On Multi-class Classification by Way of Niching

A.R. McIntyre and M.I. Heywood

Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia, Canada B3H 1W5
{armcnty, mheywood}@cs.dal.ca

Abstract. In recent literature, the niche enabling effects of crowding and the sharing algorithms have been systematically investigated in the context of Genetic Algorithms and are now established evolutionary methods for identifying optima in multi-modal problem domains. In this work, the niching metaphor is methodically explored in the context of a simultaneous multi-population GP classifier in order to investigate which (if any) properties of traditional sharing and crowding algorithms may be portable in arriving at a naturally motivated niching GP. For this study, the niching mechanisms are implemented in Grammatical Evolution to provide multi-category solutions from the same population in the same trial. Each member of the population belongs to a different niche in the GE search space corresponding to the data classes. The set of best individuals from each niche are combined hierarchically and used for multi-class classification on the familiar multi-class UCI data sets of Iris and Wine. A distinct preference for Sharing as opposed to Crowding is demonstrated with respect to population diversity during evolution and niche classification accuracy.

1 Introduction

Genetic Programming is increasingly being applied to data mining tasks [1], with advances proposed for encouraging simple solutions [2] and the provision of confidence intervals as well as class labels [3]. One of the major impediments to the wide spread utilization of GP methods in real world data mining applications, however, is still associated with scalability. That is, given the computational overheads associated with the inner loop of GP, how may the algorithm be efficiently applied to solve multi-class problems or problems based on exceptionally large datasets. In this work we are interested in the case of the multi-class problem. That is, how to evolve multi-class classifiers from single GP trials, where the standard approach is to evolve separate binary classifiers for each class [4]. Dividing this job across a Beowulf cluster, for example, provides a brute force method to arrive at a classifier for an arbitrary number of classes on the same dataset.

In this work we investigate the provision of multi category classifiers in a single GP trial through the use of niching metaphors, using algorithms derived from the Genetic Algorithm literature [5-8]. That is, each GP run provides a set of binary classifiers solving the multi-class problem as a group. To do so, this paper investigates the feasibility of a crowding / sharing hybrid algorithm that naturally extends the basic GP framework to a multi-class classifier. A comparative analysis of the niching properties of the GP model with the GA crowding experiments of Mahfoud [5] is provided

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 581–592, 2004.

along with an extension to a hybrid model using deterministic crowding and dynamic sharing [8]. We demonstrate that such a scheme is indeed capable of yielding multi-category classifiers with properties broadly in line with those originally observed by Mahfoud.

The remainder of this paper is organized as follows: Section 2 provides the background information and previous work in this area, Section 3 describes the basic objective and the methodology of this research. Section 4 summarizes the results and provides an analysis of the study and section 5 concludes the paper.

2 Background

Genetic Programs typically identify one single 'super individual' representing a single peak in the search space. Moreover, the standard Canonical Genetic Algorithm (GA) will converge to a single peak even in multimodal domains characterized by peaks of equivalent fitness. This effect is known as genetic drift. Mechanisms that allow GAs to evolve solutions that do not only converge to a single peak are known here as 'diversity mechanisms'.

The multi-modal problem has been of interest within the context of optimization problems for a considerable period of time, with modifications proposed for both Evolutionary Programming [9] and Genetic Algorithms [5-8]. In the case of this work, multimodal GAs provide a natural path for incorporating diversity mechanisms into the GP multi-category classification context. Traditional mechanisms considered as the basis for this work include Deterministic crowding [7], and Sharing [6, 8].

Crowding algorithms were originally defined in terms of a form of tournament. However, rather than tournaments taking place to define individuals to reproduce they were used to define the individuals to be replaced; the number of individuals taking place in such a tournament being denoted the crowding factor [5]. The concept of crowing was thoroughly reviewed and revised by Mahfoud, when a series of experiments were performed on a set of basic multimodal problems, resulting in the definition of a 'Deterministic Crowding' algorithm [7]. These experiments form the basis for the first part of the study conducted here with multi-category classification problems, Section 3 algorithms A1 to A5. The method does however have some shortcomings. Specifically, population drift on more taxing problems is still observed [8]. As a consequence, we extend our review to include the concept of sharing, and demonstrate this within the context of GP, Section 3 algorithm A6.

