Brock University

Department of Computer Science

The Effects of Randomly Sampled Training Data on Program Evolution

Brian J. Ross

Tech. Report # CS-99-03
November 1999

Brock University
Dept. of Computer Science
St. Catharines, Ontario
Canada L2S 3A1
www.cosc.brocku.ca

The Effects of Randomly Sampled Training Data on Program Evolution

Brian J. Ross
Brock University, Dept. of Computer Science
St. Catharines, Ontario, Canada L2S 3A1
bross@cosc.brocku.ca
(905) 688-5550 ext. 4284

Abstract

The effects of randomly sampled training data during genetic programming is empirically investigated. Sometimes the most natural, if not only, means of characterizing the target behaviour for some problems is to randomly sample training cases inherent to the problems in question. A natural question to raise about this strategy is, how deleterious is the randomly sampling of training data to evolution performance? Would such sampling reduce the evolutionary search to hill climbing? We address these questions by undertaking a suite of different GP experiments. Various sampling strategies are used, such as different training set sizes, single and multiple samples per run, and manually derived "ideal distribution" training sets. Both generational and steady–state evolution are tested, in order to see if random sampling particularly affects one or the other. Non–evolutionary search strategies, such as hill climbing and random search, are also used for comparison. The results of the experiments confirm that random sampling effectively characterizes stochastic domains during genetic programming, provided that a sufficiently representative sample is used.

1 INTRODUCTION

The most natural means of characterizing solutions for some problems is with a statistical sampling of their data distributions. Such problem characterizations can be used during the fitness evaluation step in genetic programming. For example, image processing benefits with the random sampling of image data, since using the entire image as training data can be prohibitively expensive (Poli 1996). Another example is formal language induction, in which a target language is characterized by random samples of positive and negative instances of the language (Dupont 1994, Kammeyer and Belew 1997, Longshaw 1997, Ross 2000). In some language induction problems, sampling may be the only practical means for characterizing the target language, because the precise model of its behaviour is either unknown or too complex to determine.

The question arises whether the use of random sampling for characterizing target languages has deleterious effects on genetic programming performance. A fitness function would perform random sampling when building a training set of positive examples of target behaviour. Resampling the training set during a run will correspondingly change the corresponding fitness criteria for the population. At worst, this may make fitness a "moving target", where the entire population essentially becomes obsolete when the training criteria are resampled. In such circumstances, evolution is reduced to hill climbing, in which the population is temporarily driven towards a local minimum as denoted by the current sampled fitness criteria, only to be redirected to another minimum for the next sampled example set. On the other hand, during evolution over multi-sampled example sets, a more optimistic situation is that a population may retain useful characteristics of earlier instances of training criteria. In this case, random sampling might be advantageous, since the effects of various training samples would be compounded during the entire run.

This research draws some empirical insights into the above issues. The problem domain is the induction of stochastic formal languages, and in particular, stochastic regular expressions. This problem area benefits with the use of random sampled training sets, since it is often difficult and time–consuming to manually derive example sets with ideal distributions. A suite of different genetic programming experiments are undertaken, the dependent variables including stochastic distribution complexity, random sample size, frequency of random sampling, generational and steady–state processing, and others. In addition, hill climbing and random search are also performed, as well as "perfect" training sets with ideal probability distributions.

Section 2 reviews the stochastic regular expression language. Experimental details are given in section 3. The results of the research showing the effects of random sampling of training data on evolution performance are presented in section 4. Section 5 concludes the paper with a discussion.

2 STOCHASTIC REGULAR EXPRESSIONS

Stochastic regular expressions (SRE) is a probabilistic regular language, denoted by regular expressions with probability fields (Ross 2000). SRE is similar to one in (Garg *et al.* 1999), where formal proofs of various properties of the language can be found.

