The Effects of Randomly Sampled Training Data on Program Evolution

Brian J. Ross
Department of Computer Science
Brock University
St. Catharines, Ontario
Canada L2S 3A1
bross@cosc.brocku.ca

Abstract

The effects of randomly sampled training data on genetic programming performance is empirically investigated. Often the most natural, if not only, means of characterizing the target behaviour for a problem is to randomly sample training cases inherent to that problem. A natural question to raise about this strategy is, how deleterious is the randomly sampling of training data to evolution performance? Will sampling reduce the evolutionary search to hill climbing? Can re-sampling during the run be advantageous? We address these questions by undertaking a suite of different GP experiments. Parameters include various sampling strategies (single, re-sampling, ideal samples), generational and steady–state evolution, and non–evolutionary strategies such as hill climbing and random search. The experiments confirm that random sampling effectively characterizes stochastic domains during genetic programming, provided that a sufficiently representative sample is used. An unexpected result is that genetic programming may perform worse than random search when the sampled training sets are exceptionally poor. We conjecture that poor training sets cause evolution to prematurely converge to undesirable optima, which irrevocably handicaps the population's diversity and viability.

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 (Ross *et al.* 2000). 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 can 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 re-sampled. 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. SRE is currently being applied by the author towards problems in bioinformatics. In (Ross 2000), GP is used in the induction of SRE expressions. That application is one in which the sampling of target language strings by the fitness function is both natural and convenient. Questions arose in that work, however, regarding the effectiveness of sampling during GP, and in particular, whether sampling was detrimental to evolution. The fact that SRE language has an underlying probabilistic semantics can lend insight into the nature of the samples themselves, for example, how well they characterize the intended target language. For these reasons, SRE is used as the problem domain in this paper.

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:

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

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).

An example SRE expression is:

$$E = (a : b^{*0.7})(2) + c^{*0.1}(3).$$

Let $L(E)$ denote the language for E. The string c is a member of $L(E)$, and has a probability of 0.054 (the term with c can be chosen with a probability of $\frac{3}{2+3} = 0.6$; then that term iterates once with a probability of 0.1; finally the iteration terminates with a probability of $1 - 0.1 = 0.9$, giving an overall probability of $0.6 \times 0.1 \times 0.9 = 0.054$). The string *abbb* is also a member of $L(E)$, and has a probability of $0.4 \times 0.7 \times 0.7 \times 0.7 \times 0.3 = 0.04116$. The string *bb* is not a member of $L(E)$, and its probability is 0.

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 re-sampling may impact the effectiveness of evolution. A run in which a new training set is re-sampled 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 sample more likely will 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:

$$\begin{array}{ll} L_1: & (a : c^{*.5} : a)(1) \;+\; (b : c^{*.5} : b)(1) \\ L_2: & a^{*.5} : b^{*.5} : a^{*.5} : b^{*.5} \end{array}$$

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. Thus any differences in evolution performance are due to the complexity of the underlying stochastic languages denoted by each, rather than the relative syntactic complexities of the target expressions. The intention is not 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.

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

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

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, each with an equal distribution of 0.5. Furthermore, the probability of 0.5 on the Kleene closure means that long strings have increasingly low probabilities, and therefore are uncommon during training set sampling. Language L_2 has a more robust membership set than L_1: if k random samples are taken of each language, there will likely be more instances of unique strings from L_2 than from L_1, giving L_2 a more complex distribution. 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, k 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 cut-off 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 $k = 50$ and $k = 1000$ L_2 samples.

Different frequencies of re-sampling ("gaps") are considered: re-sample every generation, re-sample 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

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

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

obtained between 50 pairs of randomly sampled training sets for L_1 and L_2. (A low χ^2 indicates a high statistical fit; higher values indicate lower correspondences.)

Two training (solution) sets with "ideal" distributions are also used, which accurately reflect the real 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.

3.3 GP parameters

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}

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, for example, due to tree size limitations, and so 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.

3.4 Fitness evaluation

Fitness evaluation uses the modified χ^2 test from (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 probability $Pr(E, s_i)$ is computed. The fitness formula is:

$$Fitness = \sum_{s_i \in S} \begin{cases} \frac{(d_i - (Pr(s_i) * N))^2}{d_i} & (\dagger) \\ \left(1 + \frac{|s_i| - |maxpref_i|}{|s_i|}\right) \cdot d_i & (\ddagger) \end{cases}$$

where ($\dagger$) is used when $Pr(s_i) \geq 0$, ($\ddagger$) when $Pr(s_i) = 0$, 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. Note that it would be more useful to determine the longest substring recognized within the entire string, rather than the longest prefix. The SRE interpreter consumes strings from left–to–right, however, which permits efficient determination of the consumed prefix.

