Automatic Generation of Domain-Specific Genetic Algorithm Operators using the Hierarchical Bayesian Optimization Algorithm.

Cezary Z Janikow
University of Missouri in St. Louis
One University Blvd.
St. Louis, MO 63109
janikowc@umsl.edu

Mark Hauschild
University of Missouri in St. Louis
One University Blvd.
St. Louis, MO 63109
hauschildm@umsl.edu

ABSTRACT

GAs started with generic mutation and crossover operators, but over the years specialized representations and/or operators designed specifically for a given domain or problem, such as TSP, proved the most effective. In this paper, we define a class of new GA operators which automatically adjust for each problem. The adjustments or instantiations are based on the domain model presented to the operators in the form of Bayesian Network, as generated in the hierarchical Bayesian Optimization Algorithm (hBOA). We then show that these operators outperform standard random operators as long as the models are of sufficient quality.

CCS CONCEPTS

•Computing methodologies → Probabilistic reasoning; Bayesian network models; Genetic algorithms;

KEYWORDS

binary genetic algorithms, crossover operators, mutation operators, informed operators, knowledge-intensive operators, hierarchical BOA, probabilistic model

ACM Reference format:
Cezary Z Janikow and Mark Hauschild. 2017. Automatic Generation of Domain-Specific Genetic Algorithm Operators using the Hierarchical Bayesian Optimization Algorithm.. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071331

1 INTRODUCTION

While genetic algorithms (GAs) have shown the ability to solve many problems, in order for them to solve problems robustly and scalably, their operators must respect the linkage between bits [4]. One solution to this problem is to design competent GAs that incorporate linkage learning, such as estimation of distribution algorithms (EDA) [2, 10, 12, 15]. EDAs work by building a probabilistic model of promising solutions and then sampling new candidate solutions from the built models. Even though EDAs have many advantages over the standard GAs and their operators [10, 17], the model building is very computationally intensive.

Over the years, there were attempts at other solutions as well. For example, many interesting problems have been solved by first designing a non-binary representation, designed to reduce the conceptual gap between genotype and phenotype, and then designing problem-specific operators in that representation, utilizing many domain heuristics. A good example of this approach is TSP [5, 11] and symbolic machine learning [6]. Of course designing such representations and operators is very time consuming. Therefore, it would be natural to somehow automate this process. One way to accomplish this is to allow the representation to evolve so that the standard operators would be able to utilize some problem regularities [7]. Unfortunately, this approach can be inefficient as not all regularities can be discovered by just rearranging bit positions.

Another approach could be to leave the representation fixed, and allow the operators to evolve, or otherwise explore the problem regularities. This is the approach taken in this paper. We design a number of operators, which always perform differently based on the supplied information about the domain, or rather about the regularities and dependencies among genes and alleles. As a source of this information, we utilize domain models discovered by hBOA [14] - Bayesian Networks.

The paper is organized as follows. Section 2 reviews the hBOA model and its properties that we will utilize. Then we design our recombination operator templates in Sections 3 and 4. In Section 5 we empirically analyze the resulting operators. Section 6 summarizes and concludes the paper.

2 MODEL

We use the model representation from hBOA [14], where the model is a Bayesian Network and it is represented using dependency trees. We use this model for two reasons. First, these models can be acquired through hBOA runs. Second, once we have operators able to utilize this model, we will subsequently try to utilize them to improve hBOA performance altogether. On the other hand, using the hBOA model does not necessarily require running hBOA on a problem - there may be other ways to construct these models.

2.1 Dependency trees

The dependency trees have the following properties that we will utilize.

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071331

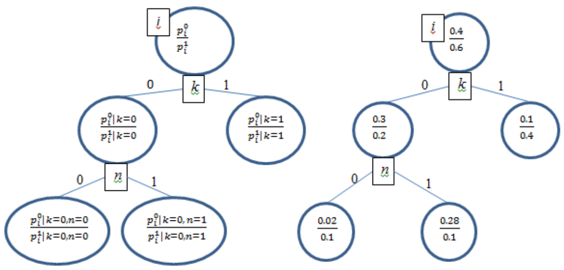


Figure 1: An example of a possible Dependency tree for bit i. In this example, bit i depends on bits k and n. On the left are the meanings of the probabilities, on the right a specific case.

