合肥學院
HEFEI UNIVERSITY

Metaheuristic Optimization

13. Differential Evolution

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

website

Section Outline

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover
- Population treatment

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover
- Population treatment:
  1. each offspring competes with its direct parent and

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover
- Population treatment:
  1. each offspring competes with its direct parent and
  2. replaces it if and only if it has better objective values

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover
- Population treatment:
  1. each offspring competes with its direct parent and
  2. replaces it if and only if it has better objective values
  3. i.e., some local form of selection!

Introduction

- Simple Evolutionary Algorithm for numerical optimization [1–8]
- Search space: vectors of real numbers, i.e., $\mathbb{G} = \mathbb{R}^n$
- Developed by Storn and Price [9] in the mid-1990s
- Many different variants, but main idea: ternary recombination operator instead of mutation and binary crossover
- Population treatment:
  1. each offspring competes with its direct parent and
  2. replaces it if and only if it has better objective values
  3. i.e., some local form of selection!
- The rest: same as in simple population EA

Section Outline

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:
  1. Changing parameters over time independently from the search process (e.g., Simulated Annealing [10–17])

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:
  1. Changing parameters over time independently from the search process (e.g., Simulated Annealing [10–17])
  2. Changing parameters according to some policy which uses information about the progress of the search (e.g., Evolution Strategy with $1/5$th rule [18, 19], exogeneous method)

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:
  1. Changing parameters over time independently from the search process (e.g., Simulated Annealing [10–17])
  2. Changing parameters according to some policy which uses information about the progress of the search (e.g., Evolution Strategy with $1/5$th rule [18, 19], exogeneous method)
  3. Encoding parameters as additional variables in individual records and let the evolutionary algorithm adapt them via selection and reproduction (e.g., Evolution Strategy with endogeneous method [18, 19])

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:
  1. Changing parameters over time independently from the search process (e.g., Simulated Annealing [10–17])
  2. Changing parameters according to some policy which uses information about the progress of the search (e.g., Evolution Strategy with $1/5$th rule [18, 19], exogeneous method)
  3. Encoding parameters as additional variables in individual records and let the evolutionary algorithm adapt them via selection and reproduction (e.g., Evolution Strategy with endogeneous method [18, 19])
- Only the latter two are self-adaptation methods, only they allow for an algorithm behavior that is not anticipated by the designer

Self-Adaptiation through Self-Organization

- So far, we know three ways for adaptation:
  1. Changing parameters over time independently from the search process (e.g., Simulated Annealing [10–17])
  2. Changing parameters according to some policy which uses information about the progress of the search (e.g., Evolution Strategy with $1/5$th rule [18, 19], exogeneous method)
  3. Encoding parameters as additional variables in individual records and let the evolutionary algorithm adapt them via selection and reproduction (e.g., Evolution Strategy with endogeneous method [18, 19])
- Only the latter two are self-adaptation methods, only they allow for an algorithm behavior that is not anticipated by the designer
- Today, we learn a fourth method: self-adaptation via self-organization without parameters!

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps
- As the population converges, the candidate solutions are located closer to each other

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps
- As the population converges, the candidate solutions are located closer to each other
- The distances $(g_1 - g_2)$ decrease

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps
- As the population converges, the candidate solutions are located closer to each other
- The distances $(g_1 - g_2)$ decrease $\Rightarrow$ the search steps get smaller, too

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps
- As the population converges, the candidate solutions are located closer to each other
- The distances $(g_1 - g_2)$ decrease $\Rightarrow$ the search steps get smaller, too
- Very simple way to perform self-adaptation without needing any additional parameter or operation!

Ternary Reproduction Operator

- Differential Evolution uses a single ternary reproduction operation $\mathrm{recombinationDE}$ which takes three arguments g_1, g_2, and g_3

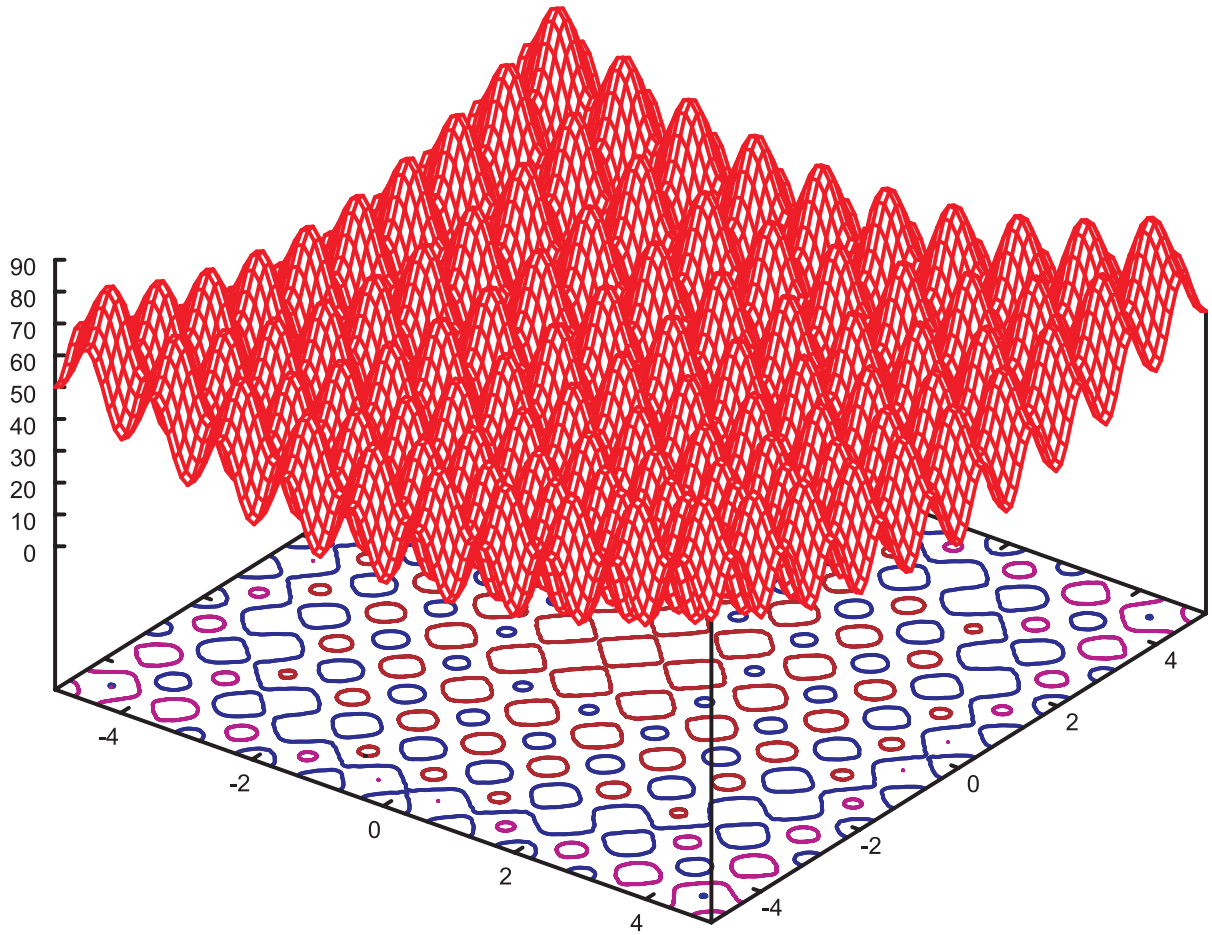
$$\mathrm{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- In the beginning, all candidate solutions are spread uniformly in search space $\mathbb{G}$
- The differential information $(g_1 - g_2)$ is large $\Rightarrow$ large search steps
- As the population converges, the candidate solutions are located closer to each other
- The distances $(g_1 - g_2)$ decrease $\Rightarrow$ the search steps get smaller, too
- Very simple way to perform self-adaptation without needing any additional parameter or operation!
- The step-width is self-organizing from the structure of the population and self-adapting along the optimization process

Section Outline

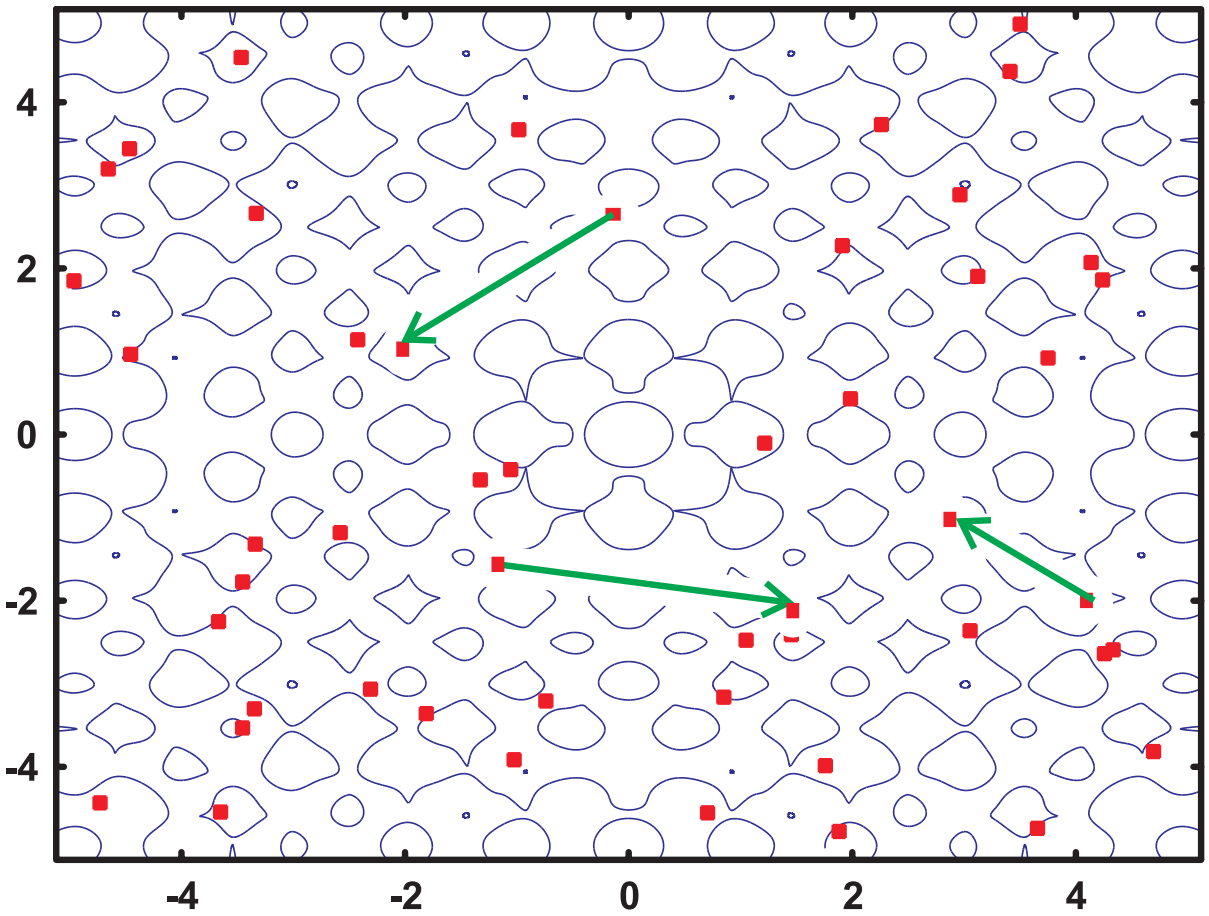
Example 1

Minimize $f(\vec{x}) = 20 + \sum_{i=1}^{2} \left(x_i^2 - 10\cos\left(2 * \pi * x_i\right)\right)$ for $-5.12 \leq x \leq 5.12$



