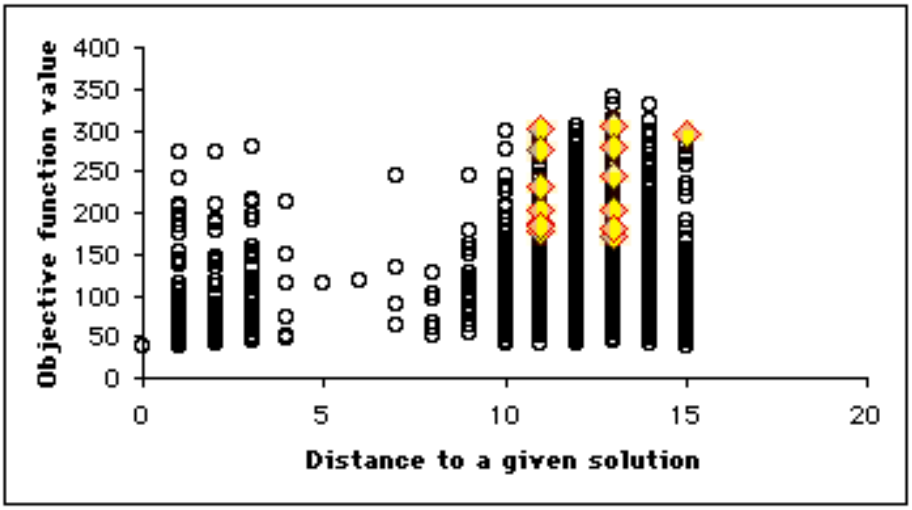


Figure 5. How the swarm moves: animation.

Some results, and discussion

Table 1. Some results using Discrete PSO on the graph br17.atsp

Swarm size	Hood size	c_1	Random c_2	ReHope type	Best solution	Hood type	Tour evaluations	Arithmetical /logical operations
16	4	0.5	]0,2]	ADM	39 (3 solutions)	social	7990	4.8 M
16	4	0.5	]0,2]	ADM	39	physical	7742	6.3 M
16	4 with queens	0.5	]0,2]	ADM	39 (3 solutions)	social or physical	9051	5.8 M
8	4	0.5	]0,2]	ADM	39	social	4701	2.8 M
1	1	0.5	]0,2]	ADM	44	social	926 to ∞	0.6 M
128	4	0.999	]0,2]	no	47	social	41837 to ∞	33.2 M to ∞
32	4	0.999	]0,2]	no	66	social	14351 to ∞	10.8 M to ∞
16	4	0.999	]0,2]	no	86	social	2880 to ∞	1.9 M to ∞

As we can see on Table 1:

- Using "physical" neighbourhood may need less tour evaluations, but is much more expensive than "social" option, if we consider the total number of arithmetical/logical operations (distances have to be recaculated at each time step). The fact that with social neighbourhood you find more solutions is not a general rule.
- Using only ReHope methods (in this case, using s particles for T time steps is equivalent to use just one particle for sT time steps) is not enough to reach the best solution.
- Using only core algorithm, with no ReHope at all, is not enough to reach the best solution, unless, probably, you use a huge swarm.
- Using queens is not interesting.

Results are not particularly good nor bad, mainly for we use a generic ReHope method which is not specifically designed for TSP. But this is on purpose: so, by just changing the objective function, you could use Discrete PSO for different problems.

Appendix

Graph br17.atsp (from TSPLIB)

In the original data, diagonal values are equal to 9999. Here, they are equal to 0. For PSO algorithm, it changes nothing, and it simplifies the visualization program.

```
NAME:  br17
```

```
DIMENSION:  17
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION
  0  3  5 48 48  8  8  5  5  3  3  0  3  5  8  8  5
  3  0  3 48 48  8  8  5  5  0  0  3  0  3  8  8  5
  5  3  0 72 72 48 48 24 24  3  3  5  3  0 48 48 24
 48 48 74  0  0  6  6 12 12 48 48 48 48 74  6  6 12
 48 48 74  0  0  6  6 12 12 48 48 48 48 74  6  6 12
  8  8 50  6  6  0  0  8  8  8  8  8  8 50  0  0  8
  8  8 50  6  6  0  0  8  8  8  8  8  8 50  0  0  8
  5  5 26 12 12  8  8  0  0  5  5  5  5 26  8  8  0
  5  5 26 12 12  8  8  0  0  5  5  5  5 26  8  8  0
  3  0  3 48 48  8  8  5  5  0  0  3  0  3  8  8  5
  3  0  3 48 48  8  8  5  5  0  0  3  0  3  8  8  5
  0  3  5 48 48  8  8  5  5  3  3  0  3  5  8  8  5
  3  0  3 48 48  8  8  5  5  0  0  3  0  3  8  8  5
  5  3  0 72 72 48 48 24 24  3  3  5  3  0 48 48 24
  8  8 50  6  6  0  0  8  8  8  8  8  8 50  0  0  8
  8  8 50  6  6  0  0  8  8  8  8  8  8 50  0  0  8
  5  5 26 12 12  8  8  0  0  5  5  5  5 26  8  8  0
EOF
```

