Towards a Framework for the Empirical Analysis of Genetic Programming System Performance

Oliver Flasch and Thomas Bartz-Beielstein

Cologne University of Applied Sciences,
{oliver.flasch, thomas.bartz-beielstein}@fh-koeln.de

Abstract. This chapter introduces the basics of a framework for statistical sound, reproducible empirical research in Genetic Programming (GP). It provides tools to understand GP algorithms and heuristics and their interaction with problems of varying difficulty. Following a rigorous approach where scientific claims are broken down to testable statistical hypotheses and GP runs are treated as experiments, the framework helps to achieve statistically verified results of high reproducibility. A prototypic software-implementation based on the R environment automates experiment setup, execution, and analysis. The framework is introduced by means of an example study comparing the performance of a reference GP system (TinyGP) with a successively more complex variants of a more modern system (GMOGP) to test the intuition that complex problems require complex algorithms.

Keywords: Genetic Programming, Symbolic Regression, Design of Experiments, Sequential Parameter Optimization, Reproducible Research, Multi-Objective Optimization

1 Introduction

The goal of this chapter is to introduce a framework for the systematic empirical analysis of Genetic Programming (GP) system components and their influence on GP system performance. Main ideas and concepts are borrowed from the empirical approach to research in evolutionary computation described in [1]. Performing a thorough and statistically well-founded experimental analysis provides valuable insight into the behavior of GP components.

In current GP research, much repeated work in experimental planning, setup and result analysis is required when proposing improvements in GP system components such as selection and variation operators, individual representations, or general search heuristics. To measure the performance benefit of an improved GP system component, a set of test functions has to be implemented, GP system parameters, both of the system under study as well as of comparison systems have to be chosen, experiments have to be designed and code for the statistical result analysis has to be written. As obtaining results of statistical significance often requires many independent runs, infrastructure for distributed execution is often necessary to render the implementation of an experiment plan practical.

Although this effort is necessary to participate in GP research, it might constitute a significant barrier of entry to newcomers to the community. Even worse, time spent re-implementing necessary GP research infrastructure is lost for working on core issues. As each group working in GP research has to provide not only their own GP system, but also proprietary and complex supporting infrastructure often tightly coupled to local conditions, reproducing results by other groups is often more difficult than needed.

Fortunately, much of the necessary infrastructure for conducting reproducible research in GP could be provided as a framework of reusable building blocks and made available as open-source software. Laying the foundation of this framework defines the goal of this work.

In recent years, several high quality GP systems have been developed that could constitute the basis of such a framework. For example, ECJ provides a modular open-source GP implementation that is easily extensible. [19] Eurequa is a modern GP system featuring a user-friendly graphical user interface that makes solving real-world symbolic regression problems with GP very simple and intuitive. [15] The system also supports large-scale runs on compute clusters. While freely available, extensibility through third parties is somewhat limited by the fact that the system is distributed in closed-source form. On the other hand, third party extensions are available that tightly couple Eurequa to computing environments like Mathematica and MATLAB. DataModeler is another modern GP system mainly geared to real-world symbolic regression. [18, 10, 17] Embedded in Mathematica, which provides sophisticated tools for data manipulation, statistics and visualization, the system is ideally suited as a basis for the framework suggested in this chapter. While DataModeler is a commercial system, academic licencing is handles liberally. The system is also extensible by third-party GP operators implemented in Mathematica code. RGP[1] is an open-source GP system hosted by the statistical software environment R. [5, 14] Compared to the aforementioned system, RGP is still quite immature, yet all basic features of a modern multi-objective typed GP system are implemented. As most performance critical operators feature alternative optimized implementations in the C programming language, this system is fast enough to be used on real-world problems. As RGP has been designed as a research tool, the system has a highly modular design. All components are extensible and replaceable on a fine-grained level. The interactive nature of the R environment alleviates quick experimentation and rapid prototyping, while performance critical-components can be easily re-implemented in compiled languages such as C.

In the future, the framework introduced in this chapter will provide a general interface to support all modern GP systems. Support for DataModeler will be provided by the authors. Presently, a prototype hosted by the statistical software environment R exists, supporting the RGP system. To support the effective and efficient experimental planning, setup and analysis of GP runs, the framework provides the following features:

[1] The RGP package is available via the Comprehensive R Archive Network (CRAN) at http://cran.r-project.org/ or directly at http://rsymbolic.org/projects/show/rgp.

- a modular GP system fast enough for real-world applications (RGP)
- a set of test problems of scalable difficulty
- tools for automatic parameter tuning based on the sequential parameter optimization toolbox (SPOT) [2]
- experiment design and setup based on SPOT
- tools for statistical analysis of GP results based on the R environment
- tools for result visualization and automated result reporting
- support for distributed execution on PBS compute clusters

The framework is still under active development and will be published as a supplemental part of the first author's PhD thesis. A preliminary version in form of an R package will soon become available on the RGP website at http://rsymbolic.org/.

This chapter will illustrate the framework by means of a small but realistic example study. To provide baseline results, the performance of the simple and well-known TinyGP system is examined on a small set of scalable symbolic regression test problems. Then, the performance benefits of several improvements to a more modern yet still simple GP system, Generational Multi-Objective GP (GMOGP), are studied empirically.

Thus, the remainder of this chapter is structured as follows: Section 2 introduces the different GP search heuristics considered in the example study. Scalable test problems used in the study are described in Section 3. Section 4 introduces the experimental setup. This section also formulates research questions relevant to the study. Results are described in Section 5, which is followed by a discussion and conclusions in Section 6. Here, answers to the research questions posed in Section 4 are formulated. As the framework introduced in this chapter is work in progress, the chapter closes with a outlook to further extensions, given in Section 7.

