Missouri Estimation of Distribution Algorithms Laboratory

Network crossover performance on NK landscapes and deceptive problems

Mark Hauschild and Martin Pelikan

MEDAL Report No. 2010003

January 2010

Abstract

By being able to build and sample models of promising solutions, Estimation of Distribution Algorithms (EDAs) gain many advantages over standard evolutionary algorithms. However, this comes at the cost of an expensive model-building phase. One way to try and eliminate this bottleneck is to incorporate prior knowledge into the model-building, but this can be a difficult task. Another approach is to modify the crossover operator in a standard genetic algorithm(GA) to exploit this prior information. This paper describes a method to build a network crossover operator that can be used in a GA to easily incorporate problem-specific knowledge such that it better respects linkages between bits. The performance of this operator in the simple genetic algorithm(GA) is then compared to other operators as well as the hierarchical Bayesian Optimization Algorithm (hBOA) on several different problem types, all with both elitism replacement and Restricted Tournament Replacement (RTR). The performance of all the algorithms and replacement mechanisms are then analyzed and the results are discussed.

Keywords

Genetic Algorithms, Crossover, Replacement, Hierarchical BOA, efficiency enhancement, model structure, model complexity, estimation of distribution algorithms

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/

Network crossover performance on NK landscapes and deceptive problems

Mark Hauschild
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
Dept. of Math and Computer Science, 320 CCB
University of Missouri at St. Louis
One University Blvd., St. Louis, MO 63121
mwh308@umsl.edu

Martin Pelikan
Missouri Estimation of Distribution Algorithms Laboratory (MEDAL)
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

Abstract

Practitioners often have some information about the problem being solved, which may be represented as a graph of dependencies or correlations between problem variables. Similar information can also be obtained automatically, for example by mining the probabilistic models obtained by EDAs or by using other methods for linkage learning. This information can be used to bias variation operators, be it in EDAs (where it can be used to speed up model building) or in GAs (where the linkages can be explored by modifying crossover). This can allow us to solve problems unsolvable with conventional, problem-independent variation operators, or speed up adaptive operators such as those of EDAs. This paper describes a method to build a network crossover operator that can be used in a GA to easily incorporate problem-specific knowledge. The performance of this operator in the simple genetic algorithm(GA) is then compared to other operators as well as the hierarchical Bayesian Optimization Algorithm (hBOA) on several different problem types, all with both elitism replacement and Restricted Tournament Replacement (RTR). The performance of all the algorithms are then analyzed and the results are discussed.

Keywords: Genetic Algorithms, Crossover, Replacement, Hierarchical BOA, efficiency enhancement, model structure, model complexity, estimation of distribution algorithms.

1 Introduction

In order for genetic algorithms (GAs) to solve problems robustly and scalably, their operators must respect the linkage between bits (Goldberg, 2002). Unfortunately, conventional variation operators of genetic algorithms often will break up these linkages(Goldberg, 2002). One of the solutions to this problem is to design competent GAs that incorporate linkage learning. Some of the most powerful of these types of algorithms are estimation of distribution algorithms (EDA) (Baluja, 1994; Mühlenbein & Paaß, 1996; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002). EDAs

work by building a probabilistic model of promising solutions and sampling new candidate solutions from the built model. While EDAs have many advantages over standard GAs (Larrañaga & Lozano, 2002; Pelikan, Sastry, & Cantú-Paz, 2006), the model building is often very computationally intensive, so much so that much of the work of EDAs has focused on efficiency enhancements such as parallelization (Cantú-Paz, 2000; Sastry, Goldberg, & Llorà, 2007), learning from experience (Schwarz & Ocenasek, 2000; Baluja, 2006; Hauschild, Pelikan, Sastry, & Goldberg, 2008), or sporadic and incremental model building (Ocenasek & Schwarz, 2000; Pelikan, 2005; Pelikan, Sastry, & Goldberg, 2006; Etxeberria & Larrañaga, 1999; Pelikan, Sastry, & Goldberg, 2008) to improve model building speed.

However, often we have prior information about the problem, which can help us speed up EDAs or directly bias EA variation operators. For example, in graph-based problems we are implicitly given a guide to the strongest dependencies internal to a problem. In addition, EDAs themselves leave us with a tremendous amount of structural information, which may in turn be used to bias operators.

The important question is how best to exploit this information to speed up problem-solving. One way is to bias the model search in EDAs towards some user-specified network model (Schwarz & Ocenasek, 2000; Mühlenbein & Mahnig, 2002) or a network model generated from trial runs of an EDA (Hauschild & Pelikan, 2009). However, this can be difficult, as if this information is not fully correct it can often lead to the algorithm being unable to solve the problem. An alternative method to take advantage of structural information given to us about a problem is to modify the crossover operator itself in a genetic algorithm to better respect the important linkages in the underlying problem structure. This paper discusses a network crossover operator that works with a user-specified network graph to determine which bits are exchanged in the crossover operation. In addition, due to not requiring a costly model building phase, it might be able to outperform EDAs.

In this paper we study the effects of our network crossover operator against uniform and two-point crossover on several test problems known to be hard for standard evolutionary algorithms. In addition, the proposed algorithm is compared against the hierarchical Bayesian Optimization Algorithm (hBOA) (Pelikan & Goldberg, 2001; Pelikan & Goldberg, 2003; Pelikan, 2005), one of the most powerful EDAs currently available.

The paper is organized as follows. Section 2 describes the network crossover operator used in this paper. Section 3 outlines the algorithms tested. Section 4 describes the test problems used in this paper. Section 5 presents the experimental results. Finally, section 6 summarizes and concludes the paper.

2 Network Crossover

While the uniform crossover operator is effective at solving many problems (Goldberg, 1989), it can too heavily disrupt problems that have many bits which are tightly linked (Thierens, 1999). The operator in this paper tries to solve this problem by creating a special crossover that better respects the linkages in the underlying problem structure.

Any two-parent crossover operator starts by creating a mask, which defines what bits to exchange and what bits to keep the same. For example, in the GA uniform crossover operator, bits are exchanged between two parents at some specified exchange rate to create a new candidate solution. Explained another way, for each uniform crossover operation we could generate a crossover mask of n bits, such that if the k^{th} bit is a 1, the child's k^{th} bit is set to the first parent, otherwise the child's k^{th} bit is set to the second parents k^{th} bit. The probability of any particular bit being 1

is usually set to 50%. In our case, we we want to create a specialized mask that respects problem structure.

The network crossover first requires a $n \times n$ matrix G that specifies the strongest connections between bits. While this does require information from the practitioner, it does not require in-depth knowledge of the strength of these interactions. For graph problems, such as graph-bipartitioning or graph coloring, this graph G is inherent to the problem definition. For problems like MAXSAT and Ising Spin glasses, it is also easy to specify such a structure. For those problems without an implicit graph, it is possible to run an EDA on trial instances of the problem to learn a network structure.

Given this network G, it is then possible to build a crossover mask m quite easily. First a random bit i is selected. This bit is then added to the crossover mask by setting $m_i = 1$. Then a randomized breadth-first search is performed on the network G, setting each corresponding bit in m to 1, until m reaches the desired size. If the breadth-first search ends before the desired size is reached, an additional random starting point is selected and the process repeated.

More formally, assuming we have a $n \times n$ network G and an empty crossover mask vector M:

1. While $|M| \leq n/2$
   (a) Select a starting point p.
   (b) Let vector $C = p$.
   (c) Repeat
      i. Set C to the set of all x, such that $G_{i \in M, x} = 1$
      ii. While $C \neq \emptyset$ AND $|M| \leq n/2$
         A. Remove an element i randomly from C.
         B. Add i to M.
   (d) Until $C \equiv \emptyset$ or $|M| > n/2$

The end result is a crossover mask that should most often disrupt bits that are separated the furthest from each other in G.

