Metaheuristic Optimization

19. Estimation of Distribution Algorithms

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

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

website

Section Outline

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

Introduction

- EAs, GP, ES, and GAs are based on principles gleaned from natural evolution, such as survival of the fittest and reproduction

Introduction

- EAs, GP, ES, and GAs are based on principles gleaned from natural evolution, such as survival of the fittest and reproduction
- Estimation of Distribution Algorithms (EDAs) are similar, but also different

Introduction

- EAs, GP, ES, and GAs are based on principles gleaned from natural evolution, such as survival of the fittest and reproduction
- Estimation of Distribution Algorithms (EDAs) are similar, but also different:
    - EAs try to evolve a solution

Introduction

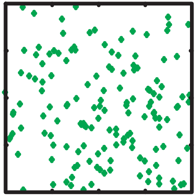
- EAs, GP, ES, and GAs are based on principles gleaned from natural evolution, such as survival of the fittest and reproduction
- Estimation of Distribution Algorithms (EDAs) are similar, but also different:
  - EAs try to evolve a solution
  - EDAs build a model of what a perfect solution should look like

Idea

- Information in a population can be represented by statistical model

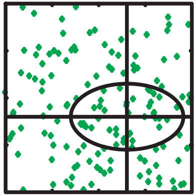
Idea

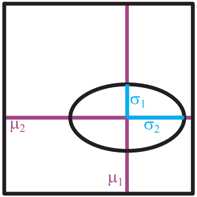
- Information in a population can be represented by statistical model



Idea

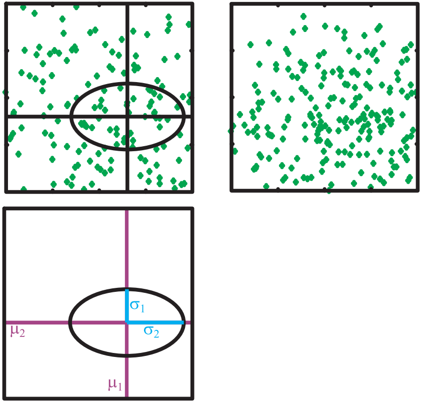
- Information in a population can be represented by statistical model





