

**ClusterMI: Building Probabilistic Models using
Hierarchical Clustering and Mutual Information**

**Thyago S. P. C. Duque
David E. Goldberg
IlliGAL Report No. 2009002**

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

ClusterMI: Building Probabilistic Models using Hierarchical Clustering and Mutual Information

Thyago S. P. C. Duque David E. Goldberg

Illinois Genetic Algorithms Laboratory,
University of Illinois at Urbana Champaign,
104 S. Mathews Ave, 117 Transportation Bldg, Urbana, IL
{thyago, deg}@illigal.ge.uiuc.edu

Abstract

Genetic Algorithms are a class of metaheuristics with applications on several fields including biology, engineering and even arts. However, simple Genetic Algorithms may suffer from exponential scalability on hard problems.

Estimation of Distribution Algorithms, a special class of Genetic Algorithms, can build complex models of the iterations among variables in the problem, solving several intractable problems in tractable polynomial time. However, the model building process can be computationally expensive and efficiency enhancements are oftentimes necessary to make tractable problems practical.

This paper presents a new model building approach, called ClusterMI, inspired both on the Extended Compact Genetic Algorithm and the Dependency Structure Matrix Genetic Algorithm. The new approach has a more efficient model building process, resulting in speed ups of 10 times for moderate size problems and potentially thousands of times for large problems. Moreover, the new approach may be easily extended to perform incremental evolution, eliminating the burden of representing the population explicitly.

1 Introduction

Evolutionary Algorithms (EA) (Goldberg, 1989b) (Holland, 1975) have been successfully used in several different applications involving search, optimization and machine learning problems. Goldberg (Goldberg, 2002) presents a design-decomposition methodology for successfully designing scalable Genetic Algorithms (GAs). A GA that can solve hard problems accurately, efficiently and reliably is called a competent GA. These GAs can solve problems that are intractable for traditional methods in a tractable polynomial time.

Estimation of Distribution Algorithms (EDA) (Larrañaga & Lozano, 2001) have been particularly successful on solving hard problems competently. These algorithms build probabilistic models that summarize the information of the current population, and then sample these models to generate a new population. By using complex models like Bayesian Networks or Marginal Product Models (MPM), these algorithms can learn the structure of the problem and use this information to guide the search.

The Extended Compact Genetic Algorithm (ECGA) (Harik, 1999) is one of such EDAs, and it uses the MPM to model non-overlapping iteration among variables that form a Building Block (BB) (Goldberg, 1989b) (Holland, 1975).

Although EDAs scale polynomially, they are still computationally expensive. In order to solve large problems it is oftentimes necessary to enhance the efficiency (Sastry, Goldberg, & Pelikan, 2004) of the algorithm using techniques like parallelization (Cantu-Paz, 1999), hybridization (Goldberg & Voessner, 1999) (Sinha & Goldberg, 1999) (Sinha, 2003), time continuation (Sastry & Goldberg, 2004), evaluation relaxation (Sastry, 2001) and incremental evolution (Harik, Lobo, & Goldberg, 1998), (Pelikan, Sastry, & Goldberg, 2008).

Particularly, the model building process of the ECGA is time consuming. This issue can be partially addressed using a cache structure (de la Ossa, Sastry, & Lobo,), relaxing the model building process (Duque, Goldberg, & Sastry, 2008) or performing sporadic model building (Pelikan, Sastry, & Goldberg, 2006). In this paper we propose a new model building algorithm, called ClusterMI, that aims on reducing the computational complexity of the model building process. Moreover, the new model building mechanism may be used to perform incremental model building, which can further improve the performance of the algorithm and will be explored on future work.

This paper is organized as follows: Section 2 presents a introduction to the ECGA and the Dependency Structure Matrix Genetic Algorithm (DSMGA), section 3 presents ClusterMI, a new approach for model building in EDAs that combine ideas from both ECGA and DSMGA. Section 4 presents the results of the new approach and compares it with the ECGA. Section 5 discusses future work, including initial ideas for incremental evolution. Section 6 presents final considerations and concludes the paper.

2 Background

This section briefly reviews the ECGA and the DSMGA, discussing computational aspects of the model building process on these algorithms.

As an EDA, the ECGA (Harik, 1999) follows the template presented on algorithm 1. Particularly, the ECGA uses the MPM as a model of the current population. The MPM has two components: (I) a partition over the variables, defining which variables are independent and which variables are linked, and (II) a probability distribution over each partition.

