GP-Beagle: A Benchmarking Problem Repository for the Genetic Programming Community

Robert Feldt
feldt@ce.chalmers.se
Computer Engineering
Chalmers University
SE-412 96, SWEDEN

Michael O'Neill, Conor Ryan
Michael.ONeill@ul.ie
Computer Science
University of Limerick
IRELAND

Peter Nordin
Complex Systems
Chalmers University
SE-412 96, SWEDEN

William B. Langdon
CWI, Kruislaan 413
1098 SJ Amsterdam
NETHERLANDS

Abstract

Experimental studies in genetic programming often only use a few, artifical problems. The results thus obtained may not be typical and may not reflect performance on problems met in the real world. To change this we propose the use of common suites of benchmark problems and introduce a benchmarking problem repository called GP-Beagle. The basic entities in the repository are problems, problem instances, problem suites and usage information. We give examples of problems and suites that can be found in the repository and identify its WWW site location.

1 INTRODUCTION

A large fraction of genetic programming (GP) research is empirical. New ideas are implemented and tested in experiments on a number of problems. Sometimes the performance of the new idea is compared to a baseline GP system. Even though this can show the relative merit of the new idea it does not easily extend to comparing the merits of different GP extensions. Furthermore, the problems used are often artificial so the results may not be representative of the performance on real-world problems. If real-world data are used the number of different problems is often limited. This can lead to happenstance results that is not typical of the performance on the majority of problems.

In this paper we introduce GP-Beagle, an infrastructure for establishing, maintaining and promoting a publically available repository of benchmarking problems for empirical investigation and performance evaluation of genetic programming systems. It includes both individual problems and benchmarking suites of problems. It defines a nomenclature and structure for different entities related to benchmarking, specifies attributes needed to describe each problem and suite and lists the publications in which they have been used. Inspired by the recent successes of the open source movement the repository is available under a GPL-like usage agreement where the use of the problems is free but published results must be reported back to the repository. This ensures that the repository can give an up-to-date view of the use of the problems and the knowledge gained. The GP-Beagle effort is supported by a WWW site, currently under development, at http://www.gp-beagle.org.

We believe that GP-Beagle will enable the GP community to make faster progress since it will promote the use of sound experimental methods, provide a common ground for comparisons, enable faster elimination of ideas that are not fruitful and evoke discussions about the problems we use in our research and their respective merits. However, this effort will be successfull only if we all, as a research community, make use of and extend the repository. We hope to convince you that taking part in this effort will be beneficial both to you and to the community as a whole.

There are a number of existing problem databases in areas related to GP and much can be gained by using them [1]. However, we think a new repository is needed for GP since GP can attack other types of problems and existing databases are mainly pools of problems and do not give an up-to-date view of the use of the problems. Furthermore, establishing a community-specific repository have the potential of raising the awareness and use of experimental studies far more than the act of pointing to existing databases.

Section 2 elaborates on experimental research and the pros and cons of using benchmarks. In section 3,

[1] One example is the UCI Machine Learning repository with about 100 classification and regression data sets [1].

the components and structure of GP-Beagle are introduced and in section 4 we detail the attributes used to describe the entities in the repository. Examples of problems and suites in the repository are described in section 5 and section 6 concludes the paper.

2 EXPERIMENTAL RESEARCH AND BENCHMARKS

Experimental research is an important part of the scientific method and it's importance in computer science has been recently stressed [7] [2]. Even though experiments can never prove a theory they allow us to test theoretic predictions in reality and to explore areas where theory can not (yet) reach. The main benefits of conducting experiments is that they help build a reliable knowledge base of adequate theories and methods, they give observations that can lead to unexpected insights and that they accelerate progress since they help to quickly eliminate unfruitful approaches and weed out erroneous claims. Experiments thus guides engineering practice and theory development in promising directions.

