GRAPH VISUALISATION BY CONCURRENT DIFFERENTIAL EVOLUTION

P. Krömer, M. Prílepok, J. Platoš, V. Snášel

Abstract: A representative dimensionality reduction is an important step in the analysis of real-world data. Vast amounts of raw data are generated by cyber-physical and information systems in different domains. They often feature a combination of high dimensionality, large volume, and vague, loosely defined structure. The main goal of visual data analysis is an intuitive, comprehensible, efficient, and graphically appealing representation of information and knowledge that can be found in such collections. In order to achieve an efficient visualisation, raw data need to be transformed into a refined form suitable for machine and human analysis. Various methods of dimension reduction and projection to low-dimensional spaces are used to accomplish this task. Sammon's projection is a well-known non-linear projection algorithm valued for its ability to preserve dependencies from an original high-dimensional data space in the low-dimensional projection space. Recently, it has been shown that bio-inspired real-parameter optimization methods can be used to implement the Sammon's projection on data from the domain of social networks. This work investigates the ability of several advanced types of the differential evolution algorithm as well as their parallel variants to minimize the error function of the Sammon's projection and compares their results and performance to a traditional heuristic algorithm.

Key words: *differential evolution, Sammon's projection, parallel processing*

Received: August 21, 2014 **DOI:** 10.14311/NNW.2015.25.019
Revised and accepted: October 13, 2014

1. Introduction

The Sammon's projection is a well-known non-linear projection algorithm interesting for real-world data analysis because of its emphasis on preserving the relative dependencies of data from the original, high-dimensional data space in the lower-dimensional projection space [26]. In contrast to widely used force-directed mapping methods, the Sammon's projection relies on the distances between particular data points (graph vertices) and aims at minimizing the changes that are introduced by the reduction of data dimension. Because of that, the distance measure

Pavel Krömer – Corresponding author, Michal Prílepok, Jan Platoš, Václav Snášel, IT4Innovations & Department of Computer Science, VŠB – Technical University of Ostrava, 17. listopadu 15, 708 33 Ostrava – Poruba, Czech Republic, E-mail: {pavel.kromer, michal.prilepok, jan.platos, vaclav.snasel}@vsb.cz

is a key attribute of the algorithm. The choice of an appropriate distance measure allows interesting modifications of the resulting projection (and eventually visualization) in order to obtain networks with desired properties.

The quality of the Sammon's projection of a data set is expressed by a projection error measure. Traditional error minimization techniques (e.g. steepest descent) are usually employed to minimize this error and obtain good low-dimensional representation of the original data. However, various metaheuristic algorithms have been recently shown to find good Sammon's projections as well. We have shown, that among them, the differential evolution algorithm (DE) has a great potential for obtaining the Sammon's projection of data sets from the domain of social networks and co-authorship [19].

This work extends our previous research on the use of differential evolution for the Sammon's projection in three ways. First, it compares a traditional and an advanced, adaptive, variant of the algorithm to asses the capability of a well-known self-adaptation scheme to improve results of the mapping. Second, it compares the quality of projections obtained by both DE variants in a sequential and a parallel configuration, respectively. Finally, it uses a set of simple, easily comprehensible graphs to conduct the experiments. That simplifies the clarity and interpretation of results.

The rest of this article is organized as follows: first, the notion of the Sammon's projection is introduced in detail in Section 2. Related work on the algorithms on Sammon's projection and its applications is presented in Section 3. The algorithms of differential evolution, self-adaptive differential evolution, and their parallel variants are summarized in Section 4. Extensive experiments with different variants of the investigated algorithms are presented in Section 5 Finally, major conclusions are drawn in Section 6.

2. Sammon's projection

The Sammon's projection [26] is one of several existing methods for projecting a data set from an original, high-dimensional data space to a space with lower dimensionality. It aims at preserving the between-point distances from the high-dimensional data space in the lower-dimensional projection space. This goal is achieved by minimizing an error criteria that penalizes the changes of distance between points in the original high-dimensional data space and in the low-dimensional projection space. For the purpose of visual data analysis, projections into two and three dimensional spaces (2D and 3D) are of utmost importance.

Suppose that we have a collection X with m data points $X = (X_1, X_2, ..., X_m)$, where each data point X_i is an n dimensional vector $X_i = \{x_{i1}, x_{i2}, ..., x_{in}\}$. At the same time we define a collection Y of m data points $Y = (Y_1, Y_2, ..., Y_m)$, where each data point Y_i is a d dimensional vector and $d < m$. The initial values of the coordinates in Y_i are chosen at random. The distance between vectors X_i and X_j is denoted d^*_{ij} while the distance between corresponding vectors Y_i and Y_j in the lower-dimensional space is denoted d_{ij}. Here, any distance measure can be used to evaluate d_{ij}. However, the distance measure suggested originally by Sammon is the traditional Euclidean metric [26]. In that case, d^*_{ij} and d_{ij} are defined in the following way:

$$d_{ij}^* = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2}, \tag{1}$$

$$d_{ij} = \sqrt{\sum_{k=1}^{d}(y_{ik} - y_{jk})^2}. \tag{2}$$

Projection error E (so-called Sammon's stress) measures how well the current configuration of m data points in the d-dimensional space matches the m points in the original m-dimensional space

$$E = \frac{1}{\sum_{i<j}^{m}[d_{ij}^*]} \sum_{i<j}^{m} \frac{[d_{ij}^* - d_{ij}]^2}{d_{ij}^*}. \tag{3}$$

In order to minimize the projection error, E, any minimization technique can be used. Sammon's original paper from 1969 [26] used widely known methods such as pseudo-Newton (steepest descent) minimization:

$$y'_{ik}(t+1) = y'_{ik}(t) - \alpha \frac{\frac{\partial E(t)}{\partial y'_{ik}(t)}}{\left|\frac{\partial^2 E(t)}{\partial y'_{ik}(t)^2}\right|}, \tag{4}$$

where y'_{ik} is the k-th coordinate of the data point's position y'_i in the projected low-dimensional space. The constant α is usually taken from a range 0.3 – 0.4, originally proposed by Sammon. However, this range is not optimal for all types of problems. Equation (4) can cause a problem at the inflection points, where second derivative is very small. Therefore the gradient descent may be used as an alternative minimization method.

In this article, Sammon's projection is performed by the means of the metaheuristic algorithms of differential evolution, self-adaptive differential evolution, and their parallel variants.

