Cognitive Bare Bones Particle Swarm Optimisation with Jumps

Mohammad Majid al-Rifaie, Tim Blackwell
Department of Computing
Goldsmiths University of London
London SE14 6NW, United Kingdom
Email: m.majid, t.blackwell @ gold.ac.uk

***Abstract*—The 'bare bones' (BB) formulation of particle swarm optimisation (PSO) was originally advanced as a model of PSO dynamics. The idea was to model the forces between particles with sampling from a probability distribution in the hope of understanding swarm behaviour with a conceptually simpler particle update rule. 'Bare bones with jumps' (BBJ) proposes three significant extensions to the BB algorithm: (i) two social neighbourhoods, (ii) a tuneable parameter that can advantageously bring the swarm to the 'edge of collapse' and (iii) a component-by-component probabilistic jump to anywhere in the search space. The purpose of this paper is to investigate the role of jumping within a specific BBJ algorithm, cognitive BBJ (cBBJ). After confirming the effectiveness of cBBJ, this paper finds that: jumping in one component only is optimal over the 30 dimensional benchmarks of this study; that a small per particle jump probability of $1/30$ works well for these benchmarks; jumps are chiefly beneficial during the early stages of optimisation and finally this work supplies evidence that jumping provides escape from regions surrounding sub-optimal minima.**

I. Introduction

PARTICLE swarm optimisation (PSO) is a population based optimisation technique developed by Kennedy and Eberhard[1]. The particle swarms of optimisation algorithms, in distinction to the swarms in models of animal flocking, swarming and herding, communicate via a social information network rather than on rules dependent on spatial proximity. Each particle remembers its best achieved position, as determined by an objective function, and shares this information with social neighbours who use this knowledge to inform their own exploration.

There have been many attempts to understand the behaviour of the particles in PSO; the studies mainly concentrating on single particle trajectories and formal convergence to a stable point in the absence of particular-particle interaction [2], [3], [4], [5], [6], [7], [8], [9]. However an understanding of the dynamics of the interacting particles is elusive. This is due to the complexity of the stochastic particle update [10] and the relationship between particle memory, network topology and objective function.

In 2003, Kennedy [10] advanced a model of PSO dynamics ('bare bones') where the velocity and position update is replaced by Gaussian sampling around the average of the neighbourhood and personal best positions. This provided an arguably simpler model of particle motion.

The original bare bones formulation is not competitive to standard PSO [11], [12], but the original idea has been extended. A version based on a broader distribution tail has been proposed [11] with the aim of improving exploration. Krohling considered a version with particle re-initialisation in which all components of particle position are randomised within the search space if a particle has not improved itself over a given number of iterations [13]. A bare bones with component-wise jumps and two social neighbourhoods, BBJ, has also been advanced [14]. The jumps of BBJ are applied probabilistically and independently of particle performance. The effect is to broaden distribution tails and promote escape from local optima [14].

The consequence of these studies is that bare bones swarms can be regarded as effective optimisers in their own right, and are not merely as a model of PSO dynamics.

BBJ, has, by virtue of its dual neighbourhoods and choice of jumping mechanism, a rich set of possible formulations. The particular instances studied here will be termed social (sBBJ) and cognitive BBJ (cBBJ).

Current particle position x in sBBJ does not influence search unless this position coincides with the historical best position achieved. In a sense, one could say that the search is governed by social rather than personal information. The search is focused on the best position, g, of any neighbour within a particular neighbourhood (the μ-neighbourhood) and the search spread is determined by the separation of neighbour best positions in a second social neighbourhood (the σ-neighbourhood). The two neighbourhoods coincide in other bare bones formulations. Good results are obtained by taking a global (the entire swarm) μ-neighbourhood and a small local σ-neighbourhood [14]. Figuratively speaking, a particle attempts to better itself by copying a public leader (global μ-neighbourhood), yet it also distinguishes itself by imitating the observed degree of non-conformity within a more intimate group (the local σ-neighbourhood).

However current particle position does play a role in cBBJ; the standard deviation of the Gaussian sampling is proportional to the distance (in each component) between x and g [15]. This introduces a *cognitive* element into the search criteria;

the degree of non-conformity is specified by the how much the particle itself differs from the leader. The search however, is still focused on the μ-neighbourhood leader.

Particle jumps in [14] and [15] are applied probabilistically on a component by component basis (per-component). The result is that the total number of component jumps per particle is not fixed. Although this is not necessarily problematic, and indeed when viewed as a tail broadening mechanism, probabilistic component jumps are entirely consistent with the general PSO update scheme, namely a component-by-component update, whether according to a velocity mediated rule, or by sampling. This is not the only possible rule; the total number of component jumps could be fixed by selecting a particle for jumping according to a fixed probability, and then instigating a set number of component jumps (per-particle). The jumping mechanism considered in this paper is the per-particle version.

The findings of this paper are:

1) Social and cognitive BBJ are valid optimisers in their own right, and not just models of PSO behaviour.
2) The main conclusion in terms of performance on the 30 dimensional set of benchmarks is that cognitive BBJ with a global search focus and with a per particle jump probability of $1/30$ in a single dimension is preferred for a fifty particle swarm, providing performance that is equivalent to some state-of-the-art PSOs on a standard test set.
3) The investigations reveal that (i) the dimensionality of the jump subspace is optimally set to one, (ii) jumps are most successful at the early stages of the optimisation, and (iii) there is evidence that the majority of jumps provide an escape from a local optimum.

The paper is organised as follows: section II presents a general formulation of PSO that emphasises the role of particle histories and memories. The paper continues with an account of bare bones PSO, with and without jumps. Section IV analyses the BBJ algorithms at stagnation, and when converging on a local optimum. A bound to α, the parameter that controls the variance of gaussian sampling is derived, as well as an estimate of the jumping probability for cBBJ.

Section V and VI report on the empirical investigations of particle jumps, and on the efficacy of the c/sBBJ algorithms on a standard benchmark. The paper ends with a summary of the main findings.

II. Particle swarm optimisation

A particle swarm consists of an interacting collection of particles that move through a search space Ω. Each particle is specified by its state at any given update step; the state is a collection of dynamic values that are associated with the particle's position in Ω. Particles move according to a stochastic rule that depends on their own state history and the history of neighbouring particle states in a social network. The rule balances the influence of individual history (the 'cognitive' influence) with the influence of neighbours' history (the 'social' component). Particles interact via the social component and in doing so attempt to better themselves by finding higher quality solutions to some problem, as determined by an objective function, $f : \Omega \to \mathbb{R}$.

More formally, suppose that particle i has a state $s_i(t)$ where $t \in \mathbb{N}^+$ counts the number of applications of the stochastic particle update rule. A swarm of N particles at update t is then specified by the collection of states $s(t) = (s_1(t), s_2(t), \ldots, s_N(t))$.

A particle is usually a very simple object; its state is essentially a vector $x_i(t)$ within the search space Ω at update t. However, for computational purposes, particle state $s_i(t)$ can be expanded to include a memory $p_i(t) \in \Omega$ of previously visited positions, a velocity $v_i(t) \in \Omega$, and a function value $f_i(t) = f(x_i(t))$, but these quantities are not independent and can be derived from the position (see below).

For simplicity we suppose that the search space is a parallelepiped (box) in D-dimensional real space, $\Omega = (\Omega_1, \Omega_2, \ldots, \Omega_D) \in \mathbb{R}^D$. Then x_i, v_i and p_i are D dimensional real vectors $(x_{i1}, x_{i2}, \ldots, x_{iD})$ etc.

Memories and histories

For the purposes of economy, a sequence of objects $a(1), a(2), \ldots, a(t)$ will be notated $[a(t)]$. The sequence of all states acquired by particle i at update t i.e. the *history* of particle i, is then represented by $[s_i(t)]$. In principle, the total information available to any algorithm at update t is the entire history of the swarm, $[s(t)]$.

The swarm evolves according to individual particle update rules and hence generates a trajectory $[x_i(t)]$ for each particle. Each position has an objective value $f(x_i(t))$; this value will occasionally be abbreviated by $f_i(t)$. The trajectory has an associated function value history $[f(x_i(t))]$.

The particle 'memory', $p_i(t)$, is usually defined as the optimal position in $[x_i(t)]$ with respect to f. In the case of minimisation,

$$p_i(t) = \arg^* \min [f(x_i(t))] \tag{1}$$

where the starred $\arg$ operator indicates the argument of an arbitrary member of $\min [f]$ if indeed $\min [f]$ is not unique. This might happen if the particle does not move, or if it moves to a position with identical function value to a previous and different position. One possible rule is to select the most recent member of $\min[f(x_i(t))]$.

According to this definition, $p_i(1) = x_i(1)$, and the memory history $[p_i(t)] \equiv p_i(1) \ldots p_i(t)$ is ordered according to function value: $f_i(1) \geq f_i(2) \geq \ldots f_i(t)$.

A particle velocity $v_i(t) \in \Omega$ can be defined as $v_i(t) \equiv x_i(t) - x_i(t-1), t > 1$ whenever this difference lies in Ω; otherwise a further rule must specify how to obtain $v_i(t)$ from $x_i(t)$ and $x_i(t-1)$. $v_i(1)$ is an arbitrary point in Ω.

For computational purposes, it may be more efficient to define a particle state as the collection $s_i(t) = (x_i(t), p_i(t), f_i(t), v_i(t))$. However $p_i(t)$, $f_i(t)$ and $v_i(t), t > 1$, are derivable from the position history $[x_i(t)]$.

Memory update rule

A memory update rule can be inferred from the definition of $p_i(t)$. One rule is[1]

$$p_i(1) = x_i(1)$$
$$p_i(t>1) = \begin{cases} x_i(t) \text{ if } f(x_i(t)) \leq f(p_i(t-1)) \\ p_i(t-1) \text{ otherwise.} \end{cases}$$

We also note that the collection of memories at step t, $(p_1(t), p_2(t), \ldots, p_N(t))$, may not include the N best positions ever attained by the swarm since memory update depends on personal and not group performance. However schemes could be devised to ensure that (for example) the N best positions are shared amongst the N particles at all steps; in that case Eq. 1 would be replaced by a function of the swarm as a whole, and not just on personal history.

The social network

Each particle participates in a social network or neighbourhood, $N_i(t)$, consisting of a subset of the swarm. The term social means that neighbours are defined by a graph of relationships, and not by spatial locality or by function value.

A neighbourhood that includes all the swarm is *global*; otherwise it is *local*. The social neighbourhood of a particle may exclude the particle itself (an *open* neighbourhood; otherwise it is a *closed* neighbourhood).

A *static* network is not subject to change during the course of the optimisation, $N_i(t) = N_i$. Otherwise, the network is *dynamic*. A collection of social networks is *symmetric* if the relationship between neighbours is the same for each particle, as in the case of the ring network $N_i^R = \{i \ominus 1, i \oplus 1\}$[2], or the global network $N_i^G = \{1, 2, \ldots, N\}$.

For the interests of notational simplicity we shall assume static and symmetric neighbourhoods in the remainder of this section.

We imagine that potentially particles have access to all the information carried by its neighbours at step t. The information carried by particle i is its state history $[s_i(t)]$. Hence particle i at step t has access to $[s_i(t-1)]$ and to $[s_j(t-1)]$ for each neighbour j in N_i[3].

The neighbourhood best memory, $g_i(t)$ of particle i at time t is defined as

$$g_i(t) = \arg^* \min\ (f(p_j(t))), \quad j \in N_i, \tag{2}$$

where, once more, one of $\arg\min$ is returned if $\arg\min$ is not unique.