We know of no studies of the current level of experimental practice in GP research. Studies on the neural network community and computer science in general have revealed that the amount of experimental evaluation is low [5] [8]. A study of 190 neural networks articles published in 1993 and 1994 showed that only 8% presented results for more than one real-world problem, 29% did not employ even a single realistic learning problem and one third did not present any quantitative comparison with a previously known algorithm [5]. Even though some of the efforts in the ANN community to raise the level of experimental assessment have probably "spilled over" to the GP community we suspect that the situation in the GP community is not much different. A collective strive for better assessment practices thus seem called for.

Benchmarks are an effective and affordable way of conducting experiments and have been successfully used in many areas [3]. A benchmark is a collection of problems with well-defined performance measurements and a prescribed method how to evaluate performance. If they are chosen in a good way they allow repeatable and objective comparisons. The essential requirements on a benchmark are (based on [6]):

- *Volume*: the benchmark should include several and diverse problems,
- *Validity*: common errors that invalidate the results should be avoided,
- *Reproducibility*: problems and experiments should be documented well enough to be reproducible,
- *Comparability*: results should be comparable with the results in other studies.

Conducting experiments with only a few problems makes it difficult to characterize a new algorithm or extension. If only problems of the same type are used the results may not show the typical performance. By including several problems of different types we can get a fuller picture of how the algorithm performs in general.

Methodological errors that threaten validity include the choice of a problem suited to the investigated algorithm, reporting the result on a data set that was used for training or using the test data set for tuning parameters. These errors can be avoided by using a well-defined evaluation procedure that separates between training, validation and testing.

If a paper does not describe the exact setup of an experiment the result can not be reproduced.

If results of different studies can not be compared it is difficult to choose between algorithms and ideas proposed in different studies. This slows down progress.

In addition to these four requirements, practitioners have the requirement of *representability*: benchmark problems should resemble the problems met in the real world. A risk with using artificial problems is that they have a limited information content [4] so that there is no room to discover and exploit different layers of complexity. For example, there might be no use in having a meta-learning ability, such as assembling information on search directions while searching, on the multiplexer problem. Comparing machine learning algorithms on simple problems with only one, central "idea" to "get" might evaluate problem solving ability in an unfair way.

The use of benchmarks has some disadvantages. One risk is that algorithms are specifically tailored to perform well on the benchmark problems. Another risk is that benchmarks focus too much on a single, numerical performance measure. This can hinder progress

[2]This section draws heavily on the two papers [7] and [6].

[3]A notable example are the "spec" problem suites used for benchmarking computer performance.

[4]In an information theoretic sense. For example, a scalable artifical problem, such as Gaussian described in table 1, has the same information content (kolmogorov complexity) regardless of how the parameters are varied.

because researchers optimize a local optima instead of exploring new and innovative avenues of research. Another problem is that it is not clear how fair comparisons should be carried out. For example, it might be unfair to compare the accuracy of two GP algorithms without taking their execution time or the time needed to set them up or the time needed to tune their parameters into account. Finally, benchmarks have to evolve with the needs of the community and application areas; if they are static they will fail to reflect new knowledge and will thus become irrelevant.

At the current level of maturity of experimental practice in the GP community we think that the advantage of establishing and using common problems and benchmarks outweighs these potential drawbacks. By constantly remind ourselves of these pitfalls their negative effects can be avoided. Furthermore we have designed GP-Beagle to explicitly try to address them.

3 THE GP-BEAGLE PROBLEM REPOSITORY

GP-Beagle is designed to be a one-stop place for all information on GP problems and benchmarking suites of problems. The basic philosophy is that GP-Beagle should define an open framework that can be easily extended were suites of problems can evolve as knowledge is gained on them and the algorithms they are used to evaluate. Thus, GP-Beagle does not simply supply a number of problems, it also collects and presents information on their use. To guarantee that the usage information is up-to-date the problems are supplied under a usage agreement. The agreement states that the problems can be freely used but that information on their use should be reported back to the repository. It also encourages researchers to submit new problems to the repository. Any problems are accepted as long as they meet basic criteria (has been used in published work and several instances of the same type of problem are not already in the repository).