3. Related work

Sammon's projection has been realized by a number of algorithms and extensively used in many different areas. Often, the particular algorithm used to achieve the mapping has been developed with respect to the investigated problem. This section provides a brief overview of some algorithms for and applications of this projection method.

Almost two decades ago, Mao and Jain [21] studied the basic Sammon's projection algorithm and developed SAMANN, a feedforward neural network with unsupervised back-propagation learning algorithm which performs the same mapping as the basic Sammon's method. This approach addressed one of the weak

points of the original algorithm, i.e. the inability to process (project) new data without the need to re-build the mapping as a whole. Similarly to traditional feedforward neural networks, SAMANN was able to generalize the projection and process previously unknown data.

Dybowski et al. [9] used Sammon's projection for visualization of the convergence of genetic algorithms. In this work, binary strings were mapped to target values with the use of Hamming distance as a distance measure. The mapping was able to indicate the presence of multiple solutions during the iterative genetic algorithm process.

Sammon's projection is in [2] used for an analysis of the relationships between proteins based on their DNA sequences. The mapping was employed to visually separate different classes of the protein kinase family.

Pal and Eluri [23] used an improved Sammon's projection as a part of a structure preserving dimensionality reduction method. They modified the basic algorithm by a combination of sub-sampling with the original Sammon's algorithm and used it as a neural network-based mapper. The neural network was trained to mimic Sammon's projection using the backpropagation algorithm. The proposed approach was shown to work better than the method of Mao and Jain presented in [21] and be less expensive from the time and space point of view.

Backer et al. [6] evaluated several non-linear dimensionality reduction techniques including the Sammon's projection for unsupervised feature extractions. Sammon's projection performed well on high-dimensional data with limited number of points, but it failed on low-dimensional data sets with a very large number of points.

Kovacs and Abonyi [18] and Abonyi and Babuska [1] modified the Sammon's projection in order to visualize results of fuzzy clustering. Their modified algorithm, called FUZSAMM, reduced the computational complexity of distance evaluation in each iteration from $O(N^2)$ to $O(N \cdot c)$, where c is the number of clusters. The results, obtained with the help of the modified algorithm, identified cluster shapes more accurately and reliably in a much faster time. Both, Sammon's projection and FUZSAMM, outperformed the linear mapping algorithm of principal component analysis (PCA). The FUZSAMM algorithm was then successfully applied for the analysis of phase-space trajectories [12].

Kim and Moon [16] applied the Sammon's projection to the visualization of the genetic algorithm iterations and individuals. The work first presented the ability of the mapping to solve graph partitioning problems. Then, it was used to visualize the fitness landscape (distribution of the local optima according to the distance between chromosomes) and for a visualization of the genetic search process. The mapping proved its ability to work well and allowed a straightforward visual comparison of the results.

Paper [22] presents a modification of the standard Sammon's projection for sparse data. The proposed scheme was also able to alleviate the well-known curse of dimensionality problem. The key concept of the proposed modification lies in processing a local topology using a newly defined dissimilarity measure.

Some other metaheuristic approaches to Sammon's projection based on artificial neural networks were proposed by de Ridder and Duin in [7]. A parallel implementation of the SAMANN algorithm was introduced by Ivanikovas et al. [15].

The differential evolution was first used to perform Sammon's projection by Kromer, et al. [19]. A simple variant of the DE algorithm, /DE/rand/1, was used to find 2D projections of a co-authorship network. It was shown that the DE is able to minimize the error function of this specially-weighted social network more than the traditional method.

This work further investigates the ability of more advanced methods from the family of differential evolution to perform Sammon's projection.

4. Differential evolution

The DE is a versatile and easy to use stochastic evolutionary optimization algorithm [24]. It is a population-based optimizer that evolves a population of real encoded vectors representing the solutions to given problem. The DE was introduced by Storn and Price in 1995 [27, 28] and it quickly became a popular alternative to the more traditional types of evolutionary algorithms. It evolves a population of candidate solutions by iterative modification of candidate solutions by the application of the differential mutation and crossover [24]. In each iteration, so called trial vectors are created from current population by the differential mutation and further modified by various types of crossover operator. At the end, the trial vectors compete with existing candidate solutions for survival in the population.

4.1 Traditional differential evolution

The DE starts with an initial population of N real-valued vectors. The vectors are initialized with real values either randomly or so, that they are evenly spread over the problem space. The latter initialization leads to better results of the optimization [24]. During the optimization, the DE generates new vectors that are scaled perturbations of existing population vectors. The algorithm perturbs selected base vectors with the scaled difference of two (or more) other population vectors in order to produce the trial vectors. The trial vectors compete with members of the current population with the same index called the target vectors. If a trial vector represents a better solution than the corresponding target vector, it takes its place in the population [24].

The two most significant parameters of the DE are scaling factor and mutation probability [24]. The scaling factor $F \in [0, \infty]$ controls the rate at which the population evolves and the crossover probability $C \in [0, 1]$ determines the ratio of elements that are transferred to the trial vector from its opponent. The size of the population and the choice of operators are another important parameters of the optimization process.

The basic operations of the classic DE can be summarized using the following formulae [24]: the random initialization of the i-th vector with N parameters is defined by

$$x_j^i = \mathrm{rand}(b_j^{\mathrm{L}}, b_j^{\mathrm{U}}), \quad j \in \{1, \ldots, N\}, \tag{5}$$

where b_j^{L} is the lower bound of j-th parameter, b_j^{U} is the upper bound of j-th parameter, and $\mathrm{rand}(a, b)$ is a function generating a random number from the range $[a, b]$. A simple form of the standard differential mutation is given by

$$\mathbf{v}^i = \mathbf{v}^{r1} + F(\mathbf{v}^{r2} - \mathbf{v}^{r3}), \tag{6}$$

where F is the scaling factor and $\mathbf{v}^{r1}$, $\mathbf{v}^{r2}$, and $\mathbf{v}^{r3}$ are three random vectors from the population. The vector $\mathbf{v}^{r1}$ is the base vector, $\mathbf{v}^{r2}$ and $\mathbf{v}^{r3}$ are the difference vectors, and $\mathbf{v}^i$ is the trial vector. It is required that $i \neq r1 \neq r2 \neq r3$.