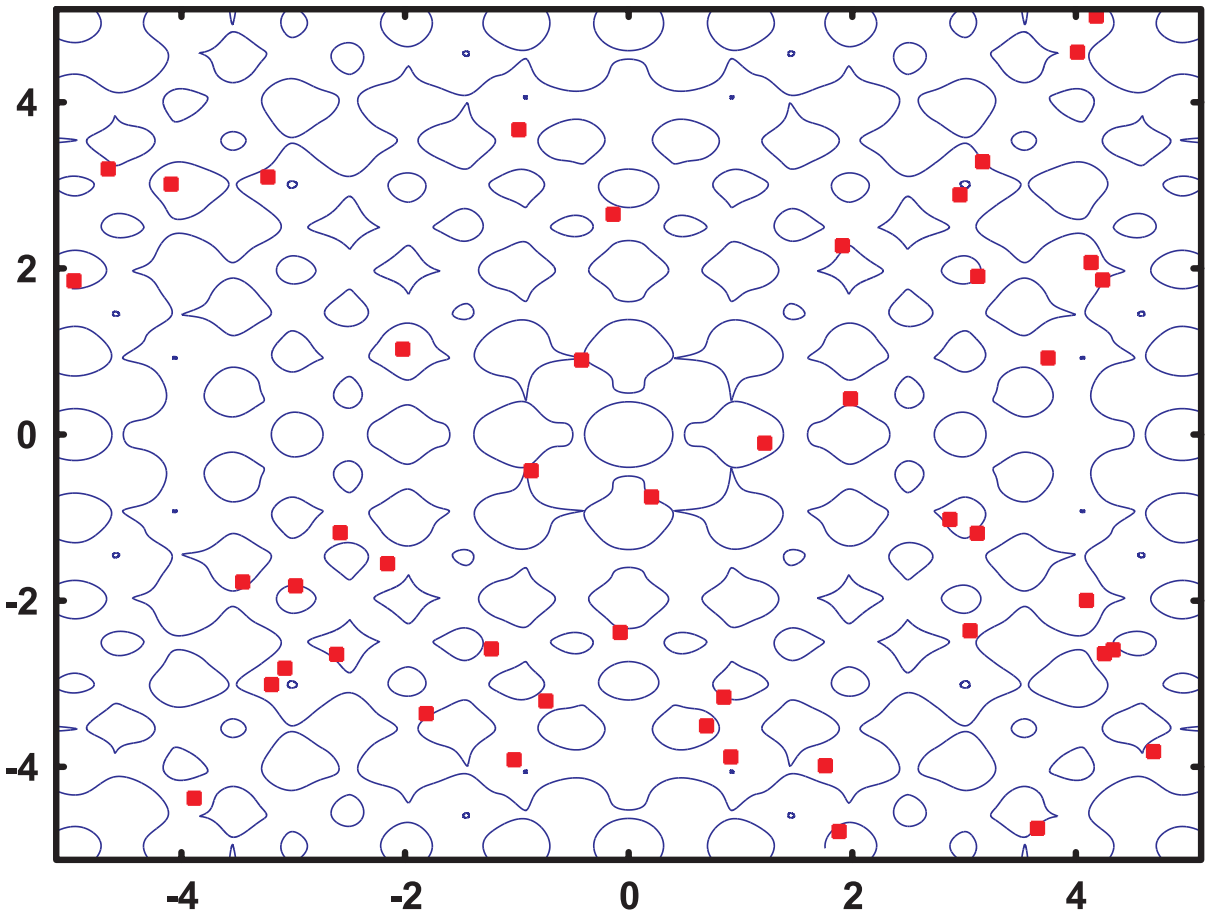
Example 1

- Population after generation 0: uniform spread, large differences



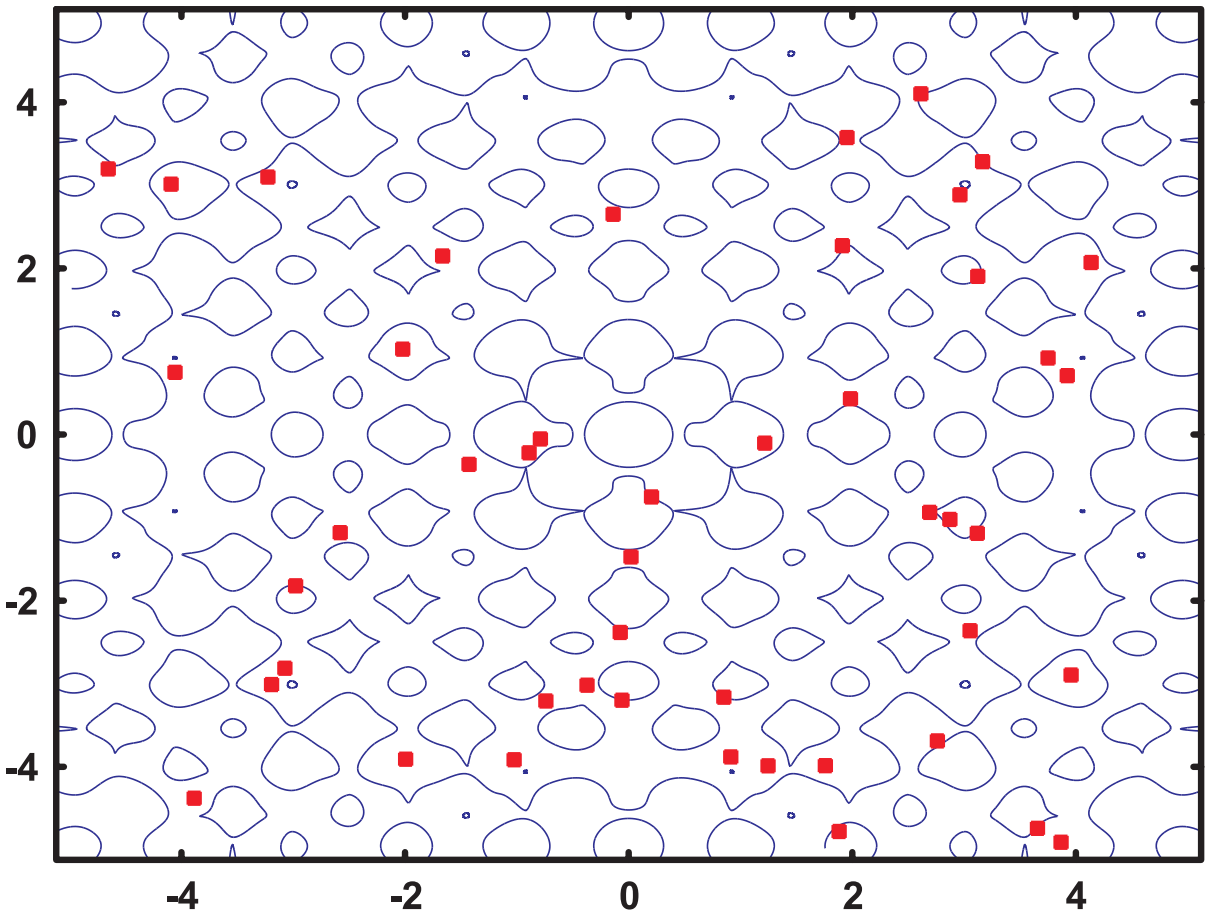
Example 1

- Population after generation 1



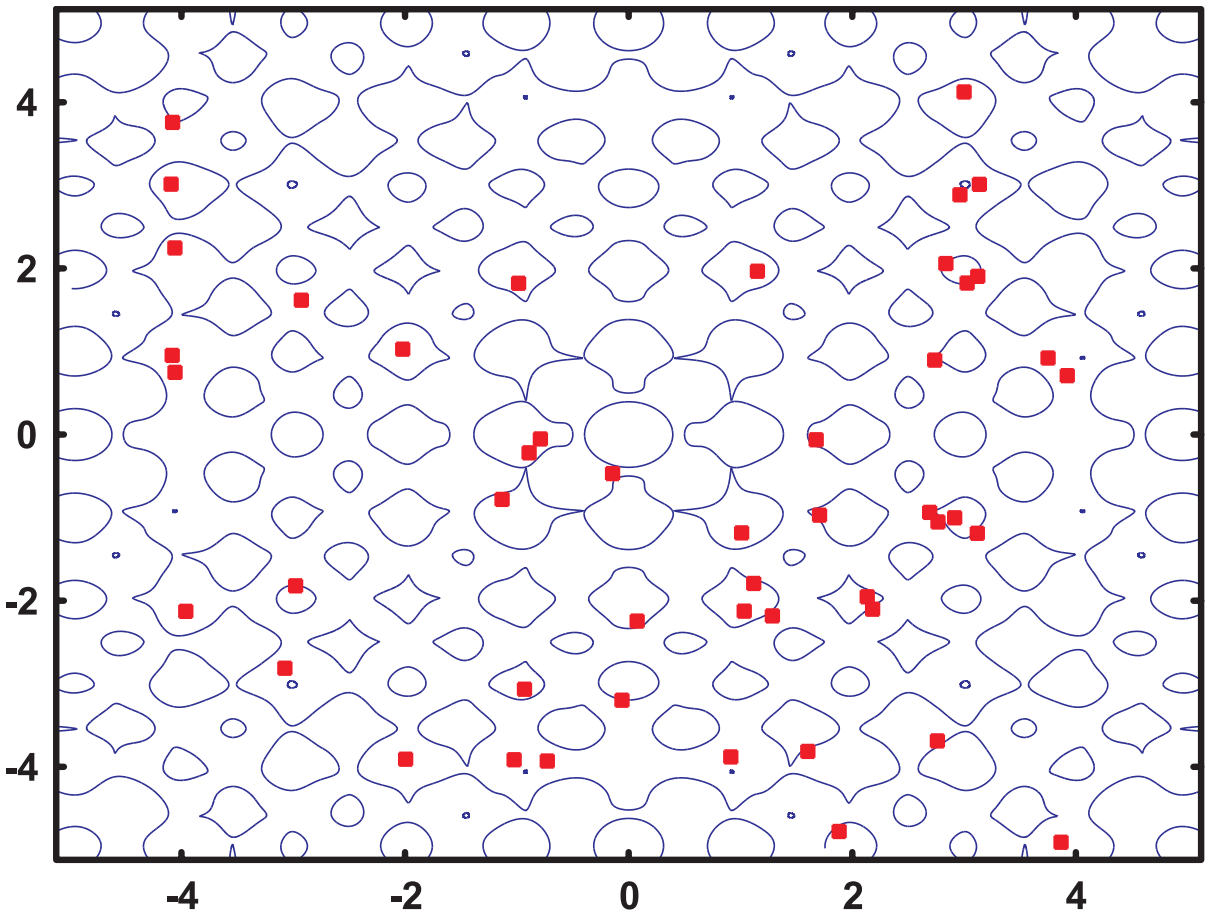
Example 1

- Population after generation 2



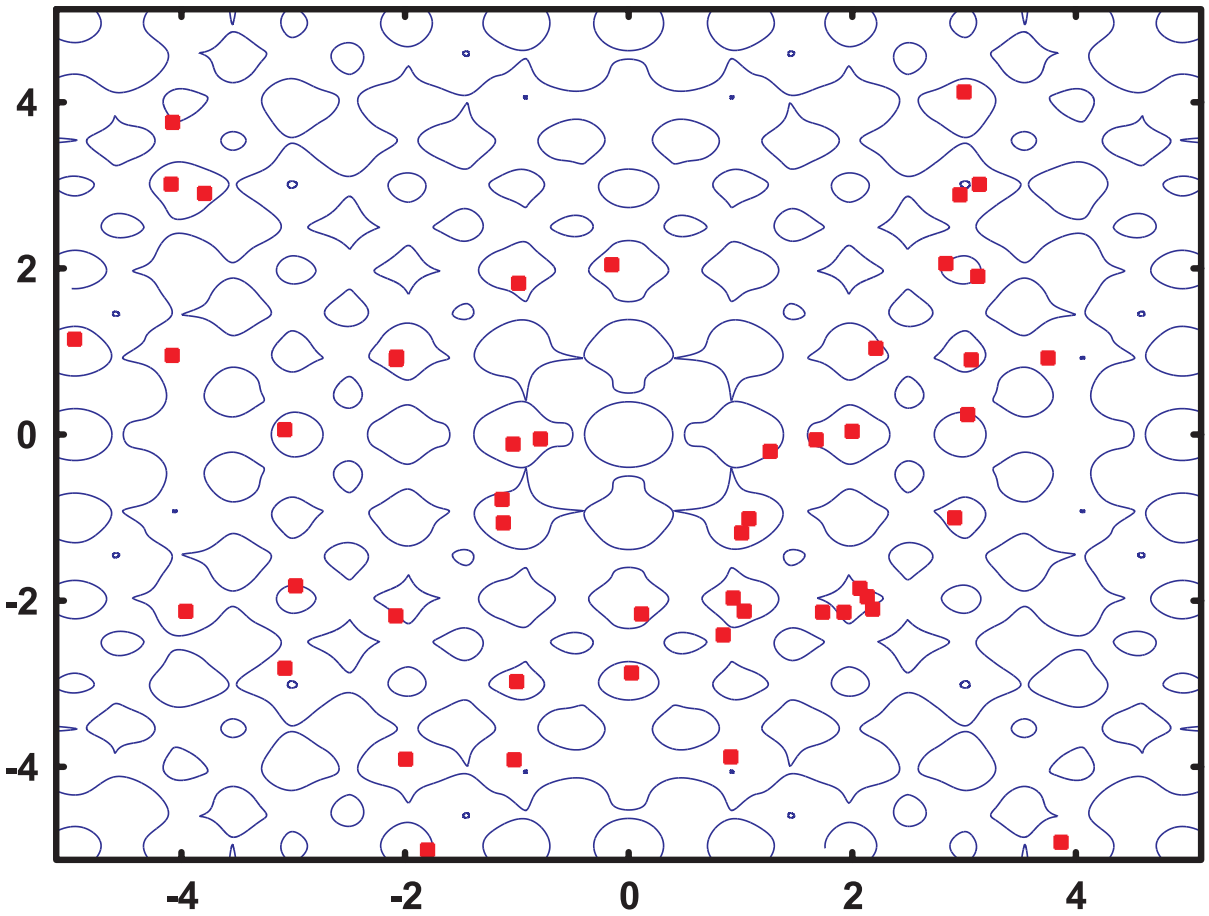
Example 1

- Population after generation 5



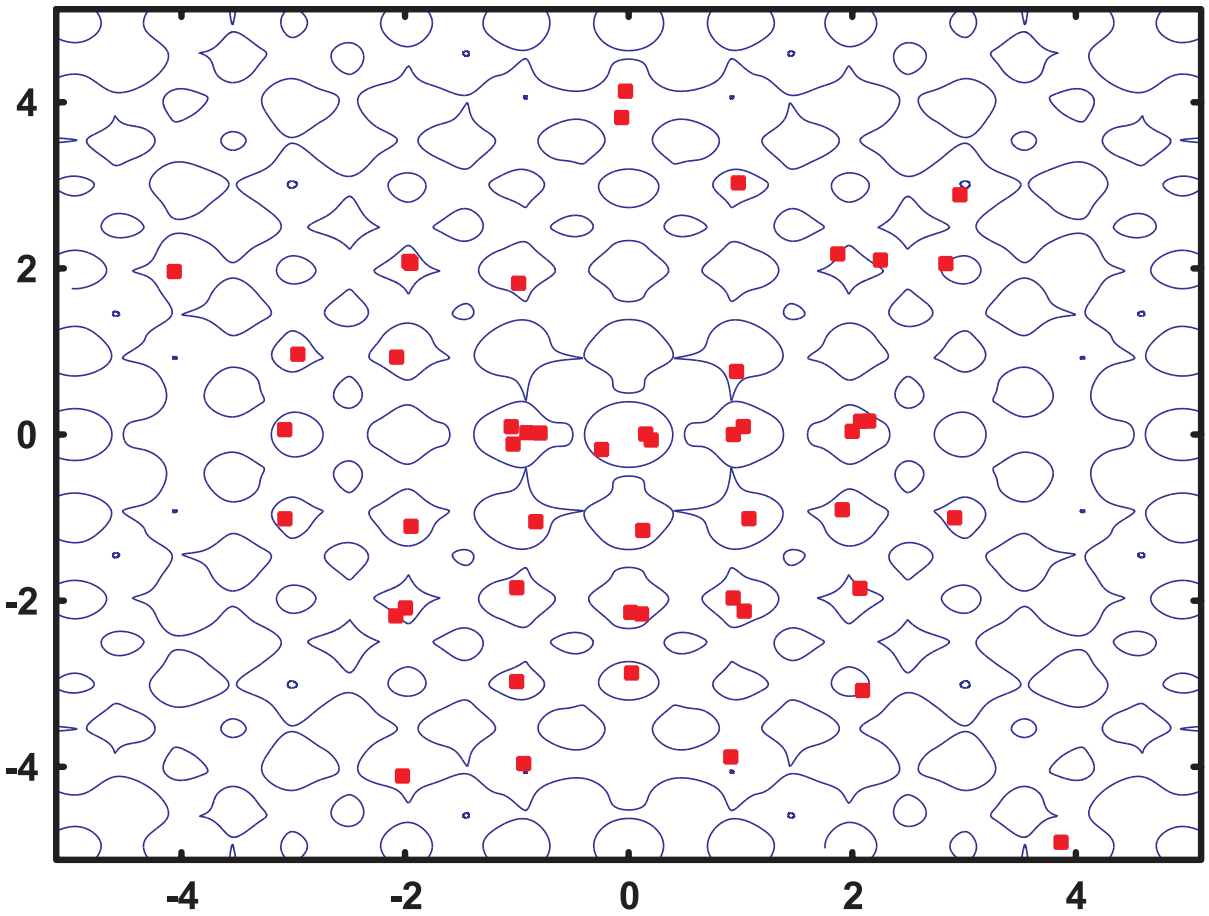
Example 1

- Population after generation 10



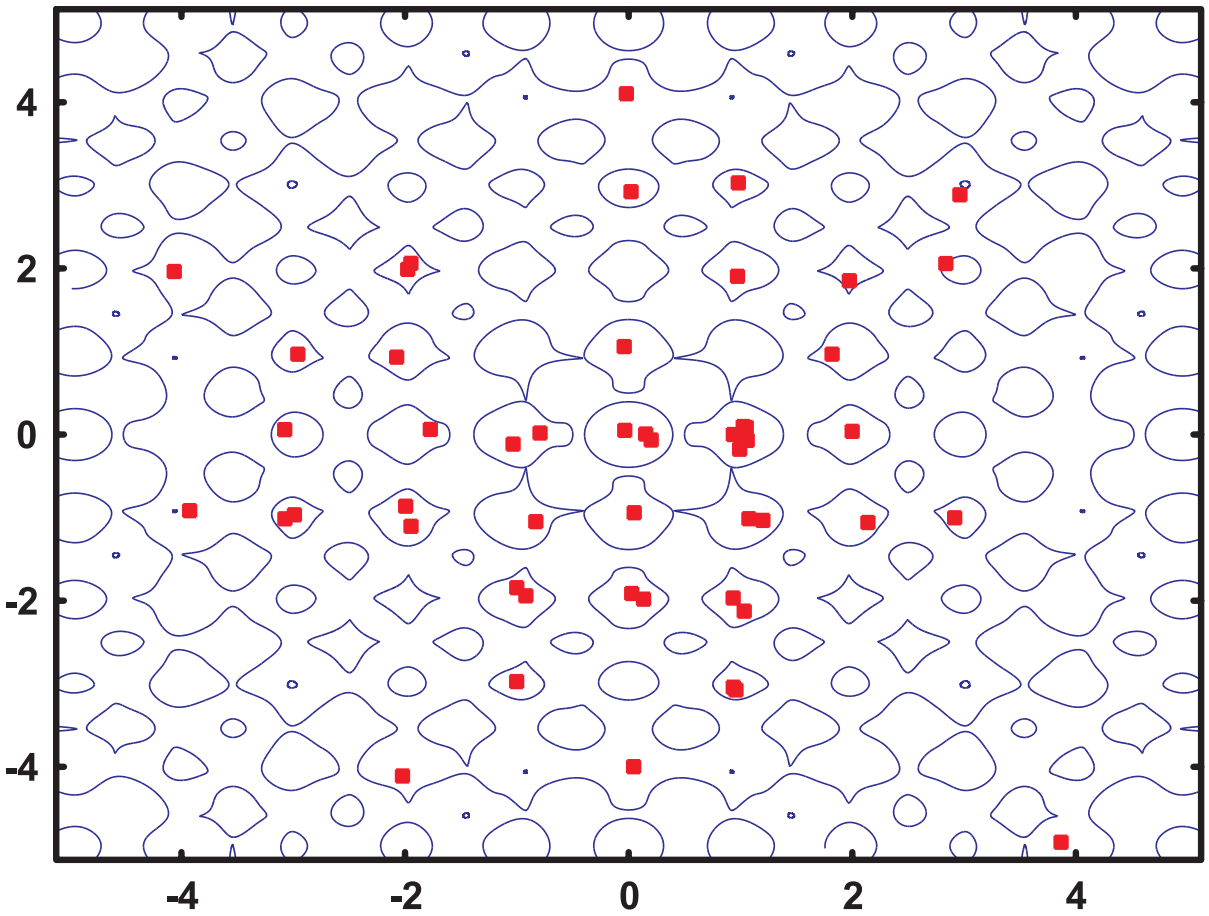
Example 1

- Population after generation 20: concentration around center



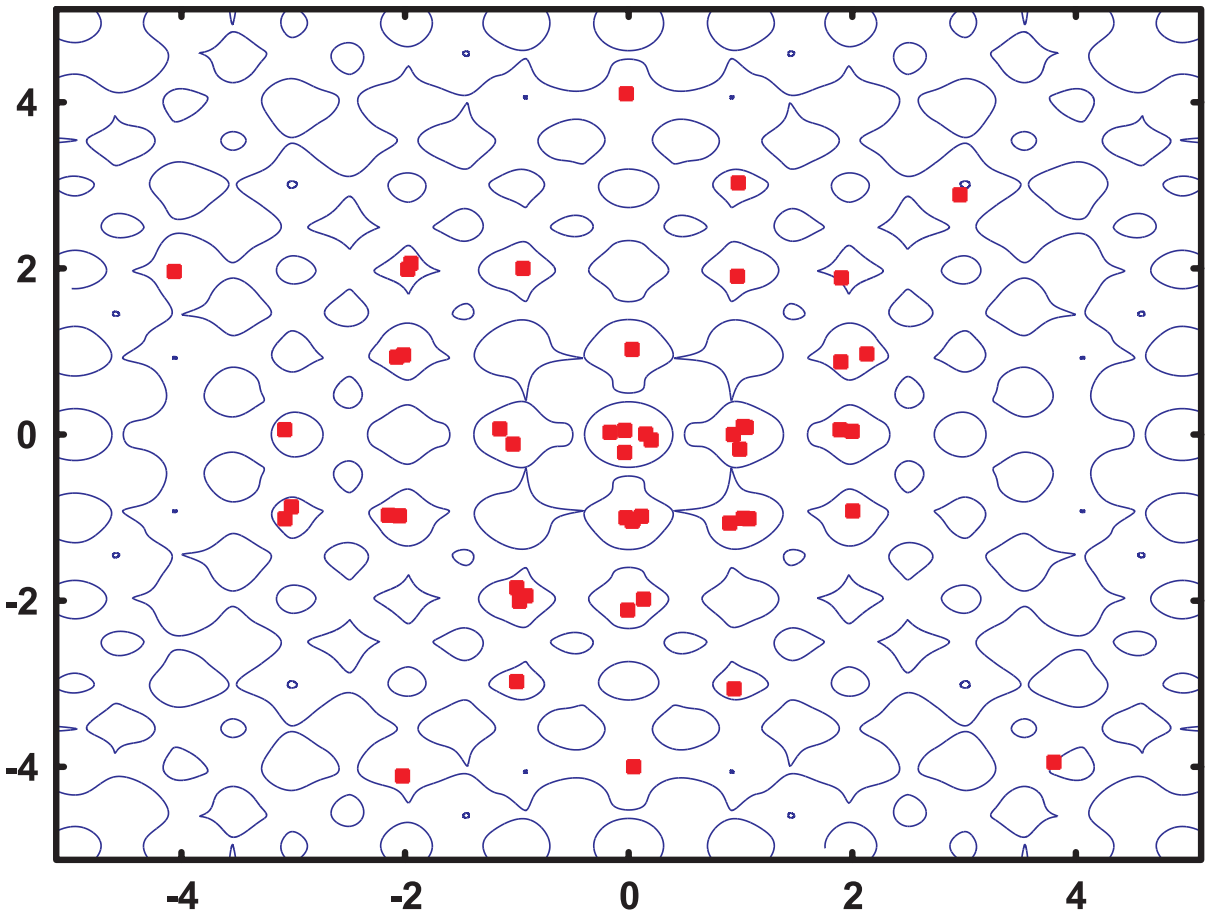
Example 1

- Population after generation 30



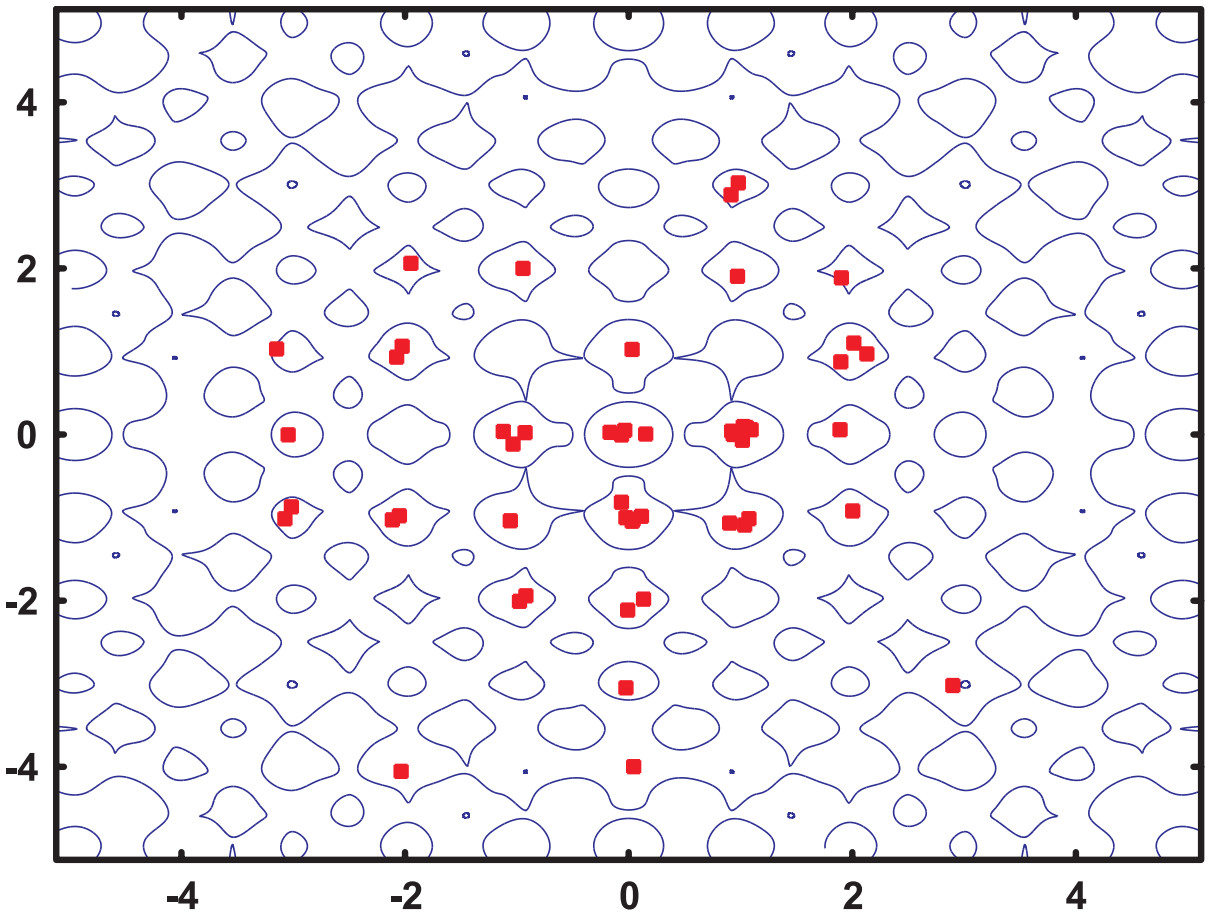
