Messy Genetic Algorithms for Subset Feature Selection

D. Whitley, J. R. Beveridge, C. Guerra-Salcedo, C. Graves
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
(303) 491-5373
whitley, ross, guerra, gravesc@cs.colostate.edu

Abstract

Subset Feature Selection problems can have several attributes which may make Messy Genetic Algorithms an appropriate optimization method. First, competitive solutions may often use only a small percentage of the total available features; this can not only offer an advantage to Messy Genetic Algorithms, it may also cause problems for other types of evolutionary algorithms. Second, the evaluation of small blocks of features is naturally decomposable. Thus, there is no difficulty evaluating underspecified strings. We apply variants of the Messy Genetic Algorithm to a application in computer vision with very good results. We also apply variants of the Fast Messy Genetic Algorithm to synthethic test problems.

Keywords: messy genetic algorithms, subset feature selection, computer vision

1 Subset Feature Selection

The subset feature selection problem occurs in several domains, including machine learning and computer vision. In machine learning, many features may be available as potential inputs to a learning system. Learning is often faster and potentially more robust if the set of inputs can be reduced to a subset which captures all or most of the information contained in the larger feature set. Applications are found in the construction of decision trees (Bala et al. 1995) and neural networks (Brill et al. 1992).

Messy Genetic Algorithms (Goldberg et al. 1989) are well suited to some types of subset feature selection problems. Messy Genetic Algorithms allow variable-length strings that may be underspecified or overspecified with respect to the problem being solved. A messy gene is an pair: $(GeneNumber, AlleleValue)$. The messy chromosome is a collection of messy genes. For example $((5,0)(1,1)(2,0)(1,0))$ is a chromosome with 3 genes. This chromosome is overspecified since gene 1 has two different allele values: 0 and 1. A messy chromosome may also be underspecified in that not all chromosomes have allele values for all possible genes. In this case, genes 3 and 4 (and perhaps others) are not represented. One difficulty with Messy Genetic Algorithms is that relatively complex methods for evaluating underspecified strings must be used.

When the Messy Genetic Algorithm is applied to subset feature selection problems, it is sometimes convenient to modify the algorithm. Rather than sampling subsets of genes which may have allele value 0 or 1, we can sample small subsets of features. In effect, we only generate subsets of genes that have allele value 1. All unspecified genes are assumed to have allele value 0. In this case, evaluation is simple. In the next section, we present a subset feature selection application in computer vision that uses this modified form of Messy Genetic Algorithm.

Messy Genetic Algorithms work well on our computer vision application in part because the best solutions tend to use only a small subset of available features. Hence solutions tend to be "sparse" with the majority of bits being set to zero; this causes serious problems for algorithms such as CHC [Whitley et al., 1995]. The second example in this paper applies variants of the Fast Messy Genetic Algorithm (FMGA) to synthetic subset selection problems previously studied by Radcliffe and George (1993) and Crawford et al. (1997). as well as deceptive trap functions (Deb and Goldberg 1993). We also apply the FMGA to a new, more difficult synthetic problem which has a sparse solution. The performance of the FMGA is particular strong on the synthetic problem with a sparse solution.

1.1 CHC

We use as a benchmark the CHC adaptive search algorithm (Eshelman 1991). CHC is a generational genetic search algorithm which uses truncation selection.

The CHC algorithm randomly pair parents, but only those string pairs which differ from each other by some number of bits (i.e., a mating threshold) are allowed to reproduce. The initial threshold is set at $l/4$, where l is the length of the string. When no offspring are inserted into the new population during truncation selection the threshold is reduced by 1. The crossover operator in CHC performs uniform crossover and randomly swaps exactly half of the bits that differ between the two parent strings.

No mutation is applied during the recombination phase of the CHC algorithm. When no offspring can be inserted into the population of a succeeding generation and the mating threshold has reached a value of 0, CHC infuses new diversity into the population via a form of restart known as *cataclysmic mutation*. Cataclysmic mutation uses the best individual in the population as a template to re-initialize the population. The new population includes one copy of the template string; the remainder of the strings are generated by repeatedly mutating some percentage of bits (e.g., 35%) in the template string.

2 The Geometric Matching Problem

Object recognition problems in computer vision can be solved by finding a discrete correspondence mapping between an object model and a subset of image features such that projected model features align with corresponding image features. There are two interrelated parts to this problem: the *correspondence* problem and the *pose* problem. The correspondence problem involves correctly pairing features of the model with a subset of features extracted from a 2D image. The pose problem is to best estimate the 3D position and orientation of the object relative to the camera.

