Overcoming Hierarchical Difficulty by Hill-Climbing the Building Block Structure

David Iclănzan
david.iclanzan@gmail.com

Dan Dumitrescu
ddumitr@cs.ubbcluj.ro

Department of Computer Science
Babeş-Bolyai University, Kogălniceanu no. 1
Cluj-Napoca, 400084, Romania

ABSTRACT

The Building Block Hypothesis suggests that Genetic Algorithms (GAs) are well-suited for hierarchical problems, where efficient solving requires proper problem decomposition and assembly of solution from sub-solution with strong non-linear interdependencies. The paper proposes a hill-climber operating over the building block (BB) space that can efficiently address hierarchical problems. The new Building Block Hill-Climber (BBHC) uses hill-climb search experience to learn the problem structure. The neighborhood structure is adapted whenever new knowledge about the underlying BB structure is incorporated into the search. This allows the method to climb the hierarchical structure by revealing and solving consecutively the hierarchical levels. It is expected that for fully non-deceptive hierarchical BB structures the BBHC can solve hierarchical problems in linearithmic time. Empirical results confirm that the proposed method scales almost linearly with the problem size thus clearly outperforms population based recombinative methods.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Global Optimization—*Analyze*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search

General Terms

Algorithms, Design, Theory

Keywords

Hill-climbing, Adaptive Neighborhood Structure, Linkage Learning, Model Building, Hierarchy

1. INTRODUCTION

One of the most important research goal regarding Evolutionary Algorithms (EAs) is to understand the class of problems for which these algorithms are most suited. Despite the major work in this field it is still unclear how an EA explores a search space and on what fitness landscapes will a particular EA outperform other optimizers such as hill-climbers.

The traditional GA theory is pillared on the Building Block Hypothesis (BBH) which states that GAs work by discovering, emphasizing and recombining low order schemata in high-quality strings in a strongly parallel manner [5].

Albeit being widely used to justify claims about EAs, the BBH remains controversial [16, 3]. While there are situations where the BBH provides a good explanation about GAs intrinsic search mechanisms, there are also cases where BBH in the current form does not provide much useful insight.

In the early 90's a systematic program was initiated by Mitchell et al. [8] to address these issues concerning the fundamentals of GAs. Their strategy was to find a set of features that are of particular relevance to GAs and test the performance of these algorithms on landscapes containing those features. It was recognized that major tenets behind the BBH are the notion of problem decomposition and the assembly of solutions from sub-solutions. Subsequently, they constructed a set of functions that clearly emphasize a gross-scale BB structure with low-order BBs that recombine to higher-order ones. These functions were expected to lay out a "royal road" for GAs, while hill-climbers were anticipated to perform poorly as a large number of positions must be optimized simultaneously to discover higher-order BBs. To much of a surprise both expectation were refuted: on these test suites GAs performed worse than expected due to the hitchhiking phenomena while a Random Mutation Hill-Climber which accepts states with equal objective function value greatly outperformed GAs.

Later developments proposed NK landscapes [6] and the expanded function method [21] which presented non-separable components on a single level.

Inspired by the fact that many real-world systems are hierarchical, Watson et al. [18, 20] proposed a class of hierarchically decomposable functions which present a strong non-linear hierarchical BB interdependency. This class of function is very hard for mutation based hill-climbers as the Hamming distance between local optima and global optima is very large. It is considered that this class of functions exemplifies those problems for which GAs are well-suited.

The objective of this paper is to develop a hill-climber that can solve hierarchical problems. The proposed method operates on BBs rather than bits and uses search experience to learn linkages and adapt the neighborhood structure. The continuous neighborhood structure accommodation allows the hill-climber to hierarchically decompose the problems, revealing the hierarchical levels one after the other; when it finishes the problem structure is delivered in an explicit manner that is transparent to human researchers.

The following section summarizes the characteristics of hierarchically decomposable problems. Section 3 revisits the notion of hill-climbing; introduces and informally describes the concept of hierarchical BB hill-climbing. The proposed method is presented in details in Section 4. Section 5 presents empirical results of the proposed method. Finally, the paper is concluded in Section 6 by discussion and some future works.

2. HIERARCHICALLY DECOMPOSABLE FUNCTIONS

Although having a gross-scale BB structure, hierarchical problems are hard to solve without proper problem decomposition as the blocks from these functions are not separable.

The fundamental of hierarchically decomposable problems is that a BB can have multiple context-optimal settings. Therefore, there is always more than one way to solve a (sub-) problem [18], leading to the separation of BBs "fitness" i.e. contribution to the objective function from their meaning. This conceptual separation induces the non-linear dependencies between BBs: providing the same objective function contribution, a BB might be completely suited for one context whilst completely wrong for another one. Therefore, the fitness of a BB can be misleading if it is incompatible with its context. However, the contribution of the BBs indicate how can the dimensionality of the problem be reduced by expressing one block in a lower level as one variable in the upper level.

Hierarchical problems are very hard for mutation based hill-climbers as they exhibit a fractal-like structure in the Hamming space with many local optima [18]. This bitwise landscape is fully deceptive; the better is a local optimum the further away is from the global ones. At the same time the problem can be solved quite easily in the BB or "crossover space", where the block-wise landscape is fully non-deceptive [18].

