Automated Software Performance Improvement with Magpie

Aymeric Blot

Université de Rennes, France

April 16, 2024 – GI@ICSE 2024, Lisbon

Université de Rennes ICSE 24

http://www0.cs.ucl.ac.uk/staff/a.blot/files/blot_gi@icse_2024_slides.pdf

Outline

Introduction

- Who am I?
- What is GI?
- What is Magpie?

GI as Seen by Magpie

The Magpie Framework

Magpie in Practice

Future Directions and Challenges

Personal Background

Aymeric Blot

- Lecturer @ Université de Rennes
- Research Fellow @ IRISA
- Member @ DiverSE (IRISA/Inria)
- Developing Magpie
- Contributed to PyGGI 2.0
- Developed MO-ParamILS

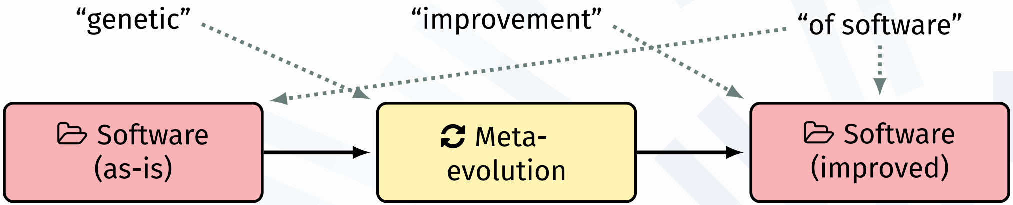
Université de Rennes

UMR IRISA Inria

DiverSE Diversity-Centric Software Engineering

http://www0.cs.ucl.ac.uk/staff/a.blot/

What is... Genetic Improvement of Software?



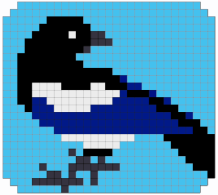
In a nutshell

- **Functional properties:** e.g., automated program repair
- **Non-functional properties:** performance improvement
  → e.g., execution time, energy/memory usage, solution quality, ...

http://geneticimprovementofsoftware.com/

What is... Magpie?

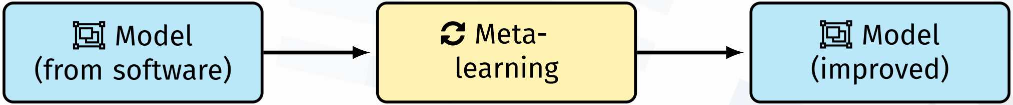
Machine Automated General Performance Improvement via Evolution (of softw...)



Magpie

- Modular Python framework for GI/MT/AC/CO
- Hack-friendly, for researchers!
- User-friendly, for developers!

Magpie is a framework that automates searching for improved model variants



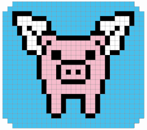
Model (from software)

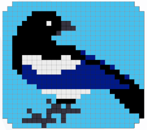
Meta-learning

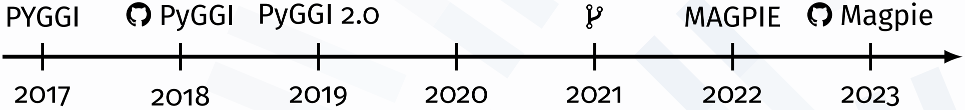
Model (improved)

