# Learning the Neighborhood with the Linkage Tree Genetic Algorithm

Dirk Thierens[1,2] and Peter A.N. Bosman[2]

[1] Institute of Information and Computing Sciences
Universiteit Utrecht, The Netherlands
[2] Centrum Wiskunde & Informatica (CWI)
Amsterdam, The Netherlands

**Abstract.** We discuss the use of online learning of the local search neighborhood. Specifically, we consider the *Linkage Tree Genetic Algorithm (LTGA)*, a population-based, stochastic local search algorithm that learns the neighborhood by identifying the problem variables that have a high mutual information in a population of good solutions. The LTGA builds each generation a *linkage tree* using a hierarchical clustering algorithm. This bottom-up hierarchical clustering is computationally very efficient and runs in $O(n^2)$. Each node in the tree represents a specific cluster of problem variables. When generating new solutions, these linked variables specify the neighborhood where the LTGA searches for better solutions by sampling values for these problem variables from the current population. To demonstrate the use of learning the neighborhood we experimentally compare iterated local search (ILS) with the LTGA on a hard discrete problem, the nearest-neighbor NK-landscape problem with maximal overlap. Results show that the LTGA is significantly superior to the ILS, proving that learning the neighborhood during the search can lead to a considerable gain in search performance.

## 1 Introduction

Stochastic local search (SLS) is a powerful class of algorithms to solve discrete optimization problems. Although many variations of SLS can be found in the literature, all of them assume that the neighborhood to be explored is given before the search starts. However, the choice of the neighborhood is a critical factor which can make all the difference between success and failure of the search. The limitations of a fixed neighborhood are sometimes recognized, and this has led to the development of algorithms like variable neighborhood search (VNS) [3], where a nested set of neighborhoods is explored when the search stagnates in its current neighborhood. However, VNS has still a static, predefined neighborhood structure. In this paper, we show that it can be beneficial to search in a dynamically changing neighborhood structure. Our aim is to learn what neighborhood to use during the search. This neighborhood learning is guided by structural similarities present in a set of good solutions found so far during the search process. In the next section we briefly review the linkage tree genetic algorithm

that precisely does this. Section 3 specifies our benchmark function, the so called nearest-neighbor NK-landscape problem with maximal overlap. In Section 4 we compare the LTGA with iterated local search to see how much can be gained from dynamically learning the neighborhood to explore.

## 2 Linkage Tree Genetic Algorithm

The linkage tree genetic algorithm [7, 8, 1] is a population-based, stochastic local search algorithm that aims to identify which variables should be treated as a dependent set during the exploration phase. The LTGA represents this dependence information in a hierarchical cluster tree of the problem variables. The linkage tree of a population of solutions is the hierarchical cluster tree of the problem variables using an agglomerative hierarchical clustering algorithm. The problem variables - or cluster of variables - that are most similar are merged first. Similarity is measured by the mutual information between individual variables, or by the average linkage clustering between clusters of variables $X_{F^i} and X_{F^j}$. This average linkage clustering is equal to the unweighted pair group method with a arithmetic mean (UPGMA) and is defined by:

$$I^{UPGMA}(X_{F^i}, X_{F^j}) = \frac{1}{|X_{F^i}||X_{F^j}|} \sum_{X \in X_{F^i}} \sum_{Y \in X_{F^j}} I(X, Y).$$

This agglomerative hierarchical clustering algorithm is computationally very efficient. Only the mutual information between pairs of variables needs to be computed which is a $O(\ell^2)$ operation. The bottom-up hierarchical clustering can also be done in $O(\ell^2)$ computation by using the *reciprocal nearest neighbor chain* algorithm [2]. For a problem of length $\ell$ the linkage tree has $\ell$ leaf nodes (the clusters having a single problem variable) and $\ell - 1$ internal nodes. Each node divides the set of problem variables into two mutually exclusive subsets. One subset is the cluster of variables at that node, while the other subset is the complementary set of problem variables. The LTGA uses this division of the problem variables as a set of variables whose values are sampled simultaneously. Sample values are obtained by taking a random solution of the current population and copying the corresponding variable values. Each generation the LTGA builds a linkage tree of the current population. New solutions are generated by greedily exploring the neighborhood of each solution in the current population. This neighborhood is defined by the linkage tree. The LTGA traverses the tree in the opposite order of the merging of the clusters by the hierarchical clustering algorithm. Therefore, LTGA first samples the variables which are the least dependent on each other. New solutions are only accepted when they have a better fitness value than the original solution. When the tree is completely traversed, the solution obtained is copied to the population of the next generation. This tree traversal process is done for each solution in the current generation.