Methods that can solve certain hierarchical problems include the Symbiogenic Evolutionary Adaptation Model [19], DevRep [1], Compact Genetic Codes [15], Hierarchical Genetic Algorithm [2], Hierarchical Bayesian Optimization Algorithm [11] and the DSGMA++ [23]. The most powerful methods can optimize problems with random linkage.

Methods optimizing by hierarchical decomposition can naturally be divided in two classes, according to how the decomposition information is stored. For example in [11], the hBOA stores the decomposition information implicitly in a Bayesian network. Another method, the DSMGA++ [23], by using dependency structure matrix clustering techniques is able to express the decomposition information explicitly. Consequently, this method is able to deliver the problem structure in a comprehensible manner for humans, which constitutes a major advantage in many real-world applications.

2.1 Hierarchical Problems

In this paper three hierarchical test functions are used: the hierarchical IFF [18], the hierarchical XOR [20] and the hierarchical trap function [11]. These problems are defined on binary strings of the form $x \in \{0,1\}^{k^p}$, where k is the number of sub-blocks in a block, and p is the number of hierarchical levels. The meaning of sub-blocks is separated from their fitness by the means of a boolean function h, which determines if the sub-block is valid in the current context or not. In the shuffled version of these problems the tight linkage is disrupted by randomly reordering the bits. The functions with their particularities are detailed as follows.

2.1.1 Hierarchical if and only if (hIFF)

The hIFF has $k = 2$ and it is provided by the if and only if relation, or equality. Let $L = x_1, x_2, \dots, x_{2^{p-1}}$ be the first half of the binary string x and $R = x_{2^{p-1}+1}, x_{2^{p-1}+2}, \dots, x_{2^p}$ the second one. Then h is defined as:

$$h_{iff}(x) = \begin{cases} 1 & \text{, if } p = 0; \\ 1 & \text{, if } h_{iff}(L) = h_{iff}(R) = 1 \text{ and } L = R; \\ 0 & \text{, otherwise.} \end{cases} \tag{1}$$

Based on h_{iff} the hierarchical iff is defined recursively:

$$H_{iff}(x) = H_{iff}(L) + H_{iff}(R) + \begin{cases} length(x), & \text{if } h_{iff}(x) = 1; \\ 0 & \text{, otherwise.} \end{cases} \tag{2}$$

At each level $p > 0$ the $H_{iff}(x)$ function rewards a block x if and only if the interpretation of the two composing sub-blocks are both either 0 or 1. Otherwise the contribution is zero.

The hIFF has two global optima: strings formed only by 0's or only by 1's. At the lowest level the problem has $2^{l/2}$ local optima where l is the problem size.

2.1.2 Hierarchical XOR (hXOR)

The global optima of hIFF are formed by all 1's or all 0's, which may ease the task of some methods biased to replicate particular allele values. To prevent the exploitation of this particular problem property the hXOR was designed [20]. This problem is much more difficult due to its reduced potential for exploiting repetitiveness.

The definition of hXOR is analogous with the hIFF, having only a modification in the validation function h, where instead of equality we do a complement check:

$$h_{xor}(x) = \begin{cases} 1 & \text{, if } p = 0; \\ 1 & \text{, if } h_{xor}(L) = h_{xor}(R) = 1 \text{ and } L = \bar{R}; \\ 0 & \text{, otherwise.} \end{cases} \tag{3}$$

$\bar{R}$ stands for the bitwise negation of R.

The two global optima of hXOR are composed by half zeros and half ones. Having the same problem structure, one would expect that an algorithm which applies problem decomposition to perform equally well on both problems. As already mentioned, this is not always the case as some methods may be biased to replicate particular alleles, solving the hIFF in an easier manner.

2.1.3 The Hierarchical Trap Function (hTrap)

The underlying structure of the hTrap is a balanced k-ary tree, where $k \geq 3$. Blocks from lower level are interpreted by a *mapping function* similar to the one from the hIFF: a block of all 0's and 1's is mapped to 0 and 1 respectively, and everything else is interpreted as '-' or *null*.

The contribution function is a trap function of unitation (its value depends only on the numbers of 1's in the input string) of order k, based on two parameters f_{high} and f_{low} which define the degree of deception.

Let u be the unitary of the input string. Then the trap function is defined as:

$$trap_k(u) = \begin{cases} f_{high} & \text{, if } u = k; \\ f_{low} \times \frac{k-1-u}{k-1} & \text{, otherwise.} \end{cases} \quad (4)$$

If any position in the input string is null ('-') then the contribution is zero.

In this paper we use hTrap function based on $k = 3$ and f_{high} and f_{low} set to 1 for all except the highest level. The decision between competing BBs can be carried out only on the highest level, where $f_{high} = 1$ and $f_{low} = 0.9$.

3. HILL-CLIMBING IN THE BUILDING BLOCK SPACE

Hill-climbing is used widely in artificial intelligence fields, for quickly reaching a goal state from a starting position. The hill-climbers are usually the fastest methods but they can easily get trapped in local optima. The current section revisits the notion of hill-climbing and neighborhood structure. Furthermore, it introduces the idea of a BB hill-climber that can solve hierarchical problems by exploiting the BB structure and adapting its neighborhood structure online.

