Parameter-less Local Optimizer with Linkage Identification for Deterministic Order-k Decomposable Problems

Petr Pošík
Czech Technical University in Prague
FEE, Dept. of Cybernetics
Technická 2, 16627 Prague 6, Czech Republic
petr.posik@fel.cvut.cz

Stanislav Vaníček
Czech Technical University in Prague
FEE, Dept. of Cybernetics
Technická 2, 16627 Prague 6, Czech Republic
vanicsta@fel.cvut.cz

ABSTRACT

A simple parameter-less local optimizer able to solve deterministic problems with building blocks of bounded order is proposed in this article. The algorithm is able to learn and use linkage information during the run. The algorithm is algorithmically simple, easy to implement and with the exception of termination condition, it is completely parameter-free—there is thus no need to tune the population size and other parameters to the problem at hand. An empirical comparison on 3 decomposable functions, each with uniformly scaled building blocks of size 5 and 8, was carried out. The algorithm exhibits quadratic scaling with the problem dimensionality, but the comparison with the extended compact genetic algorithm and Bayesian optimization algorithm shows that it needs lower or comparable number of fitness function evaluations on the majority of functions for the tested problem dimensionalities. The results also suggest that the efficiency of the local optimizer compared to both the estimation-of-distribution algorithms should be better for problem sizes under at least a few hundreds of bits.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

Local optimization, linkage learning, evolutionary algorithms, estimation-of-distribution algorithms

1. INTRODUCTION AND MOTIVATION

One of the greatest and most studied issues in black-box optimization is the *epistasis* phenomenon. The epistasis is usually defined as a situation when the effect of one solution component (its contribution to fitness function) depends on the value of another solution component. Such a phenomenon can pose huge problems to black-box optimization techiques like various forms of hill-climbers (HC) or genetic algorithms (GAs). An epistatic relation is also called *linkage*, *dependency*, or *interaction*. Despite the fact that in other scientific disciplines these terms often have similar, but not exactly the same meaning, in the evolutionary computation community these terms are usually used interchangeably.

The linkage information (which solution components are related to other ones) is an important knowledge about the function being optimized—in certain cases it enables us to decompose the problem, to perform the search more efficiently. However, in the black-box setting, the linkage information is usually not available. Several ways of addressing the problem *how to learn the linkage during the optimization* have been proposed. In this paper we concentrate mainly on the two most prominent of them: perturbation techniques and probabilistic modeling.

Yet, virtually all the existing algorithms able to learn linkage during the optimization process, no matter if they use perturbation techniques or probabilistic modeling, use a population of some sort. The population size plays a crucial role in all of them [6]. It determines if the right linkages can be discovered. The algorithms usually need certain minimal population size to have enough data to be able to find the correct linkages (or at least sufficient amount of linkages that would enable the algorithm to find the global optimum). On the other hand, too large population size may cause discovering of false linkages [21] which may deteriorate the algorithm performance. The population size must be tuned to the problem to get the best results that the algorithm can provide. This is a tedious and often a time-consuming work.

Local optimizers (LO) or hill-climbers (HC) do not use the population of solutions, they use just one solution which they iteratively improve. If they are equipped with the linkage information, they can solve also certain types of epistatic problems. The perturbation methods are directly applicable in LOs, yet, to the best of the authors' knowledge, such a simple and straightforward approach was never tested. The simple idea of alternating the linkage identification phase with the local optimizer able to take advantage of the learned linkage information was used in a bachelor thesis [25] and is further elaborated and explored in this initial study.

The first goal is to propose a parameter-free local search algorithm with binary representation able to learn the link-

GECCO'11, July 12–16, 2011, Dublin, Ireland.

age information for decomposable deterministic fitness functions with building blocks of bounded order. The second goal is to compare the proposed algorithm with representatives of EDAs with the aim to show if the parameter-free LO can compete with the tuned EDAs at least for certain problems and problem sizes.

First, in Sec. 2, we review some works relevant to the topic of this paper and describe them in more detail; the information further motivates the presented work. In Sec. 3, the proposed local optimizer with linkage identification is introduced and described. Sections 4 and 5 present the test problems and the setup of experiments, respectively. The results are presented and discussed in Sec. 6 and Sec. 7 summarizes the main contributions of the work.

2. RELATED WORK

The *perturbation techniques* of linkage identification analyze the fitness function values directly, i.e. they analyze the changes of fitness values as a response to the known perturbation of several solution components. They are usually used as part of the ordinary GAs able to exploit the learned knowledge via modified crossover and mutation operators. In the context of GAs (for binary representation), the approach was pioneered in the gene expression messy GA [10]. The idea was later made more explicit in linkage identification by non-linearity check (LINC) [16], linkage identification by non-monotonicity detection (LIMD) [17] and generalized for functions with overlapping building blocks by linkage identification with epistasis measures (LIEM [14], LIEM^2 [15]). The most general and formal description of these methods can be found in [8]. The perturbation used in the method can take into account not only pairs of bits, but generally any k-tuple; LINC is thus a special case of the method for $k = 2$. The LINC and LIMD methods are described in more detail in Sec. 3.2 since they are used as the linkage identification part of the proposed algorithm.