2.1 Grammatical Evolution

The following study is performed using Grammatical Evolution (although results are not specific to the type of Genetic Programming employed). Grammatical Evolution (GE) permits automatic and language independent evolution of programs of arbitrary complexity [10]. To some extent, there are obvious similarities between GE and GP. However GE does not operate directly on the programs themselves (as in traditional GP), rather the programs are stored as a series of Backus-Naur form (BNF) grammar rule selectors, which are in turn indirectly represented by a fixed length binary string individual. In this sense, GE is similar to GA and consequently permits the use of

simple GA search operators. Based on the findings of Kishore *et al.* regarding their success with minimal functional sets [4], we constrain the expression of our GE individuals to the following simple BNF grammar (for an *i*-input classification problem):

```
code : exp
exp  : var | const | exp op var | exp op exp
op   : + | - | * | /
var  : x1 | x2 | … | xi
const: 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

The fitness of an individual for a given class is calculated by proportionally incrementing the total fitness score (starting from zero) for each correct training set classification (binary class of 1 or 0, where the increment is weighted to equate the value of in and out of class exemplars when they are unequal in number). The details of GE and further comparisons with traditional evolutionary computing methods have been considered elsewhere [10] and will not be re-examined in this paper.

2.2 Classification and GP

A distinction is typically made between binary classifiers and multi-class (or multi-category) classifiers. A multi-class classifier predicts a single label for each example from a set of many mutually exclusive candidate labels, while a binary classifier predicts each example as either in or out of class (in other words, the prediction is binary).

GP has been successfully applied to binary and, more recently, to multi-class classifiers. However, research has only recently begun to address the multi-class nature of the problem directly [11]. To do so, fitness is explicitly evaluated with respect to each class and the individual assigned to the fittest category. Although providing a deterministic solution to the problem for the dataset considered, no investigation is made of the robustness of the algorithm (e.g. sensitivity to drift, dynamics of niche maintenance). The objective of this work is to investigate the multi-class problem from a wider perspective of a niche metaphor as developed by the Genetic Algorithm literature.

3 Methodology

The approach taken for *k*-class classification is to formulate the problem as *k* binary classification problems and hierarchically process the examples through each classifier. Using this approach allows us to separately evolve the *k* binary classifiers where each classifier can distinguish a given example as in or out of its particular class. The cumulative effect of hierarchically combining the *k* classifiers is to discriminately place each example into the class of the highest accepting classifier, while those examples that are deemed out of class for all classifiers are placed into a collection to be reclassified or labeled as indeterminate or other.

In these experiments, we present a set of necessarily modified versions of Mahfoud's crowding algorithms, A1-A5 that have been re-designed for multi-

population GP (specifically a GE multi-class classifier). A0 has been omitted due to its reliance on genotypic similarity metrics, which has no obvious, meaningful GP counterpart. Finally, algorithm A6 is introduced to investigate the potential for sharing based maintenance of niches where this has the potential to provide more robust niche maintenance [8].

3.1 Establishing Phenotypic Similarity

The main modification for these experiments was in defining the notion of phenotypic similarity between individuals. For Mahfoud's experiments, phenotypic similarity takes the form of a distance calculation between pairs of real-valued, decoded parameters (the expression of phenotype in a canonical GA). In the case of this work, we are interested in comparing the effectiveness of programs at solving a series of problems (each of which correspond to a niche, or pattern class, where an individual may acquire fitness).

Our design choice was to establish the concept of phenotypic similarity by evaluating each individual for each problem (niche), so that an individual effectively has k fitnesses (corresponding to a k niche problem). Accordingly, each individual in the population can be considered a point in k dimensional space. This approach was chosen because it seems to provide a straightforward extension of Mahfoud's phenotypic similarity measure, where we are simply extending the concept of phenotype into problem-specific fitness, giving multiple dimensions (one for each problem to be solved by GP). Finally we apply a multi-dimensional Euclidean norm to establish distance between pairs.

3.2 Niching Parameters