3.1 Hill-climbers and neighborhood structure

Hill-climbing is an optimization technique that starts from some initial solution and iteratively tries to replace the current solution by a better one, from an appropriately defined neighborhood of the current state. In *simple* or *first improvement hill-climbing*, the first better solution is chosen, whereas in *steepest ascent* or *best improvement hill-climbing* all successors are compared and the best solution is selected.

To avoid getting traped in a local optimum, usually random-restart hill-climbing is employed. This method simply runs an outer loop over hill-climbing. Each step of the outer loop chooses a random initial state s to start hill-climbing. The best solution encountered is kept.

Usually hill-climbers described in the literature use bit-flipping for replacing the current state [8, 9]. This implies a neighborhood structure which contains strings that are relatively close in Hamming distance to the original state. This make those methods unsuited for solving hierarchical problems, where local optima and global optima are distant in Hamming space. But the neighborhood can be defined as an arbitrary function, which assign to a valid state s a set of valid states $N(s)$. The main idea of the paper is to build a trajectory method which takes into account the hierarchical BB structure of the problems and defines its neighborhood structure accordingly.

3.2 Building Block wise search

As already indicated in Section 2, hierarchical problems are fully deceptive in Hamming space and fully non-deceptive in the BB space. The problem representation together with the neighborhood structure defines the search landscape. Recent local-search literature authors have emphasized the importance of using a good neighborhood operator [17] With an appropriate neighborhood structure – which operates on BBs – the search problem can be transferred from Hamming space to a very nice, fully non-deceptive search landscape, which should be easy to hill-climb.

Hill-climbing in BB space was proposed and shown to be more efficient than selectorecombinative GAs on deterministic additively-separable problems of bounded difficulty [13, 12, 7]. These studies assume that the BB-wise mutation operators have the knowledge of the BBs and thus can effectively perform local search among promising schemata. Starting from a random individual, BBs in different partitions are mutated in a sequential manner. The BB-wise mutation evaluates all possible schemas in a given partition. For a BB of size l, 2^l individuals are evaluated. The best out of 2^l individuals is chosen and used for mutating BBs of other partitions.

In the case of hierarchical problems we have high order dependencies, up to the case where all variables are interdependent. Having big partitions, searching all 2^l possible schemas is not feasible. Also, in hierarchical problems a proper niching must be applied and the competing subsolutions must be kept until the method advances to upper levels, where a correct decision can be made. Nevertheless, the knowledge about the underlaying BB structure must be evolved along with the solution(s) in order to reduce the search space. Only then the hierarchical difficulty can be overcome. We devise our method by taking into account these observations.

Let us denote the current BB knowledge at state s by

$$BB(s) = (b_1, b_2, \ldots, b_n) \quad (5)$$

where b_i-s are the BBs and n is the number known BBs.

Each BB b_i can have multiple configurations:

$$V_i = \{v | v \in \{0,1\}^l\} \quad (6)$$

where l is the length of b_i. This allows the sustenance and parallel processing of competing context-optimal schemata. For example in the case of the hIFF, a BB b_i of length 2 may have two valuable configurations: $V_i = \{(0,0),(1,1)\}$.

The current state s is formed by particular configurations of the known BBs:

$$s = (v(b_1), v(b_2), \ldots, v(b_n)) \quad (7)$$

where n is the number known BBs and $v(b_i) \in V_i$ is a candidate configuration of the BB b_i.

Instead of flipping bits as in classical mutation based hill-climbers, the proposed method hill-climbs the BB structure by choosing the best configuration v_i of every BB in a greedy manner. This approach shifts the search from the Hamming space to the BB landscape and engenders a BB-wise neighborhood structure. The hill-climb on the BB structure will get the state s to the nearest local optimum, according to the current BB knowledge.

While EAs exploit BB structure by probabilistic recombination, this approach applies a systematic combination and analysis of BBs.

3.3 Online adaptation of the neighborhood structure from search experience

It is important for a GA to conserve BBs under crossover. Theoretical studies denote that a GA that uses crossover which does not disrupt the BB structure, holds many advantages over simple GA [14]. To achieve this goal linkage learning is applied and the solution representation is evolved along with the population, during the search process.

Similarly, in order to be able to hill-climb the BB landscape, the BB structure of the problem must be learned and the representation of the individual must be evolved to reflect the current BB knowledge. The changing of representation implies the adaptation of the neighborhood structure, which is the key to conquer hierarchical problems: by exploring the neighborhood of the current BB configuration the next level of BBs can be detected.

In order to be able to identify linkages we enhance the hill-climber with a memory where hill-climbing results are stored. Evolutionary Algorithms with linkage learning mechanism extract the BB information from the population. Similar techniques can be applied to devise BB structures from search experience stored in the memory. However, the solutions stored in the memory offer an important advantage over populations: they are noise free. While individuals from populations may have parts where good schemata have not yet been expressed or have BBs slightly altered by mutation, in the case of fully non-deceptive BB landscapes, the solutions stored in memory are always exact local optima. The systematic exploration of BB configurations guarantees that in the close neighborhood of these states there are no better solutions.