(1) Each dependency tree has probabilities estimated from the current population and showing as many dependencies as observed in the population.
(2) Dependency trees show (estimated from the population) how a given position bit depends on other positions.
(3) Each bit position from a chromosome has a corresponding dependency tree, as shown in Figure 3, but some of those position-trees may have no dependencies.
(4) The probability values in the root stand for probability of the bit being 0 (top in Figure 3) and 1 on (bottom).
(5) The probability values add to 1 in the root, and add to 1 when added over all leaves.
(6) The top probabilities (for the bit being 0) added over all leaves add up to the same probability in the root. The same for bottom values and the probability of the bit being 1.
(7) A position can appear in more than one position-tree as dependency position but not in a way to introduce cycles. So if a position i depends on position j, then j cannot depend on i directly or indirectly

2.2 Utility Measure for Dependency Tree

Each dependency tree provides information about the bit at this position, based on the estimate from the current population. Some bits have dependency context, others do not.

2.2.1 Position with no Dependency Context. The simplest possible case for a position is when the positionfis bit does not depend on bits of other positions. In this case, the only information is the frequency of the positionfis bits, illustrated in Figure 1 assuming only roots are present. We use the information entropy measure $I^i = -((p_i^0)^* log ffi(p_i^0) + p_i^1 * log ffi(p_i^1))$ to determine if the bits on position i are random or following a pattern. The best case for a pattern, when either probability is 0, gives $I_i = 0$.The worst case, when both probabilities are equal, gives $I_i = 1$. Then we introduce a new utility measure $J = 1 - I$. J can be seen as a utility measure (not a true probability metric) for the position, the higher the measure the more information about the bit at this position. For example, probabilities 0.5/0.5 give $J = 0$, probabilities 0.6/0.4, as illustrated, give 0.03, 0.1/0.9 give 0.53, and probabilities 0.01/0.99 give 0.92.

2.2.2 Position with Dependencies. A given position can depend on bit values from other positions. Figure 1 shows a dependency tree for i, where i depends on k and n. The first two nodes under the root provide information on i's dependency on the first additional position, here k: for k=0 on the left and for k=1 on the right. The probabilities in those nodes are thus the probabilities of i=0 (top) and i=1 (bottom) when k=0 on the left, and the same when k=1 on the right. Then, the two additional leaf nodes on the left show how the probabilities for the first left node split when additional position n is used. To establish the utility measure for the positions with dependencies, we use the information content of the leaves instead of that of the root, weighted by importance of the nodes, and we use the above properties. We can now use these weights to establish weighted entropy over the leaves. For example, in figure 1, if the position i had no dependencies, as before, then its utility measure was J=0.03. If the nodes based on k were added (i depends on k now), then the tree would have the top 3 nodes only, and the utility measure of this tree would be J=0.154. Adding also the two leaves based on n (as shown), the utility of the entire tree is now J=0.245, an advantage over both previous cases. As seen, the higher the utility of a tree, the more context information it contains. In Figure 1, the full tree has the highest utility, meaning that if bits k and n are known then bit i can be predicted with highest certainty.

2.3 Dependency Context

The dependency context for a bit position is the set of bit positions on which the given position depends on. For example, if the bit in position 1 has two positions 2 and 3 in its dependency tree, these two positions 2 and 3 are the dependency context for position 1. In other words, the actual bit value for position 1 can be better predicted by also observing bits for positions 2 and 3.

2.3.1 Direct Dependency Context. If position i depends on positions k and n in the position-tree for position i, then positions k and n are the context in which position i should be considered. Because they come directly from a single dependency tree, we call this the direct dependency context for position *i* fi?! denoted $DDC_i = \{k, n\}$. The strength of this direct context is the $JDDC_i$ utility measure. Therefore, $JDDC_i$ can be used to measure utility of direct context DDC_i.

2.3.2 Markov Cover Context. We can also create larger masks by including transitive dependencies. For example, borrowing the concept of markov cover from Pearl [13], we can define markov cover context MCC_i as DDC_i union all positions m found in all trees depending on position i. The utility $JMCC_i$ of the cover is the utility measure computed iteratively. The utility of markov cover context based on only Dependency tree i is $JMCC_i = JDDC_i$. For each new position being added to the markov cover context, the utility becomes $JMCC_i = JMCC_i + (1 - JMCC_i) * JDDC_x$ Note that $J_M CC_i$ft$J_D DC_i$. It is strictly larger unless the two contexts are the same. This utility will grow for larger contexts, with increases decreasing with weaker dependencies - and this is what we want to measure, the larger and stronger contexts. A potential MMC is illustrated in Figure 2.