The key point to remember when constructing G is that it is not necessary to specify all the problem structure, but only the strongest linkages. For most problems, especially graph based problems, the network G can be constructed by connecting only those bits that have a edge between them. For many other classes of problems, it is also quite trivial to construct. For example, in the NK landscape instances in this work, if the bits i and j are neighbors (have a direct interaction), then $G_{i,j} = 1$. Otherwise, $G_{i,j} = 0$. For Trap-5, $G_{i,j} = 1$ if bits i and j are in the same trap partition.

The idea of using a network to modify recombination operators is not new and has inspired other work in the past (Drezner & Salhi, 2002; Drezner, 2003; Stonedahl, Rand, & Wilensky, 2008). Our version of a crossover operator modified by network information most closely resembles the work by Stonedahl (Stonedahl, Rand, & Wilensky, 2008). In Stonedahl's work the crossover mask was built using a random walk through the network structure. In our work we use a breadth first search to emphasize the short range dependencies, since these dependencies were found to be strongest in prior work (Hauschild, Pelikan, Sastry, & Goldberg, 2008). However, our work was not an attempt to improve on this operator, but rather to compare our operator to other standard GA operators and a powerful EDA.

3 Tested Algorithms

The genetic algorithm(GA) (Holland, 1975; Goldberg, 1989) evolves a population of candidate solutions typically represented by binary strings of fixed length. The starting population is generated at random according to a uniform distribution over all binary strings. Each iteration starts by selecting promising solutions from the current population; in this work we use binary tournament selection without replacement. New solutions are created by applying variation operators to the population of selected solutions. These new candidate solutions are then incorporated into the population using a replacement operator. The run is terminated when some termination criteria has been met. In this work we terminate each run either when the global optimum has been found or when a maximum number of iterations have been reached.

The most common variation operators are crossover and bit-flip mutation (Goldberg, 1989). In this work we use three different crossover operators, both that take two candidate solutions and generate two new candidate solutions. The first, network crossover, is explained in the previous section. Uniform and two-point crossover are also used.

The hierarchical Bayesian optimization algorithm (hBOA) (Pelikan & Goldberg, 2001; Pelikan & Goldberg, 2003; Pelikan, 2005) is an EDA that uses Bayesian networks to represent the probabilistic model and incorporates restricted tournament replacement (Harik, 1995) for effective diversity maintenance. Each generation, hBOA evolves a population of candidate solutions represented by fixed-length strings over a finite alphabet (for example, binary strings). The initial population is generated at random according to the uniform distribution over the set of all potential solutions. Each iteration (generation) starts by selecting promising solutions from the current population using any standard selection method of genetic and evolutionary algorithms.

After selecting the promising solutions, hBOA uses these solutions to automatically learn both the structure (edges) as well as the parameters (conditional probabilities) of the model. In this paper, a greedy algorithm is used to learn the structure of BNs with local structures (Pelikan, 2005). To evaluate structures, the Bayesian-Dirichlet metric with likelihood equivalence for BNs with local structures (Chickering, Heckerman, & Meek, 1997) is used with an additional penalty for model complexity (Friedman & Goldszmidt, 1999; Pelikan, 2005).

The next iteration is executed unless some predefined termination criteria are met. For example, the run can be terminated when the maximum number of generations is reached or the entire population consists of copies of the same candidate solution. For more details about the basic hBOA procedure, see Pelikan and Goldberg (2001) or Pelikan (2005).

3.1 Deterministic Hill-Climber

For both GA and hBOA runs, a deterministic hill climber(DHC) was incorporated to improve performance. DHC takes a candidate solution represented by a n-bit binary string and performs one-bit changes on the solution that lead to the maximum improvement. This process is terminated when no possible single-bit flip improves solution quality. Initially experiments without DHC were considered, but for all settings tested, DHC dramatically improved performance.

4 Test Problems

This section describes the test problems considered in this paper.

4.1 Trap-5: Concatenated 5-bit trap

In trap-5 (Ackley, 1987; Deb & Goldberg, 1991), the input string is first partitioned into independent groups of 5 bits each. This partitioning is unknown to the algorithm and it does not change during the run. A 5-bit fully deceptive trap function is applied to each group of 5 bits and the contributions of all trap functions are added together to form the fitness. The contribution of each group of 5 bits is computed as

$$trap_5(u) = \begin{cases} 5 & \text{if } u = 5 \\ 4 - u & \text{otherwise} \end{cases}, \tag{1}$$

where u is the number of 1s in the input string of 5 bits. The task is to maximize the function. An n-bit trap5 function has one global optimum in the string of all 1s and $(2^{n/5} - 1)$ other local optima. Traps of order 5 necessitate that all bits in each group are treated together, because statistics of lower order are misleading.

4.2 NK Landscapes

An NK fitness landscape (Kauffman, 1989; Kauffman, 1993) is fully defined by the following components: (1) The number of bits, n, (2) the number of neighbors per bit, k, (3) a set of k neighbors $\prod(X_i)$ for the i-th bit, X_i for every $i \in \{0, \ldots, n-1\}$, and (4) a subfunction f_i defining a real value for each combination of values of X_i and $\prod(X_i)$ for every $i \in \{0, \ldots, n-1\}$. Typically, each subfunction is defined as a lookup table. The objective function f_{nk} to maximize is defined as

$$f_{nk}(X_0, \ldots, X_{n-1}) = \sum_{i=0}^{n-1} f_i(X_i, \prod(X_i)) \tag{2}$$

The difficulty of optimizing NK landscapes depends on all components defining an NK problem landscape For $k > 1$, the problem of finding the global optimum of unrestricted NK landscapes is NP-complete (Wright, Thompson, & Zhang, 2000). The problem becomes polynomially solvable with dynamic programming even for $k > 1$ if the neighbors are restricted to only adjacent string positions (Wright, Thompson, & Zhang, 2000) or if the subfunctions are generated according to some distributions (Gao & Culberson, 2002). For unrestricted NK landscapes with $k > 1$, a polynomial-time approximation algorithm exists with the approximation threshold $1 - 1/2^{k+1}$ (Wright, Thompson, & Zhang, 2000).

In this paper we consider two classes of NK landscape instances, unrestricted NK landscapes and nearest neighbor NK landscape. In the first case, unrestricted NK landscapes, for each string position X_i, we generate a random set of k neighbors where each string position except for X_i is selected with equal probability. Then, a lookup table defining f_i is generated using the uniform distribution over $[0, 1)$. In this paper we consider unrestricted NK landscapes with $k = 5$ and so these instances are NP-complete.

The second class of instances, nearest neighbor NK landscapes, have the following two restrictions:

1. Bits are arranged in a circle and the neighbors of each bit are restricted to the k bits that follow this bit on the circle. This restriction to nearest neighbors ensures that even those instances of $k > 1$ can be solved in polynomial time using dynamic programming.

2. Some subproblems may be excluded to provide a mechanism for tuning the size of the overlap between subsequent subproblems. Specifically, the fitness is defined as

$$f_{nk}(X_0, X_1, \ldots, X_{n-1}) = \sum_{i=0}^{\lfloor \frac{n-1}{step} \rfloor} f_i(X_{i \times step}, \prod(X_i)) \tag{3}$$

where $step \in \{1, 2, \ldots, k+1\}$ is a parameter denoting the step with which the basis bits are selected. The amount of overlap between consequent subproblems can be reduced by increasing the value of *step*.

To make the instances more challenging, string positions in each instance are *shuffled* by reording string positions according to a randomly generated permutation using the uniform distribution over all permutations.

The dynamic programming algorithm used to solve the nearest neighbor class of NK landscape instances is based on refs. (Pelikan, Sastry, Butz, & Goldberg, 2006; Pelikan, Sastry, Goldberg, Butz, & Hauschild, 2009). The branch and bound algorithm used to solve the unrestricted NK landscapes is based on ref.(Pelikan, 2008). While this branch and bound technique is complete and thus guaranteed to find the optimum, its complexity grows explonentially fast and solving large NK landscapes becomes intractable with this algorithm.

