Missouri Estimation of Distribution Algorithms Laboratory

Hierarchical BOA on Random Decomposable Problems

Martin Pelikan, Kumara Sastry, Martin V. Butz, and David E. Goldberg

MEDAL Report No. 2006001

March 2006

Abstract

The hierarchical Bayesian optimization algorithm (hBOA) can solve nearly decomposable and hierarchical problems scalably and reliably. This paper describes a class of *random additively decomposable problems* with and without interactions between the subproblems and tests hBOA on a large number of random instances of the proposed class of problems. The performance of hBOA is compared to that of the simple genetic algorithm with standard crossover and mutation operators, the univariate marginal distribution algorithm, and the hill climbing with bit-flip mutation. The results confirm that hBOA achieves quadratic or subquadratic performance on the proposed class of random decomposable problems and that it significantly outperforms all other methods included in the comparison. The results also show that low-order polynomial scalability is retained even when only a small percentage of hardest problems are considered and that hBOA is a robust algorithm because its performance does not change much across the entire spectrum of random problem instances of the same structure. The proposed class of decomposable problems can be used to test other optimization algorithms that address nearly decomposable problems.

Keywords

Hierarchical BOA, univariate marginal distribution algorithm, genetic algorithm, hill climbing, decomposable problems, performance analysis, scalability, optimization.

Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Department of Mathematics and Computer Science
University of Missouri–St. Louis
One University Blvd., St. Louis, MO 63121
E-mail: medal@cs.umsl.edu
WWW: http://medal.cs.umsl.edu/

Hierarchical BOA on Random Decomposable Problems

Martin Pelikan
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu

Kumara Sastry
Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801
kumara@illigal.ge.uiuc.edu

Martin V. Butz
Department of Cognitive Psychology
University of Würzburg
Röntgenring 11, 97070 Würzburg, Germany
mbutz@psychologie.uni-wuerzburg.de

David E. Goldberg
Illinois Genetic Algorithms Laboratory, 107 TB
University of Illinois at Urbana-Champaign
104 S. Mathews Ave., Urbana, IL 61801
deg@illigal.ge.uiuc.edu

Abstract

The hierarchical Bayesian optimization algorithm (hBOA) can solve nearly decomposable and hierarchical problems scalably and reliably. This paper describes a class of *random additively decomposable problems* with and without interactions between the subproblems and tests hBOA on a large number of random instances of the proposed class of problems. The performance of hBOA is compared to that of the simple genetic algorithm with standard crossover and mutation operators, the univariate marginal distribution algorithm, and the hill climbing with bit-flip mutation. The results confirm that hBOA achieves quadratic or subquadratic performance on the proposed class of random decomposable problems and that it significantly outperforms all other methods included in the comparison. The results also show that low-order polynomial scalability is retained even when only a small percentage of hardest problems are considered and that hBOA is a robust algorithm because its performance does not change much across the entire spectrum of random problem instances of the same structure. The proposed class of decomposable problems can be used to test other optimization algorithms that address nearly decomposable problems.

Keywords: Hierarchical BOA, univariate marginal distribution algorithm, genetic algorithm, hill climbing, decomposable problems, performance analysis, scalability, optimization.

1 Introduction

There are three important approaches to testing optimization algorithms:

(1) *Testing on the boundary of the design envelope using artificial test problems.* For example, concatenated traps (1; 5) represent a class of artificial test problems that can be used to test whether the optimization algorithm can automatically decompose the problem and exploit the discovered decomposition effectively.

(2) *Testing on classes of random problems.* For example, to test algorithms for solving maximum satisfiability (MAXSAT) problems, large sets of random formulas in conjunctive normal form can be generated and analyzed (3; 6).

(3) *Testing on real-world problems or their approximations.* For example, the problem of designing military antennas can be considered for testing (22).

The primary purpose of this paper is to introduce a class of additively decomposable problems, which can be used to test optimization algorithms that address nearly decomposable problems. There are three goals in the design of the proposed class of problems:

(1) *Scalability.* It should be straightforward to control problem size and difficulty in order to test scalability.

(2) *Known optimum.* It should be possible to efficiently discover the global optimum of any problem instance so that it can be verified whether the global optimum was found.

(3) *Easy generation of random instances.* It should be possible to generate a large number of instances of the proposed class of problems.