2 GP Search Heuristics

In this work, the term GP *search heuristic* denotes the concrete search strategy used in a GP system, which is in principle independent of the concrete GP search space. Classical GP uses a steady state Genetic Algorithm (GA) with tournament selection to search genotype space. In the GP literature, many different concrete variants have been described. It is possible to de-couple the search heuristic from the search space, giving rise to a wide variety of possible hybrid algorithms. An example is the evolution of support vector machine kernels, by including a memetic approach of kernel constant optimization. [7] Of course, it is also possible to use search heuristics from outside the field of EAs in GP search.

Historically, every GP system implemented slightly different search heuristics, while exhaustive comparisons of GP search heuristics, isolated from the concrete GP search space, are scarce. Many modern GP systems often employ multi-objective evolutionary algorithms (EMOAs) as search heuristic. For historical reasons and to simplify parallelization on shared memory multiprocessors, steady state algorithms with Pareto tournament selection seem to be the predominant EMOA variants used in todays best-performing GP systems. In simple GP systems mainly designed for research and teaching, single-objective steady state EAs with tournament selection are still widespread.

This section describes the two GP search heuristics examined in this study. For both heuristics, the general algorithmic framework and the concrete implementation of the selection operator `sel` is described. GP search heuristics may be classified as generational, steady state, or generation gap algorithms. The *variation pipeline*, i.e. the concrete setup and order of application of variation operators (recombination and mutation) is defined, as some GP search heuristics exclusively use recombination or mutation.[2] Furthermore, the main features of the selection strategy are described, including number and details of selection criteria, as well as the trade-off made between exploration of new search space areas and exploitation of existing solutions. Most modern GP search heuristics make provisions for preserving genetic diversity and related to that, avoiding premature convergence of a population to local optima. These provisions may include fitness sharing, crowding, niching, age layering, or restarts. Diversity preservation approaches implemented are also part of the description of each GP search heuristic. Finally, for each GP search heuristic, parameters including types, ranges, default values, and constraints are given.

The first GP search heuristic selected for this study is the straight forward single-objective tournament selection-based TinyGP search heuristic. Modern GP systems used for real-world applications, such as DataModeler and Eurequa, also implement single- or multi-objective steady state EAs with tournament selection. The second GP search heuristic selected for this study is a new GP search heuristic based on a traditional generational multi-objective EA that implements modern means of controlling bloat and preserving population diversity. Steady state EAs based on tournament selection are traditionally used in GP systems to simplify parallelization on shared memory multiprocessors. Nonetheless, it is also possible to parallelize generational EAs, while the parallelization on large scale distributed memory multiprocessors is of comparable difficulty with both algorithmic schemes.

2.1 Common Components

The components of a GP system responsible for individual initialization and variation (i.e. mutation and recombination) are in principle independent of the search heuristic. The same applies to the concrete individual representation and the means used for individual evaluation. As the main focus of the study lies on comparing the performance of different search heuristics, the setup of these common components are introduced only very briefly.

[2] For example, John R. Koza's original GP system lacked mutation operators and relied exclusively on recombination.

Individual Representation RGP uses a traditional abstract syntax tree representation for GP individuals. For both implementation and execution efficiency, R's internal data structures representing expressions and functions are used. These data structures can be efficiently and simply manipulated by R and C code, facilitating the implementation of variation operators. RGP individuals are R functions that are directly executable by the highly optimized R interpreter. Individuals that represent functions on real numbers are automatically vectorized, making the typical use case of evaluating an individual on several fitness cases very efficient.

Fitness Function The RGP system provides implementations of sample-based as well as of analytical loss functions for symbolic regression. Several genotypic complexity measures are implemented. In this study, sample `rmse` is used for measuring goodness of fit, and visitation length is used for measuring genotypic complexity. Both measures minimization criteria.

Initialization Operators In this study, classical *ramped half-and-half* initialization was used in all experiments. [11] Because the success of GP runs seems to be quite sensitive to starting conditions, RGP implements multiple initialization schemes. For simplicity and because of time constraints, these alternative schemes where not considered in this study.

Variation Operators RGP provides implementations of multiple sets of variation and crossover operators. In this study, a fairly standard operator set inspired by John R. Koza's original work was used to provide a well-understood baseline. [11] This set includes unbalanced crossover as a recombination operator as well as several mutation operators. The latter includes a subtree mutation operator, a constant mutation operator, as well as a function label mutation operator. Detailed documentation and implementation of all variation operators is available in the RGP package.

Common Component Parameters The parameters of the GP system components common to all search heuristics are given in Table 1. In this example study, the indicated default parameters were used for all experiments. Automatic parameter tuning is supported through SPOT, but was not done in this work due to time constraints.

2.2 TinyGP

TinyGP is a popular small GP implementation mainly used in teaching. It implements a simple steady-state single-objective search heuristic with tournament selection that is loosely based on John R. Koza's original work on GP. [11] Steady-state search heuristics with tournament selection are very popular in GP, both for simple teaching systems as well as for complex real-world systems, as they are relatively simple to implement and allow straight-forward parallelization. The TinyGP search heuristic can be seen as a deliberately minimal

Table 1. Parameters of the RGP system independent of the search heuristic used.