Since evolutionary algorithms are general search algorithms that can be applied to a large number of areas it would not be wise to specify one benchmark suite to be used in all research. GP-Beagle does not prespecify a number of suites but starts by recording the collections of problems that are actually used. Thus, the suites are de facto collections of problems. Over time it is anticipated that special suites will evolve for different sub-areas of GP research such as for example classification, regression or artificial problems. It is also anticipated that when a mass of problems and usage data have been assembled suites can be constructed in a rigorous way, using recent ideas on how to quantify the features of benchmarking suites [3].

GP-Beagle is implemented on a WWW server as a set of Perl-scripts accessing a MySQL database. The database consists of records for each of the basic entities: problems, problem instances, de facto and benchmarking suites and usage information. This implementation minimizes [5] the amount of human resources needed to maintain the repository. Statistics on the use of problems in the repository can be automatically collected. The structure of the repository and the GP-Beagle usage agreement is further described below. Section 4 gives a detailed view of the entities in the repository.

3.1 STRUCTURE OF THE REPOSITORY

The basic entity of the repository is a *problem*. A problem is either a data set, a data generator or a simulator. Both of the latter are programs that generate data to be used in fitness evaluation. The difference is that a data generator is used off-line, ie. by generating a data set prior to starting the GP run, while a simulator is used on-line in a dynamic evaluation of a GP individual. A problem can be either artificial or real-world.

Specifying which problem has been used in an experiment is not enough to allow full reproducibility and comparability of results [6] [2]. For instance it is not enough to specify which data set has been used; one must describe how the data set have been divided into training, validation and testing sets. For a simulator or data generator we need to know which parameters have been used, how many fitness cases have been generated and so forth. To encompass this level of detail GP-Beagle introduces the concept of a *problem instance*. This is a fully specified description of the problem and how it has been used [6]. Thus, each problem in the repository can have multiple instances but each instance can only stem from one problem. A problem defines a family of possible instances.

A collection of problem instances that have been used together in an experimental study is called a *problem suite*. A *homogeneous* suite consists of problem instances from the same problem, while a *heterogeneous* suite have instances from several problems.

[5] Human assistance will be needed to review that new submissions to the repository are complete, to create new benchmarks etc.

[6] An instance may contain multiple samples from the same problem data set.

A special kind of problem suites are the *benchmarking suites*. These suites are not de facto suites that have already been used in actual research. Instead they are explicitly added to the repository to promote new kinds of experiments or to define suites consisting of diverse problem instances.

In addition to these four basic entities the GP-Beagle repository contains usage information. The usage information details in which studies each problem instance and suite have been used and the results and knowledge obtained. This information can be easily accessed when browsing the repository. GP-Beagle also collects statistics on the use of problems so that hot-lists can be presented. This way a researcher can easily find the problems that are often used and that would thus give good opportunities for comparative analysis.

3.2 THE GP-BEAGLE USAGE AGREEMENT

The problems in the GP-Beagle repository are available free for any academic or commercial use as long as any published information generated by this use is reported back to the repository. Specifically the information that should be reported includes (general and suite-specific information):

1. Reference to paper where the experiment is described, and
2. The set of problem instances used, and
3. The goal of the experiment and a rationale for choosing this specific set of problems (if any), and
4. Any knowledge gained on the set of problems such as their suitability for achieving the goal.

and for each problem instance used (instance-specific information):

1. The result obtained on the performance measure defined for the problem instance, and optionally
2. The execution time.

A new problem instance can be generated or an existing instance can be altered as long as the new instance is supplied back to the repository together with the following information:

1. The reason for creating the new instance, and
2. A description of why the previously existing instances was not adequate.