Given a pose algorithm which places the camera relative to the object for *specific* correspondences and an objective function to measure the relative quality of *alternative* correspondences, object recognition becomes a combinatorial subset feature selection problem. A variety of techniques have been suggested for searching the correspondence space. Of these, perhaps the best analyzed approach is tree search as formalized by Grimson (1990). Unfortunately, Grimson has shown that tree search requires exponential time to find an acceptable match under many common circumstances.

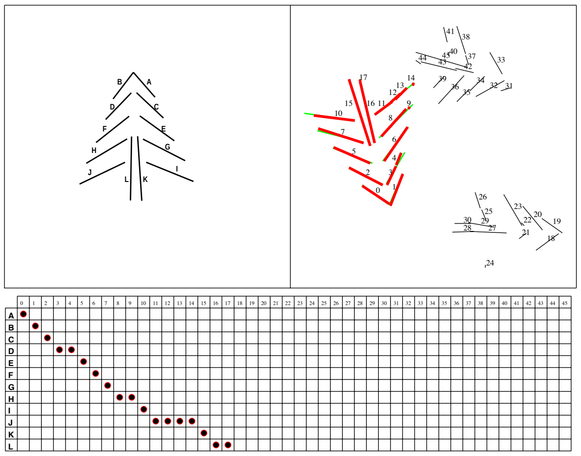


Figure 1: Example of a best match for one of the 48 test problems.

Dr. Beveridge's work on object recognition shows local search in the form of bit-climbing algorithms to be a powerful tool for finding optimal matches between features on 3D geometric object models and features in 2D images [Beveridge, 1993, Beveridge and Riseman, 1995]. These algorithms excel on problems involving poor quality image data and cluttered scenes.

Here problems involving 2D object models such as shown in Figure 1 will be considered. The line segments making up the model are labeled with letters and are shown on the left. Data line segments including three instances of the Tree are shown to the right. The model is overlaid on top of the data in the best match position. In matching, the model may be rotated, translated and scaled so as to best fit the data. The correspondence matrix in Figure 1 indicates which pairs of model and data segments are part of the best match: a dark circle signifies inclusion in the match.

Generally, the set of pairs S is the cross product of the model features M and data features D. The match space C includes *all* possible subsets of S. A bit string of length $l = |S|$ can encode a match $c = C$ using a 1 in the ith bit to indicate inclusion of the pair $s_i \in c$. The filled in squares in Figure 1 correspond to 1s in this bit string encoding.

An objective function is defined over the correspondence space which the best match c^* minimizes:

$$E(c^*) \leq E(c) \quad \forall c \in C \tag{1}$$

The match error E, described in [Beveridge, 1993], includes two terms: a fit error and an omission error. *Whenever* E is evaluated for a correspondence c, the best global 2D similarity transformation from object model to data is computed. The specific fit error minimized is the integrated, squared perpendicular distance between infinitely extended model lines and the data line segments, allowing matching to arbitrarily fragmented data. The best fit for any c, neglecting underconstrained cases, is computed by solving a quadratic polynomial. The omission part of E is computed by transforming the model to the best-fit pose and measuring how well the data covers the model.

Due to the manner in which E is computed, it is possible to rapidly compute a highly reliable estimate of the ΔE associated with a single bit toggle. This is a great advantage for local search operating on the neighborhood of single bit changes. The details of this incremental update procedure are explained in [Beveridge, 1993], page 83. The partial evaluation is one or two orders of magnitude faster than a full evaluation of E and gives local search an advantage over genetic algorithms that make larger jumps in the representation space.

2.1 A Modified Messy Genetic Algorithms

A Messy Genetic Algorithm typically has three phases:

1. Initialization.
2. Primordial Phase.
3. Juxtapositional phase.

During initialization, a population containing one copy of all substrings of length k is created. The expectaion is that recombination will find the proper building blocks and assemble them into good solutions. Given a problem with size l and building block size k, the initialization phase requires a population size of $n = 2^k \begin{pmatrix} l \\ k \end{pmatrix}$. There are a total of $\begin{pmatrix} l \\ k \end{pmatrix}$ gene combinations of size k, and for each gene combination there are 2^k different allele combinations.