The paper then applies several genetic and evolutionary algorithms to a number of random instances of the proposed class of problems. Specifically, the hierarchical Bayesian optimization algorithm (hBOA) is considered and its performance on the proposed class of test problems is compared to genetic algorithms (GAs) with standard variation and mutation operators, the univariate marginal distribution algorithm (UMDA), and the stochastic hill climbing (HC). The results show that hBOA significantly outperforms other algorithms included in the comparison, and they provide a number of other interesting insights into both the difficulty of decomposable problems as well as the performance of various evolutionary algorithms on these problems.

The paper starts with a brief description of hBOA and other genetic and evolutionary algorithms considered in this paper in Section 2. Section 3 introduces the class of random additively decomposable problems. Section 4 presents and discusses experimental results. Section 5 outlines important topics for future work. Finally, Section 6 summarizes and concludes the paper.

2 Compared Algorithms

This section describes the genetic and evolutionary algorithms included in the experimental section of this paper. Two estimation of distribution algorithms (EDAs) (15; 20; 13) are outlined first: hBOA and UMDA. Next, the simple genetic algorithm and the stochastic hill climbing are briefly described. In all cases, candidate solutions are represented by fixed-length binary strings.

2.1 Hierarchical BOA (hBOA)

The hierarchical Bayesian optimization algorithm (hBOA) (18; 19; 17) evolves a population of candidate solutions. The first population is usually generated at random according to a uniform distribution over all candidate solutions. In each iteration (generation), the population is updated using two basic operators: (1) selection and (2) variation. The selection operator selects more copies of better solutions at the expense of the worse ones from the current population, yielding a population of promising candidates. In this paper, binary tournament selection without replacement is used. The variation operator starts by learning a probabilistic model of the selected solutions. hBOA uses Bayesian networks with local structures (4) to model promising solutions. The variation operator then proceeds by sampling the probabilistic model to generate new candidate solutions. The new solutions are incorporated into the original population using the restricted tournament replacement (RTR) (11), which ensures that useful diversity in the population is maintained for long periods of time. For the window size w in RTR, we use a rule of thumb that sets $w = \min\{n, N/20\}$ where n is the problem size and N is the population size (17). The run is terminated when a good enough solution has been found, when the population has not improved for a long time, or when the number of generations has exceeded a given upper bound.

It was theoretically and empirically shown that hBOA can solve nearly decomposable and hierarchical problems of bounded difficulty scalably and reliably (17; 21).

To provide a fair comparison, both the genetic algorithm and the univariate marginal distribution algorithm share most of the basic procedure with hBOA and the only difference between these algorithms and hBOA is the variation operator.

2.2 Univariate Marginal Distribution Algorithm (UMDA)

The univariate marginal distribution algorithm (UMDA) (15) uses a simple univariate marginal probabilistic model to model promising solutions and sample the new ones. Since the probabilistic model used in UMDA considers each variable independently, UMDA fails for many problems where variables interact strongly (17; 8).

2.3 Genetic Algorithm (GA)

The basic procedure of the genetic algorithm (GA) (12; 7) is similar to that of hBOA; however, the variation operator proceeds by applying crossover and mutation to the selected population of solutions instead of building and sampling a Bayesian network. Crossover combines bits and pieces of promising solutions, whereas mutation perturbs the selected solutions slightly. Here we use two standard crossover operators: one-point crossover, which exchanges the tails after a randomly selected position between pairs of selected solutions, and uniform crossover, which exchanges the bits in each position with probability 50%. To mutate solutions, bit-flip mutation is used, which flips each bit of each selected solution with a specified probability p_m; p_m is usually small so that only one or a few bits are flipped on average.

Since standard variation operators of GAs can effectively process only low-order partial solutions, standard GAs can only solve problems where variables that interact strongly are located tightly in solution strings (23; 8).

2.4 Hill Climbing (HC)

The hill climbing (HC) differs from the aforementioned algorithms mainly by the fact that it works with only one candidate solution instead of a population of candidate solutions. HC starts with a random binary string. In each iteration, the current string is modified with a mutation operator. If the new string outperforms the original string (with respect to the objective function), it replaces the original string; otherwise, the new string is discarded. In this paper, bit-flip mutation is used to perturb the solution string in each iteration.

3 Random Additively Decomposable Problems of Bounded Difficulty

This section describes the class of random decomposable problems with and without overlap. Candidate solutions are assumed to be represented by binary strings of fixed length, but the proposed class of problems can be generalized to fixed-length strings over any finite alphabet in a straightforward manner. We also assume that the task is to *maximize* the objective function (fitness function).

The section starts by presenting the general form of additively decomposable problems. Then, the section discusses separable problems of bounded order, which represent the simplest variant of boundedly difficult additively decomposable problems. Next, interactions between different subproblems are introduced by creating an overlap between subproblems. Finally, the section describes how to generate random instances of the proposed classes of decomposable problems with and without overlap.

