Estimation-of-Distribution Algorithms. Discrete Domain.

Petr Pošík

Dept. of Cybernetics
CTU FEE

Introduction to EDAs

Genetic Algorithms and Epistasis

From the lecture on epistasis: $x^{\text{best}} = 111\ldots11$, $f(x^{\text{best}}) = 40$

GA works:

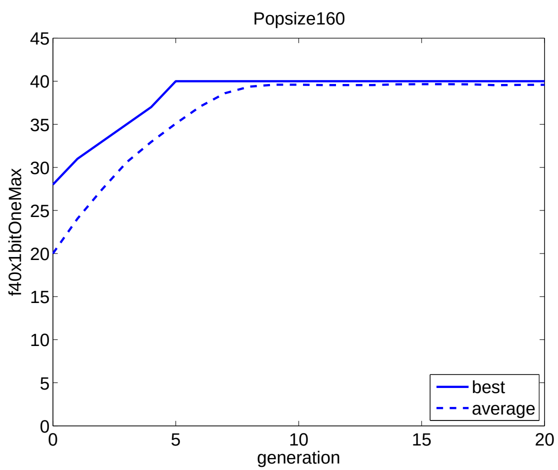
- ✔ no dependencies

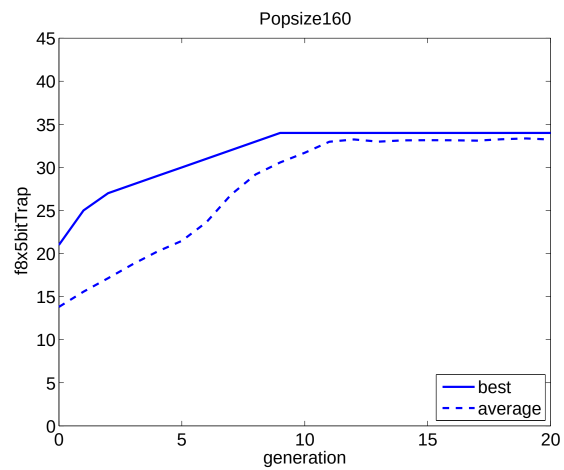
GA fails:

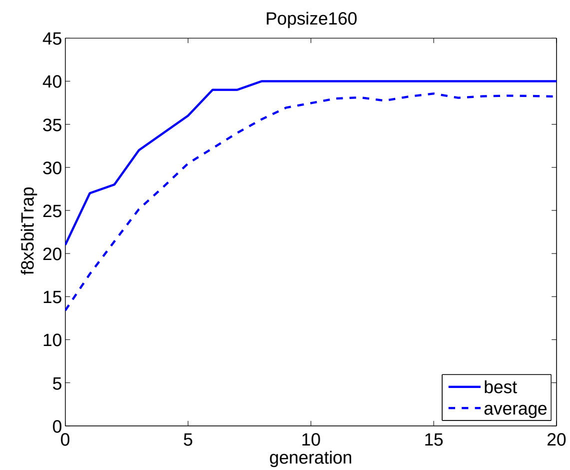
- ✔ deps. exist
- ✔ GA not able to work with them

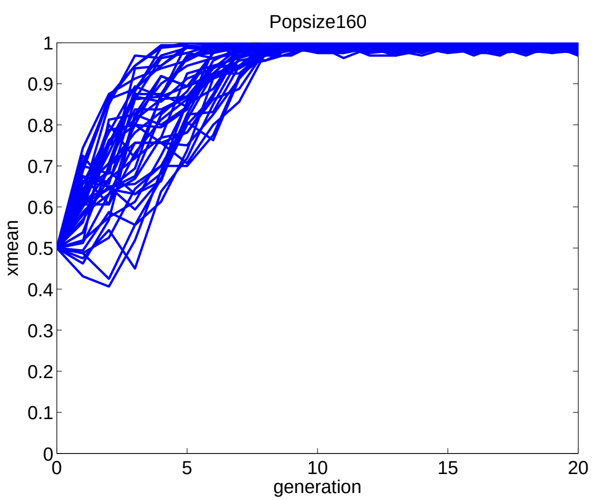
GA works again:

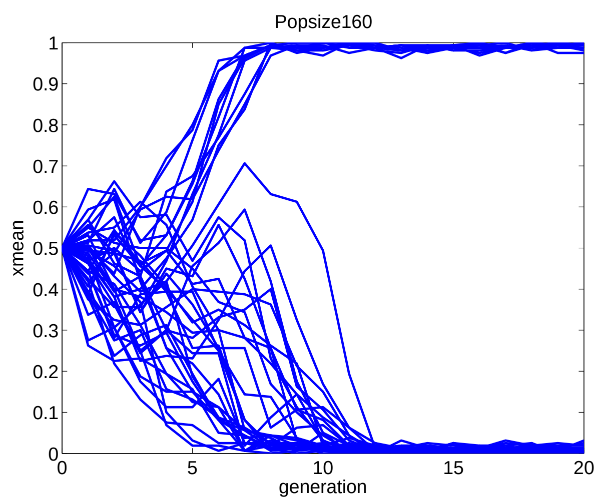
- ✔ deps. exist
- ✔ GA knows about them

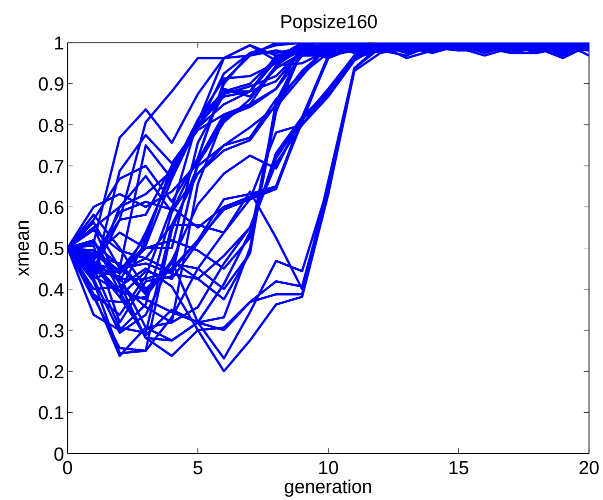












Algorithm 1: Genetic Algorithm

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Cross over the parents, create offspring.
        Mutate offspring.
        Incorporate offspring into the population.
```

Select → cross over → mutate approach

Conventional GA operators

- ✔ are not adaptive, and
- ✔ cannot (or ususally do not) discover and use *the interactions among solution components*.

Genetic Algorithms

Algorithm 1: Genetic Algorithm

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Cross over the parents, create offspring.
        Mutate offspring.
        Incorporate offspring into the population.
```

Select → cross over → mutate approach

Conventional GA operators

- ✔ are not adaptive, and
- ✔ cannot (or ususally do not) discover and use *the interactions among solution components*.

What does an intearction mean?

- ✔ we would like to create a new offspring by mutation
- ✔ we would like the offspring to have better, or at least the same, quality as the parent
- ✔ if we must modify x_i together with x_j to reach the desired goal (if it is not possible to improve the solution by modifying either x_i or x_j only), then x_i interacts with x_j.

Genetic Algorithms

Algorithm 1: Genetic Algorithm

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Cross over the parents, create offspring.
        Mutate offspring.
        Incorporate offspring into the population.
```

Select $\rightarrow$ cross over $\rightarrow$ mutate approach

Conventional GA operators

- ✔ are not adaptive, and
- ✔ cannot (or ususally do not) discover and use *the interactions among solution components*.

What does an intearction mean?

- ✔ we would like to create a new offspring by mutation
- ✔ we would like the offspring to have better, or at least the same, quality as the parent
- ✔ if we must modify x_i together with x_j to reach the desired goal (if it is not possible to improve the solution by modifying either x_i or x_j only), then x_i interacts with x_j.

The goal of recombination operators:

- ✔ Intensify the search in areas which contained "good" individuals in previous iterations.

Genetic Algorithms

Algorithm 1: Genetic Algorithm

begin
  Initialize the population.
  while *termination criteria are not met* **do**
    Select parents from the population.
    Cross over the parents, create offspring.
    Mutate offspring.
    Incorporate offspring into the population.

Select → cross over → mutate approach

Conventional GA operators

- ✔ are not adaptive, and
- ✔ cannot (or ususally do not) discover and use *the interactions among solution components*.

What does an intearction mean?

- ✔ we would like to create a new offspring by mutation
- ✔ we would like the offspring to have better, or at least the same, quality as the parent
- ✔ if we must modify x_i together with x_j to reach the desired goal (if it is not possible to improve the solution by modifying either x_i or x_j only), then x_i interacts with x_j.

The goal of recombination operators:

- ✔ Intensify the search in areas which contained "good" individuals in previous iterations.
- ✔ Must be able to take the interactions into account.

Genetic Algorithms

Algorithm 1: Genetic Algorithm

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Cross over the parents, create offspring.
        Mutate offspring.
        Incorporate offspring into the population.
```

Select → cross over → mutate approach

Conventional GA operators

- ✔ are not adaptive, and
- ✔ cannot (or ususally do not) discover and use *the interactions among solution components*.

What does an intearction mean?

- ✔ we would like to create a new offspring by mutation
- ✔ we would like the offspring to have better, or at least the same, quality as the parent
- ✔ if we must modify x_i together with x_j to reach the desired goal (if it is not possible to improve the solution by modifying either x_i or x_j only), then x_i interacts with x_j.

The goal of recombination operators:

- ✔ Intensify the search in areas which contained "good" individuals in previous iterations.
- ✔ Must be able to take the interactions into account.
- ✔ Why not directly describe the distribution of "good" individuals???

Algorithm 1: Genetic Algorithm

begin
  Initialize the population.
  while *termination criteria are not met* **do**
    Select parents from the population.
    Cross over the parents, create offspring.
    Mutate offspring.
    Incorporate offspring into the population.

Select → cross over → mutate approach

Algorithm 2: Estimation-of-Distribution Alg.

begin
  Initialize the population.
  while *termination criteria are not met* **do**
    Select parents from the population.
    Learn a model of their distribution.
    Sample new individuals.
    Incorporate offspring into the population.

Select → model → sample approach

GA vs EDA

Algorithm 1: Genetic Algorithm

begin
  Initialize the population.
  while *termination criteria are not met* **do**
    Select parents from the population.
    Cross over the parents, create offspring.
    Mutate offspring.
    Incorporate offspring into the population.

Select $\rightarrow$ cross over $\rightarrow$ mutate approach

Algorithm 2: Estimation-of-Distribution Alg.

begin
  Initialize the population.
  while *termination criteria are not met* **do**
    Select parents from the population.
    Learn a model of their distribution.
    Sample new individuals.
    Incorporate offspring into the population.

Select $\rightarrow$ model $\rightarrow$ sample approach

Explicit probabilistic model:

- ✔ principled way of working with dependencies
- ✔ adaptation ability (different behavior in different stages of evolution)

GA vs EDA

Algorithm 1: Genetic Algorithm

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Cross over the parents, create offspring.
        Mutate offspring.
        Incorporate offspring into the population.
```

Select $\rightarrow$ cross over $\rightarrow$ mutate approach

Algorithm 2: Estimation-of-Distribution Alg.