For the matching problem, it is not necessary to generate all possible substrings of size k. Only spatially proximal triples of line segments are used. Let M be the set of model lines and D the set of data line segments. For each model line $m_i \in M$, determine the closest two neighbors m_{i1} and m_{i2} as defined by Euclidean distance δ:

$$\begin{aligned} \delta(m_i, m_{i1}) &\leq \delta(m_i, m_k)\ \forall\, m_k \in M - \{m_i\} \\ \delta(m_i, m_{i2}) &\leq \delta(m_i, m_k)\ \forall\, m_k \in M - \{m_i, m_{i1}\} \end{aligned}$$

Also find the analogous nearest neighbors d_{j1} and d_{j2} for each data line segment $d_j \in D$.

Given a matching problem between M and D, each pair of segments $(m_i, d_j) \in S$ form two spatially proximate triples f_1 and f_2:

$$\begin{aligned} f_1 &= ((m_i, d_j), (m_{i1}, d_{j1}), (m_{i2}, d_{j2})) \\ f_2 &= ((m_i, d_j), (m_{i1}, d_{j2}), (m_{i2}, d_{j1})) \end{aligned} \tag{2}$$

Since each of the l pairs of model and data segments in S leads to 2 triples, there are $2l$ spatially proximate triples.

A messy gene is a pair of model-data features $s \in S$. The modified initialization phase creates the $2l$ triples: thus developing substrings of length $k = 3$ to seed the initial population. This modified form of initialization does create all possible "building blocks". However, the spatial proximity heuristic creates a set of building blocks that is likely to contain elements of the optimal match. We then rely on later phases of the Messy Genetic Algorithm to correctly assemble these blocks.

In a simplified primordial phase, the error E is computed for each to the $2l$ triples. These triples are then sorted, and some fraction of the best form the initial population. In the experiments presented here, the top 50% of triples are used. This simple selection of the better triples produces high quality building blocks.

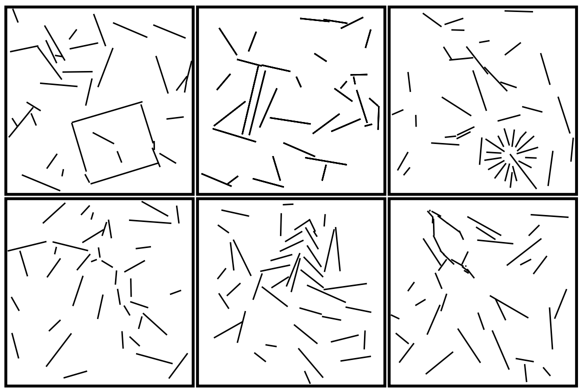
During Juxtapostion, selection is used together with two operators: cut and splice. Cut 'cuts' the chromosome at random position. Splice 'attaches' two cut chromosomes together. These two operators are the equivalents of crossover in a traditional GA. It is here that the Messy Genetic algorithm begins to construct the match out of small building blocks that appear to be good matches to some subset of features in the model.

At some point, recombination will typically construct enough of the match for local search to easily and quickly fill out the rest. For this reason, a *pass* of the steepest descent algorithm described above is periodically applied to individuals from the population. The frequency with with local search is run increases as population size decreases.

To help drive the Messy Genetic Algorithm to a solution, every three generations the least fit individual in the population is dropped and the population size correspondingly shrinks by one. Every $f = \frac{p}{2}$ generations, an individual is selected from the population and local search is run using the selected match as an initial state. If the result is better than the worst currently in the population, then it is inserted back into the population.

2.2 Results On Matching Problems

Figure 2 shows examples of 48 test problems created from six stick figure models. Model segments are randomly scaled and placed in the data images and are potentially fragmented, skewed and omitted. Finally, random clutter and structured clutter are added to the data. In 24 problems, 0, 10, 20 and 30 additional clutter segments are randomly placed about the image for each model: Figure 2a. In the another 24 problems, 0, 1, 2 and 3 additional more highly corrupted model instances are added: Figure 2b. This dataset and local search results are available through our website: `http://www.cs.colostate.edu/~vision`.