	Variable (Symbol)	*Domain*	*Default*
Subtree Mutation Probability	`mutationSubtreesP` (p_{mst})	$[0,1]$	$\frac{1}{3}$
Subtree Mut. Insert/Delete Prob.	`mutationSubtreesPinsertDelete`	$[0,1]$	0.5
Subtree Mut. Subtree Prob.	`mutationSubtreesPsubtree`	$[0,1]$	0.9
Subtree Mut. Constant Prob.	`mutationSubtreesPconstant`	$[0,1]$	0.5
Subtree Mut. Constant Minimum	`mutationSubtreesConstantMin`	$\mathbb{R}$	-1
Subtree Mut. Constant Maximum	`mutationSubtreesConstantMax`	$\mathbb{R}$	1
Subtree Mut. Depth Maximum	`mutationSubtreesDepthMax`	$\mathbb{N}$	2
Function Mutation Probability	`mutationFunctionsP` (p_{fun})	$[0,1]$	$\frac{1}{3}$
Constant Mutation Probability	`mutationConstantsP` (p_{con})	$[0,1]$	$\frac{1}{3}$
Constant Mut. Mean	`mutationConstantsMu`	$\mathbb{R}$	0
Constant Mut. SD	`mutationConstantsSigma`	$\mathbb{R}$	1
Individual Size Limit	`individualSizeLimit`	$\mathbb{N}$	64
Error Measure	`errorMeasure`	$G \to \mathbb{R}$	sample `rmse`

single-objective example for the popular class of steady-state GP search heuristics with tournament selection. For this reason, it was implemented in RGP in included and this study as a baseline.

Algorithm Structure TinyGP employs a simple single-objective steady state EA as its search search heuristic. In the first step of the algorithm, a population pop(0) of μ random individuals is created. Next, the steady-state evolution process starts by randomly selecting either a recombination or a mutation operator. The probability for selecting the recombination operator is given by the parameter $p_{\mathtt{rec}}$. In case of recombination, the algorithm selects two parents via two independent tournaments of size $s_{\mathrm{tournament}}$ as detailed in the next paragraph. Note the non-zero probability of choosing the same individual for both recombination parents, as tournaments are performed independently. In case of mutation, a single parent is chosen in a single tournament. In both cases, a single child is creating by applying the chosen variation operator to the parent(s). Next, the algorithm chooses a individual to replace by this child in a single negative tournament of size $s_{\mathrm{tournament}}$. This process is repeated until a predefined termination criterion is met. Pseudo-code for this search heuristic is shown in Figure 1.1.

Listing 1.1. Pseudo-code implementation of the TinyGP search heuristic. Variables ω denote probability spaces. ω_{RNG} is the probability space used to model the random number generator function `randomUniformNumber`, which generates uniform distributed random numbers in the closed interval between 0 and 1. The recombination operator `rec` must be defined on pairs of individuals.

```
pop ← createIndividuals(number = $\mu$, $\omega_{\mathrm{create}}$)

while (termination criterion not met) {
  child ← if (randomUniformNumber($\omega_{RNG}$ ≤ $p_{\mathtt{rec}}$)) {
    mother ← tournament(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)
    father ← tournament(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)
    rec(mother, father, $\omega_{\mathtt{rec}}$)
  } else {
    parent ← tournament(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)
    mut(parent, $\omega_{\mathtt{mut}}$)
  }

  replaced ← negativeTournament(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)

  pop[replaced] ← child
}

return(pop)
```

Selection Strategy Tournament selection in TinyGP proceeds as follows: First, an individual is selected from the population by uniform random sampling as the current best individual. This individual the fitness is then compared to competitors $s_{\text{tournament}}$ times. Each time a competitor has a better (smaller) fitness value, it takes the place as the current best individual. Competitors are chosen by uniform random sampling from the entire population. Therefore, there is a non-zero probability that the same individual enters the same tournament multiple times.

The negative tournament selection operator employs the same strategy, its only difference being that the order relation $<$ is being replaced by its converse $>$, so that the worst individual taking part in the tournament is returned as result.

Real-world implementations of the tournament selection operator often include extensions and optimizations. For example, individuals participating in a tournament are often sampled beforehand, without replacement, avoiding the inefficiency of duplicated individuals in tournaments.

Figure 1.2 gives a pseudo-code implementations of the tournament and negative tournament selection operators described above and referred to in Figure 1.1.

Listing 1.2. Pseudo-code implementation of tournament selection. Variables ω denote probability spaces.

```
tournament ← function(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)
  bestIndividual ← sampleWithoutReplacement(pop,
                          number = 1, $\omega_{\text{uniform}}$)
  bestFitness ← $\infty$

  for (i in 1:$s_{\text{tournament}}$) {
    competitor ← sampleWithoutReplacement(pop,
                        number = 1, $\omega_{\text{uniform}}$)
    if (fit(competitor) < bestFitness) {
      bestFitness ← fit(competitor)
      bestIndividual ← competitor
    }
  }

  return(bestIndividual)
}

negativeTournament ← function(pop, $s_{\text{tournament}}$, $\omega_{\text{tournament}}$)
  worstIndividual ← sampleWithoutReplacement(pop,
                              number = 1, $\omega_{\text{uniform}}$)
  worstFitness ← $\infty$

  for (i in 1:$s_{\text{tournament}}$) {
    competitor ← sampleWithoutReplacement(pop,
                           number = 1, $\omega_{\text{uniform}}$)
    if (fit(competitor) > worstFitness) {
      worstFitness ← fit(competitor)
      worstIndividual ← competitor
    }
  }

  return(worstIndividual)
}
```

Diversity Preservation The TinyGP system does not implement any internal means of diversity preservation, but can be extended with well-known external measures, such as fitness sharing, crowding, niching, and automatic restarts without much effort. For simplicity, these extensions where not implemented and not included in this study.

Parameters Table 2 gives all parameters of the TinyGP search heuristic. Note the comparatively large default population size, which is typical for classical steady state GP search heuristics.

Table 2. Parameters of the TinyGP search heuristic.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	300
Tournament Size	`tournamentSize` ($s_{\text{tournament}}$)	$\mathbb{N}$	2
Recombination Probability	`recombinationProbability` ($p_{\texttt{rec}}$)	$[0, 1]$	0.9

There are no additional hard parameter constraints in the RGP implementation of the TinyGP search heuristic, although the tournament size $s_{\text{tournament}}$ should be smaller than or equal to the population size μ.