```
begin
    Initialize the population.
    while termination criteria are not met do
        Select parents from the population.
        Learn a model of their distribution.
        Sample new individuals.
        Incorporate offspring into the population.
```

Select $\rightarrow$ model $\rightarrow$ sample approach

Explicit probabilistic model:

- ✔ principled way of working with dependencies
- ✔ adaptation ability (different behavior in different stages of evolution)

Names:

EDA Estimation-of-Distribution Algorithm

PMBGA Probabilistic Model-Building Genetic Algorithm

IDEA Iterated Density Estimation Algorithm

Content of the lectures

1. EDA for discrete domains (e.g. binary)
   - ✔ Motivation example
   - ✔ Without interactions
   - ✔ Pairwise interactions
   - ✔ Higher order interactions
2. EDA for real domain (vectors of real numbers)
   - ✔ Evolution strategies
   - ✔ Histograms
   - ✔ Gaussian distribution and its mixtures

Motivation Example

5-bit OneMax (CountOnes) problem:

- $f_{\text{Dx1bitOneMax}}(\mathbf{x}) = \sum_{d=1}^{D} x_d$
- Optimum: `11111`, fitness: 5

Algorithm: **Univariate Marginal Distribution Algorithm (UMDA)**

- Population size: 6
- Tournament selection: $t = 2$
- **Model:** vector of probabilities $p = (p_1, \ldots, p_D)$
  - each p_d is the probability of observing `1` at dth element
- **Model learning:**
  - compute p from selected individuals
- **Model sampling:**
  - generate `1` on dth position with probability p_d (independently of other positions)

Selection, Modeling, Sampling

Old population:

11001	(3)
00010	(1)
11101	(4)
10111	(4)
00001	(1)
10010	(2)

Tournaments:

11101	(4)	vs.	10111	(4)
10111	(4)	vs.	11101	(4)
11101	(4)	vs.	00001	(1)
10010	(2)	vs.	00010	(1)
00010	(1)	vs.	00010	(1)
00010	(1)	vs.	11001	(3)

Selected parents:

11101	(4)
10111	(4)
11101	(4)
10010	(2)
00010	(1)
11001	(3)

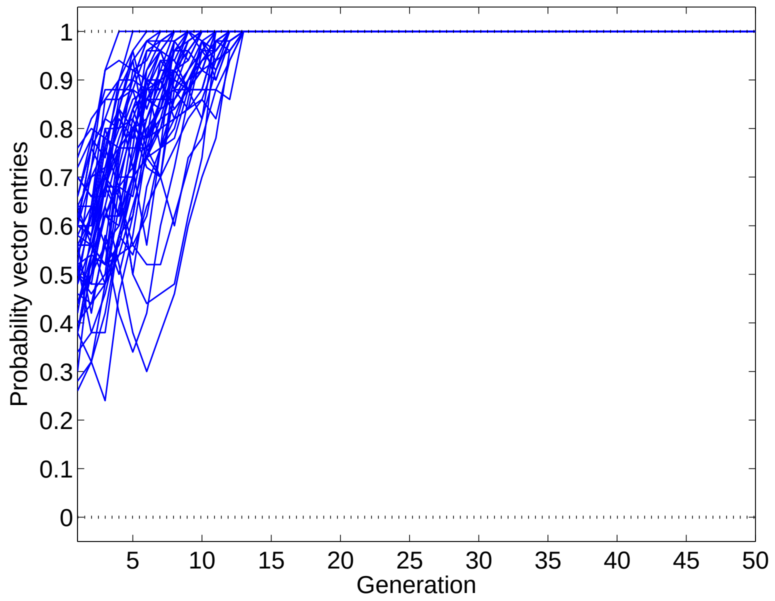
Probability vector:

$\frac{5}{6}\ \frac{3}{6}\ \frac{3}{6}\ \frac{3}{6}\ \frac{4}{6}$

Offspring:

11000	(2)
10100	(2)
11011	(4)
01011	(3)
10101	(3)
10111	(4)

UMDA Behaviour for OneMax problem



- ✔ 1s are better then 0s on average, selection increases the proportion of 1s
- ✔ Recombination preserves and combines 1s, the ratio of 1s increases over time
- ✔ If we have many 1s in population, we cannot miss the optimum

The number of evaluations needed for reliable convergence:

Algorithm	Nr. of evaluations
UMDA	$\mathcal{O}(D \ln D)$
Hill-Climber	$\mathcal{O}(D \ln D)$
GA with uniform xover	approx. $\mathcal{O}(D \ln D)$
GA with 1-point xover	a bit slower

UMDA behaves similarly to GA with uniform crossover!

What about a different fitness?

For OneMax function:

- ✔ UMDA works well, all the bits probably eventually converge to the right value.

Will UMDA be similarly successful for other fitness functions?

- ✔ Well, ...no. :-(

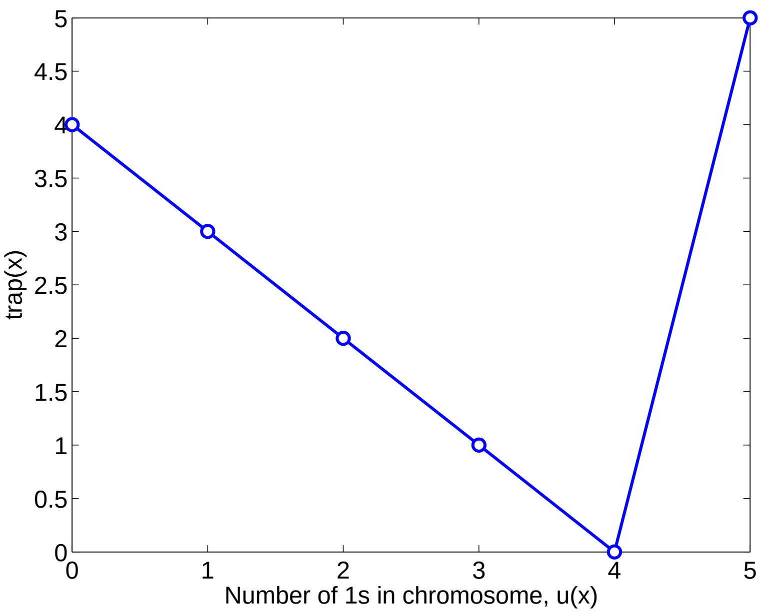
Problem: **Concatanated 5-bit traps**

$$f = f_{\text{trap}}(x_1, x_2, x_3, x_4, x_5) + \\ + f_{\text{trap}}(x_6, x_7, x_8, x_9, x_{10}) + \\ + \dots$$

The *trap* function is defined as