[1] An alternative is

$$p_i(1) = x_i(1)$$
$$p_i(t) = \begin{cases} x_i(t) \text{ if } f(x_i(t)) < f(p_i(t-1)) \\ p_i(t-1) \text{ otherwise.} \end{cases}$$

but the rule with $\leq$ is usually preferred since it enforces movement even if there is no improvement.

[2] The symbols $\oplus, \ominus$ stand for arithmetic mod N.

[3] This is for synchronous updating; in asynchronous versions, particle i has access to $S_{j<i}(t)$ and $S_{j>i}(t-1)$.

Position update rule

In principle the particle position update rule is a function of its entire history $[s_i(t)]$ (the 'cognitive' component) and the entire history of particles in its neighbourhood, $[s_{j\in N_i}(t)]$ (the 'social' component). The movement rule might also depend on random variables $\xi_i(t)$, real parameters $\alpha_i(t)$ and on details of the function domain Ω (the update rule must specify a valid point within the search space Ω.

For synchronous updating and $t > 1$, the general rule R_t is

$$x_i(t) = R_t([s_i(t-1)], [s_{j\in N_i}(t-1)], \xi_i(t), \alpha_i(t), \Omega).$$

An increasing amount of information is available at each step and in consequence the number of arguments of R_t grows, giving rise to a family of rules, R_t, where each rule is indexed by t. It is also possible to implement a different type of rule at different stages of the optimisation.

This is a very general formulation, but it encapsulates the essential elements of PSO: *a procedure for generating new trial positions for a population of searchers participating in overlapping information-sharing networks*.

In 'standard' formulations, as exemplified by the Clerc-Kennedy (CK) PSO [2], the particle update rule depends on (i) particle velocity, $v_i(t-1)$, (ii) the best current memory $g_i(t-1)$ of particles in its neighbourhood and (iii) $p_i(t-1)$. The real parameters $\alpha_i(t) = \alpha(t)$ are the same for each particle and are constant[4]. Hence, for $t > 0$:

$$x_i(t) = R^{CK}(v_i(t-1), p_i(t-1), g_i(t-1), \xi(t), \alpha, \Omega)$$

The specific form of R^{CK} is a second order difference equation with multiplicative stochasticity:

$$\begin{aligned} y_i(t) &= \alpha_0(x_i(t-1) - x_i(t-2)) \\ &\quad + \alpha_1\xi_1(t-1)(p_i(t-1) - x_i(t-1)) \\ &\quad + \alpha_2\xi_1(t-1)(g_i(t-1) - x_i(t-1)) \end{aligned} \tag{3}$$

$$x_i(t) = \begin{cases} y_i(t) \text{ if } y_i(t) \in \Omega \\ \widehat{\partial\Omega}(x_i(t-1), y_i(t)) \text{ otherwise} \end{cases} \tag{4}$$

where $\xi_{1,2}(t-1)$ are D-dimensional vectors of random numbers with the uniform distribution $U(0,1)$ and the Hadamard product is understood for products of vectors of equal size (as occurs in products such as $\xi_1 p_i$). The update rule is applied to each component x_{id} of x_i. The boundary operator $\widehat{\partial\Omega}(x_i(t-1), y_i(t))$ maps $y \notin \Omega$ to Ω[5].

It is possible that $p_i(t-1) = g_i(t-1)$ (this can happen for open or closed neighbourhoods). A further rule might be applied in this case. In synchronous updating, x_i is computed for every particle in turn, and then the memory updates are performed. In asynchronous updating, the memory updates occur immediately after each position update.

[4] The parameters are fixed in some versions of standard PSO $\alpha(t) = \alpha$; they might however have an explicit time dependence in order to promote convergence by restricting the range of probable trials (as in the linearly decreasing inertia weight PSO's [16]).

[5] For example, $\widehat{\partial\Omega}(x, y)$ projects y to the nearest point x on the boundary $\partial\Omega$

III. Bare Bones

A. The original bare bones

It is known that particle positions in a stagnant PSO swarm (one in which the memories do not change) will converge to a weighted average of their personal best and neighbourhood best positions for certain parameter values [17], [3]. However, apart from the behaviour at stagnation, further analysis is difficult due to the nature of the coupled stochastic difference equation, Eq. 3. In particular, the effective particle distribution is unknown for any function (although it is known that bursts are typical [7]).

In an attempt to understand the particle distribution, a modified first order algorithm was proposed [10]. This algorithm, BB, defines a search focus, μ and a search spread, σ:

$$\mu_{id} = \frac{1}{2}(g_{id} + p_{id}) \tag{5}$$

$$\sigma_{id} = |g_{id} - p_{id}| \tag{6}$$

and draws a trial position x_{id} from the normal distribution $N(\mu_{id}, \sigma_{id})$. Kennedy experimented with applying Eqs 5 and 6 with probability $1/2$ and otherwise $x_{id} = p_{id}$. He also added probabilistic bursts of outliers in an attempt to model the bursts that are known to occur in conventional PSO. Note that the BB algorithm does not use particle position x_i unless the particle is at its personal best, $x_i = p_i$. In a sense, one could say that the particle tries a position x_i, and if it is not successful, returns to it personal best, p_i. The algorithm, at least formally, balances cognitive (p) and social elements (g).

B. The general idea

The key idea behind bare bones PSO is to exchange the stochastic second-order difference particle update with distribution sampling based on first-order quantities[6].

The basic philosophy behind the bare bones approach is to pick a monomodal distribution such that the focus, μ, and the spread or dispersion, σ, of the distribution are given by the parameters, α, and first order quantities $x_i(t-1), p_i(t-1)$ and $g_i(t-1)$. (The more general terms focus and spread are used rather than mean and variance since some distributions (e.g. the α-stable family) do not have defined or finite means and/or variances.)

For simplicity the bare bones PSO particle update rule with a truncated normal distribution (although heavy tailed distributions such as the Cauchy and Lévy distributions have also been used), will be described.

The truncated normal distribution $N(\mu, \sigma^2, L, R)$ has density

$$\phi_{\mu,\sigma^2,L,R}(x) = \frac{\phi_{\mu,\sigma^2}(x)}{\Phi_{\mu,\sigma^2}(R) - \Phi_{\mu,\sigma^2}(L)}.$$

where $\Omega_d = [L, R]$ and $\phi_{\mu,\sigma^2}(x), \Phi_{\mu,\sigma^2}(x) = \int_{-\infty}^{x} \phi_{\mu,\sigma^2}(x')dx'$ are the density and cumulative distribution respectively.

[6]In fact both formulations can be written as stochastic difference equations or as distribution sampling; there is only a difference in emphasis.

The bare bones particle update rule is simply

$$x_i(t) \sim N(\mu_i(t-1), \sigma_i^2(t-1), L, R) \tag{7}$$

where x_i, N, μ_i and σ_i^2 are D-dimensional vectors and $N(\mu, \sigma, L, R)$ denotes the truncated normal distribution. In common with the standard PSO approach, each component of x is updated separately. There is some flexibility in how μ and σ are determined and indeed the choice of these functions effectively specifies a particular bare bones algorithm.

The bare bones particle update can be written as a stochastic difference equation:

$$x_i(t) = \mu_i(t-1) + \sigma_i(t-1)n_{L,R}(t-1) \tag{8}$$

where $n_{L,R}$ is a D-dimensional vector of random numbers from the truncated standard normal distribution $N(0, 1, L, R)$. The standard PSO update, as a stochastic difference equation is given by Eq. 3. Bare bones PSO derives its simplicity from imposing that μ, σ are first order functions of $x_i(t-1), p_i(t-1)$ and $g_i(t-1)$.

Alternatively, the update rule for standard PSO can be written as distribution sampling. The PSO update rule, Eq. 3, can be simplified:

$$\begin{aligned} y_i(t) \sim\ & a_0(t-1) - a_0(t-2) \\ & + U(0, a_1(t-1)) + U(0, a_2(t-1)) \end{aligned}$$

with, for $t > 2$,

$$\begin{cases} a_0(t-1) & = \alpha_0 x_i(t-1) \\ a_0(t-2) & = \alpha_0 x_i(t-2) \\ a_1(t-1) & = \alpha_1(p_i(t-1) - x_i(t-1)) \\ a_2(t-1) & = \alpha_2(g_i(t-1) - x_i(t-1)). \end{cases}$$

The distribution of the random variable $Z = U(0, a_1) + U(0, a_2)$ is trapezoidal. If X, Y are independent random variables then the density of the summed distribution $X + Y$ is

$$f_Z = \int_{-\infty}^{\infty} f_X(y) f_X(z - y)dy.$$

Putting $X = U(0, a_1)$ and $Y = U(0, a_2)$ and, without loss of generality, defining $a_1 \leq a_2$, gives the density:

$$\begin{aligned} f_Z &= \int_{-\infty}^{\infty} f_X(y) f_X(z - y)dy \\ &= \begin{cases} \frac{z}{a_1 a_2} & 0 \leq z \leq a_1 \\ \frac{1}{a_2} & a_1 < z \leq a_2 \\ \frac{1}{a_1} + \frac{1}{a_2} - \frac{z}{a_1 a_2} & a_2 < z \leq a_1 + a_2. \end{cases} \end{aligned}$$

Writing the distribution with the above density as $T(a_1, a_2)$ and its truncated counterpart as $T(a_1, a_2, L, R)$, the standard PSO particle update rule in distribution sampling form is

$$x_i(t) \sim a_0(t-1) - a_0(t-2) + T(a_1(t-1), a_2(t-1), L, R).$$

and second order effects enter via the shift $a_0(t-1) - a_0(t-2)$. This can be compared with the bare bones sampling rule, Eq. 7.

C. Social bare bones with jumps

The original bare bones algorithm is generalised and extended in [14]. The search focus μ and spread σ are chosen from separate neighbourhoods which might each be local or global. In principle a particle may participate in two neighbourhoods, one for the determination of its search focus (the μ-neighbourhood) and one for the determination of the search spread around this focus (the σ-neighbourhood)[7]. Furthermore, it is pointed out in this paper that Eq. 6 contains a hidden positive scaling parameter α

$$\sigma_{id} = \alpha \left| g_{id} - p_{id} \right|, \tag{9}$$

implicitly set to unity in BB. In the same study, a critical value, $\alpha_c = 0.65$, was found such that for $\alpha > \alpha_c$ the swarm resists collapse when optimising the sphere function, or indeed when optimising any local symmetric optimum. Fastest convergence occurs at the critical value, but larger values promote exploration. The swarm collapses for smaller values and optimisation ceases. Open σ-neighbourhoods (i.e. a neighbourhood without self) should be used in order to mitigate against a zero variance when $p_i = g_i$ [14] (but note that any two members of any neighbourhood might share the same position).

The global σ-neighbourhood offers no particular advantage on a standard test set of 30 dimensional problems and the open ring neighbourhood was recommended: $\sigma_i = \alpha \left| p_{i\ominus 1} - p_{i\oplus 1} \right|$. The search focus is determined by a global or local μ-neighbourhood, $\mu_i = g_i$ where the neighbourhood best g_i is given by Eq. 2. The resulting update rule is, for local ring neighbourhoods,

$$\mu_i = g_i \qquad (N_i^G \text{ or } N_i^R) \tag{10}$$

$$\sigma_i = \alpha \left| p_j - p_k \right| \qquad (N_i^R) \tag{11}$$

Finally, the algorithm of [14] provided component jumping. A particle may jump in any component with probability p_J. This can be viewed as a partial re-initialisation (since in general not every component undergoes a jump) or, alternatively, as a tail broadening mechanism (i.e. the tails of the Gaussian distribution), allowing further search in areas where the Gaussian distribution tails are thin. As reported in [14], investigations with a standard test set of 30D problems propose that the jump probability p_J should be set to 0.01.

