PSB2: The Second Program Synthesis Benchmark Suite

Tom Helmuth and Peter Kelly

Hamilton College

General Program Synthesis

- Problems that resemble those that humans solve
  - multiple general-purpose data types
  - control flow
- Supervised learning: specifications given as input/output examples
- GP and non-GP methods

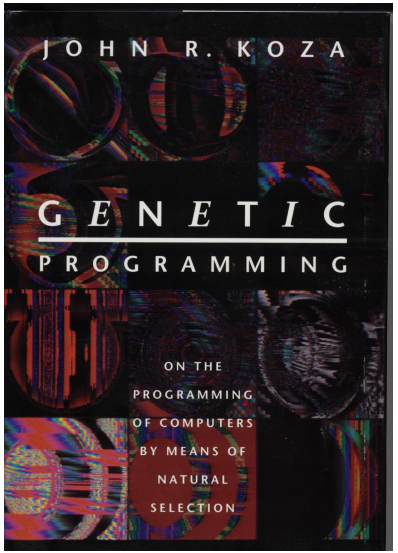
```
1  def spin_words(input1):
2      words = input1.split()
3      result = ""
4      for word in words:
5          if len(word) >= 5:
6              result += word[::-1] + " "
7          else:
8              result += word + " "
9      return result[:-1]
```

Example Python program of the type we want to generate

General Program Synthesis Benchmark Problems: A Brief History

- **Before 2015:**
  - Limited instruction sets
  - Domain specific
  - Toy problems

  } both in GP and inductive program synthesis
  - Ex: Santa Fe artificial ant problem



General Program Synthesis Benchmark Problems: A Brief History

- Before 2015:
  - Limited instruction sets
  - Domain specific
  - Toy problems

  } both in GP and inductive program synthesis
  - Ex: Santa Fe artificial ant problem
- **2015: PSB1 (Helmuth and Spector)**
  - 29 benchmark problems
  - introductory programming homework problem sets
  - has been used to benchmark 10+ GP and non-GP synthesis systems

General Program Synthesis Benchmark Suite

Thomas Helmuth
Computer Science
University of Massachusetts
Amherst, MA 01003
thelmuth@cs.umass.edu

Lee Spector
Cognitive Science
Hampshire College
Amherst, MA 01002
lspector@hampshire.edu

ABSTRACT

Recent interest in the development and use of non-trivial benchmark problems for genetic programming research has highlighted the scarcity of general program synthesis (also called "traditional programming") benchmark problems. We present a suite of 29 general program synthesis benchmark problems systematically selected from sources of introductory computer science programming problems. This suite is suitable for experiments with any program synthesis system driven by input/output examples. We present results from illustrative experiments using our reference implementation of the problems in the PushGP genetic programming system. The results show that the problems in the suite vary in difficulty and can be useful for assessing the capabilities of a program synthesis system.

Categories and Subject Descriptors

I.2.2 [Artificial Intelligence]: Automatic Programming—*Program synthesis*

Keywords

program synthesis; genetic programming; benchmarks

1. INTRODUCTION

Several genetic programming (GP) researchers have highlighted the need for better benchmark problems to guide research in the field [9, 17, 18]. While benchmarks have been proposed, few are for general programming problems (also called "traditional" or "algorithmic" programming problems) even though this category received the second highest level of interest in a recent community survey about the need for benchmarks [17].

Automating human programming has long been a goal of GP, as articulated for example in Koza's first book [7]. The purpose of a general program synthesis benchmark is to help

programming techniques including the use of control flow, modularity, and loops, diverse instruction sets covering multiple data types and data structures. Also, minimal sizes for solution programs should cover a range beyond what could be found using brute-force search. This contrasts with most existing benchmark problems used in GP and other program synthesis fields [6], which prescribe small, domain-specific instruction sets and assess a system's abilities only on a narrow range of programming techniques.