Algorithm 1 A template for EDAs

1. Start with random population.
 4. Do:
 5. Evaluate the population
 6. Select a group of good individuals
 7. Build a model to summarize the population
 8. Sample the built model
 5. while (stop criteria not met)
-

The model building process of the ECGA finds a partition that appropriately represents the population by greedily optimizing the Combined Complexity Criterion (CCC) (Harik, 1999) using algorithm 2. The CCC reflects the minimum description length (MDL) bias of the ECGA and it is presented in equation (1), where n represents the population size, m the number of partitions, M_i the i -th partition and $E(M_i)$ the entropy of the partition M_i . Observe that calculating the entropy involves estimating the probability distribution over the partition, an $O(n)$ step.

$$CCC = n \cdot \log(n) \cdot \sum_{I=0}^m E(M_i) \cdot 2^{Size[M_i]} \quad (1)$$

Algorithm 2 Greedy search for an appropriated model in the ECGA

```
1. Start with all independent variables.
2. improvement = IMPOSSIBLE
3. best = CCC(current model)
4. Do:
5.   For each partition [i]
6.     For each partition [j], j not equals i
7.       i+j = Merge i and j
8.       ccc = CCC(i+j)
9.       if (ccc < best)
10.        (bi, bj) = save(i, j)
11.        best = ccc
12.        improvement = POSSIBLE
13.   if (improvement = POSSIBLE)
14.     Merge the saved (bi, bj)
15.     Update current model
16. while (improvement = POSSIBLE)
```

The greedy search for the best partition requires $O(\ell^3)$ evaluations of the CCC criterion (ℓ indicates the problem size, see figure 3 for empirical evidence or (Duque, Goldberg, & Sastry, 2008) for a complexity analysis). For deceptive functions (Goldberg, 2002) the algorithm can be relaxed to require $O(\ell^2)$ (Duque, Goldberg, & Sastry, 2008) without losing accuracy, but it is unknown whether this result holds for other problems.

Another common way to speed-up the model building is the use of a cache structure (as implemented in (de la Ossa, Sastry, & Lobo,)). Algorithm 3 implements the greedy search using a cache structure. This algorithm is guaranteed to produce the same results as the algorithm 2, since it only replaces the calculation of the CCC by a cache lookup.

Algorithm 3 requires $O(\ell^2)$ evaluations of the CCC criterion (see figure 3). These come from line 5, which is executed $O(\ell^2)$ times and line 23, also executed $O(\ell^2)$ times. However, there is the extra cost of managing the cache. Assuming a problem with size ℓ and building blocks of size k , when the search for the best partition begins at line 8, the cache will have $\ell * (\ell - 1)/2$ entries. When the algorithm finishes its execution at line 25, the cache will have $\ell/k * (\ell/k - 1)/2$ entries of size $O(k)$. Consequently, the size of the cache ranges from $O(\ell^2)$ to $O(\ell^2/k)$. That implies an additional memory requirement almost the size of the population itself.

This work also draws inspiration from the DSMGA (Yu, 2006) (Yu & Goldberg, 2006), a competent GA that uses a Dependency Structure Matrix (DSM) to model the relationship among variables. A clustering method is used to determine groups of variables that strongly interact among each other, forming a BB. To do so, the DSMGA uses a search method and a MDL bias: minimizing the cost of describing a given DSM. Afterwards, the information from the clustering is used to perform BB-Wise crossover.

In the DSMGA, a binary string with length $M_{max} \cdot n_c$ is used to represent a clustering arrangement, with n_c as the number of variables and M_{max} as the maximum number of modules. In this representation, the $(x + yn_c)$ -th bit indicates whether or not the x -th bit belongs to the y -th cluster. Initially a simple GA was used to learn the BB structure, using the described representation as chromosome and the MDL metric as objective function. Later on a binary hillclimber was used to speed up the BB discovery process.

Algorithm 3 The use of a cache structure in the model building for ECGA

```
1. Start with all independent variables.
2. For each partition [i]
3.   For each partition [j], j not equals i
4.     i+j = Merge i and j
5.     ccc = CCC(i+j)
6.     cache.store(i, j, ccc)
7. improvement = IMPOSSIBLE
8. Do:
9.   best = CCC(current model)
10.  For every entry (i, j, ccc) in the cache
11.    if (ccc < best)
12.      improvement = POSSIBLE
13.      best = ccc
14.      (bi, bj) = save (i, j)
15.  if (improvement = POSSIBLE)
16.    Merge the saved (bi, bj)
17.    Update the current model
19.  For every entry (i, j, ccc) in the cache
18.    if (i = bi or i = bj or j = bi or j = bj)
19.      cache.remove(i, j, ccc)
21.  For each partition [k]
22.    bi+bj+k = Merge bi+bj and k
23.    ccc = CCC(bi+bj+k)
24.    cache.store(bi+bj, k, ccc)
25. while (improvement = POSSIBLE)
```

Finally, it is important to notice that both the ECGA and the DSMGA build the models using the Shannon’s entropy (Shannon, 1948). As shown by (Yu, Sastry, Goldberg, & Pelikan, 2007) these methods require a population of size $O(2^k \ell \log(\ell))$ to accurately build the model. Also, it is known that the DSMGA requires larger populations than the ECGA.