5 Experiments

5.1 Experimental Setup and Parameters

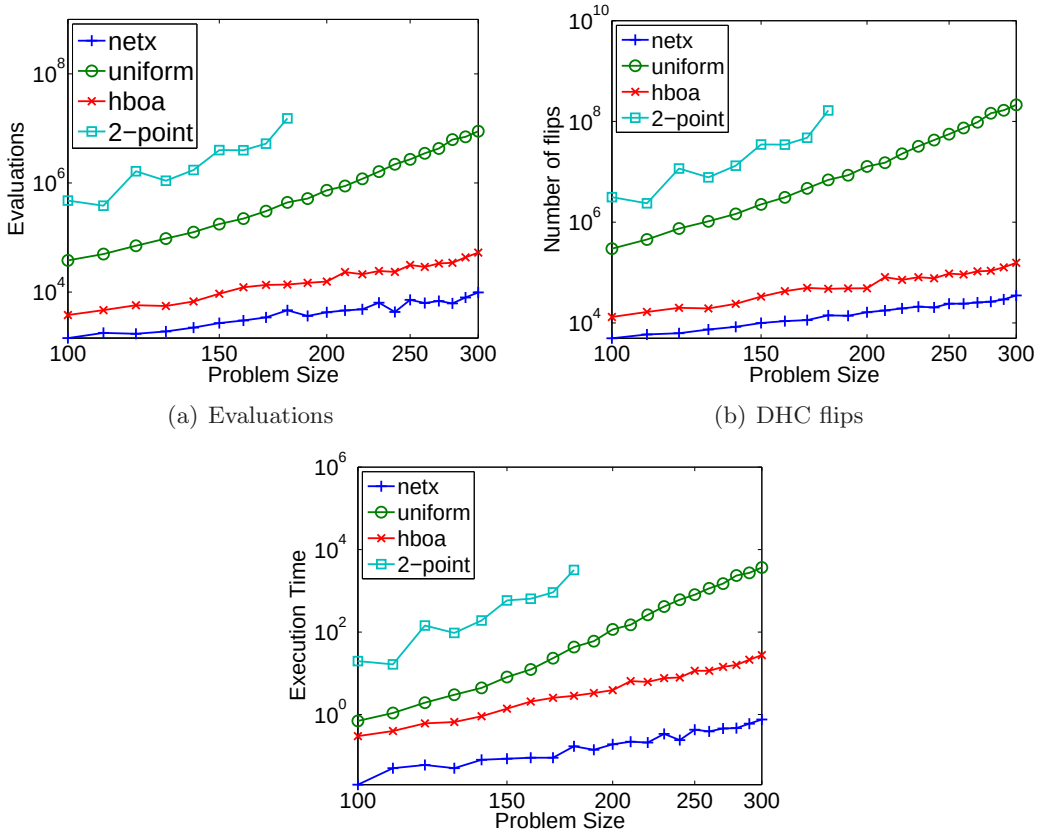
For Trap-5, problem sizes of $n = 100$ to $n = 300$ were examined for most algorithms, with bisection used to determine the minimum population size to ensure convergence to the global optimum in 10 out of 10 independent runs. For more reliable results, 10 independent bisection trials were run for each problem size and algorithm. In some cases the worst performing algorithms had their experiments cut short when the problem sizes necessary to solve the problem became extremely large.

For NK landscapes with nearest neighbor interactions, problem sizes of $n \in \{30, 60, 90, 120, 150, 180, 210\}$ were considered, with $k = 5$. Two step sizes were considered, namely $step \in \{1, 5\}$ so that we could look at the two extreme cases (most and least overlap possible between subproblems without considering those cases where the subproblems were completely disjoint). For each combination of n, k, and *step*, 1000 random problem instances were generated.

In the case of unrestricted NK landscapes, problem sizes of $n \in 20, 22, 24, 26, 28, 30, 32, 34, 36, 38$ were considered, again with $k = 5$, with 1000 random problem instances tested for each problem size.

To make sure that the performance of the algorithms was consistent regardless of the replacement technique used, new solutions were incorporated into the old population using two replacement techniques, RTR and elitism. RTR (Pelikan, 2005) is a niching method that helps to ensure diversity in a population by having new candidate solutions replace solutions that are similar to themselves in the population. Elitist replacement works by each generation keeping some portion of the population's most fit solutions. In this work, the 50% most fit solutions are kept each generation.

Both GA and hBOA were applied to all the problem instances, along with all combinations of crossover operators and replacement operators. For all GA runs, bit-flip mutation was used with a probability of flipping each bit of $p_m = 1/n$. For all crossover operators, the probability of crossover was set to 0.6.



(a) Evaluations

(b) DHC flips

(c) Execution Time

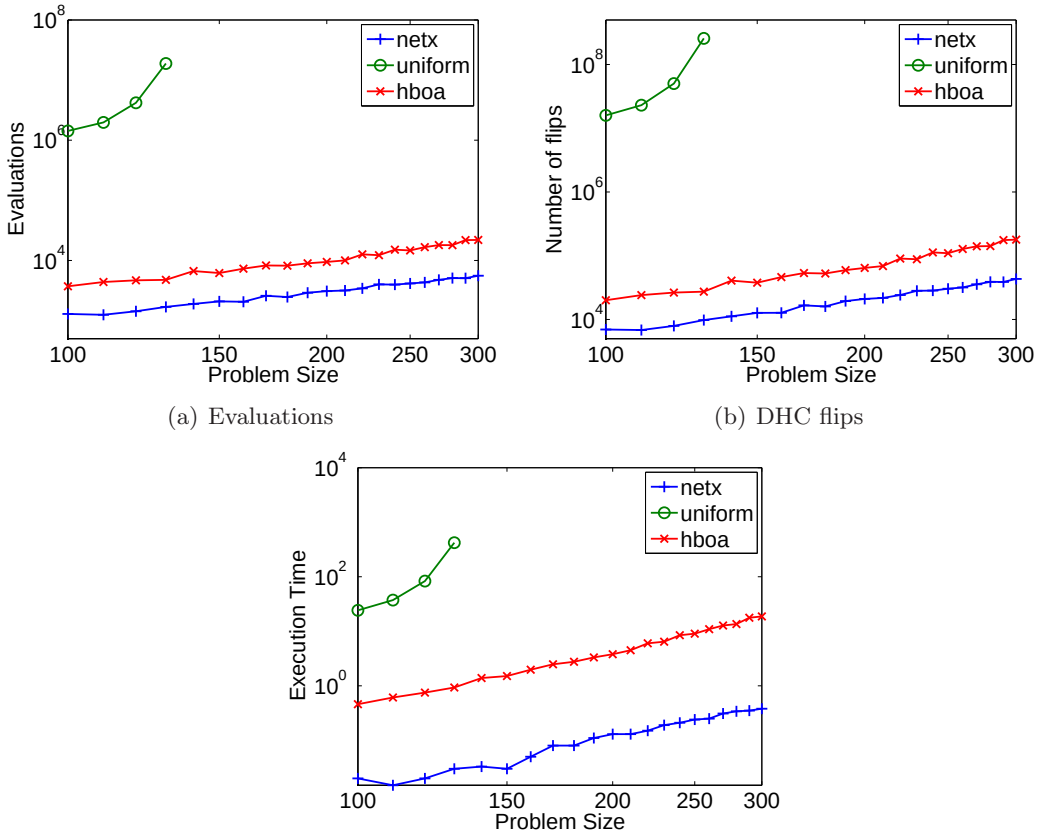
Figure 1: Performance of EAs on Trap5, RTR

For each problem instance, bisection was used to determine the minimum population size to ensure convergence to the global optimum in 10 out of 10 independent runs. Each run was terminated when the global optimum had been found (success) or when the maximum number of generations $n \times 4$ had been reached (failure). Typically when comparing algorithms, the number of evaluations is considered the key statistic. However, for hybrid evolutionary algorithms, often a great deal of time is spent in local search, so the number of DHC flips is also examined.