$$f_{\text{trap}}(\mathbf{x}) = \begin{cases} 5 & \text{if } u(\mathbf{x}) = 5 \\ 4 - u(\mathbf{x}) & \text{otherwise} \end{cases}$$

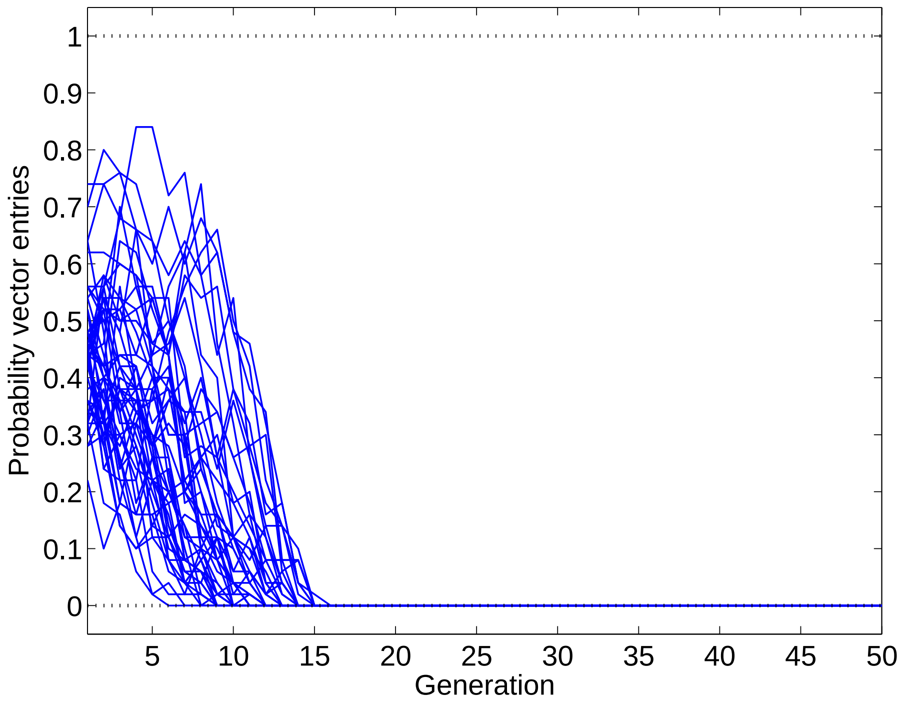
where $u(\mathbf{x})$ is the so called *unity* function and returns the number of 1s in $\mathbf{x}$ (it is actually the One Max function).



UMDA behaviour on concatanated traps

Traps:

- ✔ Optimum in `111111...1`
- ✔ But $f_{\text{trap}}(\texttt{0}****) = 2$ while $f_{\text{trap}}(\texttt{1}****) = 1.375$
- ✔ 1-dimensional probabilities lead the GA to the wrong way!
- ✔ Exponentially increasing population size is needed, otherwise GA will not find optimum reliably.



What can be done about traps?

The f_{trap} function is *deceptive*:

- ✔ Statistics over `1****` and `0****` do not lead us to the right solution
- ✔ The same holds for statistics over `11***` and `00***`, `111**` and `000**`, `1111*` and `0000*`

What can be done about traps?

The f_{trap} function is *deceptive*:

- ✔ Statistics over `1****` and `0****` do not lead us to the right solution
- ✔ The same holds for statistics over `11***` and `00***`, `111**` and `000**`, `1111*` and `0000*`
- ✔ Harder than the *needle-in-the-haystack* problem:
  - ✘ regular haystack simply does not provide any information, where to search for the needle
  - ✘ f_{trap}-haystack actively lies to you—it points you to the wrong part of the haystack

What can be done about traps?

The f_{trap} function is *deceptive*:

- ✔ Statistics over `1****` and `0****` do not lead us to the right solution
- ✔ The same holds for statistics over `11***` and `00***`, `111**` and `000**`, `1111*` and `0000*`
- ✔ Harder than the *needle-in-the-haystack* problem:
  - ✘ regular haystack simply does not provide any information, where to search for the needle
  - ✘ f_{trap}-haystack actively lies to you—it points you to the wrong part of the haystack
- ✔ But: $f_{\text{trap}}(\texttt{00000}) < f_{\text{trap}}(\texttt{11111})$, `11111` will be better than `00000` on average
- ✔ 5bit statistics should work for 5bit traps in the same way as 1bit statistics work for OneMax problem!

What can be done about traps?

The f_{trap} function is *deceptive*:

- ✔ Statistics over `1****` and `0****` do not lead us to the right solution
- ✔ The same holds for statistics over `11***` and `00***`, `111**` and `000**`, `1111*` and `0000*`
- ✔ Harder than the *needle-in-the-haystack* problem:
  - ✘ regular haystack simply does not provide any information, where to search for the needle
  - ✘ f_{trap}-haystack actively lies to you—it points you to the wrong part of the haystack
- ✔ But: $f_{\text{trap}}(\texttt{00000}) < f_{\text{trap}}(\texttt{11111})$, `11111` will be better than `00000` on average
- ✔ 5bit statistics should work for 5bit traps in the same way as 1bit statistics work for OneMax problem!

Model learning:

- ✔ build model for each 5-tuple of bits
- ✔ compute $p(\texttt{00000}), p(\texttt{00001}), \ldots, p(\texttt{11111})$,

What can be done about traps?

The f_{trap} function is *deceptive*:

- ✔ Statistics over `1****` and `0****` do not lead us to the right solution