The details of linkage learning mechanism and BB construction are detailed in the next section.

4. BUILDING BLOCK HILL-CLIMBER

BBHC involves four main steps: (i) initialization of the algorithm with each single locus as a BB; repeatedly (ii) hill-climb the search space according to a BB-wise neighborhood structure; (iii) local optima obtained in (ii) is used to detect linkages and extract BB information; (iv) the BB configuration and implicitly the neighborhood structure are updated. This section describes the framework of BBHC and details the implementation.

4.1 The BBHC Framework

Figure 1 depicts the two main phases of the BBHC optimization. The first one refers to the accumulation of search experience, provided by the repeated hill-climbing. The second phase concerns the exploitation of search experience by linkage learning and BB structure update.

The input of the second phase is the search experience stored in memory. Dependencies are detected and the output consists of an updated BB structure, which enables the first phase to combine new BBs. In hierarchical problems, modeled after the suggestions of the BBH, the assembling of lower level BBs leads to the development of higher order ones. Thus the sequence of phases can effectively overcome hierarchical levels successively by discovering and incorporating BB knowledge into the search process.

4.2 The Building Block hill-climbing

BB hill-climbing is rather straightforward: instead of flipping bits, the search focuses on the best local context-optimal BB configuration. Each BB is processed systematically by testing its configurations and selecting the one which provides the highest (or lowest in the case of minimization) objective function value. While the search for the best con-

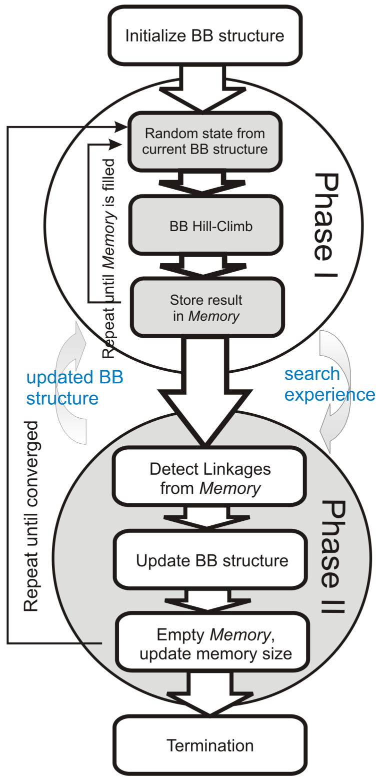

Figure 1: The framework of BBHC with the two main phases: accumulation and exploitation of search experience.

figuration of a particular BB is carried out, the configurations of the other BBs are hold still. The BB hill-climbing technique is outlined in Figure 3.

As an example, let us consider the current BB structure of an 8-bit problem instance, composed of three BBs: $BB(s) = (b_1, b_2, b_3)$. The BBs describe the loci of the instance in the following way (see Figure 2): $b_1 = \{locus\,2, locus\,6\}$, $b_2 = \{locus\,3, locus\,5\}$ and $b_3 = \{locus\,1, locus\,4, locus\,7, locus\,8\}$. Each BB defines multiple promising schema: $V_1 = \{00, 11\}$, $V_2 = \{00, 01, 11, 10\}$, $V_3 = \{0000, 1111\}$.

The state $s = (2, 3, 1)$ expresses the bit configuration obtained from the second configuration of b_1, the third configuration of b_2 and the first configuration of b_3: $decode(s) = 01101100$.

In what follows it is explained how b_2 is hill-climbed starting from the state $s = (2, 3, 1)$. All candidate configurations of b_2 taken from V_2 are analyzed in the current context, where the configurations of b_1 and b_3 remain fixed ($v(b_1) = 2$, $v(b_2) = 1$). The neighboring states of s are $s_1 = (2, \mathbf{1}, 1)$, $s_2 = (2, \mathbf{2}, 1)$, $s_3 = (2, \mathbf{3}, 1)$, $s_4 = (2, \mathbf{4}, 1)$, corresponding to the bit configurations $decode(s_1) = 01\mathbf{0}0\mathbf{0}100$, $decode(s_2) = 01\mathbf{0}0\mathbf{1}100$, $decode(s_3) = 01\mathbf{1}0\mathbf{1}100$, $decode(s_4) = 01\mathbf{1}0\mathbf{0}100$. The most suitable configuration for b_2 is chosen with respect to the objective function and the original state is updated

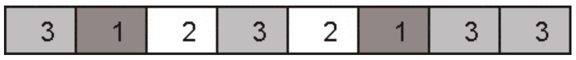

Figure 2: BB configuration of the 8-bit state.

```
HC($s$)

  1. Choose randomly an unprocessed building block
     $b_i$ from $B(s)$;

  2. Choose randomly an unprocessed building block
     configuration $v_j \in V_i$;

  3. Set $v(b_i)$ in $s$ to $v_j$;

  4. If the change results in a decrease of the
     objective function undo the change;

  5. If there exists unprocessed building block
     configuration of $V_i$ then goto 2;

  6. If there exists unprocessed building block
     from $BB(s)$ then goto 1;
```

Figure 3: The pseudo code of the deterministic, greedy building block search.

accordingly. For instance, if the first configuration provides the best objective function value, then the state becomes $s = (2, \mathbf{1}, 1)$. The algorithm carries on by hill-climbing another BB in the same manner from the new state .