5.2 Trap-5, RTR

The number of evaluations necessary to solve Trap-5 using RTR for the selected algorithms is shown in Figure 1a. The best performing algorithm is the GA with network crossover for all problem sizes. This is to be expected as the network crossover should only rarely disrupt those bits in the same trap partition. hBOA also scales well as problem size increases. However, both two-point crossover and uniform crossover perform very poorly. Due to the extremely large population sizes necessary to solve Trap-5 with two-point crossover, the experiments were cut off for two-point crossover at $n = 190$.

Figure 1b and Figure 1c shows the number of local search steps and execution time, respectively, necessary to solve Trap-5 with RTR. Again, GA with network crossover performs the best for all problem sizes, with hBOA scaling similarly. Uniform and 2-point again perform poorly and show



(a) Evaluations

(b) DHC flips

(c) Execution Time

Figure 2: Performance of EAs on Trap-5, elitism

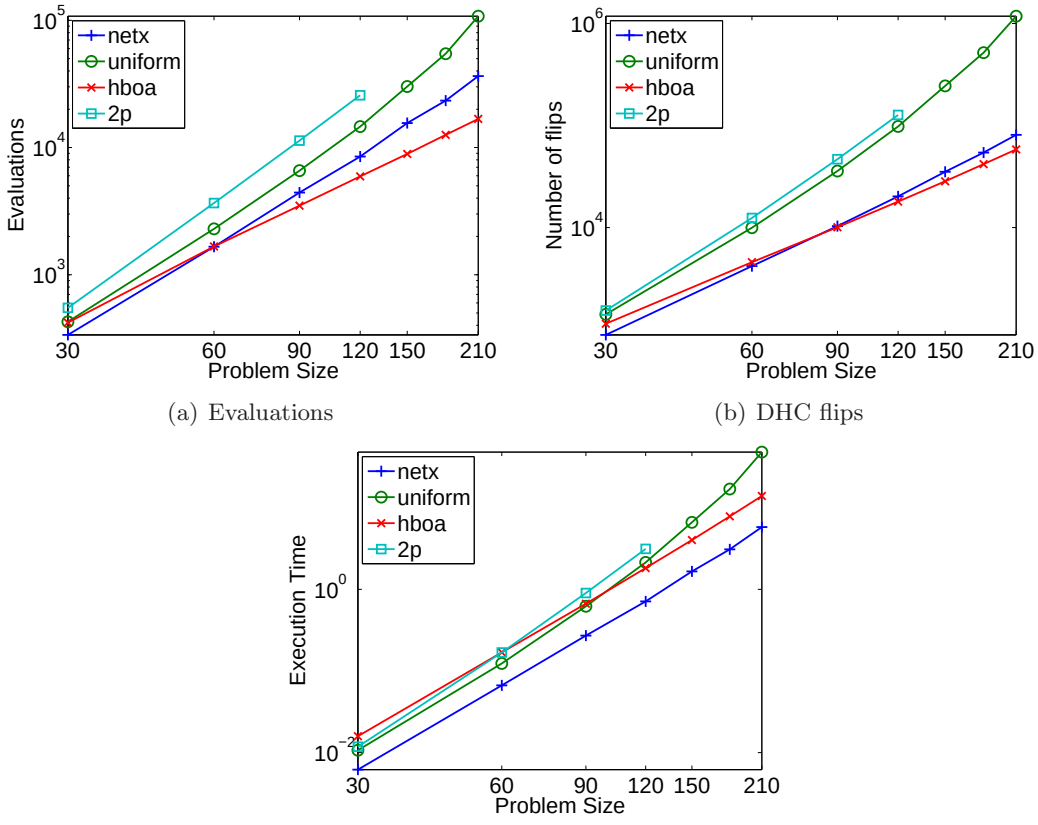
exponential scaling.

5.3 Trap-5, Elitism

While the previous results strongly showed that GA with network crossover and hBOA both scaled similarly on Trap-5 using RTR, does tihs relative performance change when the replacement technique is changed?

Figure 2a shows the number of evaluations necessary to solve Trap-5 using the selected algorithms. As with RTR, the GA with network crossover is the best performing algorithm. hBOA again scales quite well. Uniform crossover scales very poorly and the experiments were terminated for problem sizes greater than $n = 130$. Two-point crossover was not used as for $n = 100$ the GA was not able to solve all 10 independent bisection runs even using extremely large population sizes.

Figure 2b and Figure 2c shows the number of local search steps and execution time, respectively, necessary to solve Trap-5 with RTR. Again, GA with network crossover performs the best for all problem sizes, with hBOA scaling similarly. Uniform clearly shows exponential scaling of both execution time and number of local search steps.



(a) Evaluations

(b) DHC flips

(c) Execution Time

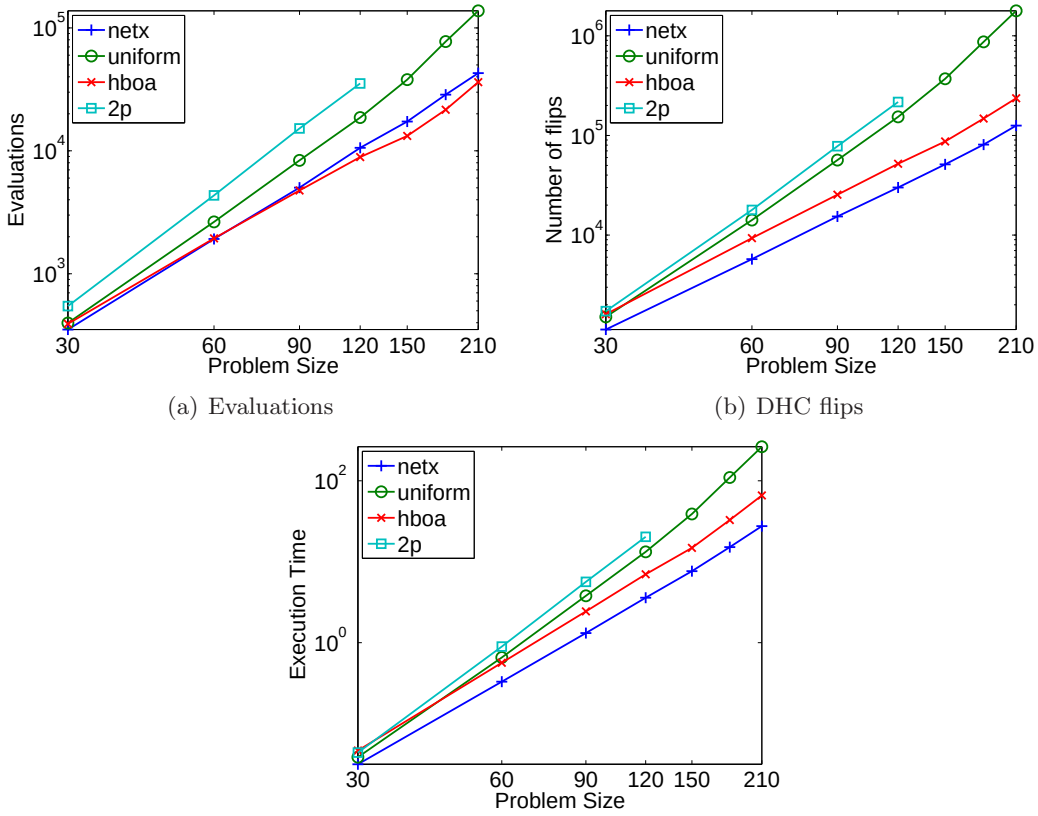
Figure 3: Performance of EAs for nearest neighor NK landscapes, $step = 5$, RTR

5.4 Nearest Neighbor NK Landscapes, RTR

Figure 3a shows the average number of evaluations used by our algorithms to solve NK landscapes with nearest neighbor interactions of $step = 5$ using RTR. In this case, for $step = 5$, we have the least interaction possible between subproblems without making them separable. We note that as problem size increases, the number of evaluations required by hBOA drops in comparison to all the GA crossover operators. The worst performing algorithm is GA with two-point crossover, taking the most evaluations for all problem sizes.

