Soft Computing manuscript No.
(will be inserted by the editor)

μJADE: Adaptive Differential Evolution with a Small Population

Craig Brown · Yaochu Jin · Matthew Leach · Martin Hodgson

Received: date / Accepted: date

Abstract This paper proposes a new differential evolution (DE) algorithm for unconstrained continuous optimisation problems, termed μJADE, that uses a small or 'micro' (μ) population. The main contribution of the proposed DE is a new mutation operator, 'current-by-rand-to-pbest.' With a population size less than 10, μJADE is able to solve some classical multimodal benchmark problems of 30 and 100 dimensions as reliably as some state-of-the-art DE algorithms using conventionally sized populations. The algorithm also compares favourably to other small population DE variants and classical DE.

Keywords Micro differential evolution · Small Population · External Archive · JADE

1 Introduction

Differential Evolution (DE) (Storn and Price, 1997) is a derivative free, population-based global optimisation algorithm. Its advantages are ease of use, ease of implementation and fast convergence. Recently, many new variants of DE algorithms have been developed Gong et al (2011b,a); Gong and Cai (2013). In addition, DE has enjoyed success in a range of applications such as bioprocess optimisation and urban energy management among many more (Das and Suganthan, 2011).

DE can also be used for on-line optimisation tasks arising in control, notably nonlinear model predictive control (NMPC) (Yu et al, 2008). For DE to be practical as an optimiser for NMPC, attention must be paid to the number of function evaluations that occur between time steps, especially if the cost function is expensive to evaluate.

When applying DE to on-line optimisation problems, such as NMPC, it can be necessary to re-evaluate the population if the optima move with time (Mendes and Mohais, 2005). Unfortunately, population based optimisation algorithms such as DE require large populations; the recommended population size NP for DE given a problem of D dimensions generally ranges from $2D$ to $40D$ (Ronkkonen et al, 2005). For real-time use the computational cost of an iteration or population re-evaluation could be restrictively large. Workarounds normally involve re-evaluating a subset of the population rather than the population in entirety. However, this approach may ignore promising search directions depending on how abruptly the fitness surface changes—it is desirable to re-evaluate the entire population between time steps.

For control applications, reliability of the optimisation is important: a poor solution could result in unacceptable plant behaviour. It is known that DE (typically) converges more quickly with smaller populations than with larger populations but at the cost of reduced reliability (Mallipeddi and Suganthan, 2008). However, the feasibility of using DE in embedded systems is improved by using small populations—the memory requirement is reduced.

This work was funded by Bosch Thermotechnology Ltd. and the Engineering and Physical Sciences Research Council (EPSRC) UK.

C. Brown · M. Hodgson
Bosch Thermotechnology Ltd.
Worcester, Worcestershire, WR4 9SW, UK
E-mail: craig.brown@uk.bosch.com

Y. Jin
Department of Computing
University of Surrey
Guildford, Surrey, GU2 7XH, UK
E-mail: yaochu.jin@surrey.ac.uk

M.Leach
Centre for Environmental Strategy
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK
E-mail: m.leach@surrey.ac.uk

In view of the above considerations, μJADE is introduced. The idea is to acquire reliability comparable to that of state-of-the-art DE but using a much smaller population than is commonly practised.

The rest of the paper is structured as follows. Firstly, an overview of classical DE, JADE (Zhang and Sanderson, 2009b) and R_{cr}-JADE (Gong et al, 2014) is given. Literature relating to the use of small populations in DE is also reviewed.

Secondly, using R_{cr}-JADE as a foundation, μJADE is introduced. The modifications are described in Section 3 and the complete pseudocode is given in Algorithm 2.

Finally, μJADE is compared to three DE variants designed for small populations using a fixed small population size. It is then compared to 2 state-of-the-art DE algorithms that use a conventionally sized population: R_{cr}-JADE-s4 and FSADE (Sharma et al, 2014), as well as classical DE. Performance is compared in terms of function evaluations and success rate on some classical benchmarks at 30 dimensions. In addition, μJADE is compared to R_{cr}-JADE-s4 at 100 dimensions.

2 Related Work

This section is an overview of the original DE algorithm, its variants designed for small populations and a description of the adaptive DE variants JADE and R_{cr}-JADE, on which μJADE is based.

2.1 Differential Evolution (DE)

DE is a method of solving optimisation problems of the form:

$$\textit{Minimise} \quad f(\boldsymbol{x}),\ \boldsymbol{x} \in \Re^D \tag{1}$$

where D is the dimensionality of the optimisation problem and $\boldsymbol{x} = [x_1, x_2, ..., x_d]^T$ is the vector of decision variables. Each variable x_j satisfies a boundary constraint:

$$L_j \leq x_j \leq U_j,\ j = 1, 2, ..., D,\ \boldsymbol{L} \in \Re^D,\ \boldsymbol{U} \in \Re^D \tag{2}$$

where L_j and U_j are the lower and upper bound of x_j respectively.

There are 4 stages to DE. Firstly, a set of candidate solutions is created (initialisation). This set is called the population. Secondly, an operator is applied to each individual or target vector to create a mutant vector (mutation). Thirdly, another operator is applied to the target vector and the mutant vector to give a trial vector (crossover). Finally, a selection operation is used to determine which trial and target vectors are used in the next population. The last 3 stages are repeated until a satisfactory solution is found—each repetition is called a generation.

2.1.1 Initialisation

Initially the population $P = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_{NP}\}$ is generated randomly. The i'th vector $\boldsymbol{x}_i \in P$ is initialised as follows:

$$x_{i,j} = L_j + \text{rand}(0,1)\,(U_j - L_j) \tag{3}$$

where rand$(0,1)$ is drawn from a uniform distribution in (0,1), $i = 1, 2, ...NP$ and $j = 1, 2, ..., D$.

2.1.2 Mutation

A mutation operator is applied to each target vector $\boldsymbol{x}_i$. The classical mutation operator denoted DE/rand/1 is as follows:

$$\boldsymbol{v}_i = \boldsymbol{x}_a + F(\boldsymbol{x}_b - \boldsymbol{x}_c) \tag{4}$$

where $i, a, b, c \in \{1, 2, ..., NP\}$ and $i \neq a \neq b \neq c$.