The *probabilistic modeling techniques* are used in the so called estimation-of-distribution algorithms (EDAs) [13, 11]. EDAs are a special type of evolutionary algorithms which use probabilistic model building and sampling instead of the usual crossover and mutation operators in the generation cycle. The model is built with the aim to describe the distribution of high-quality solutions in the search space. If the model used in the algorithm is flexible enough, it can describe and use statistical features of the data set it was trained on. In other words, it can learn e.g. which allelles are often observed together in the high-quality solutions. The first EDAs (e.g. PBIL [1], UMDA [13]) were not able to discover and use dependencies. The second generation EDAs were able to work with pairwise dependencies only (BMDA [20], MIMIC [3], COMIT [2]). The most advanced EDAs allow for multivariate dependencies (i.e. a locus may be related to more than one other loci). This class of EDAs is most importantly represented with the extended compact genetic algorithm (ECGA, [7]), learning factorized distribution algorithm (LFDA, [12]), Estimation-of-Bayesian-Network Algorithm (EBNA, [5]) and the Bayesian optimization algorithm (BOA, [19]). ECGA and BOA are described in greater detail in Sec. 3.3 since they serve as the baseline reference algorithms in this study.

The topic of this paper is also related to the discussion whether a GA would profit more from the use of crossover or mutation operator if they were able to use the linkage information. Since we study a local optimizer which uses similar principles as the mutation operator (it creates candidate solution as a perturbation of 1 particular solution), the situations where mutation works better may be also suitable areas for application of the proposed LO. On the other hand, since the probabilistic modeling in EDAs is a generalization of the multi-parent crossover operator, the areas where crossover provides better results will probably be the same where EDAs will excel over the LO. In [23, 24], the authors proposed the so-called *enumerative BB-wise mutation* which works actually in the same way as the BB-wise local optimizer presented as Alg. 2 in Sec. 3.1. *Under the assumption of the full knowledge of correct linkage information*, it was shown that the GA equipped with this linkage-aware mutation is preferable for deterministic (or low-noise) separable functions, while the GA with similar linkage-aware crossover gives better results in case of separable functions with large noise, or for functions with overlapping building blocks.

In an attempt to build a mutation-based algorithm able to learn the linkage, the enumerative BB-wise mutation operator was preceded by the ECGA model building phase [22]. The whole algorithm then works in the following manner: a sufficiently large population is created, the better individuals are then used to train the ECGA model which provides the linkage information to the BB-wise mutation operator which is run only once for the best individual in the population. With a properly sized population, this algorithm provides systematically better results than the ECGA.

A remarkable local search algorithm with linkage learning was proposed in [9]. The *building block hill climber* (BBHC) algorithm iterates between 2 phases. In phase 1, it iteratively runs BB-wise local search from randomly generated solutions and stores each locally optimal solution in the memory. After the memory is full, phase 2 starts and updates the linkage information based on the current state of memory. With the updated linkage info and empty memory, it continues with phase 1 again. The authors tested the algorithm on *hierarchically decomposable functions* with great results: the number of needed fitness evaluations scales almost linearly with the problem size!

Both the last mentioned local search algorithm provide very attractive results. Despite of that, they need a *properly sized* population (or memory) to discover correct linkage. In the next section, we propose a parameter-free local search approach based on perturbation linkage identification techniques which is ready to use without any need of tuning. The price paid is worse performance compared to both the mentioned local search techniques, but in comparison with ECGA and BOA the algorithm is competitive.

3. ALGORITHM DESCRIPTION

The proposed algorithm is very simple. It alternates 2 phases: (1) BB-wise local search, and (2) a linkage group identification procedure. The pseudocode of the whole procedure is depicted as Alg. 1. It is assumed that the linkage identification procedure is a perturbation-based technique. The algorithm uses no population (nor memory of individuals) and thus does not need to tune the population (memory) size to the problem at hand.

The linkage information is used in the same form as in the ECGA algorithm—in the form of groups of dependent loci

Algorithm 1: LO with Linkage Identification

Input : Function f to be optimized.
Output: The best solution found, $\mathbf{x}_\mathrm{B}$.

1 **begin**
2 $C \leftarrow$ `InitializeBBsAsAllBitsIndependent()`
3 $\mathbf{x} \leftarrow$ `GenerateRandomChromosome()`
4 $\mathbf{x}_\mathrm{B} \leftarrow$ `RunBBwiseLO(`$f, \mathbf{x}, BB(C)$`)`
5 **while not** `TerminationCondition()` **do**
6 $\mathbf{x} \leftarrow$ `GenerateRandomChromosome()`
7 $\{C_\mathrm{new}, \mathbf{x}_\mathrm{LD}\} \leftarrow$ `DetectLinkage(`$f, \mathbf{x}, C$`)`
8 $\mathbf{x}_\mathrm{B} \leftarrow$ `UpdateBSF(`$\mathbf{x}_\mathrm{B}, \mathbf{x}_\mathrm{LD}$`)`
9 **if** $C_\mathrm{new} \neq C$ **then**
10 $\mathrm{BB}_\mathrm{chgd} \leftarrow$ `GetChangedBBs(`C, C_new`)`
11 $\mathbf{x}_\mathrm{LO} \leftarrow$ `RunBBwiseLO(`$f, \mathbf{x}_\mathrm{B}, \mathrm{BB}_\mathrm{chgd}$`)`
12 $\mathbf{x}_\mathrm{B} \leftarrow$ `UpdateBSF(`$\mathbf{x}_\mathrm{B}, \mathbf{x}_\mathrm{LO}$`)`
13 $C \leftarrow C_\mathrm{new}$

Algorithm 2: BB-wise Local Optimizer

Input : Function `f()` to be optimized,
the initial point $\mathbf{x}$,
the building blocks BBs to be searched.
Output: The best solution found, $\mathbf{x}$.