An alternative differential mutation which favours exploitation over exploration is defined by

$$\mathbf{v}^i = \mathbf{x}^{\text{best}} + F(\mathbf{v}^{r1} - \mathbf{v}^{r2}) \tag{7}$$

and combines two randomly chosen difference vectors with the best vector in population, $\mathbf{x}^{\text{best}}$.

The uniform (binomial) crossover that combines the target vector, $\mathbf{x}^i$, with the trial vector, $\mathbf{v}^i$, is given by

$$v^i_j = \begin{cases} v^i_j & \text{if } (\text{rand}(0,1) < C) \text{ or } j = j_{\text{rand}} \\ x^i_j, & \text{otherwise} \end{cases} \tag{8}$$

for each $j \in \{1, \ldots, N\}$. The random index j_{rand} is in the above selected randomly as $j_{\text{rand}} = \text{rand}(1, N)$. The uniform crossover replaces the parameters in $\mathbf{v}^i$ by the parameters from the target vector $\mathbf{x}^i$ with probability $1 - C$. The outline of the classic DE according to [11, 24] is summarized in Algorithm 1.

Algorithm 1: A summary of classic differential evolution

```
Initialize the population P consisting of M vectors using Equation (5);
Evaluate an objective function ranking the vectors in the population;
while Termination criteria not satisfied do
    Let G = number of current generation;
    for i ∈ {1, ..., M} do
        Differential mutation: Create trial vector v^i according to Equation (6);
        Validate the range of coordinates of v^i. Optionally adjust coordinates of v^i so,
        that it is valid solution to given problem;
        Perform uniform crossover. Select randomly one parameter j_rand in v^i and
        modify the trial vector using Equation (8);
        Evaluate the trial vector.;
        if trial vector v^i represent a better solution than target vector x^i then
            add v^i to P^{G+1}
        else
            add x^i to P^{G+1}
        end
    end
end
```

The classic DE has shown the ability to solve a wide range of problems. However, its performance in particular domains strongly relies on the selection of differential mutation and crossover operators as well as parameters F and C [25]. A number of self-adaptive DE variants was designed to mitigate this dependence. Among them, the Self-Adaptive Differential Evolution (SaDE) algorithm has shown good results for many types of tasks [10, 25].

4.2 Self-Adaptive differential evolution

The SaDE is based on probabilistic selection of trial vector generation strategy based on historical performance of different strategies, randomization of scaling factor, F, and adaptation of crossover probability, C.

4.2.1 Selection of trial vector generation strategy

In each generation, G, is for every target vector, $\mathbf{x}^i$, selected a trial vector generation strategy, s_k, from a pool of strategies, $S = \{s_1, s_2, \dots s_K\}$, with respect to strategy selection probability $p_{k,G}$. The strategy selection probability, $p_{k,G}$, is adapted on the basis of the number of successes (i.e. number of times a trial vector, $\mathbf{v}^i$, is better solution than target vector, $\mathbf{x}^i$) and failures (i.e. number of times $\mathbf{v}^i$ is worse solution than $\mathbf{x}^i$) of trial vectors generated by s_k during a fixed number of past generations known as *learning period*, LP. The algorithm stores the successes and failures of each strategy into success and failure memories (**SM** and **FM**), defined by:

$$\mathbf{SM} = \begin{pmatrix} ns_{1,G-LP} & ns_{2,G-LP} & \dots & ns_{K,G-LP} \\ ns_{1,G-LP+1} & ns_{2,G-LP+1} & \dots & ns_{K,G-LP+1} \\ \vdots & \vdots & \ddots & \vdots \\ ns_{1,G-1} & ns_{2,G-1} & \dots & ns_{K,G-1} \end{pmatrix}, \tag{9}$$

$$\mathbf{FM} = \begin{pmatrix} nf_{1,G-LP} & nf_{2,G-LP} & \dots & nf_{K,G-LP} \\ nf_{1,G-LP+1} & nf_{2,G-LP+1} & \dots & nf_{K,G-LP+1} \\ \vdots & \vdots & \ddots & \vdots \\ nf_{1,G-1} & nf_{2,G-1} & \dots & nf_{K,G-1} \end{pmatrix}, \tag{10}$$

where $ns_{k,G-LP}$ and $nf_{k,G-LP}$ stand for the number of successes and failures of strategy $s_k, k \in \{1, 2, \dots, K\}$ in generation $G - LP$, respectively. Informally, the memories represent a floating window of successes and failures of the strategies during last LP generations.

Strategy selection probabilities are then in each generation, G, $G > LP$, updated by

$$p_{k,G} = \frac{S_{k,G}}{\sum_{k=1}^{K} (S_{k,G})}, \tag{11}$$

$$S_{k,G} = \frac{\sum_{g=G-LP}^{G-1} (ns_{k,g})}{\sum_{g=G-LP}^{G-1} (ns_{k,g}) + \sum_{g=G-LP}^{G-1} (nf_{k,g})} + \epsilon, \tag{12}$$

where ϵ is a small constant (here, $\epsilon = 0.01$) employed to tackle cases with zero success rate [25]. Initial strategy selection probabilities are for the first LP generations set to be equal, i.e. $p_{k,G} = \frac{1}{K}, k \in \{1, 2, \dots, K\}$.

Trial vector generation strategies can include arbitrary combinations of differential mutation and crossover. The strategies used in this study are summarized in Fig. 1. The strategies /DE/rand/1/bin and /DE/rand/2/bin have slow convergence but strong exploration capability. /DE/rand-to-best/2/bin has fast convergence, especially for unimodal problems, but tends to get trapped in local optima and suffers from premature convergence. /DE/current-to-rand/1 is a rotation invariant type of DE that has good efficiency for rotated problems [25].