- ✔ The same holds for statistics over `11***` and `00***`, `111**` and `000**`, `1111*` and `0000*`
- ✔ Harder than the *needle-in-the-haystack* problem:
  - ✘ regular haystack simply does not provide any information, where to search for the needle
  - ✘ f_{trap}-haystack actively lies to you—it points you to the wrong part of the haystack
- ✔ But: $f_{\text{trap}}(00000) < f_{\text{trap}}(11111)$, `11111` will be better than `00000` on average
- ✔ 5bit statistics should work for 5bit traps in the same way as 1bit statistics work for OneMax problem!

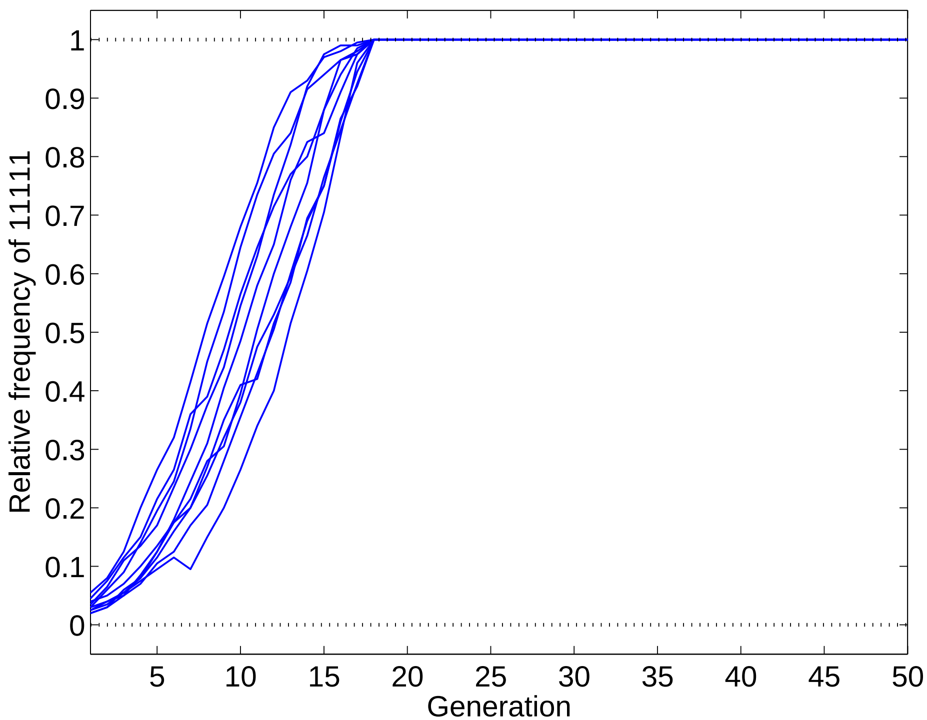
Model learning:

- ✔ build model for each 5-tuple of bits
- ✔ compute $p(00000), p(00001), \ldots, p(11111)$,

Model sampling:

- ✔ Each 5-tuple of bits is generated independently
- ✔ Generate `00000` with probability $p(00000)$, `00001` with probability $p(00001)$, ...

Good statistics work great!



Algorithm	Nr. of evaluations
UMDA with 5bit BB	$\mathcal{O}(D \ln D)$ (WOW!)
Hill-Climber	$\mathcal{O}(D^k \ln D)$, $k = 5$
GA with uniform xover	approx. $\mathcal{O}(2^D)$
GA with 1-point xover	similar to unif. xover

What shall we do next?

If we were able to

- ✔ find good statistics with a small overhead, and
- ✔ use them in the UMDA framework,

we would be able to solve order-k separable problems using $\mathcal{O}(D^2)$ evaluations.

- ✔ ... and there are many problems of this type.

The problem solution is closely related to the so-called *linkage learning*, i.e. discovering and using statistical dependencies among variables.

Discrete EDAs

Discrete EDAs: Overview

1. Overview:
   (a) Univariate models (without interactions)
   (b) Bivariate models (pairwise dependencies)
   (c) Multivariate models (higher order interactions)
2. Conclusions

EDAs without interactions

EDAs without interactions

1. **Population-based incremental learning (PBIL)**
   Baluja, 1994
2. **Univariate marginal distribution algorithm (UMDA)**
   Mühlenbein and Paaß, 1996
3. **Compact genetic algorithm (cGA)**
   Harik, Lobo, Goldberg, 1998

Similarities:

- ✔ all of them use a vector of probabilities

Differences:

- ✔ PBIL and cGA do not use population (only the vector p); UMDA does
- ✔ PBIL and cGA use different rules for the adaptation of p

Advantages:

- ✔ Simplicity
- ✔ Speed
- ✔ Simple simulation of large populations

Limitations:

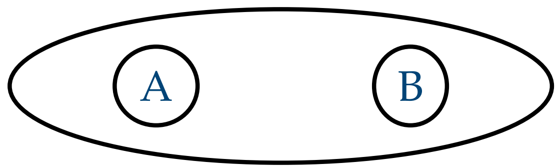
- ✔ Solves reliably only order-1 decomposable problems

EDAs with Pairwise Interactions

From single bits to pairwise models

How to describe two positions together?

- ✔ Using the joint probability distribution:

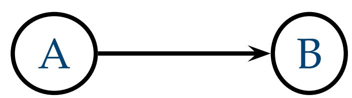


Number of free parameters:

$p(A,B)$

		B	
		0	1
A	0	$p(0,0)$	$p(0,1)$
	1	$p(1,0)$	$p(1,1)$

- ✔ Using statistical dependence:

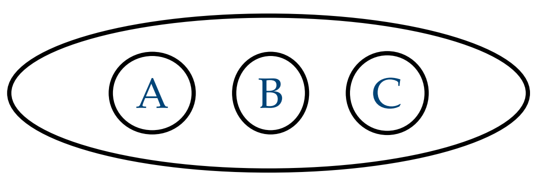


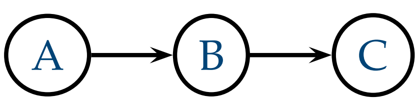
Number of free parameters:

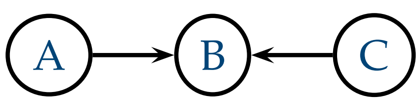
$p(A,B) = p(B|A) \cdot p(A)$:

$p(B=1|A=0)$
$p(B=1|A=1)$
$p(A=1)$

Question: what is the number of parameters in case of the following models?



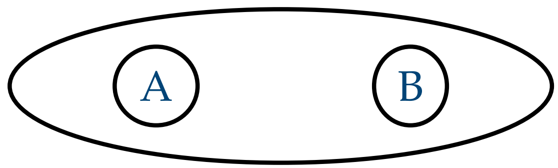




From single bits to pairwise models