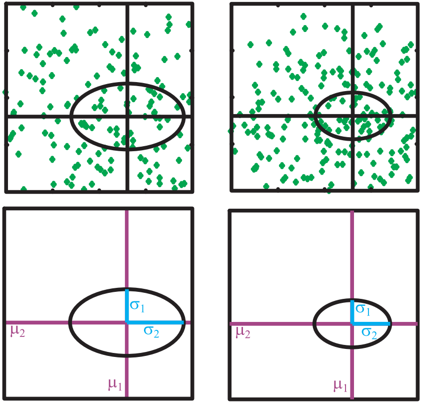
Idea

- Information in a population can be represented by statistical model



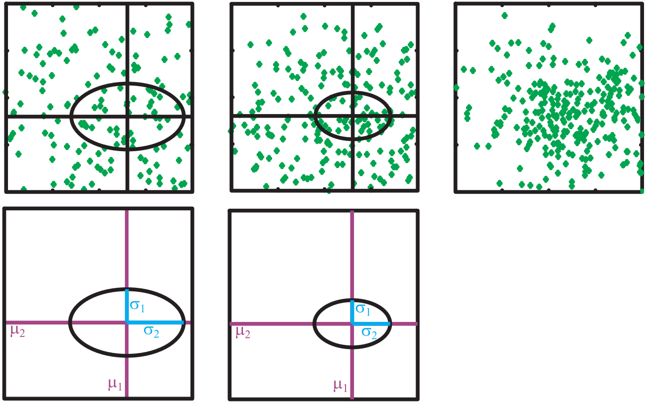
Idea

- Information in a population can be represented by statistical model



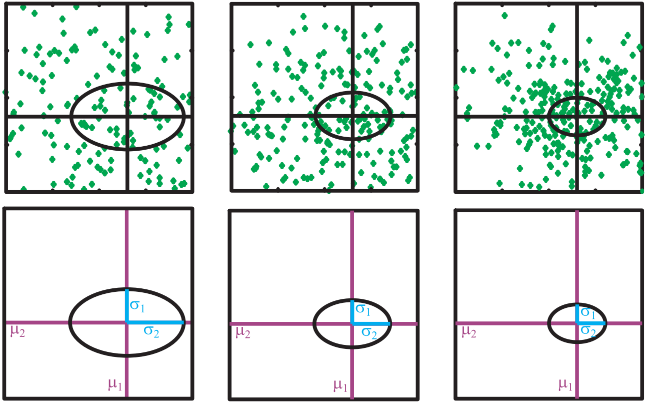
Idea

- Information in a population can be represented by statistical model



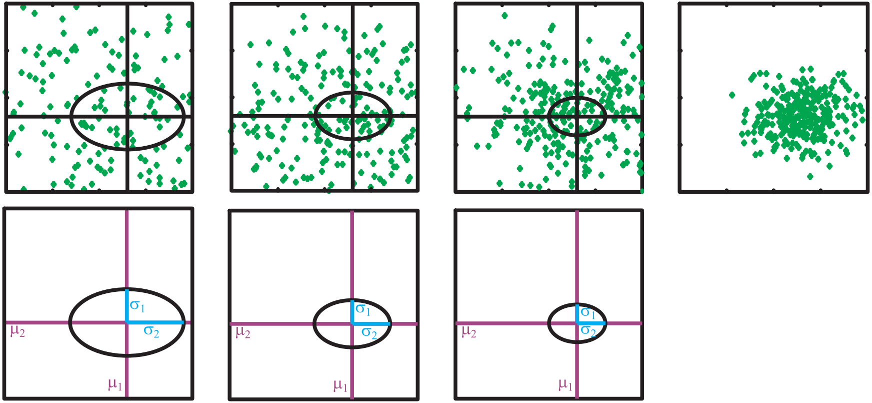
Idea

- Information in a population can be represented by statistical model

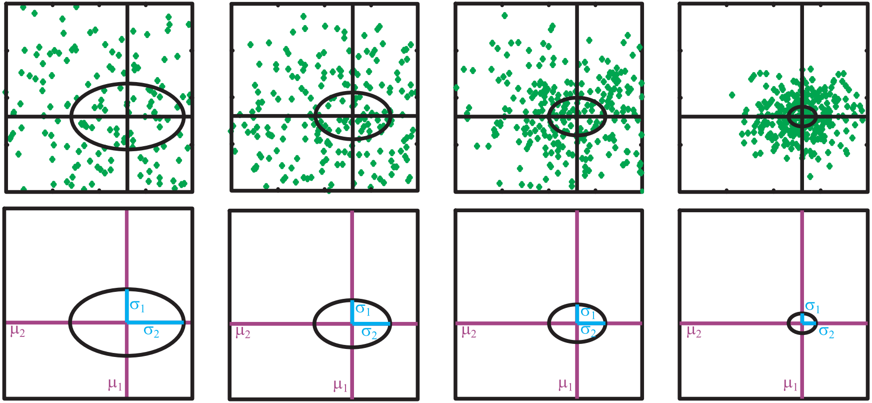


Idea

- Information in a population can be represented by statistical model



- Information in a population can be represented by statistical model



Algorithm

1. Create a statistical model of possible solutions

Algorithm

1. Create a statistical model of possible solutions
2. ps times do

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
    - **2.1** sample the model (create a new point in search space)

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
   2.1 sample the model (create a new point in search space)
   2.2 perform the genotype-phenotype mapping

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
   2.1 sample the model (create a new point in search space)
   2.2 perform the genotype-phenotype mapping
   2.3 compute the objective value

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
   - **2.1** sample the model (create a new point in search space)
   - **2.2** perform the genotype-phenotype mapping
   - **2.3** compute the objective value
3. Select mps individuals from the ps new candidate solutions

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
   - **2.1** sample the model (create a new point in search space)
   - **2.2** perform the genotype-phenotype mapping
   - **2.3** compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals

Algorithm

1. Create a statistical model of possible solutions
2. ps times do:
   - 2.1 sample the model (create a new point in search space)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2

Section Outline

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

UMDA

- Univariate Marginal Distribution Algorithm (UMDA) [1, 2]

UMDA

- Univariate Marginal Distribution Algorithm (UMDA) [1, 2]
- Search space: Fixed-Length Bit Strings of length n

UMDA

- Univariate Marginal Distribution Algorithm (UMDA) [1, 2]
- Search space: Fixed-Length Bit Strings of length n
- Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \dots n$

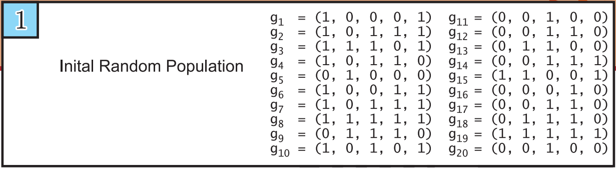
UMDA

- Univariate Marginal Distribution Algorithm (UMDA) [1, 2]
- Search space: Fixed-Length Bit Strings of length n
- Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \dots n$
- $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $\overset{\star}{g}$ should be 0

UMDA

- Univariate Marginal Distribution Algorithm (UMDA) [1, 2]
- Search space: Fixed-Length Bit Strings of length n
- Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \ldots n$
- $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $g^{\star}$ should be 0
- Initialization: $M[i] = 0.5 \forall i \in 1 \ldots n$

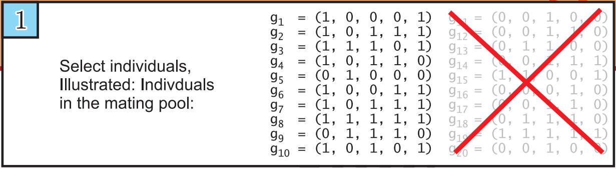
1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)



Inital Random Population

$g_1 = (1, 0, 0, 0, 1)$ $g_{11} = (0, 0, 1, 0, 0)$
$g_2 = (1, 0, 1, 1, 1)$ $g_{12} = (0, 0, 1, 1, 0)$
$g_3 = (1, 1, 1, 0, 1)$ $g_{13} = (0, 1, 1, 0, 0)$
$g_4 = (1, 0, 1, 1, 0)$ $g_{14} = (0, 0, 1, 1, 1)$
$g_5 = (0, 1, 0, 0, 0)$ $g_{15} = (1, 1, 0, 0, 1)$
$g_6 = (1, 0, 0, 1, 1)$ $g_{16} = (0, 0, 0, 1, 0)$
$g_7 = (1, 0, 1, 1, 1)$ $g_{17} = (0, 0, 1, 1, 0)$
$g_8 = (1, 1, 1, 1, 1)$ $g_{18} = (0, 1, 1, 1, 0)$
$g_9 = (0, 1, 1, 1, 0)$ $g_{19} = (1, 1, 1, 1, 1)$
$g_{10} = (1, 0, 1, 0, 1)$ $g_{20} = (0, 0, 1, 0, 0)$

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - 2.1 sample the model (create a new genotype)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value



1

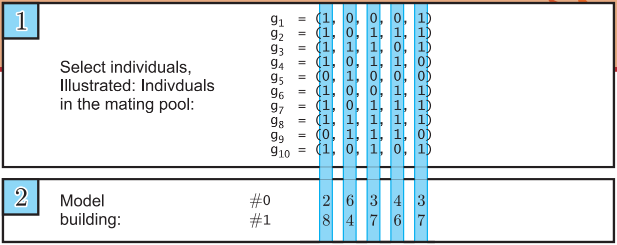
Select individuals, Illustrated: Indivduals in the mating pool:

$g_1 = (1, 0, 0, 0, 1)$
$g_2 = (1, 0, 1, 1, 1)$
$g_3 = (1, 1, 1, 0, 1)$
$g_4 = (1, 0, 1, 1, 0)$
$g_5 = (0, 1, 0, 0, 0)$
$g_6 = (1, 0, 0, 1, 1)$
$g_7 = (1, 0, 1, 1, 1)$
$g_8 = (1, 1, 1, 1, 1)$
$g_9 = (0, 1, 1, 1, 0)$
$g_{10} = (1, 0, 1, 0, 1)$

~~$g_{11} = (0, 0, 1, 0, 0)$~~
~~$g_{12} = (0, 0, 1, 1, 0)$~~
~~$g_{13} = (0, 1, 1, 0, 0)$~~
~~$g_{14} = (0, 0, 1, 1, 1)$~~
~~$g_{15} = (1, 1, 0, 0, 1)$~~
~~$g_{16} = (0, 0, 0, 1, 0)$~~
~~$g_{17} = (0, 0, 1, 1, 0)$~~
~~$g_{18} = (0, 1, 1, 1, 0)$~~
~~$g_{19} = (1, 1, 1, 1, 1)$~~
~~$g_{20} = (0, 0, 1, 0, 0)$~~

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - **2.1** sample the model (create a new genotype)
   - **2.2** perform the genotype-phenotype mapping
   - **2.3** compute the objective value
3. Select mps individuals from the ps new candidate solutions

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - 2.1 sample the model (create a new genotype)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals



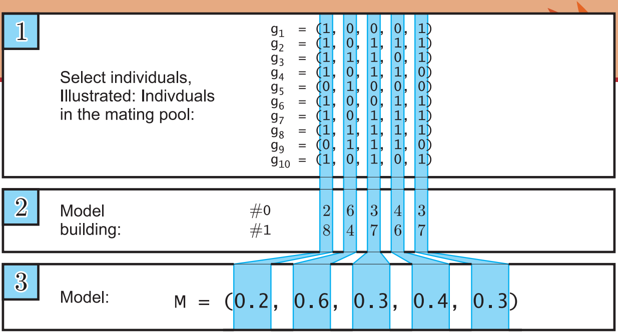
1 Select individuals, Illustrated: Indivduals in the mating pool:

$$
\begin{aligned}
g_1 &= (1, 0, 0, 0, 1)\\
g_2 &= (1, 0, 1, 1, 1)\\
g_3 &= (1, 1, 1, 0, 1)\\
g_4 &= (1, 0, 1, 1, 0)\\
g_5 &= (0, 1, 0, 0, 0)\\
g_6 &= (1, 0, 0, 1, 1)\\
g_7 &= (1, 0, 1, 1, 1)\\
g_8 &= (1, 1, 1, 1, 1)\\
g_9 &= (0, 1, 1, 1, 0)\\
g_{10} &= (1, 0, 1, 0, 1)
\end{aligned}
$$

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - 2.1 sample the model (create a new genotype)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals



1

Select individuals, Illustrated: Indivduals in the mating pool:

$g_1 = (1, 0, 0, 0, 1)$
$g_2 = (1, 0, 1, 1, 1)$
$g_3 = (1, 1, 1, 0, 1)$
$g_4 = (1, 0, 1, 1, 0)$
$g_5 = (0, 1, 0, 0, 0)$
$g_6 = (1, 0, 0, 1, 1)$
$g_7 = (1, 0, 1, 1, 1)$
$g_8 = (1, 1, 1, 1, 1)$
$g_9 = (0, 1, 1, 1, 0)$
$g_{10} = (1, 0, 1, 0, 1)$

2

Model building:

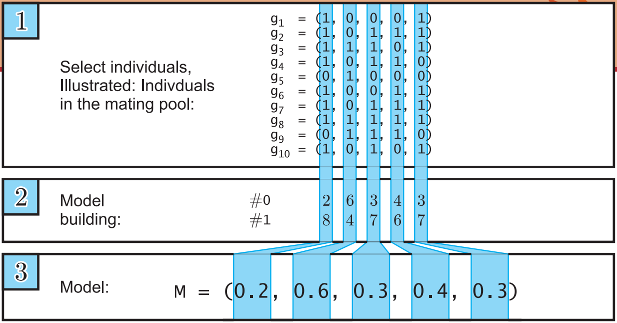
#0	2	6	3	4	3
#1	8	4	7	6	7

3

Model:

`M = (0.2, 0.6, 0.3, 0.4, 0.3)`

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - 2.1 sample the model (create a new genotype)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

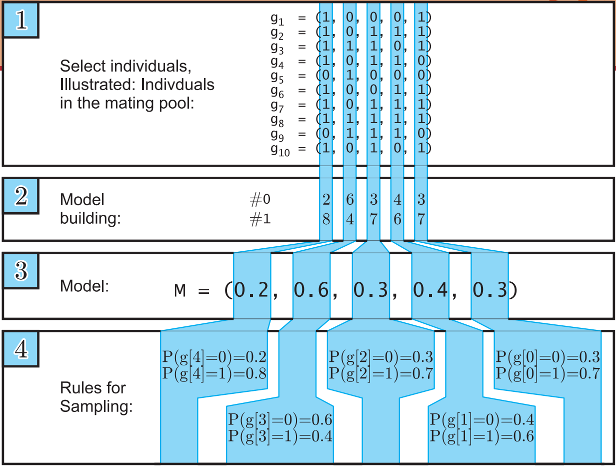
g_1 = (1, 0, 0, 0, 1)
g_2 = (1, 0, 1, 1, 1)
g_3 = (1, 1, 1, 0, 1)
g_4 = (1, 0, 1, 1, 0)
g_5 = (0, 1, 0, 0, 0)
g_6 = (1, 0, 0, 1, 1)
g_7 = (1, 0, 1, 1, 1)
g_8 = (1, 1, 1, 1, 1)
g_9 = (0, 1, 1, 1, 0)
g_{10} = (1, 0, 1, 0, 1)

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: M = (0.2, 0.6, 0.3, 0.4, 0.3)