/DE/rand/1/bin:

$$v_j^i = \begin{cases} v_j^{r1} + F(v_j^{r2} - v_j^{r3}), & \text{if rand}(0,1) < C \text{ or } j = j_{\text{rand}} \\ x_j^i, & \text{otherwise} \end{cases}, \tag{13}$$

/DE/rand-to-best/2/bin:

$$v_j^i = \begin{cases} x_j^i \ + F(x_j^{\text{best}} - x_j^i) + \\ \quad + F(v_j^{r1} - v_j^{r2}) + F(v_j^{r3} - v_j^{r4}), & \text{if rand}(0,1) < C \text{ or } j = j_{\text{rand}} \\ x_j^i, & \text{otherwise} \end{cases}, \tag{14}$$

/DE/rand/2/bin:

$$v_{i,j}^t = \begin{cases} v_j^{r1} + F(v_j^{r2} - v_j^{r3}) + F(v_j^{r4} - v_j^{r5}), & \text{if rand}(0,1) < C \text{ or } j = j_{\text{rand}} \\ x_j^i, & \text{otherwise} \end{cases}, \tag{15}$$

/DE/current-to-rand/1:

$$\mathbf{v}^i = \mathbf{x}_j^i + \kappa(\mathbf{v}^{r2} - \mathbf{x}^i) + F(\mathbf{v}^{r2} - \mathbf{v}^{r3}), \tag{16}$$

where $\mathbf{x}^{\text{best}}$ is the best solution found so far and κ is a random parameter generated for each trial vector, $\mathbf{v_i}$, using $\kappa = \text{rand}(0,1)$.

Fig. 1 *SaDE trial vector generation strategies.*

4.2.2 Randomization of scaling factor

The scaling factor, F, is in SaDE selected for each trial vector randomly from a normal distribution, $\mathcal{N}$, with mean 0.5 and standard deviation 0.3 [25]:

$$F_{i,G} = \mathcal{N}(0.5, 0.3). \tag{17}$$

Scaling factors drawn from such distribution fall in 99.7% of cases into the range $[-0.4, 1.4]$ allowing for both, exploitation (small F) and exploration (large F) [25].

4.2.3 Adaptation of crossover probability

The value of crossover probability is an important problem-dependent parameter that has a major impact on algorithm performance [25]. Crossover probability is in SaDE for each generation, G, and each trial vector generation strategy, s_k, generated using

$$C_{k,G} = \begin{cases} \mathcal{N}(0.5, 0.1), & \text{if } G < LP, \\ \mathcal{N}(Cm_{k,G}, 0.1), & \text{otherwise}, \end{cases} \tag{18}$$

where $Cm_{k,G}$ is the mean of the (normal) random distribution of Cs for strategy s_k in generation G, and LP is the learning period. The values of $C_{k,G}$ are based

on the *Cm memory*, **CmM**, that stores the crossover probabilities that were used in the past LP generations for creation of successful trail vectors.

$$\mathbf{CmM} = \begin{pmatrix} C_{1,G-LP} & C_{2,G-LP} & \dots & C_{K,G-LP} \\ C_{1,G-LP+1} & C_{2,G-LP+1} & \dots & C_{K,G-LP+1} \\ \vdots & \vdots & \ddots & \vdots \\ C_{1,G-1} & C_{2,G-1} & \dots & C_{K,G-1} \end{pmatrix} \tag{19}$$

During the first LP generations, Cs are only stored into **CmM**. Only after that, when the memory is filled by the first LP Cs used by each strategy, it is used to generate new, strategy specific, values of the C parameter.

The SaDE algorithm [10, 25] is summarized in Algorithm 2.

Algorithm 2: A summary of self-adaptive differential evolution

```
Initialize the population P consisting of M vectors using Equation (5);
Initialize trial vector strategy selection probabilities p_k = 1/K, k ∈ {1, 2, ..., K};
Evaluate an objective function ranking the vectors in the population;
while Termination criteria not satisfied do
    Let G = number of current generation;
    for k ∈ {1, ..., K} do
        ns_{k,G-LP} = nf_{k,G-LP} = 0;
    end
    for i ∈ {1, ..., M} do
        Select trial vector generation strategy, s_{k,G} from the pool of strategies S;
        Set F = N(0.5, 0.3);
        Generate C_{k,G} according to eq. (18);
        Create trial vector v^i according to selected strategy s_{k,G};
        Validate the range of coordinates of v^i. Optionally adjust coordinates of v^i so,
        that it is a valid solution to given problem;
        Evaluate the trial vector v^i;
        if trial vector v^i represent a better solution than target vector x^i then
            Add v^i to P^{G+1};
            Increment ns_{k,G-LP};
            Store C_{k,G} to CmM_k;
        else
            Add x^i to P^{G+1};
            Increment nf_{k,G-LP};
        end
        if G > LP then
            for k ∈ {1, ..., K} do
                Update p_{k,G+1} according to Equation (11);
                Set Cm_{k,G+1} to median(CmM_k);
            end
        end
    end
end
```

4.3 Parallel differential evolution

Traditional evolutionary algorithms develop only a single population of candidate solutions. The use of multiple sub-populations is a well-known extension of pop-

ulational metaheuristics that has been shown useful for both serial and parallel variants of these algorithms [3, 4]. In general, parallel evolutionary metaheuristics evolve multiple candidate solutions or populations of solutions side by side. There are three major parallel evolutionary algorithm variants:

- the *farming model* performs all operations not requiring the population as a whole in parallel, e.g. individual evaluation, crossover, and mutation. Operations involving the population as a whole (e.g. parent selection or migration) are performed by some main process of the algorithm [4]. This strategy, employing a single population of candidate solutions, is also called *panmixia* [3],
- the *island (migration) model* performs parallel evolution of isolated populations. The populations meet each other at a certain point of time (i.e. generation) and exchange alleles according to a certain migration strategy [3, 4],
- the *diffusion (cellular) model* considers individual members of the population and their neighbors. All individuals in the population are active. They seek partners among their neighbors to form new solutions. The genetic information then spreads in a diffusion-like manner. An important factor for the diffusion model is the structure and shape of the neighborhood [3, 4].

The distributed *island model* is considered one of the best strategies for parallel multipopulational metaheuristics under which a number of sub-populations evolves side-by-side independently. The existence of multiple populations introduces next level of complexity and parallelism into the metaheuristic search and optimization process. The sub-populations on each island are usually initialized with different random values and tend to follow different search trajectory and explore different areas of the fitness landscape [29]. Moreover, the algorithm instances executed on each island can be differently parametrized in order to exploit a variety of search strategies at once [4, 29].