However, as noted in the previous section, another important factor in determining the performance of hybrid algorithms is the time spent in local search (DHC). The number of DHC flips until the optimum is discovered is displayed in Figure 3b. It is very clear from this figure that as problem size increases, the number of local search steps required by the GAs with uniform crossover and two-point crossover rise much higher than for the other two algorithms. The GA with network crossover requires the lowest number of steps to start, but as problem size increases, hBOA starts to require less DHC bit flips.

To combine these results, Figure 3c shows the average execution time for these instances. Since the GA with two-point crossover took the most evaluations and local search steps, we would expect that it would take the most execution time, and as problem size increases this is the case. GA with network crossover has the lowest execution time for all instances, but hBOA execution time similarly scales. It is important to note that even though the GA with network crossover takes



(a) Evaluations

(b) DHC flips

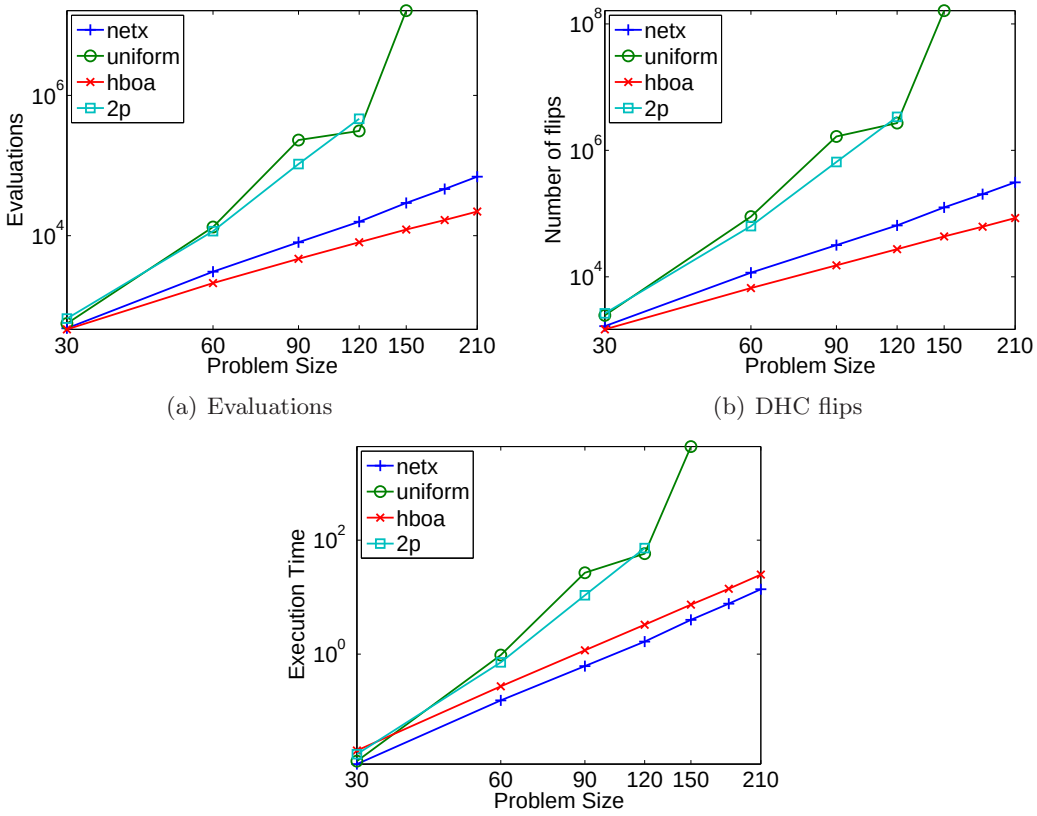
(c) Execution Time

Figure 4: Performance of EAs for nearest neighor NK landscapes, $step = 1$, RTR

more evaluations and also slightly more local search steps, it is still able to beat hBOA in execution time. This is most likely due to much less overhead in the operator, with hBOA required to build a model each generation.

The next important question is what happens when the interactions between subproblems increases? Figure 4a displays the average number of evaluations our algorithms needed for NK landscapes with nearest neighbor interactions for $step = 1$. In these experiments we are using the maximum possible overlap between subproblems. Again the GA with two-point crosssover takes the most evaluations for all problem sizes. hBOA takes the least number of evaluations for all except the smallest population size, but the GA with network crossover is still very close even for the largest problem size considered.

Figure 4b shows the number of local search steps for our algorithms when $step = 1$. The GA with network crossover has the best performance for all but the smallest problem size. On the other hand, while GA with uniform crossover starts out slightly better than the GA with network crossover for the smallest problem size, this changes dramatically as problem size increases. While hBOA takes more DHC steps than the GA with network crossover, it still scales at the same rate. These results are very closely mirrored in Figure 4c, which shows the average execution time for these instances. The worst performing algorithm is again the GA with two-point crossover. The network crossover operator leads to the best performance, but hBOA again scales similarly, taking about twice as long to solve each instance on average throughout the problem sizes.



(a) Evaluations

(b) DHC flips

(c) Execution Time

Figure 5: Performance of EAs for nearest neighor NK landscapes, $step = 5$, elitism

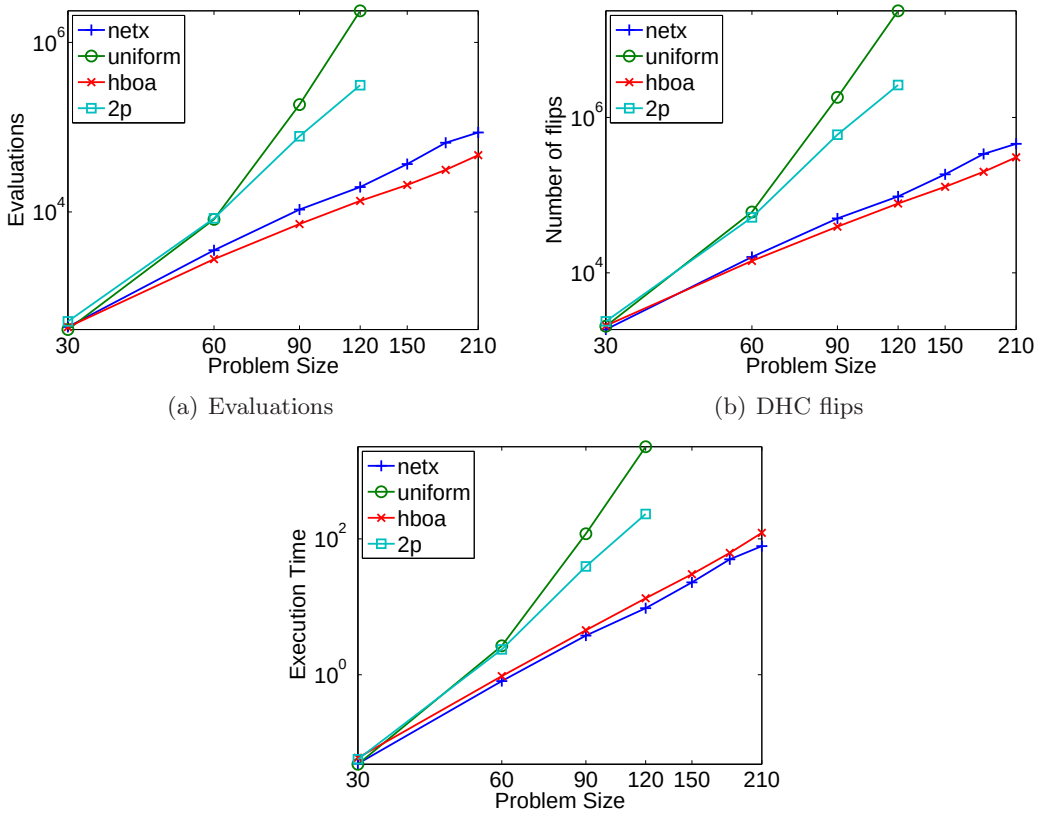
5.5 Nearest Neighor NK Landscapes, Elitism

While the previous results strongly show that GA with network crossover and hBOA both perform well with RTR, with the GAs with uniform and two-point crossover performing poorly, does this relative performance stay the same when the replacement technique is changed? To answer this question, the above experiments were repeated, but instead of using RTS, elitist replacement was used, with the worst 50% of the population replaced with new children each generation.

