Performance of Evolutionary Algorithms on Random Decomposable Problems

Martin Pelikan[1], Kumara Sastry[2], Martin V. Butz[3], and David E. Goldberg[2]

[1] Missouri Estimation of Distribution Algorithms Laboratory (MEDAL), 320 CCB; University of Missouri in St. Louis; One University Blvd., St. Louis, MO 63121
pelikan@cs.umsl.edu
[2] Illinois Genetic Algorithms Laboratory (IlliGAL), 107 TB; University of Illinois at Urbana-Champaign; 104 S. Mathews Ave., Urbana, IL 61801
kumara@illigal.ge.uiuc.edu, deg@uiuc.edu
[3] Department of Cognitive Psychology; University of Würzburg; Röntgenring 11, 97070 Würzburg, Germany
mbutz@psychologie.uni-wuerzburg.de

Abstract. This paper describes a class of *random additively decomposable problems* (rADPs) with and without interactions between the subproblems. The paper then tests the hierarchical Bayesian optimization algorithm (hBOA) and other evolutionary algorithms on a large number of random instances of the proposed class of problems. The results show that hBOA can scalably solve rADPs and that it significantly outperforms all other methods included in the comparison. Furthermore, the results provide a number of interesting insights into both the difficulty of a broad class of decomposable problems as well as the sensitivity of various evolutionary algorithms to different sources of problem difficulty. rADPs can be used to test other optimization algorithms.

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 [1,2] 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].
(3) *Testing on real-world problems or their approximations.* For example, the problem of designing military antennas can be considered for testing [4].

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

T.P. Runarsson et al. (Eds.): PPSN IX, LNCS 4193, pp. 788–797, 2006.

(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 large number of random rADP instances. Specifically, we consider the hierarchical Bayesian optimization algorithm (hBOA), genetic algorithms (GAs) with standard variation and mutation operators, the univariate marginal distribution algorithm (UMDA), and the stochastic hill climber (HC). The results show that hBOA significantly outperforms other algorithms included in the comparison. Furthermore, the results provide a number of interesting insights into both the difficulty of decomposable problems as well as the sensitivity of various evolutionary algorithms to different sources of problem difficulty in decomposable problems.

The paper starts by introducing the class of rADPs in section 2. Section 3 presents and discusses experimental results. Finally, section 4 summarizes and concludes the paper.

2 Random Additively Decomposable Problems (rADPs)

This section describes the class of random additively decomposable problems (rADPs) of bounded difficulty with and without overlap. Candidate solutions are 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. The goal is to *maximize* the objective function (fitness function).

The section first presents the general form of additively decomposable problems (ADPs). Then, the section discusses ADPs of bounded order with and without interactions between subproblems. Finally, the section describes how to generate random instances of the proposed class of decomposable problems.

2.1 Additively Decomposable Problems (ADPs)

The fitness function for an n-bit ADP 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) corresponding to 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 variables. The difficulty of ADPs depends on the order of subproblems, the subproblems themselves, and their interaction through string positions contained in multiple subproblems. It is important to note that many difficult problems, including NP-complete problems, can be defined using subproblems of relatively short order that interact in a complex structure, while many easy problems may contain interactions of high order. For example, the problem of finding ground states of Ising spin glasses can be additively decomposed into subproblems of order 2, but the problem is NP-complete as a result of complex interactions between the different subproblems that lead to strong frustration effects [5,6]. On the other hand, if we consider a simple onemax problem (maximize the sum of bits) and reinforce the global optimum by adding 1 to its fitness, any decomposition of the problem must use a subproblem that contains all variables, but for most algorithms the problem is still as easy as the original onemax or even easier.

The remainder of this section defines the proposed class of rADPs, shows how to generate random instances of the described class of problems, and outlines an efficient method to solve these problem instances.

2.2 Defining ADPs with Overlap

Here we describe a class of ADPs where the overlap is relatively simple and the optimum can be verified using an efficient procedure based on dynamic programming. The order of all subproblems is fixed to a constant k and the amount of overlap is specified by a parameter $o \in \{0, 1, \ldots, k-1\}$ called *overlap*.