## 3 Nearest-neighbor NK-landscape with maximal overlap

To show that learning the neighborhood during the search can be very useful, we compare ILS and LTGA on the nearest-neighbor NK-landscape problem with maximum overlap [5, 6]. The nearest-neighbor NK-problem is a subclass of the general NK-landscape problem where the interacting bit variables are restricted to groups of variables of length $k$. The fitness contributions of each group of $k$-bits are specified in a table of $2^k$ random numbers between 0 and 1. Formally, the nearest-neighbor NK-problem is specified by its length $\ell$, the size of the subproblems $k$, the amount of overlap between the subproblems $o$, and the number of subproblems $m$. The first subproblem is defined at the first $k$ string positions. The second subproblem is defined at the last $o$ positions of the first subproblem and the next $(k-o)$ positions. All remaining subproblems are defined in a similar way. The relationship between the problem parameters is $\ell = k+(m-1)(k-o)$. In this paper, we look at NK-problems with maximal overlap between the groups of interacting variables, thus $o = k-1$. For instance, for $\ell = 100$, $k = 5$, $0 = 4$, and $m = 96$ we get the subproblems: (0 1 2 3 4) (1 2 3 4 5) (2 3 4 5 6) ... (95 96 97 98 99) (96 97 98 99 100). An interesting property of the nearest-neighbor NK-problem is that we can compute the global optimal solution using dynamic programming when we use the structural knowledge of the position of the subfunctions. However, when used as benchmark function for ILS and LTGA the position of the bit variables are randomly shuffled and unknown to these algorithms. ILS has no mechanism to figure out what bits are possibly related. The LTGA however is be able to learn some important interactions and take advantage of this during the search process.

## 4 Experiments

The neighborhood explored by ILS is defined by single bit flips of the current solution. ILS alternates local search with stochastic perturbation of the current local optimum. When after perturbation and subsequent local search a better local optimum is found the search will proceed from that new solution. If not, the newly found solution is rejected and ILS continues from the old local optimum. The perturbation should be large enough such that the local search does not return to the same local optimum in the next iteration. However the perturbation should not be too large, otherwise the search characteristics will resemble those of a multi-start local search algorithm. To investigate the impact of the perturbation size $p_m$ of ILS on a large set of NK-problem instances we first run a series of experiments on 100 randomly generated problems. The top subfigures from Figure 1 show results for perturbation sizes varying from 2 to 10 (left), and from 5 to 20 (right). The Y-axis shows the number of runs (out of 100) that successfully found the global optimal solution to a randomly generated NK-problem instance. The X-axis shows the number of function evaluations needed. The plots are cumulative thus a value (10000, 20) means that 20 of the 100 runs have found the global optimum in less (or equal) than 10000 function evaluation [4].