Communication strategy and data exchange topology is an important aspect of distributed multipopulational algorithms [3, 13, 29]. The *topology* defines logical links between the sub-populations [3], i.e the way candidate solutions migrate between the islands. Common topologies include *fully connected* and *ring* topology (see Fig. 2). Other island model parameters include [3]: *migration rate*, i.e. the number of candidate solutions exchanged between islands; *migration period (gap)*,

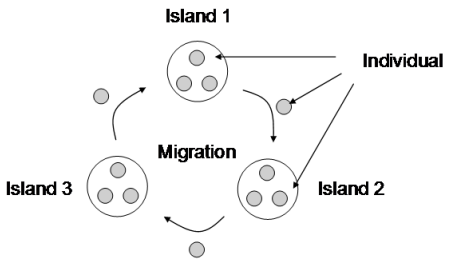


Fig. 2 *Island model of populational metaheuristics with the* ring *topology [5].*

i.e. the number of generations between migrations; and the migrant *selection and replacement strategy*.

The structure of the distributed multipopulational metaheuristic algorithms is suitable for parallelization on modern multicore and manycore platforms as well as in the environment of hybrid multi-CPU/GPU high performance computing (HPC) nodes and clusters of HPC nodes [3]. The structure is highly flexible and can be mapped to a variety of parallel architectures. Each island can be executed by a separate CPU thread or hosted on a separate compute node. The evolution of each island can be further parallelized and each sub-population can be executed using e.g. the master-slave model or the farming model.

The DE has a number of variants and modifications. They have been developed and tuned for different application areas. However, no ultimate DE variant, outperforming all others in all application scenarios, has been found. The situation has been paraphrased by the "no free lunch" theorem which states [30] that for any algorithm, including the DE, any increased performance over one class of problems is paid for in performance over another class. In another words, any algorithm performing exceptionally well in one problem (i.e. in one fitness landscape) will probably perform less well in a different problem characterized by a differently shaped fitness landscape. This observation encourages experimental evaluation of different algorithms for various tasks.

5. Experiment

This study investigates the ability of a traditional differential evolution, /DE/rand/1, self-adaptive differential evolution, and their parallel variants, to perform Sammon's projection. The evaluation is carried out on a collection of simple data sets that with a clear structure. This allows to assess the ability of considered algorithms to find a faithful low-dimensional representation of the high-dimensional data.

5.1 DE for Sammon's projection

The DE for Sammon's projection is fairly simple. A projection of n objects to a d-dimensional space is represented by an $n \cdot d$ dimensional candidate vector consisting of the coordinates of the n objects in the lower-dimensional projection space. The coordinates are randomly initialized and modified by the application of standard differential mutation and uniform crossover operators, as defined in Section 4.

Four variants of the DE algorithm were considered in this research. Their parameters are detailed in Tab. I. Fixed parameters were in all cases set on the basis of extensive initial trials. The parallel variants of the algorithms were setup so that the total number of candidate vectors in all cases was the same, i.e. the number of fitness function evaluations within a single generation was for each considered algorithm the same. All investigated DE variants were executed for 5,000 generations, so that the number of fitness function evaluations was the same as the number of iterations performed by the heuristic method. All algorithms were used to find projection of original data sets into two-dimensional space (2D). Because of the

Algorithm	Parameters
heuristic	steepest descent algorithm with 500,000 iterations,
DE	/DE/rand/1 differential evolution with scaling factor $F = 0.1$, crossover probability $C = 0.5$, and population size 100,
SaDE	self-adaptive differential evolution with population size 100,
paraDE	parallel /DE/rand/1 implementing the *island model* strategy; 4 islands with population of 25 candidate vectors each, ring topology, migration rate of 5, and migration gap 10,
paraSaDE	parallel SaDE implementing the *island model* strategy; 4 islands with population of 25 candidate vectors each, ring topology, migration rate of 5, and migration gap 10.

Tab. I *Algorithms and parameters.*

stochastic nature of the DE-based methods, all experimental runs were repeated 30 times and presented results are averages over the 30 runs.

5.2 Test data

Several data sets were used in conducted experiments. First five data sets were randomly generated artificial graphs with selected properties:

Bipartite is a bipartite graph with 5 and 10 nodes in each partition. Each node in a partition has a connection (edge, arc) to all nodes in the other group. The graph is symmetric.

Circle is a circular graph with 20 nodes. Each node has a connection to its closest left and right neighbor.

Complete is a complete graph with ten nodes.

Petersen is the well known Petersen graph - a pentagram in a pentagon. It was constructed in 1898 by Julius Petersen as the smallest bridgeless cubic graph with no three-edge-coloring [14]. The test graph contains ten nodes and 15 edges.

Tree is a tree graph with height 3 and four child nodes connected to each parent node. In total, it contains 21 nodes and 20 edges.

The other three data sets are well-known small social networks. All of them were downloaded from the UC Irvine (UCI) Networks Data repository [8].

Dolphins is an undirected social network of frequent associations between 62 dolphins in a community living in Doubtful Sound, New Zealand [20].

Karate is a social network of friendships among 34 members of a karate club at a US university in the 1970 [31].

Les Miserables is a co-appearance network describing the interactions between characters in the novel Les Miserables by Victor Hugo. In this graph, the nodes represent characters, and the edges connect any pair of characters that appear in the same chapter of the book. The weights of the edges correspond to the number of such co-appearances [17].

For Sammon's projection, each graph is represented by a distance matrix. In the presented experiments, a standard graph distance, based on the sum of weights of arcs of the shortest path connecting every two vertices, is used. Because all test graphs are strongly connected, all distances are defined.

All these graphs have no direct spatial representation, just the topological connections between nodes. Therefore, their visualisations is a projection into defined dimension. In all cases, the distance between nodes is taken as a distance measure. We do not need any further processing of these distances such as inversion or normalization.

