Model-Based Evolutionary Algorithms

Dirk Thierens

Universiteit Utrecht
The Netherlands
Evolutionary Algorithms

- Population-based, stochastic search algorithms
- **Exploitation**: selection
- **Exploration**: mutation & crossover

---

Model-Based Evolutionary Algorithms

- Population-based, stochastic search algorithms
- **Exploitation**: selection
- **Exploration**:
  1. Learn a model from selected solutions
  2. Generate new solutions from the model (& population)
Model-Based Evolutionary Algorithms (MBEA)
- a.k.a. Estimation of Distribution Algorithms (EDAs)
- a.k.a. Probabilistic Model-Building Genetic Algorithms
- a.k.a. Iterated Density Estimation Evolutionary Algorithms

MBEA = Evolutionary Computing + Machine Learning

Note: model not necessarily probabilistic
Why?

Goal: Black Box Optimization

- Little known about the structure of the problem
- Clean separation optimizer from problem definition
- Easy and generally applicable

Approach

- Classical EAs: need suitable representation & variation operators
- Model-Based EAs: learn structure from good solutions
Discrete Representation

- Typically binary representation
- Higher order cardinality: similar approach
Probabilistic Model-Building Genetic Algorithm

Type of Models

- **Univariate**: no statistical interaction between variables considered.
- **Bivariate**: pairwise dependencies learned.
- **Multivariate**: higher-order interactions modeled.
Univariate PMBGA

Model

* Model: probability vector \([p_1, \ldots, p_\ell]\) (\(\ell\): string length)
* \(p_i\): probability of value 1 at string position \(i\)
* \(p(X) = \prod_{i=1}^{\ell} p(x_i)\) (\(p(x_i)\): univariate marginal distribution)

- Learn model: count proportions of 1 in selected population
- Sample model: generate new solutions with specified probabilities

Diagram:

Current population: 11001, 10101, 01011, 11000

Selected population: 11001, 10101, 01011, 11000

Probability vector: 1.0 0.5 0.5 0.0 1.0

New population: 10101, 10001, 11101, 11001
Univariate PMBGA

Different Variants

- **PBIL** (Baluja; 1995)
  - Prob. vector incrementally updated over successive generations
- **UMDA** (Mühlenbein, Paass; 1996)
  - No incremental updates: example above
- **Compact GA** (Harik, Lobo, Goldberg; 1998)
  - Models steady-state GA with tournament selection
- **DEUM** (Shakya, McCall, Brown; 2004)
  - Uses Markov Random Field modeling
A hard problem for the univariate FOS

<table>
<thead>
<tr>
<th>Data</th>
<th>Marginal Product (MP) FOS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\hat{P}(X_0X_1X_2)$</td>
</tr>
<tr>
<td>000000</td>
<td>0.3</td>
</tr>
<tr>
<td>111111</td>
<td>0.0</td>
</tr>
<tr>
<td>010101</td>
<td>0.2</td>
</tr>
<tr>
<td>101010</td>
<td>0.0</td>
</tr>
<tr>
<td>000010</td>
<td>0.0</td>
</tr>
<tr>
<td>111000</td>
<td>0.0</td>
</tr>
<tr>
<td>010111</td>
<td>0.1</td>
</tr>
<tr>
<td>111000</td>
<td>0.0</td>
</tr>
<tr>
<td>000111</td>
<td>0.0</td>
</tr>
<tr>
<td>111111</td>
<td>0.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Univariate FOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\hat{P}(X_0)$</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

- What is the probability of generating 111111?
- **Univariate FOS**: $0.5 \cdot 0.6 \cdot 0.5 \cdot 0.5 \cdot 0.6 \cdot 0.5 = 0.0225$
- **MP FOS**: $0.4 \cdot 0.4 = 0.16$ (7 times larger!)
Learning problem structure on the fly

- Without a “good” decomposition of the problem, important partial solutions (building blocks) are likely to get disrupted in variation.