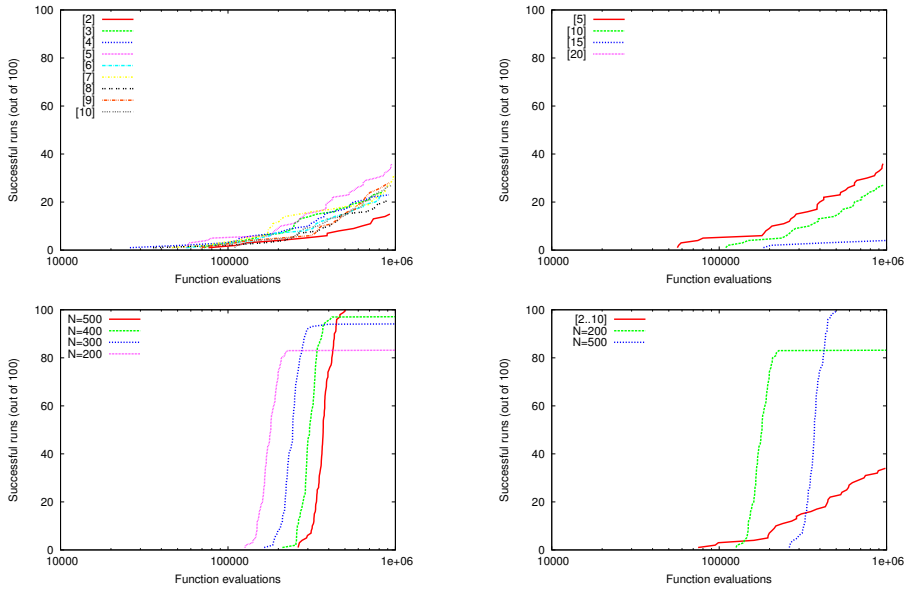


**Fig. 1.** RLDs of single run on 100 NK-problem instances for ILS with perturbation sizes varying from 2 to 20 and LTGA with population sizes varying from 200 to 500.

Whereas in Figure 1 single independent runs are performed on 100 different random NK-problem instance, Figure 2 shows results for 100 independent runs on 10 specific NK-problem instances. The top left subfigure shows ILS with perturbation size randomly chosen between 2 and 10. The top right subfigure shows results for LTGA with population size 500. The two bottom subfigures show results for ILS (perturbation size between 2 and 10) and LTGA (population size 300 and 500) for the $NK_3$ and $NK_2$ problem instances, which are respectively the easiest and second easiest problem to solve.

### 4.1 Discussion

A number of observations can be made from the experiments:

- The top subfigures of Figure 1 show that the perturbation size for ILS does not seem to be a very sensitive parameter for the NK-problems considered. For $p_m = 2$ the success rate is a bit lower than for values between 3 and 10. Closer inspection of the actual runs shows that this is due to the much higher percentage of local searches that return to the starting solution after the perturbation. Performance of ILS only drops when $p_m$ becomes larger than 10. For $p_m = 20$ none of the 100 runs did find the global solution, showing that multi-start local search is hopelessly inefficient for these nearest-neighbor NK landscape problems with maximal overlap.

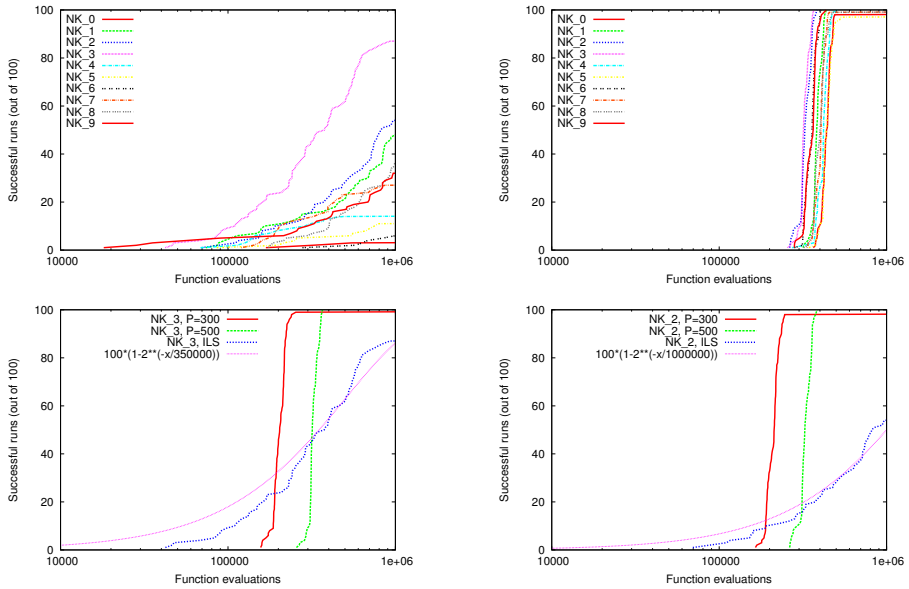


**Fig. 2.** RLDs of 100 runs on 10 NK-problem instances for ILS with random perturbation sizes between 2 and 10, and LTGA with population size 500.