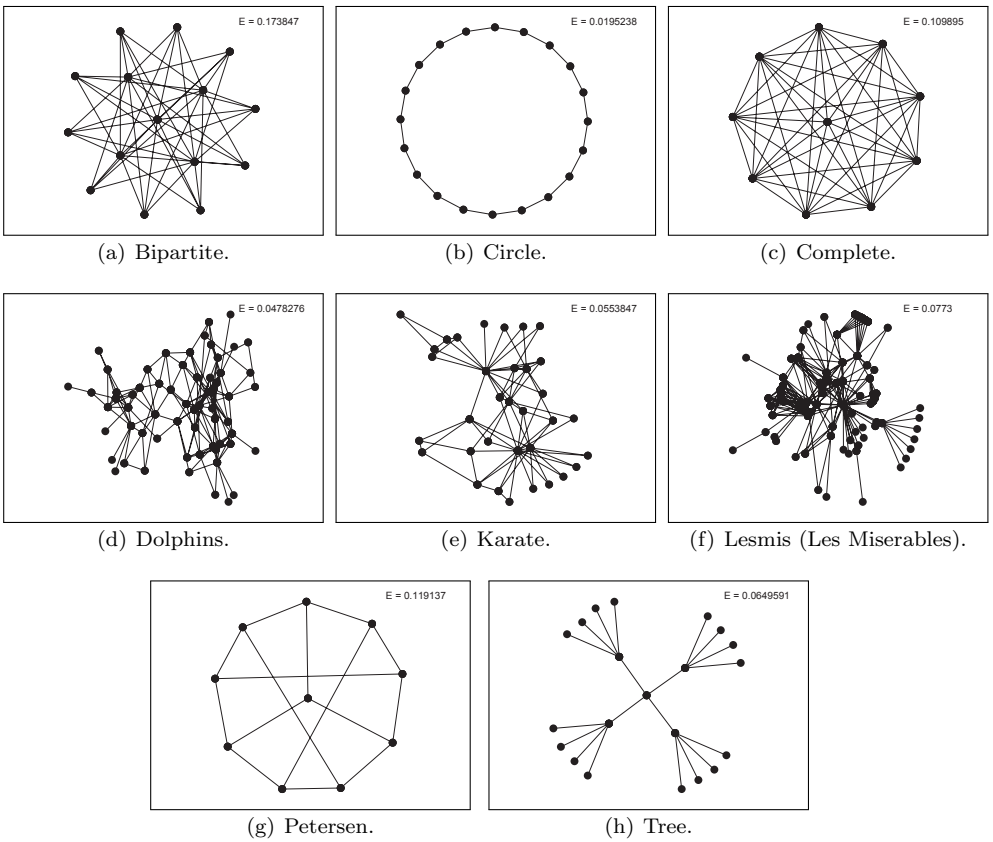
Visualization of these graphs using the standard heuristic method for Sammon's projection is depicted in Fig. 3.

5.3 Results and discussion

The results of conducted experiments are summarized in Tab. II and Tab. III, respectively. Tab. II shows the error of the best projection found by each algorithm and Tab. III shows the error of the average projection found during 30 runs of the metaheuristic algorithms. In each table, the best results (i.e. projection with lowest error) are typed in **bold**. Tab. II clearly demonstrates that the DE has found the best solution for 3 graphs, the SaDE for 6 graphs, and the paraSaDE for 7 out of 8 test graphs. Interestingly, the parallel version of the traditional DE, paraDE, was less successful than pure DE with a single population. The best projections, found by the metaheuristic algorithms, had for all graphs lower error than the ones found by the heuristic algorithm.

	heuristic	Sequential version		Parallel version	
graph	method	DE	SaDE	paraDE	paraSaDE
bipartite	0.1738470	**0.1630710**	**0.1630710**	0.1631370	**0.1630710**
circle	0.0195238	0.0170726	**0.0170573**	0.0183828	**0.0170573**
complete	0.1098950	**0.1098800**	**0.1098800**	0.1098990	**0.1098800**
dolphins	0.0478276	0.0627928	**0.0437652**	0.1513750	0.0437968
karate	0.0553847	0.0563546	0.0542477	0.0697328	**0.0542097**
lesmis	0.0773000	0.1034190	0.0730517	0.5128820	**0.0722459**
petersen	0.1191370	**0.1135110**	**0.1135110**	0.1135710	**0.1135110**
tree	0.0649591	0.0621328	**0.0621303**	0.0627946	**0.0621303**

Tab. II *Sammon's stress of the best projections found by each algorithm.*



(a) Bipartite. (b) Circle. (c) Complete.

(d) Dolphins. (e) Karate. (f) Lesmis (Les Miserables).

(g) Petersen. (h) Tree.

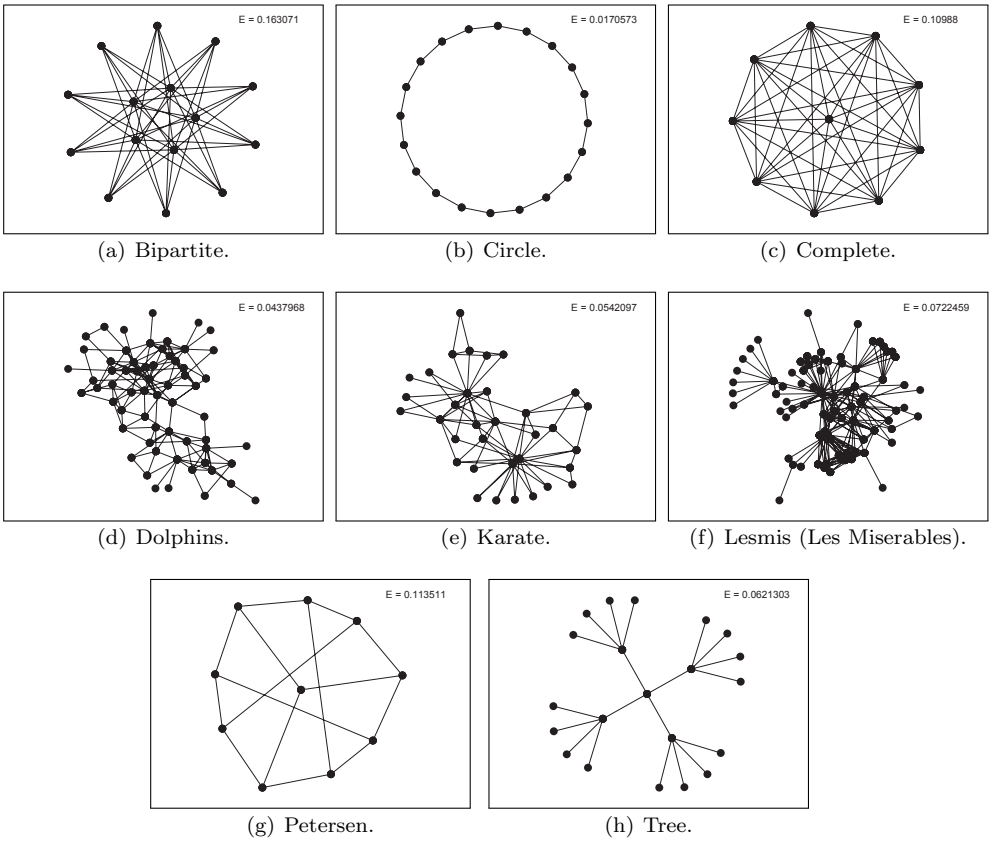
Fig. 3 *Sammon's projection of test graphs in 2D projection space obtained by the heuristic method.*