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. *ps* times do:
   - 2.1 sample the model (create a new genotype)
3. Select *mps* individuals from the *ps* new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

$$
\begin{aligned}
g_1 &= (1, 0, 0, 0, 1)\\
g_2 &= (1, 0, 1, 1, 1)\\
g_3 &= (1, 1, 1, 0, 1)\\
g_4 &= (1, 0, 1, 1, 0)\\
g_5 &= (0, 1, 0, 0, 0)\\
g_6 &= (1, 0, 0, 1, 1)\\
g_7 &= (1, 0, 1, 1, 1)\\
g_8 &= (1, 1, 1, 1, 1)\\
g_9 &= (0, 1, 1, 1, 0)\\
g_{10} &= (1, 0, 1, 0, 1)
\end{aligned}
$$

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: M = (0.2, 0.6, 0.3, 0.4, 0.3)

4 Rules for Sampling:

P(g[4]=0)=0.2
P(g[4]=1)=0.8

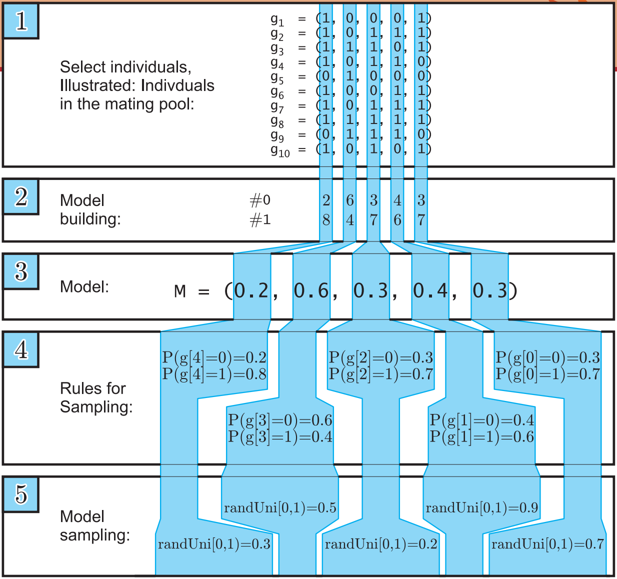
P(g[3]=0)=0.6
P(g[3]=1)=0.4

P(g[2]=0)=0.3
P(g[2]=1)=0.7

P(g[1]=0)=0.4
P(g[1]=1)=0.6

P(g[0]=0)=0.3
P(g[0]=1)=0.7

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - **2.1** sample the model (create a new genotype)
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

$g_1 = (1, 0, 0, 0, 1)$
$g_2 = (1, 0, 1, 1, 1)$
$g_3 = (1, 1, 1, 0, 1)$
$g_4 = (1, 0, 1, 1, 0)$
$g_5 = (0, 1, 0, 0, 0)$
$g_6 = (1, 0, 0, 1, 1)$
$g_7 = (1, 0, 1, 1, 1)$
$g_8 = (1, 1, 1, 1, 1)$
$g_9 = (0, 1, 1, 1, 0)$
$g_{10} = (1, 0, 1, 0, 1)$

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: M = (0.2, 0.6, 0.3, 0.4, 0.3)

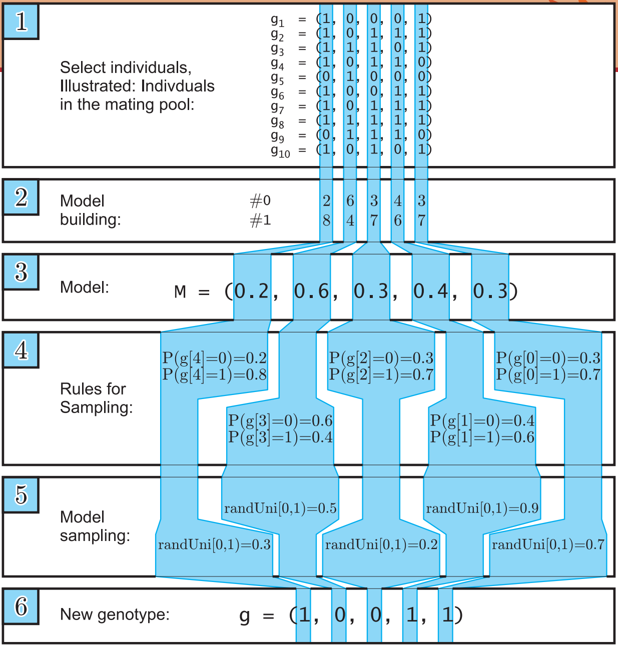
4 Rules for Sampling:

P(g[4]=0)=0.2
P(g[4]=1)=0.8
P(g[3]=0)=0.6
P(g[3]=1)=0.4
P(g[2]=0)=0.3
P(g[2]=1)=0.7
P(g[1]=0)=0.4
P(g[1]=1)=0.6
P(g[0]=0)=0.3
P(g[0]=1)=0.7

5 Model sampling:

randUni[0,1)=0.3
randUni[0,1)=0.5
randUni[0,1)=0.2
randUni[0,1)=0.9
randUni[0,1)=0.7

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   2.1 sample the model (create a new genotype)
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

g_1 = (1, 0, 0, 0, 1)
g_2 = (1, 0, 1, 1, 1)
g_3 = (1, 1, 1, 0, 1)
g_4 = (1, 0, 1, 1, 0)
g_5 = (0, 1, 0, 0, 0)
g_6 = (1, 0, 0, 1, 1)
g_7 = (1, 0, 1, 1, 1)
g_8 = (1, 1, 1, 1, 1)
g_9 = (0, 1, 1, 1, 0)
g_{10} = (1, 0, 1, 0, 1)

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: M = (0.2, 0.6, 0.3, 0.4, 0.3)

