Symbiosis, Complexification and Simplicity under GP*

Malcolm I. Heywood and Peter Lichodzijewski†

7–11 July 2010

Abstract

Models of Genetic Programming (GP) frequently reflect a neo-Darwinian view to evolution in which inheritance is based on a process of gradual refinement and the resulting solutions take the form of single monolithic programs. Conversely, introducing an explicitly symbiotic model of inheritance makes a divide-and-conquer metaphor for problem decomposition central to evolution. Benchmarking gradualist versus symbiotic models of evolution under a common evolutionary framework illustrates that not only does symbiosis result in more accurate solutions, but the solutions are also much simpler in terms of instruction and attribute count over a wide range of classification problem domains.

1 Introduction

Pioneers in the field of Genetic Programming (GP) recognized the requirement for code reuse / problem decomposition early on and proposed schemes such as Automatically Defined Functions [11], Adaptive Representations [23] and various forms of recursion [4, 10]. A second development introduced the concept of Cooperative Coevolution [22, 21]. From a GP perspective, this has influenced recent advances in Teaming methods in which programs from multiple populations are combined under different 'orthogonal' contexts in an attempt to identify the optimal set of participants [24]. Conversely, models employing economic formulations have been pursued under the guise of Learning Classifier Systems – thus, essentially a GA – while examples under an explicitly GP context have made use of auction models [14] or define entire economic systems [2]. A final recent development has included the integration of multiple mechanisms including objectives that explicitly reward diversity [16] with archiving and coevolution [19].

A neo-Darwinian model of evolution is classically associated with a gradualist view of adaptation/ variation care of Mendelian genetics [12]. Neo-Darwinism

*Genetic and Evolutionary Computation Conference (GECCO) – Copyright 2010 ACM

†Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

is thus limited to incremental refinement care of 'micro' operators that focus on the inheritance of parental traits derived from the variation of specific genes [17, 18, 12]. Conversely, symbiosis makes extensive use of inheritance to coevolve independent organisms (symbionts) over a long term relationship that may ultimately result in new species (symbiogenesis) [17, 5]. Thus, hosts inherit 'acquired characteristics' – a neo-Lamarckian model – albeit from other organisms rather than their parents. Moreover, the ensuing coevolutionary interaction itself may begin as an adversarial relationship and over time emerge into one of mutualistic coexistence in which all parties benefit/ cannot exist without out each other [5, 9]. In this work, symbiosis is therefore assumed as the underlying metaphor for evolutionary inheritance and taken to imply the following requirements: (1) a mechanism exists for explicitly identifying symbionts participating in a symbiotic relationship, and; (2) symbionts continue to adapt under natural selection during the lifetime of the relation [17, 18]. As a consequence we make the following two general observations: (i) symbionts need to have the capacity to communicate the specific context in which they operate, and; (ii) the host is responsible for explicitly identifying participating candidate symbionts from a potentially much larger pool of organisms.

With this general background in mind, this study constructs a common framework for exploring symbiotic and gradualist models of evolution under GP. Specifically, a gradualist model of evolution is taken to imply that evolution is limited to gene-wise inheritance or a host 'compartment' size of one. This limits evolution to a monolithic style of program design with all aspects of an individual evolved simultaneously. Relaxing the host compartment limit introduces symbiosis under the same common framework for evolution. We are now in a position to explicitly identify the relative contribution of gradualist versus symbiotic approaches to program evolution under a range of benchmark classification problems with specific reference to solution complexity, CPU training requirements and classifier accuracy.

2 Symbiotic Bid-Based GP

Previous examples of symbiosis under Evolutionary Computation frequently recognize the utility of modularity in for building complex systems [5, 9]. For example, a two population model for symbiotic neural evolution was adopted by [20]; thus separate populations of neurons and network 'blueprints' are maintained. As such the blueprint (host) population defines the subset of neurons (symbionts) used to construct networks and it is at the blueprint level that fitness evaluation is performed. The serial link between two populations establishes the symbiont framework where neither population can exist without the other.

Under a GP context a similar two population model was employed for Symbiotic Bid-Based GP (SBB) [15]. One population declares a set of (symbiont) programs whereas a second population defines the host organisms in terms of subsets of symbionts, Figure 1. Assuming such an architecture makes the iden-

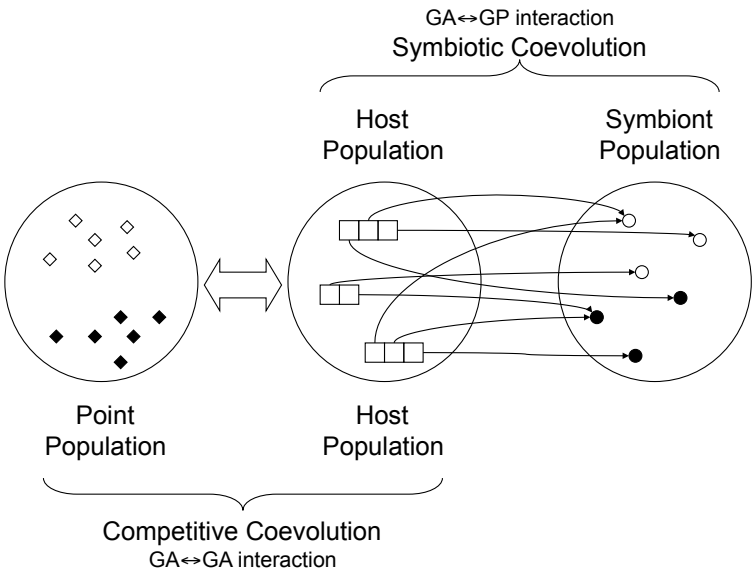

Figure 1: Basic SBB architecture. Diamonds in the point population denote indexes to exemplars from the training data. At initialization each class is sampled uniformly w.r.t. class, or a balanced initial population. Symbionts are distinguished on the right by their respective action. Under a classification domain there are as many actions as class labels. Host individuals appear in the center population.

