Global Optimization of Multimodal Deceptive Functions

David Iclănzan

HEC Lausanne, Quartier UNIL-Dorigny,
Bâtiments Internef, 1015 Lausanne, Switzerland
david.iclanzan@gmail.com

Abstract. Local search algorithms operating in high-dimensional and multimodal search spaces often suffer from getting trapped in a local optima, therefore requiring many restarts. Even with multiple restarts, their search efficiency critically depends on the choice of the neighborhood structure. In this paper we propose an approach in which the need for the restarts is exploited to improve the neighborhood definitions. Namely, a graph clustering based linkage detection method is used to mine the information from several runs, in order to extract variable dependencies and update the neighborhood structure, variation operators accordingly. We show that the adaptive neighborhood structure approach enables the efficient solving of challenging global optimization problems that are both deceptive and multimodal.

1 Introduction

Generic stochastic optimization methods like evolutionary algorithms, swarm algorithms, stochastic hill climbing, simulated and quantum annealing, tabu search etc. are often used for locating in reasonable time a good approximation of the global optimum of hard problems, defined over large search spaces and/or exhibiting multimodality, deceptiveness.

However, these methods are not always applicable as there are problems that are intractable using fixed, problem independent operators and representations [1]. Efficient solving of such problems requires methods that can learn the structure of a problem on the fly and use operators that adapt the linkage between variables. In the case of population based methods the sampling offered by the elements of the population opens the way for inferring valuable statistical information. For these methods, the replacement of operators like crossover and mutation with a repeated selection – model-building – sampling process, enables the required adaptability of the neighborhood structures and adequate performance.

The family of these methods, often called competent evolutionary algorithms [2] include such examples as the fast messy genetic algorithm [3], the linkage learning genetic algorithm [4], the extended compact genetic algorithm [5], the Bayesian Optimization Algorithm (BOA) [6], SEAM [7], DevRep [8], Compact Genetic Codes [9], Hierarchical Genetic Algorithm [10], hBOA [11] and DSGMA++ [12].

C. Blum and G. Ochoa (Eds.): EvoCOP 2014, LNCS 8600, pp. 145–156, 2014.

Incorporating statistical models in trajectory based methods like stochastic hill climbing, simulated and quantum annealing, tabu search etc. is not straightforward as these methods sample just one point of the search space at each step. However, being more lightweight and prone to premature convergence, these methods are used in conjunction with a a random restart mechanism that facilitates a better balance between exploration and exploitation of the search space. Therefore, we apply learning to the results obtained from several runs, extracting linkages and providing properly adapted search operators for future runs.

In this paper we extend the simulated annealing trajectory based stochastic optimizer with a graph clustering based linkage detection method. We investigate the proposed method performance on the concatenated trap function, a problem designed to test a global optimization method's ability to handle deceptiveness and multiomdality.

The next Section presents and discuss some preliminary notions. The proposed method is detailed in Section 3. Section 4 describes the experimental setup, followed by a discussion of the results in Section 5. Finally, the paper concludes in Section 6.

2 Preliminaries

2.1 Simulated Annealing

Simulated annealing is a generic stochastic global optimization metaheuristic [13], inspired from the annealing process, which involves heating and controlled cooling of solids in order to increase the size of their crystals and reduce their defects due to the changes in internal structure .

The simulated annealing algorithm makes an analogy between the thermodynamic free energy and the objective function. In the annealing process, a slower rate of cooling produces a bigger decrease in energy. The simulated slow, controlled cooling, effects the probability that the simulated annealing accepts a worse solution, decreasing it over time.

In the beginning, when the artificial temperature is high the algorithm will often accept solutions that are worse than the current solution, encouraging exploration, facilitating the escape from the basin of attraction of local optima. As the temperature is reduced the probability of accepting worse solutions decreases, focusing the search more on exploitation. The balance provided by the controlled, gradually decreasing artificial cooling enables the simulated annealing algorithm to perform well on large search spaces. The pseudo-code of the method is presented in Alg. 1.