- Disruption leads to inefficiency.
- Can we automatically configure the model structure favorably?
- Selection increases proportion of good building blocks and thus “correlations” between variables of these building blocks.
- So, learn which variables are “correlated”.
- See the population (or selection) as a data set.
- Apply statistics / probability theory / probabilistic modeling.
Bivariate PMBGA

Model

- Need more than just probabilities of bit values
- Model pairwise interactions: conditional probabilities

- **MIMIC** (de Bonet, Isbell, Viola; 1996)
  - Dependency Chain

- **COMIT** (Baluja, Davies; 1997)
  - Dependency Tree

- **BMDA** (Pelikan, Mühlenbein; 1998)
  - Independent trees (forest)
Bivariate PMBGA

MIMIC

- Model: chain of pairwise dependencies.
- \[ p(X) = \prod_{i=1}^{\ell-1} p(x_{i+1}|x_i)p(x_1). \]
- MIMIC greedily searches for the optimal permutation of variables that minimizes Kullack-Leibler divergence.
Bivariate PMBGA

COMIT

- Optimal dependency tree instead of linear chain.
- Compute fully connected weighted graph between problem variables.
- Weights are the mutual information $I(X, Y)$ between the variables.
  \[ I(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \frac{p(x,y)}{p(x)p(y)}. \]
- COMIT computes the maximum spanning tree of the weighted graph.
Bivariate PMBGA

BMDA

- BMDA also builds tree model.
- Model not necessarily fully connected: set of trees or forest.
- Pairwise interactions measured by Pearson’s chi-square statistics.
Multivariate PMBGA

Marginal Product Model

- **Extended Compact GA (ECGA)** (Harik; 1999) was first EDA going beyond pairwise dependencies.
- Greedily searches for the Marginal Product Model that minimizes the minimum description length (MDL).
- \[ p(X) = \prod_{g=1}^{G} p(X_g) \]
- Choose the probability distribution with the lowest MDL score.
- Start from **simplest** model: the univariate factorization.
- Join two groups that result in the largest improvement in the used scoring measure.
- **Stop** when no joining of two groups improves the score further.
Multivariate PMBGA

Minimum Description Length (MDL)

- **MDL** is a measure of **complexity** (Information Theory).
- \( \text{MDL}(M, D) = D_{\text{Model}} + D_{\text{Data}} \)
  1. Model complexity \( D_{\text{Model}} \): complexity of describing the model.
  2. Compressed population complexity \( D_{\text{Data}} \): complexity of describing the data within the model (= measure of goodness of the probability distribution estimation).

- **Best** model = the one with the **lowest** MDL score.
Minimum Description Length score

**MDL**

- Model Complexity: $D_{Model} = \log_2(N + 1) \sum_i (2^{S_i} - 1)$
- Compressed Population Complexity: $D_{Data} = N \sum_i H(M_i)$
- Combined Complexity = Model Complexity + Compressed Population Complexity

$N$: Population size  
$S_i$: size of partition $i$  
$M_i$: marginal distribution of the partition $i$

**Entropy $H(X)$** is a measure of uncertainty or unpredictability of a random variable $X$ with probability distribution $P(X)$:

$$H(X) = -\sum_k P(x_k) \log_2(P(x_k))$$
## Learning MP model

1. Start from univariate FOS:
   \[
   \{\{0\}, \{1\}, \{2\}, \ldots, \{l - 2\}, \{l - 1\}\}
   \]

2. All possible **pairs** of partitions are temporarily merged:
   \[
   \begin{align*}
   &\{\{0, 1\}, \{2\}, \ldots, \{l - 2\}, \{l - 1\}\} \\
   &\{\{0, 2\}, \{1\}, \ldots, \{l - 2\}, \{l - 1\}\} \\
   &\vdots \\
   &\{\{0\}, \{1, 2\}, \ldots, \{l - 2\}, \{l - 1\}\} \\
   &\vdots \\
   &\{\{0\}, \{1\}, \{2\}, \ldots, \{l - 2, l - 1\}\}
   \end{align*}
   \]

3. Compute **MDL** score of each factorization.

4. Choose the **best** scoring factorization if **better** than current.

5. Repeat until no better scoring factorization is found.
population size $N = 8$, string length $l = 4$
- Marginal Product Model: $[I_1], [I_2], [I_3], [I_4]$

\[
\begin{array}{cccc}
1 \ 5/8 & 1 \ 4/8 & 1 \ 3/8 & 1 \ 4/8 \\
0 \ 3/8 & 0 \ 4/8 & 0 \ 5/8 & 0 \ 4/8 \\
\end{array}
\]

- Marginal Product Model: $[I_1, I_3], [I_2], [I_4]$

\[
\begin{array}{ccc}
[I_1, I_3] & [I_2] & [I_4] \\
11 \ 0/8 & 1 \ 4/8 & 1 \ 4/8 \\
10 \ 5/8 & 0 \ 4/8 & 0 \ 4/8 \\
01 \ 3/8 & 0 \ 4/8 & 0 \ 4/8 \\
00 \ 0/8 & 0 \ 4/8 & 0 \ 4/8 \\
\end{array}
\]
Entropy calculations:

1. Marginal Product Model: $[I_1], [I_2], [I_3], [I_4]$

   \[ \text{Entropy}([I_1]) = -(5/8)\log_2(5/8) - (3/8)\log_2(3/8) = 0.954 \]

   \[ \text{Entropy}([I_2]) = -(4/8)\log_2(4/8) - (4/8)\log_2(4/8) = 1 \]

   \[ \text{Entropy}([I_3]) = -(3/8)\log_2(3/8) - (5/8)\log_2(5/8) = 0.954 \]

   \[ \text{Entropy}([I_4]) = -(4/8)\log_2(4/8) - (4/8)\log_2(4/8) = 1 \]

2. Marginal Product Model: $[I_1, I_3], [I_2], [I_4]$

   \[ \text{Entropy}([I_1, I_3]) = -(5/8)\log_2(5/8) - (3/8)\log_2(3/8) = 0.954 \]

   \[ \text{Entropy}([I_2]) = -(4/8)\log_2(4/8) - (4/8)\log_2(4/8) = 1 \]

   \[ \text{Entropy}([I_4]) = -(4/8)\log_2(4/8) - (4/8)\log_2(4/8) = 1 \]
Marginal Product Model: \([I_1], [I_2], [I_3], [I_4]\)
Model Complexity = \(\log_2(9)(1 + 1 + 1 + 1) = 12.7\)
Compressed Population Complexity = \(8 (0.945 + 1 + 0.954 + 1) = 31.3\)
Combined Complexity = \(12.7 + 31.3 = 44\)

Marginal Product Model: \([I_1, I_3], [I_2], [I_4]\)
Model Complexity = \(\log_2(9)(3 + 1 + 1) = 15.8\)
Compressed Population Complexity = \(8 (0.945 + 1 + 1) = 23.6\)
Combined Complexity = \(15.8 + 23.6 = 39.4\)
<table>
<thead>
<tr>
<th>MPM</th>
<th>Combined Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[I_1, I_2][I_3][I_4]$</td>
<td>46.7</td>
</tr>
<tr>
<td>$[I_1, I_3][I_2][I_4]$</td>
<td>39.8</td>
</tr>
<tr>
<td>$[I_1, I_4][I_2][I_3]$</td>
<td>46.7</td>
</tr>
<tr>
<td>$[I_1][I_2, I_3][I_4]$</td>
<td>46.7</td>
</tr>
<tr>
<td>$[I_1][I_2, I_4][I_3]$</td>
<td>45.6</td>
</tr>
<tr>
<td>$[I_1][I_2][I_3, I_4]$</td>
<td>46.7</td>
</tr>
</tbody>
</table>
The Marginal Product Model: \([I_1, I_3], [I_2], [I_4]\) has the lowest combined complexity so it is the best model to compress the population and therefore captures the most dependencies in the set of solutions.
Example: Deceptive Trap Function

Building block length $k = 4$; Number of building blocks $m = 10$.

<table>
<thead>
<tr>
<th>Population size</th>
<th>subfunctions solved</th>
<th>function evals.</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3.9</td>
<td>740</td>
</tr>
<tr>
<td>500</td>
<td>5.2</td>
<td>5100</td>
</tr>
<tr>
<td>1000</td>
<td>6.1</td>
<td>15600</td>
</tr>
<tr>
<td>5000</td>
<td>6.8</td>
<td>100000</td>
</tr>
<tr>
<td>10000</td>
<td>7.3</td>
<td>248000</td>
</tr>
<tr>
<td>20000</td>
<td>8.0</td>
<td>614000</td>
</tr>
<tr>
<td>50000</td>
<td>7.9</td>
<td>1560000</td>
</tr>
<tr>
<td>100000</td>
<td>8.8</td>
<td>3790000</td>
</tr>
</tbody>
</table>
### Example: Deceptive Trap Function

#### Extended Compact GA:

<table>
<thead>
<tr>
<th>Population size</th>
<th>Subfunctions solved</th>
<th>Function evals.</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4.0</td>
<td>750</td>
</tr>
<tr>
<td>200</td>
<td>5.2</td>
<td>1460</td>
</tr>
<tr>
<td>300</td>
<td>7.1</td>
<td>2610</td>
</tr>
<tr>
<td>500</td>
<td>9.3</td>
<td>4000</td>
</tr>
<tr>
<td>600</td>
<td>9.9</td>
<td>5040</td>
</tr>
<tr>
<td>1000</td>
<td>10.0</td>
<td>7300</td>
</tr>
</tbody>
</table>
Conclusion