Example 1

- Population after generation 40



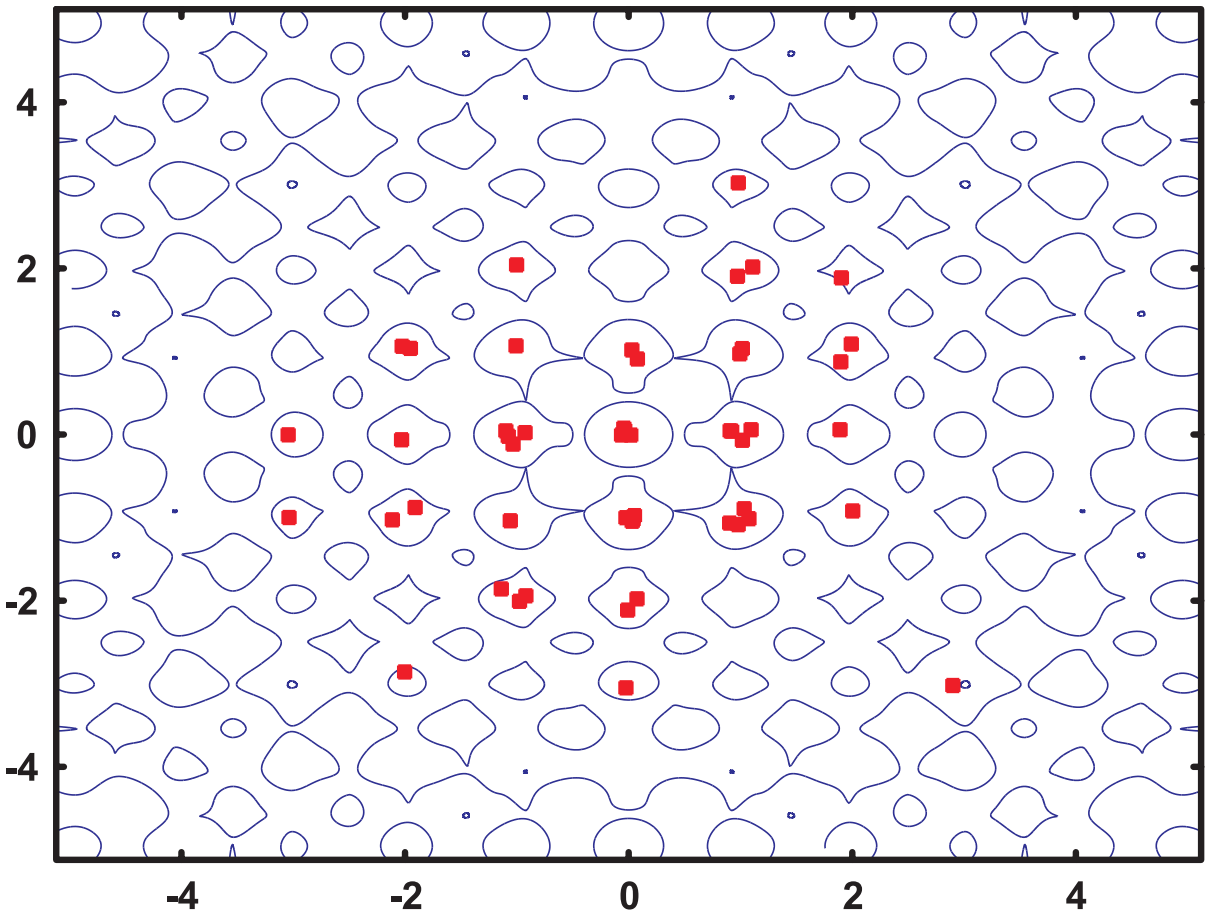
Example 1

- Population after generation 50



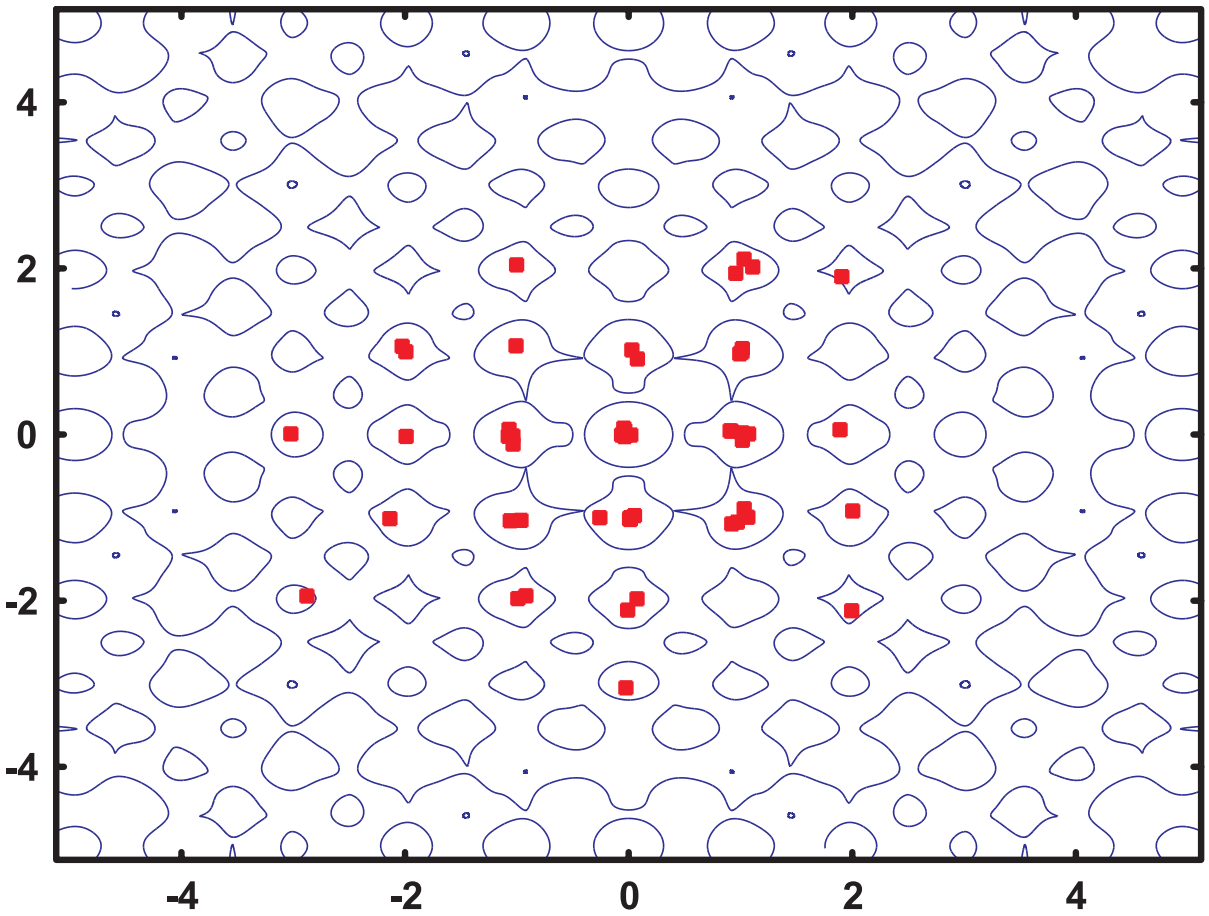
Example 1

- Population after generation 75



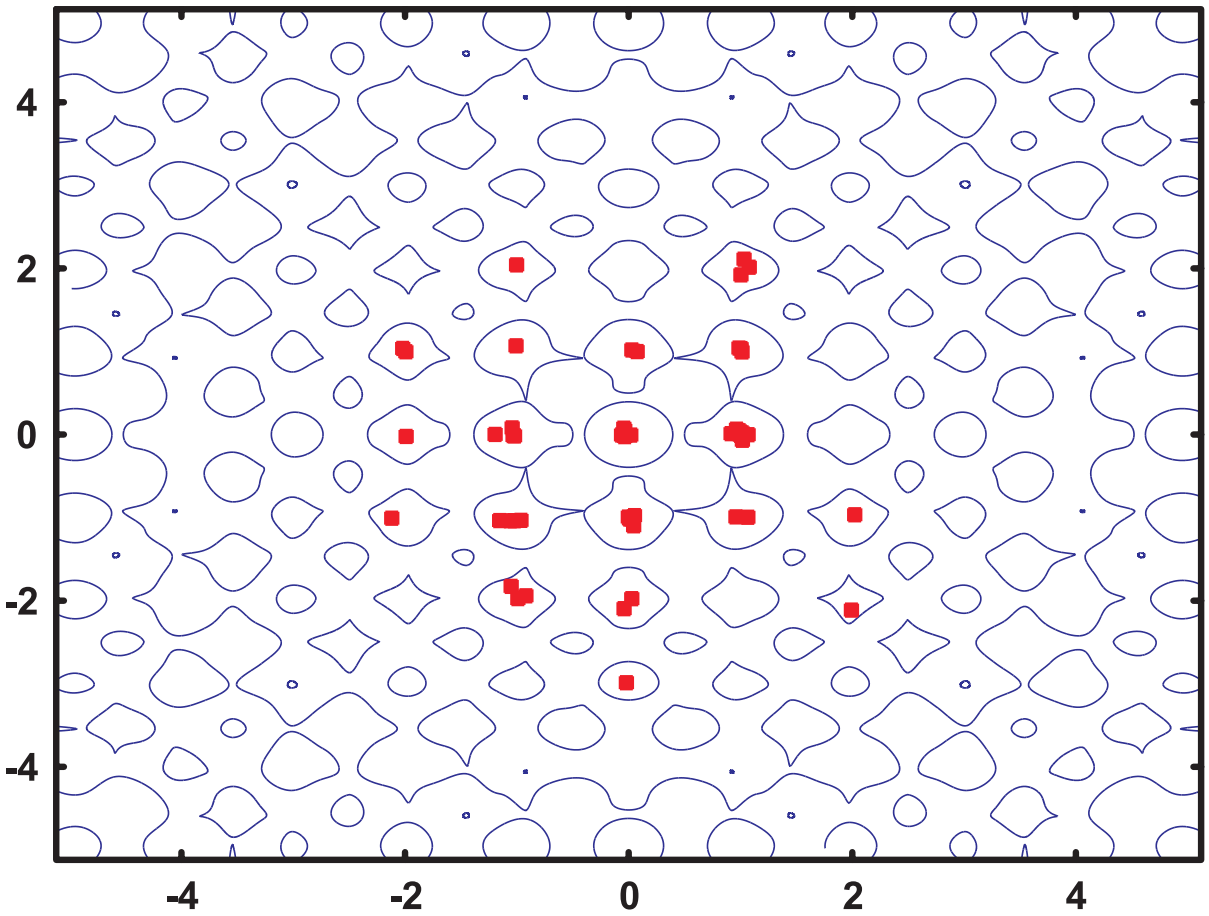
Example 1

- Population after generation 100



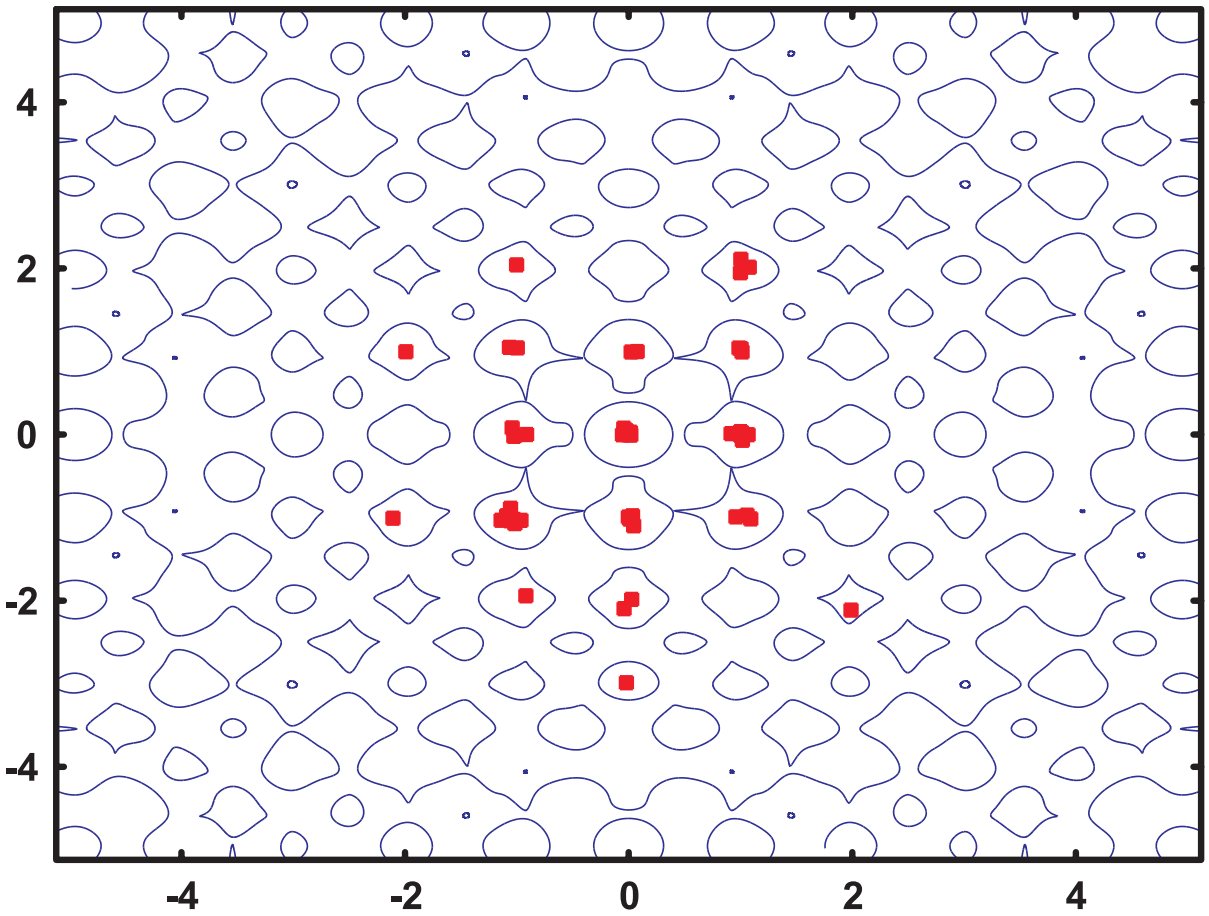
Example 1

- Population after generation 125: similar individuals in local optima



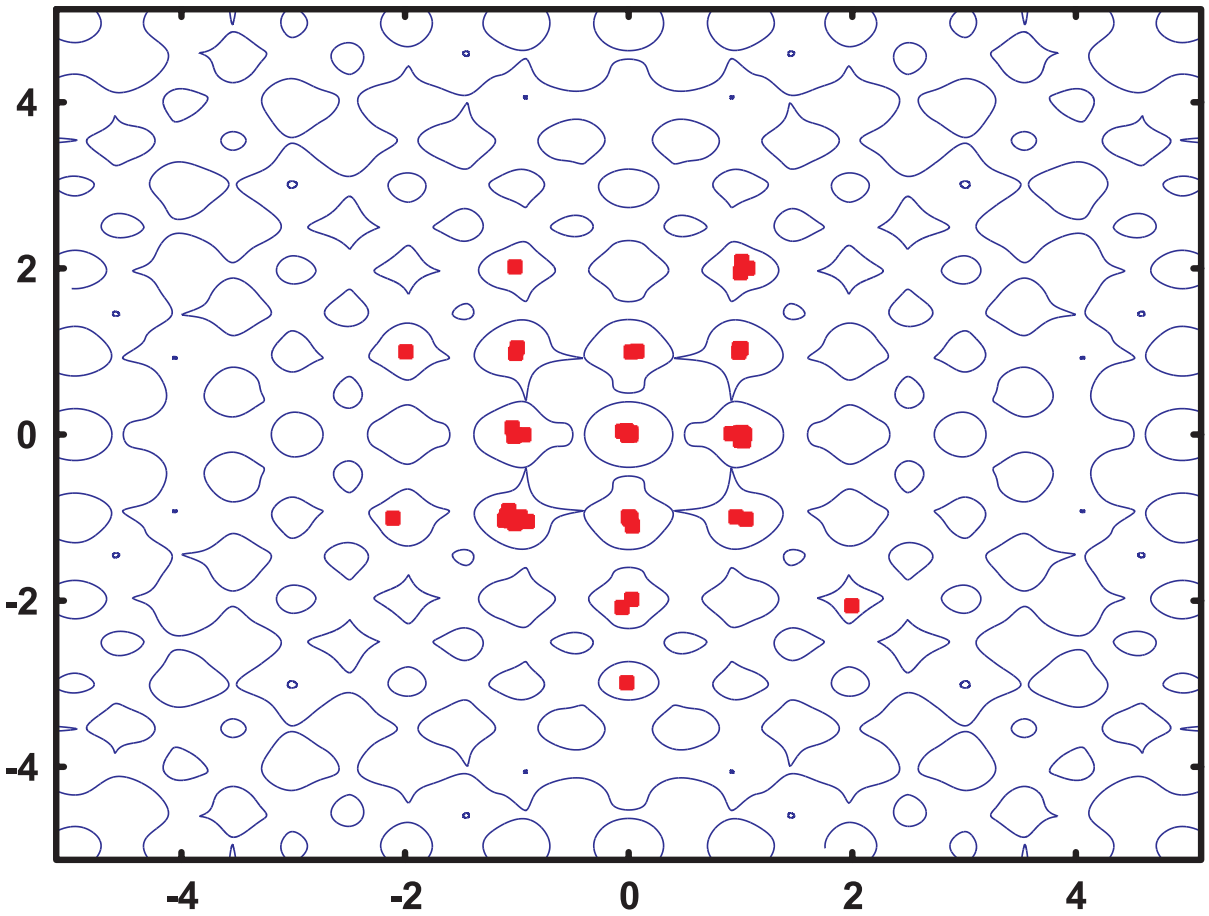
Example 1

- Population after generation 150



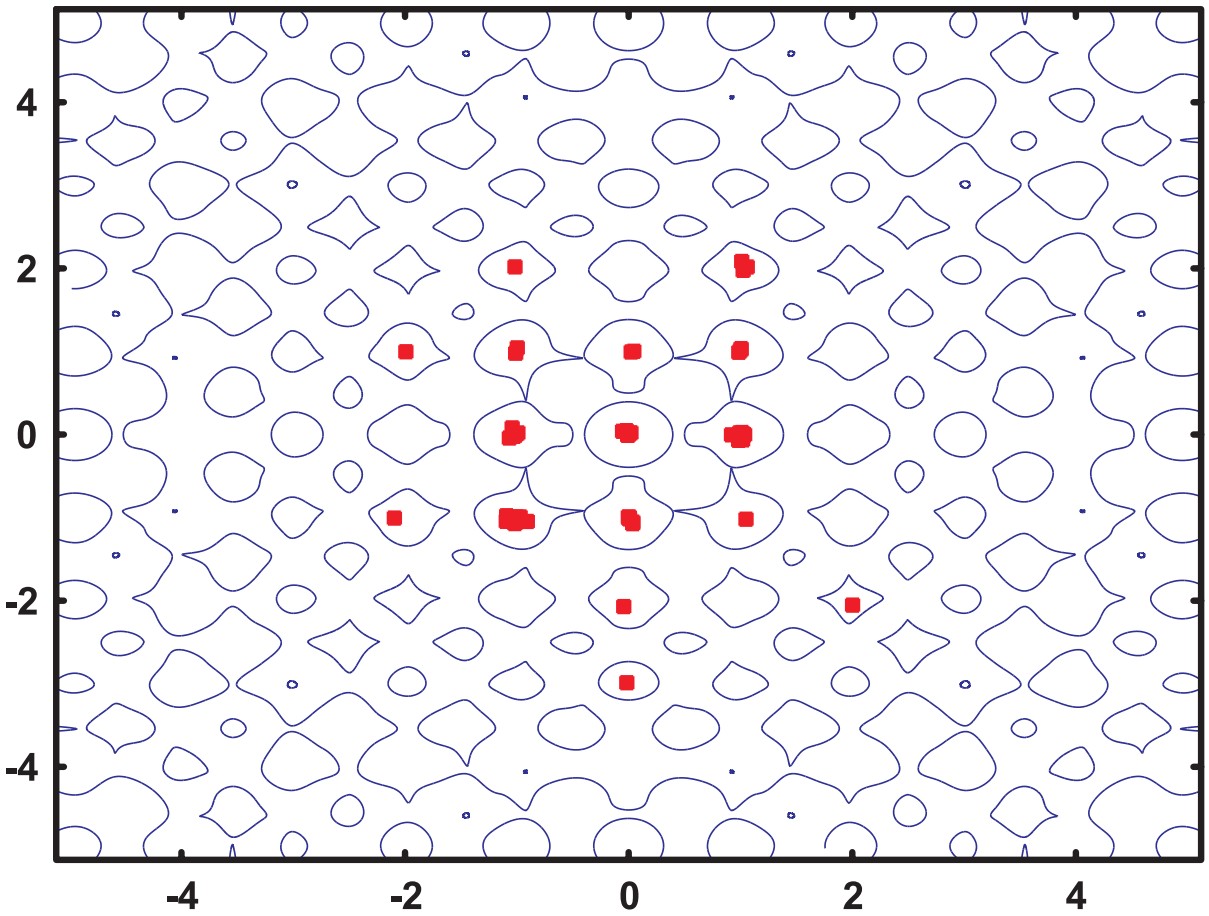
Example 1

- Population after generation 175



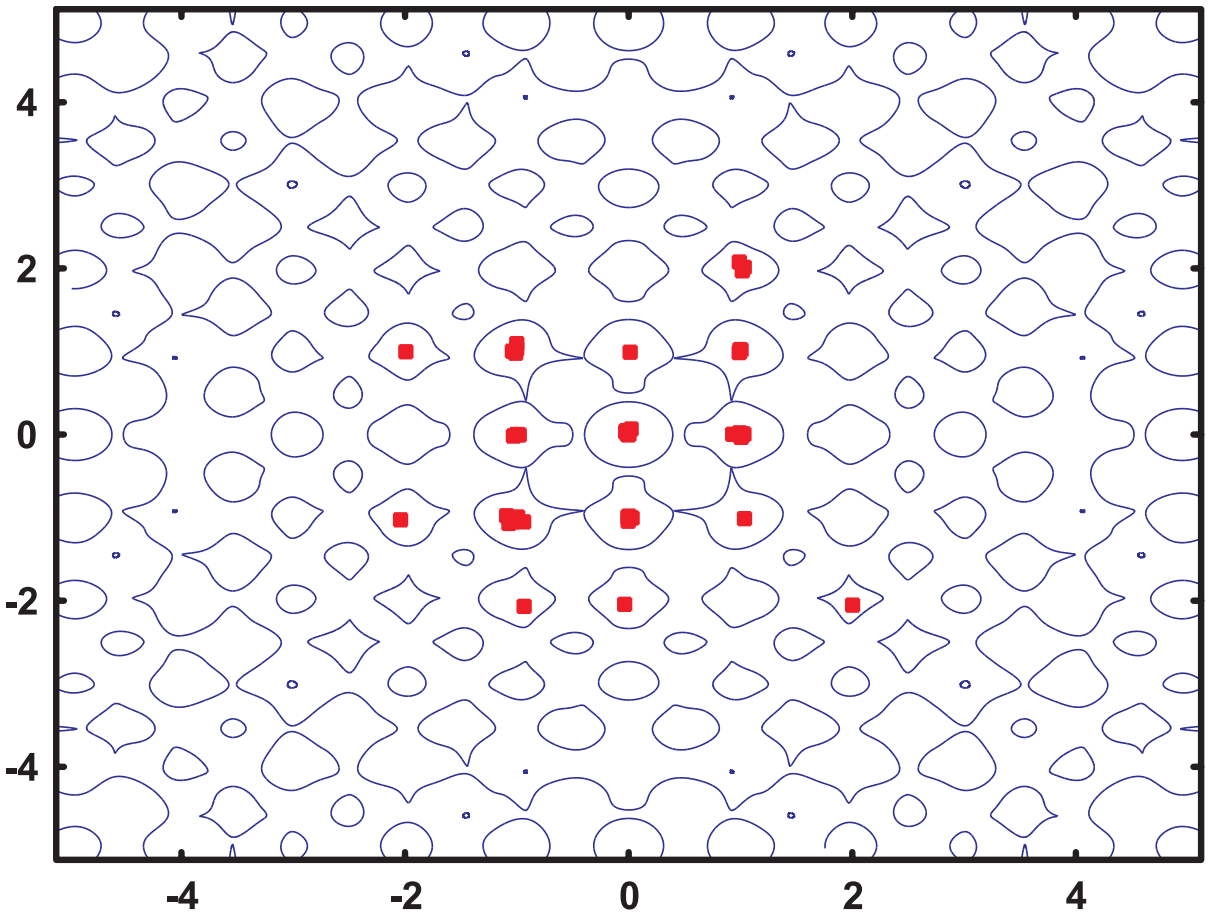
Example 1