The average number of evaluations to solve our problems with each EA using elitism is shown in Figure 5a. hBOA requires the least number of evaluations for all problem sizes. Both GAs with uniform and two-point crossover on the other hand show extremely poor performance. As problem size increases, the number of evaluations dramatically increases, so much so that even for extremely large problem sizes it was unable to find the solution. For this reason, experiments only up to $n = 150$ for uniform crossover and $n = 120$ are shown. GA with network crossover on the other hand, while scaling slightly worse than hBOA in regards to evaluations, still grows only polynomially fast.

Figure 5b shows the number of local search steps required for each of our EAs. Again, the GAs with uniform and two-point crossover are by far the worst performers, requiring an extremely large number of evaluations even for modest problem sizes. hBOA takes the least number of local search steps, with the GA with network crossover scaling slightly worse.

Next, we look at the total execution time of our instances for $step = 5$ and elitism replacement



(a) Evaluations

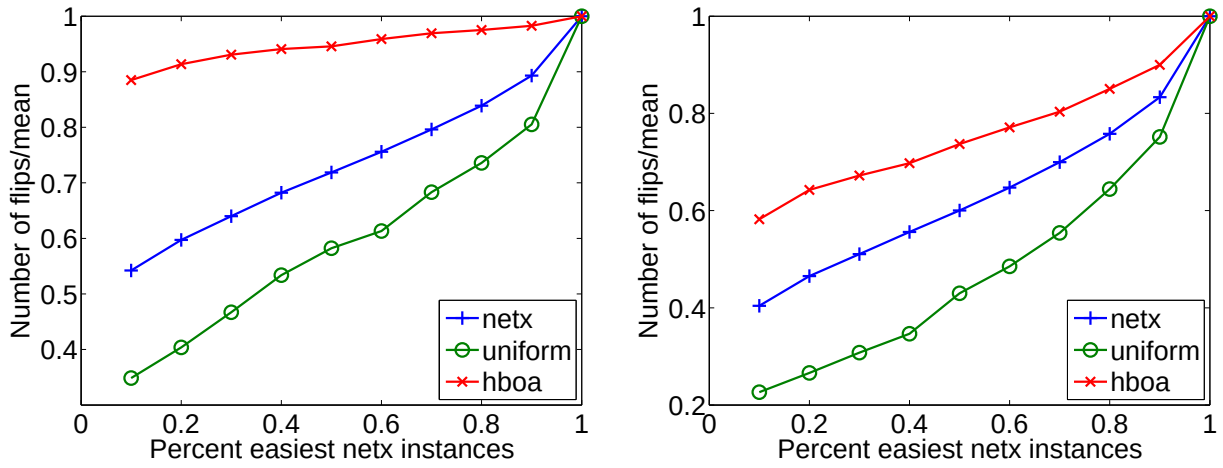
(b) DHC flips

(c) Execution Time

Figure 6: Performance of EAs for nearest neighor NK landscapes, $step = 1$, Elitism

in Figure 5c. Even though the GA with network crossover took more local search steps and more evaluations, it shows a superior execution time across the range of problem sizes. However, hBOA execution time is very close, taking about twice as on average for the largest population size. Both other GAs were much slower.

Netx, we look at the case with elitist replacement with the heaviest overlap possible between subproblems, $step = 1$. The number of evaluations required to solve these instances is shown in Figure 6a. Similarly to the case of $step = 5$, but even more severe in this case, without the niching offered by RTS the GAs with uniform and two-point crossover have trouble solving the largest instances. For this reason, the GAs with uniform and two-point crossover were only used up to $n = 120$. hBOA was the best performing algorithm in regards to evaluations. The GA with network crossover took more evaluations, with the gap between it and hBOA gradually increasing as problem size increases. Next we examine the number of local search steps required in Figure 6b. The results show that GA with network crossover spends slightly more time in local search than hBOA. Both uniform and two-point crossover performs poorly. Lastly we look at the average execution time for our EAs in Figure 6c. Again the GA with uniform crossover has the best execution time for all problem sizes, but hBOA is very close, only being slower by 30% for the largest population size.



(a) $step = 5$ (b) $step = 1$

Figure 7: Performance of EAs using RTR as a function of instance difficulty ranked by network crossover execution time, $n = 210$

5.6 Nearest Neighbor NK Instance Difficulty

In the previous results, 1000 instances of each problem size and setting were averaged. Another interesting question presents itself: As instance difficulty varies, does the performance of each algorithm vary? To attempt to answer this question, for the largest problem size($n = 210$) and using RTR with $step = 5$, we ranked the instances by their execution time when solved by the GA with network crossover (the best performing algorithm). For these results the GA with two-point crossover were omitted as it was the worst performing algorithm.

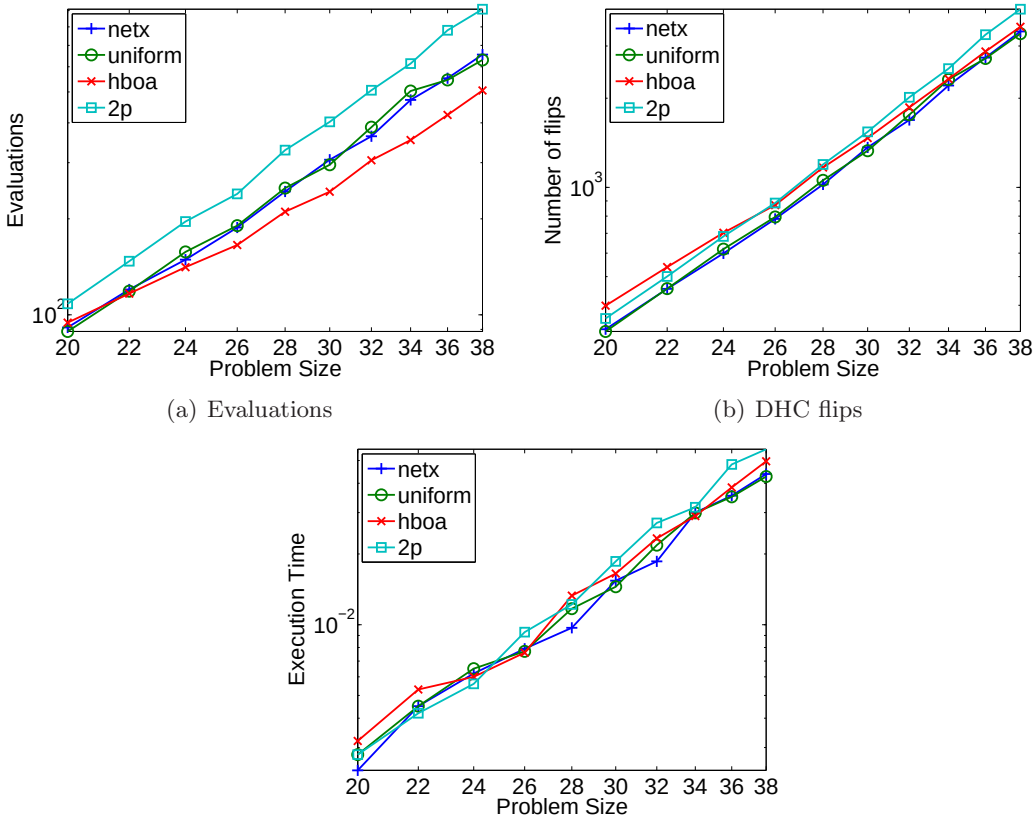
Figure 7 shows the average number of flips for a percentage of the easiest instances divided by the mean of all the instances for both step sizes. In both cases, hBOA shows the least difference in number of local search steps required between the easiest and hardest instances. In fact, for $step = 5$, hBOA spends nearly the same amount of time in local search regardless of instance difficulty. However, the other algorithms show a much more pronounced difference, with the GA with uniform crossover showing the most variance in local search steps by problem difficulty. In both cases, the GA with network crossover was between the two extremes.