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 uses conventional GP tree mutation. If the mutated expression is fitter than then original, then it replaces the original. 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

There are two aspects of performance to consider when evaluating the effects of random sampling on evolution. First, the fitness curves will indicate whether the population as a whole is converging with respect to the sampled fitness criteria. Second, the absolute quality

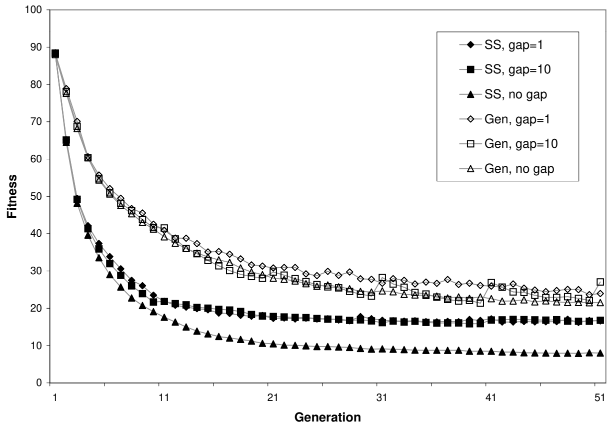

Figure 1: L_1 fitness curves (k=50, avg 25 runs)

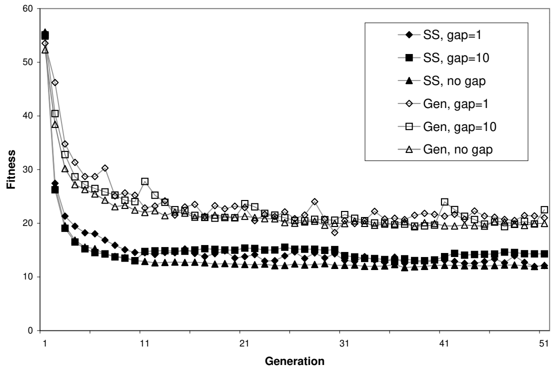

Figure 2: L_2 fitness curves (k=50, avg 25 runs)

of solutions measured against the "real" target language distribution must be considered. By definition, sampling approximates the target language distribution. 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 is especially acute if a single unrepresentative fitness sample is used for an entire run. Hence there can be disparities between the fitness scores used in a run and the absolute quality of solutions. Therefore, the quality of solutions must also be investigated.

The average population fitness curves are given in Figures 1, 2, 3 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 $k = 50$ and $k = 1000$ 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 $k = 50$ graphs in Figures 1 and 2. Most notable are the curves for the generational experiment using a re-sampling gap of 10 (white box). Since the re-sampling is done 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 re-sampled. On the other hand, the steady–state runs are not as prone to this behavior. With the steady–state algorithm, it is easier for programs to remain in the population long after the training set has been re-sampled. 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 re-sampling has occurred. This is not the case with a generational algorithm, as the entire population is regenerated each generation, and earlier fitness scores are made obsolete when new generations are created. Nevertheless, in Figure 2, there is still a slight curve fluctuation between the re-sampling points (11, 21, 31, 41) with the steady–state curve for the gap of 10 (black box).

The curves for the $k = 1000$ experiments in Figures 3 and 4 are far less vacillating than the $k = 50$ ones. In both curves, the generational runs yield very consistent population fitnesses. In Figure 3, the L_1 steady–state runs with no re-sampling (sampled without gap (black triangle), and solution set (black circle)) tend

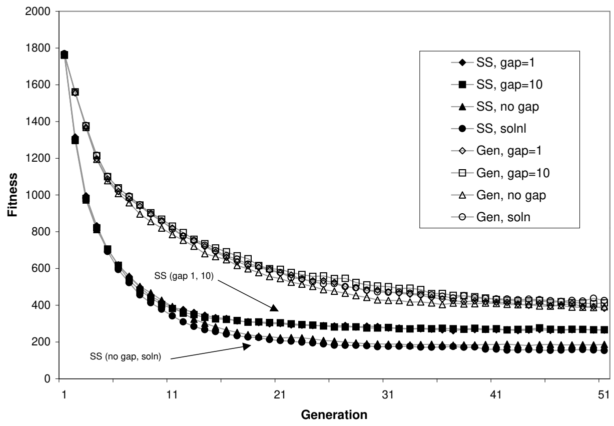

Figure 3: L_1 fitness curves (k=1000, avg 25 runs)

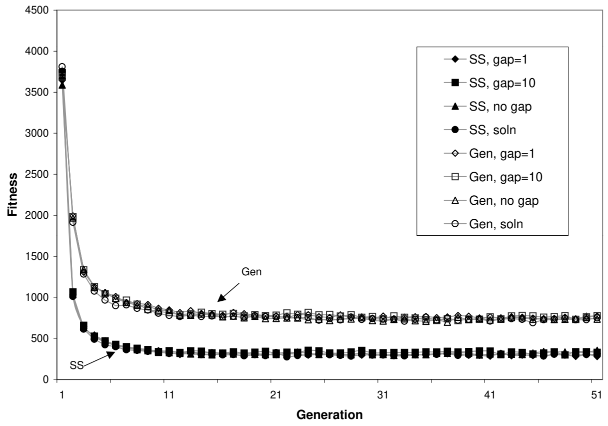

Figure 4: L_2 fitness curves (k=1000, avg 25 runs)

to converge better than runs using re-sampling. This does not imply, however, that they are converging to better quality solutions.