Let α range over alphabet $\sum$, E range over SRE expressions, n range over positive integers ($0 \leq n \leq 1000$), and f range over decimal values with a precision of 2 decimal places ($0 \leq f < 1.00$). The syntax of SRE is recursively defined as:

$$E \ ::= \ \alpha \ | \ \sum_i E_i(n_i) \ | \ E_1 : E_2 \ | \ E^{*f} \ | \ E^{+f}$$

Without loss of generality, the empty string ϵ is not included in the alphabet (although it is included in the semantics below).

Let $Pr(E, s)$ denote the probability of expression E recognizing string s. Membership in SRE is reflected by SRE expressions returning non–zero probabilities for particular strings:

$$\begin{array}{l} s \in L(E) \ iff \ Pr(E,\ s) \ > \ 0 \\ s \notin L(E) \ iff \ Pr(E,\ s) \ = \ 0 \end{array}$$

The SRE operators have the following semantics:

1. *Atomic action* α : The action α is generated, and $Pr(\alpha,\ s) = 1$ iff $\alpha = s$.

2. *Guarded Choice* $\sum_i E_i(n_i)$, where $E = (\alpha_i : E_i')$ or $E = \alpha_i$, and $\forall \alpha_i, \alpha_j : \alpha_i \neq \alpha_j$: Here, each term in the choice expression is either prefixed with a unique atomic action that is found nowhere else in the expression, or consists of a unique action by itself. The probability of term i being chosen is $n_i / (\sum_j n_j)$, and:

$$Pr(\sum_i E_i(n_i),\ s) = \sum_k \left(\frac{n_k}{\sum_j n_j} \cdot Pr(E_k,\ s) \right)$$

3. *Concatenation* "$E_1 : E_2$" : Term E_1 is interpreted, followed by that of E_2.

$$\begin{array}{l} Pr(E_1 : E_2,\ s) = \\ \sum_{i=1}^{n} (Pr(E_1,\ \alpha_1 ... \alpha_i) \cdot Pr(E_2,\ \alpha_{i+1} ... \alpha_n) \\ \quad + \ Pr(E_1,\ s) \cdot Pr(E_2,\ \epsilon) \\ \quad + \ Pr(E_1,\ \epsilon) \cdot Pr(E_2,\ s) \end{array}$$

4. *Kleene Closure* E^{*f} : Term E can be repeatedly executed 0 or more times, and each iteration occurs with a probability of f. The probability of E terminating execution is $1 - f$.

$$\begin{array}{l} Pr(E^{*f},\ \epsilon) = 1 - f \\ Pr(E^{*f},\ s) = \\ \quad \sum_{i=1}^{n} (f \cdot Pr(E, \alpha_1 ... \alpha_{i-1}) \cdot Pr(E^{*f}, \alpha_i ... \alpha_n) \\ \quad\quad + f \cdot Pr(E, s) \cdot Pr(E^{*f}, \epsilon) \qquad : s \neq \epsilon \end{array}$$

5. *+Closure* E^{+f} : Term E executes once, after which it behaves like Kleene closure. It is an abbreviation for:

$$E^{+f} \ \equiv \ E : E^{*f}$$

(Ross 2000) evolves SRE expressions using genetic programming. There, SRE is implemented in the DCTG-GP system, which is a genetic programming system using a logic–based attribute grammar (Ross 1999). The operational semantics of SRE operators are encoded in terms of semantic attributes in the context free grammar of the language. The determination of membership properties (ie. computed probabilities) of SRE expressions can be determined in polynomial time, since regular expression membership is tractable (Sipser 1996).

3 EXPERIMENT DETAILS

The goal is to determine the positive and negative effects of random sampling of training data on genetic program evolution. A variety of possible variables may contribute to such a determination. For example, the frequency of resampling may impact the effectiveness of evolution. A run in which a new training set is resampled between every generation may behave quite differently from one with a static training set. The size of the training set is also pertinent, since a larger

Table 1: Random sampled training set sizes (avg. 25 samples)

	t=50	t=1000
L_1	10	18
L_2	9 (cutoff 2)	63 (cutoff 3)

Table 2: χ^2 between randomly sampled sets (avg. 50 pairs)

	t=50	t=1000
L_1	19.0	33.6
L_2	15.3 (cutoff 2)	142.2 (cutoff 3)

sample will be more likely to be faithful to the target distribution. Whether the genetic search uses steady–state or generational processing is also worthy of investigation. The experiments undertaken will address these and other variables.