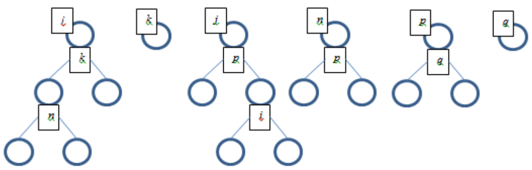


Figure 2: A potential set of related dependency trees in addition to that of Figure 1 (there could be more dependency trees that are not related). Then, $DDC_i = \{k, n\}$**,** $MCC_i = \{k, n, p, j\}$ **and** $EMC_i = \{k, n, p, j, q\}$**.**

2.3.3 Extended Markov Cover Context. We can also create an extended markov cover context EMCi by adding additional related dependencies until we have no more dependencies or the set contains half the positions in a chromosome (this will be used in crossover; such a mask will mix two chromosomes by exchanging about half of the chromosome). We extend the idea of markov cover context as follows to create the extended markov cover context EMC_i:

(1) Start with DDC_i. If the context contains at least half the positions, stop.
(2) Move on to MCC_i, with its utility measure. If the context contains at least half the positions, stop.
(3) Add DDC_x for all positions x found in MMC_i, one at a time in the order of decreasing $JDDC_x$. For every Dependency tree x added, update the utility of the context $JEMC_i = JEM_i + (1 - JEMC_i) * JDDC_x$. Stop after any x when the context contains at least half the positions.
(4) If any new positions were added in step 3, continue step 3 for these new positions. Stop if no new positions.

A potential *EMC* mask is illustrated in figure 2. Note that *EMC* contexts induce a binary partition of the set of positions in a chromosome into approximately equal-sized sets.

2.4 Masks

The previously defined contexts can be used now to create masks, which will be used in mutation and crossover. A mask is the set of all positions found in a context and the utility of the mask is the utility of the context. The smallest mask DDC links positions most directly related, the other masks link positions linked indirectly. The utility of a DDC mask will always be less or equal to the utility of its MMC mask, etc. This will not cause problems as we will never compare two different kinds of masks using the utility measure.

3 MUTATION

Mutation operates on a single chromosome until a change is made. It attempts to "improve" or "repair" the chromosome rather than making random changes and is guided by the model. Thus, mutation is less likely to produce improvements for chromosomes that are closely aligned with the model. A mutation will not be complete until there is a change (except for helper mutation which is not an operator). We propose multiple mutations. Each mutation operates on the position(s) chosen using the utility measures. Here we introduce Fix-Position-In-Context (FPiC) mutation; they attempt to repair a bit position, improving its fit to its observed context restricted by a mask. Each mutation has singular (postfix s) and also uniform (postfix u) versions. Each mutation can also use any of three different masks: (postfix d for direct mask, m for markov cover mask, e for extended markov cover mask), which would allow the mutation to propagate to other positions depending on this position, directly or indirectly. Thus, the possible mutations are: **FPiC{d,m,e}{s,u}**. First we define helper kernel mutation, which itself is not a complete mutation operator but appears in some mutations.

3.1 Kernel Mutation for Fixing Position in Context KMiC

Kernel mutation KMiC$_i$ is a helper single trial mutation attempting to mutate the bit on position i based on its direct context DDC_i.

(1) The position i is already chosen.
(2) If the context is empty (DDC_i mask is empty), select the root probabilities in the dependency tree for i. If there are other positions in the mask, it means that bit i depends on other bits. Using the observed bit values of those other positions, select the corresponding leaf node in the dependency tree for i. Whichever node was chosen, it gives us probability (weight) for $i = 0$ on the top, and separately for $i = 1$ on the bottom.
(3) Use the two weights to generate new bit $i = b^i$ (can be the same as current or different). Return success if the bit i is changed, otherwise failure. For example, assume the dependency tree for i as of Figure 3 right. Suppose the current bits for k and n are 0 and 1, respectively. These values designate the middle leaf, which provides the weights: 0.28 for $i = 0$ and 0.1 for $i = 1$. Thus $i = 0$ is generated with probability 0.737 and $i = 1$ with probability 0.263. And this results in either changing i or not depending on the current value of i. For example, if $i = 0$ was selected from the above probabilities, and its current value is 1, then change i to 0 and return success.