4 ATTRIBUTES OF ENTITIES IN GP-BEAGLE

The following attributes are kept in a record on a problem in the repository:

- Name: A unique name for the problem. Once assigned the problem will always have this name and can thus be uniquely referred to in papers and discussions.
- Description: A textual description of the problem. Should ideally give some basic knowledge on the domain, describe the parameters in a DataGenerator or Simulator, if attribute values are missing in a DataSet etc.
- Version: A version number to reflect updates to the problem.
- Type: DataSet / DataGenerator / Simulator
- Sub-type: Regression / Binary Classification / 5-Classification etc.
- Origin: Artificial/Real-world. Artificial problems are further characterized as whether their difficulty can be varied.
- Source: Who submitted the problem.
- Status: Suggested / Reviewed. Indicates if the problem have been reviewed and thus “officially” entered the repository.
- Number and type of attributes: Total number of attributes, number of continous and discrete attributes.
- Number of instances: Number of instances in a DataSet.
- File: A gzip:ped tar file with all the files in the problem.

The unique attributes of a problem instance record:

- From problem: The problem that the instance is derived from.
- Description: Describes how the instance was derived from the “parent” problem, what components it consists of, why previously existing instances of this problem was not adequate etc.
- Reason created: Reason for creating the instance.

- Performance measure: Describes the “fitness” value used to evaluate algorithms on the instance.
- Number of instances: Number of instances that can be used in evolving a solution (ie. these instances can be divided in validation and training sets).
- Number of test instances: Instances in test set that cannot be used in any way to evolve a solution.
- GP result: Give an example of a good result obtained with a GP technique.
- GP paper: Pointer to a problem instance usage info record describing the paper in which the good result was obtained.
- Other result: Give an example of a good result obtained with a non-GP technique.
- Other paper: Briefly describe the technique used and give reference to paper where result can be found.
- Simple result: Give result achieved with a simple technique (for example plurality rule in classification task or a technique based on linear separation in regression).

The record for a suite contains the following unique attributes:

- Type: DeFacto / Benchmark.
- Problem instances: Instances in the suite.
- Sub-type: Heterogeneous / Homogeneous
- Performance measure: Performance measure for suite.

In addition to the above, basic entities the repository contains two types of usage information records: instance usage info and suite usage info. The unique attributes of the instance usage info are (the suite usage info record is similar):

- Paper: Paper where experiment with instance is described. Pointer to GP bibliography.
- Technique used: Algorithm or technique used.
- Performance obtained: Performance obtained.
- Time: Execution time to evolve a solution with the performance above.

We have contemplated using a standardized way to report the execution time but we do not think that one “right” way to do it is yet available. One possible way would be to report the actual execution time normalized with the spec benchmark result for the CPU used as in [4]. However, a number of objections can be raised to this scheme so we have chosen not to specify one way on how to measure the time needed.

5 EXAMPLES

Below we give examples of some entries in the repository. One is a problem, one is a problem instance, one is a de facto suite and one is a proposed benchmark. The descriptions are brief and primarily intended to give you a picture of the kind of information that can be found in the repository. More details can be found at the GP-Beagle web site.

5.1 PROBLEM: Gaussian(n,μ_1,σ_1,μ_2,σ_2,f,f_l,f_h)

The Gaussian problem is a DataGenerator problem. It’s record in the GP-Beagle database is shown in table 1. The data file for the problem, gaussian.tar.gz, contains the following files:

- readme.txt - A description of the files included in this tar file, and
- gaussian.description - The data from the record shown in table 1, and
- gaussian.c - The DataGenerator implemented in ANSI-C, and
- usage.info - Description of how to compile and use the DataGenerator, and
- data.info - Description of the data file generated when the generator is run.

The files are typical of what should be included for a DataGenerator problem; they will differ for other types of problems.