Simple Genetic Algorithms are limited in their capability to mix or recombine non-linked building blocks

1. Design linkage into problem representation and recombination operator

or

2. Learn linkage by using probabilistic model building genetic algorithm
Multivariate PMBGA

Bayesian Network

- Probability vector, dependency tree, and marginal product model are **limited** probability models.
- Bayesian network much more **powerful** model.
  - Acyclic directed graph.
  - Nodes are problem variables.
  - Edges represent conditional dependencies.
Multivariate PMBGA

Current population → Selected population → Bayesian network → New population
Multivariate PMBGA

Bayesian network learning

- Similar to ECGA: scoring metric + greedy search
- **Scoring metric**: MDL or Bayesian measure
- **Greedy search**:
  - Initially, no variables are connected.
  - Greedily either add, remove, or reverse an edge between two variables.
  - Until local optimum is reached.
Multivariate PMBGA

Bayesian Network PMBGAs variants

- Bayesian Optimization Algorithm (BOA) (Pelikan, Goldberg, Cantú-Paz; 1998)
- Estimation of Distribution Networks Algorithm (EBNA) (Etxeberria, Larrañaga; 1999)
- Learning Factorized Distribution Algorithm (LFDA) (Mühlenbein, Mahnig, Rodriguez; 1999)

Similarities: All use Bayesian Network as probability model.
Dissimilarities: All use different method to learn BN.
Hierarchical BOA

- **hBOA** (Pelikan, Goldberg; 2001)
- **Decomposition** on multiple levels.
  - Bayesian network learning by BOA
- **Compact** representation.
  - Local Structures to represent conditional probabilities.
- **Preservation** of alternative solutions.
  - Niching with Restricted Tournament Replacement
Multivariate PMBGA

Markov Network

- **Markov Network EDA**
- Probability model is **undirected graph**.
- **Factorise** the joint probability distribution in cliques of the undirected graph and sample it.
- Most recent version: **Markovian Optimisation Algorithm** (MOA) (Shakya & Santana, 2008).
- MOA does not explicitly factorise the distribution but uses the local Markov property and Gibbs sampling to generate new solutions.
Family Of Subsets (FOS) model

FOS $\mathcal{F}$

- PMBGAs learn a **probabilistic model** of good solutions to match the **structure** of the optimization problem.
- Key idea is to identify **groups of problem variables** that together make an important contribution to the quality of solutions.
- Dependency structure generally called a **Family Of Subsets** (FOS).
- Let there be $\ell$ **problem variables** $x_0, x_1, \ldots, x_{\ell-1}$.
- Let $S$ be a set of all variable **indices** $\{0, 1, \ldots, \ell - 1\}$.
- A FOS $\mathcal{F}$ is a **set of subsets** of the set $S$.
- FOS $\mathcal{F}$ is a **subset** of the **powerset** of $S$ ($\mathcal{F} \subseteq \mathcal{P}(S)$).
Family Of Subsets (FOS) model

- FOS can be written more specifically as:

\[ \mathcal{F} = \{ F^0, F^1, \ldots, F^{|\mathcal{F}|−1} \} \]

where

\[ F^i \subseteq \{0, 1, \ldots, l − 1\}, \quad i \in \{0, 1, \ldots, |\mathcal{F}| − 1\} \]

- Every variable is in at least one subset in the FOS, i.e.:

\[ \forall i \in \{0, 1, \ldots, l − 1\} : \left( \exists j \in \{0, 1, \ldots, |\mathcal{F}| − 1\} : i \in F^j \right) \]
The Univariate Structure

- The univariate FOS is defined by:
  \[ F^i = \{i\}, \quad i \in \{0, 1, \ldots, l - 1\} \]

- For \( l = 10 \) the univariate FOS is:
  \[ \mathcal{F} = \{\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\} \]

- Every variable is modeled to be independent of other variables.
The Marginal Product Structure

- The marginal product (MP) FOS is a FOS such that:
  \[ F^i \cap F^j = \emptyset, \quad i, j \in \{0, 1, \ldots, l - 1\} \]

- Univariate FOS is a MP FOS.
- For \( l = 10 \) a possible MP FOS is:
  \[ \mathcal{F} = \{\{0, 1, 2\}, \{3\}, \{4, 5\}, \{6, 7, 8, 9\}\} \]

- Every group of variables is modeled to be independent of other variables.
The Linkage Tree Structure