μ_i and σ_i, as determined by Eqs. 10 and 11 will only involve a particle's own memory, p_i, in the case of a global and closed μ-neighbourhood and if the particle happens to be the best particle in this μ-neighbourhood. The update rule is predominantly socially determined and hence the algorithm will be termed 'social BBJ' (sBBJ) in this paper.

D. Cognitive bare bones with jumps

An unsuccessful particle position (i.e. one that does not better p_i), plays no part in the update equations of BB and social BBJ. However the role of all (even unsuccessful) particle positions was re-introduced in a subsequent extension of the social BBJ algorithm [15]. In this study, the difference between the neighbourhood best and the current position is utilised:

$$\sigma_i = \alpha \left| g_i - x_i \right|. \tag{12}$$

The underlying assumption of this approach is that this alteration might offer a wider search capability since unsuccessful trial positions x lying far from the focus μ will increase the sampling variance of following trials. If the swarm diversity is indeed increased by this measure, we might expect that a smaller jump probability will be needed and indeed in [15] a jump probability of only 0.001 was found to give good performance and to outperform social BBJ and the Clerc-Kennedy PSO for a modest test set in 30D.

In contradistinction to the social BBJ σ-update, Eq. 11, Eq. 12 has a cognitive element as manifest by the presence of the individual position x. This particular version of the jumping bare bones is therefore referred to as 'cognitive BBJ' (cBBJ).

The abbreviation BBJ will refer to the general bare bones scheme (separate μ and σ neighbourhoods and component-wise position jumps). sBBJ and cBBJ are then specific instances of BBJ.

E. Formal comparison of the algorithms

The cBBJ algorithm in the global μ and σ neighbourhoods is formally very simple; it can be summarised as

$$x_i(t) \sim U(\Omega) \text{ or } g + \alpha |g - x_i(t-1)| N(0,1)$$
$$g = BETTER(x_i(t), g)).$$

Each particle searches around the global or local best position g with a search spread determined by the separation between that particle and g. If a particle finds a better position than g then that position becomes the new neighbourhood best and this information is instantly communicated to all particles in the neighbourhood. cBBJ particles are forgetful: individual history, as manifest in the p_i's, plays no role. This is in contrast to the more complicated CK, BB and sBBJ algorithms in which particles retain a memory of past positions and communicate this knowledge via a social network.

BB and social BBJ algorithms, unlike standard PSO's, are distinguished by the absence of particle position information in the update rule. Search is focussed on the neighbourhood best position g_i, and the extent of the search is determined by informer separation, $|g_{id} - p_{id}|$ or $|p_{i-1,d} - p_{i+1,d}|$. A trial position is ignored if an informer p_i is not bettered. The particle, figuratively speaking, returns to p_i after a single trial at search centre g_i. On the other hand, cognitive BBJ retains information of an unsuccessful attempt since search spread is determined by the difference between x_i and g_i and in this sense is reminiscent of the dynamics of standard PSO.

[7] Particles might be connected by more than one network and these networks might be mutually exclusive. For example, in the human domain, consider a network of friends and a network of work colleagues.

Algorithm 1 Social BBJ, $k = 1$, local σ-neighbourhood

$d_J = -1$
`if` $(u \in U(0,1) < p_P)$
  $d_J \sim U(\{1, 2, \ldots, D\})$
`for` $d = 1$ `to` D
  `if` $(d = d_J)$
    $x_{id} \sim U(\Omega_d)$
  `else`
    $\sigma_{id} = \alpha |p_{i\oplus 1d} - p_{i\ominus 1d}|$ (N_i^R)
    $x_{id} \sim g_{id}^{\mu} + \sigma_{id} N(0, 1, L, R)$ $(N_i^G$ `or` $N_i^R)$

Algorithm 2 Cognitive BBJ, $k = 1$

$d_J = -1$
`if` $(u \in U(0,1) < p_P)$
  $d_J = U(\{1, 2, \ldots, D\})$
`for` $d = 1$ `to` D
  `if` $(d = d_J)$
    $x_{id} \sim U(\Omega_d)$
  `else`
    $\sigma_{id} = \alpha |g_{id}^{\sigma} - x_{id}|$ $(N_i^G$ `or` $N_i^R)$
    $x_{id} \sim g_{id}^{\mu} + \sigma_{id} N(0, 1, L, R)$ $(N_i^G$ `or` $N_i^R)$

F. Fixed number of jumps

The probability $P = P(k, D, p_J)$ that a BBJ particle jumps in k components in D dimensions is given by the binomial theorem,

$$P(k, D, p_J) \quad = \quad \binom{D}{k} p_J^k \times (1 - p_J)^{D-k}. \qquad (13)$$

The one jump probabilities in 30 dimensions for $p_J = 0.01$ and $p_J = 0.001$ are $P(1, 30, 0.01) = 0.224$ and $P(1, 30, 0.001) = 0.029$ and the probability that a particle jumps in two or more components[8] in 30D is approximately 0.036 or 0.0004 for the same single component probabilities p_J. A particle, therefore, if it jumps at all, will likely jump in one component only.

It is not clear that this component-by-component probabilistic jump represents the correct scaling with dimensionality. Although the probability p_P that a particle jumps at all is a function of D and p_J ($p_P = 1 - P(0, D, p_J)$) and so can be controlled, the relative proportion of single component, two component, three component jumps etc. is fixed by the binomial theorem and changes with dimension.

This paper proposes a more direct scheme: a particles is given a fixed probability p_P of jumping and a fixed number of jumped components k. If it decides to jump, then it does so in k components chosen uniformly at random from the available D components. This fixed-jumps scheme frees the dependency between the particle-wise jump probability (p_P) and the number of jumping components (k).

Possible algorithms for the case $k = 1$ are outlined in Algorithms 1 and 2. Here, $g_i^{\mu,\sigma}$, the μ, σ neighbourhood best is obtained from Eq. 2 either synchronously (at the start of the iteration over the swarm), or asynchronously (before each particle update). The specification describes a local ring σ-neighbourhood and either type of μ-neighbourhood for the g_i determination; the type of network is placed in parentheses after the σ and μ update step. $U(\{1, 2, \ldots, D\})$ indicates a uniform random choice of index and $U(\Omega_d)$ is the uniform distribution on the interval Ω_d.

[8]From the binomial theorem, the required probability is $1 - P(0, D, p_J) - P(1, D, p_J) = 1 - (1 - p_J)^D - Dp_J(1 - p_J)^{D-1}$.

IV. Analysis

The main purpose of this section is to provide a bound to α, the parameter that controls the variance of gaussian sampling, for cBBJ from a stagnation analysis, and to produce an estimate of the jumping probability for cBBJ based on the idea that jumps, which are primarily exploratory, should not hinder local optimisation. The section begins with a demonstration of tail broadening as a consequence of the hybrid truncated normal with jumps probability distribution.

A. Tail broadening

The hybrid distribution for $x \in \Omega = [L, R]$ is

$$\phi_{\mu,\sigma^2,p_J}(x, L, R) \equiv \\ p_J \frac{1}{R - L} + (1 - p_J)\phi_{\mu,\sigma^2}(x, L, R)$$

where

$$\phi_{\mu,\sigma^2}(x, L, R) \equiv \frac{\phi_{\mu,\sigma^2}(x)}{\Phi_{\mu,\sigma^2}(R) - \Phi_{\mu,\sigma^2}(L)}.$$

and $\phi_{\mu,\sigma^2}(x)$ and $\Phi_{\mu,\sigma^2}(x)$ are respectively the density and cumulative distribution of the normal distribution $N(\mu, \sigma^2)$. Figure 1 shows the tail probability

$$\begin{aligned} \text{prob}(|x| > y) &= 2 \int_y^1 \phi_{0,0.1^2,p_J}(x', -1, 1) dx' \\ &\equiv T_{p_J}(y) \end{aligned}$$

for $\mu = 0$ and $\sigma = 0.1$ at $p_J = 0.1,\ 0.01$ and 0.001. For comparison, the tail probability for $p_J = 0$ i.e. for the truncated normal without jumps, and the pure jumps case, $p_J = 1$ are also depicted. Jumping permits uniform exploration through a k-dimensional subspace and leads to broadening of the gaussian tails (see also [14]).

B. Stagnation

1) Stagnation analysis of cBBJ: The update rule per component is

$$\begin{aligned} x(t + 1) &= g + \alpha |g - x(t)| \eta(t) \\ &= \alpha | - x(t)| \eta(t) \quad (g = 0) \\ &\equiv \alpha x(t) \eta(t) \end{aligned}$$

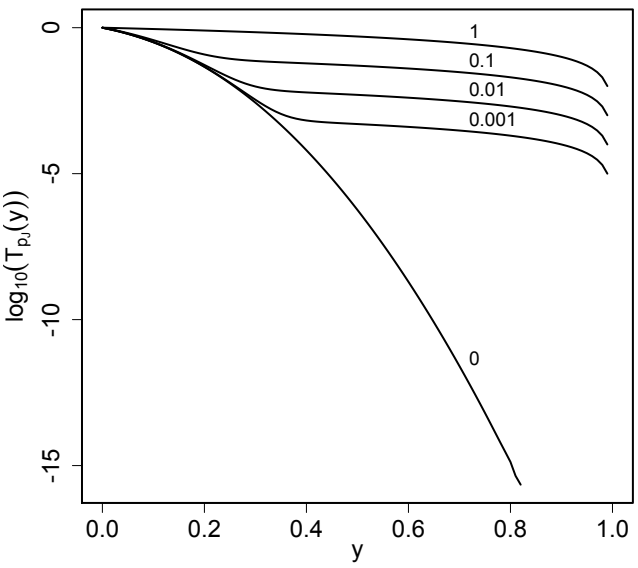


Fig. 1. Tail probabilities of the normal-with-jumps distribution for various jump probabilities. The curves depict the probability of selecting a point in the distribution tails, defined here as $|x| > y$, for $\mu = 0$ and $\sigma = 0.1$. The case $p_J = 0$ is pure truncated normal, and at $p_J = 1$ the distribution is uniform within Ω. For intermediate p_J, and for $x > 3\sigma$, the contribution from the normal is negligible and T follows a linear relationship.

where we have placed g at 0, η is a Gaussian random variable, $\eta \sim N(0,1)$, and in the last line, the symmetry of $N(0,1)$ has been employed to remove the absolute value. For simplicity, the effects of truncation to Ω have been ignored and assume a jump-free period. The rule is simply iterated at stagnation (g does not change):

$$\frac{x(t+1)}{x(1)} = \alpha^t \prod_{s=1}^{t} \eta(s) \equiv y(t+1).$$

The expected value of y (and therefore of x) at any update is zero since

$$\mathbb{E}y(t+1) = \alpha^t \mathbb{E} \prod_{s=1}^{t} \eta(s) = 0.$$

is 0 (= g). However, convergence requires that the variance shrinks to zero, **var** $x \to 0$. Since

$$\begin{aligned} \textbf{var}\ y(t+1) &= \textbf{var}\ (\alpha\eta)^t \\ &= \alpha^{2t}, \end{aligned}$$

convergence requires that $|\alpha| < 1$.