How to describe two positions together?

✔ Using the joint probability distribution:

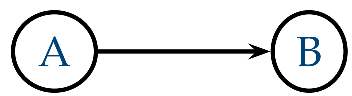


Number of free parameters: 3

$p(A, B)$

		B	
		0	1
A	0	$p(0,0)$	$p(0,1)$
	1	$p(1,0)$	$p(1,1)$

✔ Using statistical dependence:

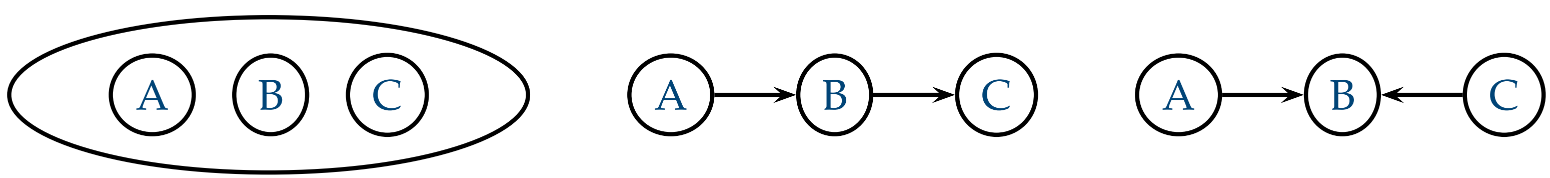


Number of free parameters:

$p(A, B) = p(B|A) \cdot p(A)$:

$p(B = 1|A = 0)$
$p(B = 1|A = 1)$
$p(A = 1)$

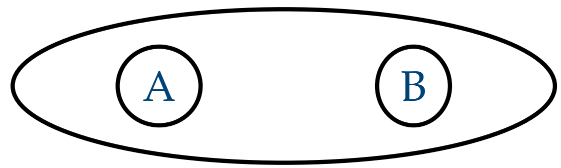
Question: what is the number of parameters in case of the following models?



From single bits to pairwise models

How to describe two positions together?

- ✔ Using the joint probability distribution:

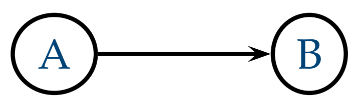


Number of free parameters: 3

$p(A, B)$

		B = 0	B = 1
A	0	$p(0,0)$	$p(0,1)$
	1	$p(1,0)$	$p(1,1)$

- ✔ Using statistical dependence:

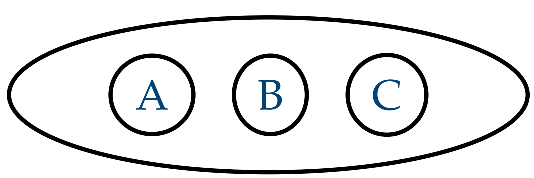


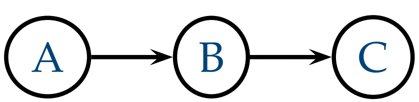
Number of free parameters: 3

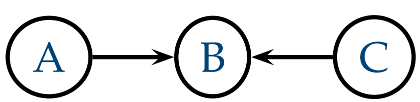
$p(A, B) = p(B|A) \cdot p(A)$:

$p(B = 1|A = 0)$
$p(B = 1|A = 1)$
$p(A = 1)$

Question: what is the number of parameters in case of the following models?







Example with pairwise dependencies: dependency tree

- ✔ Nodes: binary variables (loci of chromozome)
- ✔ Edges: dependencies among variables
- ✔ Features:
  - ✘ Each node depends at most on 1 other node
  - ✘ Graph does not contain cycles
  - ✘ Graph is connected

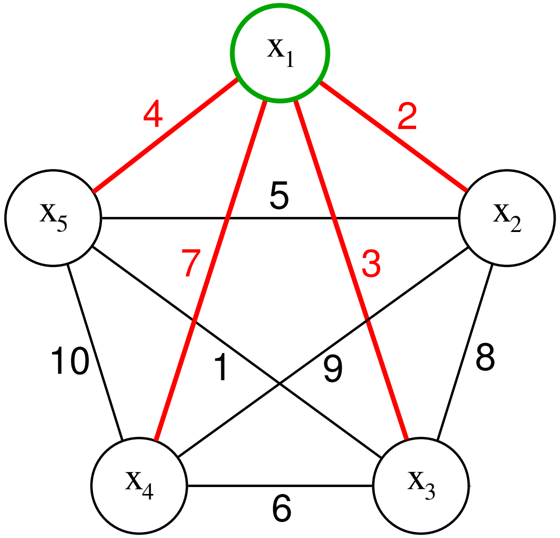
Learning the structure of dependency tree:

1. Score the edges using mutual information:

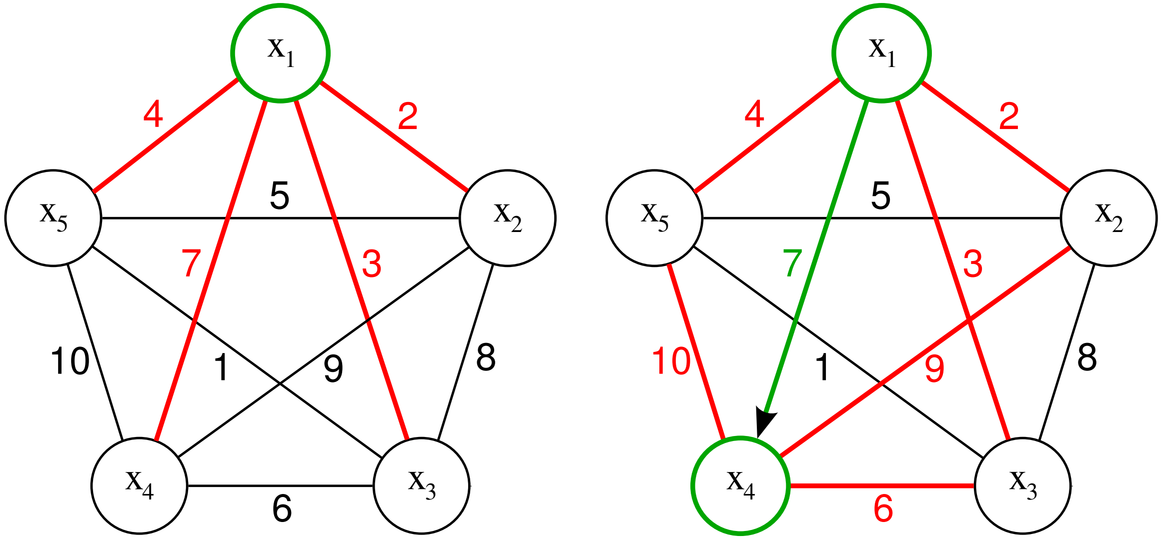
$$I(X,Y) = \sum_{x,y} p(x,y) \cdot \log \frac{p(x,y)}{p(x)p(y)}$$

2. Use any algorithm to determine the maximum spanning tree of the graph, e.g. Prim (1957)
   (a) Start building the tree from any node
   (b) Add such a node that is connected to the tree by the edge with maximum score

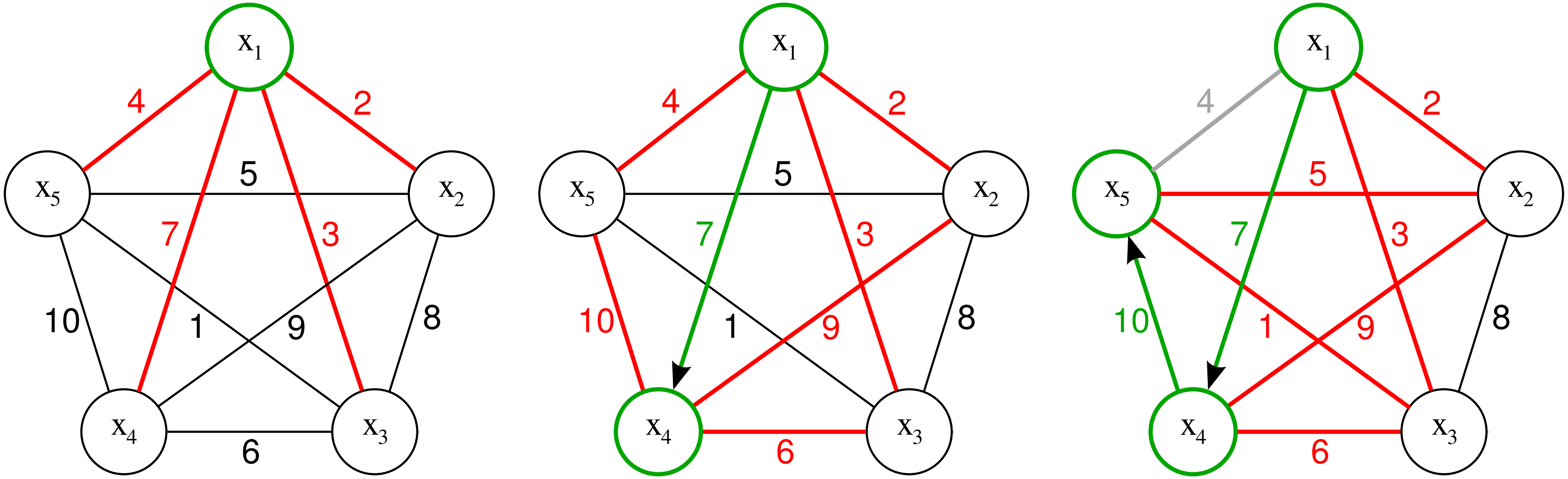
Example of dependency tree learning



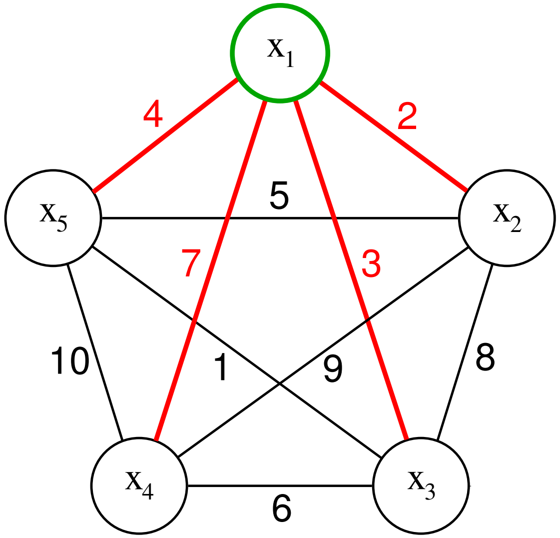
Example of dependency tree learning

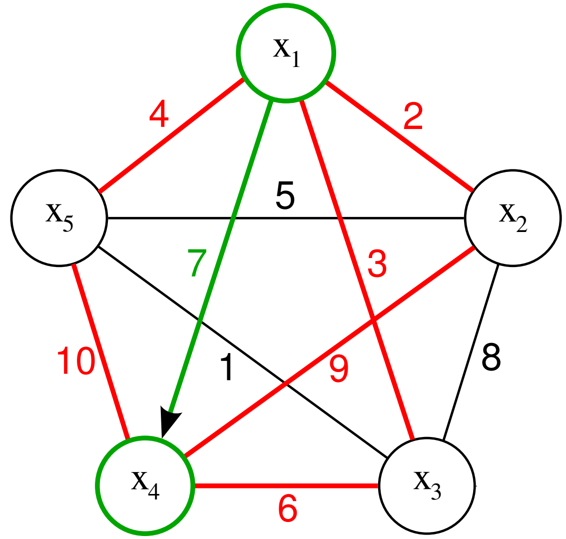


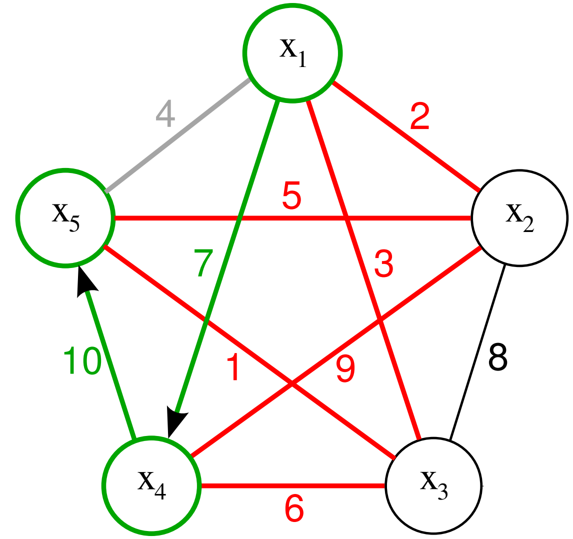
Example of dependency tree learning

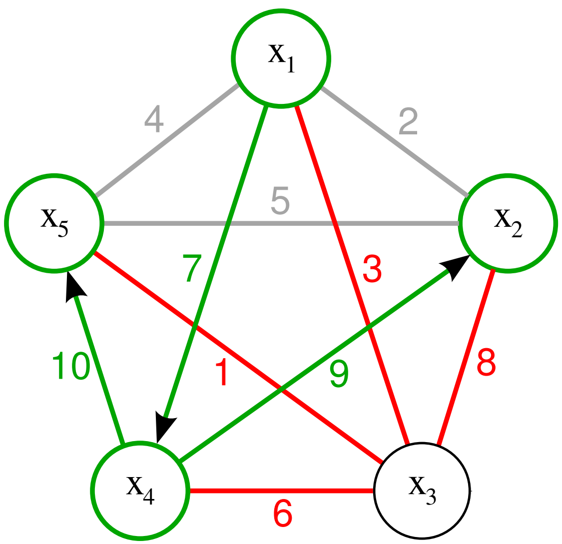


Example of dependency tree learning

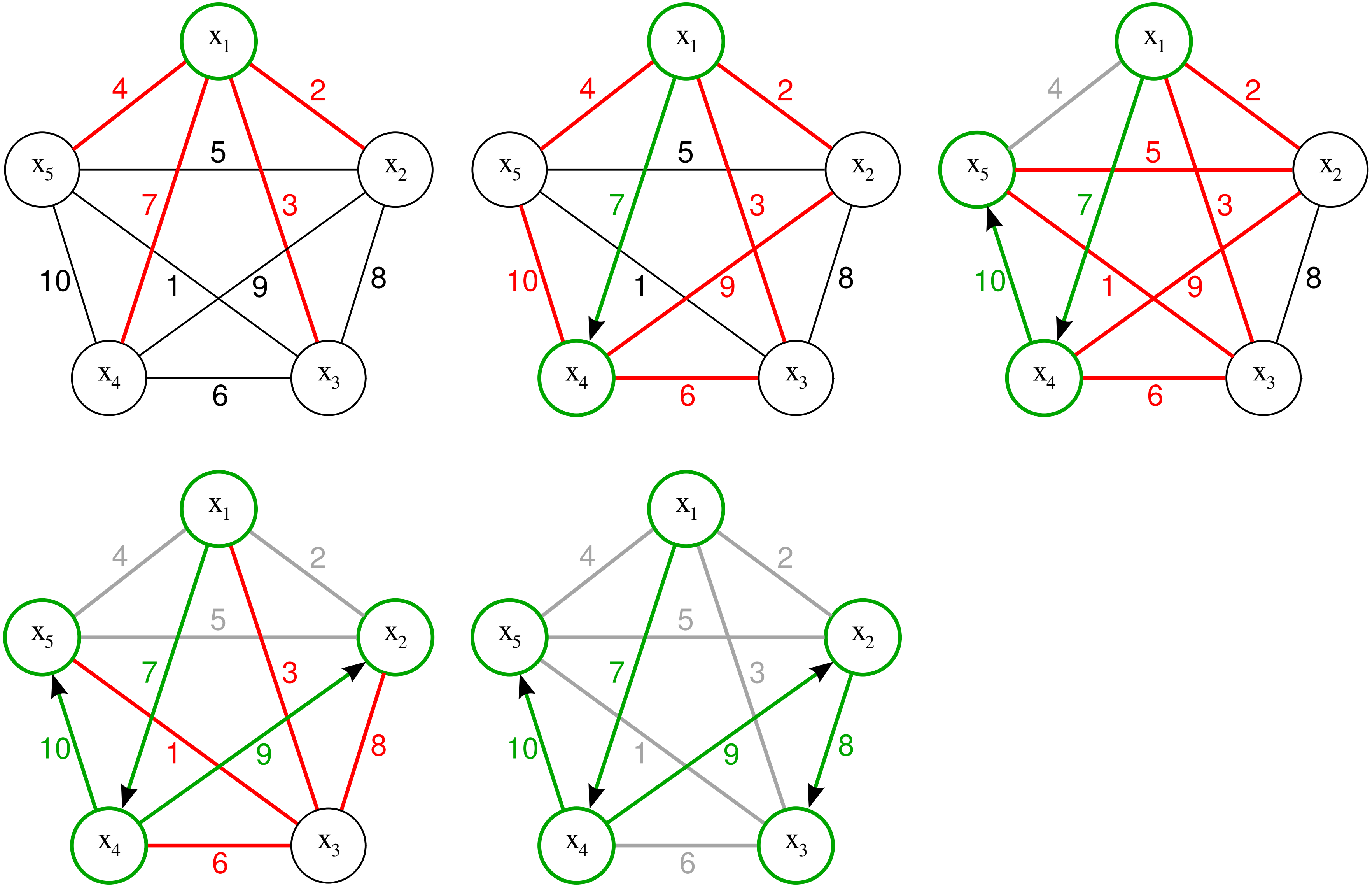




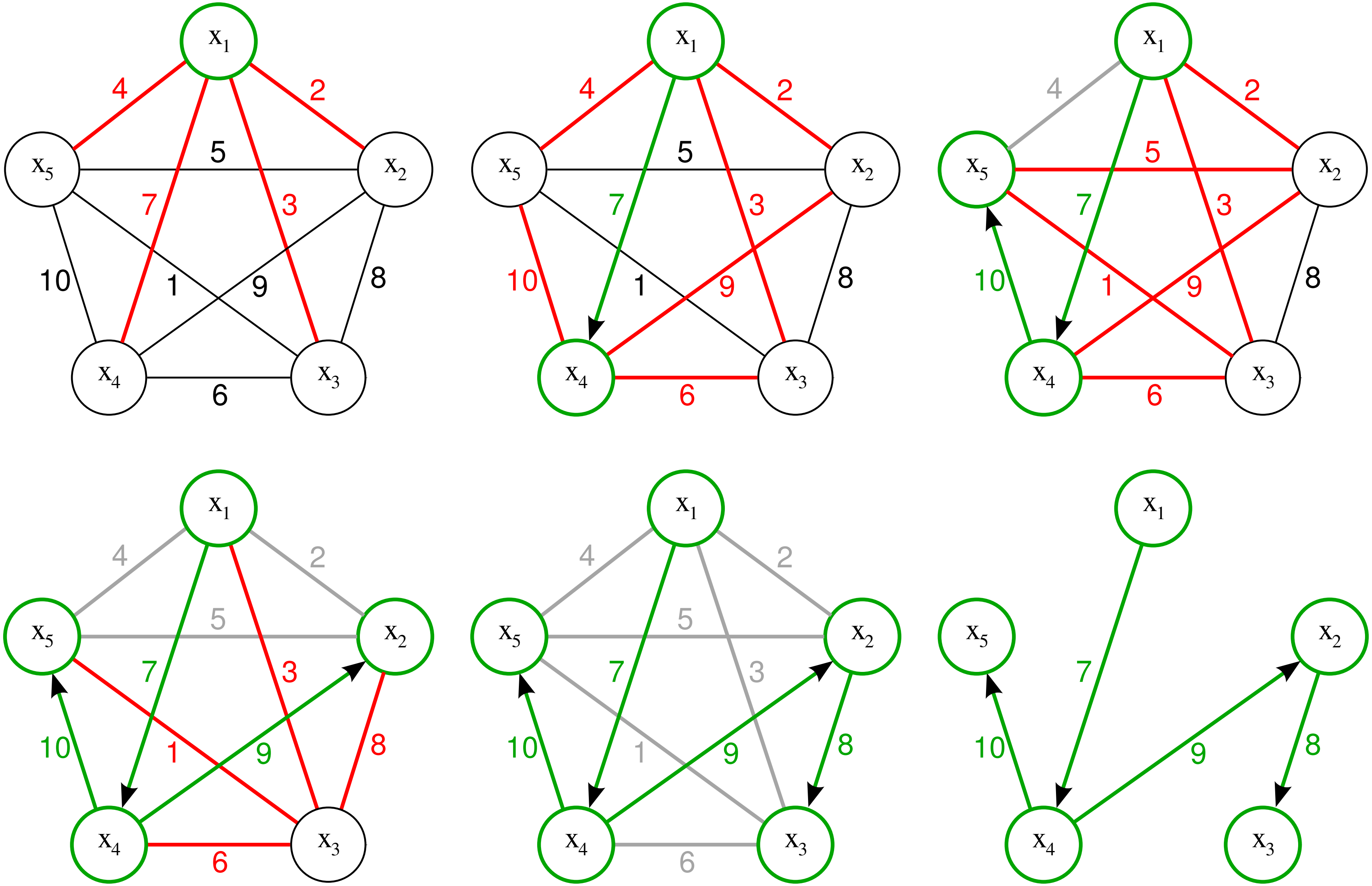




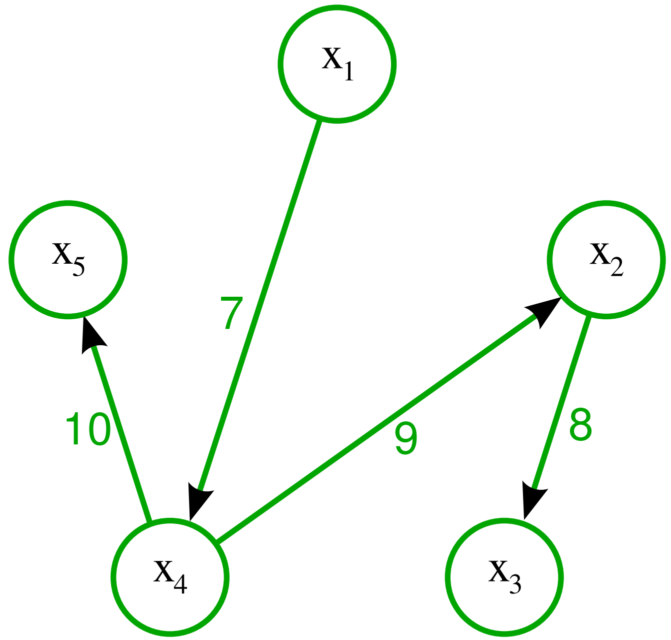
Example of dependency tree learning



Example of dependency tree learning

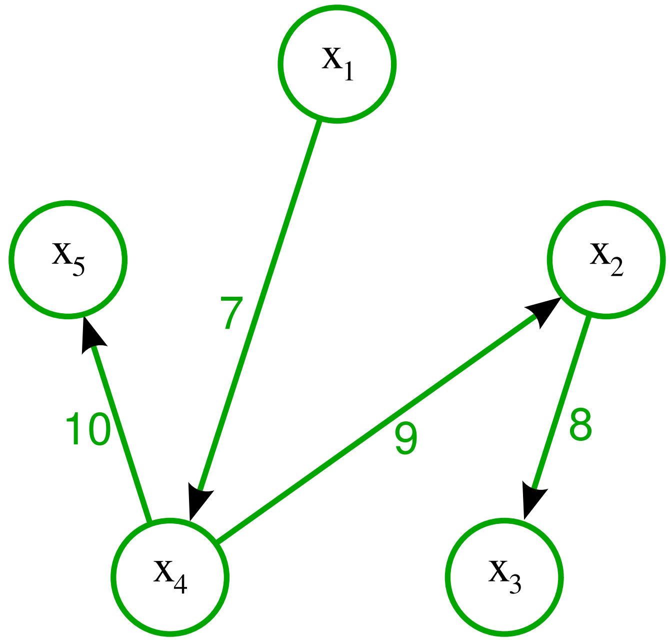


Dependency tree: probabilities



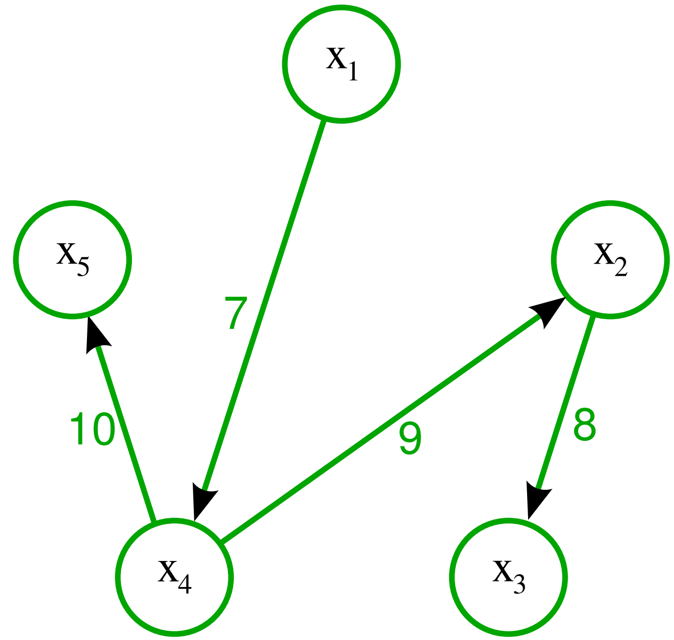
Probability	Number of params
$p(X_1 = 1)$	
$p(X_4 = 1\|X_1)$	
$p(X_5 = 1\|X_4)$	
$p(X_2 = 1\|X_4)$	
$p(X_3 = 1\|X_2)$	
Whole model	

Dependency tree: probabilities



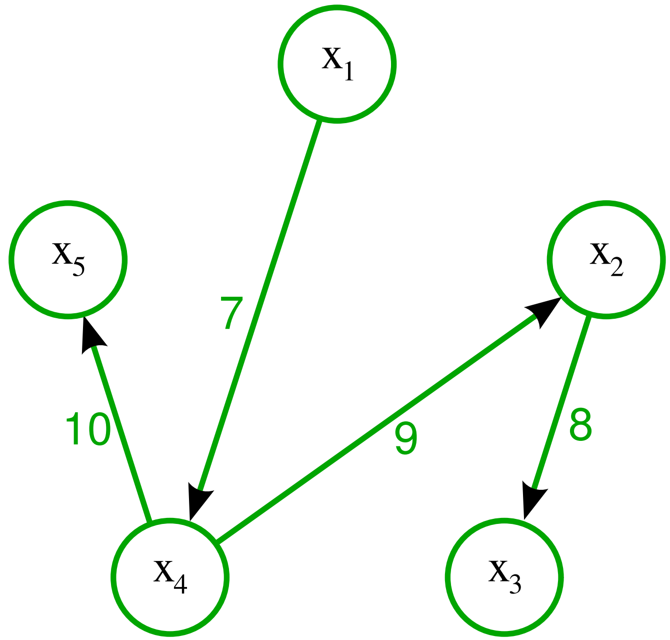
Probability	Number of params
$p(X_1 = 1)$	1
$p(X_4 = 1 \mid X_1)$	
$p(X_5 = 1 \mid X_4)$	
$p(X_2 = 1 \mid X_4)$	
$p(X_3 = 1 \mid X_2)$	
Whole model	

Dependency tree: probabilities

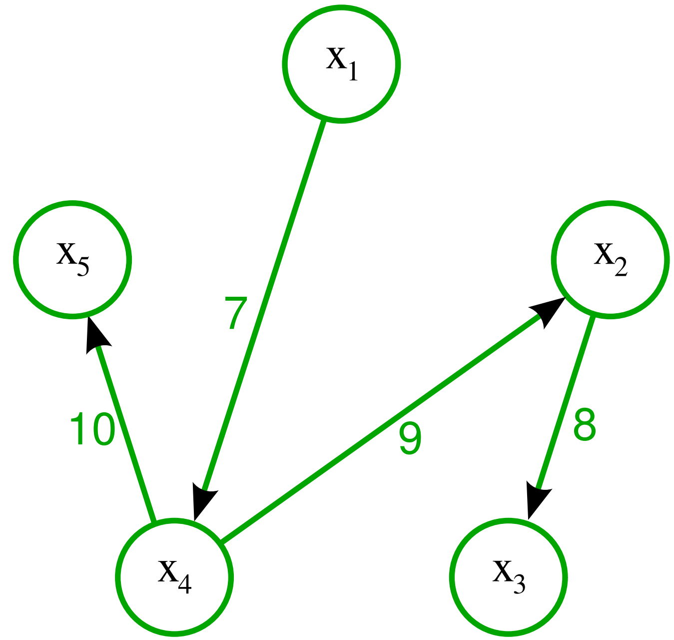


Probability	Number of params
$p(X_1 = 1)$	1
$p(X_4 = 1 \vert X_1)$	2
$p(X_5 = 1 \vert X_4)$	
$p(X_2 = 1 \vert X_4)$	
$p(X_3 = 1 \vert X_2)$	
Whole model	