5.2 PROBLEM INSTANCE: KddCup99-disctoint-1%

The KddCup99-disctoint-1% is a problem instance sampled from the KDD Cup 1999 data (a real-world 5-class classification DataSet problem). The problem instance record is shown in table 2. Note that the reference to the GP paper is given as the bibtex key in the

Table 1: Record for the Gaussian problem

Name: Gaussian(n,μ_1,σ_1,μ_2,σ_2,f,f_l,f_h)		
Type: DataGenerator	**SubType:** Binary Classification	**Version:** 1.0, 2000-05-22
VariableDifficulty: Yes	**Status:** Suggested	**Origin:** Artificial
# Instances: Varying	**Attributes:** Varying # of numerical	**File:** gaussian.tar.gz
Source: Carla Fredrica Gauss, cfgauss@math.rocks.org		
Description: Discriminate instances generated from either of two multivariate (n attributes) gaussian distributions with mean and stddev (μ_1, σ_1) and (μ_2, σ_2), respectively. The 'f' parameter governs how many false input attributes, uniformly sampled on [f_l,f_h], should be added to each instance. The difficulty of the problem (dimensionality, Bayes optimal classification rate and number of false attributes) can be varied by varying the parameters of the problem. The Bayes optimal classification rate (ultimate uncertainty in problem which no ML algorithm can do better than) can be calculated for parameter choices with f equal to 0. Generalization of a problem from Elena project.		

Table 2: Record for the KddCup99-disctoint-1% problem instance

Name: KddCup99-disctoint-1%		
FromProblem: KddCup99	**Status:** Suggested	**Version:** 1.0, 2000-05-18
# Instances: 48984	**# TestInstances:** 311029	**File:** kddcup99-disctoint-1.tar.gz
PerformanceMeasure: Average cost per test instance according to specified cost matrix		
Source: Catherine Darwin, cdarwin@evolution-rules.com		
Description: The data used in the KDD Cup 1999 competition had more than 4 million training instances and 311,029 testing instances. This problem instance contains a 1% sample of the training instances but all of the testing instances. The "disctoint" refers to the mapping from discrete input attributes to numerical integers. The task is relatively difficult since the class distribution in the test set is different from the class distribution in the training set.		
ReasonCreated: We wanted to test if a GP system can get competitive results even with the simplest possible mapping (mapping the values of an unordered discrete attribute to integers imposes an order that does not exist in the original data). We took a 1% sample because we wanted to get a more manageable data set that would give shorter execution times. The test set was kept intact since we wanted to be able to compare to the results of the algorithms in the KDD Cup.		
GPResult: 0.1985	**GPPaper:** gpbiblio:darwin:ieeetroec:2001	
OtherResult: 0.2331 with bagged and boosted decision trees (winner KDD Cup'99)		
OtherPaper: Elkan, C.: Results of the KDD'99 Classifier Learning Contest, http://www-cse.ucsd.edu/users/elkan/clresults.html, May 2000		
SimpleResult: 0.5220 with plurality rule and 0.2523 with a 1-nearest neighbor classifier.		

GP bibliography. We are planning to implement connections between GP-Beagle and the GP bibliography so that papers can easily be located and searched.

5.3 DE FACTO SUITE: Proben1-medical

A recent paper by Brameier and Banzhaf used six problems from the Proben1 benchmark suite to compare GP performance to that of neural nets [2]. Each problem used had three different samples of the same data set. We have put these three samples in the same instances and thus this de facto suite contains 6 different problem instances. Its record is shown in table 3 [7].

5.4 BENCHMARK SUITE: Classification-diverse18

To give an example of a benchmark suite we have created one by adding two large classification problems to the suite of 16 classification problems used in [4]. Note that the KddCup99-disctoint-1% problem instance described in table 2 is one of them. The record is shown in table 4. Also note that some of the problem instances used are from the same problems used in the Proben1-medical suite above. Since a different sampling and evaluation procedure (10-fold cross-validation vs. 3-fold cross-validation) was used in this suite the instances are distinct even though they stem from the same problems.

6 CONCLUSIONS

We have described GP-Beagle, an infrastructure for establishing, maintaing and promoting a publically available repository of benchmarking problems for empirical studies of genetic programming systems. By using benchmarks the genetic programming community can make faster progress since results from different studies can be more easily compared. Furthermore, benchmarks chosen in a good way promotes sound empirical studies since they include a broad and diverse set of problems and prescribe the evaluation procedure and performance measurements to be used.