The speed of convergence is exponential. This can be demonstrated by considering the statistics of the separation $|x|$ of x from g. Since

$$|y(t+1)| = \alpha^t |\eta(t)||\eta(t-1)| \dots |\eta(1)|$$

and

$$\mathbb{E}|\eta| = 2\int_0^\infty x e^{-\frac{x^2}{2}} \frac{dx}{\sqrt{2\pi}} = \sqrt{2/\pi},$$

the expectation of $|y(t+1)|$ is

$$\begin{aligned} \mathbb{E}|y(t+1)| &= \alpha^t (\mathbb{E}|\eta|)^t \\ &= \left(\alpha\sqrt{2/\pi}\right)^t. \end{aligned}$$

With α set to the critical value 0.65,

$$\mathbb{E}|x(t+1)| \approx 0.52^t |x(1)|$$

which demonstrates the very fast (exponential) convergence to g in the case of no jumps.

However an eventual jump will occur (if $p_J > 0$), and breaks the jump-free period. Suppose a jump happens at t. Then $x(t) \sim U(\Omega)$ and the exponentially converging search starts anew from a uniform random position on Ω. This pattern – exponential convergence punctuated by restarts – persists at stagnation.

2) Stagnation analysis of sBBJ: Stagnation corresponds to $|p_{left} - p_{right}| = \text{const} \equiv \delta$, so again for $g = 0$, $x(t) = \alpha\delta\eta(t)$: sBBJ continues to sample, in each component, around the focus with a constant variance, $\alpha^2\delta^2$. Component jumps do not change the fixed variance normal sampling around g since, by definition, the memory cannot change at stagnation.

C. Jump probability

Consider the following scenario: a swarm with all best memories and positions constrained to a small region $\delta\Omega$ and optimising a local optimum in $\delta\Omega$. Suppose that $|\delta\Omega| \ll |\Omega|$. A sBBJ particle jumps with probability p_j to a position x that is very likely outside $\delta\Omega$. If x does not better the current search focus g, the trial has been wasted, but gaussian sampling around g with $\sigma \sim |\delta\Omega|$ resumes on the next iteration. So at the worst, one iteration in every $1/p_J^{sBBj}$ is redundant.

On the other hand, suppose that a cBBJ particle jumps to a position $x \sim U(\Omega)$. Then, with high probability, $\sigma \equiv \alpha|x - g| = O(|\Omega|)$. Subsequent gaussian updates will, in the absence of further jumps, bring x back to $\delta\Omega$, with the variance decreasing by a factor of $\alpha\sqrt{2/\pi}$ at each step. Suppose it takes T trials to bring x back to $\delta\Omega$. In the worst case, optimisation resumes when x re-enters $\delta\Omega$. If $cBBJ$ and $sBBJ$ swarms are comparable optimisers of a local optimum, and since $1/p_J^{sBBJ}$ trials can be wasted without prejudice, then $p_J^{sBBJ} \approx T p_J^{cBBJ}$. This analysis therefore suggest the following estimate:

$$\frac{|\delta\Omega|}{|\Omega|} = \left(\alpha\sqrt{\frac{2}{\pi}}\right)^{\max\left(\frac{p_J^{sBBJ}}{p_J^{cBBJ}},\ 0\right)}$$

and taking $\alpha\sqrt{\frac{2}{\pi}} \approx 0.5$, p_J^{cBBJ} is given by

$$p_J^{cBBJ} = \frac{p_J^{sBBJ}}{1 - \log_2 \frac{|\delta\Omega|}{|\Omega|}}. \tag{14}$$

Although $|\delta\Omega|$ is a dynamic quantity, we can estimate an upper bound for p_J^{cBBJ} by an upper bound for $|\delta\Omega/\Omega|$. Since $|\delta\Omega|$ is a small region around a local optimum, it is reasonable to suppose that $|\delta\Omega|$ is smaller than the distance between optima,

a quantity that will depend on the specific objective function and on the coordinate axis. In the case of the Rastrigin function, $f(x) = x^2 - 10\cos(2\pi x),\ \Omega = [-5.12, 5.12]$ and optima are separated by approximately 1 unit, so by this reasoning, $|\delta\Omega|/|\Omega| < 0.1$. Two other popular functions, Ackley and Griewank, also, by this criterion, satisfy $|\delta\Omega|/|\Omega| < 0.1$. Eq. 14 provides an upper estimate, with $p_J^{sBBJ} = 0.01$,

$$p_j^{cBBJ} < 2.3 \times 10^{-3}.$$

We can find a lower bound on p_J, for either BBJ version, from the following argument. At stagnation, $\mathbf{var}\ y(t) = \alpha^{2(t-1)}$. No further sampling is possible when $\mathbf{var}\ y = 0$, or with finite precision arithmetic when $\mathbf{var}\ y$ is equal to the smallest floating point number on the machine. The smallest IEEE floating point number is 2^{-1074}. Solving $\alpha^{2t} \geq 2^{-1074}$ for $\alpha = 0.65$ gives $t \leq 864$. Jumping should, at the least, break arithmetic stagnation, so

$$p_J > \frac{1}{864} \approx 1.2 \times 10^{-3}.$$

Therefore, if jumps should indeed not hinder local optimisation, this analysis suggests that p_J^{cBBJ} should be placed in the range $1 - 2 \times 10^{-3}$ for comparable performance to sBBJ when sBBJ is running at $p_J^{sBBJ} = 0.01$. However the analysis depends on an upper bound estimate to the dynamic and function dependent quantity $|\delta\Omega|/|\Omega|$ and the upper bound on p_J^{cBBJ} might be underestimated.

The smaller value of p_J for cBBJ can also be understood in terms of tail broadening. The outliers resulting from a cBBJ particle jump lead to a broader search at subsequent iterations since search spread is determined by $\alpha|g_{id} - x_{id}|$. The availability of this mechanism might be compensated by a decreased amount of tail broadening (i.e. smaller p_J) in cBBJ as compared to sBBJ.

V. Investigation of jumps

A. Methodology

The experiments detailed here use a set of test functions designed for the Special Session on Real Parameter Optimization organised in the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), reported in [18], where a complete description of these benchmarks and details of the various initialisation conditions for each function is given. The functions are:

- Unimodal Functions (5):
  - F_1: Shifted Sphere Function
  - F_2: Shifted Schwefel's Problem 1.2
  - F_3: Shifted Rotated High Conditioned Elliptic Function
  - F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness
  - F_5: Schwefel's Problem 2.6 with Global Optimum on Bounds
- Multimodal Functions (9):
  - Basic Functions (7):
    - F_6: Shifted Rosenbrock's Function
    - F_7: Shifted Rotated Griewank's Function without Bounds
    - F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds
    - F_9: Shifted Rastrigin's Function
    - F_{10}: Shifted Rotated Rastrigin's Function
    - F_{11}: Shifted Rotated Weierstrass Function
    - F_{12}: Schwefel's Problem 2.13
  - Expanded Functions (2):
    - F_{13}: Expanded Extended Griewank's plus Rosenbrock's Function (F_8F_2)
    - F_{14}: Shifted Rotated Expanded Scaffer's F_6

All benchmarks have been shifted in order to ensure there are no optima in the centre of the search space. This study omits the unusual CEC2005 hybrid composition functions but includes unimodal, multimodal, separable, non-separable and highly conditioned functions, as well as functions with noise and with the global optimum on the boundary.

The dimensionality of the problems is 30 and the termination criterion for this experiment was either to reach the optimum (with function errors less than 10^{-9}) or to exceed $300,000$ function evaluations (FEs). There were 50 Monte Carlo trials for each test and the results were averaged over these independent trials. A particle that moved outside the search space was not evaluated on that iteration[9].

Parameter settings. There are no adjustable parameters (apart from swarm size) in Kennedy's Bare Bones. The constriction coefficient and acceleration constants of the CK PSO were set to $\chi = 0.72984$ and $c_1' = c_2' = 2.05$, as recommended by [2]. The scaling parameter α was fixed at the critical value, 0.65 as derived by [14]. The default particle jump probability p_P was $P(1, 30, 0.01) = 0.224$ for sBBJ and $P(1, 30, 0.001) = 0.029$ for cBBJ.

Following [14], the local ring σ-neighbourhood was used for both sBBJ μ-neighbourhoods because there is no particular advantage in a global σ-neighbourhood. However the global and local μ-neighbourhoods were paired with global and local σ-neighbourhoods in the cognitive BBJ trials in order to agree with the models studied in [15].

In order to conduct the statistical analysis measuring the presence of any significant difference in the performance of the algorithms, Wilcoxon 1×1 non-parametric statistical test is deployed. The performance measures used in this paper are error, efficiency and reliability.

Error is the absolute difference between the best solution found during a run and the known global optimum. Since the runs are terminated if the error falls below 10^{-9}, a mean error of 10^{-9} or less indicates that all runs converged.

Reliability is the percentage of trials reaching a specified error and *Efficiency* is the number of function evaluations before that error. These performance measures are defined:

$$\text{Error} \quad = \quad \frac{1}{n}\sum_{i=1}^{n} |f(g) - f(x_{opt})| \qquad (15)$$

[9] A review of bounds handling for standard PSO has been carried out by the authors of [19]. Here we adopt the commonly used *infinity* method: there is, however, an inherent danger with *infinity*, especially in high dimensions, that stray particles may never return. The BBJ search focus, however, is always inside the bounds and hence the probability of sampling within the search space, which may be quite small, is, however, always finite.

$$\text{EFFICIENCY} = \frac{1}{n'}\sum_{i=1}^{n'} FEs \quad (16)$$

$$\text{RELIABILITY} = \frac{n'}{n} \times 100 \quad (17)$$

where g is the best position found during a run and $f(x_{opt})$ is the value at the global minimiser(s) x_{opt}; n is the number of runs in the experiment, n' is the number of successful runs and *FEs* is the number of function evaluations before reaching the specified error.

B. Impact of jumping in more than one component

As stated earlier, one of the main motivations behind the fixed jump versions of bare bones with jumps is to study the impact of jumps when applied to one or more components of each particle.

cBBJ in local and global μ-neighbourhoods was tested for numbers of component jumps $k = \{0, 1, 5, 10, 15, 20, 25, 30\}$ according to the experimental methodology of subsection V-A. The per-particle jump probability was 0.027. The results are shown in Tables I and II and Fig. 2.

The tables indicate that global cBBJ is a better optimiser over the test set and that the algorithms are, in the main, fairly insensitive to k. Since about one particle in the swarm of 50 particles jumps per iteration ($p_P = 0.027$), the k insensitivity shows that in the main (i.e. for the majority of the test functions) the swarm is not disrupted by a single jumping particle. However, Fig 2 shows dips for a few functions at $k = 1$. In particular, global-μ cBBJ performs much better on Rosenbrock (f_6) and Rastrigin (f_9) when just one component is randomised.

C. Sensitivity of jump probability on cBBJ performance

Given the relative insensitivity to number of jumped components at a low per-particle jump probability, it might be wondered if more frequent jumping might boost or diminish performance.