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, $o = 0$. Other approaches that control overlap in ADPs can be found in references [7,8,9].

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 Fig. 1 to visualize the aforementioned class of decomposable problems.

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. The algorithm iterates

(a) Tight linkage. (b) Loose linkage.

Fig. 1. Examples of ADPs with 4 subproblems of 4 bits each and 1-bit overlap. Each string position (bit) is displayed as a rectangle and the string positions corresponding to one subproblem are filled with the same color. The string positions that are located in more subproblems are split along the diagonal.

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.

2.3 Generating Random Problems

To generate random instances of the class of ADPs defined above, the user must specify the number m of subproblems, the order k of decomposition, and the overlap o. After specifying m, k, and o, for each subproblem f_i, the 2^k values that specify f_i are generated randomly; overall, there are $2^k m$ values to generate. Finally, the permutation of string positions is generated to eliminate the assumption of tight linkage.

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.

3 Experiments

This section presents and discusses experimental results.

3.1 Test Problems

We performed experiments on a number of instances of rADPs 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, the compared algorithms were not provided any information about the global optimum, the locations of the 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 the compared algorithms, problems of various sizes n were examined for every value of o; the number of subproblems can be computed from n and o as $m = (n - o)/(k - o)$. For each combination of values of k, o, and n, 1000 random problem instances were generated and tested.

3.2 Compared Algorithms

The paper considers genetic algorithms (GA) with standard crossover operators, the univariate marginal distribution algorithm (UMDA) [10], stochastic hill climbing (HC), and the hierarchical Bayesian optimization algorithm (hBOA) [11].

In GA, bit-flip mutation and either two-point or uniform crossover operator were used. hBOA and UMDA are estimation of distribution algorithms

(EDAs) [12,10,13,14] where standard variation operators are replaced by building and sampling a probabilistic model of promising solutions; hBOA uses Bayesian networks with local structures to model candidate solutions, whereas UMDA uses a simple univariate model based on single-bit probabilities. In stochastic hill climbing, bit-flip mutation is used as the perturbation operator. In GA, UMDA and hBOA, restricted tournament replacement is used to ensure effective diversity maintenance.

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 [15].

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 [15]. The upper bound on the number of generations for hBOA, UMDA and GA is set according to the existing theory [16]; 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, the probability of applying two-point and uniform crossover is set to $p_c = 0.6$, whereas the probability of flipping each bit in mutation is set to $p_m = 1/n$.

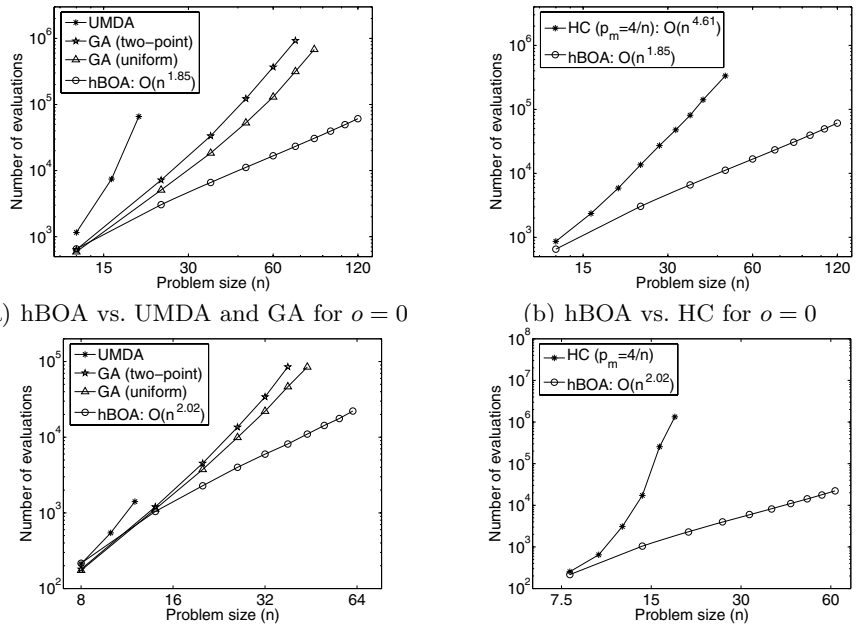
In HC, the only parameter set by the user is the probability of flipping each bit in bit-flip mutation. Here we set the mutation rate to the optimum mutation rate for order-k separable problems provided in [17] as $p_m = k/n$. The HC is ran 10 times and each run is terminated when the global optimum is found.