3.2 Mutation FPiCds

This mutation works under a direct mask and it is a single version. It considers the context for the position (using its direct mask), and based on the context it attempts to repair the position bit i. It operates until a change is made (success), but the change is not propagated to other bits.

(1) Select a chromosome for mutation.
(2) Select a direct mask with probability proportional to the $JDDC$ measures (can be deterministic starting with the highest utility or stochastic). If no more masks available, start all over from 2 with all masks available again. Assume the selected mask is for position i.
(3) Perform kernel mutation KMiC$_i$. If success, return success, else go back to 2 selecting from positions not tried yet.

3.3 Mutation FPiCms

This mutation works as FPiCds except that after the bit for i is changed (FPiCds returns only when successful), the change propagates to all positions in MCC_i that directly depend on i (i is found in their DDC). In other words, propagate the first change to all directly affected bits.

(1) Perform FPiCds mutation (it is always successful when finished).
(2) Pick all dependency trees m dependent on i (this is really a subset of MCC_i but it contains ALL bits that directly depend on i), and order them in decreasing utility measure.
(3) Perform kernel mutation KMiC$_m$ in each position m, one at a time, always assuming current bit plus any changes made in the current mutation (if m changes, its new value will be used in subsequent kernel mutations in this FPiCms mutation).
(4) Return success.

3.4 Mutation FPiCes

This mutation works as FPiCms except that if any additional bit m is changed (in addition to i which necessarily changes), these changes also propagate to all positions affected by the new bit. Note that because EMC_i mask can be cut short of true transitive closure (when its size covers half the chromosome), the bits potentially changed in this mutation can lie outside of the EMC_i mask, but this is unlikely as the cut off dependencies would be the weakest dependencies.

(1) Perform FPiCms mutation (it is always successful on i when finished, but it could have changed more bits j).
(2) Pick all dependency trees m dependent on all changed positions j (the change to i was already propagated in FPiCms) and order them in decreasing utility measure. If no more, return success.
(3) Perform kernel mutation KMiCm in each dependency tree m, one at a time, always assuming current bits plus any changes done in the current mutation. If m changes, its new value is added to j to be used in subsequent kernel mutations in this FPiCms mutation. Go back to 2.

3.5 Uniform and Random Mutation

We also implement a random mutation, and then uniform versions of all the mutations.

- Mutation **Ms**; after selecting chromosome, select a random bit and flip it
- Mutation **Mu**; as **Ms** but flip each bit of the selected chromosome with probability eMu/L where eMu is some parameter (standing for the expected number of flips) and L is the chromosome length.

4 CROSSOVER

A crossover always works with two parents, and it attempts to exchange some bits between the two parents. The objective of our crossover is to keep together bits in the same context, using masks. These masks group together bits on positions that are contextually dependent, therefore the purpose of a mask is to keep a group of bits from being disrupted. Here we introduce Chromosome-Independent-Crossover (**CiC**) crossover, which can be based on direct mask (d), markov cover mask (m), and extended markov cover mask (e).

4.1 Crossover CiCd

This crossover uses a direct mask independently of the chromosomes under consideration.

(1) Select one chromosomes.
(2) Select a direct mask based on J values.
(3) The positions in the mask are grouped together, the remaining positions form the other group for the crossover.

This operator is very asymmetric.

4.2 Crossover CiCm

This crossover uses a markov cover mask independently of the chromosomes under consideration.

(1) Select one chromosomes.
(2) Select a markov cover mask based on J values.
(3) The positions in the mask are grouped together, the remaining positions form the other group for the crossover.

This operators performs less asymmetric crossover but still not very symmetric.

4.3 Crossover CiCe

This crossover uses an extended markov cover mask independently of the chromosomes under consideration.

(1) Select one chromosomes.
(2) Select an extended markov cover mask based on J values.
(3) The positions in the mask are grouped together, the remaining positions form the other group for the crossover.

This operator is a very symmetric crossover (for many cases the two parts split but the crossover are about equal in size).

4.4 Random Crossover

We also implement random uninformed crossovers for comparative purposes.