After all BBs are processed, the result is stored in the memory and the process is repeated starting from a new random state, until the memory is full.

In the next phase, linkages from the memory are detected and the BB structure is updated.

4.3 Linkage detection and BB structure update

Several techniques for detecting gene dependency from a population are presented in the literature [10, 23]. Similar techniques can also be used to detect the linkage from the states stored in the memory. Due to the fact that the hierarchical problems under study have a nice non-deceptive structure in the BB space, a very simple method for linkage detection is considered.

The clustering of loci in new BBs is done by searching for bijective mappings. For a given block b_i, all BBs b_j are linked if distinct configurations of b_i map to distinct configurations of b_j. The configurations of b_i that can be found in the memory represent the domain while the configurations of b_j from the memory are the codomain. Thus, we say that b_i and b_j are linked if there is a one-to-one correspondence between their configurations i.e. for any particular value v_i from the domain there exist a unique configuration $v_j \in V_j$ satisfying the condition: for any state from the memory if $v(b_i) = v_i$ then $v(b_j) = v_j$.

Due to the transitivity property of bijective mappings, all relevant BBs are discovered simultaneously. The linkage detection algorithm is presented in Figure 4.

In the studied hierarchical problems there is a deterministic dependency between BBs, which corresponds to maximal mutual information. The BB-wise hill-climbing enables a noise free expression of these dependencies in the states stored in the memory, thus directly facilitates the deterministic approach of searching for bijective mappings.

Harder problems, for example problems exhibiting overlapping BB structure or problems that are heavily corrupted by noise, may require a more traditional *probabilistic* linkage analysis.

```
BBForm($s$,$M$)

  1. Choose randomly a building block $b_i$ from
     $BB(s)$ which has not yet been clustered;

  2. Compute the set of building blocks whose
     configuration from $M$ are mapped bijectively
     to $b_i$ and denote it by $L$;

  3. If $L$ is empty update the possible
     configurations $V_i$ to the configurations
     encountered in $M$;

  4. If $L$ is not empty form a new building block
     $b_{new} = b_i \bigcup L$ from the union of loci from $b_i$
     and from building blocks in $L$.  Also set the
     possible values $V_{new}$ to all distinct
     configuration encountered, on the position
     defined by the $b_{new}$, operating on the binary
     representation of states from $M$.

  5. Set $b_i$ and the building blocks from $L$ as
     clustered;

  6. If there exists building blocks which have not
     been clustered goto 1;
```

Figure 4: The linkage detection and new BB forming algorithm, where M is the memory.

It should be noted that testing for all possible bijections might be quite high cost for large problem instances. As for each BB we potentially check all other BBs, the complexity of the deterministic linkage detection is $O(n^2 \cdot m^2)$ where n is the number of BBs and m is the size of the memory.

All BBs linked together by a bijective mapping are united into a new BB. The candidate configurations of the new BBs are extracted from the binary representation of states from the memory. If a BB can not be linked with any other BB it keeps its original place and only its possible configurations are updated in the same manner as for the new BBs.

4.4 The memory size

The main choice to be made when applying BBHC concerns the number of BB hill-climbs to be performed before we proceed to linkage learning (the size of the memory). As dependencies are successively detected and bigger and bigger BBs are formed, the search space is reduced. Lower dimensionality translates to less search experience needed to reveal remaining linkages. Therefore, with the reduction of the dimensionality the size of the memory can also be diminished. On the other hand, if some context-optimal setting of a BB is not discovered and stored in the memory, the missing setting will be excluded for the remainder of the search. If the excluded configuration is critical to find the optimum then the algorithm will fail. The memory size should be chosen big enough to be able to detect all context-optimal settings for each BB with a sufficiently high probability.

For BB-wise non-deceptive hierarchical problems, all important context-optimal settings have a relatively high probability of detection. Thus repeatedly hill-climbing the BB

```
BBHC($x, c, b, k$) returns $best\_state$

 1. Initialize the building block knowledge with
    each single locus from $x$ as a building block;

 2. Initialize the memory size:
        $size[M] := c + log_b(length(x))$;

 3. Generate a random state $s$ according to the
    current BB structure knowledge $BB(s)$:
        $s := RandomState(BB(s))$;

 4. BB hill-climb from $s$ and store the result in
    memory:  $M := M \bigcup HC(s)$;

 5. If the resulted state is better then the best
    states seen so far, keep the new state:
        $s := best(s, best\_state)$

 6. If M is not filled up goto 3;

 7. Learn linkage from memory and update the BB
    configuration according to the detected
    linkages:  $BBForm(s, M)$;

 8. Empty memory:  $M = \emptyset$;

 9. Update the memory size:
        $size[M] := c + log_b(\aleph(BB(s)))$;

10. If there was an improvement in the last $k$
    epochs and the number of maximum function
    evaluations was not exceeded goto 3;
```

Figure 5: Outline of the hill-climbing enhanced with memory and linkage learning. In steps 3–6 we accumulate the search experience (phase 1) which is exploited in steps 7–9 (phase 2).

structure from randomly chosen points quickly increases the probability of detecting all relevant context-optimal settings.