3.3 Results

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

3.4 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.



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

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

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

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

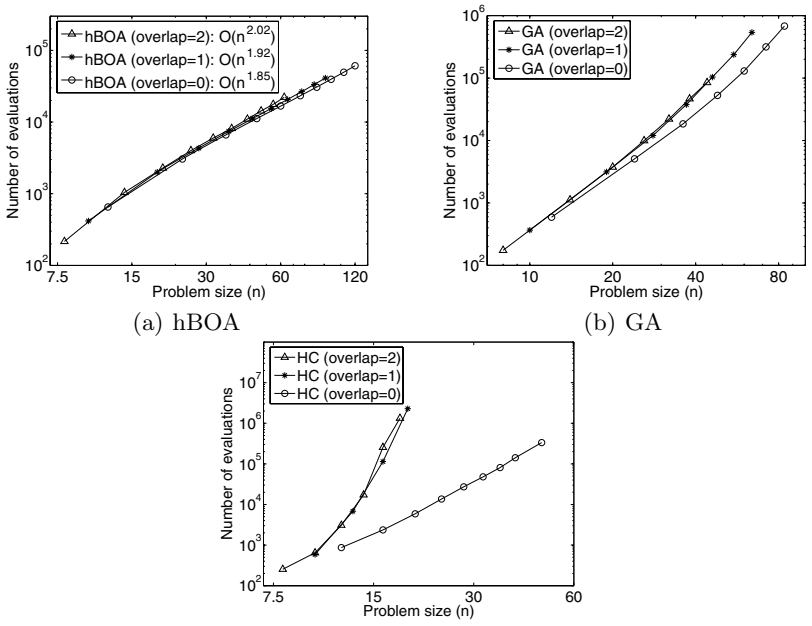
Fig. 2. Comparison on random decomposable problems

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 [18,19] for deceptive problems; the reason for this behavior is that standard recombination operators can effectively combine only short-order, tight schemata [20,21] 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.

4 Summary and Conclusions

This paper presented a class of random additively decomposable problems (rADPs) with and without overlap. The proposed class of problems differs from other comparable problem classes in several important ways. First of all, unlike in concatenated traps, onemax, and many other artificial decomposable test problems, here each subproblem of a specific problem instance is expected to be unique



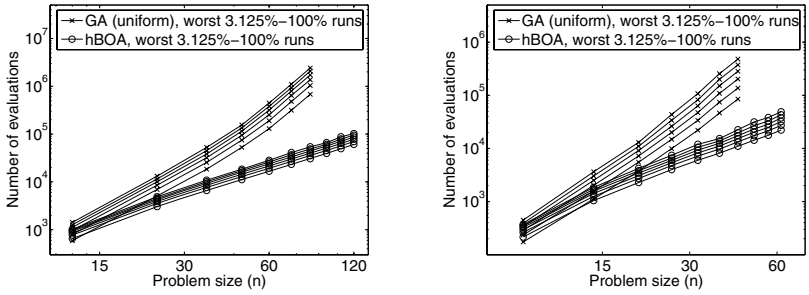
(a) hBOA (b) GA

(c) HC

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

and both the difficulty of subproblems as well as the signal-to-noise ratio are expected to vary from one subproblem to another. Second, unlike in Ising spin glasses, MAXSAT and many other difficult problems, all problem instances are relatively easy to solve given all problem-specific knowledge or effective variation operators that can automatically identify and exploit problem decomposition. Despite that, the generated problems are still difficult enough to make many standard, ineffective variation operators fail. It is also important that in the proposed class of problems, problem difficulty and the order of interactions between subproblems can be controlled in a straightforward manner. Since it is widely believed that many real-world problems are nearly decomposable and the proposed class of problems covers many potential problems of this form, the proposed class of random problems can be used to provide valuable information about the performance of various optimization algorithms in many real-world problems and to design automated methods for setting algorithm-specific parameters in a robust manner.