- **Cs**; a standard one-point crossover with a random crossover point on two parents. This crossover is expected to work better if the dependent bits for a problem are physically grouped together.
- **Cu**; a standard uniform crossover on two parents: walk over all positions in the two parents, and swap bits of each position with probability 0.5. This crossover will swap about half the bits and is expected to work better when the dependent bits and their physical locations are not correlated.

5 EMPIRICAL ANALYSIS

The objectives here are to analyze:

- Soundness of the proposed operators.
- Properties of the proposed operators with respect to quality/completeness of information provided in the model.

- Performance of the operators as compared to the random uninformed operators.
- Performance of groups of operators.

On the other hand, we do not set to compare the behavior or a GA running with these new operators as compared to that of hBOA itself. We are performing this analysis and will report the results separately.

5.1 Test Problem: Concatenated 5-bit trap

In trap-5 [1, 3], 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.

5.2 Test Problem: NK Landscapes

An NK fitness landscape [8, 9] 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 [19].

In this paper we use nearest neighbor NK landscapes, which 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 re-ording 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. [16, 18].

5.3 Experimental Setup and Parameters

To test the differing effects of model quality on the quality of the informed operators, hBOA was run for a range of generations (3,6,9,12,15,18, not all reported here) and the resulting hBOA model was subsequently used in a GA algorithm running for 300 generations. The GA started with a random uniform population, and used either one operator at a time or a pair of operators. The operators included some of the informed mutation and crossover as previously defined, as well as random operators as a reference. We will evaluate more operators in a separate work.

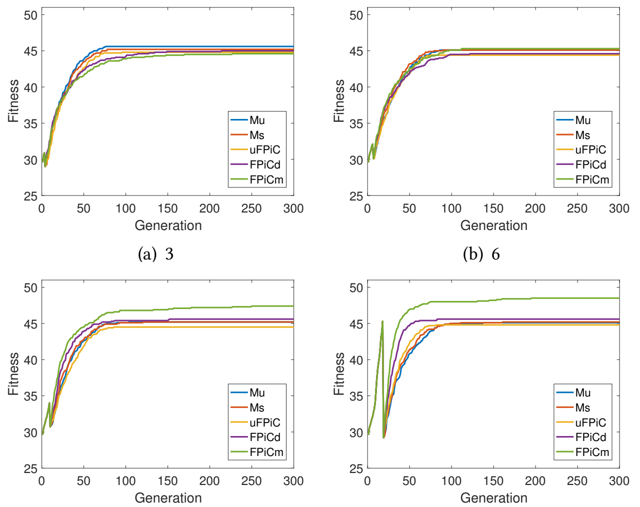
We examined trap-5 instances of 50 bits, and nearest neighbor NK landscapes of 41 bits with one bit of overlap between its partitions. This was done so that each problem we examined had ten different partitions, but the NK landscape problems had overlap between partitions and differing difficulty in each of these partitions. 100 different instances of this size of NK landscape were used in testing.

The probability of crossover during GA recombination was 60%, with 40% a simple copy performed. Informed mutations were performed until a single bit is flipped (successful operators). In order to compare the informed mutations to the random mutations, the probability of mutation for random mutation operators was set to give an expected value of one bit flipped.

A set of representative mutation operators were selected to compare against each other. For informed mutation based on knowledge gained from the hBOA model, FPiCds and FPiCms were used. In order to test the effectiveness of these operators compared to standard GA operators, Ms and Mu were used. To test the effectiveness of selecting a bit to mutate based on J values, a variant of FPiCds was used where the bit selected for mutation was selected uniformly rather than through J values, which we will refer to as uFPiCds.