Dependency tree: probabilities

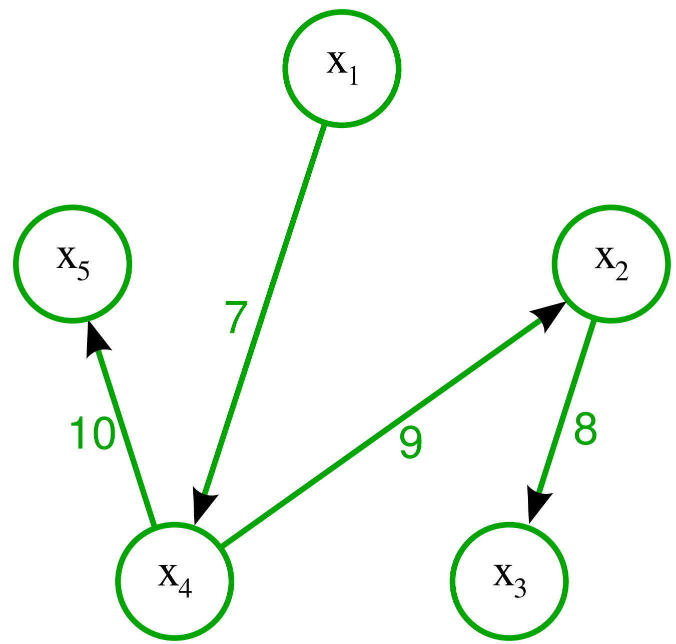


Probability	Number of params
$p(X_1 = 1)$	1
$p(X_4 = 1\|X_1)$	2
$p(X_5 = 1\|X_4)$	2
$p(X_2 = 1\|X_4)$	
$p(X_3 = 1\|X_2)$	
Whole model	



Probability	Number of params
$p(X_1 = 1)$	1
$p(X_4 = 1 \vert X_1)$	2
$p(X_5 = 1 \vert X_4)$	2
$p(X_2 = 1 \vert X_4)$	2
$p(X_3 = 1 \vert X_2)$	2
Whole model	

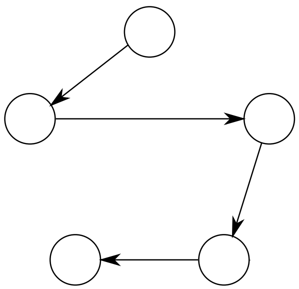
Dependency tree: probabilities



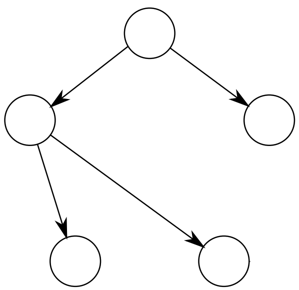
Probability	Number of params
$p(X_1 = 1)$	1
$p(X_4 = 1\|X_1)$	2
$p(X_5 = 1\|X_4)$	2
$p(X_2 = 1\|X_4)$	2
$p(X_3 = 1\|X_2)$	2
Whole model	9

EDAs with pairwise interactions

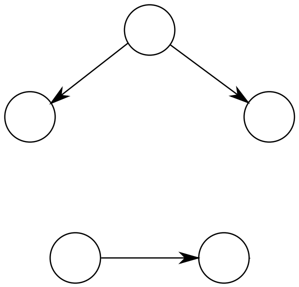
1. **MIMIC** (sequences)
   - ✔ Mutual Information Maximization for Input Clustering
   - ✔ de Bonet et al., 1996



2. **COMIT** (trees)
   - ✔ Combining Optimizers with Mutual Information Trees
   - ✔ Baluja and Davies, 1997



3. **BMDA** (forrest)
   - ✔ Bivariate Marginal Distribution Algorithm
   - ✔ Pelikan and Mühlenbein, 1998



Summary

- ✔ Advantages:
  - ✘ Still simple
  - ✘ Still fast
  - ✘ Can learn *something* about the structure
- ✔ Limitations:
  - ✘ Reliably solves only order-2 decomposable problems

EDAs with Multivariate Interactions

ECGA

Extended Compact GA, Harik, 1999

Marginal Product Model (MPM)

- ✔ Variables are treated in groups
- ✔ Variables in different groups are considered statistically independent
- ✔ Each group is modeled by its joint probability distribution
- ✔ The algorithm adaptively searches for the groups during evolution

Problem	Ideal group configuration
OneMax	[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
5bitTraps	[1 2 3 4 5] [6 7 8 9 10]

ECGA

Extended Compact GA, Harik, 1999

Marginal Product Model (MPM)

- ✔ Variables are treated in groups
- ✔ Variables in different groups are considered statistically independent
- ✔ Each group is modeled by its joint probability distribution
- ✔ The algorithm adaptively searches for the groups during evolution

Problem	Ideal group configuration
OneMax	[1] [2] [3] [4] [5] [6] [7] [8] [9] [10]
5bitTraps	[1 2 3 4 5] [6 7 8 9 10]

Learning the structure

1. Evaluation metric: Minimum Description Length (MDL)
2. Search procedure: greedy
   (a) Start with each variable belonging to its own group
   (b) Perform such a join of two groups which improves the score best
   (c) Finish if no join improves the score

ECGA: Evaluation metric

Minimum description length:
Minimize the number of bits needed to store the model and the data encoded using the model

$$DL(Model, Data) = DL_{Model} + DL_{Data}$$

Model description length:
Each group g has $|g|$ dimensions, i.e. $2^{|g|} - 1$ frequencies, each of them can take on values up to N

$$DL_{Model} = \log N \sum_{g \in G} (2^{|g|} - 1)$$

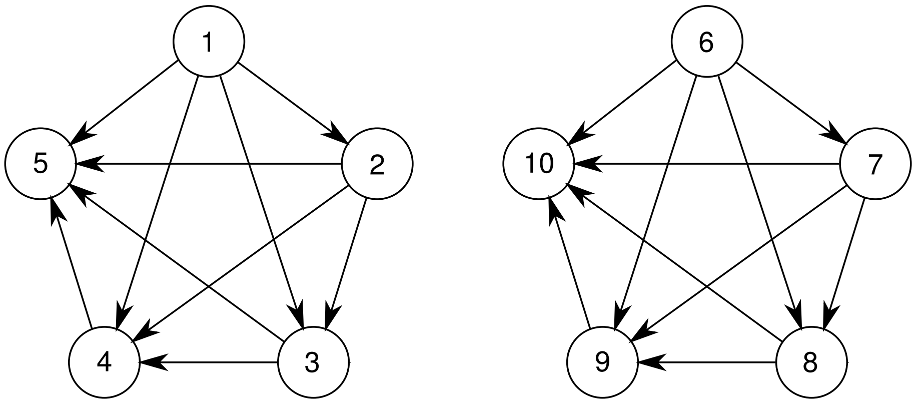
Data description length using the model:
Defined using the entropy of marginal distributions (X_g is $|g|$-dimensional random vector, x_g is its realization):

$$DL_{Data} = N \sum_{g \in G} h(X_g) = -N \sum_{g \in G} \sum_{x_g} p(X_g = x_g) \log p(X_g = x_g)$$