1 **begin**
2 **foreach** $BB \in BBs$ **do**
`/* Exhaustive search in the BB */`
3 $k \leftarrow$ `length(`BB`)`
4 **for** $i \leftarrow 1$ **to** 2^k **do**
5 $\mathbf{x}_\mathrm{cand} \leftarrow$ `ReplaceBBWithConf(`$\mathbf{x}, BB, i$`)`
6 $\mathbf{x} \leftarrow$ `UpdateBSF(`$\mathbf{x}, \mathbf{x}_\mathrm{cand}$`)`

which should be treated together. In this article, we use 2 representations of the linkage information interchangably:

- incidence representation, e.g. $C = [1133167777]$, and
- linkage group enumeration, e.g. $BB = \{(1, 2, 5), (3, 4), (6), (7, 8, 9, 10)\}$.

Both the above examples have the same meaning: bits 2 and 5 are linked with bit 1 (and thus are in the same group or cluster), bits 3 and 4 form another group of dependent bits, bit 6 is not connected with any other bit and bits 7 to 10 form the last group. The information can be also transformed from one form to the other; we shall use the formalism $BB(C)$ or $C(BB)$ for the transformations.

The algorithm starts assuming all loci independent and runs the BB-wise local search (see Sec. 3.1) from a randomly generated point (steps 2 to 4). This phase should solve all completely separable problems (with order-1 building blocks). Then, the algorithm updates the linkage information by running the linkage identification procedure (see Sec. 3.2) from a randomly generated point (steps 6 and 7). Only if the detected linkage structure changed, the algorithm runs the BB-wise local search procedure again. It starts in the best-so-far solution and searches for a better configurations *in the changed linkage groups only.*

3.1 BB-wise Local Optimizer

The BB-wise local optimizer (BBLO, see Alg. 2) is not a new idea. It is basically the same as the *enumerative BB-wise mutation operator* [23], and very similar to the *greedy BB hill-climber* [9]. The algorithm takes the initial point, $\mathbf{x}$, and the linkage groups, BBs, that should be searched. The exhaustive search for optimal configuration is executed in each BB.

3.2 Perturbation-based Linkage Detection

The `DetectLinkage` function (Alg. 3) is a straightforward implementation of the following idea: test the linkage between all possible pairs of loci; if linkage is found, join the groups of both loci.

The number of evaluations consumed by one call to this function is at most $D + D(D-1)/2$, i.e. D for all single bit inversions, $D(D-1)/2$ for all double bit inversions. In practice, the number of needed evaluations may be lower since loci that are already placed in the same linkage group are not tested again. Yet, the number of evaluations scales as $\mathcal{O}(D^2)$.

The algorithm assumes that a pairwise perturbation-based technique will be used as the `PairwiseLinkDetected()` function. Examples of such techniques are the *linkage identification by nonlinearity check* (LINC) [16] or *linkage identification by non-monotonicity detection* (LIMD) [17]. Both methods analyze a 4-tuple of fitness values: a fitness f of the original chromosome, and fitness values f_i, f_j and f_{ij} of the original chromosome with i^{th}, j^{th}, and both bits inverted, respectively.

The LINC method tests for any non-linearity, i.e. linkage between loci i and j is detected if for any chromosome the following condition is satisfied:

$$\Delta f_{ij} \neq \Delta f_i + \Delta f_j, \text{ where} \tag{1}$$

$$\Delta f_* = f_* - f. \tag{2}$$

This method is simple, but vulnerable. Assume we have a fitness function f which nonlinearly transforms the result of a purely linear function of individual bits (e.g. $f = u(\mathbf{x})^2$, where $u(\mathbf{x})$ is the number of ones in $\mathbf{x}$). The LINC procedure would find linkage between all bits, even though the function is still completely separable.

Since not all types of non-linearity are bad for hill-climbers or GAs, the LIMD procedure (Alg. 4) was proposed. It detects only "GA-hard" epistasis by checking the violation of monotonicity conditions, i.e. linkage between loci i and j is detected if for any chromosome any of the following conditions holds:

$$(\Delta f_i > 0 \wedge \Delta f_j > 0) \wedge (\Delta f_{ij} \leq \Delta f_i \vee \Delta f_{ij} \leq \Delta f_j) \tag{3}$$

$$(\Delta f_i < 0 \wedge \Delta f_j < 0) \wedge (\Delta f_{ij} \geq \Delta f_i \vee \Delta f_{ij} \geq \Delta f_j) \tag{4}$$

In the above mentioned example, the LIMD procedure would still recognize all bits as independent.

3.3 Reference Algorithms

Two EDAs are used as a reference. Both are, of course, population-based and estimate the dependencies from the statistical properties of a collection of high-quality chromosomes. The extended compact genetic algorithm (ECGA) [7] uses marginal product model (MPM) to describe the distribution of selected individuals. The MPM has the form of several groups of loci; the groups are assumed to be indepent of each other, while the loci inside each group are