3.1 Additively Decomposable Problems

The fitness function for an n-bit additively decomposable problem can be written as the sum of subfunctions defined over subsets of string positions:

$$f(X_1, X_2, \ldots, X_n) = \sum_{i=1}^{m} f_i(S_i),$$

where n is the number of bits in a candidate solution, X_i is the variable corresponding to the ith bit of a candidate solution, m is the number of subproblems or subfunctions, f_i is the ith subproblem, and $S_i \subset \{X_1, \ldots, X_n\}$ is the subset of variables (string positions) for the ith subproblem.

Clearly, any fitness function can be written in the above form because any problem can be trivially decomposed into one subproblem containing all the variables. The difficulty of additively decomposable problems depends on the order of subproblems, the subproblems themselves, and their interaction through string positions that are contained in multiple subproblems. The remainder of this section defines two subsets of additively decomposable problems, shows how to generate random instances of the described classes of decomposable problems, and outlines an efficient method to solve these problem instances.

3.2 Separable Problems of Order k

A separable problem of order k consists of m subproblems of k bits each. There is no overlap between the subproblems and each bit belongs to exactly one of the subproblems; therefore, the overall number of bits is $n = m \times k$ (see Figure 1a for an illustrative example).

(a) Separable problem.

(b) Decomposable problem with 1-bit overlaps (tight linkage).

(c) Decomposable problem with 1-bit overlaps (no tight linkage).

Figure 1: Examples of additively decomposable problems with 4 subproblems of 4 bits each. Each string position (bit) is displayed as a rectangle and the string positions corresponding to one subproblem are filled with the same color. For problems with overlapping subproblems, the string positions that are located in more subproblems are split along the diagonal.

To fully specify a separable problem, the following components must be specified:

- The number of subproblems, m.
- The size of the subproblems (order of the problem decomposition), k.
- The subsets of string positions for all subproblems, $S_1, S_2, \ldots, S_m$.
- The subfunctions $f_1, f_2, \ldots, f_m$; each subfunction can be specified by a list of 2^k values, which define the output values for all possible k-bit strings.

For a known problem decomposition specified by subsets S_1 to S_m, any separable problem can be solved using a simple deterministic algorithm that evaluates all possible combinations of k bits for each subset S_i (setting all remaining bits to some fixed value), and fixes the positions in the corresponding string positions to the values that maximize the fitness function. This can be done in $O(2^k m)$ evaluations of the fitness function. Therefore, for separable problems of bounded difficulty for which k is fixed or upper bounded by a constant, the global optimum can be obtained in $O(n)$ evaluations.

If the problem decomposition is unknown, the computational complexity of GAs with standard crossover operators can grow exponentially fast (23; 8; 17). However, several genetic and evolutionary algorithms exist that can automatically identify and exploit appropriate problem decomposition for separable problems (linkage learning); these algorithms can solve separable problems of bounded difficulty in a quadratic or subquadratic number of fitness evaluations (8; 17; 9). Even HC based on bit-flip mutation is expected to achieve polynomial performance; however, HC may require as many as $O(n^k \log n)$ evaluations to find the global optimum of order-k separable problems (14).

3.3 Incorporating Overlap

In separable problems, each subproblem can be considered independently of the remaining subproblems because the bits in any subproblem do not affect the fitness contributions of other subproblems. However, although many real-world problems can be additively decomposed into subproblems of bounded order, many of these problems are much more difficult than separable problems of bounded order because the subsets corresponding to the different subproblems *overlap*. For example, both the problem of finding ground states of Ising spin glasses and maximum satisfiability for 3-CNF formulas (MAX3SAT) are additively decomposable problems of order 2 or 3, but they both are NP-complete as a result of complex interactions between the different subproblems (2; 16). This section describes a straightforward modification of separable problems that introduces overlap between the different subproblems to analyze how overlap affects the performance of various genetic and evolutionary algorithms.

Here the amount of overlap is specified by a parameter $o \in \{0, 1, \ldots, k-1\}$ called *overlap*. Then, the first subproblem is defined in the first k string positions. The second subproblem is defined

in the last o positions of the first subproblem and the next $(k-o)$ positions. All the remaining subproblems are assigned string positions analogically, always defining the next subproblem in the last o positions of the previous subproblem, and the next $(k-o)$ positions. For example, for $k=3$, $o=1$, and $m=3$ subproblems, the first subproblem is defined in positions $(1,2,3)$, the second subproblem is defined in positions $(3,4,5)$, and the third subproblem is defined in positions $(5,6,7)$. Note that each string position is contained in one or two subproblems and that for m subproblems with overlap o, the overall number of bits is $n=k+(m-1)(k-o)$. Separable problems of order k are a special case of decomposable problems of order k with no overlap, that is, $o=0$. See Figure 1b for an illustrative example.