3.1 Stochastic language complexity

The first factor to be considered is the complexity of the stochastic problem. A simple stochastic problem may have relative immunity to the effects of training set sampling during evolution, while a problem with a rich language and corresponding distribution may be more difficult to characterize by random sampling.

Two stochastic regular languages, L_1 and L_2, are studied. They are written in SRE as follows:

$$L_1: \quad (a : c^{*.5} : a)(1) \;\; + \;\; (b : c^{*.5} : b)(1)$$
$$L_2: \quad a^{*.5} : b^{*.5} : a^{*.5} : b^{*.5}$$

These language definitions can be considered to be optimal target expressions which the GP system can evolve, although that might not necessarily occur in practice. Both SRE expressions have roughly the same structural complexity. It is not our intension to compare the relative empirical behaviours of L_1's and L_2's evolution with each other. Rather, each should be considered to be a separate experiment in which we investigate the effects of random sampling.

Although language L_1 has an alphabet of size 3, its membership set is relatively simple, since only the strings ac^*a and bc^*b are possible. Furthermore, the probability of 0.5 on the Kleene closure means that long strings have increasingly low probabilities, and hence will be uncommon during training set sampling. Language L_2 has a more robust membership set than L_1. If t random samples are taken of each language, there will likely be more instances of unique strings from L_2 than from L_1. Hence L_2 can be considered to have a more complex distribution than L_1. This is discussed in the next section.

3.2 Random sampling parameters

Sampling is done via generative probabilistic regular grammars for L_1 and L_2, where productions probabilities are equivalent to those specified in the SRE expressions above. The maximum length for sampled strings is 20. The distribution complexities of L_1 and L_2 samples are outlined in Table 1. Here, t is the total number of random samples performed on each language. L_1 has 18 unique strings from amongst 1000 samples, while L_2 has 63 unique members. The cutoff values for L_2 are the minimum total frequencies of sampled strings that must arise for their inclusion in the training set. This keeps the L_2 runs within manageable time limits, since without a cutoff there are roughly respectively 35 and 200 members in the $t = 50$ and $t = 1000$ L_2 samples.

Different frequencies of random sampling ("gaps") are considered: resample every generation, resample every 10 generations, and a single sample for the entire run.

The χ^2 values between different training samples can vary significantly. Table 2 shows the average χ^2 values obtained between 50 pairs of randomly sampled training sets for L_1 and L_2.

Two "ideal" training sets (*solution sets*) are also used. They are derived so that their distribution of members accurately reflects the true distribution of strings in L_1 and L_2. There are 18 unique strings in the solution set for L_1, and 98 strings in that for L_2, again reflecting the higher complexity of L_2.

3.3 GP parameters

Table 3 lists parameters used for GP runs. Although most parameters are self–explanatory, some require explanation. The initial population is culled at the beginning of a run. Reproduction may fail, perhaps due to tree size limitations. A maximum of 3 reproduction attempts are undertaken before new individuals are selected. The minimum grammar probability values specify the minimal probability used by the SRE evaluator before an expression interpretation is preempted. This device improves the efficiency of expression evaluation by pruning interpretation paths with negligible probabilities.