https://github.com/bloa/magpie

A Brief History of Magpie







PYGGI 2017

PyGGI 2018

PyGGI 2.0 2019

2020

⑂ 2021

MAGPIE 2022

Magpie 2023

PYGGI 🗎 An, Kim, and Yoo, KSC 2017

PyGGI 🗎 An, Kim, Lee, and Yoo, GI@GECCO 2018

PyGGI 2.0 🗎 An, Blot, Petke, and Yoo, ESEC/FSE 2019

⑂ 🗎 Blot and Petke, IEEE TEVC 2021

MAGPIE 🗎 Blot and Petke, CoRR 2022

Outline

Introduction

GI as Seen by Magpie

- Software search space
- Representing software
- Searching for variants

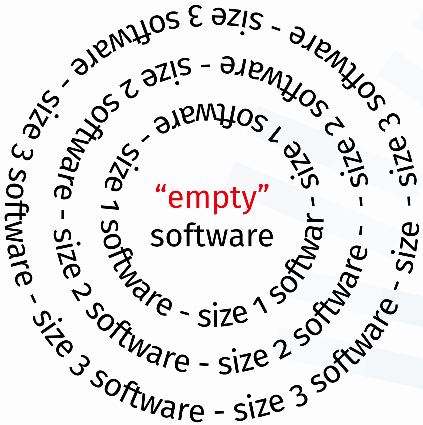
The Magpie Framework

Magpie in Practice

Future Directions and Challenges

Software Search Space

Program Synthesis: Software ∈ { All possible software }



“empty” software

size 1 software - size 1 softwar - size 1 software -

software - size 2 software - size 2 software - size 2 software - size 2

3 software - size 3 software - size 3 software - size - size 3 software - size 3

Natural representation

Software (model)

Key points

− Virtually infinite search space

− Hard to navigate

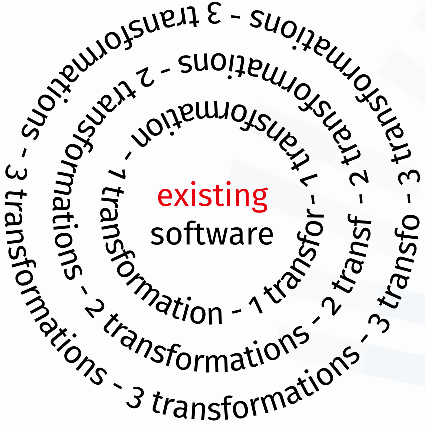
+ Maximum expressiveness

− Mostly “uninteresting”

Good software is far from the origin

Standing on the Shoulder of Giants

Genetic Improvement: Software + Transformation → Software'


existing
software
1 transformation - 1 transformation - 1 transfor
2 transformations - 2 transformations - 2 transf
3 transformations - 3 transformations - 3 transformations - 3 transfo

Ad hoc representation

Sequence of transformations

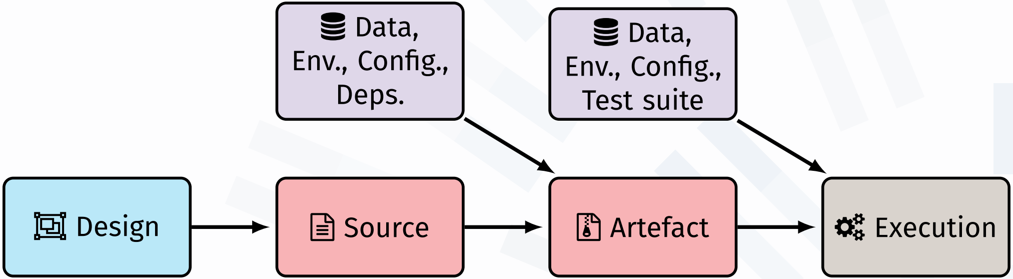
Key points

– Still virtually infinite
– Still hard to navigate
– Reduced expressiveness
+ Centred on all interesting variants

Optimised software is "very close"

What is Software?

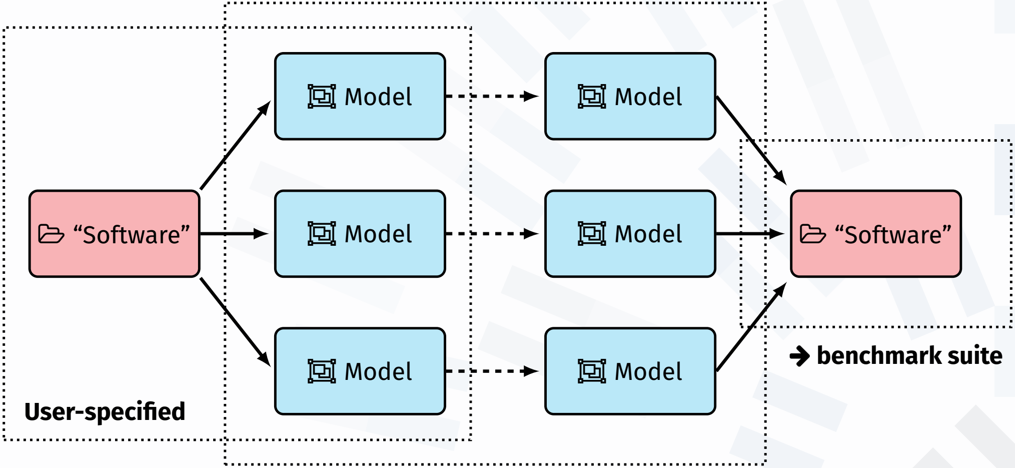
The description of an ultimately executable system?