- Population after generation 200



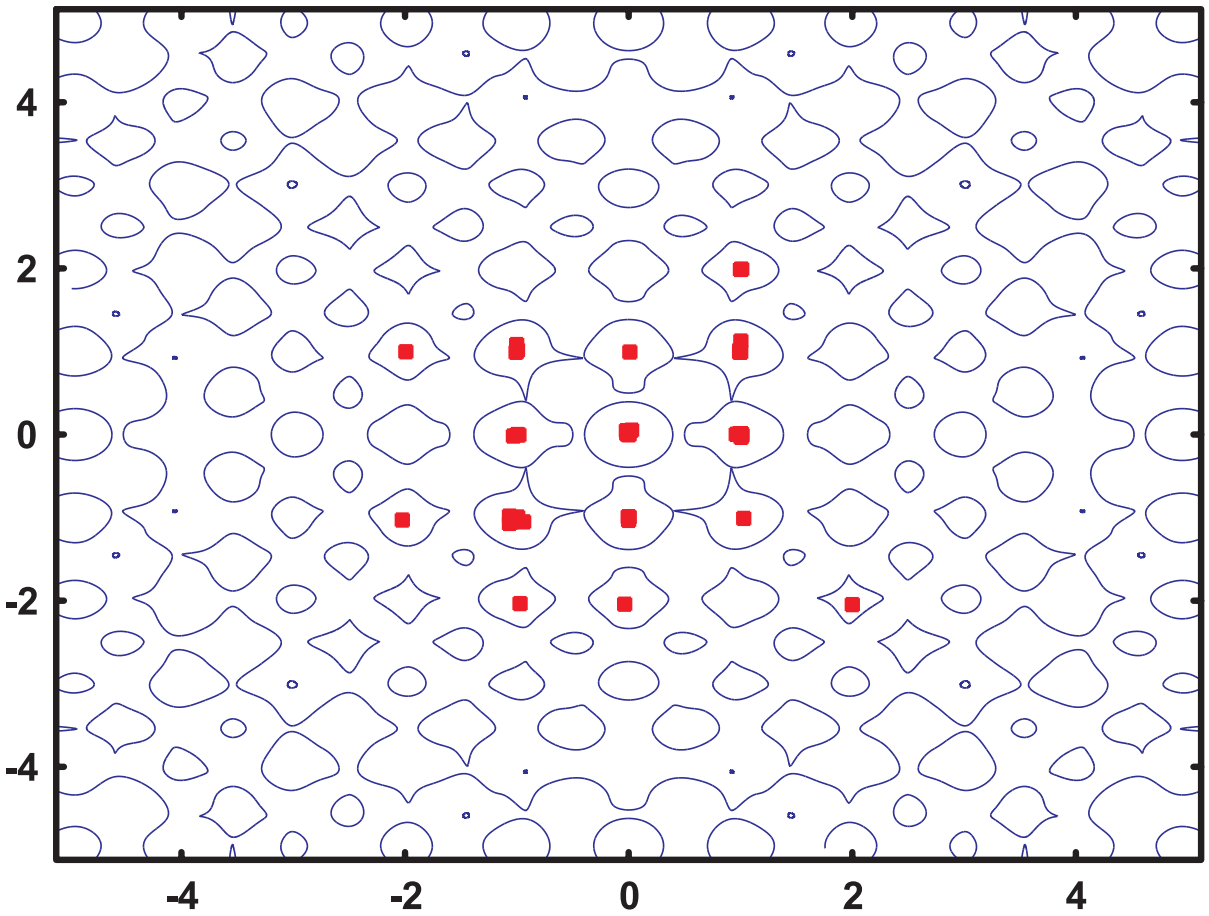
Example 1

- Population after generation 250



Example 1

- Population after generation 300: concentration in optima



Example 1

- Population initially uniformly spread
- Large differential information in population

Example 1

- Population initially uniformly spread
- Large differential information in population
- Population then collapses to (local) optimum/optima

Example 1

- Population initially uniformly spread
- Large differential information in population
- Population then collapses to (local) optimum/optima
- As candidate solutions get closer to each other, differential information decreases

Example 1