3 ClusterMI: A new approach to model building in EDAs

This section describes our approach for model building in EDAs and how it relates with ECGA and DSMGA. We also discuss computational aspects of the new algorithm, showing how it can be superior or inferior to its competitors.

This paper proposes the use of hierarchical clustering and Mutual Information (MI) to perform model building in the ECGA. The new model building algorithm, called ClusterMI (from clustering over mutual information) can be divided in two steps.

First, a matrix is created to store the degree of dependency between every two pairs of variables. The MI between two variables is used for this purpose and it is defined in equation (2). When two variables are completely unrelated (independent), the MI assumes its minimal value of 0. On the

other extreme, the MI between a variable and itself is the entropy of that variable.

$$MI(X, Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left(\frac{p(x, y)}{p(x)p(y)} \right) \quad (2)$$

After the MI between every pair of variables is calculated, a clustering method (presented on algorithm 4) is used to determine the best partition of variables. Algorithm 4 is a modified hierarchical clustering algorithm that uses the CCC from ECGA to decide the optimal number of clusters. This algorithm uses the MI (as opposed to the CCC in the ECGA) as a metric to decide which pair of partitions/clusters to merge, and relies on the CCC only to determine whether or not the chosen pair will actually be merged.

Algorithm 4 ClusterMI: a simple clustering algorithm to search for the best partition of variables in the ECGA

```

1. Start with all independent variables.
2. improvement = IMPOSSIBLE
3. Do:
4.   best_mi = -1
5.   For each cluster [i]
6.     For each cluster [j], j not equals i
7.       mi = MI_Cluster(i, j)
8.       if (mi > best_mi)
9.         best_i = i
10.        best_j = j
11.        best_mi = mi
12.   if (best_mi not equals -1)
13.     bi+bj = Merge best_i and best_j
14.     cccM = CCC(model with bi+bj merged)
15.     cccC = CCC(current model)
16.     if (cccM < cccC)
17.       Update the model accepting bi+bj merge
18.       improvement = POSSIBLE
19. while (improvement = POSSIBLE)

```

We already defined the MI between a pair of variables (equation (2)), however, to properly perform clustering we need to define a similarity measure between groups of variables. Line 7 of algorithm 4 extrapolates the notion of MI to represent the similarity for clusters of variables. The “MI” between two clusters is defined as the average of the MI between every variable of one cluster and every variable of the other, as presented in equation (3).

$$MI_Cluster(I, J) = \frac{1}{|I| + |J|} \sum_{i \in I} \sum_{j \in J} MI(i, j) \quad (3)$$

The model building approach defined by algorithm 4 can be used in step 7 of algorithm 1. The clustering of variables resulting from algorithm 4 is equivalent to a partition of variables in the MPM. The same procedure used to estimate the distribution over partitions can be used to estimate the distribution over the clusters, and the same sampling procedure used for ECGA can be used for ClusterMI. In this sense, ClusterMI can be viewed as an ECGA with a modified greedy search for the best partition.

As for the relation between DSMGA and ClusterMI, both of them use a matrix storing the MI between variables to perform linkage learning. The main difference is that the DSMGA converts the MI matrix into a binary relation (dependent or non-dependent) and uses an MDL metric based on how well the model represents the binary matrix. On the other hand, ClusterMI uses the MI matrix to establish the order in which variables and clusters should be merged, using ECGA’s MDL bias (CCC) to decide when to stop the process. Also, in DSMGA clustering is a blind optimization problem over a binary string while ClusterMI uses an explicit hierarchical clustering algorithm to perform the model building.

Concerning computational complexity of algorithm 4, three aspects will be taken into account. First, algorithm 4 has complexity of $O(\ell^3)$ with respect to the number of evaluations of the similarity among clusters (operation performed on line 7). Observe that lines 1 through 12 of algorithm 4 implement a functionality similar to the one of lines 1 through 13 of algorithm 2 and the same cache structure can be used to reduce it to $O(\ell^2)$ ¹.

Second, algorithm 4 has complexity $O(\ell)$ with respect to the number of evaluations of the CCC (empirical evidence provided on figure 3). It is clear that concerning evaluations of CCC, ClusterMI is superior to both ECGA and ECGA with cache ($O(\ell)$ v.s. $O(\ell^3)$ and $O(\ell^2)$).