The simulated annealing as many other generic metaheuristics uses a fixed neighborhood structure which can severely hinder its performance in problems that require the discovery and efficient mixing of partial sub-solutions, building-blocks. At each temperature the algorithm performs a number of iterations that is controlled by a positive parameter $nrtries$. The algorithms terminates when it finds a good enough solution, or when there are no improvements over the last $limitNI$ iterations.

Algorithm 1. Simulated annealing

```
1  Define an initially high temperature T;
2  Define a cooling schedule cooling(T);
3  Define an energy function E(state);
   /* Start from a random point                                    */
4  x ← RandomState();
5  while not converged and noimprovementcount < limitNI do
6      for i ← 1 : nrtries do
7          x_new ← neighboor(x);
8          Δ ← E(x_new) − E(x);
9          if Δ < 0 then
10             x ← x_new;
11         else
               /* Evaluate if the worse state is to be accepted    */
12             if rand < e^(−Δ/T) then
13                 x ← x_new;
       /* Decrease the temperature according to the cooling schedule */
14     T ← cooling(T);
```

2.2 Graph Clustering Based Model Building

Dependent variables are detected by applying a maximum flow clustering algorithms to an adjacency matrix which contains the pairwise interactions between variables. Therefore, the method is model evaluation free, fast, scalable and easily parallelizable.

Maximum flow clustering algorithms simulate a special flow within a graph, which promotes flow where the current is strong whilst reducing flow where the current is weak. These procedure reveal the cluster structure within the graph, as the flow across borders between different groups diminish with time, while it increases within the group.

Simulation of flow through a graph is easily done by transforming the adjacency matrix into a column-stochastic square matrix, where each column sums to 1. This matrix, denoted by M, can be interpreted as the matrix of the transition probabilities of a random walk (or a Markov chain) defined on the graph, where $M(j, i)$ represents the probability (stochastic flow) of a transition from vertex v_i to v_j. Flow expansion can be simulated by computing powers of the flow (Markov) matrix M.

One of the most well known and used maximum flow based graph clustering algorithm is the Markov Clustering Algorithm (MCL) [14]. It offers several advantages, like a simple, elegant mathematical formulation, robustness to topological noise [15], support for easy paralellization and adaptation via a simple parameter enables the obtaining of clusters of different granularity.

MCL iteratively simulates random walks within a graph by applying two operators called expansion and inflation, until convergence occurs. At the end of each inflation step a pruning step is also performed, in order to reduce the computational complexity by making M sparse. The MCL process may be regarded as alternative expansion and contraction of the flow in the graph. The expansion step is responsible in spreading the flow out of a vertex to potentially new vertices and with the strengthening of the flow to those vertices which are reachable by multiple paths. This has the effect of enhancing within-cluster flows as there are more paths between the nodes belonging to the same cluster. The inflation operator is responsible for both strengthening intra-cluster flow and weakening inter-cluster flow of current and by this, introducing a non-linearity in the distribution of the flows. At the beginning the flow distribution is relatively smooth and uniform, but with each iteration it becomes more and more peaked. In the end, all the nodes within a tightly-linked group of nodes will start to flow towards one node within the group, forming star sub-graphs associated with the MCL limits.

The idealized Markov Cluster process, consisting just from the expansion and inflation operators is known to converge quadratically in the neighborhood of so called doubly idempotent matrices [14]. In practice, the numbers of epochs until convergence is reported to be nearly always far below 100.

While several types of models can be extracted from the MCL iterants [16], here we choose a simple approach, which extracts the linkage groups from the clusters obtained after convergence of the MCL. Here, the nodes have found one "attractor" node to which all of their flow is directed, corresponding to only one non-zero entry per column in M. Nodes sharing the same "attractor" node are grouped in clusters and their corresponding variables form a linkage group that can be used by the modified simulated annealing algorithm.