2.3 Generational Multi-Objective GP

Generational Multi-Objective GP (GMOGP) is a generational multi-objective GP search heuristic that combines ideas of state-of-the-art multi-objective GP search heuristics with design concepts of modern generational multi-objective evolutionary algorithms. The main reason for its design and inclusion in the set of search heuristics provided by RGP is to answer the question of whether it is possible to reach performance comparable to complex state-of-the art search heuristics like Ordinal Pareto GP (OPGP) with an conceptually simpler generational multi-objective GP search heuristic.

Algorithm Structure As its name implies, GMOGP is based on a classical generational $(\mu + \lambda)$ strategy. After creating an initial population $\texttt{pop}(0)$ of μ random individuals, the iterative evolution process starts by choosing λ pairs of parents by uniform random sampling without replacement. Pairwise recombination is applied before mutation, yielding λ children. Next, μ individuals are chosen from the $(\mu + \lambda + \nu)$-sized set union of parents, children and ν newly initialized individuals by the Pareto selection operator detailed in the next paragraph, replacing the parent population. This iterative process is stopped when a predefined termination criterion is met. Figure 1.3 outlines the GMOGP search heuristic in pseudo-code.

Listing 1.3. Pseudo-code implementation of the GMOGP search heuristic. Variables ω denote probability spaces. The recombination operator `rec` must be defined on pairs of individuals.

```
pop ← createIndividuals(number = $\mu$, $\omega_{\text{create}}$)

while (termination criterion not met) {
parents ← sampleWithoutReplacement(pop,
                number = $2 \times \lambda$,
                $\omega_{\text{uniform}}$)
mothers ← parents[$1:\lambda$]
fathers ← parents[$(\lambda+1):2 \times \lambda$]
children ← $\texttt{mut}_{\text{pop}}$($\texttt{rec}_{\text{pop}}$(mothers, fathers, $\omega_{\texttt{rec}}$), $\omega_{\texttt{mut}}$)
newIndividuals ← createIndividuals(number = $\nu$, $\omega_{\text{create}}$)

selectionPool ← parents ∪ children ∪ newIndividuals
survivors ← $\texttt{sel}_{\langle GMOGP \rangle}$(selectionPool, number = $\mu$)
pop ← survivors
}

return(pop)
```

Selection Strategy The selection operator $\texttt{sel}\langle GMOGP \rangle$ is based on non-dominated sorting (NDS) of the selection pool based on the three criteria goodness of fit on training data, genotypic complexity, and *genotypic age* (see next paragraph).

Ties in the NDS are broken by crowding distance. Thus, the selection strategy matches the selection strategy of the well-established NSGA-IIEMOA. [4] Alternatively, ties in the NDS may be broken by hypervolume contribution, transforming the selection strategy into that of the SMSEMOA, although, due to time constraints, this option is not examined further in this work. [3]

The selection criteria genotypic complexity and genotypic age can be enabled and disabled individually, yielding three valid configurations of the GMOGP search heuristic:

- single-objective selection based on goodness of fit only (GMOGP-F)
- multi-objective selection based on goodness of fit and individual age (GMOGP-FA)
- multi-objective selection based on goodness of fit and individual complexity (GMOGP-FC)
- multi-objective selection based on goodness of fit, individual age, and genotypic individual complexity (GMOGP-FCA)

Depending on the configuration selected, other algorithm components may be disabled. Explicit diversity preservation through Age-Fitness Pareto Optimization (see next paragraph) is available only when the individual age criterion is enabled.

Diversity Preservation GMOGP implements elements of Schmidt and Lipson's Age-Fitness Pareto Optimization (AFPO) algorithm for preserving genotypic diversity and avoiding premature convergence. [16] In each generation, a fixed number of newly initialized individuals are inserted into the population to maintain genetic diversity. Of course, these new individuals will be of low fitness on average and would be quickly selected for replacement without having the chance to obtain local optima through a series of variation steps. This problem is countered by the introduction of genotypic $\mathtt{age} : \mathcal{G} \to \mathbb{N}$, as defined as follows:

$$\begin{aligned} \mathtt{age}(g_{new}) &:= 0, \\ \mathtt{age}[\mathtt{mut}(g)] &:= \mathtt{age}(g) + 1, \\ \mathtt{age}[\mathtt{rec}(g_A, g_B)] &:= \max[\mathtt{age}(g_A), \mathtt{age}(g_B)], \end{aligned} \tag{1}$$

where g_{new} is a new genotype just inserted into the selection pool, and g, g_A, and g_B are individuals already existing in a population. The genotypic age of a new individual is defined as 0, mutation of an existing individual increases its age by one, and the age of the recombination of two parents is the maximum of their ages. Therefore, the age of an individual is the age of its oldest ancestor, even if no genetic material of that ancestor is left. Genotypic age is then considered as another optimization criterion to minimize, in addition to the minimization criteria fitness and genotypic complexity. This leads to an emergent dynamic age-layering of the population, as young individuals are not dominated by older more fit or less complex genotypes. Therefore, younger individuals are allowed to evolve independently of older individuals, until they reach the same age, i.e. have been subject to the same number of variation steps. Ideally, this approach should preserve genetic diversity and enable the discovery of new local and global optima throughout the entire duration of a GP run. The diversity preservation mechanism can be disabled by setting the boolean search heuristic parameter *Age Layering* to `false`.

Parameters Table 3 presents all parameters of the GMOGP search heuristic.

Table 3. Parameters of the GMOGP search heuristic.