Finally, the memory complexity of algorithm 4 is $O(\ell^2)$, a result from the need to store the MI among every pair of variables (the MI could be calculated inside the algorithm, but that would render it inefficient). A cache structure to speed up the search for the most similar pair of clusters would require additional memory of order $O(\ell^2)$, still resulting in an overall memory complexity of $O(\ell^2)$.

To show that the complexity of ClusterMI is smaller than the complexity of the model building in ECGA (with cache) we still need to show that $O(\ell^2)$ evaluations of the similarity among clusters is better than $O(\ell^2)$ evaluations of the CCC. As mentioned earlier, calculating the CCC involves calculating the entropy of a partition. Assuming a problem of size ℓ , composed of m BBs of size k and a population size $n = O(2^k \cdot \ell \cdot \log(\ell))$, this step has complexity $O(k \cdot 2^k \cdot \ell \cdot \log(\ell))$. On the other hand, evaluating the similarity among clusters involves averaging over the pairwise distance for every pair of variables. Since the MI between the variables is stored, this step has complexity $O(k^2)$ (assuming BBs of size k and population properly sized, allowing ClusterMI to build the correct model).

In conclusion, ClusterMI has computational complexity of $O(k \cdot 2^k \cdot \ell^2 \cdot \log(\ell))$ and memory complexity of $O(\ell^2)$. The model building in the ECGA (with cache) has computational complexity of $O(k \cdot 2^k \cdot \ell^3 \cdot \log(\ell))$ and memory complexity of $O(\ell^2)$. Overall, ClusterMI is one order of complexity faster while holding the same memory complexity of ECGA.

We still need to determine how ClusterMI scales in relation to population size and number of generations. DSMGA is known to have the same scalability of ECGA, requiring larger population (by a constant factor). In the next section we present empirical evidence that indicates that ClusterMI also scales as well as ECGA, requiring larger population. This is an expected phenomenon, since ClusterMI uses the same information as DSMGA: the MI between variables.

4 Results

In this section we present the results obtained using ClusterMI. In all results presented in this paper the benchmark function used is the mk -trap (Sastry & Goldberg, 2004) with or without Gaussian noise. The mk -trap is an additively separable deceptive function (Goldberg, 1989a) composed of m trap functions of k variables. Equation (4) presents the used trap function, where u is the number

¹The algorithm will be very similar to algorithm 3, and was omitted.

of bits with value 1 in the BB. For noisy objective functions, a Gaussian noise with mean 0 and variance equals 5% of the maximum fitness was randomly sampled and added to the fitness of each individual.

$$\text{trap}(u) = \begin{cases} 1, & \text{if } u = k \\ 0.8 * \frac{(k-1-u)}{(k-1)}, & \text{otherwise,} \end{cases} \quad (4)$$

The proposed method will be evaluated for scalability using four performance measures: population size, number of function evaluations, number of evaluations of the CCC and overall computational time required for convergence. We compare the ClusterMI with the ECGA with and without cache. In the experiments reported, we used the implementation of the ECGA by Fernando Lobo (de la Ossa, Sastry, & Lobo,) as the ECGA with cache. This implementation was modified to incorporate a version of the ECGA without cache and a version of ClusterMI. Since the three codes are built over the same framework, code optimization issues can be ignored in comparative results. All three methods used tournament selection with tournament size of 16.

In the experiments, the bisection method (Sastry, 2001) was used to determine the minimal population size required to correctly solve at least $m - 1$ subproblems 29 out of 30 times. The bisection method is an empirical method for population sizing, and usually produces a good approximation to the theory.

In order to provide a clear presentation on the results, ECGAwc will refer to the ECGA with cache, ECGAwo will refer to the ECGA without cache. ECGA will be used to represent both ECGAwc and ECGAwo when no distinction is necessary.

Three sets of experiments will be reported in this section. The first set of experiments compares the performance of ECGAwc, ECGAwo and ClusterMI on a 4 bit trap problems ($k = 4$) of varying number of BBs. Fifteen runs of the bisection method were used to determine the minimum population size necessary to solve problems of sizes up to 160 with ECGAwo, 256 with ECGAwc and 512 with ClusterMI. These results were used to predict the population size used on the remaining of this set of experiments.

Figure 1 compares the population sizes required by ECGA and ClusterMI, showing the scalability of both methods, figure 2 compares the number of function evaluations required by ECGA and ClusterMI, figure 3 shows the cost of model building in ECGA with cache, ECGA without cache and ClusterMI and figure 4 compares the overall running time in seconds for all three methods.