Tab. III summarizes for each graph the average error of projections found by each algorithm in 30 experimental runs. The parallel version of the SaDE algorithm, paraSaDE, comes clearly best also in this statistics. Average projections found by the paraSaDE algorithm featured the lowest error for 5 out of 8 test graphs. In the case of two graphs (*complete*, *karate*), none of the investigated metaheuristic methods has found projections with average error lower than the error of the projections found by the heuristic method. It can be also seen that the average error of projections found by SaDE and paraDE were in some cases significantly worse (i.e. larger) than those found by SaDE, paraDE, and the heuristic method. Together with the results shown in Tab. II, it suggests that paraSaDE is the best of investigated algorithms. In the best case, it outperforms the heuristic method and is equal or better than other investigated DE-based algorithms (with a single exception for the graph *dolphins*).

In the average case, paraSaDE was able to find a better projection than the heuristic method for 6 out of 8 graphs. However, it should be noted that for one graph (*petersen*), the traditional DE has found a better projection than paraSaDE.

graph	heuristic method	Sequential version DE	SaDE	Parallel version paraDE	paraSaDE
bipartite	0.1738470	0.1630950	0.833844	0.2557920	**0.1630890**
circle	0.0195238	0.0271613	0.193350	0.0614484	**0.0170573**
complete	**0.1098950**	0.1099760	0.179209	0.1410750	0.1101930
dolphins	0.0478276	0.2886970	3.209840	1.0919900	**0.0469729**
karate	**0.0553847**	0.0862950	6.373620	1.4604100	0.0566264
lesmis	0.0773000	0.6068490	6.276180	2.5381100	**0.0758365**
petersen	0.1191370	**0.1148880**	1.092510	0.1229040	0.1158580
tree	0.0649591	0.0638157	1.059430	0.2364170	**0.0621303**

Tab. III *Sammon's stress of average projections found by each algorithm.*



(a) Bipartite. (b) Circle. (c) Complete.

(d) Dolphins. (e) Karate. (f) Lesmis (Les Miserables).

(g) Petersen. (h) Tree.

Fig. 4 *Sammon's projection of test graphs in 2D projection space obtained by SaDE.*

Both mentioned algorithms have for the graph *petersen* found on average a better projection than that generated by the heuristic method.

Visual illustrations of the projections obtained by the most successful algorithm, SaDE, are shown in Fig. 4. The visualization clearly demonstrates that the char-

acteristic properties of the original data, e.g. the regular structure of the *complete* and *bipartite* data sets, were preserved also in the projection space. The visualization of the *bipartite* graph also shows how the minimization of the Sammon's stress causes that the points from the original partitions are grouped together and the graph is symmetric. The change in the layout of the *petersen* graph is only subtle (we note that the projection error considers only relative distances between the points, not their absolute position).

6. Conclusions

This article investigated the ability of four variants of a well-known metaheuristic real-parameter optimization method, the differential evolution, to perform Sammon's projection of data. This non-linear projection method is an important tool for efficient analysis of real-world data and graphs. Due to its ability to preserve data dependencies and generally properties of the data from the original high-dimensional space, it is a suitable method for processing of data with high dimensionality and large volume. The low-dimensional projections, retaining the important topological properties of the original data, can be further investigated by the means of traditional analytical, machine learning, and pattern recognition methods that are often not feasible for application in the original high-dimensional spaces.

Sammon's projection is realized by a minimization of an error (penalty) function. In this work, a set of bio-inspired metaheuristic methods from the wide family of differential evolution was used to minimize this function. In particular, traditional, self-adaptive, and parallel (multipopulational) DEs were implemented and evaluated. The algorithms were compared to each other and also to a well-known heuristic method for Sammon's projection based on the steepest descent principle. A thorough empirical evaluation on a series of simple but clearly structured data sets has shown that a parallel, self-adaptive, DE can deliver the best projections in terms of the projection error, E, for most investigated data sets.

This a promising result, especially with regard to the constantly increasing availability and affordability of parallel multicore and distributed platforms, floating point accelerators, libraries, and development tools. It suggests, that parallel computing can be used not only to speed-up execution and/or increase the dimensionality of processed data, but also to find qualitatively better Sammon's projections of existing data sets with lower values of the error function.

Acknowledgement

Acknowledgement This work was supported by the IT4Innovations Centre of Excellence project (CZ.1.05/1.1.00/02.0070), funded by the European Regional Development Fund and the national budget of the Czech Republic via the Research and Development for Innovations Operational Programme.

References

[1] ABONYI J., BABUSKA R. Fuzzsam - visualization of fuzzy clustering results by modified Sammon mapping. In: *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2004)*, Budapest, Hungary. Piscataway, NJ: IEEE, 2004, pp. 365–370, doi: 10.1109/FUZZY.2004.1375750.

[2] AGRAFIOTIS D.K. A new method for analyzing protein sequence relationships based on Sammon maps. *Protein Sci.* 1997, 6(2), pp. 287–293, doi: 10.1002/pro.5560060203.

[3] ALBA E., LUQUE G., NESMACHNOW S. Parallel metaheuristics: recent advances and new trends. *International Transactions in Operational Research.* 2013, 20(1), pp. 1–48, doi: 10.1111/j.1475-3995.2012.00862.x.

[4] ALBA E., TALBI E.-G., LUQUE G., MELAB N. Metaheuristics and Parallelism. In: E. Alba, ed. *Parallel Metaheuristics: A New Class of Algorithms.* New York, NY: John Wiley & Sons, Inc., 2005, pp. 79–103, doi: 10.1002/0471739383.ch4.

[5] ANDO J., NAGAO T. Fast evolutionary image processing using Multi-GPUs. In: *IEEE International Conference on Systems, Man and Cybernetics (ISIC 2007)*, Montreal, QC. Piscataway, NJ: IEEE, 2007, pp. 2927–2932, doi: 10.1109/ICSMC.2007.4413831.

[6] BACKER S.D., NAUD A., SCHEUNDERS P. Non-linear dimensionality reduction techniques for unsupervised feature extraction. *Pattern Recognition Letters.* 1998, 19(8), pp. 711–720, doi: 10.1016/S0167-8655(98)00049-X.