An investigation on the impact of p_P on the performance of sBBJ and cBBJ at $k = 1$ (chosen because it imparts a small advantage on this test set) in both global and local neighbourhoods was undertaken. In order to cover a large spectrum of the possibilities, sixteen different probabilities in a geometric progression (Table III) were considered. The probabilities were chosen logarithmically according to

$$\begin{cases} P00 = 0 \text{ (no jumps at all)} \\ P15 = 1 \text{ (a jump is certain)} \\ Pn = \frac{0.9}{2^{14-n}} \text{ for } 1 \leq n \leq 14 \end{cases}$$

As before, 50 Monte Carlo simulations for each benchmark function over each p_P value were performed. The generated plots in Figs 3 and 4 show the performance of the sBBJ and cBBJ algorithms averaged over each of the p_P values in both global and local neighbourhoods.

The sBBJ plots (in both global and specially local neighbourhood) do not show great variation with regard to p_P in the majority of the benchmarks. In some cases, the performance is not diminished even if every particle jumps in one component.

On the other hand, the cBBJ plots show more sensitivity towards various p_P in local and especially global neighbourhoods, with the optimum value of p_P, if such an optimum exists, mostly in the interval $p_7 - p_{11}$ (i.e. $7 - 113 \times 10^{-3}$). Although the optimum jump probability changes from problem to problem, the value of 0.029 reported in [15] is confirmed to be a good compromise[10] and supports the hypothesis (Sec. IV-A) that a smaller probability of jumping is balanced by the increased exploration offered by the cBBJ update rule. The results also support the analysis of Sec. IV-C which, after conversion from p_J to p_P at $k = 1$ by Eq. 13, placed p_P^{cBBJ} in the range $0.035 - 0.065$.

An sBBJ swarm can tolerate jumping fairly well, often not minding if every particle jumps in one component. This might be explained by the lack of any knock-on effect. Unsuccessful particle trials do not disrupt the sBBJ swarm since the algorithm is only sensitive to successful positions (i.e. those that better a personal best). But all trials do affect cBBJ performance and hence it is reasonable to assume that it would be more sensitive to jumping, and this hypothesis is supported by these results.

D. Immediate effects of jumping during the optimisation

In order to investigate the immediate effects of one-component jumps on cBBJ, all jumps are counted during a set of trials (on an exemplary number of uni-modal and multi-modal benchmarks) and the ratio of the jumps immediately improving the personal and neighbourhood best is detailed in Table IV.

The plots in Fig. 5 depict the occurrence of successful jumps. The improving jumps occur mainly at the earlier stages of the optimisation. Once a potentially good area of the search

TABLE III
APPROXIMATE p_P VALUES

This table shows approximate p_P values from P01 – P14 and the expected number of jumps per iteration of a 50 particle swarm.

	$p_P \times 10^{-3}$	No. of Jumps In Each Iteration
P00	0	0
P01	0.11	0.0055
P02	0.22	0.011
P03	0.44	0.022
P04	0.9	0.044
P05	1.8	0.088
P06	3.5	0.18
P07	7	0.35
P08	14	0.70
P09	28	1.4
P10	56	2.8
P11	113	5.6
P12	225	11
P13	450	22
P14	900	45
P15	1	50

[10]However this study has been at constant k and there is a possibility that optimal p_P has a k-dependence. However, any k-dependence might be expected, on the grounds of the previous variable k experiments, to be weak.

TABLE I
IMPACT OF VARYING THE NUMBER OF COMPONENT JUMPS, k, IN GLOBAL CBBJ.

Top: Mean error; bottom: Function evaluations (FEs) at termination.

Fn	$k=$ 0	1	5	10	15	20	25	30
f_1	1.13E-04	5.85E-10	7.26E-10	7.32E-10	7.22E-10	7.76E-10	8.11E-10	7.95E-10
f_2	7.07E+03	9.79E-10	9.72E-10	9.85E-10	9.78E-10	9.76E-10	9.80E-10	9.80E-10
f_3	1.08E+06	2.45E+05	2.51E+05	2.14E+05	2.07E+05	2.44E+05	2.06E+05	1.87E+05
f_4	1.23E+05	4.64E+04	1.66E+04	1.63E+04	1.57E+04	1.56E+04	1.46E+04	1.30E+04
f_5	2.01E+04	1.11E+04	1.04E+04	1.10E+04	1.15E+04	1.11E+04	1.07E+04	1.04E+04
f_6	9.79E+01	5.46E-03	3.52E-01	4.28E-01	4.56E-01	5.71E-01	6.83E-01	5.79E-01
f_7	2.56E-01	2.62E-02	2.21E-02	2.52E-02	2.38E-02	2.34E-02	1.83E-02	1.90E-02
f_8	2.01E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01	2.00E+01
f_9	3.02E+02	5.48E-10	1.49E+00	3.64E+00	5.14E+00	5.37E+00	7.88E+00	7.22E+00
f_{10}	6.01E+02	5.19E+02	4.12E+02	3.57E+02	3.80E+02	4.07E+02	3.98E+02	4.23E+02
f_{11}	3.58E+01	3.65E+01	3.44E+01	3.40E+01	3.49E+01	3.43E+01	3.45E+01	3.48E+01
f_{12}	2.50E+03	2.33E+03	2.14E+03	2.17E+03	2.03E+03	1.77E+03	2.63E+03	2.28E+03
f_{13}	3.87E+01	1.42E+00	3.13E+00	3.70E+00	4.45E+00	4.62E+00	5.63E+00	5.42E+00
f_{14}	1.40E+01	1.36E+01	1.34E+01	1.33E+01	1.33E+01	1.33E+01	1.33E+01	1.33E+01

Fn	$k=$ 0	1	5	10	15	20	25	30
f_1	179452	12606	11561	11464	11171	11081	10894	10946
f_2	-	95252	85582	83425	83024	82352	81129	82351
f_6	-	298629	-	-	-	-	-	-
f_7	295698	272167	275554	265873	265621	265760	260440	280477
f_9	-	60874	287547	-	299080	-	-	-

TABLE II
IMPACT OF VARYING k FOR LOCAL CBBJ.

Top: Mean error; bottom: Function evaluations (FEs) at termination.

Fn	$k=$ 0	1	5	10	15	20	25	30
f_1	9.40E-10	9.37E-10	9.63E-10	9.54E-10	9.43E-10	9.51E-10	9.48E-10	9.49E-10
f_2	7.64E-04	4.42E-03	6.88E-03	7.60E-03	6.30E-03	6.30E-03	5.36E-03	5.29E-03
f_3	1.75E+06	1.98E+06	2.21E+06	2.20E+06	2.23E+06	2.19E+06	2.24E+06	2.18E+06
f_4	5.19E+04	2.96E+04	1.59E+04	1.09E+04	1.20E+04	1.06E+04	9.61E+03	8.63E+03
f_5	8.33E+03	8.50E+03	7.88E+03	7.83E+03	7.60E+03	8.00E+03	8.16E+03	8.33E+03
f_6	2.99E+01	2.39E+01	5.39E+01	6.42E+01	8.01E+01	4.36E+01	4.87E+01	5.08E+01
f_7	2.49E-02	2.93E-02	2.71E-02	3.69E-02	2.80E-02	2.99E-02	3.29E-02	3.78E-02
f_8	2.01E+01	2.01E+01	2.01E+01	2.01E+01	2.01E+01	2.01E+01	2.01E+01	2.01E+01
f_9	1.75E+02	8.57E-10	1.89E+00	4.30E+00	6.77E+00	6.91E+00	7.49E+00	9.20E+00
f_{10}	3.41E+01	3.28E+01	3.16E+01	3.17E+01	3.23E+01	3.21E+01	3.27E+01	3.25E+01
f_{11}	4.94E+02	4.54E+02	3.29E+02	3.50E+02	3.29E+02	3.49E+02	3.46E+02	3.26E+02
f_{12}	7.46E+03	6.44E+03	7.35E+03	6.31E+03	6.23E+03	5.12E+03	7.72E+03	6.09E+03
f_{13}	1.40E+01	1.25E+00	2.80E+00	3.49E+00	3.96E+00	4.24E+00	4.79E+00	4.16E+00
f_{14}	1.39E+01	1.34E+01	1.33E+01	1.32E+01	1.31E+01	1.32E+01	1.33E+01	1.32E+01

Fn	$k=$ 0	1	5	10	15	20	25	30
f_1	43833	53787	58381	58997	58813	59638	60277	60730
f_7	-	299881	-	-	-	-	-	-
f_9	-	122226	297828	-	-	-	-	-

TABLE IV
IMPROVING JUMPS

This table shows the mean $\pm$ standard deviation number of jumps improving the neighbourhood (personal) best in three exemplary benchmarks, $f_{1,9,13}$, along with the total number of jumps in the aforementioned benchmarks, and the ratios of successful jumps.

	No. improving jumps	All Jumps	Rel. fraction
f_1	4 ± 2 (28 ± 5)	402 ± 47	1.0% (7.0%)
f_9	15 ± 3 (44 ± 8)	2023 ± 843	0.74% (2.2%)
f_{13}	9 ± 3 (32 ± 6)	8744 ± 101	0.10% (0.37%)

space is discovered by the swarm, the number of immediately improving jumps decreases. The jump mechanism apparently enhances the exploration phase of the optimisation. Note that only the immediate improvement of jumps is studied in this section; non-improving jumps may lead to wider search spreads at later iteration.

E. Escape from sub-optimal minima

Another experiment was conducted to observe if jumps enable a particle to escape to a local minimum of the Rastrigin function, f_9. It was found that 75% ($\pm$10%) of the immediately improving jumps moved the best position by a distance greater than one unit, indicating a jump to a new local optimum since this function has an oscillatory period of one unit (see Fig. 6). This provides evidence that only 1 out