tification of coevolving programs explicit. However, care is necessary in order to ensure that selection pressure acting on the coevolved symbiont programs does not disturb other symbiotic relationships in which the same symbiont appears. Likewise, a clear behavioral context must exist for each symbiont in order to avoid general pathologies of coevolution. This is achieved by casting the symbiont objective in the form of evolving a bidding policy relative to a fixed scalar 'action'. Thus, when a symbiont out bids other symbionts within the same host (on a given exemplar) it 'wins' the right to suggest its corresponding action. Moreover, in addition to symbiosis, the framework of Figure 1 utilizes both cooperative and competitive relationships to develop host diversity (fitness sharing) and guide point selection (variation in the environment) respectively.

Relative to the earlier SBB framework (e.g., [15]) the following algorithm differs in some important respects. One of the most significant difference lies in the maintenance of host population diversity. Previously, the combinatorial search conducted at the host level (for effective symbiont combinations) made use of search operators that first copied the intersection of symbionts into the child from the parents (crossover) and then applied mutation operators to vary the remaining content. This reinforced the common symbionts, resulting in a

bias towards exploitation over exploration of symbiont combinations that do not as yet exist in the host population. Given that the SBB model is elitist this could have implications for local minima. Search operators are now limited to mutation alone with different sets of operators appearing at the host and symbiont populations. A new host–symbiont population initialization procedure was also adopted, again to encourage more diversity in the host population. The next significant area of novelty relative to the earlier work lies in the specifics of the competitive coevolutionary framework used to scale SBB to large state spaces / data sets i.e., the point population on the left of Figure 1. Specifically, a "fitness sharing scheme" is now embedded within the competitive coevolutionary model that is more efficient than the original pairwise distinction based Pareto competitive models from which it was originally derived [6]. The following subsections detail the corresponding updated SBB algorithm, whereas Section 3.1 returns to the topic of dual support for gradualist and symbiotic models under a shared evolutionary framework.

2.1 Initialization

A breeder generational model of evolution is assumed in which the (competitively coevolved) point population P and host population M replace a fixed number of individuals at each generation; or the 'gap'. Thus, $P_{size} - P_{gap}$ individuals are initialized under the point population using a balanced sampling heuristic in which each class is represented equally i.e., provides some robustness to class imbalance [13]. Individuals comprising host and symbiont populations are initialized under the two stage process of Figure 2. The end result is a population of $M_{size} - M_{gap}$ hosts indexing (subsets of) a symbiont population initially consisting of twice as many (symbiont) programs.

Without loss of generality, a linear representation is assumed for the symbionts, with each individual defining bid and (scalar) action or Bid-GP (Section 2.6). The action for each symbiont is initialized with uniform probability at initialization from a set of a priori defined actions (e.g., class labels) and remains *fixed* during the lifetime of the symbiont. The bid component is associated with the corresponding symbionts' 'program' and is therefore subject to the regular type-wise process of initialization i.e., select instruction type and then instance of the type, both with uniform probability. Evaluation of a host implies that each symbiont program is evaluated under the current exemplar with the symbiont returning the maximum bid gaining the right to suggest it's corresponding (scalar) action i.e., class label.

2.2 Selection and Search Operators

At each generation P_{gap} new points and M_{gap} new hosts are added; *after fitness evaluation* a corresponding number of point and host individuals will be removed. Point generation follows the same procedure as at (point) initialization. However, adding new hosts can also result in new symbionts appearing in the symbiont population.

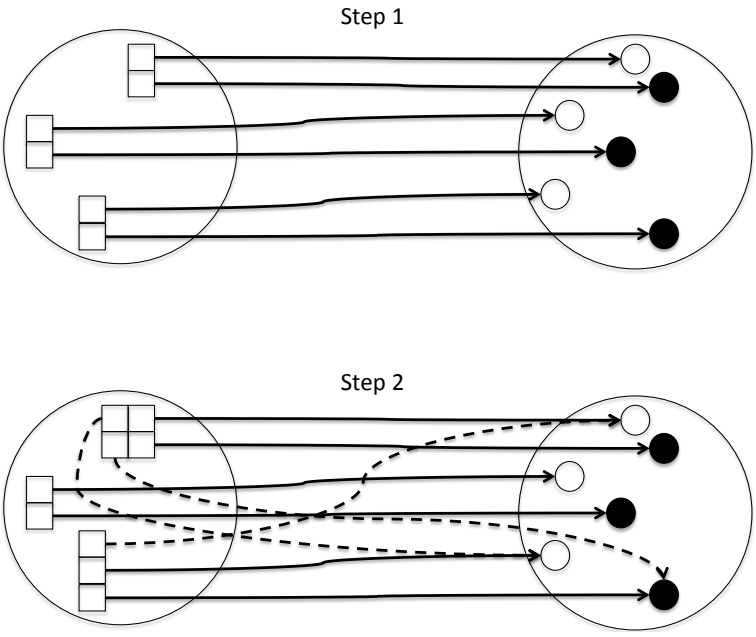

Figure 2: Initialization of host and symbiont populations. Step 1: Add $(M_{size} - M_{gap})$ hosts, one at a time, subject to: i) each host containing two unique symbiont actions; ii) symbiont actions are selected with uniform p.d.f.; iii) Programs are initialized over different lengths with instructions selected with uniform p.d.f. Step 2: Add additional symbiont indexing to the hosts. Host 'compartment' size is selected over the interval $\{2, ..., \omega\}$. The number of hosts and symbionts does not change during Step 2.