- The linkage tree (LT) FOS is a hierarchical structure.
- Group of all variables is in there.
- For any subset $F^i$ with more than one variable, there are subsets $F^j$ and $F^k$ such that:
  $$F^j \cap F^k = \emptyset, \quad |F^j| < |F^i|, \quad |F^k| < |F^i| \quad \text{and} \quad F^j \cup F^k = F^i$$
- For $l = 10$ a possible LT FOS is
  $$\mathcal{F} = \{\{7, 5, 8, 6, 9, 0, 3, 2, 4, 1\}, \{7, 5, 8, 6, 9\}, \{0, 3, 2, 4, 1\}, \{7\}, \{5, 8, 6, 9\}, \{0, 3\}, \{2, 4\}, \{5, 8\}, \{6\}, \{0\}, \{3\}, \{2\}, \{4\}, \{5\}, \{8\}\}$$
- Variables sometimes independent, sometimes dependent.
- $\approx$ Path through dependency space, from univariate to joint.
Linkage Tree

- **Linkage Tree** structure: subsets of FOS $F$ form a hierarchical clustering.

- $F = \{\{0,1,2,3,4,5,6,7,8,9\}, \{0,1,2,3,4,5\}, \{6,7,8,9\}, \{0,1,2\}, \{3,4,5\}, \{7,8,9\}, \{0,1\}, \{4,5\}, \{8,9\}, \{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}\}$

- Each subset (of length $> 1$) is split in two **mutually exclusive** subsets.

- Problem variables in subset are considered to be **dependent** on each other but become **independent** in a child subset.

- For a problem of length $\ell$ the linkage tree has $\ell$ **leaf** nodes (the clusters having a single problem variable) and $\ell – 1$ **internal** nodes.
Linkage Tree Learning

- Start from univariate structure.
- Build linkage tree using bottom-up hierarchical clustering algorithm.
- Similarity measure:
  1. Between individual variables $X$ and $Y$: mutual information $I(X, Y)$.
     \[ I(X, Y) = H(X) + H(Y) - H(X, Y) \]
  2. Between cluster groups $X_{Fi}$ and $X_{Fj}$: average pairwise linkage clustering (= unweighted pair group method with a arithmetic mean: UPGMA).
     \[ I_{UPGMA}(X_{Fi}, X_{Fj}) = \frac{1}{|X_{Fi}||X_{Fj}|} \sum_{X \in X_{Fi}} \sum_{Y \in X_{Fj}} I(X, Y). \]

($H(X), H(Y), H(X, Y)$ are the marginal and joint entropies)
Linkage Tree Learning

- This agglomerative hierarchical clustering algorithm is computationally efficient.
- Only the mutual information between pairs of variables needs to be computed once, which is a $O(\ell^2)$ operation.
- The bottom-up hierarchical clustering can also be done in $O(\ell^2)$ computation by using the reciprocal nearest neighbor chain algorithm.
Optimal Mixing Evolutionary Algorithms (OMEA)

- **OMEA** is a Model-Building EA that uses a **FOS** as its linkage model (Thierens & Bosman, 2011).
- Characteristic of **Optimal Mixing Evolutionary Algorithm (OMEA)** is the use of **intermediate** function evaluations (inside variation)
- Can be regarded as **greedy improvement** of existing solutions
- Coined “**Optimal**” Mixing because **better** instances for substructures are **immediately accepted** and not dependent on “**noise**” coming from other parts of the solution
- **Recombinative OM (ROM) and Gene-pool OM (GOM)**
  - ROM is GA-like: select single solution to perform OM with
  - GOM is EDA-like: select new solution for each substructure in OM
Optimal Mixing EA (GOMEA)

- **FOS** linkage models specify the linked variables.
- A subset of the FOS is used as crossover mask
- Crossover is **greedy**: only improvements (or equal) are accepted.
- Each generation a new FOS model is build from selected solutions.
- For each solution in the population, all subsets of the FOS are tried with a donor solution randomly picked from the population
- **Recombinative OM (ROM)** and **Gene-pool OM (GOM)**
  - **ROMEA**: each solution uses a single donor solution.
  - **GOMEA**: new donor selected for each FOS subset.
Gene-pool Optimal Mixing EA

GOMEA()

Pop ← InitPopulation()
while NotTerminated(Pop)
    FOS ← BuildFOS(Pop)
    forall Sol ∈ Pop
        forall SubSet ∈ FOS
            Donor ← Random(Pop)
            Sol ← GreedyRecomb(Sol, Donor, SubSet, Pop)
    return Sol