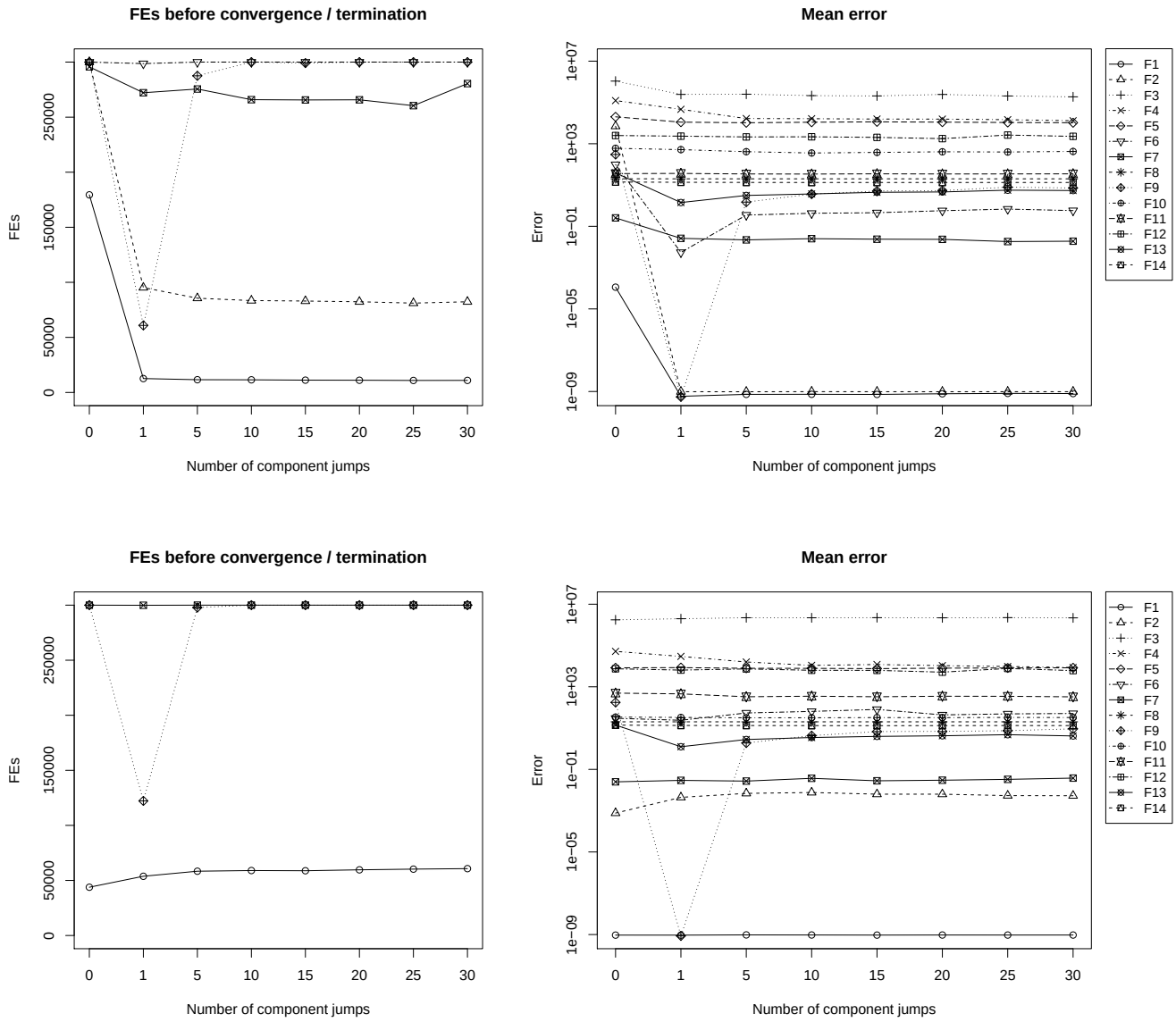


Fig. 2. Impact of varying k for global (top) and local (bottom) cBBJ.

of 4 improving jumps move the particle from one position to a better one within the current optimum, while 3 out of 4 jumps either discover better optima (or possibly discover a better position on a worse or equivalent optimum).

F. Summary

The results of the experiments conducted within this section are summarised below:

1) Fixed jumping in one component only ($k = 1$) appears to be optimal for s/cBBJ at $p_P = 0.224/0.029$
2) Although the optimal value of p_P in cBBJ is problem dependent, a value in the range $0.007 - 0.113$ at $k = 1$ works well and $p_P \approx 0.03$ represents a good compromise

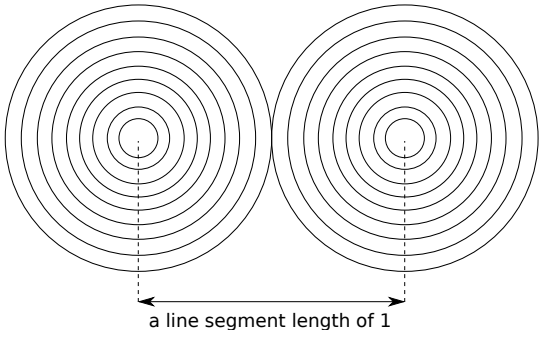


Fig. 6. 2D illustration of Rastrigin's hills.

3) Immediately successful jumps, defined to be those jumps

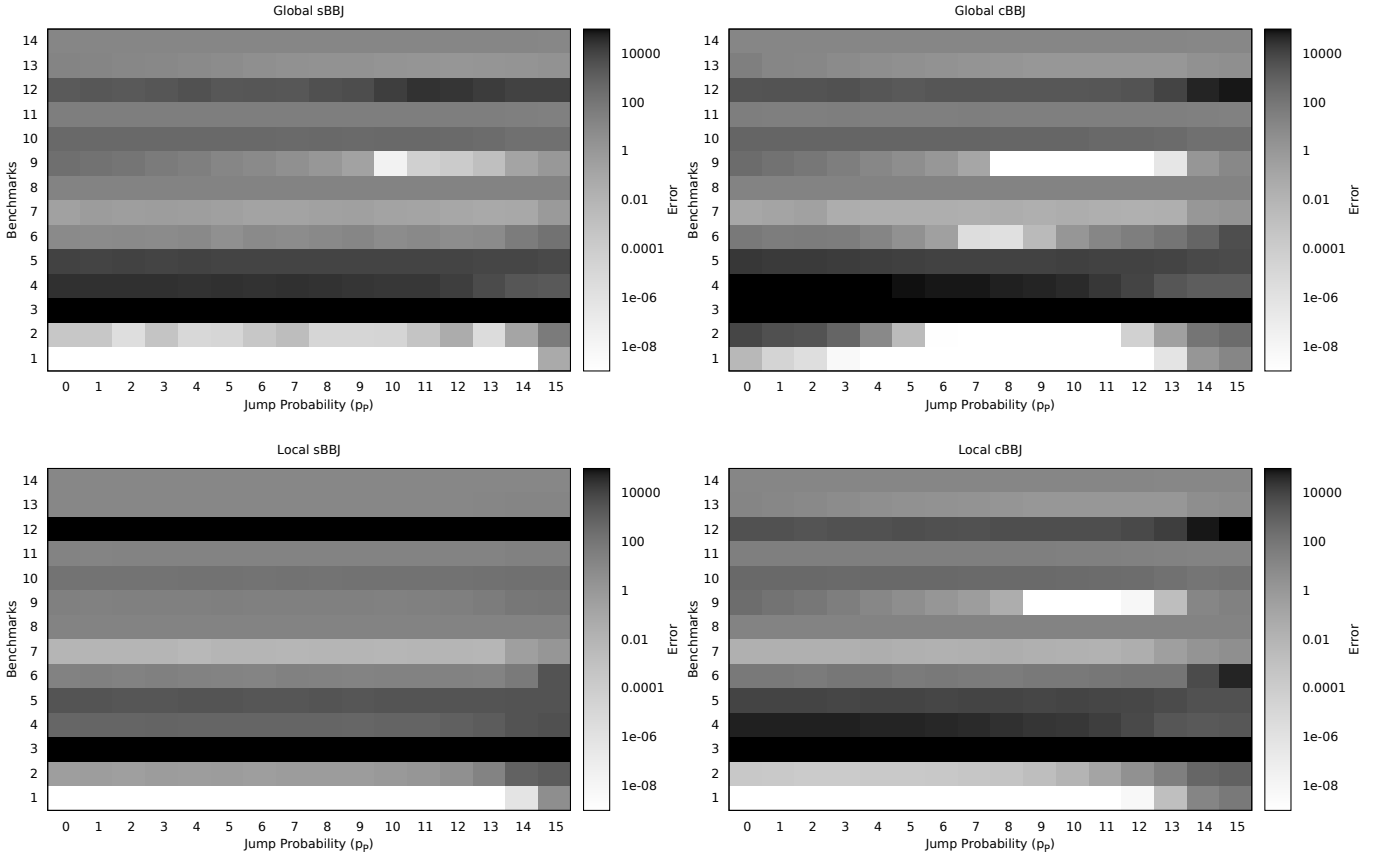


Fig. 3. The plots show the impact of jumps probability (from $p_P = 0$ to $p_P = 1$) on sBBJ and cBBJ in both global and local neighbourhoods. For details on P00 – P15 refer to Table III.

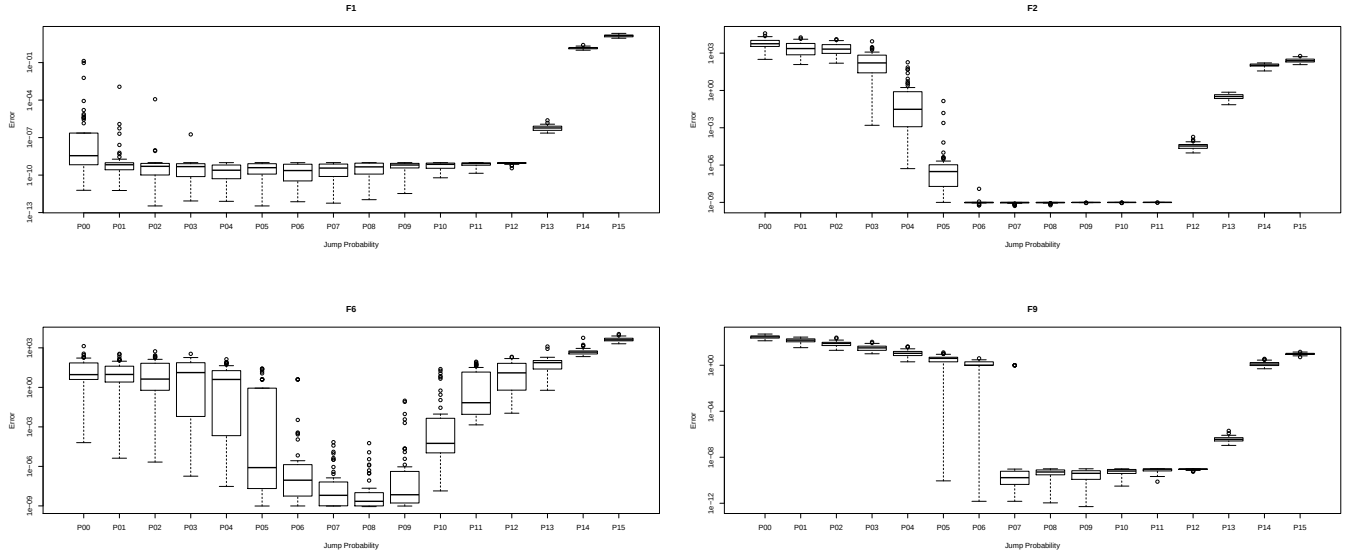
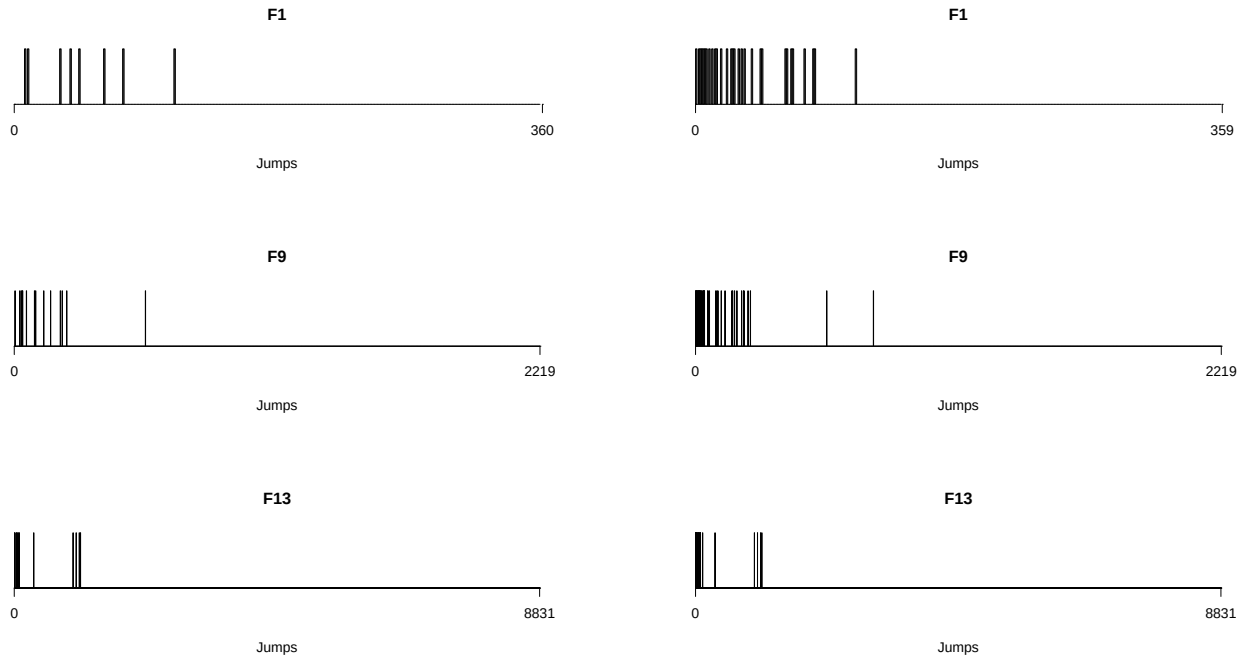


Fig. 4. Box and whisker plots of error versus jump probability for global cBBJ.

that improve a personal best, occur mainly at the early stages of the optimisation; however unsuccessful jumps may still play a role in cBBJ at later stages since a far-flung position will lead to a greater search spread at the subsequent iteration

4) A study on the Rastrigin function suggests that the



(a) g_i (b) p_i

Fig. 5. Jumps of improvement in three exemplary benchmarks ($f_{1,9,13}$). The bar charts in this figure show the behaviour of a randomly chosen trial in either of the aforementioned benchmarks and illustrate the distribution of successful jumps throughout the course of the optimisation. For details on the numeric figures, refer to Table IV.

majority of jumps move a personal best out of the basin of attraction of a local minimum; this confirms the intuition that jumping primarily enhances diversity rather than convergence.