The following four part selection–search operator is assumed (Figure 3): **(1) Select parent** host from the $M_{size} - M_{gap}$ hosts available at generation t with uniform probability; **(2) Remove symbiont** indexes with uniform probability, $(p_{md})^{i-1}$, under the limit of maintaining a minimum of two symbionts per host. The index i ranges from 1 to the number of symbionts i.e., the probability of removing a symbiont decreases as the test is reapplied. After deleting the second symbiont the process stops. Symbionts are selected with uniform p.d.f. to avoid sequence preference artifacts; **(3) Add symbiont** indexes from the current symbiont population content with probability, $(p_{ma})^{i-1}$, under the limit of not exceeding ω symbionts per host, and; **(4) Modify symbionts** by applying a symbiont-wise test with uniform probability, p_{mm}. On testing true, the symbiont is first copied and a unique index created. The bid component of the symbiont is then modified through the repeated application of GP search operators (Section 2.6). The symbiont action can also be changed at this point, at a uniform p.d.f., p_{mn}. Note operators testing for symbiont modification are not exponentially weighted but applied at a constant rate; the process iterating until at least one symbiont is modified.

In summary, steps 2 and 3 introduce variation in the indexes that exist in the host population, whereas step 4 extends variation down to the symbiont population. In the latter case, the modified symbionts receive a unique index, thus ensuring that other hosts using the original copy of the symbiont remain unaffected[1]. In doing so, the host population is effectively conducting a combinatorial search for the best combinations of symbionts, whereas the symbiont population is optimized for building programs which are consistent relative to the host fitness evaluation i.e., symbionts are rewarded for continuing to adapt to their host context.

2.3 Evaluation

The outcome $G(m_i, p_k)$ of applying each host $m_i \in M(t)$ to each point $p_k \in P(t)$ at generation t is established as follows:

$$G(m_i, p_k) = \begin{cases} 1 & \text{if host } m_i \text{ classifies point } p_k \text{ correctly} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

2.4 Point removal

Point fitness, f_k, is expressed relative to the count of hosts, c_k, that correctly classify it, or

$$f_k = \begin{cases} 1 + \frac{1-c_k}{M_{size}} & \text{if } c_k > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

[1]The symbiont population size may vary between $2 \times M_{size}$ and $\omega \times M_{size}$ individuals i.e., each host can contain between 2 and ω symbionts which, if all indexes are unique, represents the upper bound on symbiont population size.

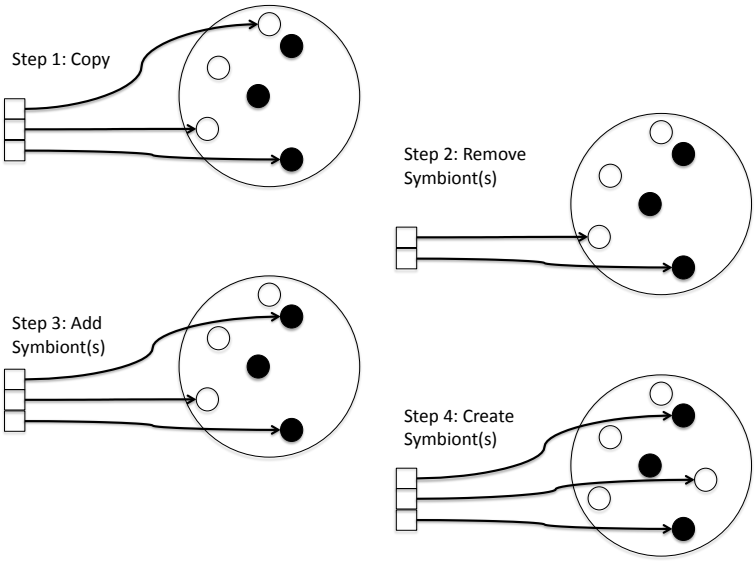

Figure 3: Hierarchical process for applying search operators to a single child host. Steps 1 through 3 modify the symbiont membership in a child host; Step 4 applies symbiont specific mutation operators to create a new symbiont.

where this establishes a linear weighting relative to the number of hosts correctly classifying the point; points that have a lower number of hosts correctly classifying them are rewarded – up to the limit of a point 'defeating' all hosts, which is also penalized. The P_{gap} points with lowest fitness are then deterministically removed at each generation.

2.5 Host–symbiont removal

Host–symbiont removal follows from the outcome vector of Eq. (1), albeit renormalized as a shared fitness function, s, to reward hosts which perform well on the less frequently correctly classified points, or

$$s_i = \sum_{p_k} \left(\frac{G(m_i, p_k)}{\sum_{m_j} G(m_j, p_k)} \right)^3 \tag{3}$$

where for each host, m_i, index p_k iterates over all points on which at least one host in the current population received a non-zero reward. The denominator establishes the aggregate reward on point p_k across all hosts i.e., a larger discount is applied to points for which many teams do well.

Once the shared score for each host is established, the M_{gap} hosts with lowest shared fitness values, s_i, are deleted. In addition we also test the symbiont population for any symbionts who are no longer indexed by the host population i.e., as a consequence of host deletion. Any such symbionts are considered to

Table 1: Instruction Set. Linear GP representation under register-register addressing. Index x denotes the target and first source register; whereas y denotes the second source register or an attribute reference. A mode bit would toggle between the two second source register scenarios.

Operation	Definition
Add	$R[x] \leftarrow R[x] + R[y]$
Sub	$R[x] \leftarrow R[x] - R[y]$
Multiply	$R[x] \leftarrow R[x] \times R[y]$
Divide	$R[x] \leftarrow R[x] \div R[y]$
Cosine	$R[x] \leftarrow cos(R[y])$
Logarithm	$R[x] \leftarrow ln(R[y])$
Exponential	$R[x] \leftarrow exp(R[y])$
Conditional	if $(R[x] < R[y])$ then $(R[x] \leftarrow -R[x])$

have 'died'. Also – when the host size reaches the limit, ω – we test for whether any symbiont fails to place a winning bid. Such cases are deleted from the team. Likewise, symbionts are tested for removal at the end of training, the latter irrespective of host symbiont count.