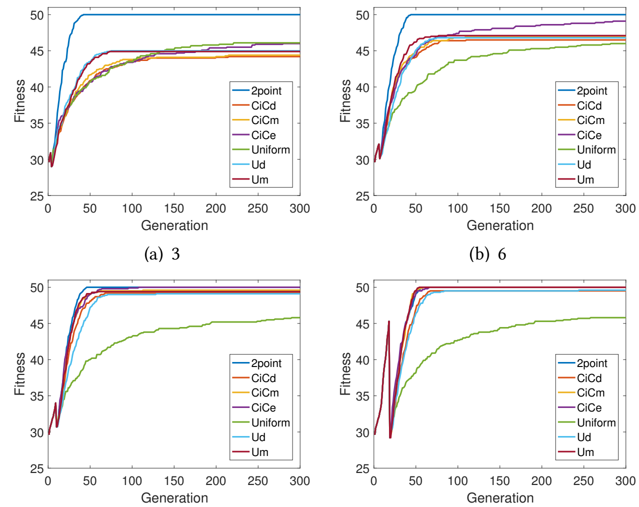
A set of recombination operators was also selected. For informed operators, CiCd, CiCm and CiCe were used. In order to test the effectiveness of these operators compared to standard GA operators, 2-point and uniform crossover were also used - 2 point with 5-bit trap and uniform with NK landspace as these operators are expected to perform best for these problems. To verify the usefulness of selecting the initial crossover point based on J values, two variants of CiCd and CiCm were used where the crossover points were selected uniformly rather than based on J (called uCiCd and uCiCm respectively).

For each problem and parameter setting, 10 independent runs were used.



(a) 3 (b) 6 (c) 9 (d) 18

Figure 3: Maximum fitness by generation for trap-5 using only mutation.



(a) 3 (b) 6 (c) 9 (d) 18

Figure 4: Maximum fitness by generation for trap-5 using only recombination.

5.4 Concatenated 5-bit trap results

Figure 3 shows the performance of the GA using selected informed and random mutations and various quality hBOA models on the 5-bit trap problem. The four various model cases are those collected after running hBOA for 3, 6, 9 and 18 generations.

Figure 3a and b show the result when using the lowest quality models and, as expected, all of the mutation operators perform similarly. However it can be seen that the informed runs are under performing especially in case (a) as the poor quality model misleads the operators. In case (c) we finally notice that the informed runs begin to outperform the runs with purely random operators. The two informed operators perform the best, with FPiCms continuing to increase the quality of the solutions gradually. The worst performing informed operator at this point is uniform FPiC, which points to a strong performance gain by selecting our bit to mutate based on J rather than uniformly. Finally, in case (d) we can see that hBOA was able to build a good quality model before moving on to GA. The FPiCms mutation quickly outperforms the other operators. FPiCds is the second best. uFPiCds initially performs better but ten falls short of the others.

Figure 4 shows the performance of the GA using selected informed and random recombination operators and the same various quality hBOA models on the same trap problem.

Figure 4a shows the GA results when the model is very poor quality. As expected, the so-poorly informed operators do not perform much better than uniform crossover. On the other hand, the 2-point simple crossover clearly outperforms all other crossovers because, for this problem, the operator tends to preserve the building blocks. In case (b), where the hBOA model is still poor but better than in case (a), we can see that some of the informed operators start to improve substantially but still do not match the performance of the well-suited 2-point crossover. Among the informed operators, Ud and Um, the operators selecting crossover points uniformly but then using the direct tree or the markov blanket to select crossover neighborhoods, are somehow better than the other informed operators - which need more reliable information. Cases (c) and (d) show that the fully informed operators perform better and better as the quality of the model improves, eventually matching the behavior of the well suited 2-point crossover. CiCe, CiCm, CiCe and Um all perform at nearly the same level, finding the maximum fitness of the problem instances in many cases. CiCd and Ud are the two worst of the informed operators, pointing to the necessity of a larger cover for recombination.

5.5 Nearest neighbor NK landscape results

In the previous experiments we saw that the informed operators were superior in most cases to the random operators when given a sufficient quality model. In addition, the use of J to select crossover and mutation points was beneficial. However, trap-5 has no overlap between subproblems and all the partitions have the same difficulty, and this is why a simple 2-point crossover was found to perform as well as the informed operators. In this section we repeat the same experiments but for the NK landscape problem.

Figure 5 shows GA performance when using selected informed and random mutations on the latter problem when provided with various quality models. In cases (a) and (b) the random mutations tend to perform better, as before, as the information provided in the models is misleading the informed mutations. However, in case (c) and especially (d) when the model is relatively good, the informed mutations start to outperform the random mutations. Among the informed mutations, the one based on the markov cover seems to outperform the mutations operating on one mask at a time - FPiCm is the best operator, with FPiCd the best of the other operators but notably worse. uFPiC initially performs better than Ms and Mu but later on in generations are able to match its performance.