3 Extended Simulated Annealing

The extended simulated annealing has two phases that are alternatively applied. In the first phase, several runs of simulated annealing are performed according to the current linkage model and the obtained result are collected in a memory. In the second phase this accumulated search experience is exploited in order to learn which variables are linked and the model is updated accordingly. Then the optimization procedure continues again with the first phase but now using a model that is more suited to the particular optimization problem. The details of these two phases are outlined in Algorithm 2.

The parameters of the algorithm are n - number of variables, n_S - number of simulated annealing runs in the search experience accumulation phase, @*stopping_cond* - stopping criteria which usually places a bound on the quality of the found solution and/or time, number of objective function evaluations.

$R \leftarrow corrcoef(memory)$ returns a matrix R of correlation coefficients calculated from an input matrix *memory* whose columns are variables and whose rows are the bits of the solution found by the particular simulated annealing runs.

Algorithm 2. Extended simulated annealing

```
   Input: n, n_S
   /* Initially each variable is independent                          */
1  M ← 1 : n;
2  while not @stopping_cond do
       /* Phase I - accumulate search experience                      */
3      for i = 1, n_S do
           /* Apply the simulated annealing algorithm                 */
4          Define an initially high temperature T;
5          Define a cooling schedule cooling(T);
6          Define an energy function E(state);
           /* Start from a random point                               */
7          x ← RandomState();
8          while not converged and noimprovementcount < limitNI do
               /* Perturbation according to the model M               */
9              x_new ← neighboor(x, M);
10             Δ ← E(x_new) − E(x);
11             if Δ < 0 then
12                 x ← x_new;
13             else
                   /* Evaluate if the worse state is to be accepted   */
14                 if rand < e^(−Δ/T) then
15                     x ← x_new;
               /* Decrease the temperature                            */
16             T ← cooling(T);
17         memory[i] ← x;
       /* Phase II - update the model                                 */
       /* Calculate correlation coefficients between variables        */
18     R ← corrcoef(memory);
       /* Apply MCL to extract groups of dependent variables          */
19     M ← MCL(R);
```

The matrix $R \leftarrow corrcoef(memory)$ is obtained from the covariance matrix $C \leftarrow cov(memory)$ according to the formula

$$R(i,j) = \frac{C(i,j)}{\sqrt{C(i,i)C(j,j)}} \tag{1}$$

The essence and capabilities of the proposed method lies in that the perturbation for obtaining x_new operates over a linkage model that is continuously adapted. Variables that are found to be connected are always altered together.

In the beginning each variable forms a separate linkage group, for an 8 variable problem, the initial model would be $M = \{[x_1], [x_2], \ldots [x_8]\}$. With this model the method is equivalent to the standard simulated annealing, when generating x_new each variable is perturbed according to a mutation probability p_mut.

However, if from several runs it can be determined for example that the first four, respectively the last two variables are correlated, the model is updated to reflect this knowledge: $M = \{[x_1, x_2, x_3, x_4], [x_5], [x_6], [x_7, x_8]\}$. In the subsequent runs the simulate annealing will perturb states respecting these dependencies, always altering the clustered variables together, thus eliminating sub-solution disruption and facilitating the mixing of these building-blocks.

4 Experiments

Deceptive functions are among the most challenging problems as they exhibit one or more deceitful optima located far away from the global optimum. The basins of attraction of the local-optima are much bigger than the attraction area of the optimal solutions, thus following the objective function gradient is misleading on average.

Order k deceptive trap function or simply k-Trap, is a function of unitation (its value depends only on the numbers of 1's in the input string), based on two parameters f_{high} and f_{low} which define the degree of deception and the fitness signal-to-noise ratio. The input that contains all but ones is maximally rewarded; for the other cases, the fitness of the block is directly proportional to the number of zeros, the string of all zeros being a strong local optima.