Often it is important that the bounds of the problem aren't violated by the mutation operation. One scheme for ensuring this is given in Zhang and Sanderson (2009b):

$$v_{i,j} = \begin{cases} \dfrac{(L_j + x_{i,j})}{2} \text{ if } v_{i,j} < L_j \\ \dfrac{(U_j + x_{i,j})}{2} \text{ if } v_{i,j} > U_j \\ v_{i,j} \text{ otherwise} \end{cases} \tag{5}$$

2.1.3 Crossover

Following mutation, a crossover operator is applied to each target vector $\boldsymbol{x}_i$ and its associated mutant vector $\boldsymbol{v}_i$ to give a trial vector $\boldsymbol{u}_i$. A popular crossover operator is the binomial crossover:

$$u_{i,j} = \begin{cases} v_{i,j} \text{ if OR}(\text{rand}(0,1) < CR, j = j_{rand}) \\ x_{i,j} \text{ otherwise} \end{cases} \tag{6}$$

where $CR \in [0,1]$ and $j_{rand} \in \{1, 2, ..., D\}$ and is randomly selected.

2.1.4 Selection

Finally, the selection operation replaces members of the population with the corresponding trial vector if the trial vector has a better fitness.

$$\boldsymbol{x}_i = \begin{cases} \boldsymbol{u}_i \text{ if } f(\boldsymbol{u}_i) < f(\boldsymbol{x}_i) \\ \boldsymbol{x}_i \text{ otherwise} \end{cases} \tag{7}$$

where $f(\boldsymbol{x})$ is the objective function to be optimised.

An alternative approach is to replace the target vector with the trial vector if the trial vector has a better or equal fitness. In the case that the population lies entirely on a plateau it keeps moving as long as the population is not identical.

This is useful for small populations where firstly, this scenario is more likely and secondly, the number of possible outcomes per mutation and target vector is smaller (moving the population creates new mutation possibilities even if it does not improve the fitness). Contrast this to the former selection method where the population will remain stationary on a plateau unless a fitness improvement can be made—the number of possible trial vectors is relatively limited. This increases the risk of stagnation (Lampinen and Zelinka, 2000).

$$\boldsymbol{x}_i = \begin{cases} \boldsymbol{u}_i \text{ if } f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i) \\ \boldsymbol{x}_i \text{ otherwise} \end{cases} \tag{8}$$

2.2 Small populations in DE

Mallipeddi and Suganthan (2008) carried out a study of the effects of population size on DE using 2 mutation operators, DE/best/1 and DE/rand to best/2, with NP ranging from $2D$ to $10D$. They concluded that smaller populations with greedy mutation strategies converge quickly but are more likely to stagnate or converge prematurely. Conversely, a large population with an exploratory mutation operator significantly reduces the probability of this happening at the cost of slower convergence.

Ren et al (2010) developed a new mutation operator for smaller population sizes in DE. They were able to solve some 30 dimensional test problems using a population size as low as 5. They added a random disturbance to DE/rand/1/bin:

$$\boldsymbol{v}_i = \boldsymbol{x}_a + F(\boldsymbol{x}_b - \boldsymbol{x}_c + \text{rand}(-1,1)\boldsymbol{\delta}) \tag{9}$$

where rand$(-1,1)$ generates a random number in the interval (-1,1) and $\boldsymbol{\delta}$ is a function of the fraction of the population IR that improves at each generation:

$$\boldsymbol{\delta}(IR) = \begin{cases} \boldsymbol{\delta}\eta \text{ if } IR < 0.2 \\ \dfrac{\boldsymbol{\delta}}{\eta} \text{ if } IR > 0.2 \\ \boldsymbol{\delta} \text{ otherwise} \end{cases} \tag{10}$$

where $\boldsymbol{\delta}$ is initialised as follows:

$$\delta_j = \alpha(U_j - L_j) \tag{11}$$

where α is a constant.

Brest and Maučec (2011) made use of the different behaviour of small and large populations with different mutation operators in their jDEl-scop algorithm. The algorithm uses an adaptive population size, adaptive F_i and CR_i and an ensemble of mutation and crossover strategies. Using a starting population size of 100, they were able to consistently solve some benchmark functions at $D = 200$, $D = 500$ and $D = 1000$. Some other adaptive population schemes are given in Teo (2006), Teng et al (2009), Wang and Zhao (2013), Yang et al (2013), Zhao et al (2014) and Choi and Ahn (2014).

Fajfar et al (2012) investigated population sizes as low as 10 for a set of $D = 30$ benchmark problems. They combined random perturbation of the trial vector (Equation 12) with a new selection operation (Algorithm 1) to improve the performance of DE/rand/1/bin, with the improvement of performance most pronounced at low population sizes. The selection operation works by allowing each trial vector to be compared to each target vector and to the first half of the population sequentially. If the fitness function is improved compared to the population vector, that vector is replaced by the trial vector and the selection process is restarted for the next target vector.

$$u_{i,j} = \begin{cases} L_j + \text{rand}(0,1) \times (U_j - L_j) \text{ if rand}(0,1) \leq 0.005 \\ u_{i,j} \text{ otherwise} \end{cases} \tag{12}$$

Algorithm 1: Selection operation of Fajfar et al (2012)

```
if f(u_i) < f(x_i) then
    c = i
else
    c = −1
    for p = 1 to NP/2 do
        if f(u_i) < f(x_p) then
            c = p
            exit loop
if c ≠ −1 then
    x_c = u_i
```

Salehinejad et al (2014) increased the diversity of the population in small population DE by vectorising the scaling factor, F_i. Rather than F_i being a scalar for each target vector, $\boldsymbol{F}_i$ is a vector of length D and each element is drawn from a uniform distribution in (0.1, 1.5) for each population member. The mutation becomes:

$$\boldsymbol{v}_{i,j} = \boldsymbol{x}_{a,j} + F_{i,j}(\boldsymbol{x}_{b,j} - \boldsymbol{x}_{c,j}) \tag{13}$$

where $j = 1, 2, ..., D$.

In summary, the literature indicates that the performance of DE at small populations can be improved using the right modifications. Perturbation appears an important theme. Improving the number of possible trial vectors is also important—Salehinejad et al (2014) achieved this by randomising

the scaling factor for each individual and each dimension. The main difficulty in using small populations in DE is overcoming their limited exploration ability. Selection may also be an important area of enquiry that has received little attention in the literature so far (Fajfar et al, 2011).