4 Rules for Sampling:

P(g[4]=0)=0.2
P(g[4]=1)=0.8

P(g[3]=0)=0.6
P(g[3]=1)=0.4

P(g[2]=0)=0.3
P(g[2]=1)=0.7

P(g[1]=0)=0.4
P(g[1]=1)=0.6

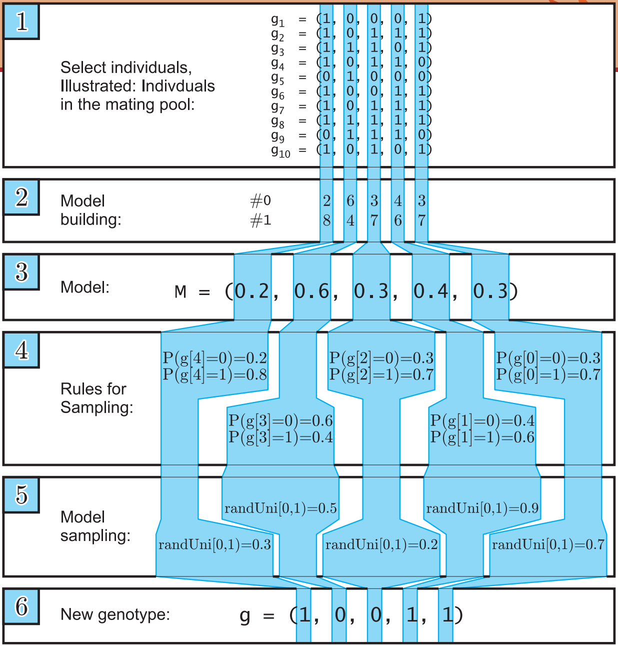
P(g[0]=0)=0.3
P(g[0]=1)=0.7

5 Model sampling:

randUni[0,1)=0.3
randUni[0,1)=0.5
randUni[0,1)=0.2
randUni[0,1)=0.9
randUni[0,1)=0.7

6 New genotype: g = (1, 0, 0, 1, 1)

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - **2.1** sample the model (create a new genotype)
   - **2.2** perform the genotype-phenotype mapping
   - **2.3** compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

```
g1  = (1, 0, 0, 0, 1)
g2  = (1, 0, 1, 1, 1)
g3  = (1, 1, 1, 0, 1)
g4  = (1, 0, 1, 1, 0)
g5  = (0, 1, 0, 0, 0)
g6  = (1, 0, 0, 1, 1)
g7  = (1, 0, 1, 1, 1)
g8  = (1, 1, 1, 1, 1)
g9  = (0, 1, 1, 1, 0)
g10 = (1, 0, 1, 0, 1)
```

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: M = (0.2, 0.6, 0.3, 0.4, 0.3)

4 Rules for Sampling:

P(g[4]=0)=0.2
P(g[4]=1)=0.8

P(g[3]=0)=0.6
P(g[3]=1)=0.4

P(g[2]=0)=0.3
P(g[2]=1)=0.7

P(g[1]=0)=0.4
P(g[1]=1)=0.6

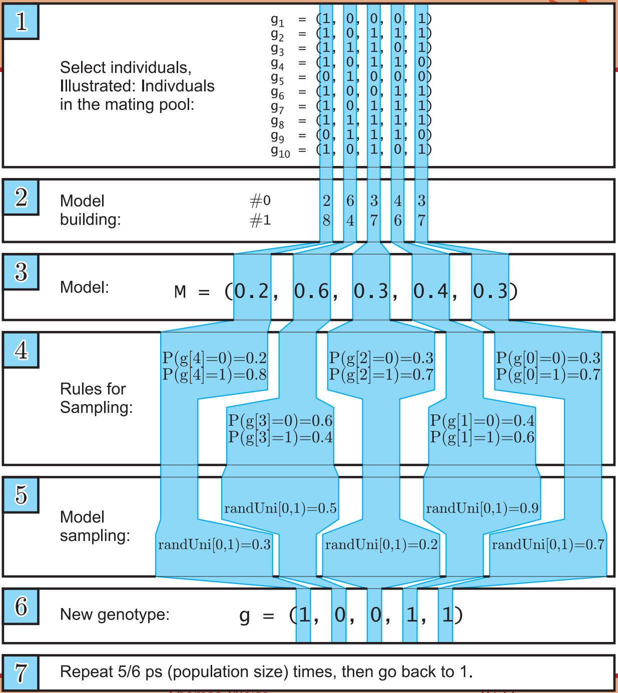
P(g[0]=0)=0.3
P(g[0]=1)=0.7

5 Model sampling:

randUni[0,1)=0.3
randUni[0,1)=0.5
randUni[0,1)=0.2
randUni[0,1)=0.9
randUni[0,1)=0.7

6 New genotype: g = (1, 0, 0, 1, 1)

1. Create a statistical model of possible solutions (UMDA: $(0.5, \cdots, 0.5)$)
2. ps times do:
   - 2.1 sample the model (create a new genotype)
   - 2.2 perform the genotype-phenotype mapping
   - 2.3 compute the objective value
3. Select mps individuals from the ps new candidate solutions
4. Update the model based on the selected individuals
5. If termination criterion not met, go back to 2



1 Select individuals, Illustrated: Indivduals in the mating pool:

```
g1  = (1, 0, 0, 0, 1)
g2  = (1, 0, 1, 1, 1)
g3  = (1, 1, 1, 0, 1)
g4  = (1, 0, 1, 1, 0)
g5  = (0, 1, 0, 0, 0)
g6  = (1, 0, 0, 1, 1)
g7  = (1, 0, 1, 1, 1)
g8  = (1, 1, 1, 1, 1)
g9  = (0, 1, 1, 1, 0)
g10 = (1, 0, 1, 0, 1)
```