GreedyRecomb(Sol, Donor, SubSet, Pop)

NewSol ← ReplaceSubSetValues(Sol, SubSet, Donor)
    if ImprovementOrEqual(NewSol, Sol)
        then Sol ← NewSol
    return Sol
Recombinative Optimal Mixing EA

ROMEAA()

Pop ← InitPopulation()

while NotTerminated(Pop)

FOS ← BuildFOS(Pop)

forall Sol ∈ Pop

Donor ← Random(Pop)

forall SubSet ∈ FOS

Sol ← GreedyRecomb(Sol, Donor, SubSet, Pop)

return Sol

GreedyRecomb(Sol, Donor, SubSet, Pop)

NewSol ← ReplaceSubSetValues(Sol, SubSet, Donor)

if ImprovementOrEqual(NewSol, Sol)

then Sol ← NewSol

return Sol
The LTGA is an instance of GOMEA that uses a Linkage Tree as FOS model (Thierens & Bosman, 2010, 2011).

- Each generation a new hierarchical cluster tree is build.
- For each solution in population, traverse tree starting at the top.
- Nodes (= clusters) in the linkage tree used as crossover masks.
- Select random donor solution, and its values at the crossover mask replace the variable values from the current solution.
- Evaluate new solution and accept if better/equal, otherwise reject.
Deceptive Trap Function

Interacting, non-overlapping, deceptive groups of variables.

\[ f_{DT}(x) = \sum_{i=0}^{l-k} f_{DT}^{\text{sub}}(x_{(i,...,i+k-1)}) \]
Nearest-neighbor NK-landscape

- **Overlapping**, neighboring random subfunctions

\[
f_{\text{NK-S1}}(x) = \sum_{i=0}^{l-k} f_{\text{NK}}^{\text{sub}}(x(i,\ldots,i+k-1)) \quad \text{with} \quad f_{\text{NK}}^{\text{sub}}(x(i,\ldots,i+k-1)) \in [0..1]
\]

- eg. 16 subsfcts, length \( k = 5 \), overlap \( o = 4 \) ⇒ stringlength \( \ell = 20 \)

- Global optimum computed by dynamic programming
- Benchmark function: **structural information is not known**!
- ⇒ Randomly shuffled variable indices.
Experiments

- Benchmark functions: randomly linked deceptive trap function and randomly linked nearest-neighbor NK function.
- Compare GA, EDA, and GOMEA learning the Marginal Product (MP) FOS structure, and GOMEA learning the Linkage Tree (LT) as FOS structure.
- Note:
  - EDA using MP = Extended Compact GA (ECGA).
  - GOMEA using LT = Linkage Tree Genetic Algorithm (LTGA).
# Function Evaluations / Problem size

![Graphs showing function evaluations vs problem size for different algorithms and fitness landscapes.](image-url)
Experiments

Minimal Population Size / Problem Size

![Graphs showing relationship between population size and problem size for experiments.](image)
Experiments

Runtime (seconds) / Problem Size

![Graphs showing runtime vs problem size for different algorithms.](image-url)
Experiments

Figure: LTGA vs. ILS: 100 NK problems

ILS perturbation size each time randomly picked between 2 and 10 bits (= better than any fixed value).
Experiments: conclusion

- LTGA (= GOMEA with LT FOS) very efficient on Deceptive Trap function, Nearest-Neighbor NK landscape, and Hierarchical Trap function.

- Other FOS models possible: Linkage Neighborhood OM (Bosman & Thierens, 2012).

- **Linkage Tree** seems to be good compromise between FOS model complexity and search efficiency.
Predetermined vs. Learned FOS

- Problem structure unknown: learn a FOS model.
- Problem structure Information available: predetermined FOS model.
- What is a good predetermined FOS model?
- Direct mapping of dependency structure of problem definition to a predetermined FOS model?
- Predetermined linkage models mirroring the static structure of the problem not sufficient (Thierens & Bosman, 2012).
- Dynamically learned tree model superior to mirror structured models and to static tree model.
- Question: is there an optimal, predetermined linkage model that outperforms the learned (tree) model?
Conclusions

- “Blind” Evolutionary Algorithms are limited in their capability to detect and mix/exploit/re-use partial solutions (building blocks).
- One requires knowledge, luck or analysis and design of structure exploitation directly into problem representation and search operators.
- Having a configurable model can help overcome this.
- Algorithm then must learn a model of dependencies and exploit structure online during optimization (e.g. EDAs, OMEAs).