In this paper we present a suite of 29 general program synthesis benchmark problems, systematically selected from sources of introductory computer science programming problems. We present each problem's specifications in the form of input/output examples, making them suitable to a wide range of program synthesis techniques, including GP. While the problems are not particularly challenging for skilled human programmers, they are reasonably challenging for beginners and many are arguably too difficult for existing program synthesis systems, including GP. As textbook problems, they are not likely representative of real general program synthesis applications, yet they should prove useful for assessing progress toward this goal.

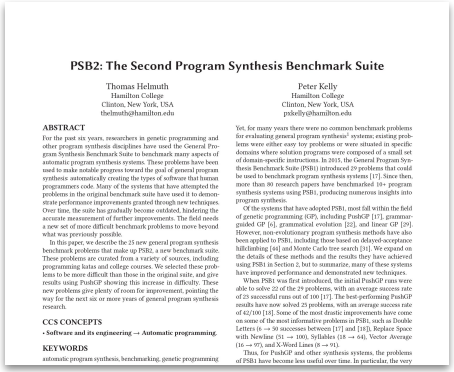
2. BENCHMARK-BASED COMPARISONS

We designate the solution of a general program synthesis problem as a program that perfectly maps inputs to the correct outputs. While one might argue that human-written software is often useful even if it has known bugs, the goal here is to pass all input/output tests. Therefore, we are not interested in programs that are only approximately correct, as might be appropriate in the context of other problems for which GP is used, such as symbolic regression. We recommend measuring performance on the problems presented here primarily in terms of success rates, quantifying how often a stochastic algorithm finds a successful program across a set of runs[1]. A more thorough argument for assessment in terms of success rates can be found in [3]. Furthermore,

General Program Synthesis Benchmark Problems: A Brief History

- Before 2015:
  - Limited instruction sets
  - Domain specific
  - Toy problems

  } both in GP and inductive program synthesis
  - Ex: Santa Fe artificial ant problem
- 2015: PSB1 (Helmuth and Spector)
  - 29 benchmark problems
  - introductory programming homework problems
  - has been used to benchmark 10+ GP and non-GP synthesis systems
- **2021: PSB2 - this talk!**



PSB2: The Second Program Synthesis Benchmark Suite

Thomas Helmuth
Hamilton College
Clinton, New York, USA
thelmuth@hamilton.edu

Peter Kelly
Hamilton College
Clinton, New York, USA
pskelly@hamilton.edu

ABSTRACT

For the past six years, researchers in genetic programming and other program synthesis disciplines have used the General Program Synthesis Benchmark Suite to benchmark many aspects of automatic program synthesis systems. These problems have been used to make notable progress toward the goal of general program synthesis: automatically creating the types of software that human programmers code. Many of the systems that have attempted the problems in the original benchmark suite have used it to demonstrate performance improvements granted through new techniques. Over time, the suite has gradually become outdated, hindering the accurate measurement of further improvements. The field needs a new set of more difficult benchmark problems to move beyond what was previously possible.

In this paper, we describe the 25 new general program synthesis benchmark problems that make up PSB2, a new benchmark suite. These problems are curated from a variety of sources, including programming katas and college courses. We selected these problems to be more difficult than those in the original suite, and give results using PushGP showing this increase in difficulty. These new problems give plenty of room for improvement, pointing the way for the next six or more years of general program synthesis research.

CCS CONCEPTS

• Software and its engineering → Automatic programming.

KEYWORDS

automatic program synthesis, benchmarking, genetic programming

Yet, for many years there were no common benchmark problems for evaluating general program synthesis* systems: existing problems were either easy toy problems or were situated in specific domains where solution programs were composed of a small set of domain-specific instructions. In 2015, the General Program Synthesis Benchmark Suite (PSB1) introduced 29 problems that could be used to benchmark program synthesis systems [17]. Since then, more than 80 research papers have benchmarked 10+ program synthesis systems using PSB1, producing numerous insights into program synthesis.

Of the systems that have adopted PSB1, most fall within the field of genetic programming (GP), including PushGP [17], grammar-guided GP [6], grammatical evolution [22], and linear GP [29]. However, non-evolutionary program synthesis methods have also been applied to PSB1, including those based on delayed-acceptance hillclimbing [16] and Monte Carlo tree search [31]. We expand on the details of these methods and the results they have achieved using PSB1 in Section 2, but to summarize, many of these systems have improved performance and demonstrated new techniques.