2.6 Bid-GP

The program of the symbiont defines a bidding strategy, with the result from the output register R_0 normalized to the unit interval using a sigmoid i.e., all symbionts use the same bidding interval. Actions are expressed as a scalar which is not subject to modification once initialized (Section 2.1). Following initialization of the symbiont population, symbiont creation/ bid program modification is asexual – care of step 4 of the host initiated process of variation (Section 2.2).

Search operators applied to Bid-GP take the form of four mutation operators: delete an instruction with uniform probability (p_{delete}); add an instruction with uniform probability (p_{add}); change a current instruction/ register reference with uniform probability (p_{mutate}); and interchange/ swap two existing instructions with uniform probability (p_{swap}). Following each round of application, the resulting bidding behavior is again assessed.

In order to guarantee diversity of the symbiont behaviors, each descendent Bid-GP individual has it's bidding behavior compared with the bidding behaviors from the current symbiont population on an independent sample of 50 exemplars drawn from the training partition.[2] If the new Bid-GP symbiont results in a profile with at least one of the 50 exemplars returning a bid value differing by more than 10^{-4}, then the new symbiont is deemed sufficiently unique. Should this not be satisfied then repeated rounds of Bid-GP search operators are applied until variation is detected.

[2]Sample is drawn once at initialization and is not necessarily unique from Point population content.

Table 2: Benchmarking Data sets. Parenthesis indicates percent major class contribution under training (test distributions are within 1% of training).

Data set	Attributes	Training	Test
Gisette (gis)	5 000	6 000 (50%)	1 000
Census (cen)	40	199 523 (93.8%)	99 762
Bupa (bpa)	6	310 (58.1%)	35
Pima (pma)	8	691 (65.1%)	77

3 Evaluation

3.1 Monolithic–gradualist GP

The gradualist framework for constructing solutions as monolithic programs under a common evolutionary framework with the SBB algorithm of Section 2 will enable any variation in observed performance to be attributed to the lack of support for symbiosis. This goal is achieved by applying two constraints to the SBB algorithm. Firstly, the host population is limited to individuals with a single index ($\omega = 1$), thus solutions can only consist of a single 'symbiont' i.e., solutions take the form of a single monolithic program. Secondly, the referenced program does not have a separate action and bid, but only an action as per a canonical linear GP classifier [3]. Thus, each individual provides a class label i.e., binary classification care of a sigmoid activation function thresholded at the origin. A common linear representation – for both SBB symbionts and monolithic GP – was assumed throughout with the instruction set as defined in Table 1.

3.2 Data Sets

For the purposes of this study[3] we consider the following three specific properties of interest, with Table 2 summarizing the specific properties of the data sets employed as sourced from the UCI repository (http://archive.ics.uci.edu/ml/):

Large attribute space: GP represents and example of an 'embedded' – as opposed to wrapper or filter – learning paradigms [8]. As such credit assignment simultaneously identifies the most appropriate attribute subspaces as well as classifier parameterization. Symbiosis provides an architecture able to associate different individuals with different subsets of exemplars and therefore unique attribute subspaces. The relevance of this should be most apparent under domains with thousands of attributes. Conversely, monolithic approaches to model building are forced to explicitly join multiple classifier/ attribute subspaces into a single program, potentially hindering classification performance and/ or solution simplicity. With this in mind the Gisette data set from the

[3]Additional properties such as performance under multi-class data sets and performance relative to alternate models of classification where the focus of earlier versions of SBB [15, 7].

Table 3: Parameterization at Host and Symbiont (program) populations under a common framework of monolithic and SBB evolution. As per Linear GP, a fixed number of general purpose registers are assumed ($numRegisters$) and variable length programs subject to a max. instruction count ($maxProgSize$). Values in parenthesis denote values specific to the evolution of monolithic models (Section 3.4).

Host (solution) level			
Parameter	Value	Parameter	Value
t_{max}	1 000	ω	10 (1)
P_{size}, M_{size}	120	P_{gap}, M_{gap}	20, 60
p_{md}	0.7	p_{ma}	0.7
p_{mm}	0.2	p_{mn}	0.1
Symbiont (program) level			
$numRegisters$	8	$maxProgSize$	48 (480)
p_{delete}, p_{add}	0.5	p_{mutate}, p_{swap}	1.0

2003 NIPS competition is employed where this consists of some 2,500 unique attributes and another 2,500 linearly translated 'dummy' attributes [8];

Large class imbalance: Real-world data sets do not necessarily represent each class with equal frequency. Indeed, the 'robust' classification of both minor and major classes is of particular relevance to a number of high impact applications such as medical diagnosis, fault and/ or intrusion detection. For both symbiotic and monolithic frameworks, credit assignment needs to resist the temptation to 'cherry pick' in favor of the majority class, resulting in potentially degenerate classifiers.[4] In this work the Census repository is employed as a representative example;

Historical relevance: Over time certain 'small' data sets have demonstrated their appropriateness for benchmarking by recording historically high rates of error across a wide number of machine learning paradigms [1]. In this case we utilize the Bupa Liver Disorder and Pima Indian Diabetes data sets, both of which tend to result in error rates in the region of thirty percent.