Table 3: GP Parameters

Parameter	Value
Functions	SRE
Terminals	L_1 {a,b,c}, L_2 {a,b}
Fitness function	χ^2 analysis
Initial population size	750
Population size (after culling)	500
Unique population members	yes
Maximum generations	50
Maximum runs	25
Probability of crossover	0.90
Probability of mutation	0.10
Retries for reproduction	3
Max. depth initial population	L_1: 12, L_2: 10
Max. depth offspring	L_1: 24, L_2: 17
Tournament size, selection	5
Tournament size, replacement	5
Min. grammar probability	L_1: 10^{-6}, L_2: 10^{-5}

3.4 Fitness evaluation

The fitness evalution strategy used in the experiments is the modified χ^2 test used in (Ross 2000). The known distribution is taken to be the set S of test examples, and the experimental set will be the results of the SRE recognition algorithm on each member $s_i \in S$. Each test set example string is given to the SRE processor, and an overall probility $Pr(E, s_i)$ is computed. The fitness formula is:

$$Fitness = \sum_{s_i \in S} \begin{cases} \frac{(d_i - (Pr(s_i)*N))^2}{d_i} & : Pr(s_i) \geq 0 \\ \frac{|s_i| - |maxpref_i|}{|s_i|} \cdot d_i & : Pr(s_i) = 0 \end{cases}$$

where d_i is the frequency of example s_i in test set S, $N = |S|$, and $maxpref_i$ is the maximum prefix of s_i recognized. The first term is the χ^2 formula, and it is used when the example string s_i is completely recognized. The second formula is used when only a prefix of s_i is recognized, and its value is inversely proportional to the size of this prefix. Should none of s_i be recognized, then this value becomes $2 \cdot d_i$ (a normal χ^2 formula would use just d_i). This prefix scoring gives credit for recognizing a portion of the examples, which contributes additional evolutionary pressure towards the recognition of complete strings.

3.5 Other search paradigms

Random search and hill climbing are included in the experiments. Random search is implemented by setting the tournament sizes to 1 during a GP run. Hill climbing search is undertaken by taking an expression, and mutating it using conventional GP tree mutation. If the mutated expression is fitter than then original, then it replaces the original program. This is done for each individual in the population once per generation, resulting in a total of 50 mutations per expression during a run.

4 RESULTS

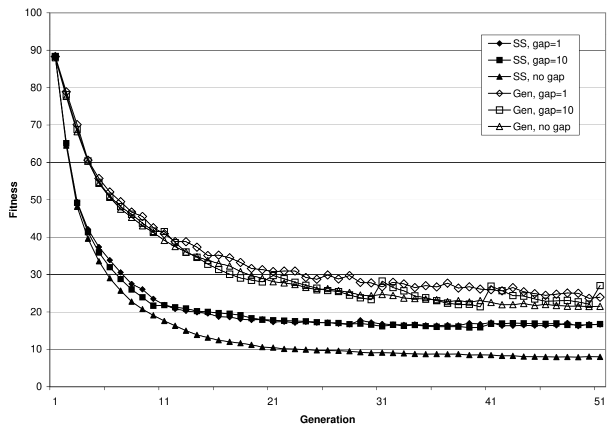

Figure 1: L_1 fitness curves (t=50, avg 25 runs)

There are two aspects of performance to consider when evaluating the effects of random sampling on evolution. First, the fitness curves as measured during runs must be studied. Fitness curves will indicate whether the population as a whole is improving with respect to the fitness criteria, and whether sampling is detrimental to the progress of search. The second criteria that must be evaluated is the absolute quality of solutions when measured against the "real" target language distribution. With the use of sampling, the fitness criteria during a run uses a statistical approximation of the target language distribution. This means that, even though an expression might closely match a given sampled training set, and hence have a good fitness score, both may be stochastically distant from the real target distribution. This will be especially acute if a single unrepresentative sample is used during an entire run. On the other hand, if the absolute solution quality is relatively good, then random sampling is not detri-

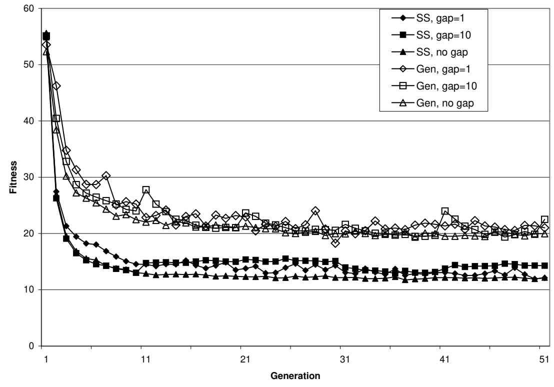

Figure 2: L_2 fitness curves (t=50, avg 25 runs)