Bayesian Optimization Algorithm: Pelikán, Goldberg, Cantù-Paz, 1999

Bayesian network (BN)

- ✔ Conditional dependencies (instead groups)
- ✔ Sequence, tree, forrest — special cases of BN
- ✔ For trap function:



- ✔ The same model used independently in
  - ✘ Estimation of Bayesian Network Alg. (EBNA), Etxeberria et al., 1999
  - ✘ Learning Factorized Density Alg. (LFDA), Mühlenbein et al., 1999

BOA: Learning the structure

1. Evaluation metric:
   - ✔ Bayesian-Dirichlet metric, or
   - ✔ Bayesian information criterion (BIC)
2. Search procedure: greedy
   (a) Start with graph with no edges (univariate marginal product model)
   (b) Perform one of the following operations, choose the one which improves the score best
      - ✔ Add an edge
      - ✔ Delete an edge
      - ✔ Reverse an edge

   (c) Finish if no operation improves the score

BOA solves order-k decomposable problems in less then $\mathcal{O}(D^2)$ evaluations!

$$n_{evals} = \mathcal{O}(D^{1.55}) \text{ to } \mathcal{O}(D^2)$$

Scalability Analysis

Test functions

One Max:

$$f_{D\text{x1bitOneMax}}(\mathbf{x}) = \sum_{d=1}^{D} x_d$$

Trap:

$$f_{D\text{bitTrap}}(\mathbf{x}) = \begin{cases} D & \text{if } u(\mathbf{x}) = D \\ D - 1 - u(\mathbf{x}) & \text{otherwise} \end{cases}$$

Equal Pairs:

$$f_{D\text{bitEqualPairs}}(\mathbf{x}) = 1 + \sum_{d=2}^{D} f_{\text{EqualPair}}(x_{d-1}, x_d)$$

$$f_{\text{EqualPair}}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = x_2 \\ 0 & \text{if } x_1 \neq x_2 \end{cases}$$

Sliding XOR:

$$f_{D\text{bitSlidingXOR}}(\mathbf{x}) = 1 + f_{\text{AllEqual}}(\mathbf{x}) + \\ + \sum_{d=3}^{D} f_{\text{XOR}}(x_{d-2}, x_{d-1}, x_d)$$

$$f_{\text{AllEqual}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = (000\ldots0) \\ 1 & \text{if } \mathbf{x} = (111\ldots1) \\ 0 & \text{otherwise} \end{cases}$$

$$f_{\text{XOR}}(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } x_1 \bigoplus x_2 = x_3 \\ 0 & \text{otherwise} \end{cases}$$

Concatenated short basis functions:

$$f_{N\text{x}K\text{bitBasisFunction}} = \sum_{k=1}^{K} f_{\text{BasisFunction}}(x_{K(k-1)+1}, \ldots, x_{Kk})$$

Test function (cont.)

1. $f_{\text{40x1bitOneMax}}$
   - ✔ order-1 decomposable function, no interactions
2. $f_{\text{1x40bitEqualPairs}}$
   - ✔ non-decomposable function
   - ✔ weak interactions: optimal setting of each bit depends on the value of the preceding bit
3. $f_{\text{8x5bitEqualPairs}}$
   - ✔ order-5 decomposable function
4. $f_{\text{1x40bitSlidingXOR}}$
   - ✔ non-decomposable function
   - ✔ stronger interactions: optimal setting of each bit depends on the value of the 2 preceding bits
5. $f_{\text{8x5bitSlidingXOR}}$
   - ✔ order-5 decomposable function
6. $f_{\text{8x5bitTrap}}$
   - ✔ order-5 decomposable function
   - ✔ interactions in each 5-bit block are very strong, the basis function is deceptive

Scalability analysis

Facts:

- ✔ using small population size, population-based optimizers can solve only easy problems
- ✔ increasing the population size, the optimizers can solve increasingly harder problems
- ✔ ... but using a too big population is wasting of resources.

Scalability analysis