5.7 Unrestricted NK Landscapes, RTR

The previous results show that the GA with network crossover is the best performing algorithm in regards to execution time for nearest neighbor NK landscapes. However, for unrestricted NK landscapes, the interactions between bits can become much more complicated.

Figure 8a shows the average number of evaluations for all compared algorithms on unrestricted NK landscapes using RTR. The worst performing algorithm with respect to evaluations is the GA with two-point crossover, requiring the most evaluations for all problem sizes. For the smallest problem size, hBOA requires slightly more evaluations than both uniform and network crossover, but for all other problem sizes it requires the least evalations and scales the best. GA with uniform and network crossover perform almost identically.

The number of local search steps for all compared algorithms on unrestricted NK landscapes using RTR is shown in Figure 8b. The GAs with uniform and network crossover performed very



(a) Evaluations

(b) DHC flips

(c) Execution Time

Figure 8: Performance of EAs on unrestricted NK landscapes, RTR

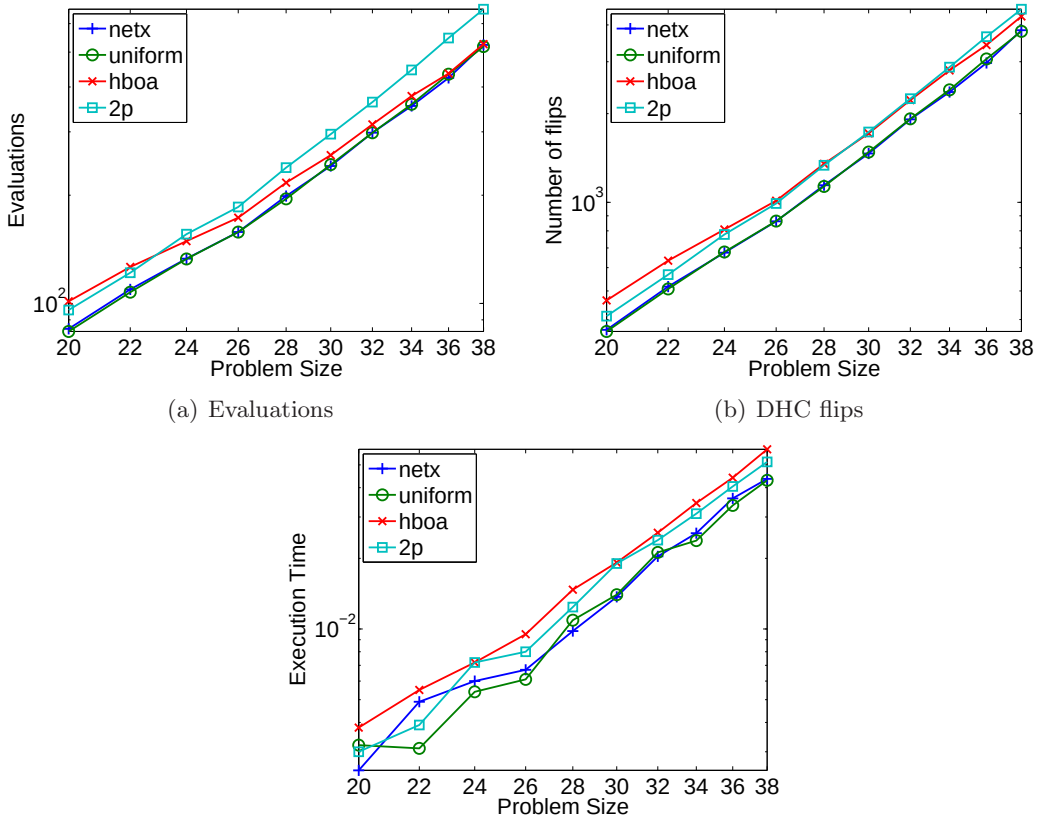
similarly, required the least local search steps. For the smaller problem sizes, hBOA requires the most local search steps, but it shows superior scaling to two-point crossover as problem size increases.

Figure 8c shows the overall average execution time for unrestricted NK landscapes using RTR. The results are very similar to evaluations, with hBOA scaling the best of all examined algorithms. GA with two-point crossover is again the worst, with uniform and network crossover performing almost identically.

5.8 Unrestricted NK Landscapes, Elitism

As in the previous problem types, again we examine the effect of changing the replacement technique. Figure 9a shows the average number of evaluations used to solve unrestricted NK landscapes using elitist replacement. GA with uniform and network crossovers performed the best in regards to evaluations. For all but the smallest problem sizes, two-point crossover performs the worst. While hBOA starts off requiring the most evaluations, it scales the best of all compared algorithms and for the largest problem size it matches the performance of the best performing algorithms.

The results for average execution time and average local search steps are show in Figure 9b and Figure 9c. Much the same pattern is observed in both. On the smallest size, hBOA is the worst performing algorithm. However, as problem size increases, hBOA shows superior scaling and in



(a) Evaluations

(b) DHC flips

(c) Execution Time

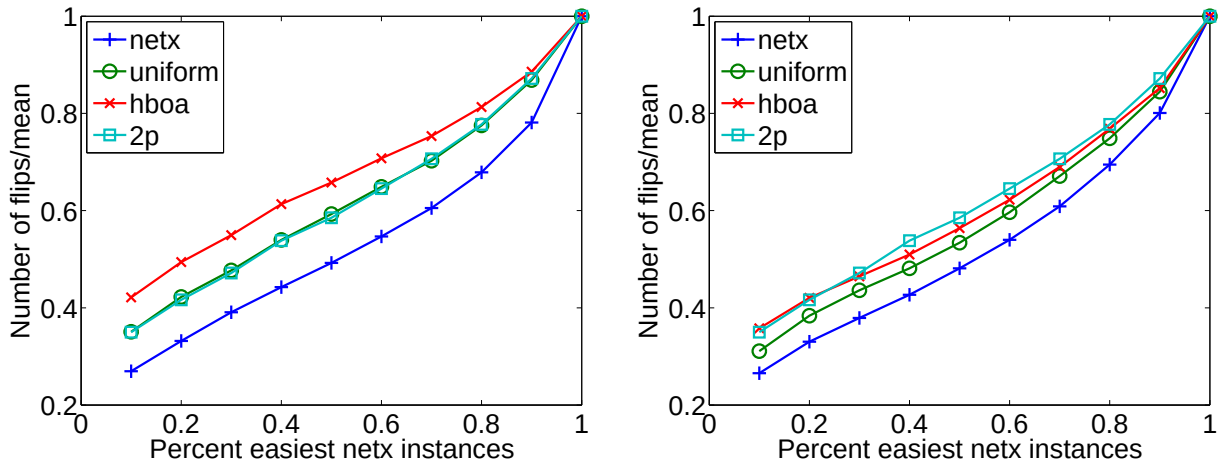
Figure 9: Performance of EAs on unrestricted NK landscapes, elitism

regards to execution time matches the GAs with uniform and network crossover. The GA with two-point crossover is clearly the worst performing algorithm as problem size increases.