Algorithm 3: DetectLinkage

```
Input  : Function f() to be analyzed,
         the initial point x,
         the current known linkage structure C,
         the pairwise linkage detection procedure,
         PairwiseLinkDetected, e.g. LINC or LIMD.
Output: Updated linkage information C,
         the best point found during the analysis, x_LD.
1  begin
2    D ← length(x)
3    x_LD ← x
4    f ← f (x) /* if not already evaluated */
5    for i ← 1 to D do
6      x_i ← InvertBits(x, i)
7      f_i ← f(x_i)
8      x_LD ← UpdateBSF(x_LD, x_i)
9    for i ← 1 to D − 1 do
10     for j ← i + 1 to D do
         /* Skip bits from the same BB. */
11       if C(i) = C(j) then continue
12       x_ij ← InvertBits(x, i, j)
13       f_ij ← f(x_ij)
14       x_LD ← UpdateBSF(x_LD, x_ij)
15       if PairwiseLinkDetected(f, f_i, f_ij, f_j) then
           /* Join groups of i and j.  */
16         for k ← 1 to D do
17           if C(k) = C(j) then C(k) ← C(j)
```

Algorithm 4: LIMD

```
Input  : Function values f, f_i, f_ij, f_j.
Output: Boolean b: do the four function values violate
         the condition of monotonicity?
1  begin
2    b ← false
     /* Arrange func.values into an array.  */
3    f ← [f, f_i, f_ij, f_j]
4    for i ← 1 to 4 do
5      if (f(1) ≤ f(2) ∧ f(1) ≤ f(4)) then
6        if (f(3) < f(2) ∨ f(3) < f(4)) then
7          b ← true
8          break
9      if (f(1) ≥ f(2) ∧ f(1) ≥ f(4)) then
10       if (f(3) > f(2) ∨ f(3) > f(4)) then
11         b ← true
12         break
       /* Rotate the func.values array.  */
13     f ← rotate(f)
```

described by the joint probability distribution. The model is built by a greedy procedure which starts from MPM describing all bits as independent, and then tries to perform such join of groups that results in the largest improvement of a score. The score used here is the minimum description length (MDL) principle which favours models that describe the data well, but penalizes those which are overly complex.

The other algorithm is the Bayesian optimization algorithm [18]. It uses a Bayesian network (BN) to describe the distribution of selected individuals. The BN is a directed acyclic graph; the vertices correspond to individual loci and the oriented edges to conditional dependencies. Similarly to ECGA, the model is built using a greedy procedure. It starts from a BN with no edges (all loci independent) and then it tries to add, remove, or reverse an edge. The modification that improves a score the most is carried out. The process stops when no improvement is possible. The scores used in BN learning are usually Bayesian (K2 or Bayesian-Dirichlet metric) or based on MDL principle. BOA with MDL score and restricted tournament replacement was used for experiments.

4. TEST FUNCTIONS

In this paper, 3 additively decomposable functions (ADFs) are used. The ADFs are given by

$$f(\mathbf{x}) = \sum_{i=1}^{m} a_i f_i(\mathrm{BB}_i), \tag{5}$$

where all the BB_is are groups of loci and the a_is are scaling coefficients of the individual BBs. The functions f_i are base functions used to evaluate individual BBs and may be different. In this paper, all $a_i = 1$ and all f_is are of the same type. Three different base functions are used: IFF, Sliding IFF and Trap. Since the base versions of all of them are not decomposable, the following evaluation procedure is used to build decomposable functions:

$$f_{m\mathrm{x}k\mathrm{bitBaseFunc}}(\mathbf{x}_\pi) = \sum_{i=1}^{m} f_{k\mathrm{bitBaseFunc}}(x_{k(i-1)}, \ldots, x_{k\cdot i}), \tag{6}$$

where $\mathbf{x}_\pi$ is the chromosome $\mathbf{x}$ shuffled by a permutation π. The permutation is randomly generated, but constant, for each run of the algorithm, making the linked loci to be randomly spread over the whole chromosome. The IFF and Sliding IFF functions are proposed to introduce gradually more complicated interactions inside the building blocks, while the Trap function represents a fully deceptive function [4].

4.1 IFF Function

This function is constructed using the *if and only if* logical function. It relates each pair of neighboring loci and adds 1 to the fitness if both bits are equal. It is defined by the following equations:

$$f_{k\mathrm{bitIFF}}(\mathbf{x}) = 1 + \sum_{i=2}^{k} f_{\mathrm{IFF}}(x_{i-1}, x_i) \tag{7}$$

$$f_{\mathrm{IFF}}(x_1, x_2) = \begin{cases} 1 & \text{if } x_1 = x_2 \\ 0 & \text{if } x_1 \neq x_2 \end{cases} \tag{8}$$

The function is symetric in the sense that each chromosome has the same evaluation as its inversion. Minimal value of this function is 1 and it is attained at chromosomes in which 0s and 1s alternate (i.e. there are no equal neighbors). There are 2 maxima (optima) of this function: in chromosomes full of 0s and full of 1s. Due to the pairwise dependencies, all the loci in the chromosome are linked and form a BB. However, the dependencies are relatively weak.

The decomposable variant constructed using Eq. 6 has 2^m global optima; an algorithm solves the function if it finds any of them.

4.2 Sliding IFF Function

This function also uses the IFF function, but in a different way. It relates 3 consecutive loci. It is defined in the following way:

$$f_{k\text{bitSlidingIFF}}(\mathbf{x}) = 1 + f_{\text{AllEqual}}(\mathbf{x}) + \\ + \sum_{i=3}^{k} f_{\text{IFFPattern}}(x_{i-2}, x_{i-1}, x_i) \quad (9)$$

$$f_{\text{AllEqual}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} = (000\ldots0) \\ 1 & \text{if } \mathbf{x} = (111\ldots1) \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

$$f_{\text{IFFPattern}}(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } f_{\text{IFF}}(x_1, x_2) = x_3 \\ 0 & \text{otherwise} \end{cases} \quad (11)$$

Each 3 bits contribute 1 to the fitness if they form a row of the IFF function truth table, i.e. if $f_{\text{IFF}}(x_1, x_2) = x_3$. A good chromosome is thus constructed of the triples 001, 010, 100, or 111. This gives 2 possibilities for higly fit chromosomes: 001001...001 or 111111...111. The global optimum is, however, the chromosome composed of 1s, since it is favoured by the f_{AllEqual} function.