3.3 Performance Metrics

Solutions will be compared from three perspectives: Classification performance, model complexity, and CPU training requirements; using metrics summarized as follows. **Classification performance** is to be summarized from the perspective of multi-class detection rate (mcdr) or individual i has a $mcdr$ of $\frac{1}{|C|}\sum_{c \in C} detection(i, C)$ where $|C|$ is the number of classes and $detection(i, C)$ is the detection rate of the individual with respect to class C. Naturally such a metric is robust to class imbalance (unlike accuracy) so under binary classification problems a detection rate of 0.5 (i.e., $\frac{1}{|C|}$) is an indicator of degenerate

[4] Naturally both symbiotic and gradualist algorithms utilize the same model of competitive coevolution which should help mitigate the impact of class imbalance.

behavior (all exemplars labeled as a single class). **Complexity** will be measured in terms of the number of unique attributes a model utilizes and the number of effective instructions per solution. That is to say, individuals will be pruned of introns post training (Section 3.4) and the complexity metrics assessed. Naturally, the shared linear representation of monolithic and SBB solutions permits such a direct method of comparison. **CPU training requirements** will be evaluated care of the UNIX *getrusage* command (user time) as benchmarked under a common computing platform (no competing users). As such we are able to assess to what degree the SBB and monolithic solutions tradeoff algorithmic complexity for greater efficiency in locating solutions over a common generation limit.

3.4 Parameterization

Learning parameters are summarized as per Table 3 with the only differences between the parameterization adopted for monolithic and SBB runs being with respect to *maxProgramSize*. SBB could potentially construct host compartments consisting of up to 10 programs, each with 48 instructions. Thus, the equivalent maximum program limit under the monolithic model building framework was 480 instructions per individual. All runs are conducted over 60 initializations per training partition.[5]

Structural introns are identified prior to fitness evaluation during training and test[6] using specific tests for argument two instructions [3] and argument one instructions. In the latter case it is necessary to check for argument one instructions that target R_0 using an attribute for the argument. This potentially results in all earlier instructions associated with the R_0 path appearing redundant.

4 Results

4.1 Classification Performance

Performance under the training partition is used to define the representative host/ monolithic solution per run. Corresponding test partition performance w.r.t. the 4 test data sets and the multi-class detection rate (mcdr) metric (Section 3.3) is summarized in terms of a combined density of distribution/ box plot (Figure 4). In all but the case of Pima, the superiority of the SBB framework is apparent; a trend confirmed with a Kruskal-Wallis non-parametric hypothesis test at a significance level of 0.001. Moreover, it is also apparent that the consistency of SBB classification results is also generally better than that returned under the monolithic case.

[5] Census and Gisette having a single training–test partition; Bupa and Pima are assessed over 10 such partitions i.e., stratified 10-fold cross validation.

[6] Introns are naturally retained during genotypic processes such as applying search operators [3].

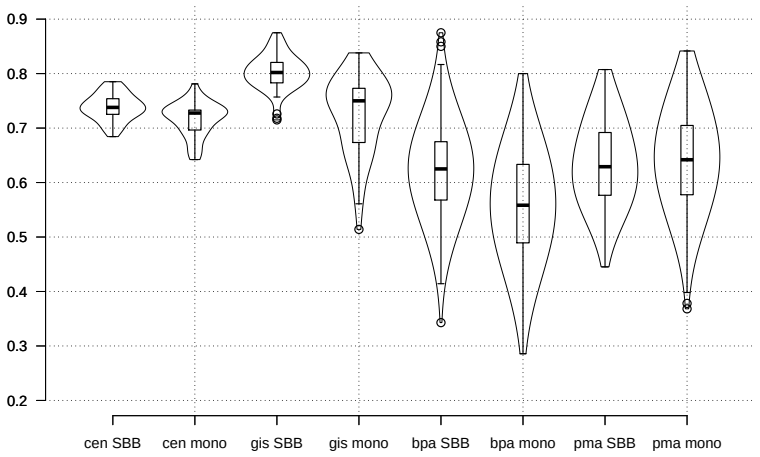

Figure 4: Multi-class Detection Rate (mcdr) over the four data sets. 0.5 denotes degenerate classifier behavior. 'mono' identifies monolithic-GP.

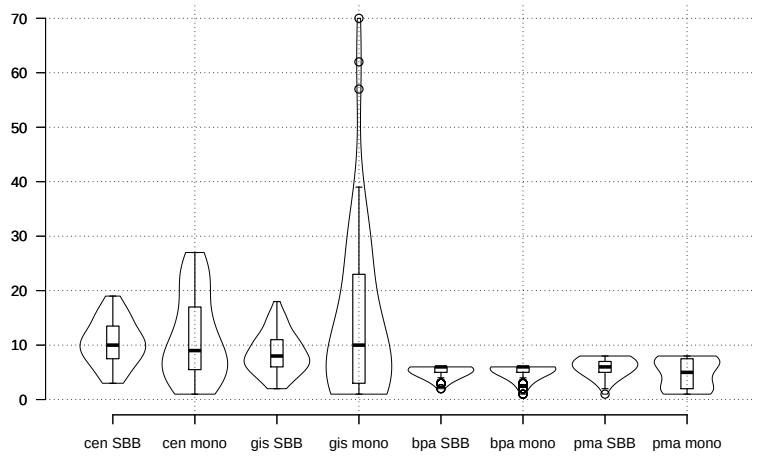

Figure 5: Total number of unique attributes per solution.