Observations

- ► Most tools only ever target source code
- ► GI can target many stages of software development
- ► It doesn’t really impact the overall search process

A Model-Centred Approach



Model transformations

Terminology in Magpie

Model

A *representation* of one facet (e.g., file) of the target software

Variant

A list of models

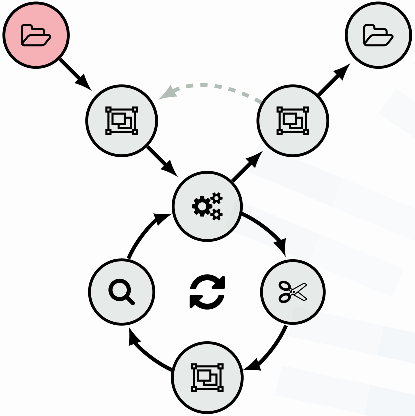
Edit

The *description* of a transformation

Patch

A list of edits

Searching for Variants



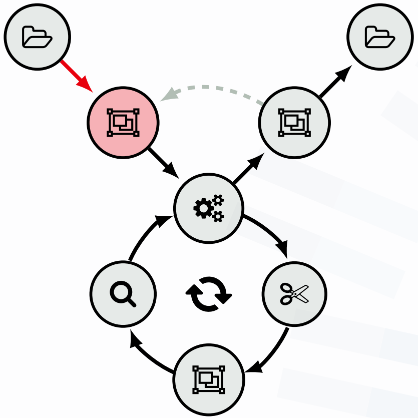
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   3.1 Add/remove transformation
   3.2 Apply to new variant
   3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



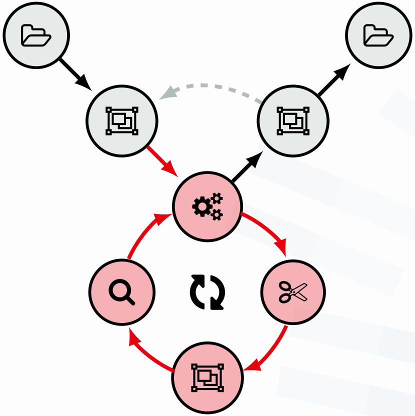
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   3.1 Add/remove transformation
   3.2 Apply to new variant
   3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



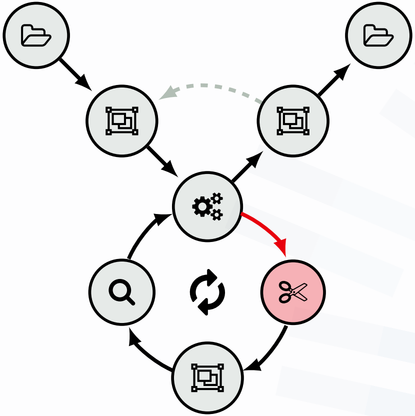
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   - 3.1 Add/remove transformation
   - 3.2 Apply to new variant
   - 3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



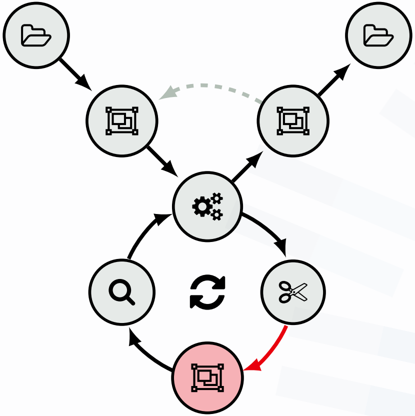
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   - 3.1 Add/remove transformation
   - 3.2 Apply to new variant
   - 3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