As can be observed, both ClusterMI and ECGA have the same scalability concerning population size (Figure 1). ClusterMI, however, requires slightly larger populations, an expected phenomenon due to its relation with DSMGA. The population size reflects directly in the number of function evaluations and once again both ECGA and ClusterMI have the same scalability (Figure 2). When the model building is considered ClusterMI strongly outperform ECGA both with and without cache (Figure 3).

The relatively small difference in population size is counterbalanced with the large difference in model building efficiency, resulting in a significantly better performance for ClusterMI when compared with ECGA (Figure 4). The improvement is dependent on the relative computational cost of the function evaluation. For computationally expensive function evaluations the ECGA is expected to outperform ClusterMI in relatively small problems. However, the extra population size reflects a constant disadvantage while the model building efficiency reflects a complexity reduction. Consequently, for any function evaluation there is always a problem large enough so that ClusterMI's speedup in model building will surpass the overhead caused by the larger population size.

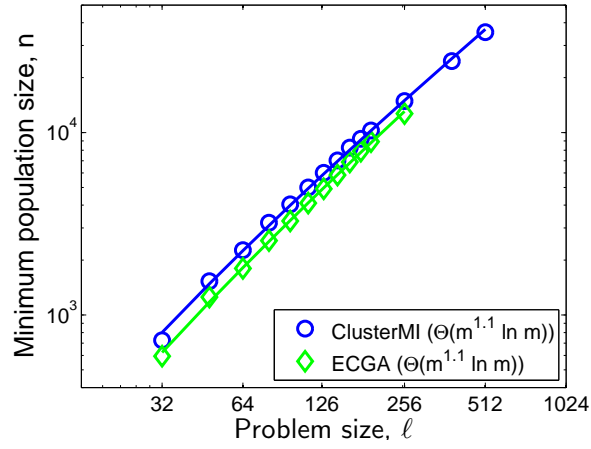


Figure 1: A comparison between the ECGA and ClusterMI shows that both algorithms have the same scalability, with ClusterMI using slightly larger population.

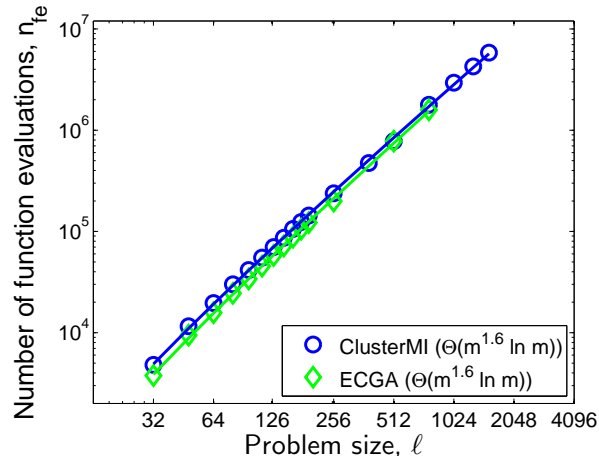


Figure 2: The comparison between the number of function evaluations required by the ECGA and ClusterMI shows that both methods have the same scalability.

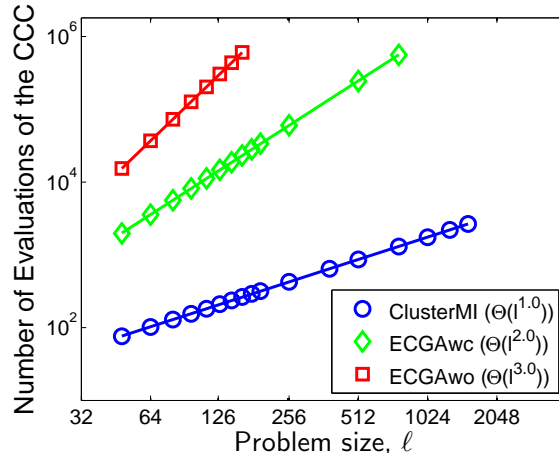


Figure 3: Considering the number of evaluations of the CCC necessary to accurately build the model ClusterMI outperforms its competitors both in absolute values and in scalability. ClusterMI scales one order of complexity faster than ECGAwc and two orders of complexity faster than ECGAwo.