Let u be the unitation of the binary input string. The k-trap function is defined as:

$$trap_k(u) = \begin{cases} f_{high} & \text{, if } u = k; \\ f_{low} - \frac{u+1}{k} & \text{, otherwise.} \end{cases} \tag{2}$$

Concatenating m copies of this trap function [17] gives a global additively separable, boundedly deceptive function over binary strings:

$$CTF(x) = -\sum_{i=0}^{m-1} trap_k \left(\sum_{j=ki}^{ki+k-1} x_j \right) \tag{3}$$

In this paper we operate over 120 bit length CTF problems, concatenating $m = 20$ k-Trap functions based on $k = 6$, $f_{high} = f_{low} = 1$. For this setup, the test function have 240 local optima. The global optima, the string of all ones, has a value of -20.

In a first experiment, we tested how the simple simulated annealing performs on this 120 bit CTF function. The performance of a stochastic search method can severely depend on its parametrizations. As there are no definite guidelines on how to achieve the best configuration, we decided to test a comprehensive range of parametrizations over 20000 runs. For each run we randomly determined values for the perturbation probability ($p\_mut \in [0.01, 0.25]$), the number of maximum allowed consecutive rejections ($1000 \leq limitNI \leq 20000$) and the number of tries per temperature ($100 \leq nrtries \leq 2000$).

In the second experiment we tested the performance of the extended method averaged over 100 runs, using a perturbation probability p_mut of 0.05, the allowed maximum number of rejections $limitNI$ was set to 2000 and we performed

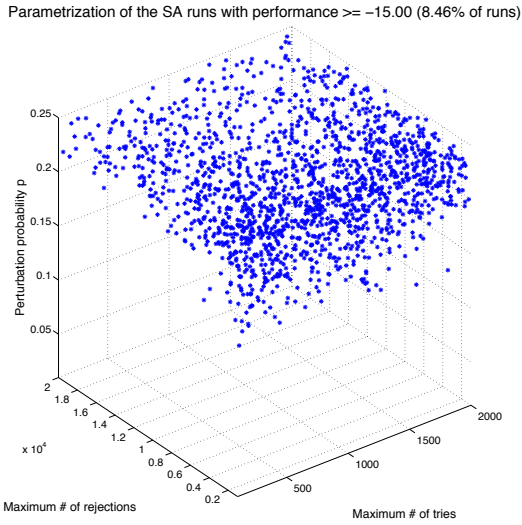

Fig. 1. Parametrizations leading to worst performance of the classical simulated annealing

just on iteration per temperature. In the search experience accumulation phase, the simulated annealing is run $n_S = 100$ times.

For both methods the cooling schedule was set to $T_{k+1} = 0.9T_k$.

5 Results

5.1 Performance of the Classical Simulated Annealing

The simple simulated annealing spent 567980353 functions evaluations in the 20000 runs. Despite the high number of trials, it was never able to find the global optimum, always being mislead by the strong deceptiveness and multimodality of the CTF.

Figure 1 depicts the parameter configurations which resulted in very poor performance for the classical simulated annealing, with objective function values greater than -15. As seen, this corresponds to parametrizations where the mutation probability is very high or both the number of allowed rejections and trials per temperature is low.

The parametrizations leading to the best performances for the classical method are shown in Figure 2. The plots in the parameter space reveal opposite trends, namely that the method requires high values for the number of allowed rejections and trials per temperature. Surprisingly, the values of the perturbation rates cover a wide spectrum, varying between 0.02 and 0.15. Represented by a dot, this figure also shows the best solution found, which had a value of -18.5. This value corresponds to solutions for which the method found the optimal

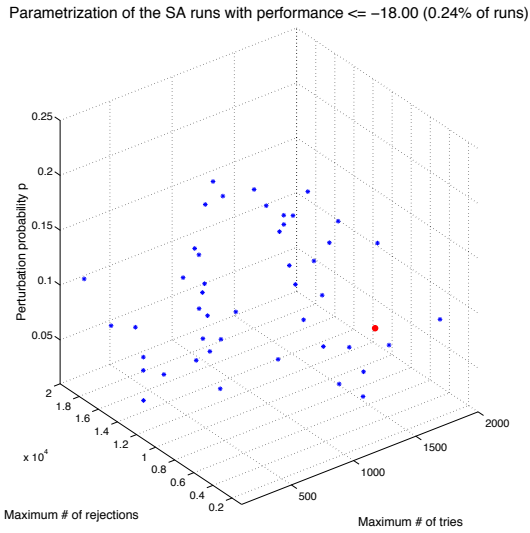

Fig. 2. Parametrizations leading to best performance of the classical simulated annealing, where the method managed to correctly identify 8-11 trap functions out of the 20. The dot denotes the best solution found, corresponding to the -18.5 fitness value.