The decomposable variant constructed using Eq. 6 has one global optimum and $2^m - 1$ local optima.

4.3 Trap Function

The well-known Trap function is defined as

$$f_{k\text{bitTrap}}(\mathbf{x}) = \begin{cases} k & \text{if } u(\mathbf{x}) = k \\ k - 1 - u(\mathbf{x}) & \text{otherwise} \end{cases}, \quad (12)$$

where $u(\mathbf{x})$ is the *unitation* function, i.e. the number of 1s in the chromosome. It has one global optimum in chromosome of 1s and one local optimum in chromosome of 0s. The difficulty of this function is given by the fact that the local optimum has much larger basin of attraction. The linkage identification is also difficult since almost in the whole space the individual loci seem to be independent of each other.

The decomposable variant constructed using Eq. 6 has one global optimum and $2^m - 1$ local optima.

5. SETUP OF EXPERIMENTS

In the next section, we present the results of experiments comparing the local search algorithm with linkage identification described in Sec. 3 and two EDAs: ECGA and BOA. They are compared using the IFF, Sliding IFF and Trap functions described in Sec. 4 for the building block sizes 5 and 8 and dimensionalities ranging from 5 to 240 for $k = 5$ and from 8 to 256 for $k = 8$.

To report the best possible results for BOA and ECGA, the search for the optimal population size was carried out for each problem instance. The reported results belong to the smallest population size with which the algorithm was able to find the solution in 30 consecutive runs. The LO is parameter-free, it was thus simply run 30 times.

In Figs. 1–3, the number of evaluations needed to solve the problem is averaged over 30 runs. For $D > 20$, error bars for LOs in the form ± 1 st.d. were not discernible from the point markers used in the graphs; they would only clutter the figures and thus are not plotted.

6. RESULTS AND DISCUSSION

In this section, the results of experiments are presented and discussed. Before going to the results for the 3 test functions, let us just note that the LO algorithm applied to the OneMax problem (as expected) scales linearly with D and needs at most $D + 1$ evaluations to find the optimum.

6.1 Functions with Easy-to-Detect Linkage

The IFF and SlidingIFF functions contain interactions which are easy to detect by the perturbation methods. It can be easily verified that the LO+LINC method needs only 1 run of LINC to get the right linkage groups no matter what the initial context chromosome is. In this case, LO+LINC performs only 3 steps: (1) the initial local search with the assumption that all bits are independent, (2) linkage detection phase (it tests at most all pairs of bits), and (3) final BB-wise local search to get the optimal solution (exhaustive search in all linkage groups). Thus, the number of evaluations can be upper bounded by

$$n_{\text{evals}} \leq \underbrace{D + 1}_{\text{Step 1}} + \underbrace{D + \frac{D(D-1)}{2}}_{\text{Step 2}} + \underbrace{\frac{D}{k} 2^k}_{\text{Step 3}} \\ = 1 + D\left(2 + \frac{D-1}{2} + \frac{2^k}{k}\right) \quad (13)$$

This upper bound is plotted in Figs. 1 and 2 as the dash-dotted line. The same statement holds also for LO+LIMD with the exception of Sliding IFF function with 8bit BBs. As can be seen in Fig. 2 (right), the number of evaluations needed by LO+LIMD is slightly higher than the upper bound for problem dimensions around 100 and above. For Sliding IFF function with building block of size $k \geq 8$, there exist chromosomes which provide such a context that some of the pairwise interactions are not GA-hard: the nonlinearity is detected (LINC detects the right structure), but not the non-monotonicity (LIMD does not indicate linkage). Thus, sometimes more than 1 run of the LIMD procedure is required and thus the upper bound does not hold.

The effect of easy-to-detect linkage is that the LO+LI needs virtually the same number of evaluations independently of the particular fitness function (compare the left-hand side graphs in Figs. 1 and 2 for functions with 5bit BBs, and the right-hand side graphs for functions with 8bit BBs). The changes in differences between EDAs and LOs are caused almost entirely by the EDAs performance.

Regarding the performance of both EDAs, it can be observed that with increasing BB size, the difficulty of the problem for EDAs increases more dramatically than for LOs (compare the steepness of the lines in the left and right graphs in Figs. 1 to 3). Also, on the contrary to LOs, the Sliding IFF function with its order-3 dependencies inside BBs is harder for EDAs than the IFF function with order-2 dependencies only. These two factors—larger BBs and higher-order interactions inside BBs—may cause rather large gap between the performance of EDAs and LOs: for 5bit Sliding IFF function, EDAs seem to be almost 10 times slower than LOs, while for 8bit Sliding IFF function, the difference gets larger with increasing dimensionality.