- In the bottom right subfigure of Figure 1 we have also plotted the results for ILS with a perturbation size randomly chosen from the interval [2..10]. Each time ILS perturbs a local optimum, $p_m$ is randomly selected. This way a robust parameter setting is obtained that returns good overall performance.
- LTGA finds the global optima very reliably when the population size $N = 500$. Population sizes 300 and 400 result in a success rate well above 90%, while for $N = 200$ the success rate becomes 83%. Of course a smaller population size means that we need less function evaluations to reach a certain level of performance. For $N = 200$ we could ideally cut the search after 200000 evaluations and then restart the search. This would result in a combined success rate of $1 - (1 - 0.83)^2 = 97\%$ after 400000 function evaluations. Interestingly, this is about the same performance and efficiency result as with a population size $N = 500$, so here there is no gain in restarting the LTGA using a smaller population.
- The top subfigures of 2 show the success rate (out of 100 runs) of ILS ($p_m = \texttt{random}[2..10]$) and LTGA ($N = 500$) for 10 different NK-landscape problems. Clearly, the LTGA is not only more performant and more efficient, it is also more consistent. There is much less variance in the required function evaluations to reach a certain performance level.
- ILS's result varies significantly with the specific instances of the NK-landscape problem. The bottom subfigures of 2 compare the results on the two easiest instances for ILS with the LTGA. We have also drawn a tangent cumulative

exponential distribution function to ILS's success rate. In both cases the success rate of ILS is not dropping lower than the cumulative exponential - rather on the contrary - showing that here also nothing can be gained from restarting ILS after a certain number of function evaluations.

## 5 Conclusions

We have shown that learning a neighborhood structure during the search process can be very beneficial in terms of performance, efficiency, and consistency. To do so, we have compared iterated local search (ILS) using optimally tuned perturbation sizes with the linkage tree genetic algorithm (LTGA) on a set of nearest-neighbor NK-landscape problems with maximal overlap. The LTGA builds each generation a linkage tree using a hierarchical clustering algorithm. Each node in the tree represents a specific groups of problem variables that are linked together. When generating new solutions, these linked variables specify the neighborhood where the LTGA searches for better solutions by sampling values for these problem variables from the current population. In future work, we plan to investigate the use of neighborhood learning for other problem types.

## References


1. P. A. N. Bosman and D. Thierens. The roles of local search, model building and optimal mixing in evolutionary algorithms from a BBO perspective. In N. Krasnogor and P. L. Lanzi, editors, *GECCO (Companion)*, pages 663–670. ACM, 2011.
2. I. Gronau and S. Moran. Optimal implementations of UPGMA and other common clustering algorithms. *Inf. Process. Lett.*, 104(6):205–210, 2007.
3. P. Hansen, N. Mladenovic, and J. A. Moreno-Pérez. Variable neighbourhood search: methods and applications. *Annals OR*, 175(1):367–407, 2010.
4. H. H. Hoos and T. Stützle. Evaluating Las Vegas algorithms: pitfalls and remedies. In *Proc. of the 14th UAI*, pages 238–245. Morgan Kaufmann, 1998.
5. M. Pelikan, K. Sastry, M. V. Butz, and D. E. Goldberg. Performance of evolutionary algorithms on random decomposable problems. In T. P. Runarsson et al., editors, *PPSN*, volume 4193 of *LNCS*, pages 788–797. Springer, 2006.
6. M. Pelikan, K. Sastry, D. E. Goldberg, M. V. Butz, and M. Hauschild. Performance of evolutionary algorithms on NK landscapes with nearest neighbor interactions and tunable overlap. In G. Raidl, editor, *GECCO*, pages 851–858. ACM, 2009.
7. D. Thierens. The linkage tree genetic algorithm. In R. Schaefer et al., editors, *Parallel Problem Solving from Nature*, pages 264–273, Berlin, 2010. Springer.
8. D. Thierens and P. A. N. Bosman. Optimal mixing evolutionary algorithms. In N. Krasnogor and P. L. Lanzi, editors, *GECCO*, pages 617–624. ACM, 2011.