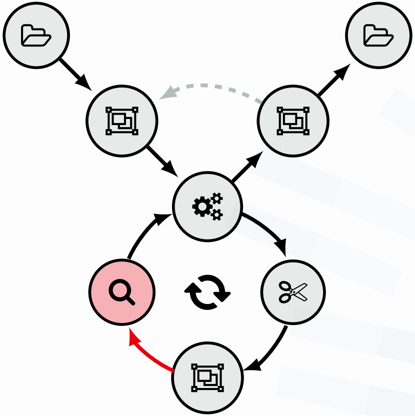
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   - 3.1 Add/remove transformation
   - 3.2 Apply to new variant
   - 3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



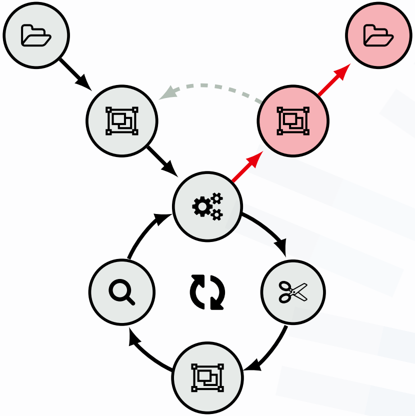
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   3.1 Add/remove transformation
   3.2 Apply to new variant
   3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



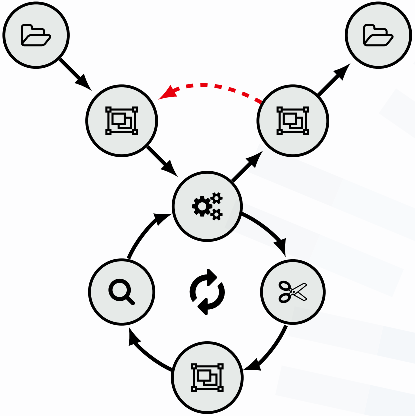
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   3.1 Add/remove transformation
   3.2 Apply to new variant
   3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Searching for Variants



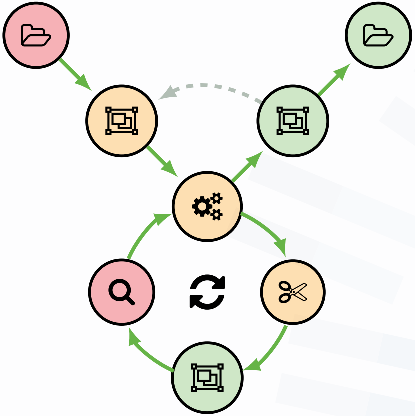
Standard search procedure

1. Start with existing software
2. Abstract parts on which to focus
3. Evolve model(s)
   - 3.1 Add/remove transformation
   - 3.2 Apply to new variant
   - 3.3 Evaluate fitness function
4. Enjoy your improved software

Optional additional steps

- Test for generalisation
- Minimise patch size
- Manually assess semantics

Components' Origin



User-provided (software)

- Reference software
- Evaluation benchmark

User-specified (scenario)

- Models
- Search process
- Model transformations
- Performance indicator

Automated (Magpie)

- Procedure

Why Does it Work?

The "*competent programmer*" hypothesis (from MT)

Competent programmers write programs that are close to being correct

The "*plastic surgery*" hypothesis (from APR)

1. Program source code changes occurring during development contain snippets that already exist in the codebase
2. These snippets can be efficiently found and exploited

The "*your software is not unique*" observation (brand new!)

There are *infinitely many* equivalent software, in a fractal fashion

Outline

Introduction

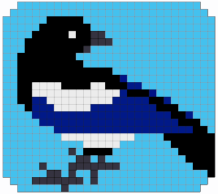
GI as Seen by Magpie

The Magpie Framework
- Philosophy
- Architecture
- Key components

Magpie in Practice

Future Directions and Challenges

Magpie in a Nutshell



Magpie

- Modular Python framework for GI/MT/AC/CO
- Hack-friendly, for researchers!
- User-friendly, for developers!