A bigger issue concerns the chance of getting false linkage detection. This phenomena can appear when the memory size is small, the number of BBs is relatively high and BBs have only a few context-optimal settings. In this case, a bijective mapping between two BBs may appear in the memory by pure chance, even if there is no mutual dependency between them. Increasing the memory size to a number proportional with the number of BBs can overcome this problem.

In this paper we use the formula $mem\_size(s) = \lfloor c + log(\aleph(BB(s)))\rfloor$ for the memory size, where c is a constant and $\aleph(BB(s))$ is the number of elements from $BB(s)$.

In the case of hierarchical non-deceptive problems (in the BB space), the probability of failure can be bound as a function of the memory size. If one knows the probability of each context-optimal setting in advance, the size of memory required to achieve correctness with sufficiently high probability can be calculated. Results concerning the probability of failure for problems where context-optimal settings have the same probability of instantiation was reported elsewhere.

The proposed model can be summarized by the algorithm presented in Figure 5.

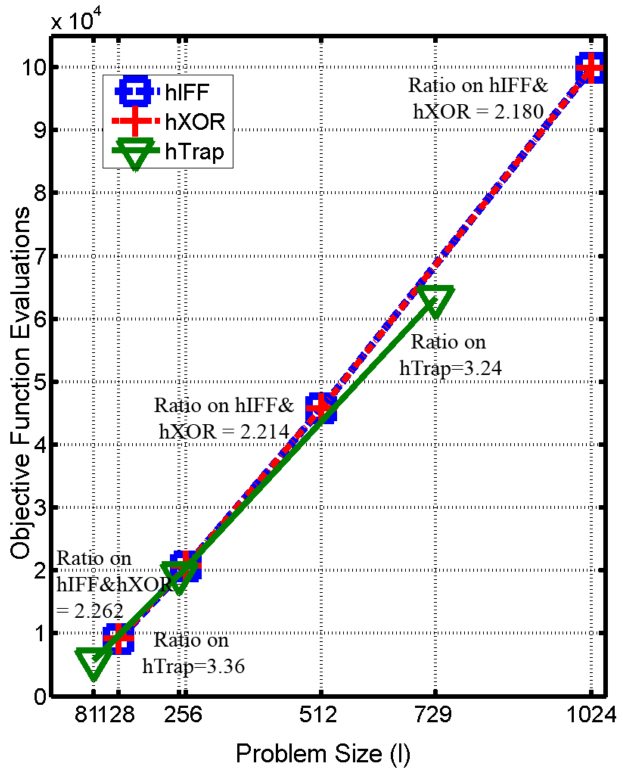

Figure 6: Arithmetic scaling of BBHC on hIFF, hXOR and hTrap. The number of function evaluations scales almost linearly. The ratio between neighborhood points is decreasing towards 2 as the problem sizes are doubled in the case of hIFF and hXOR and towards 3 in the case of the hTRap.

5. RESULTS

We tested the scalability of BBHC on 128-bit, 256-bit, 512-bit and 1024-bit shuffled hIFF and hXOR problems, respectively on 81-bit, 243-bit and 729-bit shuffled hTrap problem instances. Lower problem sizes were not addressed as they may be too easy to solve; any conclusion from them may be misleading. For each test suit a total number of 100 independent runs were averaged.

The size of the memory was set to $\lfloor 8+log_2(length(s))\rfloor$ on hIFF and hXOR, respectively to $\lfloor 18 + log_3(length(s))\rfloor$ on hTrap. With these settings there was no need for a random restart mechanism as the algorithm converged in all runs to a global optimum.

The arithmetic scaling results with the ratio between neighborhood points is presented in Figure 6. On the test suites, the proposed method scales up almost linearly with the problem size, with the slope between neighbor points decreasing towards 2 as the problem size doubles on hIFF and hXOR and towards 3 on hTrap as problem size is tripled.

The almost linear scaling is the direct result of the fact that each level of the hierarchical problems is solved by the BBHC with $O(L \cdot log(L))$ complexity, where L is the size of the level. The slopes of hIFF and hXOR line up due to the identical problem structure.

The experimental results were approximated with functions of the form $f(x) = ax^b \cdot log(x)$ where a and b are

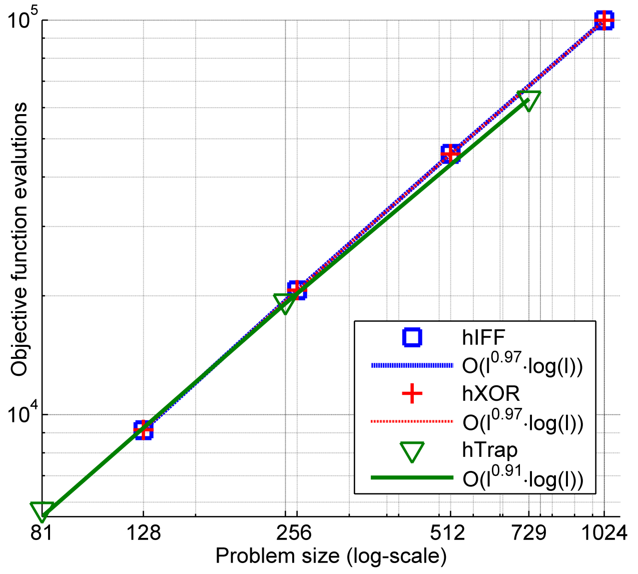

Figure 7: The number of function evaluations of BBHC approximately scales as $O(l^{0.97} \cdot log(l))$ on hIFF and hXOR and $O(l^{0.91} \cdot log(l))$ on hTrap, where l is the problem size.