VI. Empirical comparisons

The aim of this set of experiments was to compare the performance of bare bones with fixed jumps, sBBJ and cBBJ, with a canonical PSO (CK), standard bare bones PSO (BB) and with a number of PSO variants. A version of cBBJ without jumps (cBBNJ) was also tested as a control.The experimental set-up follows Sec. V-A.

A. Comparison of sBBJ and cBBJ in global and local μ neighbourhoods with standard bare bones and a standard PSO

The results of the comparative studies are recorded in Tables V and VI. Table V shows the results for global neighbourhoods. sBBJ and cBBJ offer comparable performance to a standard PSO, CK, in the global neighbourhood. cBBJ is noticeably better than the base line bare bones algorithm (BB). The dramatic ameliorating effect of jumps is clear from the comparison between cBBJ and cBBNJ. cBBJ seems to offer a clear advantage to sBBJ in the global μ-neighbourhood.

The performance of either BBJ algorithm relative to local CK falls off in the local μ-neighbourhood (Table VI). This is in accordance with the results of [15]. Local-μ cBBJ again proves better than BB and local-μ sBBJ, and the good effects of jumps are again verified.

The results suggest that cBBJ with per-particle jumps is a better optimiser of the test set than sBBJ with per-particle jumps, and that it performs better in the local μ-neighbourhood. These results are in broad agreement with the per-component trials reported in [15].

B. A comparison of cBBJ and CK in their preferred topologies

The global μ-neighbourhood in standard PSO is not generally favoured because of premature convergence [20]. A comparison was therefore made between local CK and the best BBJ version, global-μ cBBJ. The results, Table VII, for both best error and Wilcoxon testing provide evidence that cBBJ is at least the equal of CK on this test set. Note however that the results reported here have been obtained with 50 particles. It is possible that different swarm sizes are optimal for different algorithms, and that these optimal swarm sizes are a function of dimensionality. Indeed, a version of standard PSO (SPSO 2007) downloadable from Clerc's website [21] uses 20 particles in 30D.

In summary, the comparative study suggests that, for 50 particle swarms on the 14 function CEC 2005 test set,

- jumping and dual neighbourhoods significantly enhance bare bones performance,
- the cognitive version of the per-particle BBJ algorithm has the edge on the social version and
- the best BBJ algorithm (local-μ cBBJ) is more than just a model of PSO behaviour; it is a good optimiser in its own right.

TABLE V
MEAN ERROR IN GLOBAL μ-NEIGHBOURHOOD.

Top: Mean error is shown with two decimal places after 50 trials of 300,000 function evaluations; bottom: based on Wilcoxon 1×1 Non-Parametric Statistical Test, if the difference between each pair of algorithms is significant at the 5% level, the pairs are marked. X $\leftarrow$ o shows that the left algorithm is significantly better than the right one; and o $\rightarrow$ X shows that the right one is significantly better than the left algorithm. n – m in the row labeled Σ is a count of the number of X's in the column above.

Fn	CK	BB	sBBJ	cBBJ	cBBNJ
f_1	9.08E-10	8.01E-10	6.59E-10	5.36E-10	4.70E-03
f_2	9.74E-10	1.60E+03	2.50E-04	9.66E-10	8.38E+03
f_3	6.94E+05	6.52E+06	5.41E+05	2.41E+05	1.08E+06
f_4	9.72E+01	5.42E+03	8.50E+03	5.04E+04	1.30E+05
f_5	4.97E+03	7.94E+03	1.02E+04	1.08E+04	2.09E+04
f_6	1.53E+01	2.51E+02	9.75E+00	1.22E-02	7.55E+01
f_7	1.70E-02	1.32E+01	8.67E-02	1.97E-02	1.09E-01
f_8	2.09E+01	2.10E+01	2.07E+01	2.00E+01	2.01E+01
f_9	8.17E+01	3.67E+01	5.40E-04	5.31E-10	2.96E+02
f_{10}	1.89E+02	1.24E+02	3.21E+02	5.21E+02	6.01E+02
f_{11}	2.87E+01	2.85E+01	3.55E+01	3.61E+01	3.60E+01
f_{12}	3.94E+03	9.19E+04	1.91E+04	1.76E+03	2.93E+03
f_{13}	4.99E+00	3.01E+00	1.53E+00	1.38E+00	3.28E+01
f_{14}	1.28E+01	1.22E+01	1.33E+01	1.37E+01	1.41E+01

Fn	CK-sBBJ	CK-cBBJ	BB-cBBJ	sBBJ-cBBJ	cBBNJ-cBBJ
f_1	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	-	o $\rightarrow$ X
f_2	X $\leftarrow$ o	-	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_3	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_4	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
f_5	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	-	o $\rightarrow$ X
f_6	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_7	X $\leftarrow$ o	-	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_8	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_9	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_{10}	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
f_{11}	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	-	-
f_{12}	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_{13}	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_{14}	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
$\sum$	8 - 6	5 - 7	5 - 9	3 - 8	0 - 13

C. Comparing cBBJ with other PSO's

This section presents a comparison between cBBJ and four competitive PSO variants, namely:

- CLPSO or comprehensive learning particle swarm optimiser [22]. This algorithm uses a novel learning scheme based on $\{\vec{p}_i\}$ to update each particle's velocity. The diversity of the swarm as a whole mitigates against premature convergence. CLPSO has been shown to be more effective than a range of contemporary PSO variants multi modal problems (including rotated test functions).
- DMSPSO or dynamic multi-swarm particle swarm optimiser [23] creates multiple small swarms and a random regrouping strategy with the aim of introducing a dynamically changing neighbourhood structure.
- UPSO or unified PSO [24], [25]. The main aim is to propose a unified scheme via an update equation that combines the features of the canonical local and global PSO's. The result is reportedly a very competitive PSO.
- FIPS or fully informed particle swarm [26] updates the position of each particle using the weighted sum of all its neighbours; therefore all the neighbouring particles and the topology of the network play an even more influential role.

With the exception of cBBJ, the error information of the aforementioned algorithms used in this comparison are borrowed from a recent paper [27]. Table VIII shows that cBBJ is indeed a competitive PSO on the test set. In terms of convergence, cBBJ outperforms all other algorithms in the survey.

VII. CONCLUSION

This paper describes a family of bare bones swarm optimisation which was originally proposed to provide a better understanding of the behaviour of the particle swarm algorithm. The key features of this family are: the distinction between search focus and search spread as implemented in two separate informing networks (μ and σ), the use of a subspace jumping mechanism and the interaction between particle position and neighbourhood best position for the search spread (variance of the normal sampling) determination.

Although the intention was not to enhance the optimisation capability of standard PSO, in fact the fixed jump models introduced here (social and cognitive BBJ) offer promising results. In social BBJ, the distribution variance is determined by the separation of two neighbouring informers, scaled by a fixed parameter, α; in cognitive BBJ, the same quantity is

TABLE VI
MEAN ERROR IN LOCAL μ-NEIGHBOURHOOD.

Top: Mean error is shown with two decimal places after 50 trials of 300,000 function evaluations; bottom: based on Wilcoxon 1×1 Non-Parametric Statistical Test, if the difference between each pair of algorithms is significant, the pairs are marked. X $\leftarrow$ o shows that the left algorithm is significantly better than the right one; and o $\rightarrow$ X shows that the right one is significantly better than the left algorithm.

Fn	CK	BB	sBBJ	cBBJ	cBBNJ
f_1	9.48E-10	9.41E-10	9.47E-10	9.50E-10	9.49E-10
f_2	1.34E-01	1.54E+02	4.39E+00	3.36E-03	9.03E+01
f_3	1.22E+06	2.00E+07	1.42E+07	1.81E+06	1.49E+06
f_4	7.32E+03	7.14E+03	1.25E+04	2.61E+04	9.44E+04
f_5	5.05E+03	4.92E+03	7.81E+03	8.87E+03	9.86E+03
f_6	2.25E+01	9.22E+01	4.34E+01	4.71E+01	1.41E+02
f_7	1.26E-02	4.51E-02	7.70E-01	2.21E-02	2.63E-02
f_8	2.09E+01	2.10E+01	2.09E+01	2.01E+01	2.01E+01
f_9	9.04E+01	1.57E+01	2.53E-01	8.65E-10	2.21E+02
f_{10}	1.22E+02	1.89E+02	1.63E+02	4.46E+02	4.77E+02
f_{11}	3.04E+01	3.37E+01	3.03E+01	3.29E+01	3.40E+01
f_{12}	1.04E+04	1.44E+05	2.49E+04	4.30E+03	6.18E+03
f_{13}	5.80E+00	4.55E+00	1.89E+00	1.35E+00	2.11E+01
f_{14}	1.27E+01	1.27E+01	1.29E+01	1.33E+01	1.42E+01

Fn	CK–sBBJ	CK–cBBJ	BB–cBBJ	sBBJ–cBBJ	cBBNJ–cBBJ
f_1	-	-	-	-	-
f_2	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_3	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	X $\leftarrow$ o
f_4	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
f_5	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	-	o $\rightarrow$ X
f_6	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X	-	o $\rightarrow$ X
f_7	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_8	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	-
f_9	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_{10}	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
f_{11}	-	X $\leftarrow$ o	o $\rightarrow$ X	X $\leftarrow$ o	-
f_{12}	X $\leftarrow$ o	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	-
f_{13}	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X	o $\rightarrow$ X
f_{14}	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	X $\leftarrow$ o	o $\rightarrow$ X
$\sum$	10 - 2	8 - 5	4 - 9	4 - 7	1 - 9

given by the separation between the neighbourhood best and the particle position, again scaled by α.

Empirical tests over a challenging test set show that cBBJ in global μ and σ neighbourhoods performs significantly better than other bare bones algorithms and at least as well as the Clerc-Kennedy PSO when mean error is considered; nonetheless, cBBJ has the edge in terms of efficiency and reliability measures in all instances where there is convergence. A comparative study of current state-of-the-art PSO's and cBBJ also shows performance parity.