2 Model building:

#0	2	6	3	4	3
#1	8	4	7	6	7

3 Model: `M = (0.2, 0.6, 0.3, 0.4, 0.3)`

4 Rules for Sampling:

P(g[4]=0)=0.2
P(g[4]=1)=0.8

P(g[2]=0)=0.3
P(g[2]=1)=0.7

P(g[0]=0)=0.3
P(g[0]=1)=0.7

P(g[3]=0)=0.6
P(g[3]=1)=0.4

P(g[1]=0)=0.4
P(g[1]=1)=0.6

5 Model sampling:

randUni[0,1)=0.5
randUni[0,1)=0.9
randUni[0,1)=0.3
randUni[0,1)=0.2
randUni[0,1)=0.7

6 New genotype: `g = (1, 0, 0, 1, 1)`

7 Repeat 5/6 ps (population size) times, then go back to 1.

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
    - Search space: Fixed-Length Bit Strings of length n

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
  - Search space: Fixed-Length Bit Strings of length n
  - Model: real vector M of length n with $M[i] \in [0, 1] \forall i \in 1 \dots n$

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
  - Search space: Fixed-Length Bit Strings of length n
  - Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \dots n$
  - $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $g^{\star}$ should be 0

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
    - Search space: Fixed-Length Bit Strings of length n
    - Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \dots n$
    - $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $\overset{\star}{g}$ should be 0
    - Initialization: $M[i] = 0.5 \forall i \in 1 \dots n$

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
  - Search space: Fixed-Length Bit Strings of length n
  - Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \dots n$
  - $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $g^{\star}$ should be 0
  - Initialization: $M[i] = 0.5 \forall i \in 1 \dots n$
- Truncation selection: select $mps = 1$ individuals and build temporary model M_T according to UMDA

PBIL

- Population-Based Incremental Learning (PBIL) [3–5]
- Similar to UMDA:
  - Search space: Fixed-Length Bit Strings of length n
  - Model: real vector M of length n with $M[i] \in [0,1] \forall i \in 1 \ldots n$
  - $M[i] =$ estimated proability that the bit at locus i of a globally optimal genotype $g^{\star}$ should be 0
  - Initialization: $M[i] = 0.5 \forall i \in 1 \ldots n$
- Truncation selection: select $mps = 1$ individuals and build temporary model M_T according to UMDA
- Learning rate λ determines influence of the old model M' and temporary model M_T on new model: $M = (1-\lambda)M' + \lambda M_T$

cGA

- Compact Genetic Algorithm (cGA) [6, 7]

cGA

- Compact Genetic Algorithm (cGA) [6, 7]
- Model M similar to UDMA (n-dimensional bit strings)

cGA

- Compact Genetic Algorithm (cGA) [6, 7]

- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.

cGA

- Compact Genetic Algorithm (cGA)[6, 7]:

  only $2n$ bits + n doubles = $66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.

Compact: Ideal for implementation in hardware or on small/weak devices

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:

  only $2n$ bits $+$ n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:

  only $2n$ bits $+$ n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps

$M \longleftarrow \text{buildModelCompGA}(g_1, g_2, M', ps)$

```
In: $M$' – the old model;   Out: $M$ – the new model
begin
    $M \longleftarrow M'$
    if $g_2$ is better than $g_1$ then
      exchange $g_1$ and $g_2$     // $g_1$ is now always better than $g_2$

    for $i \longleftarrow 1$ up to $n$ do
        if $g_1[i] \neq g_2[i]$ then
            if $g_1[i] = 0$ then $M[i] \longleftarrow M'[i] + \frac{1}{ps}$

            else $M[i] \longleftarrow M'[i] - \frac{1}{ps}$
```

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:

  only $2n$ bits $+$ n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps
- In other words:
  - Each set of genotypes has an influence of ${}^{1}/_{ps}$ on the model M

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:
  only $2n$ bits $+$ n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps
- In other words:
  - Each set of genotypes has an influence of $^1/_{ps}$ on the model M
  - Model is modified "into the direction of" the better genotype g_1

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:

  only $2n$ bits + n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps
- In other words:
  - Each set of genotypes has an influence of $^1/_{ps}$ on the model M
  - Model is modified "into the direction of" the better genotype g_1
  - If a bit in g_1 is 1, the zero-probability of that gene in the model is reduced, otherwise it is increased

cGA

- Compact Genetic Algorithm (cGA) [6, 7]:

  only $2n$ bits $+$ n doubles $= 66n$ bits memory!
- Model M similar to UDMA (n-dimensional bit strings)
- In each generation, two genotypes g_1 and g_2 are created and evaluated.
- Virtual population size ps
- In other words:
  - Each set of genotypes has an influence of $^1/_{ps}$ on the model M
  - Model is modified "into the direction of" the better genotype g_1
  - If a bit in g_1 is 1, the zero-probability of that gene in the model is reduced, otherwise it is increased
- Convergence: Achieved when $M[i] \in \{0, 1\} \forall i \in 1 \ldots n$

SHCLVND

- Stochastic Hill Climbing with Learning by Vectors of Normal Distribution (SHCLVND) [8]

SHCLVND

- Stochastic Hill Climbing with Learning by Vectors of Normal Distribution (SHCLVND) [8]
- PBIL version for real-vector based search spaces $\mathbb{G} = \left[\underline{G}, \overline{G}\right]^n$

SHCLVND

- Stochastic Hill Climbing with Learning by Vectors of Normal Distribution (SHCLVND) [8]
- PBIL version for real-vector based search spaces $\mathbb{G} = \left[\underline{G}, \overline{G}\right]^n$
- Model M consists of n-dimensional mean vector $M.\vec{\mu}$ and n-dimensional standard deviation vector $M.\vec{\sigma}$

SHCLVND

- Stochastic Hill Climbing with Learning by Vectors of Normal Distribution (SHCLVND) [8]
- PBIL version for real-vector based search spaces $\mathbb{G} = \left[\underline{G}, \overline{G}\right]^n$
- Model M consists of n-dimensional mean vector $M.\vec{\mu}$ and n-dimensional standard deviation vector $M.\vec{\sigma}$
- The algorithm works as follows