A distinct but related field in DE is compact DE (cDE) (Mininno et al, 2011). In cDE, the population is replaced by a statistical representation whose memory requirement is equivalent to a population of 4 individuals regardless of the dimensionality of the problem, though the search behaves as if the population were larger due to the randomised creation of individuals at each iteration. Compact DE is not investigated here—the emphasis of this paper is on using DE with small non-virtual populations.

2.3 JADE: Adaptive Differential Evolution

Zhang and Sanderson (2009b) introduced an adaptive DE variant called JADE with an optional external archive for conventionally sized populations. The archive contains previous population members that had been replaced by a trial vector. The relevance of the archive to small populations is that there will be a larger set of possible outcomes for a given trial vector as if the population were larger—a higher number of possible trial outcomes lowers the risk of the population stagnating (Lampinen and Zelinka, 2000). The mutation operator for JADE, denoted DE/current-to-pbest/1, without an external archive is:

$$\boldsymbol{v}_i = \boldsymbol{x}_i + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_i) + F_i(\boldsymbol{x}_a - \boldsymbol{x}_b) \tag{14}$$

where $\boldsymbol{x}^p_{best}$ is randomly chosen from the top 100p% population members. The mutation for JADE with an external archive is:

$$\boldsymbol{v}_i = \boldsymbol{x}_i + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_i) + F_i(\boldsymbol{x}_a - \tilde{\boldsymbol{x}}_b) \tag{15}$$

where $\tilde{\boldsymbol{x}}_b$ is randomly chosen from $P \cup A$. The effect is to improve the diversity of the population.

Alternatively, DE/rand-to-pbest/1, introduced by Zhang and Sanderson (2009a), is:

$$\boldsymbol{v}_i = \boldsymbol{x}_a + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_a) + F_i(\boldsymbol{x}_b - \boldsymbol{x}_c) \tag{16}$$

and with archive:

$$\boldsymbol{v}_i = \boldsymbol{x}_a + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_a) + F_i(\boldsymbol{x}_b - \tilde{\boldsymbol{x}}_c) \tag{17}$$

where $\tilde{\boldsymbol{x}}_c$ is randomly chosen from $P \cup A$.

Following the mutation step, JADE uses the binomial crossover operator given in Equation 6 and the selection operation given in Equation 7.

In JADE, F_i and CR_i are randomly generated at the beginning of each generation according to:

$$CR_i = \text{randn}_i(\mu_{CR}, 0.1); \tag{18}$$

$$F_i = \text{randc}_i(\mu_F, 0.1); \tag{19}$$

where CR_i is drawn from a normal distribution of mean μ_{CR} and standard deviation 0.1 and F_i is drawn from a Cauchy distribution of location parameter μ_F and scale factor 0.1. As long as $F_i \leq 0$ it is redrawn from the distribution. If $F_i > 1$ it is truncated to 1. CR is truncated to [0, 1]. At the end of each generation μ_{CR} and μ_F are updated as follows:

$$\mu_{CR} = (1-c)\mu_{CR} + c \cdot \text{L}_1(S_{cr}) \tag{20}$$

$$\mu_F = (1-c)\mu_F + c \cdot \text{L}_2(S_F) \tag{21}$$

where S_{cr} is the set of successful CR values in the current generation, S_F is the set of successful F values in the current generation and:

$$\text{L}_p(\{z_1, z_2, \ldots z_n\}) = \frac{\sum_{k=1}^{n} z_k^p}{\sum_{k=1}^{n} z_k^{p-1}} \tag{22}$$

2.4 R_{cr}-JADE

As discussed in Section 2.3, JADE uses an adaptive CR and F scheme. Gong et al (2014) introduced a modification to JADE whereby CR is corrected at each generation based on the actual crossover rate a posteriori. The crossover operation becomes:

$$d_{i,j} = \text{rand}(0,1) \tag{23}$$

$$u_{i,j} = \begin{cases} v_{i,j} & \text{if OR}(d_{i,j} < CR_i, j = j_{rand}) \\ x_{i,j} & \text{otherwise} \end{cases} \tag{24}$$

$$b_{i,j} = \begin{cases} 1 & \text{if OR}(d_{i,j} < CR_i, j = j_{rand}) \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$CR_i = \frac{\sum_{j=1}^{D} b_{i,j}}{D} \tag{26}$$

Another change from JADE is that R_{cr}-JADE uses the selection operation given in Equation 8 rather than that of Equation 7.

3 Modifying R_{cr}-JADE for small populations: μJADE

In this section, four modifications to R_{cr}-JADE intended specifically for use with small populations are introduced. The aim is to retain the desirable property of small populations, that is, fast convergence, whilst improving the robustness, which is typically associated with larger populations.

DE with small populations is known as μDE or micro DE. Therefore, the new algorithm is denoted μJADE, to indicate both its origin and its suitability to small populations.

3.1 New mutation operator for R_{cr}-JADE

A new mutation operator is introduced denoted current-by-rand-to-pbest/1:

$$\boldsymbol{v}_i = \boldsymbol{x}_i + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_a) + F_i(\boldsymbol{x}_b - \tilde{\boldsymbol{x}}_c) \tag{27}$$

where $\tilde{\boldsymbol{x}}_c$ is randomly chosen from $P \cup A$. The idea of the mutation is to improve the exploratory power of small populations whilst retaining good convergence performance. When $\boldsymbol{x}_i \not\approx \boldsymbol{x}_a$ and $\boldsymbol{x}_i \neq \boldsymbol{x}_a$, current-by-rand-to-pbest/1 is exploratory. However, when $\boldsymbol{x}_i \approx \boldsymbol{x}_a$, current-by-rand-to-pbest/1 is similar to current-to-pbest/1. The latter is more likely as the population and archive converges. The aim is to accelerate convergence in the later stages of optimisation and reduce the likelihood of the population accumulating at a false optimum in the early stages of optimisation.

Conventionally in JADE, $\tilde{\boldsymbol{x}}_c \neq \boldsymbol{x}_b$ and $\tilde{\boldsymbol{x}}_c \neq \boldsymbol{x}_a$. These constraints are removed for μJADE. Instead, the constraint $\boldsymbol{x}^p_{best} \neq \boldsymbol{x}_a$ is upheld. Then, when $\tilde{\boldsymbol{x}}_c = \boldsymbol{x}_b$ and $\boldsymbol{x}_i \approx \boldsymbol{x}_a$, the mutation is greedy. The constraint $\boldsymbol{x}_a \neq \boldsymbol{x}_b$ is upheld to prevent the second displacement term reversing the first if $\boldsymbol{x}^p_{best} = \tilde{\boldsymbol{x}}_c$ and $\boldsymbol{x}_b = \boldsymbol{x}_a$.