(a)

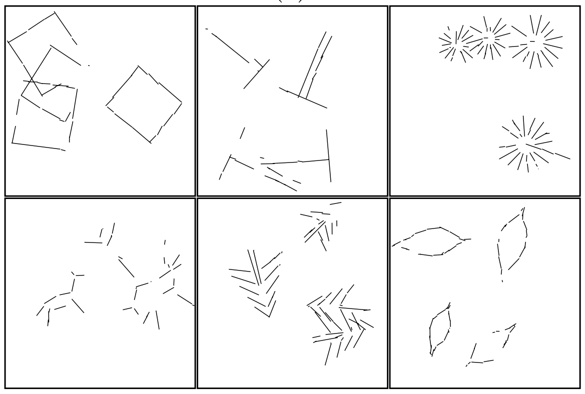

(b)

Figure 2: Test suite. a) Random clutter, b) Multiple model instances.

Previously [Whitley et al., 1995], the CHC and Genitor algorithms have been shown to perform poorly on this data. In contrast, this dataset is readily solved using different bit-climbing algorithms [Beveridge et al., 1995]. By hybridizing Genitor with the bit-climbing algorithm results comparable to those obtained using local search have been obtained.

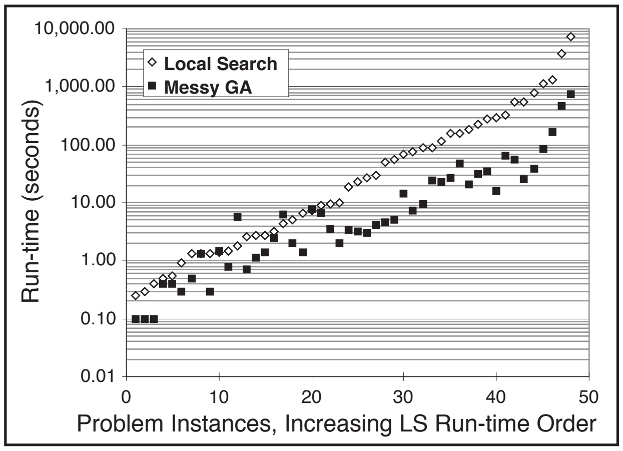


Figure 3: Comparison of run-times.

Here the Messy Genetic Algorithm described above performs is shown to perform much better than the random starts local search algorithm. To be more precise, we have run many trials of each algorithm in order to measure the probability P_s of finding the known best match in a single execution of either local search or the Messy Genetic Algorithm. Based upon these estimates of P_s, the number of trials t_s required to find an optimal match for given match with confidence Q_s is determined:

$$t_s = \lceil \log_{P_f} Q_f \rceil \quad Q_f = 1 - Q_s \quad P_f = 1 - P_s \tag{3}$$

For the Messy Genetic algorithm, the average t_s over the 48 problems is 2, the median is 1, the minimum is 1 and the maximum is 9. For local search, the the average t_s is 111, the median is 42, the minimum is 5 and the maximum is 998.

An estimate of the time required to solve each problem with 95% confidence is the average run-time per trial times the number of trials t_s. These run-times for a Sparc 20 are shown in Figure 3. On average, the Messy Genetic Algorithm is 5.9 times faster than local search.

The Messy Genetic Algorithm is doing better on the harder problems. Divide the problems into the the 24 solved quickly by local search and the 24 requiring the most time. On the easier problems, the Messy Genetic Algorithm runs on average 2.5 faster. In contrast, for the harder 24 problems the Messy Genetic Algorithm runs 9.4 times faster. In other words, for the problems taking thousands of seconds to solve using local search, the Messy Genetic Algorithm is dropping run-times by an order of magnitude.

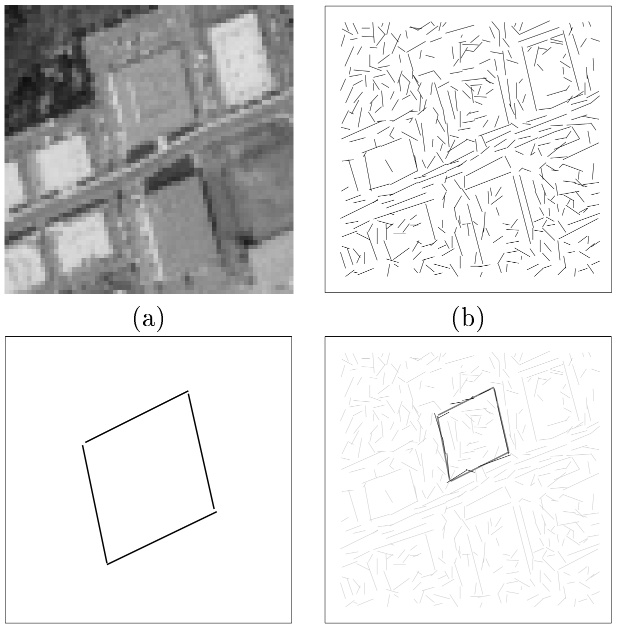

(a) (b) (c) (d)

Figure 4: Real data example. a) aerial photograph, b) Burns line segments [Burns et al., 1986], c) building model, d) best match.