To ensure that the subproblems are not always located in consequent string positions, the string can be reordered according to a randomly generated permutation. See Figure 1c for an example of a randomly reordered decomposable problem with overlap.

Assuming that the problem is decomposable according to the above definition and that we know the subsets S_1 to S_m and the subfunctions f_1 to f_m, it is possible to solve any problem instance using a deterministic algorithm based on dynamic programming in $O(2^k n)$ fitness evaluations. However, the algorithm for solving problem instances with non-zero overlap is somewhat more complicated than the one for separable problems, because the subproblems interact and in the general case it is impossible to set the bits corresponding to any subproblem independently of any other subproblem. We discuss a deterministic algorithm for solving this class of problems that can be directly applied to overlap $o \leq k/2$; the presented algorithm can be easily extended to problems with higher values of o by grouping consequent subproblems.

The deterministic algorithm for solving decomposable problems with overlap iterates through all subproblems, starting with one of the two subproblems that overlap with only one other subproblem via o string positions. Each next iteration considers one of the unprocessed subproblems that interacts with the last processed subproblem via o string positions. For example, consider the aforementioned problem with $n=7$, $k=3$, and $o=1$, where the subproblems are defined in the following subsets of positions: $(1,2,3)$ for subproblem f_1, $(3,4,5)$ for f_2, and $(5,6,7)$ for f_3. The dynamic programming algorithm could consider the subproblems in either of the following permutations: (f_1,f_2,f_3) or (f_3,f_2,f_1).

The dynamic programming algorithm starts by creating a matrix $G=(g_{i,j})$ of size $(m-1)\times 2^o$, where $g_{i,j}$ for $i \in \{1,2,\ldots,m-1\}$ and $j \in \{0,1,\ldots,2^o-1\}$ encodes the maximum fitness contribution of the first i subproblems according to the considered permutation of subproblems under the assumption that the o bits that overlap with the next subproblem (that is, with the $(i+1)$th subproblem) are equal to j using integer representation for these o bits. For example, for the above example problem of $n=7$ bits and permutation (f_1,f_2,f_3), $g_{2,0}$ represents the best fitness contribution of f_1 and f_2 (ignoring f_3) under the assumption that the 5th bit is 0; analogically, $g_{2,1}$ represents the best fitness contribution of f_1 and f_2 under the assumption that the 5th bit is 1.

The algorithm starts by considering all 2^k instances of the k bits in the first subproblem, and records the best found fitness for each combination of values of the o bits that overlap with the second subproblem; the resulting values are stored in the first row of G (elements $g_{1,j}$ for $j \in \{0,\ldots,2^o-1\}$). Then, the algorithm goes through all the remaining subproblems except for the last one, starting in the second subproblem, and ending in the $(m-1)$th subproblem. For ith subproblem, all 2^k instances of the k bits in this subproblem are examined. For each instance, the algorithm first looks at the column j' of G that corresponds to the o bits of ith subproblem that overlap with the previous subproblem. Then, the algorithm computes the fitness contribution of the first i subproblems assuming that the first $(i-1)$ subproblems are set optimally and the ith subproblem is set to the considered instance of k bits; this fitness contribution is computed as the

sum of $g_{i-1,j'}$ and the fitness contribution of the considered instance of the ith subproblem. The values in the ith row of G are then computed from the fitness contributions computed as described above.

In the last step, all 2^k instances of the last subproblem are considered and their fitness contributions are computed analogically to other subproblems, using the $(m-1)$th row of G and the fitness contributions of the mth subproblem. The maximum of these values is the best fitness value we can obtain. The values that lead to the optimum fitness can be found by examining all choices made when choosing the best combination of bits in each subproblem.

If the decomposition is unknown, overlap can be expected to further increase problem difficulty of additively decomposable problems. In other words, it can be expected that separable problems will be as difficult or easier for all algorithms discussed in this paper; this hypothesis is confirmed by a number of experiments in Section 4. The experiments also show that the effects of overlap on evolutionary algorithms based on local operators (HC) are much stronger than those on evolutionary algorithms based on recombination (hBOA,GA and UMDA).