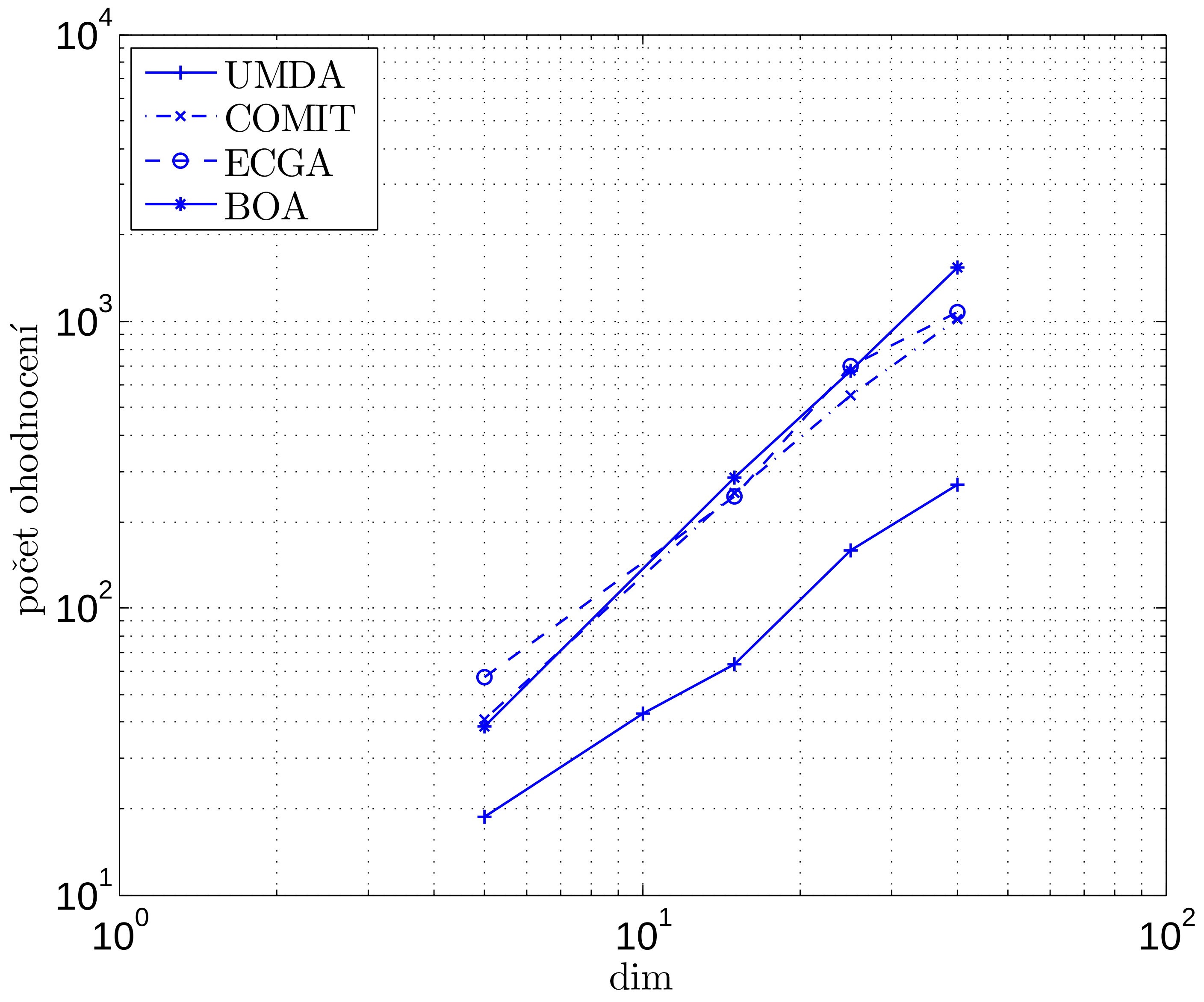
Facts:

- ✔ using small population size, population-based optimizers can solve only easy problems
- ✔ increasing the population size, the optimizers can solve increasingly harder problems
- ✔ ... but using a too big population is wasting of resources.

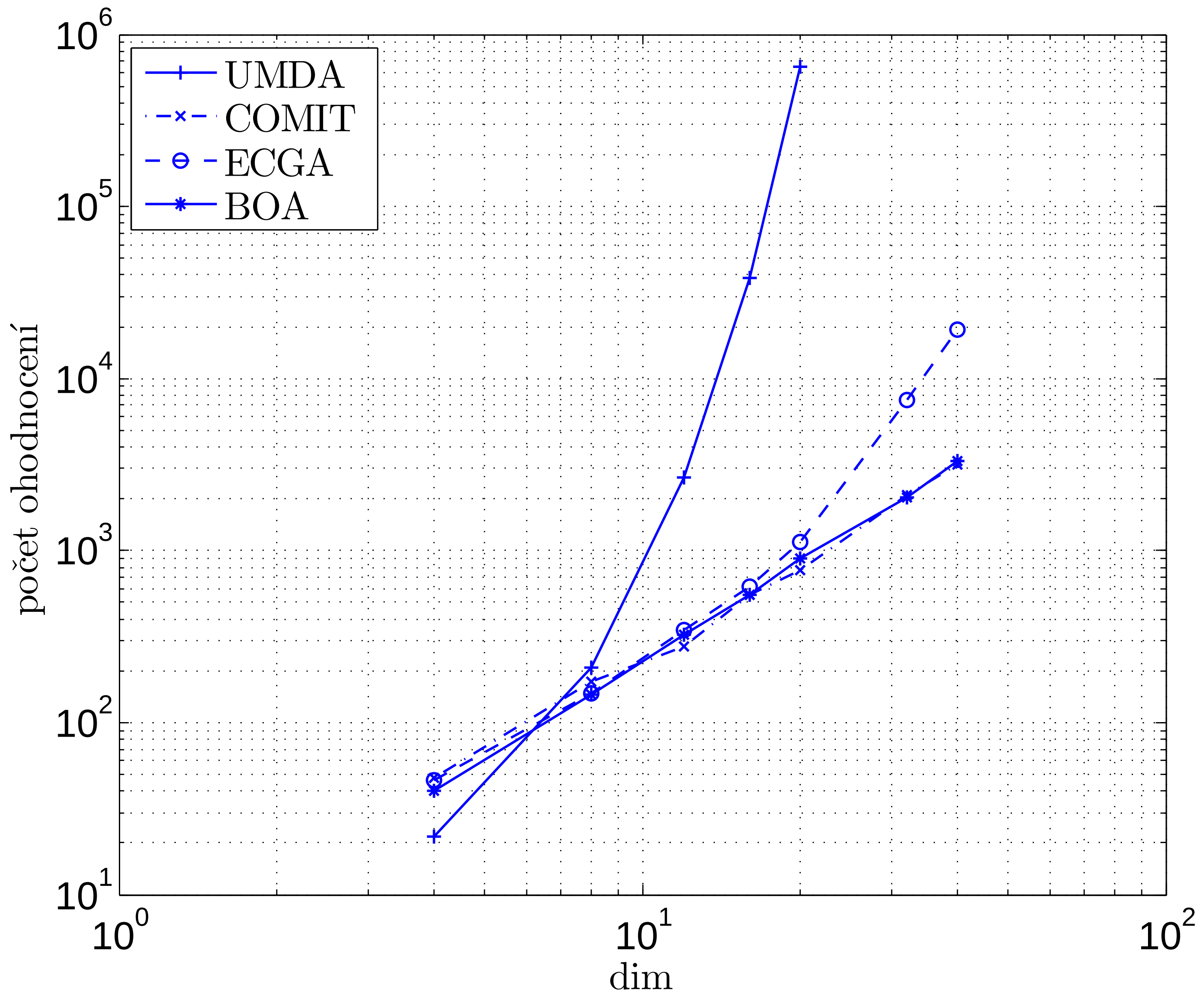
Scalability analysis:

- ✔ determines the optimal (smallest) population size, with which the algorithm solves the given problem reliably
  - ✘ reliably: algorithm finds the optimum in 24 out of 25 runs)
  - ✘ for each problem complexity, the optimal population size is determined e.g. using the bisection method
- ✔ studies the influence of the problem complexity (dimensionality) on the optimal population size and on the number of needed evaluations

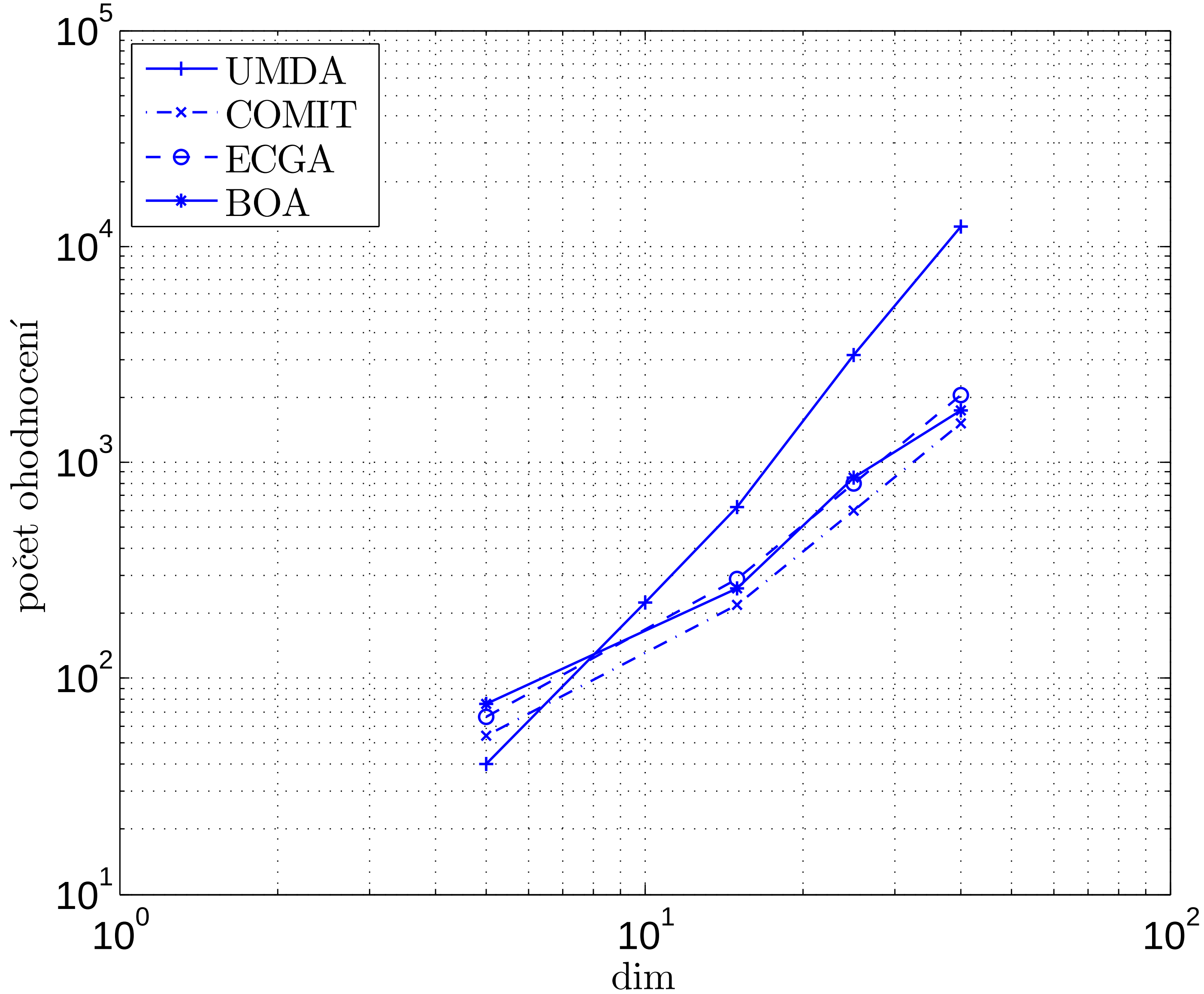
Scalability on the One Max function



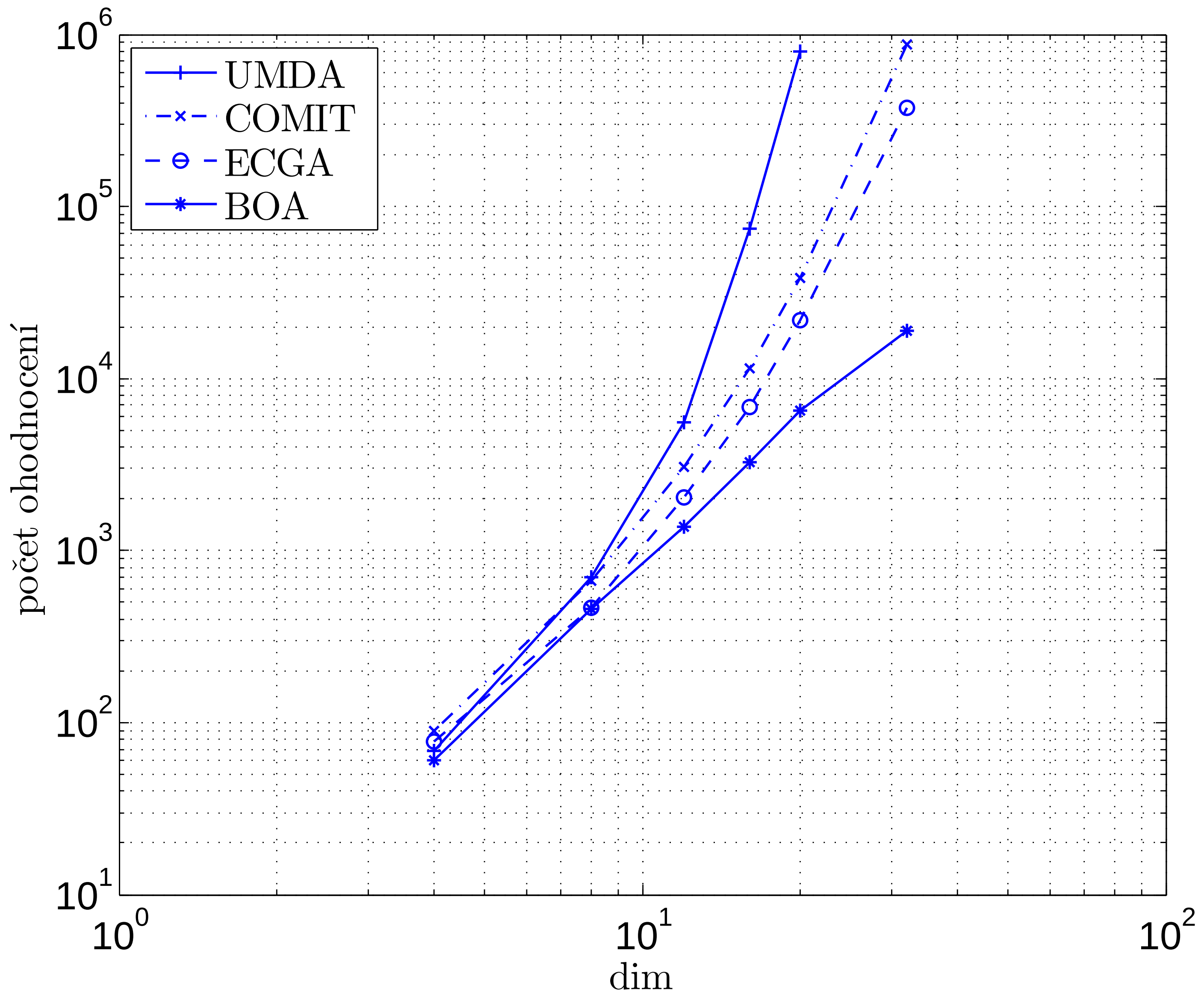
Scalability on the non-decomposable Equal Pairs function

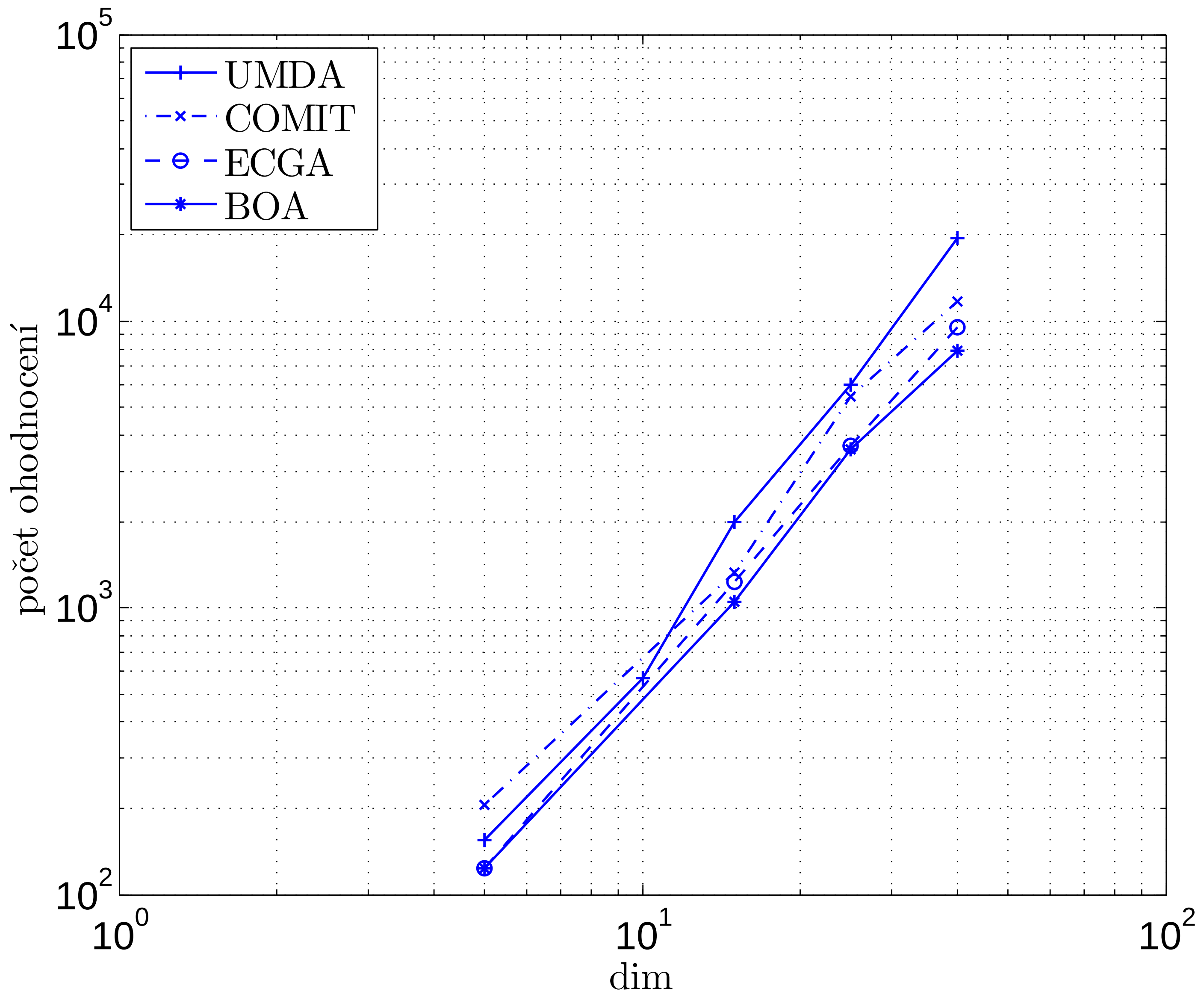


Scalability on the decomposable Equal Pairs function

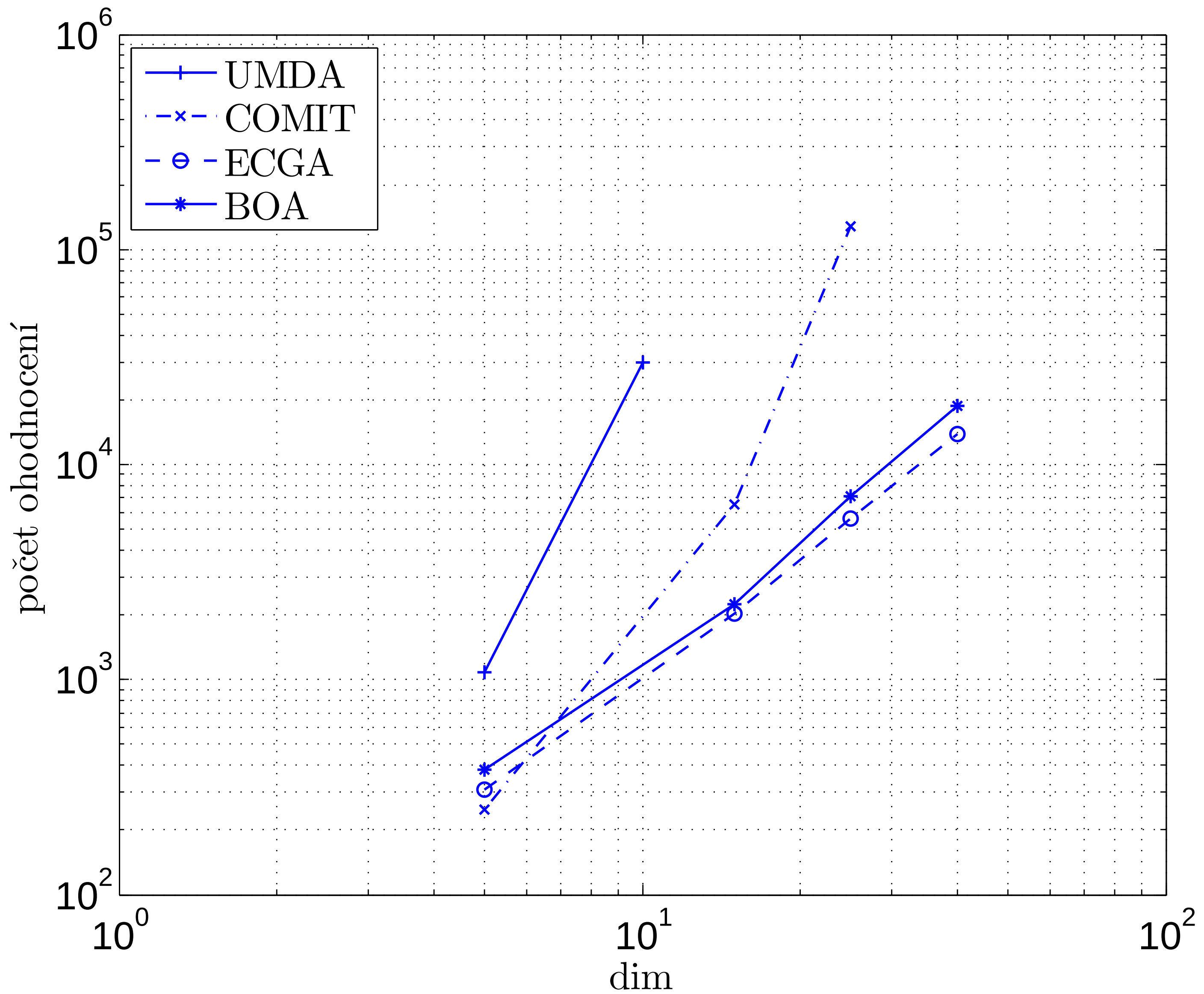


Scalability on the non-decomposable Sliding XOR function




UMDA
COMIT
ECGA
BOA
10^5
10^4
10^3
10^2
10^0
10^1
10^2
počet ohodnocení
dim

Scalability on the decomposable Trap function



Model structure during evolution

During the evolution, the model structure is increasingly precise and at the end of the evolution, the model structure describes the problem structure exactly.

Model structure during evolution

During the evolution, the model structure is increasingly precise and at the end of the evolution, the model structure describes the problem structure exactly.

NO! That's not true!

Why?

- ✔ In the beginning, the distribution patterns are not very discernible, models similar to uniform distributions are used.
- ✔ In the end, the population converges and contains many copies of the same individual (or a few individuals). No interactions among variables can be learned. Model structure is wrong (all bits independent), but the model describes the position of optimum very precisely.
- ✔ The model with the best matching structure is found somewhere in the middle of the evolution.
- ✔ Even though the right structure is never found during the evolution, the problem can be solved successfully.

Conclusions

Summary

Models:

- ✔ Bayesian networks are general models of joint probability
- ✔ High-dimensional models are hard to train
- ✔ High-dimensional models are very flexible

Advantages:

- ✔ Reliably solves problems decomposable to subproblems of bounded order

Limitations:

- ✔ Does not solve problems decomposable to logarithmic subproblems (hierarchical problems)

Suggestions for discrete EDAs

For simple problems:

- ✔ PBIL, UMDA, cGA
- ✔ they behave similarly to simple GAs

For harder problems:

- ✔ MIMIC, COMIT, BMDA
- ✔ they are able to account for bivariate dependencies

For hard problems:

- ✔ BOA, ECGA, EBNA, LFDA
- ✔ they can take into account more general dependencies, problems with hierarchichal structures

For even harder problems:

- ✔ hBOA (hierarchical BOA)