Motivations

- Free, libre, and open-source → MIT license, on GitHub
- Small sized but very versatile → ≈ 4000 loc, *many functionalities*
- Multi-purpose and reusable → F/NF properties, scenario-based

https://github.com/bloa/magpie

Magpie’s Architecture (Simplified)

```
magpie/ (← the Git repository)
    docs/ (← Diátaxis-based)
    magpie/
        core/ (← base classes + scenario config ≈ 1600 loc)
        models/ (← astor, line, config, xml, ≈ 1350 loc)
        algos/ (← LS, GP, validation, ablation ≈ 750 loc)
        bin/ (← entry points, ≈ 200 loc)
        scripts/ (← utilities, ≈ 270 loc)
        __main__.py (← main entry point)
    examples/
        triangle-c/ (← target software)
            _magpie/ (← scenario files)
    tests/
```

https://diataxis.fr/

Magpie's Project Structure

```
/
├── magpie/ (← the one containing "__main__.py")
├── your_software/
├── scenario.txt
├── _magpie_work/ (← automatically generated)
│   └── your_software_167874800/
└── _magpie_logs/ (← automatically generated)
    ├── your_software_167874800.log
    ├── your_software_167874800.patch
    └── your_software_167874800.diff
```

In practice

```
› python3 magpie local_search --scenario scenario.txt
```

Scenario File: INI Configuration File

```
[software]
path = examples/triangle-rb
target_files =
    triangle.rb
fitness = repair

init_cmd = bash init_bug.sh
test_cmd = ruby test_triangle.rb

[search]
target_fitness = 0
max_steps = 100
possible_edits =
    LineReplacement
    LineInsertion
    LineDeletion
```

Magpie's Models

4 types of models

- AstorModel → statement replacement, insertion, deletion
- LineModel → line replacement, insertion, deletion
- XmlModel → node replacement, insertion, deletion,
  node text setting, wrapping
- ConfigModel → parameter setting

Mapped through the scenario file

```
[software]
model_rules =
    *.params : ParamFileConfigModel
    *.xml : SrcmlModel
    * : LineModel
```

XML example

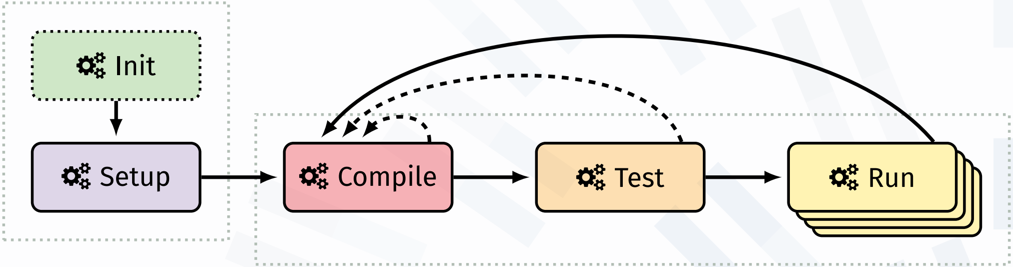
```
<cpp:include>#<cpp:directive>include</cpp:directive> <cpp:fil

<comment type="line">// rotate three values</comment>
<function><type><name>void</name></type> <name>rotate</name><

  <comment type="line">// copy original values</comment>
  <decl_stmt><decl><type><name>int</name></type> <name>tn1</n

  <comment type="line">// move</comment>
  <expr_stmt><expr><name>n1</name> <operator>=</operator> <na
  <expr_stmt><expr><name>n2</name> <operator>=</operator> <na
  <expr_stmt><expr><name>n3</name> <operator>=</operator> <na
```

Notes

- SrcML supports C, C++, C#, Java
- similar XML can be obtained using reflection and the Visitor pattern
- Magpie provides XML cleaning and processing utilities

https://www.srcml.org/

Multi-Step Evaluation



Configured in scenario file

- `{step}_cmd`
- `{step}_timeout`
- `{step}_lengthout`

Step-specific logging

- `{step}_CLI_ERROR`
- `{step}_CODE_ERROR`
- `{step}_TIMEOUT`
- `{step}_LENGTHOUT`
- `{step}_PARSE_ERROR`
- `SUCCESS` (w.r.t. evaluation)

Magpie’s Fitness Functions

Execution time

- `time`
- `posix_time` `/real (\S+)/` ∈ run.stderr
- `perf_time` `/(\S+) seconds time elapsed/` ∈ run.stderr
- `perf_instructions` `/(\S+) instructions/` ∈ run.stderr

Program repair

- `repair` currently supports JUnit, pytest, and minitest
  e.g., `/Failures: (\d+)/` and `/Tests run: (\d+)/` ∈ test.stdout

Misc.