setting of all ones for 11 k-trap sub-functions out of 20 (discovering 55% of the correct subsolutions) while converging to the deceptive optima of all zeros on the remaining 9 sub-functions.

Figure 3 depicts the histogram of the results obtained from the 20000 runs. With moderate settings for the perturbation probability the classical simulated annealing surpasses the solution of all zeros (which has a value of -16.6666) but is rarely able to resolve - find the optimal setting of all ones for half or more of the $m = 20$ k-trap sub-functions (values of -18.3333 or lower). These empirical results confirm that classical simulated annealing is not a suitable solver for the the CTF as this function requires the efficient mixing of building-blocks with deceptive intra-block gradient.

5.2 Performance of the Extended Simulated Annealing

In contrast, in our second experiment the proposed extended simulated annealing found the global optima in every single run. Figure 4 depicts the juxtaposition of the evolution of these 100 runs, their line style signaling how many learning cycles, model updates they required until convergence to the global optima. The solid lines, accounting for 95% of the cases, corresponds to runs where global optima was found after just one linkage learning phase. The remaining 5% of cases, depicted with dashed lines, corresponds to the cases where 2 or 3 neigborhood structure updates were required before convergence.

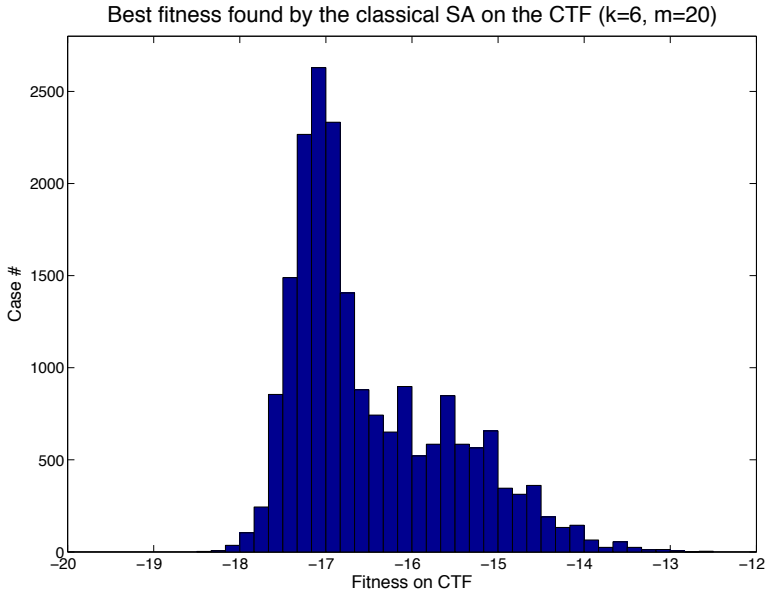

Fig. 3. Histogram of the results obtained from 20000 runs of the classical simulated annealing on CTF with $k = 6$, $m = 20$

In the first 100 epochs, when the model is the trivial one with every variable independently perturbed, the search is equivalent to the classical simulated annealing. Therefore, as expected, the performance is mediocre, the search never reaching the -18 fitness threshold.

After the first 100 runs, when the first round a linkage learning is applied we can see a huge qualitative change in performance. 95% of the time, through epochs 101-200 the search repeatedly founds the global optimum, which has the fitness value -20. This sharp performance gain is due to the adapted neighborhood structure, linkage based perturbation which maximizes the chances of solving k-trap sub-functions while also preventing the disruption of already converged sub-solutions.