3.2 Changes to F and CR adaptation

In JADE and R_{cr}-JADE μ_{CR} and μ_F are updated every generation according to Equations 20 and 21 respectively. As long as $S_{cr} = \emptyset$, CR decays at each generation. Similarly, as long as $S_F = \emptyset$, μ_F decays at each generation. For small populations, the probability of achieving a successful trial vector at each generation is lower than for large populations, resulting in F and CR values quickly diminishing. Put another way, the sample size of successful F and CR values is not large enough to give reliable estimates for μ_F and μ_{CR}.

In order to solve this, μJADE updates μ_{CR} and μ_F every $\max(100, 10D)$ generations rather than every 1 generation. The lower limit, 100, through trial and error was found to perform reasonably. This modification can cause the sets S_{CR} and S_F to become very large. Therefore, it is recommended to calculate μ_F and μ_{CR} recursively in practice.

3.3 Perturbation

In order to give μJADE a chance of escaping false optima and improve diversity, the perturbation method of Fajfar et al (2012) is incorporated into μJADE after crossover. To incorporate this perturbation mechanism without disrupting the crossover repair introduced by Gong et al (2014), b_i is corrected after the perturbation step before calculating the corrected crossover:

$$r_{i,j} = \text{rand}(0,1) \tag{28}$$

$$u_{i,j} = \begin{cases} L_j + \text{rand}(0,1)(U_j - L_j) & \text{if } r_{i,j} \leq 0.005 \\ u_{i,j} & \text{otherwise} \end{cases} \tag{29}$$

$$b_{i,j} = \begin{cases} 0 & \text{if } r_{i,j} \leq 0.005 \\ b_{i,j} & \text{otherwise} \end{cases} \tag{30}$$

3.4 Restart

As an 'insurance' for the worst case scenario where the best fitness stagnates despite the aforementioned modifications, the population (excluding the best member) is re-initialised if the best fitness doesn't improve for $\max(1000, 100D)$ generations.

4 Experimental Setup

μJADE is first compared to some small population DE variants for problems of 30 dimensions. Experimentally, it was found that 8 is the smallest population with which μJADE works effectively. In order to compare to other small population DE variants, a fixed population size of 8 is also used. The population size is fixed across all the variants since even a small difference in population size is proportionally significant when using very small populations.

Firstly, the small population algorithms DESP (Ren et al, 2010), MDEVM (Salehinejad et al, 2014) and MDEVM with the perturbation and selection modifications of Fajfar et al (2012) (denoted MDEVM-Fajfar) are compared (we found this combination to perform better than DE/rand/1/bin-Fajfar). As DESP, MDEVM and MDEVM-Fajfar are closely related to DE/rand/1/bin and use static parameters, the standard setting of $CR = 0.9$ is used. Additionally, $F = 0.5$ for DESP and DE/rand/bin/1 (Montgomery and Chen, 2010; Zhang and Sanderson, 2009b). Otherwise, DESP uses the parameter settings given in Ren et al (2010).

For μJADE, only the value for p differs from that specified by Zhang and Sanderson (2009b); in the original JADE algorithm $p = 0.05$ which is too small for the population sizes used in μJADE (pNP should be an integer greater than 1). Therefore, for μJADE $p = 3/NP$.

μJADE is then compared to some state-of-the-art DE variants that use conventionally sized populations—r_{cr}-JADE-s4 (Gong et al, 2014) and FSADE (Sharma et al, 2014). μJADE is also compared to DE/rand/1/bin. For R_{cr}-JADE-s4, $\mu_F = 0.5$, $\mu_{CR} = 0.5$, $p = 0.05$, $c = 0.1$ and $NP = 100, 400$ for $D = 30, 100$ respectively (Zhang and Sanderson, 2009b; Gong et al, 2014). For FSADE the settings given in Sharma et al (2014) are used.

In addition, the mutations rand-to-pbest/1 and current-by-rand-to-pbest/1 are compared in μJADE for a small fixed number of function evaluations.

Algorithm 2: μJADE

Initialise population
$\mu_{CR} = 0.5$
$\mu_F = 0.5$
$A = \emptyset$
for $g = 1$ to number of generations **do**
  for $i = 1$ to NP **do**
    $CR_i = \text{randn}_i(\mu_{CR}, 0.1)$
    $F_i = \text{randc}_i(\mu_F, 0.1)$
    Randomly select $\boldsymbol{x}_a \neq \boldsymbol{x}_i$
    Randomly select $\boldsymbol{x}_b \neq \boldsymbol{x}_a \neq \boldsymbol{x}_i$
    Randomly select $\boldsymbol{x}^p_{best} \neq \boldsymbol{x}_a$ from pNP best population members
    Randomly select $\boldsymbol{x}_c$ from $P \cup A$
    Randomly select $j_{rand} \in \mathbb{N}^+_{\leq D}$
    /* Mutation */
    $\boldsymbol{v}_i = \boldsymbol{x}_i + F_i(\boldsymbol{x}^p_{best} - \boldsymbol{x}_a) + F_i(\boldsymbol{x}_b - \tilde{\boldsymbol{x}}_c)$
    $v_{i,j} = \begin{cases} \frac{(L_j + x_{i,j})}{2} \text{ if } v_{i,j} < L_j \\ \frac{(U_j + x_{i,j})}{2} \text{ if } v_{i,j} > U_j \\ v_{i,j} \text{ otherwise} \end{cases}$
    /* Crossover */
    for $j = 1$ to D **do**
      $d_{i,j} = \text{rand}(0,1)$
      $u_{i,j} = \begin{cases} v_{i,j} \text{ if OR}(d_{i,j} < CR_i, j = j_{rand}) \\ x_{i,j} \text{ otherwise} \end{cases}$
      $b_{i,j} = \begin{cases} 1 \text{ if OR}(d_{i,j} < CR_i, j = j_{rand}) \\ 0 \text{ otherwise} \end{cases}$
    /* Perturbation */
    for $j = 1$ to D **do**
      $r_{i,j} = \text{rand}(0,1)$
      $u_{i,j} = \begin{cases} L_j + \text{rand}(0,1)(U_j - L_j) \text{ if } r_{i,j} \leq 0.005 \\ u_{i,j} \text{ otherwise} \end{cases}$
      $b_{i,j} = \begin{cases} 0 \text{ if } r_{i,j} \leq 0.005 \\ b_{i,j} \text{ otherwise} \end{cases}$
    /* Crossover Repair */
    $CR_i = \frac{\sum_{j=1}^{D} b_{i,j}}{D}$
    /* Selection */
    if $f(\boldsymbol{u}_i) \leq f(\boldsymbol{x}_i)$ **then**
      $\boldsymbol{x}_i \rightarrow A$
      $\boldsymbol{x}_i = \boldsymbol{u}_i$
      $CR_i \rightarrow S_{CR}$
      $F_i \rightarrow S_F$
      if $\boldsymbol{u}_i$ is fitter than best population member **then**
        $BIR = BIR + 1$
  /* Archive Update */
  Randomly remove solutions from A so that $|A| \leq NP$
  /* Parameter Update */
  if mod(g,max(100,10D) = 0 **then**
    $\mu_{CR} = (1-c)\mu_{CR} + cL_1(S_{CR})$ // $L_1(\emptyset) = 0$
    $\mu_F = (1-c)\mu_F + cL_2(S_F)$ // $L_2(\emptyset) = 0$
    $S_{CR} = S_F = \emptyset$
  /* Reset */
  if mod(g,max(1000,100D) = 0 **then**
    if $BIR = 0$ **then**
      Reinitialise population apart from best member
    $BIR = 0$