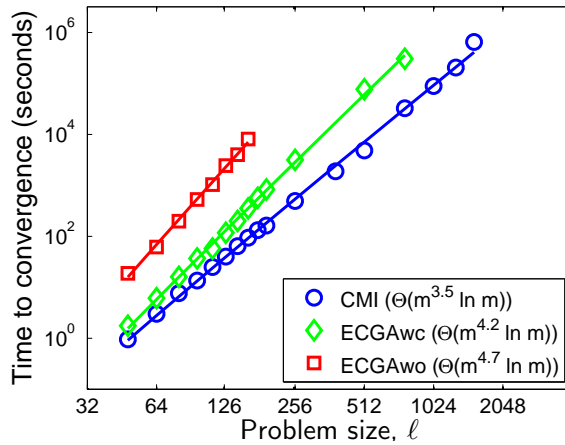


Figure 4: When absolute running time is used as the performance measure, ClusterMI clearly outperforms both ECGAwc and ECGAwo. ClusterMI is not only faster in absolute values, but also scales better by almost one order of complexity when compared ECGAwc and more than one order of complexity when compared to ECGAwo.

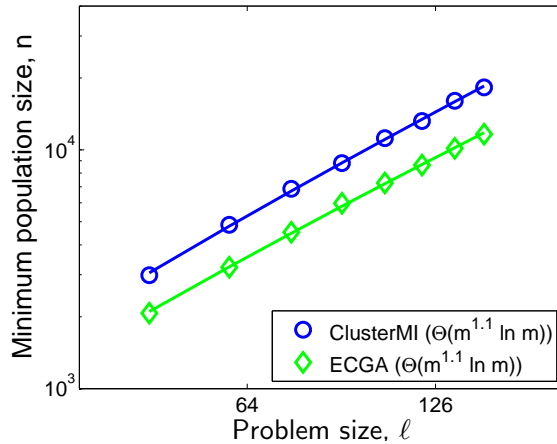


Figure 5: The scalability of ECGA and ClusterMI concerning population size is held constant when the size of the trap function is increased. Once again ClusterMI requires a larger population size than ECGA.

The second set of experiments uses 5 bits trap to compare ECGA and ClusterMI on a harder problem. On this experiment we only used ECGA with cache since it is the most efficient implementation. Fifteen runs of the bisection method were used to determine the minimum population size required to solve problems of size up to 150. Figure 5 presents a comparison between the minimum population size required by the ECGA and ClusterMI and figure 6 presents the running time for each method. A comparison of the number of function evaluations was omitted since it can be derived from figure 5. The number of evaluations of CCC was also omitted since it is equivalent of what was presented on figure 3.

The third set of experiments is based on 4 bits trap functions with additive Gaussian noise of 5% of the maximum fitness. Fifteen runs of the bisection method were used to determine the minimum population size required by ECGA and ClusterMI to solve problems of size up to 160. Figure 7 compares both methods concerning population size and figure 8 compares them using running time as the performance measure. The number of function evaluations and evaluations of CCC were omitted for the same reasons as in the second set of experiments.

The results observed on the second and the third sets of experiments are similar to those presented on the first experiment. ECGA and ClusterMI still present the same scalability concerning population size, with ClusterMI requiring larger population size. However, ClusterMI is still more efficient in the model building process resulting in faster running times. Once again there is a trade-off between the use of ECGA and ClusterMI, and once again ECGA will only outperform ClusterMI for relatively small problems with computationally expensive function evaluations.

This section showed that ClusterMI has the same scalability of ECGA concerning population size and number of function evaluations. However, ClusterMI is one order of complexity more efficient in the model building process resulting in an improvement of at least half an order of complexity for overall running time. The next section presents perspectives of future work, including possible extensions to allow for incremental evolution.

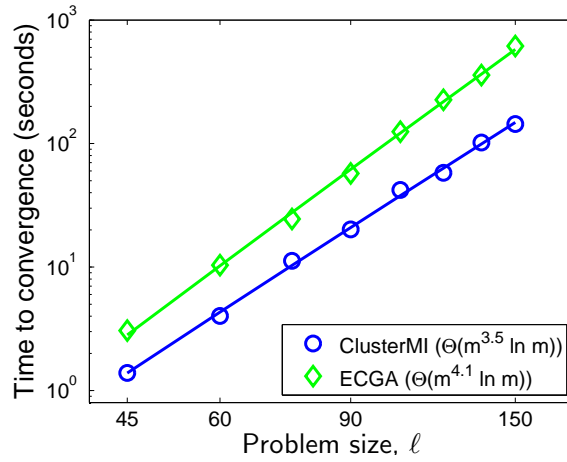


Figure 6: When absolute running time is considered, ClusterMI still outperforms ECGA both in absolute values and in scalability.