However, Figure 4 also reveals that in a few instances, on the same epoch range severely degenerates. This happens in cases when the learnt model is not accurate, containing many spurious, false linkages. Following through epochs 200-400 we can observe that even in this cases, the method recovers each time, eventually learning an accurate model and converging to the global optima. Algorithms using probabilistic models had been long prone to the issue of premature convergence, where the methods can not recover from an inadequate model derived in the early phases from a biased sample due insufficient sampling. As the observed runs suggest, the continuous cycling and transition between exploration and exploitation provided by the simulated annealing search strategy seem to alleviate this problem.

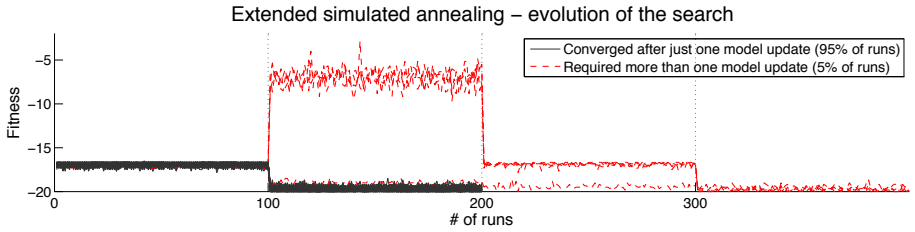

Fig. 4. Juxtaposition of the 100 runs of the extended simulated annealing on CTF with $k = 6$, $m = 20$

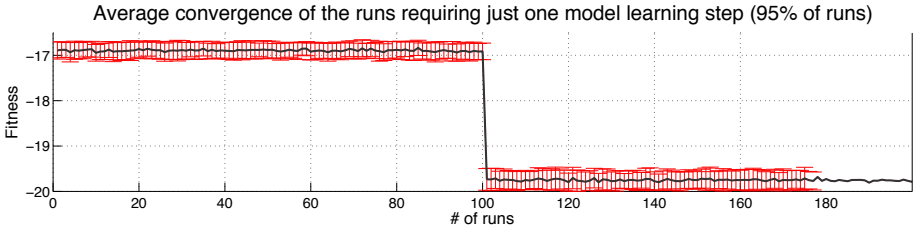

Fig. 5. Evolution of the average performance of the extended simulated annealing on CTF with $k = 6$, $m = 20$. After epoch 100, following the linkage adaptation, we observe a huge qualitative increase in the average performance, while standard deviation remains the same.

For the runs requiring just one linkage learning round, the average convergence rate along with its standard deviation is depicted in Figure 5. Again, the huge gain in performance can be observed immediately after epoch 100, when the neighborhood structure is adapted according to the linkage groups revealed by the graph clustering algorithm. Aside from the greatly shifted average performance, the extended simulated annealing exhibits the same search dynamics (stable, almost constant average, same standard deviation) before and after the neighborhood adaptation.

Figure 6 depicts the required number of objective function evaluations until convergence of the proposed method. The cases above the box are the outliers, where the method required more than one model update due to the spurious linkages incorporated in the first model. The average on the figure is located at 665702.35 evaluations, with a standard deviation of 229118.249553. The quickest convergence took 579841 evaluations while the longest one required 1872947.

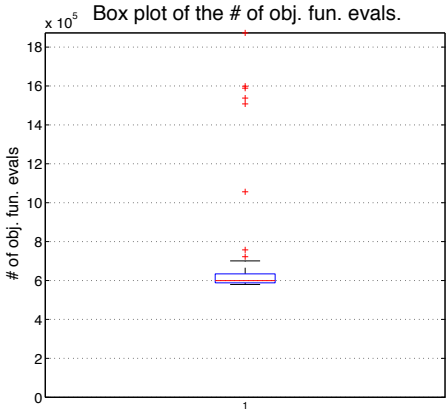

Fig. 6. Box-plot of the required number of CTF evaluations by the extended simulated annealing

6 Conclusions

Problems requiring the discovery and mixing of sub-solutions often can not be solved by problem independent search operators that while optimizing one sub-problem may disrupt already converged sub-solutions.