Some solutions found by Discrete PSO

```
 1  3 14 11 13  2 10 15  7  6 16  4  5  9 17  8 12 1

 1  6  7 15 16  5  4 17  9  8 10  2 13 11 14  3 12 1

 1  7  6 15 16  5  4  9  8 17 10 11 13  2 14  3 12 1

 1  9  8 17  5  4 15  6  7 16 13 10 11  2  3 14 12 1

 1 12  3 14 10  2 13 11 17  8  9  5  4  7  6 16 15 1

 1 12  3 14 10 11  2 13 17  8  9  4  5 16  6 15  7 1

 1 12  6  7 15 16  5  4  9 17  8  2 11 13 10 14  3 1

 1 12  8  9 17  5  4 16 15  7  6 10  2 11 13 14  3 1

 1 12 14  3  2 10 11 13 17  9  8  5  4 15  6  7 16 1

 1 12 14  3 11  2 10 13 15  6 16  7  4  5  9  8 17 1

 1 12 15 16  7  6  5  4  8 17  9 11 10  2 13 14  3 1

 1 12 16 15  7  6  4  5  9  8 17 10  2 11 13 14  3 1

 1 12 16 15  7  6  5  4  8  9 17 10 11  2 13 14  3 1

 1 12 16  7  6 15  5  4  9 17  8 11 10 13  2 14  3 1

 1 12 17  8  9  5  4 15 16  7  6  2 10 11 13  3 14 1

 1 12 17  8  9  5  4 15 16  7  6  2 10 11 13  3 14 1

 1 14  3 10 13 11  2 17  8  9  4  5 15  6 16  7 12 1

 1 14  3 10 13 11  2  8 17  9  4  5 15  6  7 16 12 1

 1 14  3 10 11  2 13  8 17  9  4  5 15  7  6 16 12 1

 1 16  6  7 15  4  5  9  8 17 13 11 10  2  3 14 12 1
```

```
 1 17  9  8  4  5  7  6 16 15 10 11  2 13 14  3 12 1

 1 12 14  3  2 10 13 11 16  6  7 15  4  5 17  9  8 1
```

References

Angeline, P. J. (1998). Using Selection to Improve Particle Swarm Optimization. IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9.

Clerc, M. (1999). The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. Congress on Evolutionary Computation, Washington D.C., IEEE.

Clerc, M. and J. Kennedy (under review). "The Particle Swarm: Explosion, Stability, and Convergence in a Multi-Dimensional Complex Space." .

Eberhart, R. C. and J. Kennedy (1995). A New Optimizer Using Particles Swarm Theory. Proc. Sixth International Symposium on Micro Machine and Human Science, Nagoya, Japan, IEEE Service Center, Piscataway, NJ.

He, Z., C. Wei, et al. (1999). A New Population-based Incremental Learning Method for the Traveling Salesman Problem. Congress on Evolutionary Computation, Washington D.C., IEEE.

Helsgaun, K. (1997). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, Department of Computer Science,Roskilde University, Denmark.

Kennedy, J. (1997). The Particle Swarm: Social Adaptation of Knowledge. IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana, IEEE Service Center, Piscataway, NJ.

Kennedy, J. (1999). Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. Congress on Evolutionary Computation, Washington D.C., IEEE.

Kennedy, J. (2000). Stereotyping: Improving Particle Swarm Performance With Cluster Analysis. submitted.

Kennedy, J. and R. C. Eberhart (1995). Particle Swarm Optimization. IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ.

Kennedy, J. and R. C. Eberhart (1997). A discrete binary version of the particle swarm algorithm. International Conference on Systems, Man, and Cybernetics.

Kennedy, J. and W. M. Spears (1998). Matching Algorithms to Problems: An Experimental Test of the Particle Swarm and Some Genetic Algorithms on the Multimodal Problem Generator. IEEE International Conference on Evolutionary Computation.

Shi, Y. and R. C. Eberhart (1998). Parameter Selection in Particle Swarm Optimization, Indiana University Purdue University Indianapolis.

Shi, Y. H. and R. C. Eberhart (1998). A Modified Particle Swarm Optimizer. IEEE International Conference on Evolutionary Computation, Anchorage, Alaska, May 4-9.

Wolpert, D. H. and W. G. Macready (1995). No Free Lunch for Search, The Santa Fe Institute.