6.2 Function with Hard-to-Detect Linkage

The most prominent example of function containing hard-to-detect linkage is the Trap function. By analyzing the

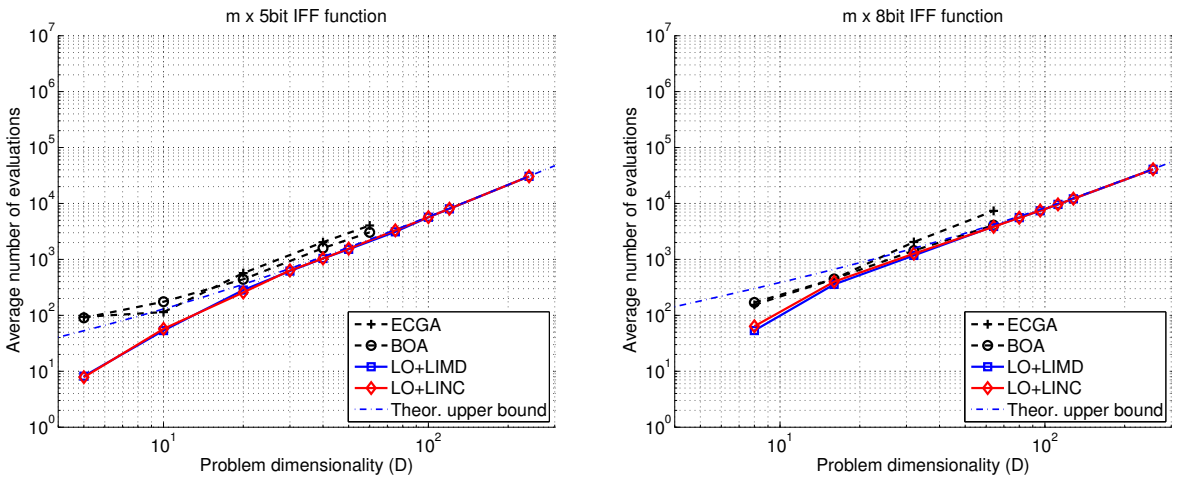

Figure 1: The average number of evaluations needed to solve the IFF problem as a function of the problem dimensionality for the building block size $k = 5$ (left) and $k = 8$ (right).

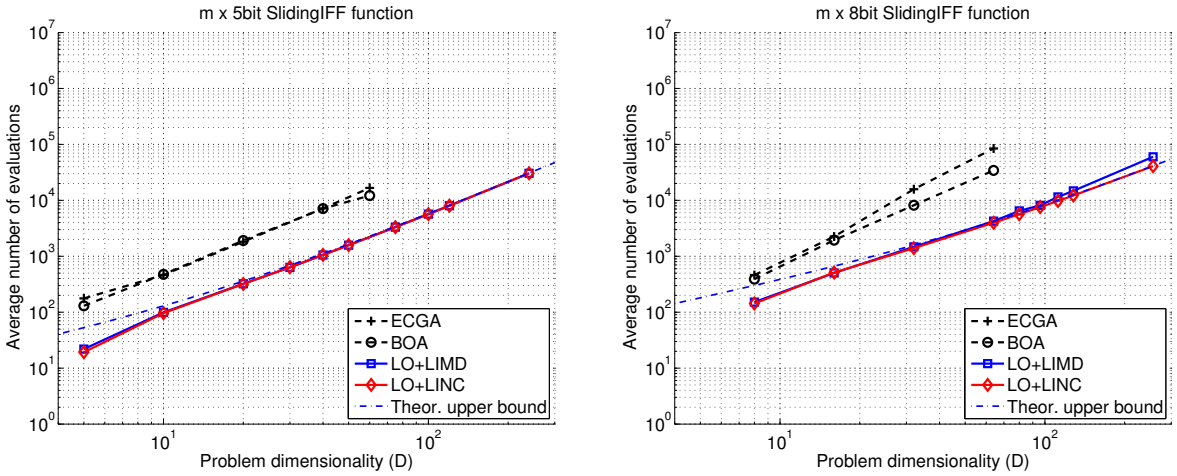

Figure 2: The average number of evaluations needed to solve the SlidingIFF problem as a function of the problem dimensionality for the building block size $k = 5$ (left) and $k = 8$ (right).

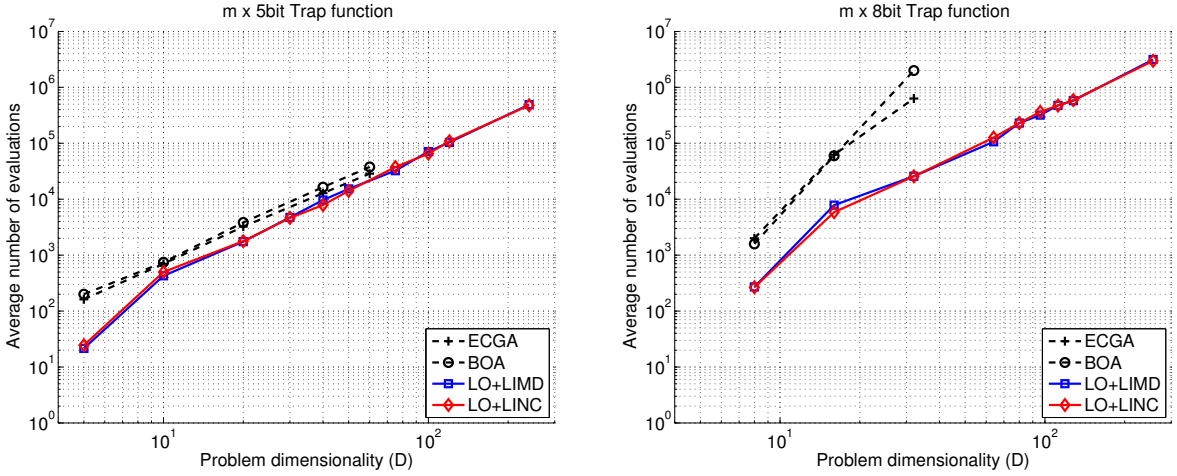

Figure 3: The average number of evaluations needed to solve the Trap problem as a function of the problem dimensionality for the building block size $k = 5$ (left) and $k = 8$ (right).