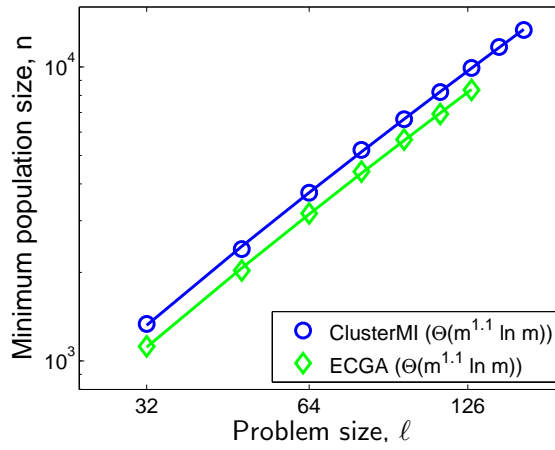


Figure 7: The scalability of ECGA and ClusterMI also remains constant for functions with additive Gaussian noise. Once again ClusterMI requires a larger population size than ECGA.

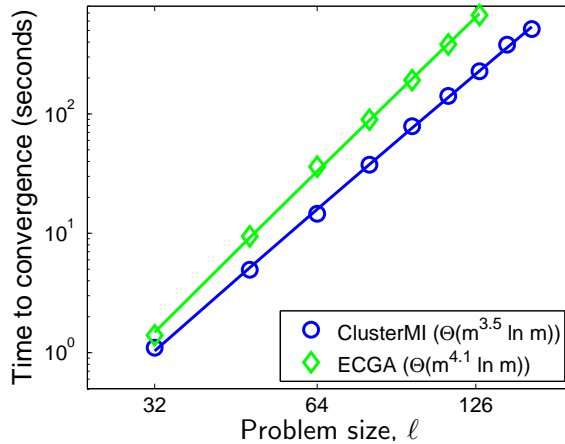


Figure 8: When absolute running time is considered, ClusterMI still outperforms ECGA both in absolute values and in scalability.

5 Future Work

This paper presents ClusterMI, a new approach to perform model building on EDAs. ClusterMI uses the MI between pairs of variables to choose which partitions to merge, based on a greedy hierarchical clustering. Although ClusterMI may produce speed ups of thousands of times, the perspective of future work is even more promising. Particularly, ClusterMI can be used to perform incremental evolution, excluding the need to explicitly representing the population and thereby reducing memory usage or even communication cost for parallel implementations. This section presents initial ideas for incremental evolution. Other topics that will be addressed in future works are also outlined.

To build a model of the population we need the MI between every pair of variables, which requires the knowledge of the individual probability distribution for each variable, as well as the joint probability distribution for every pair of variables. It is easy to update both distributions incrementally, using a procedure similar to the one used on (Harik, Lobo, & Goldberg, 1998) or (Pelikan, Sastry, & Goldberg, 2008).

$$P(x_1, \dots, x_n, y) = \frac{P(x_1, \dots, x_n) * J(y)}{\sum_{\bar{y} \in Y} J(\bar{y})} \quad (5)$$

where,

$$J(y) = \prod_{i=1}^n P(X_i = x_i, Y = y)$$

We also need to calculate joint probability distributions of sets of variables that represent a partition or cluster. These distributions are used for sampling and on the calculation of the CCC. Estimating multivariate distributions is a common problem for incremental EDAs. iBoa solves this problem by adding one variable at a time to the distribution and by assuming the variables are independent (Pelikan, Sastry, & Goldberg, 2008). Similarly, we propose the use of induction to construct multivariate probability distributions by adding one variable at a time to an already estimated distribution. The induction starts with a pairwise distribution and incrementally adds variables until the desired distribution is estimated. Instead of assuming independence among variables, given

a joint distribution $P(X_1, \dots, X_n)$, we can estimate the joint distribution $P(X_1, \dots, X_n, Y)$ using equation (5).

Other topics worth of investigation include:

- The hybridization of ClusterMI with a preprocessing local search to reduce the required population size (Duque, Goldberg, & Sastry, 2008);
- Development of a parallel model building algorithm;
- Development of an incremental model building algorithm, which would consist of small incremental changes on previous models reducing the model building cost even further.
- Application of ClusterMI to real world problems and to large scale problems.

6 Conclusion

Some EDAs are among the most versatile problem solvers available, being able to solve problems that are intractable for traditional methods in a tractable polynomial time. However, EDAs may still be too time consuming to be practical. Several efficiency enhancement techniques have been proposed in the literature to turn EDAs from tractable to practical. This work focus on the model building, a time consuming but necessary process for EDAs. We propose the use of hierarchical clustering to build an MPM representing the current population.