Figure 4 illustrates a matching problem involving data from a real image. For this problem, there are 4 model line segments, 443 data line segments, generating $1,772$ possible pairs of segments. This problem is hard both because the search space is large, $2^{1,772}$ matches, and because the model interacts with other buildings and road structures to produce false matches.

Local search finds the optimal match 12 times in $10,000$ trials, yielding $P_s = 0.0012$ and $t_s = 2,494$. To run $2,494$ trials takes roughly 18 hours. In contrast, the Messy Genetic Algorithm finds the optimal match in 10 out of 100 trials. The average time for a trial of the Messy Genetic Algorithm to converge to a solution is 38 seconds, and $t_s = 29$. Hence, the Messy Genetic Algorithm reliably solves this problem in under 20 minutes.

3 Fast Messy Genetic Algorithms

The Fast Messy Genetic Algorithm (FMGA) was designed to cope with the problem of the large population size used by the Messy GA (Goldberg et al. 1993; Kargupta 1995). During the initialization phase the FMGA uses 'Probabilistically Complete Initialization.' The initial chromosome length is set to l', $k < l' < l$ (e.g. l' is $l - k$). The number of strings of size l' choosen from strings of size l is:

$$\binom{l}{l'}.$$

The probability of randomly selecting a gene combination of size k in a string of length l' with l genes is given by Kargupta (1995):

$$\binom{l-k}{l'-k} / \binom{l}{l'}.$$

Inverting this suggest that in strings created at random of size l', one string on average will have the desired gene combination of size k. To include all alleles combinations Goldberg et al. (1993) used the population-sizing equation developed for simple GA's (Goldberg et al 1992). The population sizing equation for FMGA's becomes:

$$\binom{l-k}{l'-k} / \binom{l}{l'} 2c(\alpha)\beta^2(m-1)2^k$$

where $c(\alpha)$ is the square of the ordinate of a normal random deviate whose tail area is α. The parameter β^2 is the maximum signal-to-noise ratio and m is the number of subfunctions to be solved (Kargupta 1995).

Because we are going to evaluate the Fast Messy Genetic Algorithm on existing problems (and also, to make our FMGA consistent with its original specification) we retained the practice that genes with allele value '0' can be included in chromosomes. We developed a C version of the Fast Messy Genetic Algorithm, based on Deb and Goldberg's (1991) Messy GA in C and Kargupta's (1995) thesis. Since we are ultimately interested in subset selection problems, our initial evaluation template fills all unspecified genes with value '0'.

A process of building block filtering takes place after the probabilistically complete initialization. Building block filtering is an iterative process that selects, filters and shrinks chromosomes. Selection is performed in order to increase the number of chromosomes with good evaluations. Selection has to assure competition between chromosomes that share genes in common (Kargupta 1995). After selection, a random gene deletion takes place designed to reduce chromosome size to building block size k. This is followed by the juxtapositional phase, which is basically the same used in the Messy Genetic Algorithm.

After the juxtaposition phase begins, the best chromosome found so far is used as a template. This means that genes with allele value '1' found in the best-so-far chromosome are added to those in the chromosome being evaluated; all other genes are assumed to have allele value '0'.

The main advantage of the FMGA is the relatively small population size compared with Messy GA. Nevertheless, for hard problems the size of the initial population is on the order of thousands and it remains unchanged during all phases of the FMGA (Kargupta 1995).

3.1 Block Insertion Fast Messy Genetic Algorithm (BIF-MGA)

We develop a modification on the final phase of the Fast Messy Genetic Algorithm that introduces more variability to individuals.