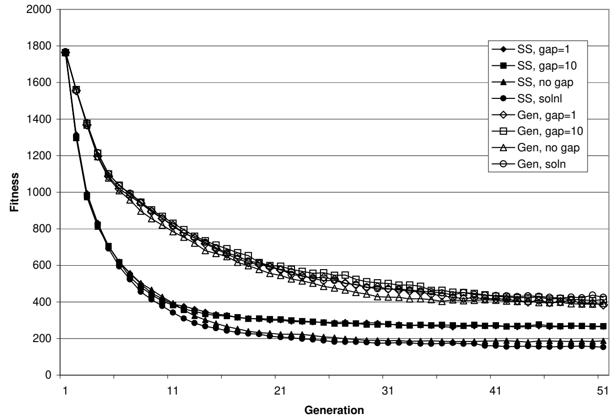

Figure 3: L_1 fitness curves (t=1000, avg 25 runs)

mental to evolution. Moreover, if random sampling is the only means for characterizing the target behaviour, then it is indeed a useful tool for genetic programming of stochastic systems.

The average population fitness curves are given in Figures 1, 3, 2 and 4. Note that the modified χ^2 test used for fitness evaluation generate values that are proportionally to the size of the training set. Hence the graphs for the $t = 50$ and $t = 1000$ experiments use different scales of fitness. One feature common in all the graphs is the natural performance difference between steady–state and generational runs, as steady–state runs converge faster and have fitter populations on the whole. In addition, all the graphs indicate that population fitness generally improves during the course of runs. This means that search is progressing, at least with respect to the sampled fitness criteria.

Hill climbing behaviour can be seen in a few experiments in the $t = 50$ graphs in Figures 1 and 2. Most notable are the curves for the generational experiment using a resampling gap of 10. Since the resampling is performed every 10 generations, there are clear increases in average fitness at generations 31, 41, and 51 in Figure 1, and generations 11, 21, 31, 41 and 51 in Figure 2. Between each jump, the population fitness improves. This indicates that the population is converging to local optima between jumps, only to be redirected towards new optima when the training set is resampled. On the other hand, the steady–state runs are not as prone to this behavior. The reason is that, with the steady–state algorithm, it is easier for programs to remain in the population long after the training set has been resampled. Fitness scores calculated in earlier generations are used in later generations, and a strong fitness score might prevent a program from being selected for replacement long after resampling has occurred. This is not the case with a generational algorithm, as the entire population is regenerated each generation, and earlier fitness scores are ignored when adding new members to newly generated populations. Nevertheless, there is still a slight curve disparity between the resampling points with the steady–state curve for the gap of 10 in Figure 2.

The curves for the $t = 1000$ experiments in Figures 3 and 4 are far less vacillating than the $t = 50$ ones. In both curves, the generational runs yield very consistent population fitnesses. This is also the case for the steady–state runs for L_2 in Figure 4. In Figure 3, the L_1 steady–state runs with no resampling (sampled without gap, and solution set) tend to perform better than the ones using resampling.

The quality of solutions obtained for the experiments are tabulated in Table 4. Lower χ^2 values indicate better statistical fits to the target languages. These statistics are calculated for 25 runs per experiment type. To get these statistics, the χ^2 value for the solution of each run is computed relative to the solution set for the appropriate target language. Naturally, this ta-

Table 4: Solution quality (χ^2). 25 runs per experiment.