- `bloat_{lines,words,chars}`
- `output` `/MAGPIE_OUTPUT: (\S+)/` ∈ run.stdout

Magpie's Entry Points

Usage

```
> python3 magpie [[magpie/]{bin,scripts}/]ENTRY[.py]
> mv magpie/{bin,scripts}/ENTRY.py .; python3 ENTRY.py
> python3 -m magpie.{bin,scripts}.ENTRY
```

Search

- bin/local_search.py
- bin/genetic_programming.py

Validation

- bin/show_patch.py
- bin/revalidate_patch.py
- bin/minify_patch.py
- bin/ablation_analysis.py

Misc.

- bin/show_locations.py

Scripts

- scripts/clear_xml.py
- scripts/line_to_xml.py
- scripts/python_to_xml.py

Outline

Introduction

GI as Seen by Magpie

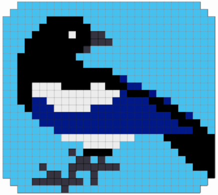
The Magpie Framework

Magpie in Practice

- First steps
- Ditching the training wheels
- Harder, better, faster, stronger

Future Directions and Challenges

Magpie in Practice



Magpie

- Modular Python framework for GI/MT/AC/CO
- Hack-friendly, for researchers!
- User-friendly, for developers!

Try it now!

```
> git clone https://github.com/bloa/magpie.git
> cd magpie
> python3 magpie local_search \
      --scenario examples/triangle-c/_magpie/scenario_slow.txt
```

First Steps with triangle-c

```
magpie/ (← the Git repository)
└── examples/
    └── triangle-c
        ├── triangle.c
        ├── triangle.h
        ├── run_triangle.c
        ├── test_triangle.c
        ├── makefile
        ├── init_slow.sh
        └── _magpie/
            ├── scenario_slow.txt
            ├── triangle_slow.c.diff
            ├── triangle_slow.h.diff
            └── triangle_slow.c.xml
```

Init (only once, at the start)

```
> bash init_slow.sh
```

Setup (skipped)

Compile

```
> make
```

Test

```
> ./test_triangle
```

Run

```
> ./run_triangle
```

`triangle_slow.c.xml` **was obtained using SrcML**

The Bug

triangle_slow.c.diff

```
--- triangle.c
+++ triangle_slow.c
@@ -1,9 +1,16 @@
 #include "triangle.h"

+void delay() {
+  const struct timespec ms = {0, 0.001*1e9}; //(0.001s)
+  nanosleep(&ms,NULL); /*ignores possible errors*/
+}
+
 int classify_triangle(double a, double b, double c) {
   double tmp;

+  delay();
+
   // sort the sides so that a <= b <= c
   if(a > b) {
```

The Scenario

scenario_slow.txt

```
[software]
path = examples/triangle-c
target_files = triangle.c.xml
fitness = time

init_cmd = bash init_slow.sh
compile_cmd = make test_triangle run_triangle
test_cmd = ./test_triangle
run_cmd = ./run_triangle
run_timeout = 1

[search]
max_steps = 100
max_time = 60
possible_edits =
    SrcmlStmtReplacement
    SrcmlStmtInsertion
    SrcmlStmtDeletion
```

Searching for Variants

```
> python3 magpie local_search \
      --scenario examples/triangle-c/_magpie/scenario_slow.txt
```

```
==== SEARCH: FirstImprovement ====
~~~~ WARMUP ~~~~
WARM  SUCCESS                 0.07 (--) [0 edit(s)]
WARM  SUCCESS                 0.08 (--) [0 edit(s)]
WARM  SUCCESS                 0.08 (--) [0 edit(s)]
REF   SUCCESS                 0.08 (--) [0 edit(s)]

~~~~ START ~~~~
1     TEST_CODE_ERROR         None (--) [1 edit(s)]
2     SUCCESS                *0.08 (96.18%) [1 edit(s)]
3     SUCCESS                *0.07 (92.74%) [2 edit(s)]
4     SUCCESS                 0.08 (102.29%) [1 edit(s)]
5     SUCCESS                 0.08 (96.18%) [1 edit(s)] [cached]
6     TEST_CODE_ERROR         None (--) [3 edit(s)]
7     TEST_CODE_ERROR         None (--) [3 edit(s)]
8     SUCCESS                 0.15 (188.92%) [3 edit(s)]
```