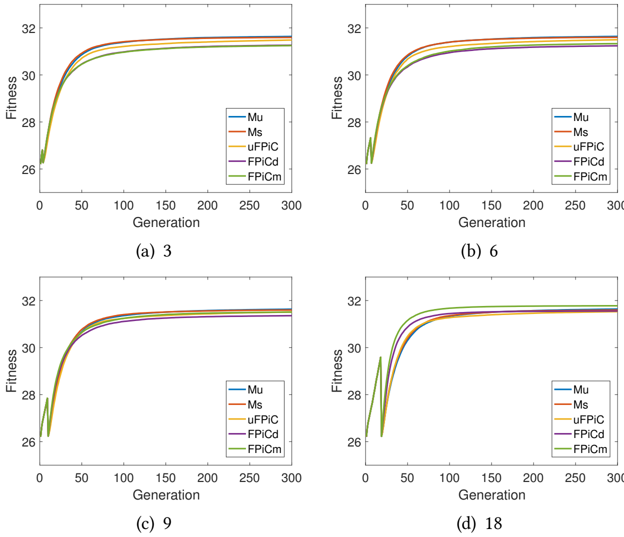


Figure 5: Maximum fitness by generation for NN NK landscapes using only mutation.

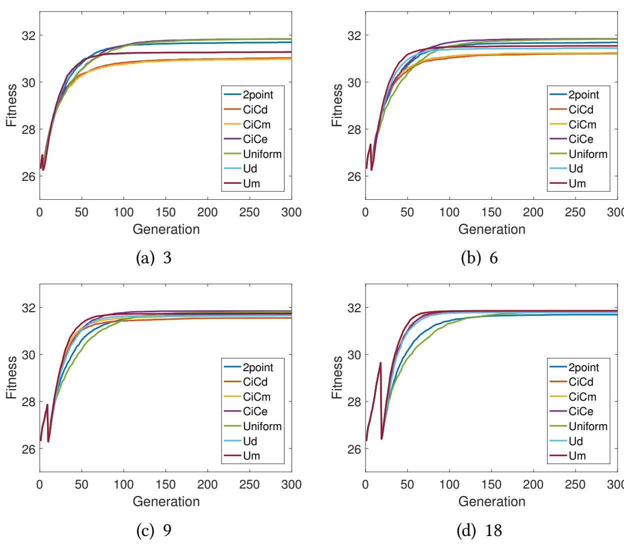


Figure 6: Maximum fitness by generation for NN NK landscapes using only recombination.

Figure 6 shows GA performance when using selected informed and random recombination operators on the NK landscape problem when provided with various quality models. Figure 6a shows the results obtained with a very poor model. As expected, the 2-point crossover does not outperform the other operators as the problem characteristics changed. In fact, in this case (a) the uniform crossover seems to outperform the other operators as the information in the model is misleading. However, CiCe performs surprisingly well for an informed operator based on bad information. However, this operator swaps 50% of the bits from the two

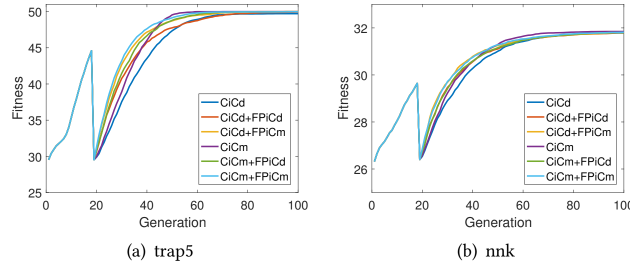


Figure 7: Maximum fitness by generation for trap5 and nnk using recombination and mutation after 18 generations of hBOA model building.

parents, becoming similar to the uniform crossover. Also, while uniform and CiCe do the best over time, they do poorly in comparison to the other operators at the beginning of the runs.

In cases (b) and (c) the informed operators catch up and exceed the quality of the uniform crossover, as the models become more reliable. Overall, CiCe is the best performing operator. CiCd and CiCm initially do well but eventually plateau.

Finally in case (d), the hBOA model is much better even though it is far from good (the GA still clearly outperforms the hBOA alone thus the hBOA model was not complete yet). Now Um and CiCm both perform nearly equally with this level of model quality, beating out all the other operators during the entirety of the runs. CiCe, CiCd and Ud are nearly equal in performance, starting out worse than Um and CiCm, but managing to reach nearly the same average fitness given enough generations. The two worst operators are the random operators, with two-point doing slightly better early on but then uniform surpassing its performance later.