Experiment			L_1					L_2				
	t	*gap*	*avg*	*median*	*min*	*max*	*std.dev*	*avg*	*median*	*min*	*max*	*std.dev*
GP (steady state)	50	1	243.3	280.8	35.7	750.6	157.9	554.2	584.8	177.4	1457.7	241.8
	50	10	235.7	258.6	36.1	479.3	122.1	572.1	470.4	95.6	1169.6	292.7
	50	–	228.0	238.1	49.2	366.3	85.9	548.5	377.2	164.6	2396.7	486.8
	1000	1	175.2	251.0	3.8	442.3	144.0	73.9	84.7	21.1	136.0	32.6
	1000	10	170.9	251.5	1.4	563.6	148.7	81.0	84.7	22.2	151.8	32.5
	1000	–	112.6	43.9	1.6	331.9	123.2	78.4	77.5	23.2	156.6	22.1
	soln	–	77.4	12.9	0.03	254.4	112.0	63.2	66.3	20.5	155.0	63.2
GP (genera-tional)	50	1	176.6	203.0	22.6	325.6	107.8	477.2	423.6	185.8	1799.2	318.0
	50	10	202.2	201.4	17.3	609.2	144.1	458.8	426.1	145.4	972.8	225.5
	50	–	188.7	220.9	37.9	292.2	90.0	580.5	488.6	163.4	1540.2	354.4
	1000	1	120.0	59.4	2.5	259.2	111.6	85.0	85.4	44.6	151.1	21.2
	1000	10	111.6	55.9	1.0	255.3	110.6	82.8	84.2	27.9	114.2	16.7
	1000	–	112.9	76.6	1.7	259.5	112.9	83.5	86.0	40.2	114.0	14.7
	soln	–	108.3	28.7	0.05	259.6	120.6	69.4	71.6	20.7	168.1	28.1
Hill climbing	50	1	815.8	860.2	513.2	1017.4	126.6	422.9	377.8	210.8	816.0	142.4
	50	10	708.0	643.9	503.1	967.4	153.7	361.1	328.6	193.1	992.6	143.2
	50	–	758.6	837.7	501.4	1090.8	178.4	339.9	318.3	227.9	670.9	86.1
	soln	–	761.3	839.2	380.1	922.6	151.7	288.4	310.3	186.5	341.2	41.7
Random	50	1	781.4	714.4	515.2	2216.3	330.5	368.4	335.3	243.0	645.8	97.4

ble should not be interpreted literally, but rather, the values should be used to gain intuition about the relative successes of the various experiments. Note that this table is analyzing the best individuals obtained from the runs plotted in the previous graphs. The χ^2 formula used for this table is measured with respect to the solution training sets, while the fitness values from the graphs are relative to the sampled training sets. Therefore, the table values do not necessarily coincide with those plotted in the graphs. Table 2 in Section 3 should also be referenced, as it shows the statistical variability arising between sampled training sets themselves.

A general observation for the genetic programming runs is that higher quality results are obtained with the large sampled training sets ($t = 1000$) than the smaller $t = 50$ samples. The best runs are produced with the fixed solution sets. This confirms that higher quality training sets will evolve better quality solutions.

Note that the frequency of resampling does not contribute a significant effect, and the resampled experiments (gaps 1 and 10) yield the same quality of solutions as the sampled gapless ones. This characteristic likely holds because multiple runs are being performed for each experiment. Should a single run be used instead, the gapless run might by chance use an exceptionally poor sample, which would result in a correspondingly poor solution.

For the L_1 experiments, all the GP runs obtained better solutions than the hill climbing and random searches. Surprisingly, this was not the case for L_2, as both hill climbing and random search produced generally better results than the GP runs that used $t = 50$ samples (although the best *min* results from the GP runs were still marginally superior). One hypothesis for this phenomenon is that these GP runs are being led astray by evolution. L_2 is considerably more complex than L_1, and unlike the L_1 runs, the $t = 50$ samples cannot adequately characterize L_2's language distribution. However, evolution still "believes" it is converging towards optima (see Figure 2). The net effect is that the GP populations converge to points that are far removed from acceptable solutions. This is supported by the high standard deviations found in the solutions for these runs. Hill climbing and random search are not as prone to this effect, simply because their populations do not converge at all, and so individuals in the population remain autonomous from one another. This results in a wider swath during the search, which is advantageous when the training criteria varies dramatically between samples.

