
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, artificial 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 successful 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 them¹. 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,

¹One 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 its 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 areas³. 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

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

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

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 content⁴ 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

⁴In an information theoretic sense. For example, a scalable artificial 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 pre-specify 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 minimizes⁵ 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 used⁶. 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.

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

⁶An 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 continuous 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.