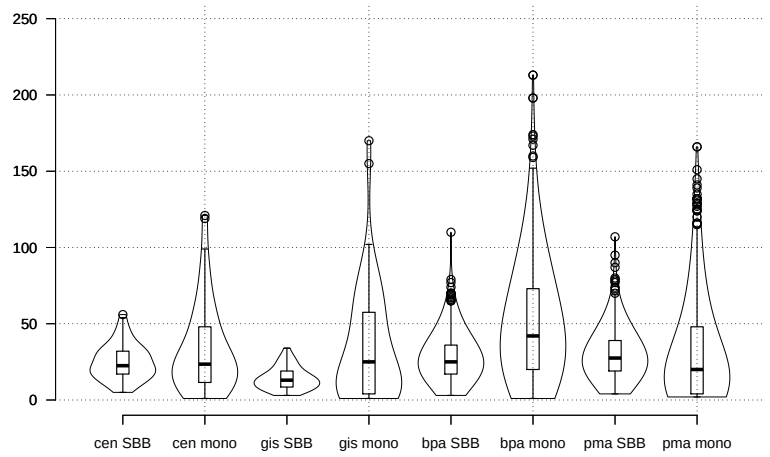

Figure 6: Total number of effective instructions per solution.

4.2 Complexity

Solution complexity will be assessed from the perspective of the number of unique attributes utilized by a solution and the number of effective instructions per solution (Section 3.3). Figures 5 and 6 present these results in terms of the corresponding combined density of distribution/ box plot. Relative to the total number of unique attributes indexed (Figure 5), it is apparent that there is little attribute reduction under Bupa and Pima, whereas both approaches provide significant reductions appear under Census and Gisette (75 and 99.5 percent reductions respectively), albeit with much greater constancy under SBB.

Monolithic solutions are also capable of identifying solutions with low instruction counts (1st quartiles in Figure 6), however, the third quartile count across the four data sets was always comparatively high (around 50 instructions). Conversely all third quartile instruction counts under SBB are considerably lower. This appears to indicate that the explicit provision of support for a divide-and-conquer approach to constructing models provides a significant benefit so solution simplicity / consistency without negatively impacting on classification performance.

4.3 CPU training requirements

Figure 7 summarizes the CPU training time under a common dedicated computing platform and generation limit. Although both monolithic and SBB algorithms share the same competitive coevolutionary and intron removal frameworks during evolution, the consistency of SBB is again readily apparent. Particularly impressive is that this consistency is achieved across very wide ranges of training partition size. Indeed, the data with the largest training partition (Census) has the fastest training time; a vindication of the shared competitive-

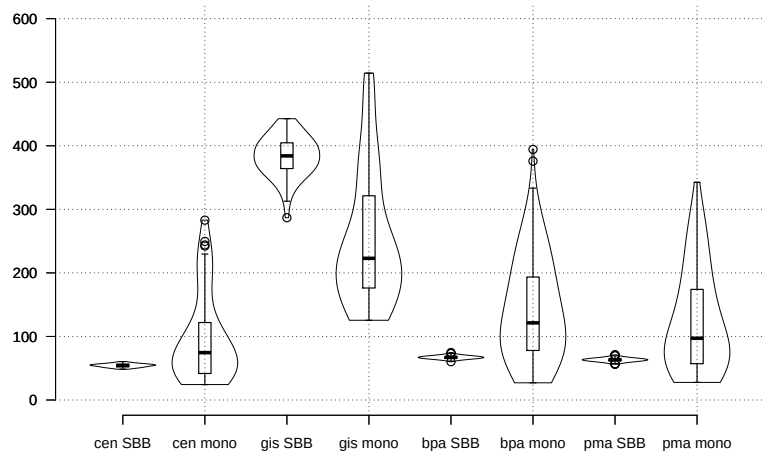

Figure 7: Training time (in seconds).

coevolutionary component of the evolutionary framework. Only under Gisette was the monolithic model significantly faster than SBB. This is likely an artifact of two properties: the relative difference in classification performance combined with the larger attribute space associated with the Gisette data set. Specifically, under Gisette, each exemplar is a vector of 5,000 attributes; whereas under Census, for example, each exemplar consists of 40 attributes or the equivalent of 125 Census exemplars for every Gisette exemplar. This will make achieving cache constancy more difficult; particularly when the problem is magnified by the need to support the identification of stronger solutions under SBB.[7]

5 SBB specific properties

The solution complexity comparison of Section 4.2 was conducted in terms of the total effective instruction count (post intron removal) and total attributes indexed to support the direct comparison between monolithic and SBB solutions. However, such a framework ignores the additional structure provided by the participation of multiple symbionts in a solution under SBB. Thus, we can also characterize solution complexity from the perspective of the number of participating symbionts – as a whole or class wise; or the uniqueness of attributes associated with symbionts. Hence provision of an explicit architecture for a divide-and-conquer approach to problem solving enables us to introduce another level of insight as to the construction of solutions which is not available under monolithic style solutions.

[7]Competitive coevolution represents a dual learning problem in which a point and learner population incrementally improve. If the learners reach a stronger behavior post training, it is fair to assume that there was a greater turn over in point population content i.e., decreasing cache hit rate.

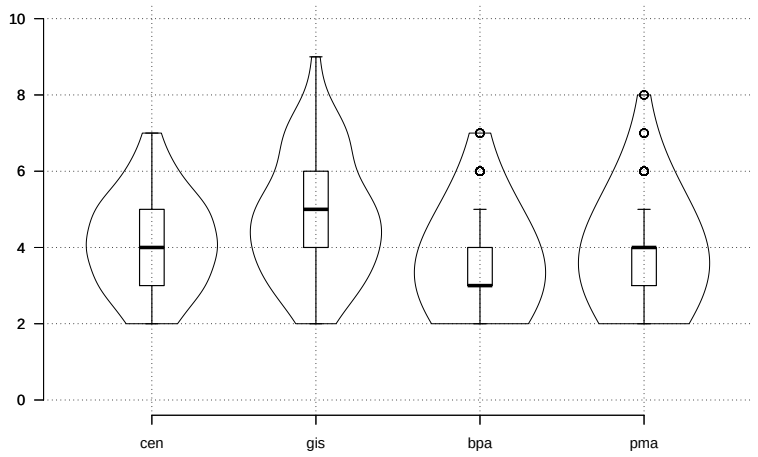

Figure 8: Number of symbionts per SBB host.