3.4 Generating Random Problems

To generate random instances of the class of additively decomposable problems defined above, the following parameters must be first specified by the user:

1. Number of subproblems, m.
2. Order of decomposition, k.
3. Overlap, o.

Then, the following parameters are generated randomly:

1. The 2^k values that specify each subfunction f_i where $i \in \{1, \ldots, m\}$; overall, there are $2^k m$ values to generate.
2. The permutation of string positions to eliminate the assumption of tight linkage (so that the bits in each subproblem are not always located in consequent positions).

A key issue is what random distributions to use to generate the subfunctions and random permutations. In this work, we generate all 2^k values of each subfunction from a uniform distribution over interval $[0, 1)$. The permutation is also generated from a uniform distribution so that each permutation has the same probability. Of course, other distributions can be considered for both the subfunctions and the permutations; for example, the 2^k values for each subfunction can be distributed according to a Gaussian distribution and the permutations can be biased to enforce loose linkage.

4 Experiments

This section outlines experiments and presents experimental results. The section starts with a description of test problems and experiments. Then, the results of all experiments are presented.

4.1 Test Problems

We performed experiments on a number of random instances of additively decomposable problems with and without overlap that were generated as described above. All problems were solved using the presented deterministic algorithm so that we could ensure that all algorithms would find the actual global optimum. However, no algorithms were provided any information about the global optimum, the locations of the different subproblems, the order of decomposition, or the overlap.

All problems tested in this paper have the same order of subproblems, $k = 4$. Three values of the overlap parameter were considered, specifically, $o = 0$, $o = 1$, and $o = 2$. To examine scalability of tested algorithms, problems of various size were examined for every value of o, where the size of each problem can be expressed in terms of either the overall number of bits denoted by n or the number of subproblems denoted by m. For each value of k, o, and n, 1000 random problem instances were generated and tested.

4.2 Description of Experiments

Performance of all algorithms is expressed in terms of the number of evaluations until the global optimum has been found because in difficult real-world problems, fitness evaluation is usually the primary source of computational complexity. Furthermore, in all compared algorithms, the computational overhead excluding evaluations can be upper bounded by a low-order polynomial of the number of evaluations (17).

For hBOA, UMDA and GA, for every problem instance the minimum population size to ensure convergence to the optimum in 10 out of 10 independent runs is found using the bisection method. The upper bound on the number of generations for hBOA, UMDA and GA is set according to the existing theory for decomposable problems to grow proportionally to the overall number of bits denoted by n (24); specifically, the number of generations for hBOA is upper bounded by n whereas it is upper bounded by $5n$ for all other algorithms. In GA, common parameter settings are used for crossover and mutation rates; the probability of applying crossover is set to $p_c = 0.6$, whereas the probability of flipping each bit in mutation is set to $p_m = 1/n$. Two standard crossover operators are used in GA: two-point crossover and uniform crossover.

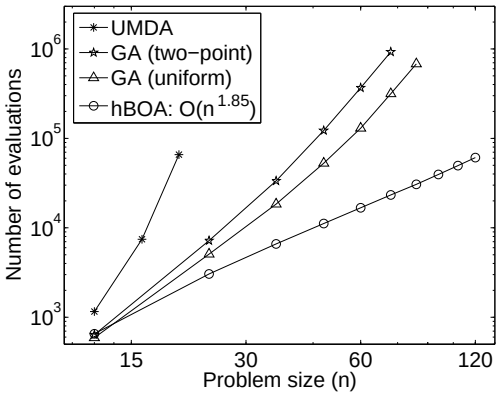
In HC, the only parameter to be set by the user is the probability of flipping each bit in bit-flip mutation. Here we set the mutation rate to the optimum mutation rate for order-k separable problems provided in (14) as $p_m = k/n$. The HC is ran 10 times and each run is terminated when the global optimum is found. The number of evaluations until each run is terminated is then averaged over the 10 runs.

4.3 Results

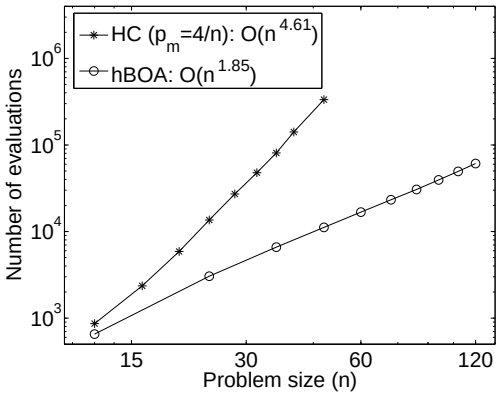
Figure 2 compares the performance of hBOA, GA, UMDA, and HC on random problems with $o = 0$, $o = 1$, and $o = 2$.