Table 1: Scalability models for 5bit BBs

Function	Alg.	Scalability: $N \approx \ldots$	R^2
IFF	LOLIMD	$0.73 \cdot D^{1.95}$	0.9974
		$1.26 \cdot D^{1.68} \cdot \log D$	0.9979
IFF	LOLINC	$0.74 \cdot D^{1.95}$	0.9985
		$1.28 \cdot D^{1.68} \cdot \log D$	0.9991
SlidingIFF	LOLIMD	$1.44 \cdot D^{1.80}$	0.9991
		$2.51 \cdot D^{1.53} \cdot \log D$	0.9965
SlidingIFF	LOLINC	$1.40 \cdot D^{1.81}$	0.9993
		$2.44 \cdot D^{1.54} \cdot \log D$	0.9970
Trap	LOLIMD	$2.43 \cdot D^{2.23}$	0.9991
		$4.23 \cdot D^{1.96} \cdot \log D$	0.9983
Trap	LOLINC	$2.70 \cdot D^{2.20}$	0.9982
		$4.69 \cdot D^{1.93} \cdot \log D$	0.9959

Table 2: Scalability models for 8bit BBs

Function	Alg.	Scalability: $N \approx \ldots$	R^2
IFF	LOLIMD	$3.22 \cdot D^{1.70}$	0.9998
		$5.18 \cdot D^{1.45} \cdot \log D$	0.9994
IFF	LOLINC	$3.96 \cdot D^{1.66}$	0.9996
		$6.37 \cdot D^{1.41} \cdot \log D$	0.9984
SlidingIFF	LOLIMD	$4.07 \cdot D^{1.69}$	0.9940
		$6.56 \cdot D^{1.44} \cdot \log D$	0.9883
SlidingIFF	LOLINC	$6.14 \cdot D^{1.57}$	0.9981
		$9.87 \cdot D^{1.32} \cdot \log D$	0.9947
Trap	LOLIMD	$15.14 \cdot D^{2.18}$	0.9947
		$24.36 \cdot D^{1.93} \cdot \log D$	0.9912
Trap	LOLINC	$10.82 \cdot D^{2.26}$	0.9991
		$17.40 \cdot D^{2.01} \cdot \log D$	0.9987

fitness changes when playing with pairs of bits, only a few context chromosomes actually allow to detect any linkage, although there is epistasis between all pairs of loci. The results for both EDA and LOs are depicted in Fig. 3.

Comparing the results for 5bit Trap function visually, the LOs are slightly better (needing only about half of the evaluations) and scale similarly to EDAs (at least for the tested problem dimensionalities). Regarding the 8bit Trap function, the differences are much larger (EDAs are at least 10 times slower) and increase with the problem dimensionality.

6.3 Scalability

Since we do not have any sound scalability model for our algorithm, we have used the models $N \approx aD^b$ (polynomial scaling suggested by the almost straight lines in the log-log space) and $N \approx aD^b \log D$ (appropriate for EDAs) and fitted them to the results of the LOs (omiting the first data point for 1 repetition of BB) by linear regression in log-log space. (We do not provide results of similar fits for EDAs due to small number of available measurements.) The results are shown in Tables 1 and 2.

The coefficients of determination, R^2, suggest that both models would be acceptable if we should base our opinion just on the available data. (However, the purely polynomial model results in a bit better fit more often.)

Although the LOs were found to be similarly or more efficient then both the EDAs for the tested problems and dimensionalities, it should be noted that the number of evaluations for LOs can scale quadratically (or a bit worse) with the problem dimension. The scaling models reported for EDAs on the Trap function are of the form $\mathcal{O}(D^b \log D)$, where b was observed to be from $(1.5, 2)$. This means that from certain value of D, the EDAs should be more efficient than LOs. The results shown at Figs. 1 to 3 do not indicate when this happens (perhaps with the exception of the IFF function with 8bit BBs), however the observed trends suggest that LOs should hold their superiority at least for problems with a few hundreds of bits.

7. SUMMARY AND CONCLUSIONS

We have proposed a completely parameter-free local search algorithm that is able to learn and use the linkage information LO+LI. As a proof of concept, the results on 3 additively decomposable functions were reported and compared to the results of ECGA and BOA. For the tested problems and dimensions, it can be stated that the results are at least comparable, or that LO+LI provided better results than EDAs with carefully tuned population size. From the trends in scalability graphs, it can be hypothesized that the LO+LI algorithm should remain superior to BOA and ECGA for a decomposable problems with a few hundreds of bits.

The algorithm has *no parameters to tune*—after specifying the termination condition, the algorithm is ready to use. This the unique distinguishing feature of the algorithm. The other neat feature is the fact that after the algorithm stops, it can provide correct (though not necessarilly complete) linkage information to the user. This may not be the case for ECGA and BOA—their linkage models may be corrupted by building the model on overly converged population.

Regarding the behaviour of the algorithm on functions with *non-uniformly scaled BBs*: Since the linkage detector is perturbation-based and analyzes the fitness function values, we expect the algorithm to cope with non-uniformly scaled BBs the same way as with uniformly scaled ones. No matter if the fitness differences are large or low, perturbation techniques should detect them.