In this work, all variants apply Equation 5 after the mutation to prevent bounds violation.

The scalable benchmark functions are given in Table 1. Respectively, they are known as the Sphere, Schwefel 2.22, Schwefel 1.2, Schwefel 2.21, Rosenbrock, Step, Noisy Quartic, Schwefel 2.26, Rastringin, Ackley, Griewank, and the two Generalized Penalty Functions (Yao et al, 1999).

The comparisons are made in terms of success rate and number of function evaluations required to achieve a solution accuracy of less than 1.0e-02 for f_7 and 1.0e-08 for all other functions. If the required solution accuracy isn't achieved after $100000D$ function evaluations the run is considered unsuccessful. This large number of function evaluations ensures the algorithms are tested to exhaustion, emphasising reliability over convergence speed.

Table 1 Benchmark Functions (Yao et al, 1999)

Function	Initial Range
$f_1(x) = \sum_{i=1}^{D} x_i^2$	$[-100,100]^D$
$f_2(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10,10]^D$
$f_3(x) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} x_j \right)^2$	$[-100,100]^D$
$f_4(x) = \max_i \{ \lvert x_i \rvert \}$	$[-100,100]^D$
$f_5(x) = \sum_{i=1}^{D-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right]$	$[-30,30]^D$
$f_6(x) = \sum_{i=1}^{D} \lfloor x_i + 0.5 \rfloor^2$	$[-100,100]^D$
$f_7(x) = \sum_{i=1}^{D} i x_i^4 + \text{rand}[0,1)$	$[-1.28,1.28]^D$
$f_8(x) = \sum_{i=1}^{D} -x_i \sin\left(\sqrt{\lvert x_i \rvert} \right)$	$[-500,500]^D$
$+418.98288727243369D$	
$f_9(x) = \sum_{i=1}^{D} \left[x_i^2 - 10\cos(2\pi x_i) + 10 \right]$	$[-5.12,5.12]^D$
$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2} \right)$	$[-32,32]^D$
$-\exp\left(\frac{1}{D}\sum_{i=1}^{D} \cos(2\pi x_i) \right) + 20 + e$	
$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}} \right) + 1$	$[-600,600]^D$
$f_{12}(x) = \frac{\pi}{D}\{10\sin(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2$	$[-50,50]^D$
$\left[1 + 10\sin^2(\pi y_{i+1}) \right] + (y_D - 1)^2 \} +$	
$\sum_{i=1}^{D} u(x_i, 10, 100, 4)$	
where $y_i = 1 + 0.25(x_i + 1)$ and	
$u(x_i,a,k,m) = \begin{cases} k(x_i - a)^m \\ 0 \\ k(-x_i - a)^m \end{cases}$	
$f_{13}(x) = 0.1\{\sin^2(3\pi x_1) + \sum_{i=1}^{D-1}(x_i - 1)^2$	$[-50,50]^D$
$[1 + \sin^2(3\pi x_{i+1})] + (x_D - 1)^2[1 + \sin^2(2\pi x_D)]\} +$	
$\sum_{i=1}^{D} u(x_i, 5, 100, 4)$	

5 Experimental Results and Analysis

Table 2 shows the Success Rate (SR) and mean number of Function Evaluations (FE) in successful runs of MDEVM-Fajfar, MDEVM, DESP and μJADE in the 30 dimensional test problems. μJADE is clearly the most reliable small population algorithm overall.

DESP performs poorly across most of the benchmarks. In DESP, individuals will be more greatly perturbed as CR increases because the perturbation is incorporated into the

Table 2 Comparison of small population algorithms for D = 30. Mean over 50 independent runs. Best results are in boldface and are ranked first by reliability then by convergance speed (Wilcoxon $\alpha = 0.05$)

f	MDEVM-Fajfar$_{NP=8}$		MDEVM$_{NP=8}$		DESP$_{NP=8}$		μJADE$_{NP=8}$	
	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)
f_1	98	8.6e+04 (4.1e+05)	100	2.8e+04 (3.3e+03)	100	**1.9e+04** (8.0e+02)	100	2.2e+04 (7.8e+02)
f_2	86	6.0e+04 (1.5e+05)	100	**3.1e+04** (4.7e+03)	0	—	100	3.7e+04 (1.2e+03)
f_3	54	3.4e+05 (4.7e+05)	100	**1.4e+05** (1.5e+04)	0	—	100	1.6e+05 (7.7e+03)
f_4	0	—	0	—	0	—	**100**	2.2e+05 (1.6e+04)
f_5	0	—	52	2.3e+06 (4.0e+05)	0	—	**98**	2.1e+05 (5.5e+04)
f_6	100	4.1e+05 (3.1e+05)	100	1.8e+04 (6.3e+03)	0	—	100	**1.2e+04** (3.8e+03)
f_7	100	3.4e+05 (2.2e+05)	76	8.9e+05 (7.1e+05)	0	—	100	**2.3e+05** (1.6e+05)
f_8	96	6.9e+04 (1.6e+05)	0	—	0	—	**100**	1.0e+05 (5.1e+03)
f_9	90	1.6e+05 (4.0e+05)	0	—	0	—	**100**	1.2e+05 (6.8e+03)
f_{10}	70	9.5e+04 (1.0e+05)	0	—	0	—	**100**	3.8e+04 (5.0e+03)
f_{11}	24	4.0e+04 (4.7e+04)	20	2.6e+04 (3.1e+03)	50	3.1e+04 (1.7e+03)	**100**	4.6e+04 (3.7e+04)
f_{12}	100	9.5e+04 (2.3e+05)	40	6.6e+04 (6.5e+04)	2	1.8e+06 (0.0e+00)	100	3.8e+04 (7.5e+03)
f_{13}	92	4.2e+05 (6.8e+05)	32	7.0e+04 (6.4e+04)	0	—	**100**	3.2e+04 (1.8e+04)
	70		48		12		**99.85**	