In this paper we extended the simulated annealing stochastic optimizer with a neighborhood structure adapting mechanism, linkage based perturbation. The linkages, groups of dependent variables are extracted from a correlation coefficient matrix, describing the pairwise dependencies between variables by means of MCL, a maximum flow based graph clustering algorithm. The proposed method showed a very robust behavior, 100% success rate on a problem used to test a method ability to overcome deceptiveness and multimodality. Comprehensive test runs showed that the same problem can not be solved by a simulated annealing using fixed neighborhood structure.

Also, the experiments revealed a very promising phenomena, namely that the extended simulated annealing, by alternating exploration and exploitation phases can recover from erroneous linkage models. This opens the way to study problems where other model based search methods struggle with the issue of premature convergence as they can not reco.

Future work will also experiment with the incorporation of richer, probabilistic models and will address other classes of optimization problems.

Acknowledgments. We acknowledge the support of the Sapientia Institute for Research Programs (KPI).

References

1. Pelikan, M.: Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms. Springer (2005)
2. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Norwell (2002)
3. Goldberg, D.E., Deb, K., Kargupta, H., Harik, G.: Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In: Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo, CA, pp. 56–64. Morgan Kaufman (1993)
4. Harik, G.R., Goldberg, D.E.: Learning linkage. In: Belew, R.K., Vose, M.D. (eds.) FOGA, pp. 247–262. Morgan Kaufmann (1996)
5. Harik, G.R., Lobo, F.G., Goldberg, D.E.: The compact genetic algorithm. IEEE-EC 3(4), 287 (1999)
6. Pelikan, M., Goldberg, D.E., Cantú-Paz, E.: BOA: The Bayesian optimization algorithm. In: Banzhaf, W., et al. (eds.) GECCO 1999, Orlando, FL, July 13-17, vol. I, pp. 525–532. Morgan Kaufmann Publishers, San Fransisco (1999)
7. Watson, R.A., Pollack, J.: A computational model of symbiotic composition in evolutionary transitions. Biosystems 69(2-3), 187–209 (2003), Special Issue on Evolvability, ed. Nehaniv
8. de Jong, E.D.: Representation Development from Pareto-Coevolution. In: Cantú-Paz, E., et al. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 262–273. Springer, Heidelberg (2003)
9. Toussaint, M.: Compact Genetic Codes as a Search Strategy of Evolutionary Processes. In: Wright, A.H., Vose, M.D., De Jong, K.A., Schmitt, L.M. (eds.) FOGA 2005. LNCS, vol. 3469, pp. 75–94. Springer, Heidelberg (2005)
10. de Jong, E.D., Thierens, D., Watson, R.A.: Hierarchical genetic algorithms. In: Yao, X., et al. (eds.) PPSN VIII. LNCS, vol. 3242, pp. 232–241. Springer, Heidelberg (2004)
11. Pelikan, M., Goldberg, D.E.: Escaping hierarchical traps with competent genetic algorithms. In: Spector, L., et al. (eds.) GECCO 2001, July 7-11, pp. 511–518. Morgan Kaufmann, San Francisco (2001)
12. Yu, T.L., Goldberg, D.E.: Conquering hierarchical difficulty by explicit chunking: substructural chromosome compression. In: GECCO 2006, pp. 1385–1392. ACM Press, NY (2006)
13. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
14. van Dongen, S.: Graph Clustering by Flow Simulation. PhD thesis, U. of Utrecht (2000)
15. Brohée, S., van Helden, J.: Evaluation of clustering algorithms for protein-protein interaction networks. BMC Bioinformatics 7, 488 (2006)
16. Iclănzan, D., Dumitrescu, D.: Graph clustering based model building. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 506–515. Springer, Heidelberg (2010)
17. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In: Whitley, L.D. (ed.) Foundations of Genetic Algorithms 2, San Mateo, pp. 93–108. Morgan Kaufmann (1993)