Figure 3 visualizes the effects of the overlap parameter o on the performance of hBOA, GA, and HC. We omit the results for UMDA because UMDA could solve only smallest problem instances.

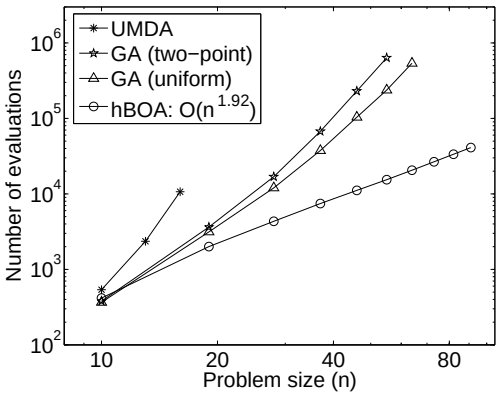
Figure 4 shows how the performance of the two best methods (hBOA and GA with uniform crossover) varies across the class of random decomposable problems by showing not only the results for the entire set of 1000 random problems, but also those for the most difficult 50%, 25%, 12.5%, 6.25%, and 3.125% problem instances (the difficulty is measured by the number of evaluations until convergence).



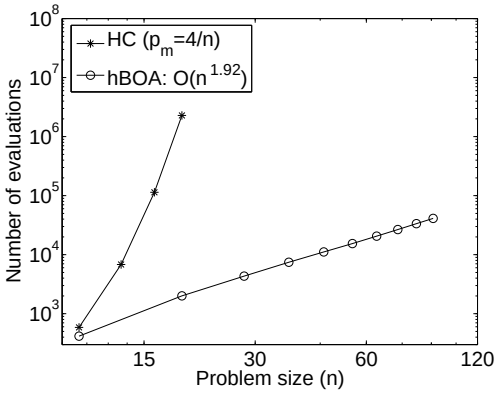
(a) hBOA vs. UMDA and GA for $o = 0$



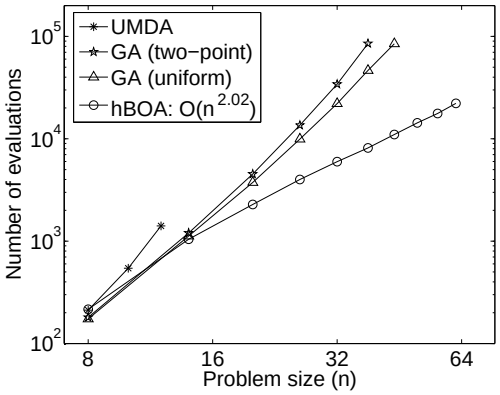
(b) hBOA vs. HC for $o = 0$



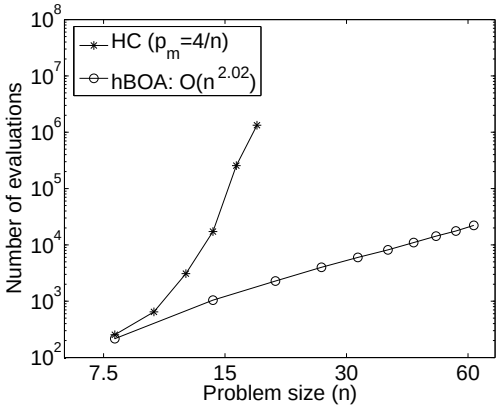
(c) hBOA vs. UMDA and GA for $o = 1$



(d) hBOA vs. HC for $o = 1$

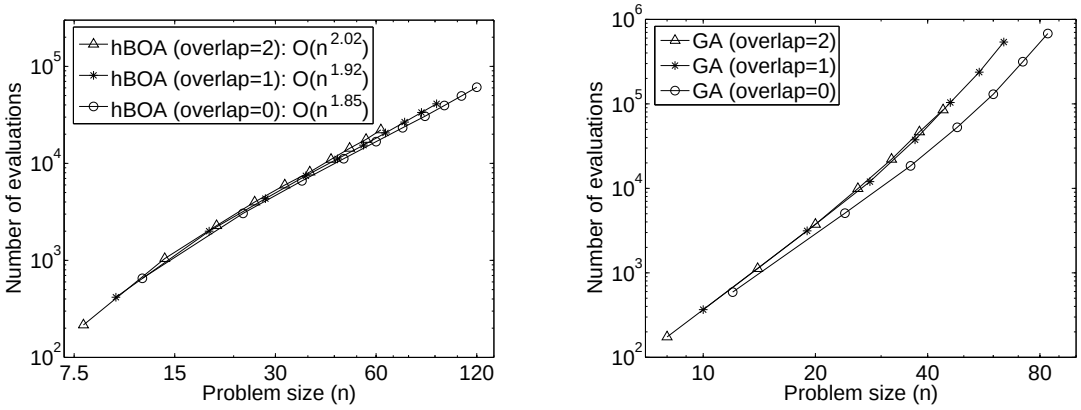


(e) hBOA vs. UMDA and GA for $o = 2$



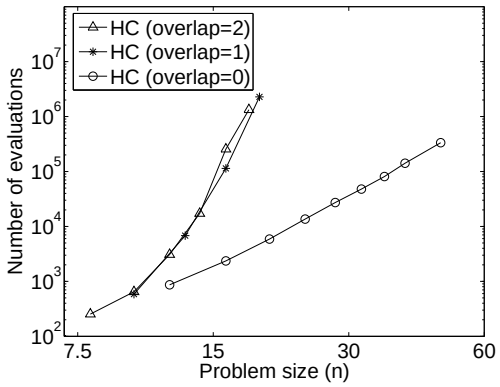
(f) hBOA vs. HC for $o = 2$

Figure 2: Comparison on random decomposable problems.



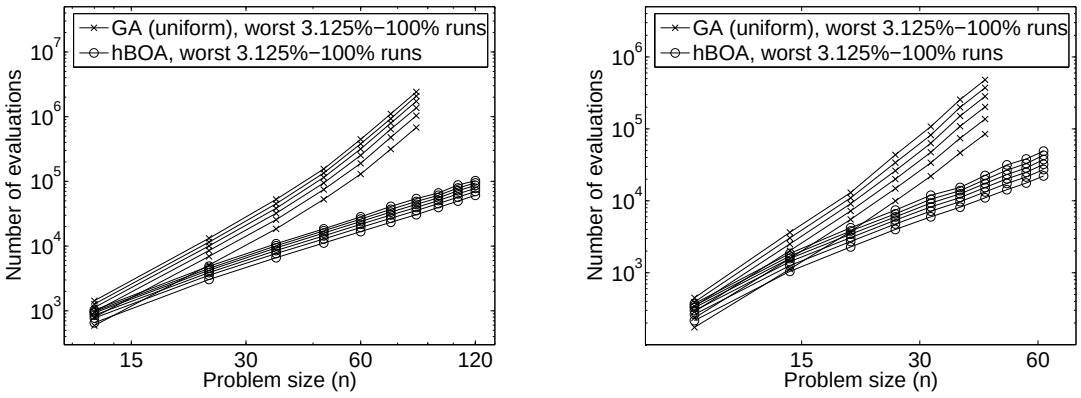
(a) hBOA

(b) GA



(c) HC

Figure 3: The effects of overlap on the performance of hBOA, GA, and HC.



(a) No overlap

(b) Overlap $o = 2$

Figure 4: Performance of hBOA using the worst $100\%, 50\%, 25\%, 12.5\%, 6.25\%$, and 3.125% of results.

4.4 Discussion

The results indicate a low-order polynomial growth of the number of function evaluations required by hBOA to solve random instances of the proposed class of problems for any value of o; specifically, the number of evaluations can be approximately upper bounded by $O(n^{1.85})$ for $o = 0$, $O(n^{1.92})$ for $o = 1$, and $O(n^{2.02})$ for $o = 2$. The low-order polynomial performance can be observed even if one considers only the most difficult problem instances.

On the other hand, the results indicate that GA with standard crossover and mutation operators requires exponential time because it is not capable of combining promising solutions effectively as argued in (23; 8) for deceptive problems; the reason for this behavior is that standard recombination operators can effectively combine only short-order, tight schemata (12; 7) and this may often be insufficient if the short-order, tight schemata do not lead to the optimum. The performance of hBOA, GA and UMDA gets slightly worse with overlap although all algorithms perform similarly for different values of overlap.

Mutation by itself is also inefficient and its performance is much worse than the performance of hBOA even on separable problems where there is no overlap. Overlap further affects mutation, making this operator work much less efficiently even with the overlap of only $o = 1$; the effects of overlap are much stronger for HC than for the recombination-based methods hBOA, GA and UMDA.