SHCLVND

- Stochastic Hill Climbing with Learning by Vectors of Normal Distribution (SHCLVND) [8]
- PBIL version for real-vector based search spaces $\mathbb{G} = \left[\underline{G}, \overline{G}\right]^n$
- Model M consists of n-dimensional mean vector $M.\vec{\mu}$ and n-dimensional standard deviation vector $M.\vec{\sigma}$
- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{\underline{G}}+\overline{G}}{2}, \frac{\underline{\underline{G}}+\overline{G}}{2}, \cdots, \frac{\underline{\underline{G}}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{\underline{G}}}{2}, \frac{\overline{G}-\underline{\underline{G}}}{2}, \cdots, \frac{\overline{G}-\underline{\underline{G}}}{2})^T$

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{G}}{2}, \frac{\overline{G}-\underline{G}}{2}, \dots, \frac{\overline{G}-\underline{G}}{2})^T$
  3. Sample genotypes g via normal distribution: $g[i] = \ N(M.\vec{\mu}[i], M.\vec{\sigma}[i])$

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{G}}{2}, \frac{\overline{G}-\underline{G}}{2}, \dots, \frac{\overline{G}-\underline{G}}{2})^T$
  3. Sample genotypes g via normal distribution: $g[i] = \ N(M.\vec{\mu}[i], M.\vec{\sigma}[i])$
  4. Create ps genotypes in each generation, select mps genotypes via truncation selection

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{G}}{2}, \frac{\overline{G}-\underline{G}}{2}, \dots, \frac{\overline{G}-\underline{G}}{2})^T$
  3. Sample genotypes g via normal distribution: $g[i] = \ N(M.\vec{\mu}[i], M.\vec{\sigma}[i])$
  4. Create ps genotypes in each generation, select mps genotypes via truncation selection
  5. Compute the mean $\mathrm{mean}(g)$ of the selected mps genotype vectors

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{G}}{2}, \frac{\overline{G}-\underline{G}}{2}, \dots, \frac{\overline{G}-\underline{G}}{2})^T$
  3. Sample genotypes g via normal distribution: $g[i] = \ N(M.\vec{\mu}[i], M.\vec{\sigma}[i])$
  4. Create ps genotypes in each generation, select mps genotypes via truncation selection
  5. Compute the mean $\mathrm{mean}(g)$ of the selected mps genotype vectors
  6. $M.\vec{\mu} = (1-\lambda)M'.\vec{\mu} + \lambda\mathrm{mean}(g)$ (where M' is the old model)

SHCLVND

- The algorithm works as follows:
  1. $M.\vec{\mu}$ is initialized as center of search space, i.e., $M.\vec{\mu} = (\frac{\underline{G}+\overline{G}}{2}, \frac{\underline{G}+\overline{G}}{2}, \dots, \frac{\underline{G}+\overline{G}}{2})^T$
  2. $M.\vec{\sigma}$ is initialized to large values to emulate uniform distribution: $M.\vec{\sigma} = (\frac{\overline{G}-\underline{G}}{2}, \frac{\overline{G}-\underline{G}}{2}, \dots, \frac{\overline{G}-\underline{G}}{2})^T$
  3. Sample genotypes g via normal distribution: $g[i] = \ N(M.\vec{\mu}[i], M.\vec{\sigma}[i])$
  4. Create ps genotypes in each generation, select mps genotypes via truncation selection
  5. Compute the mean $\mathrm{mean}(g)$ of the selected mps genotype vectors
  6. $M.\vec{\mu} = (1-\lambda)M'.\vec{\mu} + \lambda\mathrm{mean}(g)$ (where M' is the old model)
  7. $M.\vec{\sigma} = \sigma_{\text{red}} * M'.\vec{\sigma}$ (where $\sigma_{\text{red}} \in [0, 1)$ reduces the standard deviation step by step)

Section Outline

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions
- Univariat $\equiv$ Implicit assumption: variables are independent from each other

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions
- Univariat $\equiv$ Implicit assumption: variables are independent from each other
- Multi-variate EDAs: represent gene interaction/epistasis information in model

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions
- Univariat $\equiv$ Implicit assumption: variables are independent from each other
- Multi-variate EDAs: represent gene interaction/epistasis information in model
- Real-valued EDA: use covariance matrix

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions
- Univariat $\equiv$ Implicit assumption: variables are independent from each other
- Multi-variate EDAs: represent gene interaction/epistasis information in model
- Real-valued EDA: use covariance matrix
- Bit-String EDAs: Model $=$ Bayesian Networks

Univariate EDAs vs. Multivariate EDAs

- So far, all EDAs we had used n-dimensional univariate distributions
- Univariat $\equiv$ Implicit assumption: variables are independent from each other
- Multi-variate EDAs: represent gene interaction/epistasis information in model
- Real-valued EDA: use covariance matrix
- Bit-String EDAs: Model = Bayesian Networks
- Multi-variate EDAs are more complicated but can deal with epistasis while univariate ones will produce bad results in these cases

Section Outline

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]
- EDA for trees, i.e., same search space as tree-based Genetic Programming [11]

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]
- EDA for trees, i.e., same search space as tree-based Genetic Programming [11]
- Model: $M =$Probabilistic Prototype Tree (PPT)

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]
- EDA for trees, i.e., same search space as tree-based Genetic Programming [11]
- Model: $M =$Probabilistic Prototype Tree (PPT)
- Each node has maximum arity[1] and holds a vector with the probability for each possible child type

[1] number of children

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]
- EDA for trees, i.e., same search space as tree-based Genetic Programming [11]
- Model: $M =$Probabilistic Prototype Tree (PPT)
- Each node has maximum arity[1] and holds a vector with the probability for each possible child type
- The PPT is samples top-down according to the node probabilities

[1] number of children

PIPE

- Probabilistic Incremental Program Evolution (PIPE) [9, 10]
- EDA for trees, i.e., same search space as tree-based Genetic Programming [11]
- Model: M =Probabilistic Prototype Tree (PPT)
- Each node has maximum arity[1] and holds a vector with the probability for each possible child type
- The PPT is samples top-down according to the node probabilities
- Probabilities updated in a rather complex way towards the best tree found