5.6 Combined results

In Section 5.4 and Section 5.5 it was observed that given sufficient model quality, the informed operators (both mutation and recombination) performed well individually, clearly outperforming the random operators (except when specific problem characteristics match the operator). But even though a GA needs both mutation and crossover, it needs them both together to fully succeed. Therefore, in this section we pair the best crossovers with the best mutations to see if in combination they can indeed outperform the singular runs.

CiCd and CiCm were the strongest recombination operators across NNK and trap-5 instances, so we selected them for this experiment. For mutation, we picked FPiCd and FPiCm, as they were the overall winners. Due to the noisiness of the results, in the following results 100 instances for each parameter setting for trap 5 were used. We show results only up to 100 generations, for more detail. We also only show the results for the best models (18 hBOA generations) as this is when the informed operators performed best.

Figure 7 shows the GA results for both trap5 (a) and nnk (b). As seen, the combinations of crossover and mutations improve on the results with one operator at a time. The combinations of operators based on the markov cover seem to outperform those based on single covers, in both cases.

6 SUMMARY AND CONCLUSIONS

We have defined generic mutation and crossover operators that do account for some problem-specific information presented to them in the form of hBOA model. Using two standard problems with various characteristics, we have shown that the so-informed operators outperform the standard random operators as long as the information given to the operators is fairly reliable even if not perfect. We have then shown that a mix of such operators will outperform these operators alone.

With these results in place, we validated the assumption that a generic information model, hBOA model here, can be useful to create GAs which will outperform standard random GAs. This can be useful if building domain/problem specific representations and operators is costly or difficult. The next step will be to perform more empirical analysis to better assess and understand various properties of these operators, and the impact of model quality on their performance. Finally, with such informed operators in place, the final step will be to assess if a combination of hBOA and such informed GA can be a good alternative to hBOA alone.

REFERENCES

[1] D. H. Ackley. An empirical study of bit vector function optimization. *Genetic Algorithms and Simulated Annealing*, pages 170–204, 1987.
[2] S. Baluja. Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA, 1994.
[3] K. Deb and D. E. Goldberg. Analyzing deception in trap functions. IlliGAL Report No. 91009, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1991.
[4] D. E. Goldberg. *The design of innovation: Lessons from and for competent genetic algorithms.* Kluwer, 2002.
[5] D. E. Goldberg and R. Lingle, Jr. Alleles, loci, and the traveling salesman problem. *Proc. of the International Conf. on Genetic Algorithms and Their Applications*, pages 154–159, 1985.
[6] C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13(2):189–228, 1993.
[7] H. Kargupta. The gene expression messy genetic algorithm, 1996.
[8] S. Kauffman. Adaptation on rugged fitness landscapes. In D. L. Stein, editor, *Lecture Notes in the Sciences of Complexity*, pages 527–618. Addison Wesley, 1989.
[9] S. A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, 1993.
[10] P. Larrañaga and J. A. Lozano, editors. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation.* Kluwer, Boston, MA, 2002.
[11] Z. Michalewicz and C. Janikow. Handling constraints in genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 151–157. Morgan Kaufmann, 1991.
[12] H. Mühlenbein and G. Paaß. From recombination of genes to the estimation of distributions I. Binary parameters. *Parallel Problem Solving from Nature*, pages 178–187, 1996.
[13] J. Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann, San Mateo, CA, 1988.
[14] M. Pelikan. *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms.* Springer-Verlag, 2005.
[15] M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, 21(1):5–20, 2002.
[16] M. Pelikan, K. Sastry, M. V. Butz, and D. E. Goldberg. Performance of evolutionary algorithms on random decomposable problems. In *PPSN*, pages 788–797, 2006.
[17] M. Pelikan, K. Sastry, and E. Cantú-Paz, editors. *Scalable optimization via probabilistic modeling: From algorithms to applications.* Springer-Verlag, 2006.
[18] 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. MEDAL Report No. 2009002, Missouri Estimation of Distribution Algorithms Laboratory, University of Missour–St. Louis, St. Louis, MO, 2009.
[19] A. H. Wright, R. K. Thompson, and J. Zhang. The computational complexity of n-k fitness functions. *IEEE Trans. Evolutionary Computation*, 4(4):373–379, 2000.