All parameters used in our experimental runs are taken directly from those established by Mahfoud [7] or have been changed as little as possible in an attempt to recreate similar incremental testing conditions. All algorithms use phenotypic comparison (as described in section 3.1), crossover rate of 0.9 (in the case of A1) or 1.0 (A2-A6) and no mutation. Each algorithm is run for the equivalent of 20,000 evaluations and results are averaged over 100 runs. To maintain consistency with population sizes, we have scaled population sizes as presented in Mahfoud (5 peak multi-modal functions were allowed 100 individuals [7]) down to 80 in order to fit the number of classes in the problems considered.

3.3 Performance Criteria

For the purposes of comparative analysis, this study is primarily concerned with measures of performance that are more general in nature than the sensitivity and specificity figures (defined in terms of true positive (tp), true negative (tn), false positive (fp) and false negative (fn)) that are traditionally associated with classification:

$$sensitivity = \frac{tp}{tp + fn} \tag{1}$$

$$specificity = \frac{tn}{tn + fp} \tag{2}$$

Although these measures will be established class wise for the test sets of the problems under consideration, here the focus will be on the degree of success of the niching algorithm in question, particularly as it relates to preserving population diversity, maintaining multiple niche optima, and minimizing effects of genetic drift in the GE. These criteria are assessed via the niche population distributions, number of peaks maintained and number of replacement errors made.

Population distributions are simply defined as the counts of population members whose best fitness performance corresponds to the given niche. As defined by Mahfoud, a peak is considered maintained at any iteration if its corresponding subpopulation contains at least one member whose fitness is 80% of the peak's optimal value. Finally, a replacement error refers to the event where a child belonging to niche N_i replaces a parent belonging to niche N_j where $i \neq j$.

3.4 Data Sets and Partitioning

In the following experiments, data sets were partitioned into training and test sets. Partitioning is performed by randomly assigning patterns without replacement to training or test such that 50% of patterns appear in training and 50% appear in test. In and out of class data are stratified in order to achieve proportional representation from each class within the two partitions. Note that the test partition represents a disjoint set ('unseen' exemplars) from the classifier's perspective (the training set). This property of the partitioning affords analysis of the classifier's generalization ability.

Two widely utilized multi-category classification benchmarks from the UCI repository [12] are considered,

1. Iris: Fisher's iris plant database. 150 exemplars each of four continuous inputs and one of three output classes (Iris Setosa, Iris Versicolor, or Iris Verginica);
2. Wine: Wine recognition database; using chemical analysis to determine origin of wine. 178 exemplars each of 13 continuous inputs and one of three output classes.

3.5 Crowding Algorithms as Adapted for GE

3.5.1 Algorithm A1 (equivalent to A0 from [7], with phenotypic similarity): This approach is representative of the original crowding algorithm [5]. Generation gap, *G*, defines the percentage of the population (size POP_SIZE) that is chosen for crossover via fitness proportionate selection. Following the application of crossover (or copy of

parents in the event of failed test for crossover), G × POP_SIZE individuals are chosen randomly for replacement. For each child, a random sample of CF (crowding factor) individuals is chosen without replacement. From this sample, the child replaces the most similar, phenotypically. For algorithm A1 G = 0.1, PC (Probability of application of crossover) is 0.9 and CF = 3:

```
1.  FOR( POP_SIZE )
        a.  Randomly initialize individual
        b.  Evaluate raw fitness (one for each niche) and best fitness
            (best case of raw fitnesses)
2.  FOR( GENS )
        a.  Proportionally select G = 0.1 ( PAIRS ) of the population
            for reproduction
        b.  FOR( PAIR )
                i.  Test / Apply crossover ( PC = 0.9 )
               ii.  Evaluate children (c1, c2) (raw fitnesses, best fit-
                    ness)
              iii.  FOR (child)
                        1.  Randomly choose CF( = 3 ) individuals of
                            population for possible replacement and cal-
                            culate their distance (dci) by phenotype (raw
                            fitnesses) from c
                        2.  Replace the nearest of the CF individuals
                            with c
```

3.5.2 Algorithm A2: Here the algorithm maintains the basic procedure of A1, with the following simplifications: Crossover is applied deterministically (PC = 1.0) and the generation gap (G) is reduced to the minimum to allow for selection of a single pair. The original algorithm is therefore modified by changing step 2 as follows:

```
2.  FOR( GENS )
        a.  Proportionally select 1 PAIR ( Gmin% ) of the population for
            reproduction
        b.  FOR( PAIR )
                i.  Test / Apply crossover ( PC = 1.0 )
```

This simplification was originally chosen in order to rule out parameter settings (G, PC) that may otherwise detract from the thorough analysis of results [7]. The motivation for this also simplification holds here.

3.5.3 Algorithm A3: This algorithm is the same as A2, but here the CF is increased to that of POP_SIZE. Although this is a more costly algorithm in terms of overall time complexity, the hypothesis is that it will correspond to 100% correct replacements. Step 2.b.iii.1 of the algorithm now becomes:

```
1.  Randomly choose CF( = POP_SIZE ) individuals of population for
    possible replacement and calculate their distance (dci) by phe-
    notype (raw fitnesses) from c
```

3.5.4 Algorithm A4: The concept of a crowding factor is removed, instead we proceed by replacing parents at each generation with the children such that the sum of similarity between child-parent replacement pairing is maximized [7]. The hypothesis at this stage is that children will typically be most similar to their parents, thus removing the computationally expensive CP=POP_SIZE. Step 2.b.iii of the algorithm is now replaced and extended with:

```
iii.  Compare children and parent distances (dij) by phenotype (raw
      fitnesses)
 iv.  IF( d11 + d22 < d12 + d21 )
            r1 = p1; r2 = p2
  v.  ELSE
            r1 = p2; r2 = p1
 vi.  replace r1 with c1
vii.  replace r2 with c2
```

3.5.5 Algorithm A5: Also known as Deterministic Crowding [7], all details remain the same as A4 except that generation gap and fitness proportionate selection are dropped. Instead, the population is paired up randomly (without replacement) at each generation. This algorithm also differs from A4 in that a child can only replace the chosen parent if the child's best fitness is an improvement on that of the parent. Step 2 is now:

```
2.  FOR( GENS )
       a.  Pair up individuals without replacement (PAIRS)
       b.  FOR( PAIR )
              i.  Test / Apply crossover ( PC = 1.0 )
             ii.  Evaluate children ( c1, c2 ) (raw fitnesses, best
                  fitness)
            iii.  Compare children  and parent distances (dij) by
                  phenotype (raw fitnesses)
             iv.  IF( d11 + d22 < d12 + d21 )
                        r1 = p1; r2 = p2
              v.  ELSE
                        r1 = p2; r2 = p1
             vi.  IF( best fitness( c1 ) > best fitness ( r1 ))
                        replace r1 with c1
            vii.  IF( best fitness( c2 ) > best fitness ( r2 )) re-
                  place r2 with c2
```

3.5.6 Algorithm A6: Adding the element of sharing to the deterministic crowding algorithm (A5) takes the form of calculating the sharing parameters following each new generation and replacing the chosen parent only if the child has a larger value of dynamic shared fitness (DSF) [8]. Section 2.b.vi and vii become:

```
 vi.  IF( DSF( best fitness( c1 ) ) > DSF( best fitness ( r1 ) ))
      replace r1 with c1
vii.  IF( DSF( best fitness( c2 ) ) > DSF( best fitness ( r2 ) ))
      replace r2 with c2
```

Table 1. Mean niching properties for algorithms A1 – A6 on Wine and Iris Training Data.

				FNS		
		TRE	**TPM**	**C1**	**C2**	**C3**
IRIS	**A1**	4626.6	1.29	47.33	17.96	14.71
	A2	5803.26	1.46	45.34	17.74	16.92
	A3	188.44	2.1	42.95	19.04	18.01
	A4	712.46	0.97	75.8	2.63	1.57
	A5	135.11	1.89	76.89	0.03	3.08
	A6	295.16	2.44	36.89	16.14	26.97
WINE	**A1**	4779.69	1.24	24.58	14.3	29.12
	A2	6134.01	1.52	29.33	19.7	30.97
	A3	125.24	2.6	22.58	20.3	36.32
	A4	1689.03	0.57	19.3	3.89	56.81
	A5	224.04	2.18	5.43	1.97	72.6
	A6	343.29	3	27.12	25.63	27.25

4 Results and Analysis

Results for the GE algorithms (A1-A6) are summarized in Tables 1 and 2 as mean performance over 100 runs. These results report on performance criteria in terms of both classifier quality (sensitivity, specificity, denoted SENS and SPEC respectively, over each problem and class, C1-C3) as well as the relative success of each niching algorithm on a per problem basis as characterized by total replacement errors (TRE), total peaks maintained (TPM), and final niche sizes (FNS, over classes C1-C3).

Table 1 indicates a general decreasing trend to the replacement error over algorithms A1 to A6. However, in terms of the average number of niches maintained, A3 and A6 always retain the correct niche count, where this is also reflected in the final niche sizes for each class. It is interesting to note that of the various niching algorithms investigated, Deterministic Crowding (A5) did not provide particularly strong niche identification.

From Table 2, it is clear that algorithm A3 does not result in a particularly effective classifier in spite of the comparatively good niche characterization from Table 1. In effect A3 would consistently become stuck in a set of suboptimal niches. Algorithm A6, the sharing algorithm, provides the best overall performance, with Deterministic Crowding (A5) also very effective. Classifier results for separate binary runs (SB) have been included for comparative purposes, where the population size was reduced by a factor of $1/k$ that of the niching runs in order to balance computational costs; k (=3) being the number of classes. From Table 2, it appears that the reduced population size was detrimental to the success of the separate binary runs, establishing the value of the niching approach over separate runs under similar resource constraints.

Table 2. Mean classification results for algorithms A1 - A6: Wine and Iris Test Data