The proposed algorithm is called ClusterMI and it is related both to the ECGA and to the DSMGA. Consequently, ClusterMI have advantages and disadvantages drawn from both algorithms. Specifically, ClusterMI produces a comprehensive and human interpretable model that can be used to identify BBs on additively separable functions. The scalability of ClusterMI is equivalent to the scalability of ECGA and DSMGA, however, both ClusterMI and DSMGA require larger population sizes, mostly due to the use of MI to guide the model building decisions.

The advantage of ClusterMI relies on the efficient model building process, which is one order of complexity faster than the ECGA with cache, resulting in faster overall running times. The ECGA is expected to outperform ClusterMI on relatively small problems with computationally expensive function evaluations due to the smaller population size. However, the difference in population size is constant, while the improvement in the model building reduces the complexity of the algorithm. Consequently, for any function evaluation there is a problem large enough so that the improvement in model building compensates the extra evaluations and ClusterMI outperforms ECGA.

ClusterMI can produce speed ups of 10 times in moderate size problems (768) and potentially thousands of times for large scale problems (projected speed up for a problem with 2^{20} variables is 1515). Moreover, ClusterMI may be used to perform incremental model building or even incremental evolution, eliminating the need to explicitly represent the population and possibly improving the performance even further.

References

- Cantu-Paz, E. (1999). *Designing Efficient and Accurate Parallel Genetic Algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign. Illigal Report No 99017.
- de la Ossa, L., Sastry, K., & Lobo, F. χ -ary *Extended Compact Genetic Algorithm in C++* (Technical Report). Illigal Report 2006013, Illinois Genetic Algorithms Lab, University of Illinois at Urbana-Champaign, 2006.

- Duque, T., Goldberg, D., & Sastry, K. (2008). Enhancing the Efficiency of The ECGA. *Proceedings of the X Parallel Problem Solving From Nature (PPSN2008): Dortmund, Germany, September 13-17, 2008*, 165.
- Goldberg, D. (1989a). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, 3(2), 153–171.
- Goldberg, D. E. (1989b). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- Goldberg, D. E. (2002). *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers.
- Goldberg, D. E., & Voessner, S. (1999). Optimizing Global-Local Search Hybrids. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Volume 1 (pp. 220–228). Morgan Kaufmann.
- Harik, G. (1999). *Linkage Learning via probabilistic modeling in the ECGA* (Technical Report). Urbana, IL: University of Illinois at Urbana Champaign.
- Harik, G., Lobo, F., & Goldberg, D. (1998). The compact genetic algorithm. In *Proceedings of 1998 IEEE International Conference on Evolutionary Computation* (pp. 523–528).
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. The MIT Press.
- Larrañaga, P., & Lozano, J. A. (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers.
- Pelikan, M., Sastry, K., & Goldberg, D. (2006). Sporadic model building for efficiency enhancement of hierarchical BOA. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 405–412). ACM Press New York, NY, USA.
- Pelikan, M., Sastry, K., & Goldberg, D. (2008). iBOA: The Incremental Bayesian Optimization Algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2008): Atlanta, GA, USA*.
- Sastry, K. (2001). *Evaluation-relaxation Schemes for Genetic and Evolutionary Algorithms*. Doctoral dissertation, University of Illinois at Urbana-Champaign.
- Sastry, K., & Goldberg, D. (2004). Let’s Get Ready to Rumble: Crossover Versus Mutation Head to Head. *Genetic and Evolutionary Computation–GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004: Proceedings*.
- Sastry, K., Goldberg, D., & Pelikan, M. (2004). *Efficiency enhancement of probabilistic model building genetic algorithm* (Technical Report). Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana Champaign, Urbana, IL.
- Shannon, C. (1948). A mathematical theory of communication *Bell Syst. Tech. J.*, 27(3), 379–423.
- Sinha, A. (2003). Designing efficient genetic and evolutionary algorithm hybrids. Master Thesis, University of Illinois at Urbana Champaign (IlliGal Report No. 2003020).
- Sinha, A., & Goldberg, D. (1999). *A survey of hybrid genetic and evolutionary algorithms* (Technical Report). Urbana, IL: University of Illinois at Urbana Champaign. IlliGal Report No. 2003004.
- Yu, T. (2006). *A Matrix Approach for Finding Extreme: Problems with Modularity, Hierarchy, and Overlap*. Doctoral dissertation, Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

- Yu, T., & Goldberg, D. (2006). Conquering hierarchical difficulty by explicit chunking: sub-structural chromosome compression. *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 1385–1392.
- Yu, T., Sastry, K., Goldberg, D., & Pelikan, M. (2007). Population sizing for entropy-based model building in discrete estimation of distribution algorithms. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation* (pp. 601–608). ACM Press New York, NY, USA.