determined by the least square error method. In Figure 7 the log-log scaled plot of the test results and approximation functions are shown. The approximated number of objective function evaluations scales as $O(l^{0.97} \cdot log(l))$ on hIFF and hXOR and $O(l^{0.91} \cdot log(l))$ on hTrap, where l is the problem size. The approximations are very close to the expected linearithmic time. The best results reported till now scale up sub-quadratically with a lower bound of $O(l^{1.5} \cdot log(l))$ [10].

The hBOA, one of the best known optimizers that operate via hierarchical decomposition, with hand tuned parameters solves the 256-bit shuffled hIFF in approximately 88000 function evaluations. The BBHC performance on the same test suit is 20666, approximately four times quicker than the hBOA. Due to the linearithmic scaling the BBHC is able to solve the 512-bit version of the same problem approximately twice as fast as the hBOA does the 256-bit one, requiring only 45793 function evaluations!

Similarly to other methods like the DSMGA++, the BBHC uses explicit chunking mechanism enabling the method to deliver the problem structure. While DSGMA++ and other stochastic methods have to fight the sampling errors which sometimes induce imperfections, the BBHC was able to detect the perfect problem structure in all runs, due to its more systematic and deterministic approach. The enhanced capability of BBHC to capture the problem structure is also revealed by the fact that hIFF and hXOR are solved approximately in the same number of steps as their underlying BB structures (a balanced binary tree) coincide. For the DSGMA++ the time needed to optimize the two problems differs significantly, being $O(l^{1.84} \cdot log(l))$ for the hIFF and $O(l^{1.96} \cdot log(l))$ on hXOR.

On hIFF and hXOR where there are two global optima, the multiple runs of the BBHC showed an unbiased behavior, finding in almost half-half proportion both solutions.

A final remark concerns the stability of BBHC: the highest standard deviation encountered is 210, while other methods deal with standard deviations of much higher magnitude on the same test suites.

6. CONCLUSION AND FUTURE WORK

The concept of BB hill-climber (BBHC) a generic method for solving problems via hierarchical decomposition is proposed. The BBHC operates in the BB space, where it combines BBs in a systematic and exhaustive manner. Past hill-climbing experience is used to learn the underlying BB structure of the search space expressed by linkages. The continuous update of the BB representation of the individual results implicitly in the adaptation of the neighborhood structure to the combinative neighborhood of the current BBs. In hierarchical problems – where BBH holds – moving the search to the combinative vicinity of the current BB representation facilitates the discovery of new BBs, as BBH implies that low-order BBs can be combined to form higher-order ones.

An important aspect of the proposed method is that similar to DSMGA++, BBHC delivers the problem structure in a form comprehensible to humans. Gaining knowledge about the hidden, complex problem structure can be very useful in many real-world applications.

BBHC clearly outperforms population based recombinative methods on different issues of hierarchical problems. So far hBOA proved the greatest ability to solve hierarchical problems with random linkage. Nevertheless, the proposed method solves the 512-bit shuffled version of hIFF and hXOR approximately twice as fast then the hBOA solves the smaller version of 256-bit!

Preliminary scalability test of the proposed method indicates that BBHC holds not only a quantitative advantage over other methods but also a qualitative one too: it scales linearithmic with the problem size.

Recent results have shown that in dynamical environments induced by exogenous noise recombinative methods clearly outperform hill-climbers [4, 13]. Nevertheless, this result reposes some questions on the fundamentals of EAs as we now must again look for those topological features which make a *static* fitness landscape suited for EAs. It had been shown that some of the earlier proposed landscape features, like hierarchically structured BBs, deception, multimodality can be efficiently overcome by a hill-climber using a proper neighborhood structure and local search operators aware of the underlaying BB embodiment.

Also the suggestion of BBH regarding the use of population is put at question: according to the BBH population is needed to combine BBs in a highly parallel manner. Other results [13, 7] and this paper show that a systematic exploration of BBs combination by a hill-climber can be more efficient.

The No Free Lunch Theorem [22] guarantees that there exists problems where GAs outperform other methods. The main issue regards relevance: how much if any of this class of function is related to real-world problems?

We think that the task of finding problems for which GAs are well suited must be approached from direction that so far got little or no attention in the literature, due to doubtful expectances regarding the role of different features of GAs. The results of the paper suggest that if a problem has a nice structure, even if "hidden" like the BB space, a proper hill-climber can outperform population based recombinative methods, *without requiring extra domain knowledge*. The idea of a GA marching on a fitness landscape is maybe a little bit romantic; a suitable hill-climber is almost certainly quicker if there is a nice structure of the problem to be exploited. Maybe we should look for hard problems which can be solved somewhat slothfully by GAs but are intractable using other methods.