		SENS			SPEC		
		C1	C2	C3	C1	C2	C3
IRIS	**A1**	0.98	0.76	0.81	0.95	0.93	0.93
	A2	0.99	0.87	0.82	0.99	0.91	0.95
	A3	0.97	0.71	0.70	0.93	0.88	0.96
	A4	0.95	0.88	0.85	0.98	0.93	0.97
	A5	1.00	0.85	0.83	0.99	0.92	0.97
	A6	0.99	0.85	0.85	0.99	0.93	0.96
	SB	0.71	0.53	0.76	0.93	0.68	0.67
WINE	**A1**	0.82	0.73	0.93	0.92	0.94	0.89
	A2	0.76	0.74	0.90	0.93	0.91	0.89
	A3	0.46	0.50	0.73	0.88	0.93	0.66
	A4	0.90	0.82	0.96	0.95	0.97	0.93
	A5	0.79	0.76	0.87	0.93	0.97	0.89
	A6	0.82	0.80	0.97	0.96	0.96	0.93
	SB	0.48	0.46	0.70	0.77	0.75	0.64

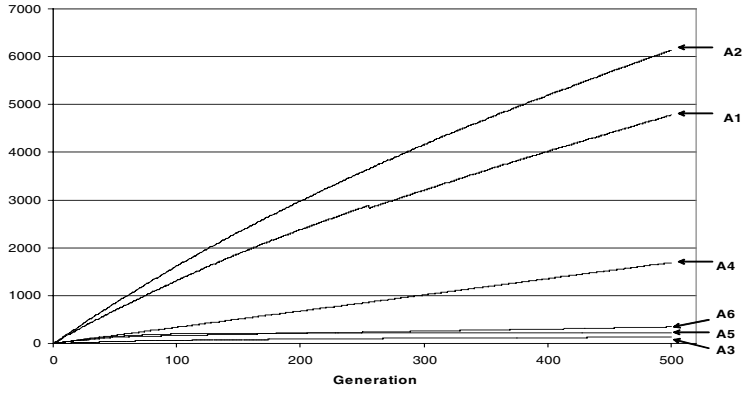

Fig. 1. Mean Cumulative Replacement Errors, A1 to A6 on Wine Training Dataset

4.1 Analysis of Niche Maintenance

This analysis concentrates on the Wine results, although similar properties were demonstrated for the case of Iris. The cumulative replacement error profile illustrates some important differences with respect to the original GA multimodal optimization context [7]. Firstly, 100% crossover resulted in A2 making substantially more replacement errors than the 90% crossover of A1. Secondly, algorithm A4 in which parents were always over-written with the most similar child actually resulted in an

increase in the replacement error, emphasizing the difficulty in comparing programs in GP. In the case of niche maintenance profile, Figure 2, algorithm A4 is again highlighted, thus reinforcing the observation that children of a GP parent often perform much worse. Algorithms A2, A3 and A6 all provide incremental advantages over their respective predecessor. Performance of Deterministic Crowding, A5, relative to Dynamic Niche Sharing, A6, demonstrates a tendency of Deterministic Crowding to drift after initially identifying the correct number of niches. This latter observation is further demonstrated by Figures 3 and 4 that summarize the evolution of each niche under A5 and A6 respectively.

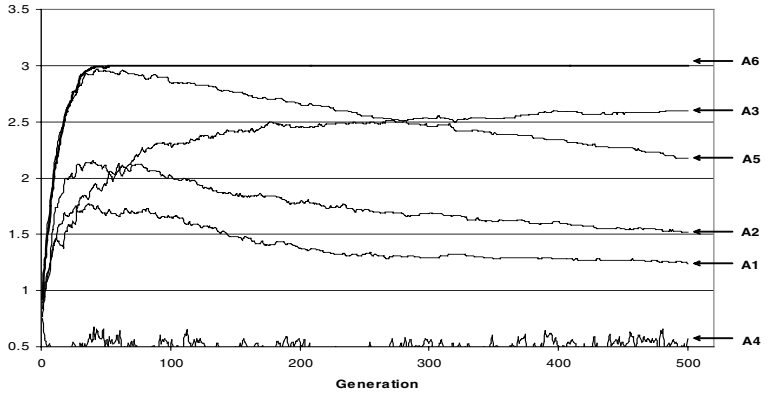

Fig. 2. Mean Niches Maintained, A1 to A6 on Wine Training Dataset

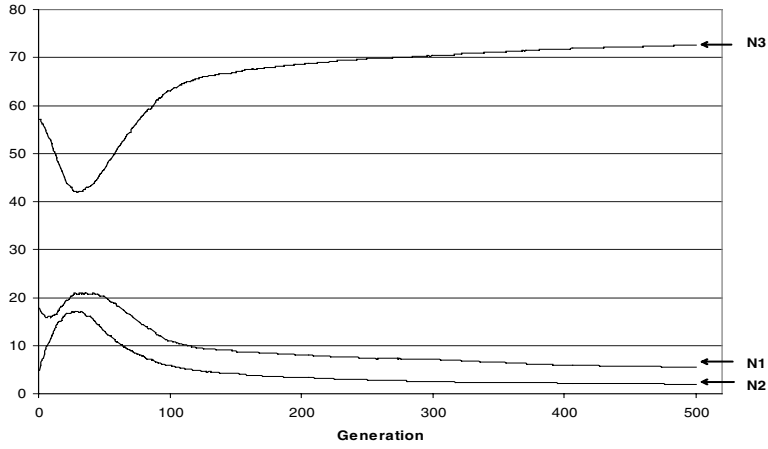

Fig. 3. Mean Niche Count with A5, Deterministic Crowding on Wine Training Dataset

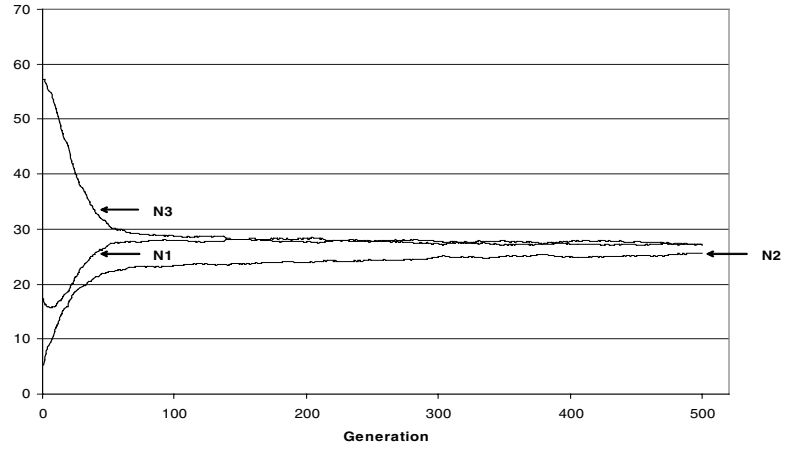

Fig. 4. Mean Niche Count with A6, Dynamic Niche Sharing on Wine Training Dataset

5 Conclusions

The utility of crowding and sharing algorithms as developed for multi-objective optimization with GAs has been demonstrated under GP multi-category classification. Dynamic Niche Sharing appears to provide the best support for the GP context. Deterministic Crowding appears to suffer from a combination of poor child fitness and genetic drift, the combination of which results in the correct number of niches first being located, but thereafter the largest niche continues to consume the majority of population diversity.