1. After the initial phase, the chromosome length is k. The chromosome length is increased to approximately l by using cut and splice during the juxtapositional phase multiple times. Splice is done with probability 1.0 and cut with probability 0.03, as in the FMGA (Kargupta 1995).
2. After most chromosomes have grow to length l or more, not all members in the population have the same length. In order to regularize the length of all individuals, a new length nl is fixed to be 0.75 l. For each individual with length il, if $il > nl$, il is reduced to nl by randomly deleting genes. Otherwise no gene deletion is performed.
3. A procedure called *Block Insertion* redefines the chromosome by inserting new random fixed-size messy gene blocks. Each messy-gene block has the following characteristics:
   (a) The block length is $l/3$.
   (b) The gene numbering is continuous starting at $l/3 * \mathit{Random}(0, 2)$ with randomly-generated allele values.
   (c) Let cl be the individual length, if $cl > l/3$ then the block is inserted at the begining of the chromosome by changing the first $l/3$ messy genes and leaving $cl - l/3$ messy genes without change. If $cl \geq l/3$ then cl messy genes taken from the block replaces the chromosome. HERE ASK CESAR
4. Another juxapositional phase of cut and splice is applied to again increased chromosome length to greater than l.

These changes were made via empirical experimentation in an effort to reduce the population size required by the FMGA. The population size of the BIF-MGA was not larger than 100 for all of our experiments.

4 Tests Results

We test the BIF-MGA on several problems with different degrees of complexity. Results were compared against both CHC and the standard FMGA.

4.1 Some Existing Tests Problems

Experiments were conducted using the following tests problems.

a) **Trap Functions** with j bits on the function and n attached functions as defined by Kargupta (1995). The basic trap function is defined as follows :

$$f(x) = \begin{cases} l & \text{if } u = l \\ l - 1 - u & \text{otherwise} \end{cases}$$

Experiments were conducted using two versions of the problem; $j = 3$, $n = 30$ and $j = 5$, $n = 20$. Messy genetic algorithms are particular well suited to this kind of decomposable problems. Results are shown in table 4.1.

b) **Subset selection problems** defined by Radcliffe and George (1993) and used by Crawford et al. (1997). A subset of s elements has to be selected among t elements. Within the subset there are g groups of neg elements. For our experiments neg is the same in all g groups. Experiments were conducted for non-epistatic 120-60-1 and epistatic 120-60-4 problems. We should note that Radcliffe and George defined two other more difficult epistatic problem than those studied here, but their algorithms also failed to solve the 120-60-4 problem. Our results are shown in table 4.1.

The performance of the BIF-MGA on the epistatic version of the subset selection problem 120-60-4 was better than CHC. The population size for the FMGA for the 120-60-1 problem was 2500 and no mutation was used. For the 120-60-4 problem the FMGA used a population of 3500, again with no mutation.

In order to assure that the block insertition process is not just a way of implementing high mutation rates, we ran experiments using the FMGA while varying gene and allele mutation probabilities from 0 to 1 by 0.1 increments for each parameter. The performance of BIF-MGA was much better than FMGA using various mutation rates.

4.2 The Sparse Subset Problem

We found the subset problems posed by Radcliffe et al. (1993), including those studied by Crawford et al. (1997), to be relatively easy to solve. A new synthethic test propblem, the *sparse subset problem* was

Problem	Opt	Algorithm Used	Best Result	Function Evaluations
Trap 90 3-bit	90	CHC	90	19442
		FMGA*	90	256500
		BIF-MGA	90	95863
Trap 100 5-bit	100	CHC	87	1500000
		FMGA*	100	1005000
		BIF-MGA	100	852000
Subset 120-60-1	60	CHC	60	1301
		FMGA	60	55252
		BIF-MGA	60	22726
Subset 120-60-4	60	CHC	60	566980
		FMGA	56	735015
		BIF-MGA	60	314952

Table 1: Tests results for different problems using CHC, FMGA and BIF-MGA. For FMGA* the results are taken from Kargupta (1995). CHC, FMGA and BIF-MGA results are averages of ten independent runs. Opt gives the optimal solution.

developed with the following characteristics. Two 60-bit blocks are composed of ten 6-bit subblocks. Each subblock of 6 bits uses the following evaluation function based on the number of 1 bits in the subblock.