5.9 Unrestricted NK Instance Difficulty

As with the nearest neighbor NK landscapes, we also examined how the instance difficulty effected each algorithm. For the largest problem size($n = 38$) and using RTR, the instances were ranked by their execution time when solved by the GA with network crossover (the best performing algorithm).

Figure 10 shows the average number of flips for a percentage of the easiest instances divided by the mean of all the instances using RTR. The results show that for all compared algorithms, much less time in local search is spent on the easiest instances. This is in contrast to the difficulty results on nearest neighbor NK landscapes, where hBOA spent almost the same amount of time in local search on the easiest instances as it did on the hardest. It is also of note that unlike for nearest neighbor NK landscapes, network crossover now shows the greatest difference in time spent in local search on the easiest instances. Also of note is that uniform and two-point crossover perform almost identically.



(a) RTR (b) elitism

Figure 10: Performance of EAs as a function of instance difficulty ranked by network crossover execution time, $n = 38$

6 Summary and Conclusions

This paper describes a method to construct a network crossover operator that can be used in a GA to easily take into account previous knowledge of problem structure. The GA with this network crossover was then used in a hybrid GA and compared to other common crossover operators and one of the strongest EDAs available, the hierarchical Bayesian optimization algorithm(hBOA). In addition, two different replacement methods were used.

It was shown that for NK landscapes with nearest neighbor interactions and trap-5, the network crossover GA required less running time than hBOA or other common GA crossover operators for all instances and parameter settings. In addition, it was shown that RTR was superior to elitist replacement on these problems.

For unrestricted NK landscapes the results showed that hBOA scaled the best out of all tested algorithms, with network crossover performing very similarly to uniform crossover. Also, the replacement method had less effect on the results, with very similar results for both elitist replacement and RTR.

An important point is that the improvements over hBOA required that the user specify a network representing the strongest dependencies between bits in the underlying problem. However, there are many classes of problems where defining such a structure is trivial, from problems in graph theory (such as graph coloring and graph bipartitioning) to Ising Spin glasses and MAXSAT. In addition, as shown in ref. (Hauschild & Pelikan, 2009), it is also possible to use runs of an EDA to learn about the structure of a problem and use this information to bias EDAs.

There are several key areas for future research on network crossover. First of all, the crossover should be tested on other problems where it is possible to create a network of important dependencies, such as graph coloring, minimum vertex cover and MAXSAT. Using an EDA to learn a network structure for futher GA network crossover runs should also be done. Next, other methods of using network information to modify crossover should be explored and tested. Finally, the GA with network crossover should be tested against a version of hBOA that takes into account problem specific knowledge to increase model-building speed.

Acknowledgments

This project was sponsored by the National Science Foundation under CAREER grant ECS-0547013, by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant FA9550-06-1-0096, 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.

The U.S. Government is authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Air Force Office of Scientific Research, or the U.S. Government. Some experiments were done using the hBOA software developed by Martin Pelikan and David E. Goldberg at the University of Illinois at Urbana-Champaign and most experiments were performed on the Beowulf cluster maintained by ITS at the University of Missouri in St. Louis.

References

Ackley, D. H. (1987). An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, 170–204.

Baluja, S. (1994). *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning* (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University.

Baluja, S. (2006). Incorporating a priori knowledge in probabilistic-model based optimization. In Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.), *Scalable optimization via probabilistic modeling: From algorithms to applications* (pp. 205–219). Springer.

Cantú-Paz, E. (2000). *Efficient and accurate parallel genetic algorithms*. Boston, MA: Kluwer.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Deb, K., & Goldberg, D. E. (1991). *Analyzing deception in trap functions* (IlliGAL Report No. 91009). Urbana, IL: University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory.

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, *15*(3), 320–330.

Drezner, Z., & Salhi, S. (2002). Using hybrid metaheuristics for the one-day and two-way network design problem. *Naval Research Logistics*, *49*(5), 449–463.

Etxeberria, R., & Larrañaga, P. (1999). Global optimization using Bayesian networks.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (pp. 421–459). MIT Press.

Gao, Y., & Culberson, J. C. (2002). An analysis of phase transition in NK landscapes. *Journal of Artificial Intelligence Research (JAIR)*, *17*, 309–332.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer.

Harik, G. R. (1995). Finding multimodal solutions using restricted tournament selection. *International Conf. on Genetic Algorithms (ICGA-95)*, 24–31.

Hauschild, M., Pelikan, M., Sastry, K., & Goldberg, D. E. (2008). Using previous models to bias structural learning in the hierarchical BOA. *Genetic and Evolutionary Computation Conf. (GECCO-2008)*, 415–422.

Hauschild, M. W., & Pelikan, M. (2009). Intelligent bias of network structures in the hierarchical boa. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (pp. 413–420). New York, NY, USA: ACM.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Kauffman, S. (1989). Adaptation on rugged fitness landscapes. In Stein, D. L. (Ed.), *Lecture Notes in the Sciences of Complexity* (pp. 527–618). Addison Wesley.

Kauffman, S. A. (1993). *The origins of order: Self-organization and selection in evolution.* Oxford University Press.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer.

Mühlenbein, H., & Mahnig, T. (2002). Evolutionary optimization and the estimation of search distributions with applications to graph bipartitioning. *International Journal on Approximate Reasoning*, *31*(3), 157–192.

Mühlenbein, H., & Paaß, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, 178–187.

Ocenasek, J., & Schwarz, J. (2000). The parallel Bayesian optimization algorithm.

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

Pelikan, M. (2008). Analysis of estimation of distribution algorithms and genetic algorithms on nk landscapes. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 1033–1040).

Pelikan, M., & Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms. *Genetic and Evolutionary Computation Conf. (GECCO-2001)*, 511–518.

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

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20.

Pelikan, M., Sastry, K., Butz, M. V., & Goldberg, D. E. (2006). Performance of evolutionary algorithms on random decomposable problems. In *PPSN* (pp. 788–797).

Pelikan, M., Sastry, K., & Cantú-Paz, E. (Eds.) (2006). *Scalable optimization via probabilistic modeling: From algorithms to applications*. Springer-Verlag.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2006). Sporadic model building for efficiency enhancement of hierarchical BOA. *Genetic and Evolutionary Computation Conf. (GECCO-2006)*, 405–412.

Pelikan, M., Sastry, K., & Goldberg, D. E. (2008). iboa: the incremental bayesian optimization algorithm. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 455–462). New York, NY, USA: ACM.

Pelikan, M., Sastry, K., Goldberg, D. E., Butz, M. V., & Hauschild, M. (2009). *Performance of evolutionary algorithms on nk landscapes with nearest neighbor interactions and tunable overlap* (MEDAL Report No. 2009002). Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO.

Sastry, K., Goldberg, D. E., & Llorà, X. (2007). Towards billion bit optimization via parallel estimation of distribution algorithm. *Genetic and Evolutionary Computation Conf. (GECCO-2007)*, 577–584.

Schwarz, J., & Ocenasek, J. (2000). A problem-knowledge based evolutionary algorithm KBOA for hypergraph partitioning. Personal communication.

Stonedahl, F., Rand, W., & Wilensky, U. (2008). Crossnet: a framework for crossover with network-based chromosomal representations. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation* (pp. 1057–1064). New York, NY, USA: ACM.

Thierens, D. (1999). Scalability problems of simple genetic algorithms. *Evolutionary Computation*, *7*(4), 331–352.

Wright, A. H., Thompson, R. K., & Zhang, J. (2000). The computational complexity of n-k fitness functions. *IEEE Trans. Evolutionary Computation*, *4*(4), 373–379.