To address some of the pitfalls of using benchmarks GP-Beagle is an open framework where benchmarks and problems can evolve; we have not pre-specified some benchmarks that must be used. We anticipate that over time the GP community, in a collective effort, can assemble benchmarks for different sub-areas of GP research in the framework supplied by GP-Beagle.

[7]In the paper, Brameier and Banzhaf does not report an aggregated performance measure as is indicated in table 3.

The basic entities in GP-Beagle are problems, problem instances and problem suites. Problem instances are concrete instances of a problem with a full description of how they should be used. They allow for full reproducibility of results. The repository also contains information on the use of the problems and suites. All problems are freely available as long as published results and problem extensions are reported back to the repository.

GP-Beagle is implemented as a set of records in a MySQL database. Perl scripts are used to extract information and update the data base. The interface to the repository is via a web site at http://www.gp-beagle.org. In order for this effort to really take off we encourage you to visit the site, start using the repository and submitting your problems and results.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] Markus Brameier and Wolfgang Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, in press, 2000.

[3] Jozo J. Dujmovic. Universal benchmark suites. In *Proc. 7th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 197–205, 1999.

[4] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, Forthcoming, 2000.

[5] L. Prechelt. A quantitative study of experimental evaluations of neural network learning algorithms: Current research practice. *Neural Networks*, 9(3):457–462, 1996.

[6] Lutz Prechelt. Some notes on neural learning algorithm benchmarking. *Neurocomputing*, 9(3):343–347, 1995.

[7] W. Tichy. Should Computer Scientist Experiment More? *IEEE Computer*, 31(5):32–40, 1998.

[8] Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz. Experimental evaluation in computer science: A quantitative study. *The Journal of Systems and Software*, 28(1):9–??, January 1995.

Table 3: Record for the Proben1-medical de facto suite

Name: Proben1-medical		
Type: DeFacto	**Status:** Suggested	**Version:** 1.0, 2000-05-24
# Instances: 6	**Id number:** 1	**File:** proben1-medical.tar.gz
Instances: Cancer-proben1, Diabetes-proben1, Gene-proben1, Heart-proben1, Horse-proben1, Thyroid-proben1		
PerformanceMeasure: Average classification error on the 3*6=18 test sets		
Source: Markus Brameier and Wolfgang Banzhaf (originally from the Proben1 benchmark), banzhaf@not.valid-email.de		
Description: A subset of six medical classification problems was extracted from the Proben1 neural network benchmark. Each instance consists of three different samples from one and the same problem.		

Table 4: Record for the Classification-diverse18 benchmark suite

Name: Classification-diverse18		
Type: Benchmark	**Status:** Suggested	**Version:** 1.0, 2000-05-24
# Instances: 18	**Id number:** 2	**File:** classification-diverse18.tar.gz
Instances: Cancer-lim, Cmc-lim, Dna-lim, Heart-lim, Boston-housing-lim, Led-lim, Liver-lim, Pima-indians-lim, Satimage-lim, Image-segmentation-lim, Smoking-lim, Thyroid-lim, Vehicle-lim, Voting-lim, Waveform-lim, Ta-evaluation-lim, KddCup99-disctoint-1%, KddCup98-disctoint-5%		
PerformanceMeasure: Average classification error rate		
Source: Robert Feldt, feldt@ce.chalmers.se		
Description: A broad and diverse suite of classification problems. Includes five binary, seven ternary, one 4-class, two 5-class, one 6-class, one 7-class and one 10-class classification problems. On “small” problems (less than 1000 instances in test set) 10-fold cross-validation is used to estimate the classification error rate. Sixteen of the problems have been used on 33 different ML techniques in a study by Tien-Sien Lim et al. This allows for comparisons to a large number of machine learning algorithms. Two additional data sets from the 1998 and 1999 KDD Cup competitions were added to the benchmark because many of the problems used in the Lim et al study was “small”.		