Count of 1 bits	Contribution to Fitness
six	20
five	15
four	12
three	9
two	6
one (except 000001)	3
000001	12

For each of the two blocks of 60 bits, if the number of ones exceeds 13 there will be a penalty of $-2 * ones(block)$ where $ones(block)$ returns the number of ones in a block of 60 bits. Thus the maximum value of 240 is achieved when every subblock of 6 bits has the pattern 000001. Thus the solution is “sparse”. Results for this problem using 10 random runs of BIF-MGA and for CHC and FMGA are shown in Figure 5. Gene and allele mutation was set at 0.10 for FMGA; we in fact tested many different mutation levels, but FMGA never was better than CHC. The BIF-MGA solved the Sparse Subset Problem every time, while the other algorithm never found the optimal solution.

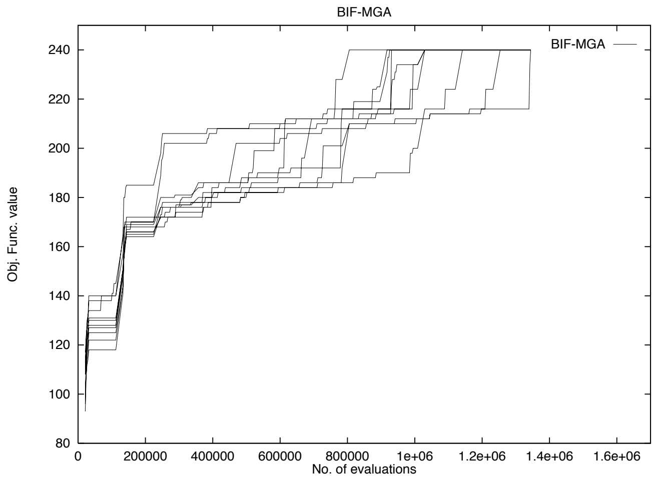
4.3 Discussion of BIF-MGA

Clearly, the Block Insertion F-MGA (BIF-MGA0 uses some ad hoc mechanisms to improve the performance of the FMGA. We initially conjectured that Block Insertion was just a form of mutation, but we failed to replicate the performance of the BIF-MGA by using mutation operators that randomly inserted new genes and/or changed allele values. Also, the BIF-MGA was not tuned for individual problems; rather it worked well across all of the problems on which it was tested without tuning. Note that Block Insertion is not used until after the standard mechanism of the FMGA have constructed strings that are largely of length l. It may be that in these later stages of the Messy Genetic Algorithm most of the work of putting together good building blocks has been done and there is some advantage in now re-organizing chromosomes back into some regular configuration. Certainly, this is one side effect of the BIF-MGA and it would seem to be the only side effect of the BIF-MGA that could not be emulated via some form of mutation. More work needs to be done to understand the impact of biffing the Fast Messy Genetic Algorithm with block insertion during the later stages of search.

5 Conclusions

We have shown that Messy Genetic Algorithms are extremely well suited to the problem of geometric matching in computer vision. The customized MGA we used for this problem yielded dramatic improvements over algorithms that represented the state of the art for the this set of test problems for the past 5 years. It also allowed us to tackle new large real world problems with thousands of line segments extracted from actual photographic images. Based on the Messy Genetic Algorithm results, we also developed a new Key-feature approach to these problems. Currently the new Key-feature approach and the Messy Genetic Algorithm produce similar results, but the Key-feature approach is somewhat more brittle and we believe that the Messy Genetic Algorithm can be further improved by exploiting more problem specific information. These results do demonstrate the potential for applying Messy Genetic Algorithms to subset feature selection problems, particularly when solutions tend to be sparse such that only a small percentage of the available features are actually selected.

On the synthetic subset selection problems, the Fast Messy Genetic Algorithm performs well compared to CHC. Again this supports the notion that Messy Genetic Algorithm may be particularly well suited to subset selection problems. A new variant of the FMGA, the Block Insertion Fast Messy Genetic Algorithm produced very good results, particularly on more difficult problems.



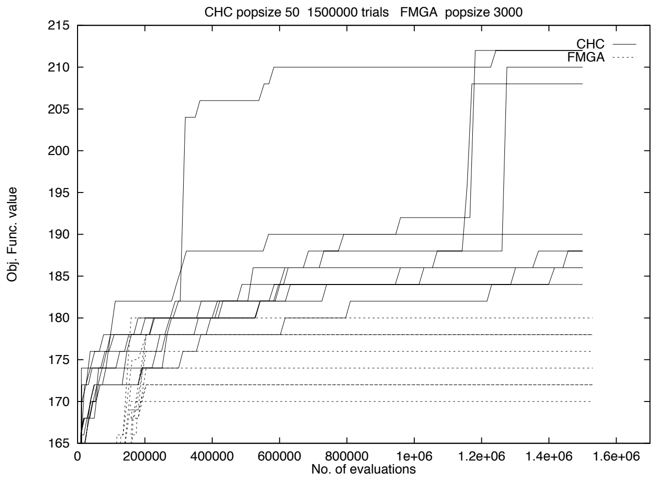


Figure 5: Results for the 120-bit Sparse Subset Problem.