Another observation is that test problems for GAs were usually developed under the intuitions of the BBH. As it is believed that crossover should produce successful offspring on average, test problems were devised accordingly. To best of our knowledge, so far there are no test suites that exploit the creativity potential of the crossover operator.

Based on these two observations a future paper is intended to present a class of problems for which GAs are hopefully well-suited.

7. ACKNOWLEDGMENTS

This work was sponsored by MEdC-CNCSIS under grant AT-8/70-2006 and by the Sapientia Institute for Research Programs (KPI).

8. REFERENCES

[1] E. D. de Jong. Representation Development from Pareto-Coevolution. In E. C.-P. et al., editor, *GECCO '03: Proc. of the Genetic and Evolutionary Computation Conference*, pages 262–273. Springer, LNCS, vol. 2723, July 2003.

[2] E. D. de Jong, D. Thierens, and R. A. Watson. Hierarchical Genetic Algorithms. In X. Y. et al., editor, *Proc. Parallel Problem Solving from Nature*, pages 232–241, Birmingham, UK, 18-22 Sept. 2004. Springer, LNCS.

[3] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, NY, 1995.

[4] D. E. Goldberg, K. Sastry, and X. Llora;. Toward routine billion-variable optimization using genetic algorithms: Short communication. *Complex.*, 12(3):27–29, 2007.

[5] J. H. Holland. *Adaptation in natural artificial systems*. University of Michigan Press, Ann Arbor, 1975.

[6] S. A. Kauffman. *The Origins of Order*. Oxford University Press, Oxford, 1993.

[7] C. F. Lima, K. Sastry, D. E. Goldberg, and F. G. Lobo. Combining competent crossover and mutation operators: a probabilistic model building approach. In *GECCO '05: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 735–742, NY, USA, 2005. ACM Press.

[8] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela, editor, *Proc. of the First European Conference on Artificial Life*, pages 245–254, Cambridge, MA, 1992. MIT Press.

[9] H. Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In R. Männer and B. Manderick, editors, *Proc. of the Second Conference on Parallel Problem Solving from Nature (PPSN II)*, pages 15–25, Amsterdam, 1992. North-Holland.

[10] M. Pelikan. *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer Verlag, 2005.

[11] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. In L. S. et al., editor, *GECCO '01: Proc. of the Genetic and Evolutionary Computation Conference*, pages 511–518, San Francisco, California, USA, 7-11 2001. Morgan Kaufmann.

[12] K. Sastry and D. E. Goldberg. Designing competent mutation operators via probabilistic model building of neighborhoods. In *GECCO '04: Proc. of the Genetic and Evolutionary Computation Conference, Seattle, WA, USA,*, pages 114–125. Springer, LNCS, vol. 3103, June 26–30, 2004.

[13] K. Sastry and D. E. Goldberg. Let's get ready to rumble: Crossover versus mutation head to head. In *GECCO '04: Proc. of the Genetic and Evolutionary Computation Conference, Seattle, WA, USA,*, pages 126–137. Springer, LNCS, vol. 3103, June 26–30, 2004.

[14] D. Thierens and D. E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *Proc. of the 5th International Conference on Genetic Algorithms*, pages 38–47, San Francisco, CA, USA, 1993. Morgan Kaufmann.

[15] Toussaint. Compact genetic codes as a search strategy of evolutionary processes. In *International Workshop on Foundations of Genetic Algorithms (FOGA)*, volume 8. Springer, LNCS, 2005.

[16] M. D. Vose. A critical examination of the schema theorem. Technical Report ut-cs-93-212, University of Tennessee, Computer Science Department, Knoxville, TN, 1993.

[17] Watson, Beck, Howe, and Whitley. Problem difficulty for tabu search in job-shop scheduling. *AIJ: Artificial Intelligence*, 143, 2003.

[18] R. A. Watson, G. Hornby, and J. B. Pollack. Modeling building-block interdependency. In *PPSN V: Proc. of the 5th International Conference on Parallel Problem Solving from Nature*, pages 97–108, London, UK, 1998. Springer, LNCS.

[19] R. A. Watson and J. Pollack. A computational model of symbiotic composition in evolutionary transitions. *Biosystems*, 69(2-3):187–209, May, 2003. Special Issue on Evolvability, ed. Nehaniv.

[20] R. A. Watson and J. B. Pollack. Hierarchically consistent test problems for genetic algorithms: Summary and additional results. In S. Brave, editor, *GECCO '99: Late Breaking Papers at the Genetic and Evolutionary Computation Conference*, pages 292–297, Orlando, Florida, USA, 13 July 1999.

[21] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In L. Eshelman, editor, *Proc. of the Sixth International Conference on Genetic Algorithms*, pages 239–246, San Francisco, CA, 1995. Morgan Kaufmann.

[22] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997.

[23] T.-L. Yu and D. E. Goldberg. Conquering hierarchical difficulty by explicit chunking: substructural chromosome compression. In *GECCO '06: Proc. of the Genetic and Evolutionary Computation Conference*, pages 1385–1392, NY, USA, 2006. ACM Press.