Acknowledgements. The authors gratefully acknowledge the support of NSHRF and NSERC Research Grants and a CFI New Opportunities Infrastructure Grant.

References

1. Ghosh A., Freitas A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms: Guest Editorial, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 517-518
2. Zhou C., Xiao W., Tirpak T.M., Nelson P.C.: Evolving Accurate and Compact Classification Rules with Gene Expression Programming, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 519-531
3. Au W.-H., Chan K.C.C., Yao X.: A Novel Evolutionary Data Mining Algorithm with Applications to Churn Prediction, IEEE Transactions on Evolutionary Computation. 7(6) (2003) 532-545

4. Kishore, J. K., Patnaik, L. M., Mani, V., Agrawal, V. K.: Application of Genetic Programming for Multicategory Pattern Classification. IEEE Transactions on Evolutionary Computation, Vol. 4, No. 3 (2000) 242-258
5. De Jong, K. A.: An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. Dissertation Abstracts International, Vol 36 No 10 (1975)
6. Deb, K., Goldberg, D. E.: An Investigation of Niche and Species Formation in Genetic Function Optimization. In: Schaffer, J. D. (ed.): Proceedings of Third International Conference of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA (1989) 42-50
7. Mahfoud, S. W.: Crowding and Preselection Revisited. In: Manner, R. and Manderick, B (eds.): Parallel Problem Solving from Nature, 2, Amsterdam, Elsevier Science (1992) 27-36
8. Miller B.L., Shaw M.J.: Genetic Algorithms with dynamic Niche Sharing for Multimodal Function optimization. Uni. Of Illinois at Urbana-Champaign, Dept. General Engineering, IlliGAL Report 95010 (1995) 11 pages
9. Yao X., Liu Y., Lin G.: Evoutionary Programming Made Faster. IEEE Transactions on Evolutionary Computation. 3(2), (1999) 82-102
10. O'Neill, M., Ryan, C.: Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language. Kluwer Academic Publishers. (2003)
11. Bojarczuk, C. C., Lopes, H. S., Freitas, A. A.: An Innovative Application of a Constrained Syntax Genetic Programming System to the Problem of Predicting Survival of Patients. In C. Ryan et al. (eds.) EuroGP 2003, Lecture Notes in Computer Science, Vol. 2610. Springer-Verlag, Berlin Heidelberg (2003) 11-21
12. Blake, C. L., Merz, C. J.: UCI Repository of Machine Learning Databases. University of California, Irvine, Dept. of Information Computer Sciences, http://www.ics.uci.edu/~mlearn (1998)