No tests on functions with *overlapping BBs* have been performed yet. The linkage detector as used in this paper takes advantage of LINC or LIMD procedures which decide if two bits are linked together or not. It gives no distinction between strong and weak epistasis. If the reference functions were not separable, the whole chromosome would be identified as one large block, and the exhaustive search would be infeasible (scales exponentially with the BB size). The majority of real-world problems belong to this class. As a future work, we plan to use a different linkage identification procedure, e.g. LIEM [14] or LIEM^2 [15], able to provide a strength of the epistasis. This will allow us to reasonably decompose non-decomposable problems into BBs that would cover the strongest dependencies, and test the algorithm on real-world problems.

We also have not tested *the influence of noise* yet. Based on [24], we hypothesize that for noise with high magnitude the EDA algorithms would perform better thanks to population and the "averaging" features of model building. Yet, it remains to be confirmed.

Thanks to the parameter-free nature of the algorithm, we think that the algorithm could become a suitable baseline algorithm for various empirical studies.

8. ACKNOWLEDGMENTS

The project was supported by the Ministry of Education, Youth and Sport of the Czech Republic with the grant No. MSM6840770012 entitled "Transdisciplinary Research in Biomedical Engineering II".

9. REFERENCES

[1] S. Baluja. Population based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Technical Report CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.

[2] S. Baluja and S. Davies. Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In D. Fisher, editor, *14th International Conference on Machine Learning*, pages 30–38. Morgan Kaufmann, 1997.

[3] J. S. de Bonet, C. L. Isbell, and P. Viola. MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems*, 9:424–431, 1997.

[4] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. Technical Report IlliGAL Report No. 91009, University of Illinos, Urbana-Champaign, IL., 1991.

[5] R. Etxeberria and P. Larrañaga. Global optimization using bayesian networks. In A. Rodriguez, M. Ortiz, and R. Hermida, editors, *CIMAF 99, Second Symposium on Artificial Intelligence*, Adaptive Systems, pages 332–339, La Habana, 1999.

[6] D. E. Goldberg, K. Sastry, and T. Latoza. On the supply of building blocks. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO–2001))*, pages 336–342, 2001.

[7] G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical Report IlliGAL Report No. 99010, University of Illinois, Urbana-Champaign, 1999.

[8] R. B. Heckendorn and A. H. Wright. Efficient Linkage Discovery by Limited Probing. *Evol. Comput.*, 12(4):517–545, December 2004.

[9] D. Iclanzan and D. Dumitrescu. Overcoming hierarchical difficulty by hill-climbing the building block structure. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1256–1263, New York, NY, USA, 2007. ACM.

[10] H. Kargupta. The Gene Expression Messy Genetic Algorithm. In *Proceedings of the International Conference on Evolutionary Computation*, pages 814–819, 1996.

[11] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms*. GENA. Kluwer Academic Publishers, 2002.

[12] H. Mühlenbein and T. Mahnig. FDA -a scalable evolutionary algorithm for the optimization of additively decomposed functions. *Evolutionary Computation*, 7(4):353–376, Dec. 1999.

[13] H. Mühlenbein and G. Paass. From recombination of genes to the estimation of distributions i. binary parameters. In *Parallel Problem Solving from Nature*, pages 178–187, 1996.

[14] M. Munetomo. Linkage identification based on epistasis measures to realize efficient genetic algorithms. In *Proceedings of the World Congress on Computational Intelligence*, volume 2, pages 1332–1337, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[15] M. Munetomo. Linkage identification with epistasis measure considering monotonicity conditions. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*, pages 550–554, 2002.

[16] M. Munetomo and D. E. Goldberg. Identifying Linkage by Nonlinearity Check. Technical Report IlliGAL Report No. 98012, University of Illinois, Urbana-Champaign, October 1998.

[17] M. Munetomo and D. E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7:377–398, December 1999.

[18] M. Pelikan. *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. Studies in Fuzziness and Soft Computing. Springer, 1 edition, March 2005.

[19] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage problem, distribution estimation, and bayesian networks. Technical Report IlliGAL Report No. 98013, University of Illinois, Urbana-Champaign, 1998.

[20] M. Pelikan and H. Mühlenbein. The bivariate marginal distribution algorithm. In *Advances in Soft Computing – Engineering Design and Manufacturing*, pages 521–535, 1999.

[21] R. Santana, P. Larrañaga, and J. A. Lozano. Challenges and open problems in discrete EDAs. Technical Report EHU-KZAA-IK-1/07, Department of Computer Science and Artificial Intelligence, University of the Basque Country, October 2007.

[22] K. Sastry and D. E. Goldberg. Designing Competent Mutation Operators Via Probabilistic Model Building of Neighborhoods. In *Genetic and Evolutionary Computation Conference, GECCO 2004*, pages 114–125, 2004.

[23] K. Sastry and D. E. Goldberg. Let's Get Ready to Rumble: Crossover Versus Mutation Head to Head. In *Genetic and Evolutionary Computation Conference, GECCO 2004*, pages 126–137, 2004.

[24] K. Sastry and D. E. Goldberg. Let's get ready to rumble redux: crossover versus mutation head to head on exponentially scaled problems. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1380–1387, New York, NY, USA, 2007. ACM.

[25] S. Vaníček. Binary local optimizer with linkage learning. Technical report, Czech Technical University in Prague, Prague, Czech Republic, 2010.