5 Future Work

One of the important topics for future work is to examine the influence of k on the performance of hBOA and other evolutionary algorithms. Although existing theory can be used to predict the influence of k on the performance of the compared algorithms (8; 10; 17), experiments should be completed to confirm these theoretical results. Another topic for future work is to examine the distribution of running times for random decomposable problems with and without overlap. Studying the distribution of the population size and the number of generations for various settings of n, k and o can also lead to interesting insights that can allow the design of robust estimates of these parameters for real-world applications. It would also be interesting to analyze decomposable problems generated with other distributions than those studied in this paper. Finally, analogical analyses should be performed on more complex classes of additively decomposable problems, such as NK landscapes and maximum satisfiability.

6 Summary and Conclusions

This paper presented a class of additively decomposable problems with and without overlap, which can be used to test optimization algorithms that attempt to solve nearly decomposable problems of bounded difficulty, such as BOA and hBOA. A method for generating random instances of the proposed class of problems was described and it was shown how the optimum of these problem instances can be verified efficiently.

The paper then applied a number of evolutionary algorithms to random instances of the proposed class of problems; specifically, the algorithms included in the tests were the hierarchical BOA (hBOA), the genetic algorithm (GA) with standard crossover and mutation operators, the univariate marginal distribution algorithm (UMDA), and the hill climbing (HC) with bit-flip mutation. The results showed that the best performance is achieved with hBOA, which can solve all variants of random decomposable problems with only $O(n^{2.02})$ function evaluations or faster. GA, UMDA

and HC perform much worse than hBOA, usually requiring the number of evaluations that appears to grow exponentially fast. The performance of recombination-based methods appears to be much less sensitive to overlap, whereas the performance of mutation-based methods appears to depend strongly on the amount of overlap.

Acknowledgments

This work was supported by the Research Award and the Research Board at the University of Missouri, the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-03-1-0129, the National Science Foundation under ITR grant DMR-99-76550 (at Materials Computation Center), and ITR grant DMR-0121695 (at CPSD), and the Dept. of Energy under grant DEFG02-91ER45439 (at Fredrick Seitz MRL). The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Most experiments were completed at the Beowulf cluster at the University of Missouri–St. Louis. Some experiments presented in this work were done using the hBOA software developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign.

References

[1] D. H. Ackley. An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, pages 170–204, 1987.

[2] F. Barahona, R. Maynard, R. Rammal, and J. Uhry. Morphology of ground states of a two dimensional frustration model. *J. Phys. A*, 15:673, 1982.

[3] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 331–337, 1991.

[4] D. M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. Technical Report MSR-TR-97-07, Microsoft Research, Redmond, WA, 1997.

[5] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. IlliGAL Report No. 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1991.

[6] I. Gent, H. H. Hoos, P. Prosser, and T. Walsh. Morphing: Combining structure and randomness. *Proceedings of the American Association of Artificial Intelligence (AAAI-99)*, pages 654–660, 1999.

[7] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, Reading, MA, 1989.

[8] D. E. Goldberg. *The design of innovation: Lessons from and for competent genetic algorithms*, volume 7 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, 2002.

[9] G. Harik. Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.

[10] G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999.

[11] G. R. Harik. Finding multimodal solutions using restricted tournament selection. *Proceedings of the International Conference on Genetic Algorithms (ICGA-95)*, pages 24–31, 1995.

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

[13] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer, Boston, MA, 2002.

[14] H. Mühlenbein. How genetic algorithms really work: I.Mutation and Hillclimbing. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature*, pages 15–25, Amsterdam, Netherlands, 1992. Elsevier Science.

[15] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, pages 178–187, 1996.

[16] C. H. Papadimitriou. The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, 4:237–244, 1977.

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

[18] M. Pelikan and D. E. Goldberg. Escaping hierarchical traps with competent genetic algorithms. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 511–518, 2001. Also IlliGAL Report No. 2000020.

[19] M. Pelikan and D. E. Goldberg. A hierarchy machine: Learning to optimize from nature and humans. *Complexity*, 8(5):36–45, 2003.

[20] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002. Also IlliGAL Report No. 99018.

[21] M. Pelikan, K. Sastry, and D. E. Goldberg. Scalability of the Bayesian optimization algorithm. *International Journal of Approximate Reasoning*, 31(3):221–258, 2002. Also IlliGAL Report No. 2001029.

[22] S. Santarelli, D. E. Goldberg, and T.-L. Yu. Optimization of a constrained feed network for an antenna array using simple and competent genetic algorithm techniques. *Proceedings of the Workshop Military and Security Application of Evolutionary Computation (MSAEC-2004)*, 2004.

[23] D. Thierens. *Analysis and design of genetic algorithms*. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, 1995.

[24] D. Thierens, D. E. Goldberg, and A. G. Pereira. Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, pages 535–540, 1998.