[1] number of children

Section Outline

1. Introduction
2. Univariate EDAs
3. Multivariate EDAs
4. Tree-based EDA
5. Summary

Summary

- EDAs build a model of the perfect solution

Summary

- EDAs build a model of the perfect solution
- Model is sampled in each iteration

Summary

- EDAs build a model of the perfect solution
- Model is sampled in each iteration
- After selection, model is updated

Summary

- EDAs build a model of the perfect solution
- Model is sampled in each iteration
- After selection, model is updated
- Easy to implement for both bit strings and real vectors

Summary

- EDAs build a model of the perfect solution
- Model is sampled in each iteration
- After selection, model is updated
- Easy to implement for both bit strings and real vectors
- Univariate EDAs: simple

Summary

- EDAs build a model of the perfect solution
- Model is sampled in each iteration
- After selection, model is updated
- Easy to implement for both bit strings and real vectors
- Univariate EDAs: simple
- Multivariate EDAs: more complicated, but can work better if epistasis is present

Summary

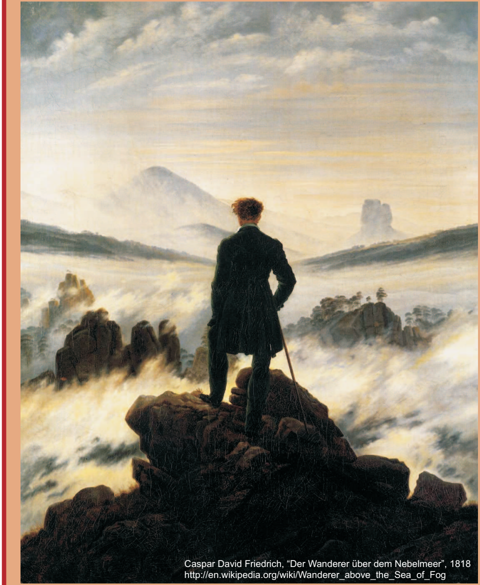
- EDAs build a model of the perfect solution
- Model is sampled in each iteration
- After selection, model is updated
- Easy to implement for both bit strings and real vectors
- Univariate EDAs: simple
- Multivariate EDAs: more complicated, but can work better if epistasis is present
- Anything which can be evolved with an EA can be evolved with an EDA (and vice versa)

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. Heinz Mühlenbein and Gerhard Paaß. From recombination of genes to the estimation of distributions i. binary parameters. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN IV)*, volume 1141/1996 of *Lecture Notes in Computer Science (LNCS)*, pages 178–187, Berlin, Germany, September 22–24, 1996. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-61723-X_982. URL ftp://ftp.ais.fraunhofer.de/pub/as/ga/gmd_as_ga-96_04.ps.
2. Heinz Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation*, 5(3): 303–346, Fall 1997. doi: 10.1162/evco.1997.5.3.303. URL http://citeseer.ist.psu.edu/old/730919.html.
3. Shumeet Baluja. Population-based incremental learning – a method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Pittsburgh, PA, USA: Carnegy Mellon University (CMU), School of Computer Science, Computer Science Department, June 2, 1994. URL http://www.ri.cmu.edu/pub_files/pub1/baluja_shumeet_1994_2/baluja_shumeet_1994_2.pdf.
4. Shumeet Baluja and Richard A. Caruana. Removing the genetics from the standard genetic algorithm. In Armand Prieditis and Stuart J. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 38–46, Tahoe City, CA, USA, July 9–12, 1995. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. URL http://www.cs.cornell.edu/~caruana/ml95.ps.
5. Shumeet Baluja. An empirical comparison of seven iterative and evolutionary function optimization heuristics. Technical Report CMU-CS-95-193, Pittsburgh, PA, USA: Carnegy Mellon University (CMU), School of Computer Science, Computer Science Department, September 1, 1995. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.1108.
6. Georges Raif Harik, Fernando G. Lobo, and David Edward Goldberg. The compact genetic algorithm. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(4):287–297, November 1999. doi: 10.1109/4235.797971.
7. Georges Raif Harik, Fernando G. Lobo, and David Edward Goldberg. The compact genetic algorithm. IlliGAL Report 97006, Urbana-Champaign, IL, USA: University of Illinois at Urbana-Champaign, Department of Computer Science, Department of General Engineering, Illinois Genetic Algorithms Laboratory (IlliGAL), August 1997. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.906.
8. Stefan Rudlof and Mario Köppen. Stochastic hill climbing with learning by vectors of normal distribution. In Takeshi Furuhashi, editor, *First Online Workshop on Soft Computing (WSC1)*, pages 60–70, Nagoya, Aichi, Japan: Nagoya University, August 19–30, 1996. URL http://eprints.kfupm.edu.sa/66958/1/66958.pdf. Second corrected and enhanced version, 1997-09-03.

Bibliography II

9. Rafał Sałustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution: Stochastic search through program space. In Maarten van Someren and Gerhard Widmer, editors, *9th European Conference on Machine Learning (ECML'97)*, volume 1224/1997 of *Lecture Notes in Computer Science (LNCS)*, pages 213–220, Prague, Czech Republic, April 23–25, 1997. Berlin, Germany: Springer-Verlag GmbH. doi: 10.1007/3-540-62858-4_86. URL `ftp://ftp.idsia.ch/pub/juergen/ECML_PIPE.ps.gz`.
10. Rafał Sałustowicz and Jürgen Schmidhuber. Probabilistic incremental program evolution: Stochastic search through program space. *Evolutionary Computation*, 5(2):123–141, February 1997. doi: 10.1162/evco.1997.5.2.123.
11. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books. Cambridge, MA, USA: MIT Press, December 1992. ISBN 0-262-11170-5 and 978-0-262-11170-6. URL `http://books.google.de/books?id=Bhtxo60BV0EC`. 1992 first edition, 1993 second edition.