Table 3 Comparison for D = 30. Mean over 50 independent runs.

f	DE/rand/1/bin$_{NP=100}$		FSADE$_{NP=50}$		R_{cr}-JADE-s4$_{NP=100}$		μJADE$_{NP=8}$	
	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)
f_1	100	9.2e+04 (2.0e+03)	100	3.5e+04 (5.3e+02)	100	**2.1e+04** (4.9e+02)	100	2.2e+04 (7.8e+02)
f_2	100	1.5e+05 (2.6e+03)	100	4.5e+04 (5.7e+02)	100	**3.3e+04** (7.6e+02)	100	3.7e+04 (1.2e+03)
f_3	100	3.9e+05 (1.1e+04)	20	2.6e+06 (4.8e+05)	98	6.7e+04 (3.6e+03)	100	**1.6e+05** (7.7e+03)
f_4	2	3.2e+05 (0.0e+00)	92	2.2e+05 (4.2e+03)	100	**1.0e+05** (3.5e+04)	100	2.2e+05 (1.6e+04)
f_5	**100**	3.7e+05 (1.4e+04)	18	2.8e+06 (6.1e+04)	98	7.8e+04 (2.7e+03)	98	2.1e+05 (5.5e+04)
f_6	100	3.4e+04 (1.6e+03)	100	1.3e+04 (4.7e+02)	100	**8.8e+03** (3.6e+02)	100	1.2e+04 (3.8e+03)
f_7	100	1.4e+05 (3.3e+04)	100	1.1e+05 (2.6e+04)	100	**2.2e+04** (9.8e+03)	100	2.3e+05 (1.6e+05)
f_8	56	3.2e+05 (6.6e+04)	90	4.4e+04 (1.8e+03)	92	7.0e+04 (2.8e+03)	**100**	1.0e+05 (5.1e+03)
f_9	0	—	98	8.9e+04 (4.8e+03)	100	**9.1e+04** (1.3e+03)	100	1.2e+05 (6.8e+03)
f_{10}	100	1.4e+05 (2.6e+03)	100	4.8e+04 (5.0e+02)	100	**3.1e+04** (6.4e+02)	100	3.8e+04 (5.0e+03)
f_{11}	100	9.6e+04 (2.5e+03)	100	3.9e+04 (3.7e+03)	100	**2.2e+04** (6.4e+02)	100	4.6e+04 (3.7e+04)
f_{12}	100	8.5e+04 (3.1e+03)	100	4.0e+04 (1.0e+03)	100	**1.9e+04** (5.5e+02)	100	3.8e+04 (7.5e+03)
f_{13}	100	9.1e+04 (2.9e+03)	100	4.1e+04 (1.1e+03)	100	**2.1e+04** (4.9e+02)	100	3.2e+04 (1.8e+04)
	81		86		99		**99.85**	

Table 4 Comparison of the mutation operators for $D = 30$, $NP = 8$. Mean fitness after 3e+04 function evaluations over 50 independent runs.

f	current-by-rand-to-pbest/1	rand-to-pbest/1
	$f(\boldsymbol{x})$ Mean (Std)	$f(\boldsymbol{x})$ Mean (Std)
f_1	**9.0e-13** (2.5e-12)	5.3e-01 (9.5e-01)
f_2	**1.3e-06** (1.6e-06)	1.2e-01 (8.2e-02)
f_3	**9.7e+02** (4.2e+02)	2.6e+03 (1.4e+03)
f_4	**4.9e+00** (2.5e+00)	1.1e+01 (1.7e+00)
f_5	**4.3e+01** (2.9e+01)	6.2e+02 (7.0e+02)
f_6	**2.0e-02** (1.4e-01)	2.8e+00 (3.1e+00)
f_7	**4.6e-02** (1.4e-02)	1.1e-01 (5.2e-02)
f_8	4.0e+03 (4.5e+02)	**3.4e+00** (6.5e+00)
f_9	1.2e+02 (1.4e+01)	**2.8e+00** (1.4e+00)
f_{10}	**2.3e-02** (1.6e-01)	1.1e+00 (5.4e-01)
f_{11}	**2.1e-03** (4.8e-03)	4.5e-01 (3.0e-01)
f_{12}	3.2e-01 (6.1e-01)	**9.4e-03** (2.8e-02)
f_{13}	**2.6e-03** (1.4e-02)	5.2e-02 (7.3e-02)

mutation. High values of *CR*, normally recommended for solving nonseparable problems, will cause a higher degree of perturbation in DESP. In contrast, Fajfar-MDEVM and μJADE apply perturbation after selection and independently of *CR*. Using a *CR* value of 0.9 will cause DESP to rely much more on perturbation than Fajfar-MDEVM and μJADE. Ren et al (2010) used *CR* values as low as 0.05 for some problems. This may explain the poor performance observed in this study. The algorithm may benefit from a pool of mutations with different *CR* and *F* parameters similar to that described in Wang et al (2011).

Another observation is that MDEVM-Fajfar generally outperforms MDEVM. Especially on the multimodal functions, the modifications of Fajfar et al (2012) strongly benefit its performance. However, the performance on some of the unimodal benchmarks deteriorates.