	Variable (Symbol)	*Domain*	*Default*
Population Size	`mu` (μ)	$\mathbb{N}$	300
Children per Generation	`lambda` (λ)	$\mathbb{N}$	20
New Individuals per Generation	`nu` (ν)	$\mathbb{N}_0$	1
Enable Complexity Criterion	`complexityCriterion`	$\mathbb{B}$	`true`
Enable Age Criterion	`ageCriterion`	$\mathbb{B}$	`true`
Recombination Probability	`recombinationProbability` ($p_{\texttt{rec}}$)	$[0, 1]$	0.1

In the RGP implementation of this search heuristic, these parameters are subject to the following constraint:

$$\lambda \leq \left\lfloor \frac{\mu}{2} \right\rfloor \qquad \text{(Children Set Size)}$$

As parents are sampled without replacement from an uniform distribution and two parents are needed for recombination, the number of children per generation must be smaller than or equal to half the population size.

3 Scalable Test Functions

Test problems of controllable difficulty, i.e. *scalable test functions*, are a flexible tool for assessing the relative performance benefits of different GP system components under varying conditions. This study focuses on symbolic regression, today perhaps the GP application of highest practical importance. Unfortunately, defining finely scalable test functions for symbolic regression holds many challenging surprises. Conventional measures of problem difficulty, such as information criteria [6], that are good predictors for the performance of fixed-structure modelling approaches like modern statistical regression techniques, often fail to predict the performance of symbolic regression runs.[3] Symbolic regression on test functions of comparatively simple structure can prove extremely difficult for state-of-the-art symbolic regression systems. [8]

The framework described in this chapter implements three scalable test problem classes for symbolic regression:

– Taylor polynomial approximations of analytical functions

[3] See Section 4 for definitions of performance measures for symbolic regression.

- Time series compositions
- Test functions with spurious variables

Taylor polynomial approximations of arbitrary analytical functions can be can be symbolically created by the framework.[4] Consider, as an example, Taylor polynomial approximations of increasing degree of the sine function at point 0. Approximations of higher degree should be more difficult to fit by a GP system with a standard set of arithmetic building blocks, as the required (genotypic) expressions quickly grow in size. The polynomial degree therefore determines problem difficulty in a comprehensible manner. Unfortunately, problem difficulty in this test function class is only controllable in coarse steps, as the discovery of polynomials of higher degrees quickly becomes intractable even for state-of-the-art symbolic regression. Nonetheless, this test function class can be of use for symbolic regression techniques that support the definition of background information on the solution structure.

Time series compositions provide a test function class inspired by classical time series analysis. Starting from simple linear or quadratic trend components, complications such as sinusoidal periodic components and noise can be added to gradually increase test function difficulty. A benefit of this test function class is that it provides realistic test cases that are solvable by comparatively compact symbolic expressions and therefore discoverable by modern symbolic regression techniques. Difficulty control is still quite coarse-grained, as it depends on adding periodic components and noise.

The third class of scalable test problems is based on the simple yet very effective idea of adding *spurious variables* to the set of fitness cases to conceal the true functional dependency between driving variables and output for arbitrary test functions. [9] To increase the difficulty of a given function, additional function arguments (spurious variables) are added that do not influence the function value. As this fact is hidden from the symbolic regression system, and spurious variables might be correlated with the function value by chance, problem difficulty can be gradually increased in fine-grained steps by increasing the number of spurious variables.

3.1 Spurious Variable Test Functions

In the experiments conducted for this study, three scalable test functions described below where used. For each test function, 1 to 10 spurious variables where added to gradually increase problem difficulty. Fitness cases where created by uniform random sampling in the indicated training and test intervals. Both training and tests set consisted of the number of fitness cases indicated below.

[4] This is realized by the YACAS computer algebra system embeddable into the R environment.[13]

P1 *(Simple Sine)* Discovering the input-output relation of the simple sine function should be trivial even for very simple symbolic regression system, if no spurious variables are introduced.

$$f_{\mathrm{P1}}(x_1) := \sin(\pi x_1) \tag{2}$$

The training interval of the Simple Sine test function is fixed to $[-\pi, \pi]$, the test interval is fixed to $[-\frac{3}{2}\pi, \frac{3}{2}\pi]$, and the number of fitness cases is fixed to $N_{\mathrm{P1}} := 32$. The GP function set used with this test function is $\{+, -, *, /, \sin\}$.

P2 *(Newton Problem)* Compared to the Simple Sine, the Newton Problem poses a slightly harder test case as the minimal true expression has slightly larger genotypic size. It was mainly included in this study as a more practical example based on a well known natural law.

$$f_{\mathrm{P2}}(x_1, x_2, x_3) := \frac{x_1 x_2}{x_3^2} \tag{3}$$

The training interval of the Newton Problem test function is $[0, 1]$, the test interval $[0, 2]$, and the number of fitness cases is fixed to $N_{\mathrm{P2}} := 64$. The GP function set used with this test function is $\{+, -, *, /\}$.

P3 *(Sine Cosine)* This test function is a simplified variant of test problem *(P12)* given in Michael F.Korn's work on accuracy in symbolic regression. [8] It is considered difficult because it constraints constants as arguments to non-linear functions.

$$f_{\mathrm{P3}}(x_1, x_2) := 6\sin(x_1 - 3)\cos(x_2 - 3) \tag{4}$$

The training interval of the Sine Cosine test function is $[-\pi, \pi]$, the test interval $[-\frac{3}{2}\pi, \frac{3}{2}\pi]$, and the number of fitness cases is fixed to $N_{\mathrm{P3}} := 128$. The GP function set used is $\{+, -, *, /, \sin, \cos\}$.