When PSB1 was first introduced, the initial PushGP runs were able to solve 22 of the 29 problems, with an average success rate of 25 successful runs out of 100 [17]. The best-performing PushGP results have now solved 25 problems, with an average success rate of 62/100 [18]. Some of the most drastic improvements have come on some of the most informative problems in PSB1, such as Double Letters (6 → 50 successes between [17] and [18]), Replace Space with Newline (51 → 100), Syllables (18 → 64), Vector Average (16 → 97), and X-Word Lines (8 → 91).

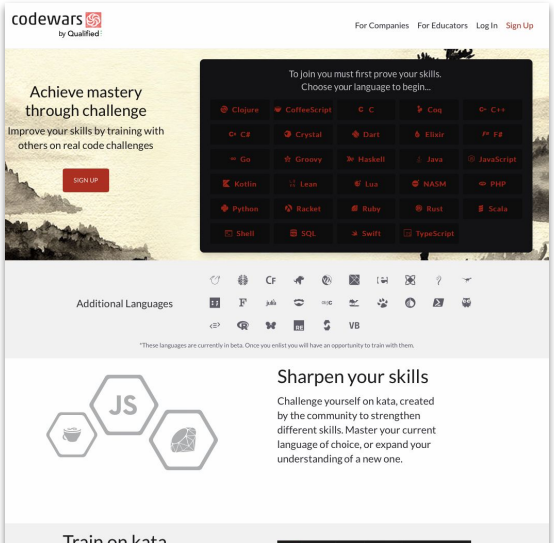
Thus, for PushGP and other synthesis systems, the problems of PSB1 have become less useful over time. In particular, the very

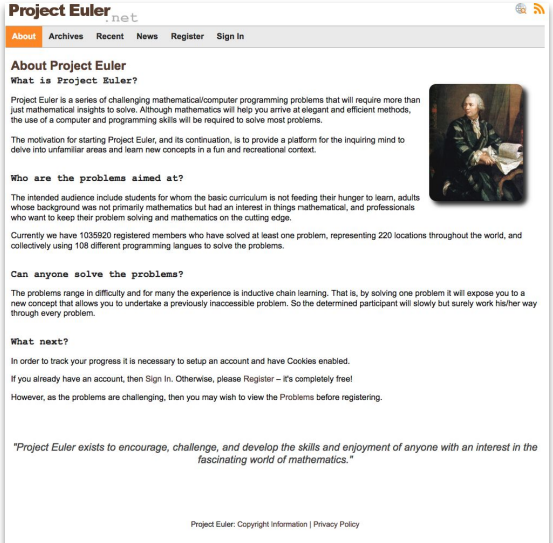
Need for New Benchmark Problems

- Synthesis systems improved on PSB1
- Need new, harder problems to drive research forward
  - **subjective** - problems we expected to be more difficult
  - **objective** - no problems that our GP system solved more than 65% of the time
    - most selected problems have success rates < 10%
- Don't overfit on one set of problems

Problem Sources for PSB2

- **Code Wars** - user-created *coding kata*
- **Advent of Code** - advent calendar of coding problems
- **Project Euler** - archive of mathematical computer programming problems
- **Homework Problems** - from undergraduate classes





Problems in PSB2

25 benchmark problems

- Considered 75+ problems
- Implemented 50+ problems

Basement	Leaders
Bouncing Balls	Luhn
Bowling	Mastermind
Camel Case	Middle Character
Coin Sums	Paired Digits
Cut Vector	Shopping List
Dice Game	Snow Day
Find Pair	Solve Boolean
Fizz Buzz	Spin Words
Fuel Cost	Square Digits
GCD	Substitution Cipher
Indices of Substring	Twitter
Vector Distance	

Problem names

Input/Output types across all problems

Type	Input Count	Output Count
Integer	7	9
Float	2	5
String	10	7
Boolean	0	1
Vector of Integers	6	3
Vector of Floats	2	0