- Population initially uniformly spread
- Large differential information in population
- Population then collapses to (local) optimum/optima
- As candidate solutions get closer to each other, differential information decreases
- Due to different optima, differential information still available

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \end{aligned}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \left(\right)$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ & & \end{pmatrix} = \begin{pmatrix} & & \end{pmatrix}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ & & \end{pmatrix} = \begin{pmatrix} 0.28 \\ \end{pmatrix}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ 0.4 & +0.3(0.45 & -0.5) \\ & & \end{pmatrix} = \begin{pmatrix} 0.28 \\ \\ \end{pmatrix}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ 0.4 & +0.3(0.45 & -0.5) \\ & & \end{pmatrix} = \begin{pmatrix} 0.28 \\ 0.385 \\ \end{pmatrix}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ 0.4 & +0.3(0.45 & -0.5) \\ 0.6 & +0.3(0.7 & -0.7) \end{pmatrix} = \begin{pmatrix} 0.28 \\ 0.385 \\ \end{pmatrix}$$

Example 2

- Example for

$$\text{recombinationDE}bg_1g_2g_3 = g_3 + F\left(g_1 - g_2\right) \tag{1}$$

$$\begin{aligned} g_1 &= (0.3, 0.45, 0.7)^T \\ g_2 &= (0.2, 0.5, 0.7)^T \\ g_3 &= (0.25, 0.4, 0.6)^T \\ F &= 0.3 \longleftarrow \text{strength parameter} \end{aligned}$$

$$g' = \begin{pmatrix} 0.25 & +0.3(0.3 & -0.2) \\ 0.4 & +0.3(0.45 & -0.5) \\ 0.6 & +0.3(0.7 & -0.7) \end{pmatrix} = \begin{pmatrix} 0.28 \\ 0.385 \\ 0.6 \end{pmatrix}$$

Section Outline

DE Recombination

Listing: The Ternary DE Recombination Method

```
public class RnRecombineDE extends Rn implements
    ITernarySearchOperation<double[]> {
  /** the strength */
  final double F;
  public double[] recombine(final double[] g1, final double[] g2, final
      double[] g3, final Random r) {

    final double[] res = new double[g1.length];

    for (int i = res.length; (--i) >= 0;) {
      res[i] = Math.max(this.min, Math.min(this.max, //
          (g3[i] + (this.F * (g1[i] - g2[i])))));
    }

    return res;
  }
}
```