The **number of symbionts per host** – or programs per solution – is identified by first pruning any symbionts that never provide a winning bid under training conditions (Figure 8). The problem domains with the larger attribute spaces tend to result in more symbionts contributing. However, relative to the corresponding **average number of effective instructions per symbiont**, Figure 9, the much smaller data sets typically require more instructions. This appears to indicate that as the attribute count increases it is easier to build useful subspace behaviors in which subsets of class-wise behavior can be clearly determined. As this becomes less possible, the total number of cooperating symbionts decreases and the complexity of those symbionts which do cooperate increases.

This property is further reinforced by considering the **uniqueness of attributes identified by symbionts**. That is to say, if symbionts are able to locate an attribute subspace unique to their bidding strategy – relative to the total set of attributes indexed by the set of symbionts in the same host – then insight can be gained as the the structure of the problem domain. Figure 10 summarizes the percentage of attributes which are unique to symbionts in the same solution. The uniqueness of attributes under the Gisette data set is effectively 100 per cent. Even under Census with an attribute space of 40, typically 90 percent of attributes indexed by any given solution were unique to a specific symbiont; whereas the Bupa and Pima data sets demonstrated much more overlap in the corresponding symbiont attribute subspaces.

Finally, we consider the **average number of (distinct) attributes supported per action** (Figure 11). That is to say, we count the distinct attributes used by each symbiont, and average over the symbionts with the same action. We are now able to resolve attribute uniqueness on a class-wise basis. It is now apparent that attribute uniqueness is generally balanced over both symbiont actions, with Census being the only exception; requiring a lower support un-

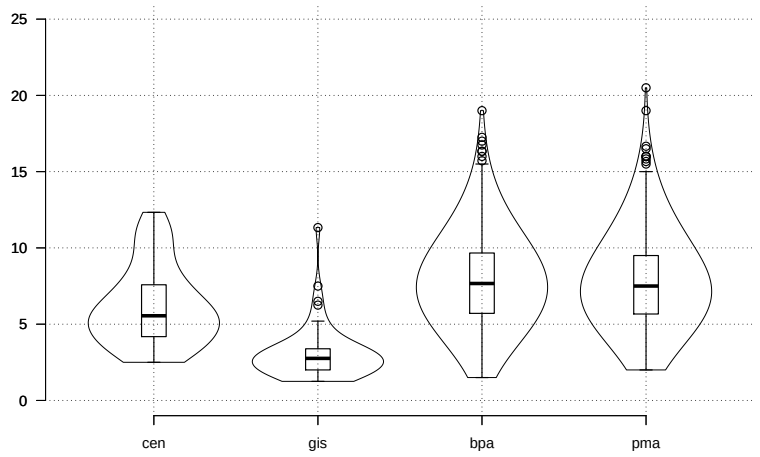

Figure 9: Average number of effective instructions per SBB symbiont.

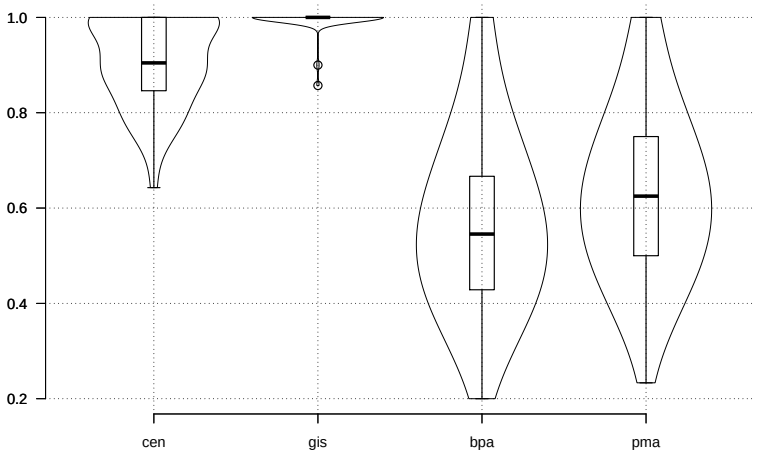

Figure 10: Proportion of attributes accessed by a host that are unique to a participating SBB symbiont.

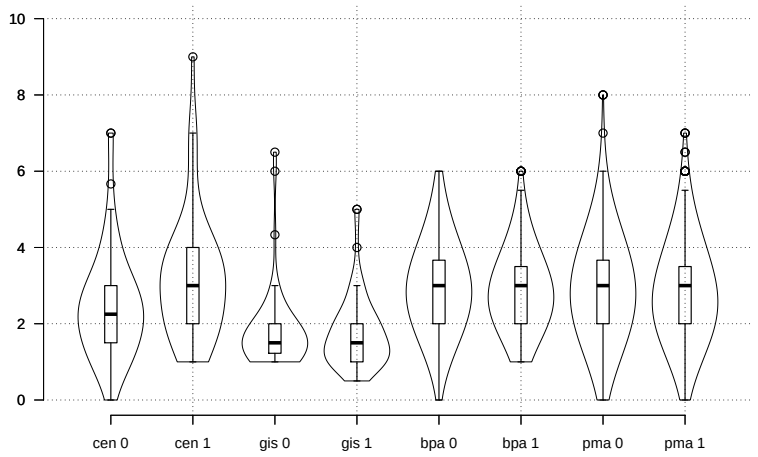

Figure 11: Average number of (distinct) attributes supported per action. 0/ 1 denote action (class).

der class 0 (majority class). Symbionts with less than one attribute are also frequently observed. This represents symbionts which learn to bid a constant value, against which the symbionts with the alternate action learn to bid dynamically. Moreover, this property is correlated with the majority class in the case of Census. Thus, symbionts representing the major class (more than 90 percent of the data) are more likely to learn the simplest bidding strategy whereas the minority class symbionts require a more complex bidding strategy to effectively model the minor class behavior.