Table 3 shows the comparison of μJADE to DE variants that use much larger populations. μJADE compares favourably to FSADE and DE/rand/1/bin both in terms of reliability and convergence speed. It is slightly more reliable than

Table 5 Comparison for D = 100. Mean over 50 independent runs.

f	R_{cr}-JADE-s4$_{NP=400}$		μJADE$_{NP=8}$	
	SR(%)	FE's Mean (Std)	SR(%)	FE's Mean (Std)
f_1	100	**1.1e+05** (1.2e+03)	100	1.5e+05 (7.1e+03)
f_2	100	**1.7e+05** (1.8e+03)	100	2.1e+05 (1.4e+04)
f_3	94	9.3e+05 (3.1e+04)	**100**	2.5e+06 (1.3e+05)
f_4	0	—	**100**	6.6e+06 (1.0e+05)
f_5	100	**8.3e+05** (1.6e+04)	100	2.1e+06 (8.4e+05)
f_6	100	**4.6e+04** (6.6e+02)	100	2.1e+05 (7.9e+04)
f_7	**100**	1.3e+05 (1.4e+04)	28	7.0e+06 (2.4e+06)
f_8	100	8.7e+05 (1.3e+04)	100	**5.6e+05** (7.4e+04)
f_9	100	1.1e+06 (7.5e+03)	100	**8.4e+05** (1.3e+05)
f_{10}	100	**1.6e+05** (1.6e+03)	100	2.7e+05 (1.6e+04)
f_{11}	100	**1.1e+05** (1.3e+03)	100	3.5e+05 (2.5e+05)
f_{12}	98	9.5e+04 (1.3e+03)	**100**	4.0e+05 (7.8e+04)
f_{13}	98	1.1e+05 (1.3e+03)	**100**	2.2e+05 (6.0e+04)
	92		**95**	

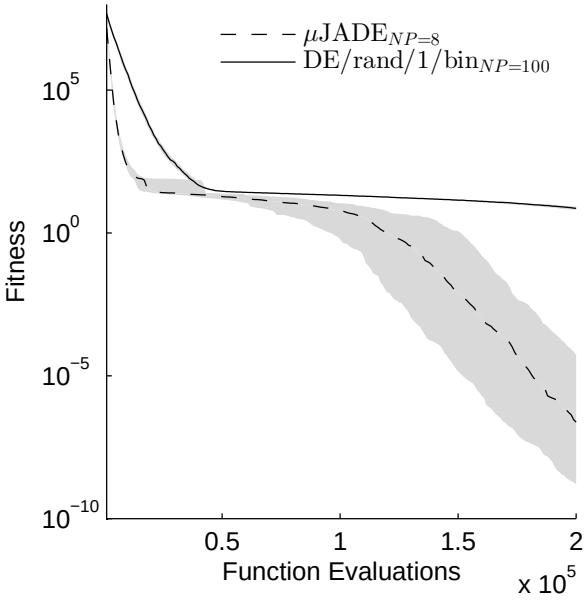

Fig. 1 Convergence plots of the median fitness for the Rosenbrock function at 30 dimensions. The grey shaded regions are bounded by the upper and lower quartile

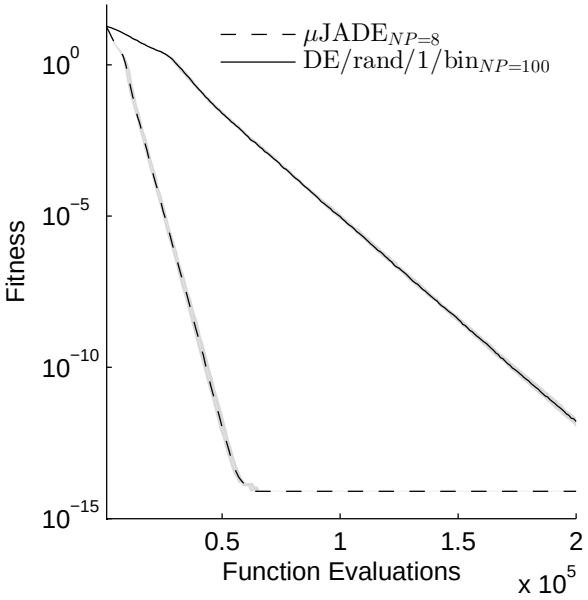

Fig. 2 Convergence plots of the median fitness for the Ackley function at 30 dimensions. The grey shaded regions are bounded by the upper and lower quartile

R_{cr}-JADE-s4 overall. However, μJADE is generally slower than R_{cr}-JADE-s4 on successful runs.

Generally, μJADE and the other small population variants show greater variance in the number of function evaluations on successful runs. Larger populations sample the fitness landscape more thoroughly following initialisation compared to smaller ones. This may explain the superior consistency of the DE algorithms using larger populations observed in this study.

The restart mechanism can increase the variance of function evaluations to solve a given problem. Waiting for 1 restart will add $\max(1000, 100D)$ function evaluations. Restarts are more likely on multimodal problems that cause premature convergence above the error threshold.

The progress in median fitness of μJADE and classical DE over the nonseparable Rosenbrock function at 30 dimensions is shown in Figure 1. The former shows greater interquartile range as indicated by the wider grey band. In contrast, for the Ackley function (Figure 2), both algorithms show a comparable interquartile range. This may be because μJADE is more sensitive to the initial population for the Rosenbrock function than for the Ackley function. Population initialisation and re-initialisation could be an important avenue of further enquiry for using small populations more effectively (Kazimipour et al, 2014).

Table 4 shows the performance difference between μJADE with current-by-rand-to-pbest/1 and μJADE with rand-to-pbest/1 for a population size of 8. It can be seen that much of the performance of μJADE can be attributed to the new mutation operator.

For $D = 100$, μJADE and R_{cr}-JADE-s4 are compared. The results are given in Table 5. μJADE is of comparable reliability to R_{cr}-JADE-s4 on the majority of problems despite the large difference in population size. The new mutation, current-by-rand-to-pbest/1 is exploratory in the early stages of optimisation: $\boldsymbol{x}_i$, in the most part, is displaced by difference vectors calculated independently of $\boldsymbol{x}_i$. This enables μJADE to be competitive even on multimodal optimisation problems as it can spend many generations exploring before showing greedier behaviour. Moreover, as perturbation occurs independently of CR, the parameter adaptation mechanism can work unhindered. However, the noisy quar-

tic function f_7 causes μJADE to converge extremely slowly and often not within the $100000D$ function evaluation limit.