6 Acknowledgements

César Guerra-Salcedo is a visiting researcher at Colorado State University supported by CONACyT under registro No. 68813 and by ITESM. This work was also supported in part by NSF grant IRI-9503366.

References

[Bala et al., 1995] Bala, J., Jong, K. D., Huang, J., Vafaie, H., and Wechsler, H. (1995). Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification. In *14th Int. Joint Conf. on Artificial Intelligence (IJCAI).*

[Beveridge, 1993] Beveridge, J. R. (1993). *Local Search Algorithms for Geometric Obejct Recognition: Optimal Correspondence and Pose.* PhD thesis, University of Massachuesetts at Amherst.

[Beveridge and Riseman, 1995] Beveridge, J. R. and Riseman, E. M. (1995). Optimal Geometric Model Matching Under Full 3D Perspective. *Computer Vision and Image Understanding*, 61(3):351 – 364. (short version in IEEE Second CAD-Based Vision Workshop).

[Beveridge et al., 1995] Beveridge, J. R., Riseman, E. M., and Graves, C. (1995). Demonstrating polynomial run-time growth for local search matching. In *Proceedings: International Symposium on Computer Vision*, pages 533 – 538, Coral Gables, Florida. IEEE PAMI TC, IEEE Computer Society Press.

[Brill et al., 1992] Brill, F., Brown, D., and Martin, W. (1992). Fast genetic selection of features for neural network classifiers. *IEEE Trans. on Neural Networks*, 3(2):324–328.

[Burns et al., 1986] Burns, J. B., Hanson, A. R., and Riseman, E. M. (1986). Extracting straight lines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI–8(4):425 – 456.

[Deb and Goldberg, 1993] Deb, K. and Goldberg, D. (1993). Analyzing Deception in Trap Functions. In Whitley, L. D., editor, *FOGA - 2*, pages 93–108. Morgan Kaufmann.

[Deb and Goldberg, 1991] Deb, K. and Goldberg, D. E. (1991). mga in c: A messy genetic algorithm in c. Technical report, Department of General Engineering University of Illinois at Urbana-Champaign.

[Eshelman, 1991] Eshelman, L. (1991). The CHC Adaptive Search Algorithm. How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlins, G., editor, *FOGA -1*, pages 265–283. Morgan Kaufmann.

[Goldberg et al., 1989] Goldberg, D., Korb, B., and Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results. *Complex Systems*, 4:415–444.

[Goldberg et al., 1992] Goldberg, D. E., Deb, K., and Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems.*

[Goldberg et al., 1993] Goldberg, D. E., Deb, K., Kargupta, H., and Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Forrest, S., editor, *Proc. of the 5th Int'l. Conf. on GAs*, pages 56–64. Morgan Kauffman.

[Grimson, 1990] Grimson, W. E. L. (1990). *Object Recognition by Computer: The Role of Geometric Constraints.* MIT Press, Cambridge, MA.

[Kargupta, 1995] Kargupta, H. (1995). *SEARCH, Polynomial Complexity, And The Fast Messy Genetic Algorithm.* PhD thesis, Department of Computer Science University of Illinois at Urbana Champaign.

[Kelly D. Crawford and Schoenefeld, 1997] Kelly D. Crawford, Cory J. Hoelting, R. L. W. and Schoenefeld, D. A. (1997). A study of fixed-length subset recombination. In Belew, R. and Vose, M., editors, *FOGA - 4*. Morgan Kaufmann.

[Radcliffe and George, 1993] Radcliffe, N. J. and George, F. A. W. (1993). A study in set recombination. In Forrest, S., editor, *Proc. of the 5th Int'l. Conf. on GAs*, pages 23–30. Morgan Kauffman.

[Whitley et al., 1995] Whitley, D., Beveridge, R., Graves, C., and Mathias, K. (1995). Test driving three 1995 genetic algorithms: New test functions and geometric matching. *Journal of Heuristics*, 1(1):77 – 104.