In some sense, cBBJ which retains particle position but lacks velocity, extrapolates between the pure bare bones idea and the particle dynamics of PSO, and this might be the key for its success. The global cBBJ algorithm is reminiscent of a swarm simulation with particles swarming around a leader, or around a marker left by the leader. Particles have no memory, but interact stigmergetically via the marker. It is remarkable therefore that the good optimisation performance emanates from such a simple algorithm; global cBBJ is perhaps the simplest interaction between particles that we can imagine.

Cognitive BBJ has some interesting features that distinguish it from other bare bones optimisers. The search variance is in general greater than in social BBJ by virtue of the retention of particle position. Outlying trial positions will lead immediately to a large variance at the next sampling and this effect will persist in subsequent trials. Particle interaction, as implemented by the communication strategy of social BBJ might have some advantage on very difficult problems such as the rotated Rastrigin, but more work is necessary for clarification of the issue.

Jumping plays a crucial role. Experiments show that a single jump component is optimal in 30D and that jumps that immediately improve a personal best tend to occur in the early phases of the optimisation. The recommended jump probability for cBBJ is 0.03 in this dimensionality; this means that one or two particles jump in a single component in each iteration of the swarm.

The fixed jump scheme proposed here, namely that a particle jumps with a probability p_P in a subspace of k dimensions, facilitates the scalability of the jump probability with dimension, albeit at the expense of introducing a further parameter (i.e. k) to the model. The scalability of bare bones swarms to higher dimensions remains, as yet, unexplored.

One of the curious features of the bare bones with jumps algorithms is there two global information sharing networks in contradistinction to the single local topology that is advocated for standard PSO. This feature too requires further study since the replacement of a local topology with a global one is

TABLE VII

A COMPARISON BETWEEN CK AND CBBJ IN THEIR PREFERRED TOPOLOGIES (I.E. CK: LOCAL, CBBJ: GLOBAL μ, σ-NEIGHBOURHOOD).

X–o shows that the left algorithm is better than the right one; and o–X shows that the right algorithm performs better than the left one. Table (a) summarises the performance based on errors (using two different criteria: best found error value, and Wilcoxon statistical test). Table (b) summarises the details based on the efficiency (number of function evaluations needed to reach the specified error of 10^{-10}) and, in parentheses, reliability (the number of trials in which the specified error is reached). The last row reports the reliability of each algorithm in percentage as well the number of times cBBJ significantly outperforms CK in terms of efficiency).

(a)

Fn	CK–cBBJ Best Err	CK–cBBJ Wilcoxon Test
f_1	o → X	o → X
f_2	o → X	o → X
f_3	o → X	o → X
f_4	X ← o	X ← o
f_5	X ← o	X ← o
f_6	o → X	o → X
f_7	o → X	X ← o
f_8	o → X	o → X
f_9	o → X	o → X
f_{10}	X ← o	X ← o
f_{11}	o → X	X ← o
f_{12}	o → X	o → X
f_{13}	o → X	o → X
f_{14}	X ← o	X ← o
$\sum$	4 - 10	6 - 8

(b)

Fn	CK	cBBJ	CK–cBBJ Wilcoxon
f_1	69,634 (50)	12,822 (50)	o → X
f_2	-	95,699 (50)	o → X
f_3	-	-	-
f_4	-	-	-
f_5	-	-	-
f_6	-	294,109 (2)	o → X
f_7	268,929 (1)	66,879 (6)	o → X
f_8	-	-	-
f_9	-	59,414 (50)	o → X
f_{10}	-	-	-
f_{11}	-	-	-
f_{12}	-	-	-
f_{13}	-	-	-
f_{14}	-	-	-
$\sum$	7.29% (51)	22.57% (158)	0 - 5

TABLE VIII

COMPARING CBBJ WITH COMPETITIVE PSOS.

Top: mean error of cBBJ is shown with two decimal places after 50 trials of 300,000 function evaluations with standard deviation in parentheses. 0.0(0) indicates convergence at 1e-8. Bottom: this table shows the outperformance of cBBJ algorithm in the majority of cases compared with other PSO varieties.

Fn	cBBJ	CLPSO	DMSPSO	UPSO	FIPS
f_1	0.0(0)	0.0(0)	3.14(4.15)E+02	1.31(0.73)E+03	5.25(5.57)E+02
f_2	0.0(0)	3.83(1.06)E+02	7.80(0.21)E+02	7.60(5.29)E+03	1.47(0.23)E+04
f_3	2.41(1.20)E+05	1.19(0.31)E+07	5.62(6.23)E+06	5.30(3.86)E+07	1.95(1.11)E+07
f_4	5.04(1.07)E+04	5.40(1.25)E+03	8.56(12.9)E+02	1.88(0.61)E+04	2.07(0.31)E+04
f_5	1.08(0.28)E+04	4.00(0.43)E+03	4.26(1.87)E+03	1.28(0.23)E+04	1.17(0.14)E+04
f_6	1.22(5.79)E-02	1.78(2.29)E+01	2.72(7.29)E+07	1.19(1.36)E+07	2.46(3.49)E+07
f_7	1.97(1.65)E-02	4.70(0)E+03	4.34(0.22)E+03	7.52(0.34)E+03	7.48(0.22)E+03
f_8	2.00(0)E+01	2.07(0)E+01	2.09(0)E+01	2.10(0)E+01	2.09(0)E+01
f_9	0.0(0)	0.0(0)	4.85(1.51)E+01	7.84(1.69)E+01	5.40(1.10)E+01
f_{10}	5.21(1.69)E+02	8.02(1.50)E+01	8.00(2.00)E+01	1.59(0.55)E+02	1.53(0.25)E+02
f_{11}	3.61(0.39)E+01	2.53(0.19)E+01	2.90(0.23)E+01	3.14(0.47)E+01	2.69(0.26)E+01
f_{12}	1.76(2.16)E+03	1.32(0.42)E+04	7.84(6.84)E+04	8.98(5.43)E+04	5.19(3.21)E+04
f_{13}	1.38(0.37)E+00	1.89(0.40)E+00	1.13(0.56)E+01	9.23(4.56)E+00	9.64(1.73)E+00
f_{14}	1.37(0.4)E+01	1.25(0.03)E+01	1.21(0.07)E+01	1.28(0.04)E+01	1.23(0.03)E+01

Fn	cBBJ–CLPSO	cBBJ–DMSPSO	cBBJ–UPSO	cBBJ–FIPS
f_1	-	X ← o	X ← o	X ← o
f_2	X ← o	X ← o	X ← o	X ← o
f_3	X ← o	X ← o	X ← o	X ← o
f_4	X ← o	o → X	o → X	o → X
f_5	o → X	o → X	X ← o	X ← o
f_6	X ← o	X ← o	X ← o	X ← o
f_7	X ← o	X ← o	X ← o	X ← o
f_8	X ← o	X ← o	X ← o	X ← o
f_9	-	X ← o	X ← o	X ← o
f_{10}	o → X	o → X	o → X	o → X
f_{11}	o → X	o → X	o → X	o → X
f_{12}	X ← o	X ← o	X ← o	X ← o
f_{13}	X ← o	X ← o	X ← o	X ← o
f_{14}	o → X	o → X	o → X	o → X
$\sum$	8 - 4	9 - 5	10 - 4	10 - 4

surprising, given the known superiority of the local topology for canonical PSO on multi-modal functions.

Excessive jumping, which amounts to subspace re-initialisation, arguably slows convergence. On the other hand too little jumping also weakens performance. The fact that jumping appears to be less necessary in cBBJ than in sBBJ

is perhaps attributable to the greater search diversity inherent in the computation of the search spread parameter σ. These heuristic arguments need to be rigorously explored.

REFERENCES

[1] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. IV. Piscataway, NJ: IEEE Service Center, 1995, pp. 1942–1948.

[2] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002.

[3] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317–325, 2003.

[4] Y. Yang and M. Kamel, "Clustering ensemble using swarm intelligence," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*. IEEE, 2003, pp. 65–71.

[5] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Hoboken, NJ: Wiley, 2005.

[6] F. van den Bergh and E. A. P., "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[7] T. Blackwell and D. Bratton, "Origin of bursts," in *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 2613–2620.

[8] R. Poli, "Mean and variance of the sampling distribution of particle swarm optimizers during stagnation," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 4, pp. 712–721, 2009.

[9] F. Van den Bergh and A. P. Engelbrecht, "A convergence proof for the particle swarm optimiser," *Fundamenta Informaticae*, vol. 105, no. 4, pp. 341–374, 2010.

[10] J. Kennedy, "Bare bones particle swarms," in *Proceedings of Swarm Intelligence Symposium, 2003 (SIS'03)*. IEEE, 2003, pp. 80–87.

[11] T. Richer and T. Blackwell, "The Lévy particle swarm," in *IEEE congress on evolutionary computation*, 2006, pp. 3150–3157.

[12] J. Peña, "Theoretical and empirical study of particle swarms with additive stochasticity and different recombination operators," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, ser. GECCO '08. New York, NY, USA: ACM, 2008, pp. 95–102. [Online]. Available: http://doi.acm.org/10.1145/1389095.1389109

[13] R. A. Krohling, "Gaussian particle swarm with jumps," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 2. IEEE, 2005, pp. 1226–1231.

[14] T. Blackwell, "A study of collapse in bare bones particle swarm optimisation," *IEEE Transactions on Evolutionary Computing*, vol. 16, no. 3, pp. 354–372, 2012.

[15] M. M. al-Rifaie and T. Blackwell, "Bare bones particle swarms with jumps," in *ANTS 2012, Lecture Notes in Computer Science series*, M. Dorigo and et al., Eds., vol. 7461. Springer, Heidelberg, 2012, pp. 49–60.

[16] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the IEEE International Conference on Evolutionary Computation*. Piscataway, NJ: IEEE Press, 1999, pp. 1945–1949.

[17] F. V. den Bergh, "An analysis of particle swarm optimizers," Ph.D. dissertation, University of Pretoria, South Africa, 2002.

[18] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technological University, Singapore and Kanpur Genetic Algorithms Laboratory, IIT Kanpur, Tech. Rep., 2005.

[19] S. Helwig, J. Branke, and S. Mostaghim, "Experimental analysis of bound handling techniques in particle swarm optimization," 2012.

[20] M. Clerc, "From theory to practice in particle swarm optimization," *Handbook of Swarm Intelligence*, pp. 3–36, 2010.

[21] ——. (2013) Particle swarm central, http://www.particleswarm.info. [Online]. Available: http://www.particleswarm.info/

[22] J. Liang, A. Qin, P. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 281–295, 2006.

[23] J. Liang and P. Suganthan, "Dynamic multi-swarm particle swarm optimizer," in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*. IEEE, 2005, pp. 124–129.

[24] K. Parsopoulos and M. Vrahatis, "Upso: A unified particle swarm optimization scheme," *Lecture Series on Computer and Computational Sciences*, vol. 1, pp. 868–873, 2004.

[25] ——, *Particle swarm optimization and intelligence: advances and applications*. Information Science Reference Hershey, 2010.

[26] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: simpler, maybe better," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 204–210, 2004.

[27] M. Epitropakis, V. Plagianakos, and M. Vrahatis, "Evolving cognitive and social experience in particle swarm optimization through differential evolution: A hybrid approach," *Information Sciences*, 2012.