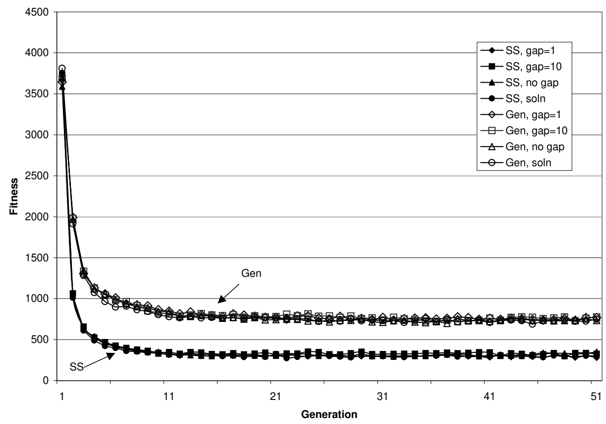

Figure 4: L_2 fitness curves (t=1000, avg 25 runs)

5 CONCLUSION

The main result of this research is that stochastic sampling of training data is not necessarily detrimental to genetic programming, provided that the samples adequately characterize the target behaviour. With our simpler L_1 stochastic language, the small sample of size 50 adequately described L_1's distribution, and hence runs discovered acceptable solutions. On the other hand, with L_2's more complex language distribution, the samples of size 50 did not model the target language very well. Exceptionally poor training sets are distinctly destructive for evolution, and in fact perform worse than random search. When the population converges to a poor suboptimal solution, genetic diversity is lost, and the population is handicapped towards changing its direction of evolution throughout the remainder of the run, should resampling be performed. When the random samples are large enough, however, few negative effects on evolution can be seen. Of course, this can negatively impact processing time, since larger training sets require more computation to process.

The experiments also found that resampling during a run does not contribute any clear advantages or disadvantages. The runs which used one sample per run performed as well as those that used periodic resampling. Should the sampling process itself consume resources, the use of a single sample per run may be preferred.

As expected, the best results are obtained with better characterizations of the target behaviour. We experienced this when using our manually derived solution sets. In many circumstances, however, such ideal training sets are difficult or impossible to obtain. In these situations, randomly sampling the behaviour of interest is an acceptable and recommended strategy. The sizes of such samples must balance efficiency of fitness calculation with the quality of solutions obtained. In any case, multiple runs should always be undertaken, since under most circumstances it is unlikely an adequate solution can be discovered in a single attempt.

Acknowledgement: This work is supported though NSERC Operating Grant 138467-1998.

References

Dupont, P. (1994). Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: the GIG method. In: *2nd Intl. Coll. on Grammatical Inference and Applications*. Springer-Verlag. pp. 236–245.

Garg, V.K., R. Kumar and S.I Marcus (1999). Probabilistic Language Formalism for Stochastic Discrete Event Systems. *IEEE Trans. Automatic Control* **44**, 280–293.

Kammeyer, T.E. and R.K. Belew (1997). Stochastic Context-free Grammar Induction with a Genetic Algorithm Using Local Search. In: *Foundations of Genetic Algorithms IV* (R.K. Belew and M. Vode, Eds.). Morgan-Kaufmann.

Longshaw, T. (1997). Evolutionary learning of large grammars. In: *Proc. Genetic Programming 1997* (J.R. Koza *et al*, Ed.). Morgan Kaufmann. Stanford University, CA, USA. pp. 406–409.

Poli, R. (1996). Genetic programming for image analysis. In: *Proc. Genetic Programming 1996* (J.R. Koza *et al.*, Ed.). MIT Press. pp. 363–368.

Ross, B.J. (1999). Logic-based Genetic Programming with Definite Clause Translation Grammars. Technical Report CS-99-02. Brock University, Dept. of Computer Science.

Ross, B.J. (2000). Probabilistic Pattern Matching and the Evolution of Stochastic Regular Expressions. *Applied Intelligence*. Accepted for publication.

Sipser, M. (1996). *Introduction to the Theory of Computation*. PWS Pub. Co.