The paper applied a number of evolutionary algorithms to random instances of the proposed class of problems. Specifically, the paper considered 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 a number of evaluations that appears to grow exponentially fast. The results also provided



(a) No overlap (b) Overlap $o = 2$

Fig. 4. hBOA on most difficult 100%, 50%, 25%, 12.5%, 6.25%, and 3.125% instances

insight into the sensitivity of recombination and mutation operators to overlap in decomposable problems; specifically, recombination-based methods appear to be much less sensitive to overlap than the methods based on local search operators. Although deception is not enforced for any subproblem, linkage learning remains important. Finally, the difficulty of random decomposable problems does not seem to vary much within the same setting of n, k, and o.

The source code of the proposed problem generator and other related functions in ANSI C is provided online at MEDAL web page, `http://medal.cs.umsl.edu/`.

Acknowledgments. This work was supported by the National Science Foundation under CAREER grant ECS-0547013, ITR grant DMR-03-25939 (at Materials Computation Center, UIUC), and ITR grant DMR-0121695 (at CPSD), by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, by the Dept. of Energy under grant DEFG02-91ER45439 (at Frederick Seitz MRL), by the European Commission contract No. FP6-511931, and by the University of Missouri in St. Louis through the High Performance Computing Collaboratory sponsored by Information Technology Services, and the Research Award and Research Board programs. 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. Ackley, D.H.: An empirical study of bit vector function optimization. Genetic Algorithms and Simulated Annealing (1987) 170–204
2. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. IlliGAL Report No. 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL (1991)
3. Cheeseman, P., Kanefsky, B., Taylor, W.M.: Where the really hard problems are. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-91) (1991) 331–337

4. Santarelli, S., Goldberg, D.E., Yu, T.L.: 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)
5. Barahona, F., Maynard, R., Rammal, R., Uhry, J.: Morphology of ground states of a two dimensional frustration model. J. Phys. A **15** (1982) 673
6. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. Theoretical Computer Science **4** (1977) 237–244
7. Gao, Y., Culberson, J.: Space complexity of EDA. Evolutionary Computation **13**(1) (2005) 125–143
8. Gao, Y., Culberson, J.: On the treewidth of NK landscapes. Genetic and Evolutionary Computation Conference (GECCO-2003) **II** (2003) 948–954
9. Weinberger, E.D.: Local properties of kauffman's N-k model: A tunably rugged enegy landscape. Physical Review A **44**(10) (1991) 6399–6413
10. Mühlenbein, H., Paaß, G.: From recombination of genes to the estimation of distributions I. Binary parameters. Parallel Problem Solving from Nature (1996) 178–187
11. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (2001) 511–518 Also IlliGAL Report No. 2000020.
12. Baluja, S.: Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA (1994)
13. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21**(1) (2002) 5–20 Also IlliGAL Report No. 99018.
14. Larrañaga, P., Lozano, J.A., eds.: Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer, Boston, MA (2002)
15. Pelikan, M.: Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms. Springer-Verlag (2005)
16. Thierens, D., Goldberg, D.E., Pereira, A.G.: Domino convergence, drift, and the temporal-salience structure of problems. Proceedings of the International Conference on Evolutionary Computation (ICEC-98) (1998) 535–540
17. Mühlenbein, H.: How genetic algorithms really work: I.Mutation and Hillclimbing. In Männer, R., Manderick, B., eds.: Parallel Problem Solving from Nature, Amsterdam, Netherlands, Elsevier Science (1992) 15–25
18. Thierens, D.: Analysis and design of genetic algorithms. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (1995)
19. Goldberg, D.E.: The design of innovation: Lessons from and for competent genetic algorithms. Volume 7 of Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers (2002)
20. Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, MI (1975)
21. Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading, MA (1989)