4 Experimental Setup

Following the experimental framework presented in [1], a sound experimental setup requires the specification of the
1. optimization problem
2. performance measure
3. initialization method
4. termination method

4.1 Optimization Problems in GP

Optimization problems in GP have two components: (a) a test function, say f, and (b) a measure, which determines the distance between the true function f and its GP approximation, say $\hat{f}$. Test functions are presented in Sect. 3. The measure, which will be used in our comparisons can be describes as follows. Consider an input value, say x_i, a prediction model $\hat{f}(x)$, and the true value y_i. The prediction model $\hat{f}$ is estimated by GP from a training sample. The *loss function*, which measures the errors between the y_i's and the $\hat{f}(x_i)$'s, is defined as

$$L(y, \hat{f}(x)) = (y - \hat{f}(x))^2. \tag{5}$$

The *training error* is the square root of the average loss over the training sample

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i))}, \tag{6}$$

where N denotes the number of fitness cases as introduced in Sect. 3.1. This value depends on the test function, e.g., $N = 32$ was chosen for f_{P1}.

1. During the pre-experimental planning phase, a large number of function evaluations n_0, the so-called *budget*, is used the determine an adequate percentage of GP runs that reach a pre-defined (small) distance to the optimum.
2. *Run length distributions* (RLDs), as introduced by [12], are generated to estimate a suitable budget $n \leq n_0$ for the actual experiments.

To avoid floor and ceiling effects, we determine adequate problem design, e.g., number of function evaluations as well as reasonably chosen training, validation, and test sets.

4.2 Performance Measures

To avoid overfitting the *test error*

$$E[L(y_t, \hat{f}(x_t)], \tag{7}$$

i.e., the expected prediction error over an independent test sample x_t, is preferred. The comparison of the algorithm designs is based on these test-set data.[5] To enable fair comparisons and to avoid floor and ceiling effects, RLDs of the TinyGP implementation are generated.

4.3 Initialization and Termination Methods

Classical ramped half-and-half initialization was used in all experiments. To determine suitable termination methods, RLDs were generated. The number of fitness case evaluations, i.e., the number of fitness cases times the number of test function evaluations was chosen as the termination criterion.

[5] In addition to our considerations, we will sketch out further enhancements of our testing framework: Typically, models will have tuning parameters, say α, so we can describe predictions as $\hat{f}_\alpha(x)$. In forthcoming steps of our experimental analysis, we are interested in finding good α values while avoiding overfitting, i.e., two different problems are to be solved: First, estimating the performance of different models $\hat{f}_\alpha$ in order to chose the best model, and second, estimating its prediction error on new data. If sufficient data is available, the data set can be partitioned into a *training set* x_{train}, a *validation set* x_{val}, and a *test set* x_{test}.

4.4 Research Questions

TinyGP is considered as a baseline algorithm, which implements essential GP features. A common belief in the GP community can be stated as follows:

Scientific Claim 1 *Complex problems require complex algorithms.*

Or, stated differently: TinyGP can solve tiny problems whereas hard problems require more complex algorithms.

From a naive point of view, Claim 1 goes without saying—one may simply consider that the opposite assumption is true. Therefore, it can be taken as a guideline for performing experiments and present results that are based on solid scientific and statistical assumptions. We will proceed as follows: In order to perform a sound statistical analysis, we will define a hierarchy of complex (hard) problems. Scalable test functions as introduced in Sec. 3.1 are well suited for our goals: increasing the number of spurious variables should decrease the success rate of the GP system.

Then we will define a reference GP implementation that comprehends the essential features of GP systems in a very simple and understandable manner. The TinyGP system is considered as an ideal candidate, because it is well-known, easy to obtain, and easy to implement.

4.5 Pre-Experimental Planning

During the first stage of our experiments, no parameter tuning will be performed. Our main goal is to discover (positive) correlations between algorithm complexity and problem difficulty using default algorithm parameter settings.

First experiments were set up to calibrate the reference algorithm, i.e., TinyGP, to the problem.

Statistical Hypothesis 1 (H-1) *TinyGP requires more function evaluations to reach the same success rate if the problem complexity increases.*

RLDs are suitable means to measure performance and to determine an adequate budget. This is necessary, because floor and ceiling effects can be avoided. A large number of function evaluations, i.e. a budget of $n_0 = 1e6$, was chosen to generate the RLDs. The set of test functions consists of $\{f_{\mathrm{P1}}, f_{\mathrm{P2}}, f_{\mathrm{P3}}\}$. Program code and results from these runs are available on-line and can be used to check if the experimental setup for new studies is correct.

The investigation of H-1 has two significant results: First, we ensure that the reference algorithm is able to solve this problem (at least it should be able to improve an existing candidate solution). Second, the correct number of test function evaluations for comparisons is determined. We detect test functions which are far too easy (or too hard) for the reference algorithm. The case, that the reference algorithm is unable to solve the test function needs further considerations.

Next, we introduce the base-line variant of the new algorithm, i.e., GMOGP-F, which should be analyzed.

Statistical Hypothesis 2 (H-2) GMOGP-F *is competitive with the reference algorithm.*

To analyze H-2 the same setup as for H-1 was used.

These two series of experiments belong to the pre-experimental planning phase, because they are needed to set up fair comparisons. If one hypothesis has to be rejected, the experimental set up should be reconsidered, e.g., the set of test function should be modified or the computational budget should be increased.

4.6 Experiments

The third series of experiments is performed to analyze the influence of new GP components on the algorithm's performance. Here, we will analyze new selection schemes, namely aging, complexity, and a combination of these two.

Statistical Hypothesis 3 (H-3) *Introducing multi-objective selection based on goodness of fit and individual age improves the* GMOGP-F *performance.*

To investigate H-3, the following setup was used. The set of test functions consists of $\{f_{\mathrm{P1}}, f_{\mathrm{P2}}\}$, and a budget of $n_0 = 250,000$ was used. The age criterion as described in Table 3 was set to `true`. With respect to the experimental setup for hypothesis H-2, the remaining parameters remained unmodified.