Listing: Implementation of the Differential Evolution Algorithm

```
public class DE<X> extends OptimizationAlgorithm<double[], X> {
  public Individual<double[], X> solve(final IObjectiveFunction<X> f) {
    Individual<double[], X>   pbest, pcur;
    Individual<double[], X>[] pop, ofs;
    int i, j, k;

    pbest = new Individual<>();
    pop   = new Individual[this.ps];  // the population: all elements are null
    ofs   = new Individual[this.ps];  // the set of offsprings

    for (i = ofs.length; (--i) >= 0;) {
      ofs[i] = pcur = new Individual<>();
      pcur.g = this.nullary.create(this.random);
    }

    for (;;) {
      for (i   = ofs.length; (--i) >= 0;) {
        pcur   = ofs[i];                        // pcur is the ith offspring
        pcur.x = this.gpm.gpm(pcur.g);          // do gpm
        pcur.v = f.compute(pcur.x);             // get objective value
        if ((pop[i] == null) || (pcur.v <= pop[i].v)) { // offspring better than parent?
          pop[i] = pcur;                                // replace parent
          if (pcur.v < pbest.v) {               // check if we found best
            pbest.assign(pcur);
          }
        }
        if (this.termination.shouldTerminate()) {
          return pbest;
        }
      }

      for (i = pop.length; (--i) >= 0;) {     // create offspring
        ofs[i] = pcur = new Individual<>();   // pick first parent

        do {
          j = this.random.nextInt(pop.length);
        } while (j == i); // different parents!
        do {
          k = this.random.nextInt(pop.length);
        } while ((k == i) || (k == j)); // different parents!

        pcur.g = this.ternary.recombine(pop[j].g, pop[k].g, pop[i].g, this.random);
      }
    }
  }
}
```

Section Outline

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once
- Instead, a gene crossover rate cr is used to determine which genes should computed according to the ternary principle

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once
- Instead, a gene crossover rate cr is used to determine which genes should computed according to the ternary principle
- The rest are just copied from g_3 directly

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once
- Instead, a gene crossover rate cr is used to determine which genes should computed according to the ternary principle
- The rest are just copied from g_3 directly
- Two basic methods

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once
- Instead, a gene crossover rate cr is used to determine which genes should computed according to the ternary principle
- The rest are just copied from g_3 directly
- Two basic methods
  1. binomial: For each gene in the offspring, use ternary crossover principle with probability cr and copy the corresponding value from g_3 with probability $1 - cr$

Gene-wise Recombination

$$\text{recombinationDE}(g_1, g_2, g_3) = g_3 + F\,(g_1 - g_2) \tag{1}$$

- Modification: crossover is not applied to all genes at once
- Instead, a gene crossover rate cr is used to determine which genes should computed according to the ternary principle
- The rest are just copied from g_3 directly
- Two basic methods
  1. binomial: For each gene in the offspring, use ternary crossover principle with probability cr and copy the corresponding value from g_3 with probability $1 - cr$
  2. exponential: use ternary crossover principle for a consecutive group of genes, with a exponentially distributed length according to cr; copy the rest from g_3

Binomial DE Recombination

Listing: The Binomial DE Recombination

```
public class RnRecombineDEBin extends RnRecombineDE {
  /** the crossover rate */
  private final double cr;


  public double[] recombine(final double[] g1, final double[] g2, final
      double[] g3, final Random r) {

    final double[] res = g1.clone();

    for (int i = res.length; (--i) >= 0;) {
      if (r.nextDouble() < this.cr) {
        res[i] = Math.max(this.min, Math.min(this.max, //
            (g3[i] + (this.F * (g1[i] - g2[i])))));
      }
    }

    return res;
  }
}
```

Section Outline

Summary

- EA for numerical optimization

Summary

- EA for numerical optimization
- Simple self-adaption without any parameter by ternary recombination

Summary

- EA for numerical optimization
- Simple self-adaption without any parameter by ternary recombination
- Parents compete with their children in the population

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing Series. Basel, Switzerland: Birkhäuser Verlag, 2005. ISBN 3-540-20950-6, 3-540-31306-0, 978-3-540-20950-8, and 978-3-540-31306-9. URL `http://books.google.de/books?id=S67vX-KqVqUC`.
2. Vitaliy Feoktistov. *Differential Evolution – In Search of Solutions*, volume 5 of *Springer Optimization and Its Applications*. New York, NY, USA: Springer New York, December 2006. ISBN 0-387-36895-7, 0-387-36896-5, 978-0-387-36895-5, and 978-0-387-36896-2. URL `http://books.google.de/books?id=kG7aP_v-SU4C`.
3. Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos Artemio Coello Coello. A comparative study of differential evolution variants for global optimization. In Maarten Keijzer and Mike Cattolico, editors, *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)*, pages 485–492, Seattle, WA, USA: Renaissance Seattle Hotel, July 8–12, 2006. New York, NY, USA: ACM Press. doi: 10.1145/1143997.1144086. URL `http://delta.cs.cinvestav.mx/~ccoello/conferences/mezura-gecco2006.pdf.gz`.
4. Janez Brest, Viljem Žumer, and Mirjam Sepesy Maučec. Control parameters in self-adaptive differential evolution. In Bogdan Filipič and Jurij Šilc, editors, *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA'06)*, Informacijska Družba (Information Society), pages 35–44, Ljubljana, Slovenia: Jožef Stefan International Postgraduate School, October 9–10, 2006. Ljubljana, Slovenia: Jožef Stefan Institute. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.8106`.
5. Jouni A. Lampinen and Ivan Zelinka. On stagnation of the differential evolution algorithm. In Pavel Osmera, editor, *Proceedings of the 6th International Conference on Soft Computing (MENDEL'00)*, pages 76–83, Brno, Czech Republic: Brno University of Technology, June 7–9, 2000. Brno, Czech Republic: Brno University of Technology, Ústav Automatizace a Informatiky. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.7932`.
6. Roberto R. F. Mendes and Arvind S. Mohais. Dynde: A differential evolution for dynamic optimization problems. In David Wolfe Corne, Zbigniew Michalewicz, Robert Ian McKay, Ágoston E. Eiben, David B. Fogel, Carlos M. Fonseca, Günther R. Raidl, Kay Chen Tan, and Ali M. S. Zalzala, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05)*, volume 3, pages 2808–2815, Edinburgh, Scotland, UK, September 2–5, 2005. Piscataway, NJ, USA: IEEE Computer Society. doi: 10.1109/CEC.2005.1555047. URL `http://www3.di.uminho.pt/~rcm/publications/DynDE.pdf`.

Bibliography II

7. Patricia Besson, Jean-Marc Vesin, Vlad Popovici, and Murat Kunt. Differential evolution applied to a multimodal information theoretic optimization problem. In Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, Ernesto Jorge Fernandes Costa, Carlos Cotta, Rolf Drechsler, Evelyne Lutton, Penousal Machado, Jason H. Moore, Juan Romero, George D. Smith, Giovanni Squillero, and Hideyuki Takagi, editors, *Applications of Evolutionary Computing – Proceedings of EvoWorkshops 2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, and EvoSTOC (EvoWorkshops'06)*, volume 3907/2006 of *Lecture Notes in Computer Science (LNCS)*, pages 505–509, Budapest, Hungary, April 10–12, 2006. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/11732242_46.
8. Rainer M. Storn. Differential evolution (de) for continuous function optimization (an algorithm by kenneth price and rainer storn), 2010. URL `http://www.icsi.berkeley.edu/~storn/code.html`.
9. Rainer M. Storn and Kenneth V. Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, Berkeley, CA, USA: International Computer Science Institute (ICSI), University of California, March 1995. URL `http://http.icsi.berkeley.edu/~storn/TR-95-012.pdf`.
10. Scott Kirkpatrick, Charles Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science Magazine*, 220(4598):671–680, May 13, 1983. doi: 10.1126/science.220.4598.671. URL `http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf`.
11. Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985. doi: 10.1007/BF00940812. URL `http://mkweb.bcgsc.ca/papers/cerny-travelingsalesman.pdf`. Communicated by S. E. Dreyfus. Also: Technical Report, Comenius University, Mlynská Dolina, Bratislava, Czechoslovakia, 1982.
12. Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. *ACM SIGMICRO Newsletter*, 13(4):143–148, December 1982.
13. Dean Jacobs, Jan Prins, Peter Siegel, and Kenneth Wilson. Monte carlo techniques in code optimization. In *International Symposium on Microarchitecture – Proceedings of the 15th Annual Workshop on Microprogramming (MICRO 15)*, pages 143–146, Palo Alto, CA, USA, October 5–7, 1982. Piscataway, NJ, USA: IEEE (Institute of Electrical and Electronics Engineers).
14. Peter Salamon, Paolo Sibani, and Richard Frost. *Facts, Conjectures, and Improvements for Simulated Annealing*, volume 7 of *SIAM Monographs on Mathematical Modeling and Computation*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics (SIAM), 2002. ISBN 0898715083 and 9780898715088. URL `http://books.google.de/books?id=jhAldlYvClcC`.

Bibliography III

15. Peter J. M. van Laarhoven and Emile H. L. Aarts, editors. *Simulated Annealing: Theory and Applications*, volume 37 of *Mathematics and its Applications*. Norwell, MA, USA: Kluwer Academic Publishers, 1987. ISBN 90-277-2513-6, 978-90-277-2513-4, and 978-90-481-8438-5. URL http://books.google.de/books?id=-IgUab6Dp_IC.
16. Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. London, UK: Pitman, 1987. ISBN 0273087711, 0934613443, 9780273087717, and 978-0934613446. URL http://books.google.de/books?id=edfSSAAACAAJ.
17. James C. Spall. *Introduction to Stochastic Search and Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, first edition, June 2003. ISBN 0-471-33052-3, 0-471-72213-8, 978-0-471-33052-3, and 978-0-471-72213-7. URL http://books.google.de/books?id=f66OIvvkKnAC.
18. Hans-Georg Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. New York, NY, USA: Springer New York, May 27, 2001. ISBN 3-540-67297-4 and 978-3-540-67297-5. URL http://books.google.de/books?id=8tbInLufkTMC.
19. Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Berlin, Germany: Technische Universität Berlin, 1971. URL http://books.google.de/books?id=QcNNGQAACAAJ.