Interestingly, R_{cr}-JADE-s4 is unable to solve f_4 whereas the original JADE with archive is (Zhang and Sanderson, 2009b). R_{cr}-JADE uses the selection operation given in Equation 8 rather than Equation 7 used in JADE. Since f_4 is only concerned with the maximum value in $|\boldsymbol{x}_i|$, the condition $f(\boldsymbol{x}_i) = f(\boldsymbol{u}_i)$ will occur frequently. It is possible that Equation 8 causes R_{cr}-JADE-s4 to behave too greedily in the absence of a fitness improvements. Though μJADE also uses the selection operation given in Equation 8, it is not necessarily greedy as discussed in Section 3.1.

6 Conclusion

DE with a small population, or micro DE, is useful for dynamic and resource constrained optimisation tasks. This paper presented a new DE algorithm, μJADE, that despite using a population size much smaller than the number of decision variables, is more reliable than some state-of-the-art DE algorithms using conventionally sized populations. Our results indicate that small population DE has a promising future.

In this paper the classical benchmark functions were used to test μJADE. Further work is needed to determine whether the same reliability can be achieved for more difficult benchmark problems, such as those that are shifted and rotated and dynamic optimisation benchmarks. In addition, due to its apparent reliability with a small population size, μJADE could be a suitable algorithm for incorporation into a cooperative coevolution scheme for large scale optimisation.

Acknowledgements The authors would like to thank the anonymous reviewers for helping to improve this paper.

References

Brest J, Maučec M (2011) Self-adaptive differential evolution algorithm using population size reduction and three strategies. Soft Comput 15:2157–2174

Choi T, Ahn C (2014) An adaptive differential evolution algorithm with automatic population resizing for global numerical optimization. In: Pan L, Păun G, Pérez-Jiménez M, Song T (eds) Bio-Inspired Computing—Theories and Applications, Communications in Computer and Information Science, vol 472, Springer, pp 68–72

Das S, Suganthan P (2011) Differential evolution: A survey of the state-of-the-art. IEEE Trans on Evolutionary Comput 15:4–31

Fajfar I, Puhan J, Tomažič S, Bűrmen A (2011) On selection in differential evolution. Elektrotehniški Vestnik 78:275–280

Fajfar I, Tuma T, Puhan J, Olenšek J, Bűrmen A (2012) Towards smaller populations in differential evolution. J of Microelectron, Electron Compon and Materials 42:152–163

Gong W, Cai Z (2013) Differential evolution with ranking-based mutation operators. IEEE Transactions on Cybernetics 43:2066–2081

Gong W, Cai Z, Ling C (2011a) De/bbo: A hybrid differential evolution with biogeography-based optimization for global numerical optimization. Soft Computing 15:645–665

Gong W, Cai Z, Ling C, Li H (2011b) Enhanced differential evolution with adaptive strategies for numerical optimization. IEEE Transactions on Systems, Man, and Cybernetics: Part B - Cybernetics 41:397–413

Gong W, Cai Z, Wang Y (2014) Repairing the crossover rate in adaptive differential evolution. Appl Soft Comput 15:149–168

Kazimipour B, Li X, Qin A (2014) Effects of population initialization on differential evolution for large scale optimization. In: 2014 IEEE Congress on Evolutionary Computation, pp 2404–2411

Lampinen J, Zelinka I (2000) On stagnation of the differential evolution algorithm. In: 6th International Conference on Soft Computing MENDEL, pp 76–83

Mallipeddi R, Suganthan P (2008) Empirical study on the effect of population size on differential evolution algorithm. In: The 2008 IEEE Congress on Evolutionary Computation, pp 3663–3670

Mendes R, Mohais A (2005) DynDE: a differential evolution for dynamic optimization problems. In: The 2005 IEEE Congress on Evolutionary Computation, vol 3, pp 2808–2815

Mininno E, Neri F, Cupertino F, Naso D (2011) Compact differential evolution. IEEE Trans on Evol Comput 15:32–54

Montgomery J, Chen S (2010) An analysis of the operation of differential evolution at high and low crossover rates. In: 2010 IEEE Congress on Evolutionary Computation, pp 1–8

Ren X, Chen Z, Ma Z (2010) Differential evolution using smaller population. In: 2010 Second International Conference on Machine Learning and Computing, pp 76–80

Ronkkonen J, Kukkonen S, Price K (2005) Real-parameter optimization with differential evolution. In: The 2005 IEEE Congress on Evolutionary Computation, vol 1, pp 506–513

Salehinejad H, Rahnamayan S, Tizhoosh H, Chen S (2014) Micro-differential evolution with vectorized random mutation factor. In: 2014 IEEE Congress on Evolutionary Computation, pp 2055–2062

Sharma H, Shrivastava P, Bansal J, Tiwari R (2014) Fitness based self adaptive differential evolution. In: Terrazas G,

Otero F, Masagosa A (eds) Nature Inspired Cooperative Strategies for Optimization (NICSO 2013), Studies in Computational Intelligence, vol 512, Springer, pp 71–84

Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J of Glob Optim 11:341–359

Teng N, Teo J, Hijazi M (2009) Self-adaptive population sizing for a tune-free differential evolution. Soft Comput 13:709–724

Teo J (2006) Exploring dynamic self-adaptive populations in differential evolution. Soft Comput 10:673–686

Wang X, Zhao S (2013) Differential evolution algorithm with self-adaptive population resizing mechanism. Math Probl in Eng 2013, Article ID 419372

Wang Y, Cai Z, Zhang Q (2011) Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans on Evol Comput 15:55–66

Yang M, Cai Z, Li C, Guan J (2013) An improved adaptive differential evolution algorithm with population adaptation. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp 145–152

Yao X, Liu Y, Lin G (1999) Evolutionary programming made faster. IEEE Trans on Evol Comput 3:82–102

Yu X, Huang D, Wang X, Jin Y (2008) DE-based neural network nonlinear model predictive control and its application for the pH neutralization reactor control. In: Chinese Control and Decision Conference 2008, pp 1597–1602

Zhang J, Sanderson A (2009a) Adaptive Differential Evolution: A Robust Approach to Multimodal Problem Optimization. Adaptation, Learning and Optimization, Springer, Berlin

Zhang J, Sanderson A (2009b) JADE: Adaptive differential evolution with optional external archive. IEEE Trans on Evolutionary Comput 13:945–958

Zhao S, Wang X, Chen L, Zhu W (2014) A novel self-adaptive differential evolution algorithm with population size adjustment scheme. Arab J for Science and Eng 39:6149–6174