Statistical Hypothesis 4 (H-4) *Introducing multi-objective selection based on goodness of fit and individual complexity improves the* GMOGP-F *performance.*

To investigate H-4, a similar setup as for hypothesis H-3 was used. The complexity criterion as described in Table 3 was set to `true`. With respect to the experimental setup for hypothesis H-2, the remaining parameters remained unmodified.

Statistical Hypothesis 5 (H-5) *Introducing multi-objective selection based on goodness of fit, individual age and individual complexity improves the* GMOGP-F *performance.*

To investigate H-5, a similar setup as for hypothesis H-3 was used. The complexity as well as the age criterion (Table 3) was set to `true`. With respect to the experimental setup for hypothesis H-2, the remaining parameters remained unmodified.

5 Results

To generate the experimental data needed to test the statistical hypotheses established in Section 4, experiments for each hypothesis where declaretively formulated in the test framework. These experiments where then automatically executed on a 48 core PBS-compatible compute cluster. Generation of result reports and visualizations is also automated by the framework.

Statistical Hypothesis H-1 Table 4 and Figure 2(a) show that TinyGP's success rates decrease as problem complexities increase. As the algorithm can get stuck in local optima and basing RLD calculation on test data instead of training data adds additional noise, this decrease is not strictly monotonic. Nonetheless, data do not indicate that H-1 has to be rejected. During pre-experimental planning, the Sine Cosine test problem proved too difficult for the reference algorithm (TinyGP), and was therefore excluded from the main experiments.

Statistical Hypothesis H-2 On both test functions studied, GMOGP-F shows slightly better performance than the reference, as visible in Table 4. Hypothesis H-2 does not have to be rejected. This concludes the analysis of the pre-experimental planning phase.

Statistical Hypothesis H-3 As visible in Table 4, as well as in Figure 5, introducing age as a secondary optimization criterion (GMOGP-FA) significantly increases algorithm performance on the Newton Problem test function. Hypothesis H-3 holds.

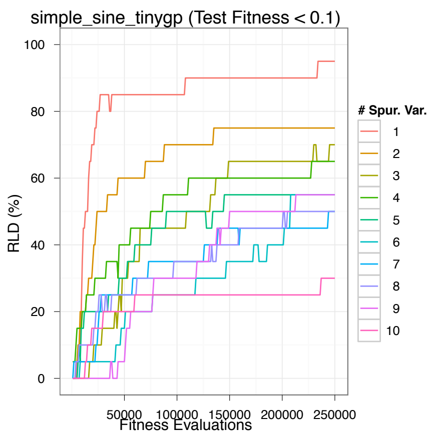
Statistical Hypothesis H-4 In contrast, introducing complexity as a secondary optimization criterion (GMOGP-FC) degrades algorithm performance in this experimental framework, as visible in Table 4.

Statistical Hypothesis H-5 Introducing both complexity and age as additional optimization criteria (GMOGP-FCA) also degrades algorithm performance in comparison with the single-objective case (GMOGP-F), but not so much as GMOGP-FC.

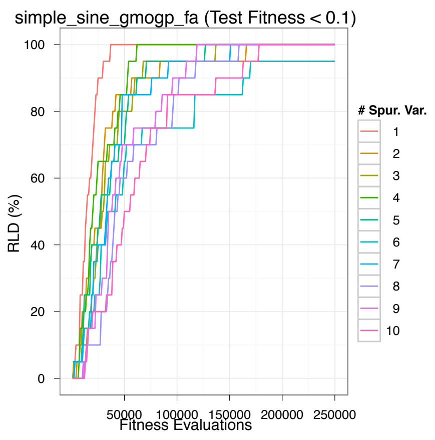
Table 4. Run length distributions at $250,000$ fitness function evaluations.

		Difficulty (# Spurious Variables)									
Search Heuristic	*Test Function*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
TinyGP	Simple Sine	95	75	70	65	55	55	50	50	55	30
	Newton Problem	80	80	60	70	60	35	40	45	35	10
	Sine Cosine	10	20	20	15	10	0	5	15	5	5
GMOGP-F	Simple Sine	100	100	100	100	100	100	100	95	100	95
	Newton Problem	90	90	90	80	50	75	65	55	40	30
GMOGP-FA	Simple Sine	100	100	100	100	100	95	100	100	100	100
	Newton Problem	100	100	85	85	80	95	80	75	65	55
GMOGP-FC	Simple Sine	100	85	100	80	65	85	75	90	75	70
	Newton Problem	65	60	30	30	15	10	20	15	0	5
GMOGP-FCA	Simple Sine	100	100	100	100	100	95	95	95	100	90
	Newton Problem	95	80	75	50	30	50	60	55	35	35

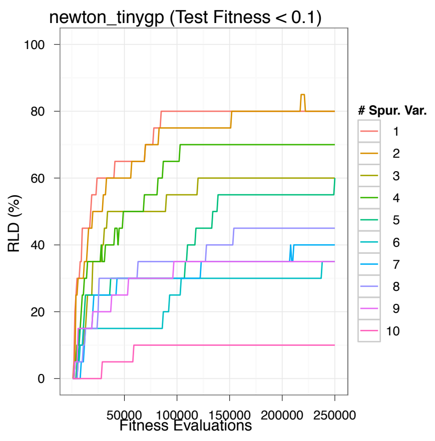
Fig. 1. Run length distribution plots: The number of fitness function evaluations are shown on the x-axis, algorithm success rates (test fitness ¡ 0.1) are shown on the y-axis. Each plot shows 10 difficulty levels (number of spurious variables) of a single test function for a single search heuristic. For each combiniation of search heuristic, test function, and difficulty level, 20 independent GP runs were performed.