Example Problem: Spin Words

- Given a string of one or more words (separated by spaces), reverse all of the words that are five or more letters long and return the resulting string

- Input: String
- Output: String

- Example:
  `"Hello all at GECCO"` → `"olleH all at OCCEG"`

Example Problem: Vector Distance

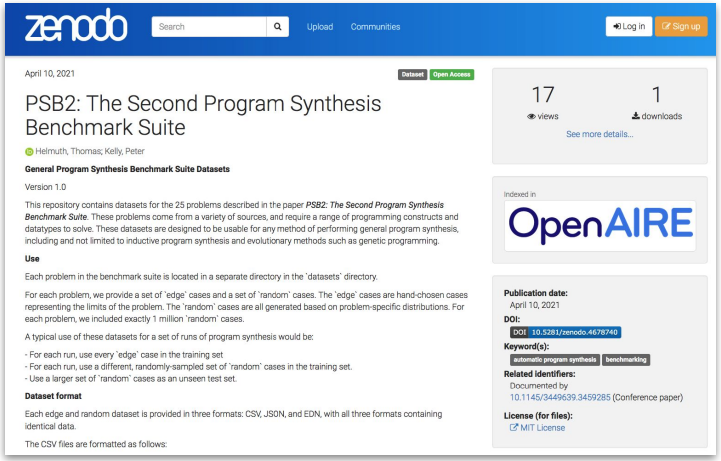
- Given two n-dimensional vectors of floats, return the Euclidean distance between the two vectors in n-dimensional space

- Input: Two vectors of floats in n-dimensions
- Output: Float

- Example:
  `[-77.6 -45.2 58.3] [6.5 48.6 17.5]` → `132.4`

Using PSB2

- PSB2 emphasizes ease of use for wider adoption
- Datasets
  - sample training and test data for each run
- Python and Clojure libraries for easy sampling
  - Easy install right now: **`$ pip install psb2`**



Datasets and documentation link

https://zenodo.org/record/4678739

Experimental Methods

- PSB2 works with any program synthesis system
  - Experiments: PushGP
- Standardized error functions per output data type
  - integers outputs: absolute difference
  - string outputs: Levenshtein string edit distance
  - etc.
- Performance metric:
  - success rate
  - on unseen test set

```
(in1 in2 exec_do*vector_integer (output_integer1
integer_sub in1 output_integer2
vector_integer_occurrencesof exec_do*times
exec_stackdepth integer_mod exec_yankdup
vector_integer_yank in2))
```

Push solution to Find Pair

Results - Successes out of 100 runs

- 13 out of 25 problems solved at least once
- *Comparison with PSB1:*
  - 26 out of 29 problems solved

Problem	Successes
Basement	1
Bouncing Balls	0
Bowling	0
Camel Case	1
Coin Sums	2
Cut Vector	0
Dice Game	0
Find Pair	4
Fizz Buzz	25
Fuel Cost	50
GCD	8
Indices of Substring	0
Leaders	0

Problem	Successes
Luhn	0
Mastermind	0
Middle Character	57
Paired Digits	8
Shopping List	0
Snow Day	4
Solve Boolean	5
Spin Words	0
Square Digits	0
Substitution Cipher	61
Twitter	31
Vector Distance	0

Size of Smallest Simplified Solution Program

- After automatic simplification
- *Comparison with PSB1:*
  - 8 problems solved with < 9 instructions

Problem	Size
Basement	18
Camel Case	20
Coin Sums	33
Find Pair	16
Fizz Buzz	19
Fuel Cost	9
GCD	19
Middle Character	10
Paired Digits	15
Snow Day	11
Solve Boolean	18
Substitution Cipher	9
Twitter	22

Conclusions

- PSB2 - new suite of program synthesis benchmark problems
- Easier to use than PSB1
- Results with PushGP:
  - solutions to 13/25 problems
  - harder than PSB1

Get the datasets and sampling libraries here!

https://zenodo.org/record/4678739