6 Conclusion

A framework is constructed for comparing a gradualist (neo-Darwinian) model of evolution against a symbiotic model of evolution, thus an explicitly neo-Lamarckian (divide-and-conquer) process of inheritance. Naturally, the gradualist framework produces solutions in the form of a single monolithic program whereas symbiosis results in solutions taking the form of multiple programs. Both models of evolution share a common representation and utilize advanced algorithmic features such as competitive coevolution, intron removal and fitness sharing. Results from the ensuing benchmarking study indicate that adopting symbiosis provides for much more consistency than the gradualist/ monolithic model. This was apparent across multiple performance factors – classification rate, complexity, computational requirements – and data sets designed to test different factors of classifier design – non-linearity, large attribute spaces, and class imbalance. This implies that the although neo-Lamarckian inheritance requires additional support for symbiont programs to recognize context – the basis for problem decomposition – the overall advantages of this appear to outweigh

additional costs.

A case can naturally be made for and against the relative importance of including sexual search operators in gradualist models of evolution. However, the effective operation of such (crossover style) search operators is dependent on the ability to establish the specific genetic context/ alignment of the genotypic material exchanged e.g., competent selecto-recombinative models [25]. By adopting a symbiotic model of evolution we leave the search for relevant context to the combinatorial search conducted at the host and the inter-symbiont bidding, thus no genotypic knowledge is necessary at the program level to guide the process of modular model building / divide-and-conquer. However, assuming a symbiotic framework does not preclude the utility of crossover/ sexual reproduction where suitable a priori information is available to guide gene alignment.

7 Acknowledgments

P. Lichodzijewski is a recipient of Killam and NSERC pre-doctoral scholarships. M. Heywood is supported through NSERC and MITACS research grants.

References

[1] S. Ali and K. A. Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6:119–138, 2006.

[2] E. B. Baum and I. Durdanovic. Toward code evolution by artificial economies. In *Evolution as Computation*, pages 314–332. Springer, 2002.

[3] M. Brameier and W. Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001.

[4] S. Brave. Evolving recursive programs for tree search. In P. J. Angeline and K. E. Kinnear, editors, *Advances in Genetic Programming*, volume 2, chapter 10, pages 203–220. MIT, 1996.

[5] J. M. Daida, C. S. Grasso, S. A. Stanhope, and S. J. Ross. Symbionticism and complex adaptive systems I: Implications of having symbiosis occur in nature. In *Proceedings of the Annual Conference on Evolutionary Programming*, pages 177–186. MIT Press, 1996.

[6] E. D. de Jong. A monolithic archive for Pareto-coevolution. *Evolutionary Computation*, 15(1):61–94, 2007.

[7] J. Doucette, P. Lichodzijewski, and M. I. Heywood. Evolving coevolutionary classifiers under large attribute spaces. In *Genetic Programming Theory and Practice VII*, pages 37–54. Springer, 2009.

[8] I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors. *Feature Extraction: Foundations and Applications*, volume 207 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.

[9] M. I. Heywood and P. Lichodzijewski. Symbiogenesis as a mechanism for building complex adaptive systems: A review. In *EvoApplications*, volume 6024 of *LNCS*, pages 51–60, 2010.

[10] L. Huelsbergen. Learning recursive sequences via evolution of machine-language programs. In *Proceedings of the Annual Conference on Genetic Programming*, pages 186–194, 1997.

[11] J. R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT, 1992.

[12] U. Kutschera. Symbiogenesism natural selection, and the dynamic Earth. *Theory in Biosciences*, 128:191–203, 2009.

[13] M. Lemczyk and M. I. Heywood. Training binary GP classifiers efficiently: A Pareto-coevolutionary approach. In *European Conference on Genetic Programming*, volume 4445 of *LNCS*, pages 229–240, 2007.

[14] P. Lichodzijewski and M. I. Heywood. GP classifier problem decomposition using first-price and second-price auctions. In *European Conference on Genetic Programming*, volume 4445 of *LNCS*, pages 137–147, 2007.

[15] P. Lichodzijewski and M. I. Heywood. Managing team-based problem solving with symbiotic bid-based genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 363–370, 2008.

[16] Y. Liu, X. Yao, and T. Higuchi. Evolutionary Ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, 2000.

[17] L. Margulis and R. Fester, editors. *Symbiosis as a Source of Evolutionary Innovation*. MIT Press, 1991.

[18] J. Maynard Smith and E. Szathmary. *The origins of life*. Oxford University Press, 1999.

[19] A. R. McIntyre and M. I. Heywood. Pareto cooperative-competitive Genetic Programming: A classification benchmarking study. In *Genetic Programming Theory and Practice VI*, pages 43–60. Springer, 2008.

[20] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399, 1998.

[21] L. Panait, S. Luke, and R. P. Wiegand. Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Transactions on Evolutionary Computation*, 10(6):629–645, 2006.

[22] M. Potter and K. De Jong. Cooperative coevolution: an architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[23] J. P. Rosca and D. H. Ballard. Hierarchical self-organization in genetic programming. In *International Conference on Machine Learning*, pages 251–258, 1994.

[24] R. Thomason and T. Soule. Novel ways of improving cooperation and performance in ensemble classifiers. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1708–1715, 2007.

[25] S. van Dijk, D. Thierens, and M. de Berg. On the design and analysis of competent selecto-recominative GAs. *Evolutionary Computation*, 12(2):243–267, 2004.