The quality of solutions obtained for the runs 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. To get these statistics, the χ^2 value for the solution of each run is computed relative to the solution distribution for the appropriate target language. Naturally, this table 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 the table tabulates 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 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.

First, it is clear from the table that higher quality results are obtained with the large sampled training sets ($k = 1000$) than the smaller samples ($k = 50$). The best runs are produced with the idealized solution sets. This confirms that higher quality training evolves better quality solutions.

There is a slight advantage in the runs using a re-sampling gap of 10 generations, at least for all the GP runs except the L_1 steady state runs with $k = 50$. This indicates that cumulative re-sampling more accurately characterizes target behavior, than simply sampling once for the entire run. It also appears that giving a periodic gap between re-samples is advantageous for GP, as it permits the population to converge to the current sample. When re-sampling is done every generation, the opportunity to converge is lessened.

For the L_1 experiments, all the GP runs obtained better solutions than the hill climbing and random searches. This is because the samples for all the GP runs accurately characterize L_1. 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 $k = 50$ 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 the $k = 50$ samples cannot adequately L_2. However, evolution still "believes" it is converging towards optima (Figure 2). During convergence to a poor optima, the population diversity is being significantly reduced. The net effect is that the GP populations quickly converge to points that are far removed from acceptable solutions. Moreover, they are incapable of recovering from early convergence during the rest of the run, even when

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

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

the training sets are re-sampled. This is supported by the high standard deviations found in the solutions for these runs, which indicates significant variability in their quality. Hill climbing and random search are not prone to this effect, simply because their populations do not converge at all. Individuals in their populations remain autonomous from one another, and so the population remains diverse. This results in search that is not influenced by poor training distributions, which is advantageous when the training set quality varies dramatically between samples.

5 CONCLUSION

The main result of this empirical case study 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, GP was shown to perform worse than random search. When the population converges to a poor sub-optima, genetic diversity is lost. The population becomes permanently handicapped throughout the remainder of the run, even if re-sampling is undertaken. When the random samples are adequately large, 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.

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. Multiple runs is recommended in any case.

The effects of re-sampling are not as clear, and further work is required to study the relationship between re-sampling rates and GP performance. There appears to be a qualitative advantage to runs that re-sampled every 10 generations, compared to those that re-sampled every generation or those that used one sample for the run. Periodic re-sampling may permit the cumulative effect of multiply sampled training sets, while giving the population time to converge between re-sampling points. When the samples are inadequate, however, re-sampling in GP cannot compensate for the poor performance we observed in comparison to hill climbing or random search.

Work in (Miller and Goldberg 1996, Giguere and Goldberg 1998) analyzes the effects of population and sample sizes on genetic algorithm performance. In particular, the Onemax problem is studied, which is a contrived problem amenable to precise mathematical modeling. (Miller and Goldberg 1996) establishes a predictive model of GA performance, based upon known factors such as the computational GA overhead in time per individual per generation, the cost of performing fitness sampling, the fitness variance of the initial population, and the variance of the (sampled) fitness function itself. With their model, a lower bound on the sample size is obtained which ensures particular performance criteria. (Giguere and Goldberg 1998) extends this work by determining a minimum population size given particular sample sizes and other performance parameters. A similar analysis of the effects of sample and population sizes on the performance of GP is needed. Unfortunately, the results presented in the above are not directly applicable to GP, because many of the assumptions and models used for the Onemax problem currently do not have known analogs in GP.

Future analytical and empirical studies of the effects of sampling on GP should study the effects of sampling and the structural complexity of the search space for particular problems. In addition, there is not a linear relationship between sample sizes, re-sampling rates and the quality of training samples during a GP run, and further studies on these relationships are required. One advantage of using a stochastic problem domain in such analyses, such as SRE induction done in this paper, is that stochastic measurements between the sampled training set distribution, the target problem's distribution, and program fitness, are all readily available (eg. χ^2). Further study of SRE itself should lend insight into issues such as how sample sizes correspond to overall training set quality for particular stochastic languages.

Acknowledgement: Helpful comments from anonymous referees are gratefully acknowledged. 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.

Giguere, P. and D.E. Goldberg (1998). Population Sizing for Optimum Sampling with Genetic Algorithms: A Case Study of the Onemax Problem. In: *Proc. Genetic Programming 1998* (J.R. Koza *et al*, Ed.). Morgan Kaufmann. pp. 496–503.

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.

Miller, B.L. and D.E. Goldberg (1996). Optimum Sampling for Genetic Algorithms. In: *Artificial Neural Networks in Engineering (ANNIE '96).* ASME Press. pp. 291–298.

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.

Ross, B.J., F. Fueten and D.Y. Yashkir (2000). Edge Detection of Petrographic Images Using Genetic Programming. In: *Proc. GECCO 2000* (D. Whitley *et al.*, Ed.). Morgan Kaufmann.

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