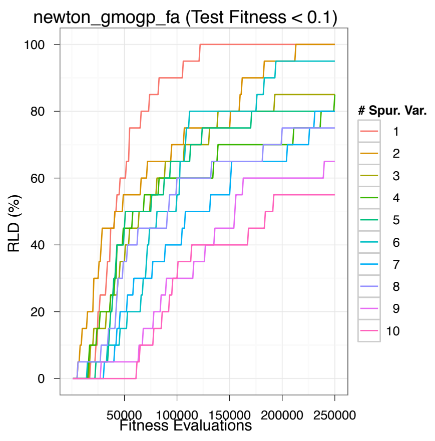
(a) Simple Sine test function, TinyGP search heuristic



(b) Simple Sine test function, GMOGP-FA search heuristic



(c) Newton Problem test function, TinyGP search heuristic



(d) Newton Problem test function, GMOGP-FA search heuristic

6 Discussion and Conclusions

In relation to hypotheses H-1 and H-2, it became clear that the scalable test function set based on the introduction of spurious variables works as expected on all studied search heuristics. TinyGP could be established as a practical reference algorithm. Therefore, a conceptual framework for statistically well-founded comparisons the relative performance benefits of GP system components is available. An R prototype realizing this framework in software is actively developed.

Within this framework, experiments show that GMOGP-F is competitive with TinyGP, i.e. that a simple generational search heuristic is a viable replacement for steady-state heuristics more common in GP. Introducing an age criterion (GMOGP-FA) improves GP performance significantly. On the other hand, the introduction of a complexity criterion (GMOGP-FC and GMOGP-FCA) does not result in the expected performance gain, contrary to results from literature. This indicates an implementation or parameterization problem, which demands further investigation. As all experiments where conducted with default parameters, parameter optimization via SPO should help to provide optimal parameter settings. This matter shows the value of the test framework as an automated tool for discovering possible implementation and parameterization problems.

7 Outlook

Only the most basic features of the test framework were demonstrated in this chapter, many extensions are available or under active development. These include additional classes of scalable test functions, additional performance measures, additional result visualizations, tools for testing the statistical significance of performance differences in GP components, and tools for parameter optimization.

In further work, the applicability of the additional scalable test function classes will be experimentally analyzed. A study on automatic parameter tuning for TinyGP, GMOGP and DataModeler based on SPOT will be conducted to obtain statistically reliable results on the sensitivity of different GP systems to their parameter settings. This statistical analysis requires enhanced experimental designs, e.g., nested and split-plot designs, which are subject of our current research.

In summary, the hypotheses-driven approach encouraged by the framework introduced in this chapter, should lead to statistically validated results of high reproducibility. In the future, this framework will be applied to study the performance characteristics of real-world GP systems based on a larger set of realistic scalable test problem classes.

Acknowledgements This work was supported the Bundesministerium für Bildung und Forschung (BMBF) under the grant FIWA (AiF FKZ 17N2309). Many thanks to Dr. Boris Naujoks, Tobias Brandt, and Jörg Stork for valuable ideas and suggestions.

References

1. T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation—The New Experimentalism.* Natural Computing Series. Springer, Berlin, Heidelberg, New York, 2006.
2. T. Bartz-Beielstein. SPOT: An R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CIOP Technical Report 05/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, June 2010. Comments: Related software can be downloaded from http://cran.r-project.org/web/packages/SPOT/index.html.
3. N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.
4. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Proceedings of the Parallel Problem Solving from Nature VI*, pages 849–858, Berlin, Heidelberg, New York, 2000. Springer.
5. O. Flasch, O. Mersmann, and T. Bartz-Beielstein. RGP: An open source genetic programming system for the R environment. In M. Pelikan and J. Branke, editors, *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon*, pages 2071–2072. ACM, 2010.
6. I. A. Kiesepp. Akaike information criterion, curve-fitting and the philosophical problem of simplicity. *British Journal for the Philosophy of Science*, 48:21–48, 1997.
7. P. Koch, B. Bischl, O. Flasch, T. B. Beielstein, and W. Konen. On the tuning and evolution of support vector kernels. CIOP Technical Report 04/11, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, March 2011.
8. M. F. Korns. Accuracy in symbolic regression. In R. Riolo, E. Vladislavleva, and J. H. Moore, editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, pages 129–151. Springer New York, 2011.
9. M. Kotanchek, G. Smits, and E. Vladislavleva. Pursuing the pareto paradigm tournaments, algorithm variations & ordinal optimization. In R. L. Riolo, T. Soule, and B. Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, chapter 12, pages 167–186. Springer, Ann Arbor, 2006.
10. M. Kotanchek, G. Smits, and E. Vladislavleva. Trustable symoblic regression models. In R. L. Riolo, T. Soule, and e. Worzel, Bill, editors, *Genetic Programming Theory and Practice V*, pages 203–222, 2007.
11. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.
12. A. J. Parkes and J. P. Walser. Tuning local search for satisfiability testing. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI'96)*, pages 356–362, 1996.
13. A. Z. Pinkus and S. Winitzki. Yacas: A do-it-yourself symbolic algebra environment. In *Proceedings of the Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation*, AISC '02/Calculemus '02, pages 332–336. Springer, 2002.

14. R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0.
15. M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009.
16. M. D. Schmidt and H. Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, pages 543–544, New York, NY, USA, 2010. ACM.
17. G. Smits and E. Vladislavleva. Ordinal pareto genetic programming. In G. G. Yen et al., editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 3114–3120, Vancouver, BC, Canada, 16-21 July 2006. IEEE Press.
18. E. Vladislavleva. *Model–based Problem Solving through Symbolic Regression via Pareto Genetic Programming*. PhD thesis, Tilburg University, 2008.
19. D. R. White. Software review: the ecj toolkit. *Genetic Programming and Evolvable Machines*, 13(1):65–67, 2012.