[7] DE RIDDER D., DUIN R.P. Sammon's mapping using neural networks: A comparison. *Pattern Recognition Letters.* 1997 18(11–13), pp. 1307–1316. doi: 10.1016/S0167-8655(97)00093-7.

[8] DUBOIS C.L. UCI network data repository [online]. University of California Irvine, 2008 [viewed 2014-01-17]. Available from: `http://networkdata.ics.uci.edu`

[9] DYBOWSKI R., COLLINS T.D., WELLER P.R. Visualization of binary string convergence by Sammon mapping. In: L.J. Fogel, P.J. Angeline, and T. Baeck, eds. *Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP96)*, San Diego, CA. Cambridge, MA: MIT Press, 1996, pp. 377–383.

[10] ELSAYED S.M., SARKER R.A., ESSAM D.L. An improved self-adaptive differential evolution algorithm for optimization problems. *IEEE Trans. Industrial Informatics.* 2013, 9(1), pp. 89–99, doi: 10.1109/TII.2012.2198658.

[11] ENGELBRECHT A. *Computational Intelligence: An Introduction.* 2nd ed. New York, NY: Wiley, 2007.

[12] FEIL B., BALASKO B., ABONYI J. Visualization of fuzzy clusters by fuzzy Sammon mapping projection: application to the analysis of phase space trajectories. *Soft Computing.* 2007, 11(5), pp. 479–488, doi: 10.1007/s00500-006-0111-5.

[13] GUAN W., SZETO K.Y. Topological effects on the performance of island model of parallel genetic algorithm. In: I. Rojas, G. Joya, and J. Cabestany, eds. *Advances in Computational Intelligence - Proceedings of the 12th International Work-Conference on Artificial Neural Networks (IWANN 2013)*, Puerto de la Cruz, Tenerife, Spain. Lecture Notes in Computer Science (series title). Berlin, Heidelberg: Springer, 2013, 7903, pp. 11–19, doi: 10.1007/978-3-642-38682-4_2.

[14] HOLTON D., SHEEHAN J. *The Petersen graph.* Australian Mathematical Society Lecture Notes. vol. 7, Cambridge, UK: Cambridge University Press, 1993.

[15] IVANIKOVAS S., MEDVEDEV V., DZEMYDA G. Parallel realizations of the SAMANN algorithm. In: B. Beliczynski, A. Dzielinski, M. Iwanowski, B. Ribeiro, eds. *Adaptive and Natural Computing Algorithms*, Warsaw, Poland. Lecture Notes in Computer Science (series title). Berlin, Heidelberg: Springer, 2007, 4432, pp. 179–188, doi: 10.1007/978-3-540-71629-7_21.

[16] KIM Y.-H., MOON B.-R. New usage of Sammon's mapping for genetic visualization. In: E. Cantú-Paz, J.A. Foster, K. Deb, L.D. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A.

Potter, A.C. Schultz, K.A. Dowsland, N. Jonoska, J. Miller, eds. *Genetic and Evolutionary Computation (GECCO 2003)*, Chicago, IL. Lecture Notes in Computer Science (series title). Berlin, Heidelberg: Springer, 2003, 2723, pp. 1136–1147, doi: 10.1007/3-540-45105-6_122.

[17] KNUTH D.E. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Reading, MA: Addison-Wesley, 1993.

[18] KOVÁCS A., ABONYI J. Visualization of fuzzy clustering results by modified Sammon mapping. In: *3rd International Symposium of Hungarian Researchers on Computational Intelligence*, Budapest, Hungary. Budapest: Budapest Polytechnic, 2002, pp. 177–188.

[19] KROMER P., KUDELKA M., SNASEL V., RADVANSKY M., HORAK Z. Computing Sammon's projection of social networks by differential evolution. In: *IEEE 28th International Conference on Advanced Information Networking and Applications (AINA 2014)*, Victoria, BC. Piscataway, NJ: IEEE, 2014, pp. 1001–1006, doi: 10.1109/AINA.2014.121.

[20] LUSSEAU D., SCHNEIDER K., BOISSEAU O.J., HAASE P., SLOOTEN E., DAWSON S.M. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*. 2003, 54, pp. 396–405.

[21] MAO J., JAIN A. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*. 1995, 6(2), pp. 296–317, doi: 10.1109/72.363467.

[22] MARTIN-MERINO M., MUNOZ A. A new Sammon algorithm for sparse data visualization. In: *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, Cambridge, UK. Piscataway, NJ: IEEE Computer Society, 2004, pp. 477–481, doi: 10.1109/ICPR.2004.1334168.

[23] PAL N., ELURI V. Two efficient connectionist schemes for structure preserving dimensionality reduction. *IEEE Transactions on Neural Networks*. 1998, 9(6), pp. 1142–1154, doi: 10.1109/72.728358.

[24] PRICE K.V., STORN R.M., LAMPINEN J.A. *Differential Evolution A Practical Approach to Global Optimization*. Natural Computing Series. Berlin, Germany: Springer-Verlag, 2005.

[25] QIN A., HUANG V., SUGANTHAN P. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*. 2009, 13(2), pp. 398–417, doi: 10.1109/TEVC.2008.927706.

[26] SAMMON J.W. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.* 1969, 18(5), pp. 401–409, doi: 10.1109/T-C.1969.222678.

[27] STORN R. Differential evolution design of an IIR-filter. In: *Proceeding of the IEEE Conference on Evolutionary Computation ICEC*, Nagoya, Japan. Piscataway, NJ: IEEE, 1996, pp. 268–273, doi: 10.1109/ICEC.1996.542373.

[28] STORN R., PRICE K. Tech. rep. TR-95-012: *Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces* [online]. California: University of California Berkeley, 1995 [viewed 2014-03-18]. Available from: `ftp://ftp.icsi.berkeley.edu/pub/techreports/1995/tr-95-012.pdf`

[29] WHITLEY D., RANA S., HECKENDORN R.B. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*. 1998, 7, pp. 33–47, doi: 10.1.1.36.7225.

[30] WOLPERT D.H., MACREADY W.G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*. 2002, 1(1), pp. 67–82, doi: 10.1109/4235.585893.

[31] ZACHARY W.W. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*. 1977, 33(4), pp. 452–473.