Finding Variants

```
24      TEST_CODE_ERROR        None  (--)  [4 edit(s)]
25      COMPILE_CODE_ERROR     None  (--)  [4 edit(s)]
26      SUCCESS               *0.00  (4.08%)  [2 edit(s)]
^C~~~~ END ~~~~

==== REPORT ====
Termination: keyboard interrupt
Log file: /home/aymeric/git/magpie/_magpie_logs/triangle-c_17
Patch file: _magpie_logs/triangle-c_1712601481.patch
Diff file: _magpie_logs/triangle-c_1712601481.diff
Reference fitness: 0.08
Best fitness: 0.00

==== BEST PATCH ====
SrcmlStmtInsertion(('triangle.c.xml', '_inter_block', 35), ('

==== DIFF ====
--- before: triangle.c
+++ after: triangle.c
```

Final Patch

```
--- before: triangle.c
+++ after: triangle.c
@@ -2,7 +2,7 @@

 void delay() {
   const struct timespec ms = {0, 0.001*1e9}; //tv_sec=0, tv_
-  nanosleep(&ms,NULL); /*ignores possible errors*/
+    /*ignores possible errors*/
 }

 int classify_triangle(double a, double b, double c) {
@@ -40,6 +40,7 @@
   if(a == b || b == c)/*auto*/{

     return ISOSCELES;
+    return EQUILATERAL;
   }/*auto*/
   return SCALENE;
 }
```

What to do with a Patch?

Look at it (and possibly `--keep` it)

```
› python3 magpie show_patch \
      --scenario examples/triangle-c/_magpie/scenario_slow.txt \
      --patch _magpie_logs/triangle-c_1712601481.patch \
      --keep
```

Revalidate it (possibly on a different scenario)

```
› python3 magpie revalidate_patch --scenario ...  --patch ...
```

Minify it (possibly on a different scenario)

```
› python3 magpie minify_patch --scenario ...  --patch ...
```

Study it

```
› python3 magpie ablation_analysis --scenario ...  --patch ...
```

Changing the Fitness Function

Execution time (Python) → noisy, precise, “free”

- [software] fitness = time
- [software] run_cmd = ./run_triangle

Execution time (POSIX) → noisy, not very precise, UNIX

- [software] fitness = posix_time
- [software] run_cmd = /usr/bin/time -p ./run_triangle

Execution time (perf) → noisy, very precise, linux

- [software] fitness = perf_time
- [software] run_cmd = perf stat ./run_triangle

CPU instructions (perf) → “deterministic”, extremely precise, linux/HW

- [software] fitness = perf_instructions
- [software] run_cmd = perf stat ./run_triangle

Changing More Scenario Specifics

Setting a random seed

- [magpie] seed = ... (integer)

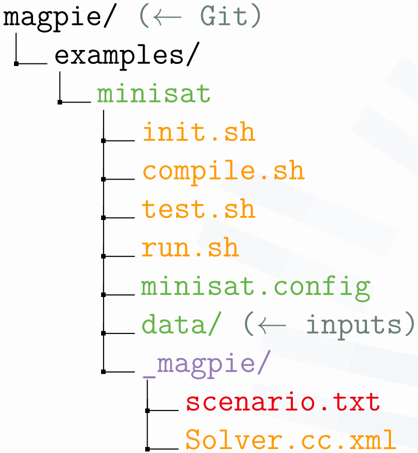
Configuring warmup

- [search] warmup = ... (strictly positive integer)
- [search] warmup_strategy = ... ({last, min, max, mean, median})

Configuring the stopping condition

- [search] max_steps = ... (integer or empty)
- [search] max_time = ... (integer or empty)
- [search] target_fitness = ... (integer or empty)

Ditching the Training Wheels with MiniSAT



magpie/ (← Git)

examples/

minisat

init.sh

compile.sh

test.sh

run.sh

minisat.config

data/ (← inputs)

_magpie/

scenario.txt

Solver.cc.xml

Init (only once, at the start)

- Download, cache, and extract `minisat-2.2.0.tar.gz`
- Insert `Solver.cc.xml`

Setup (only once, at the start)

- Prophylactic compilation

Compile

- Basically just `make`

Test

- Check on small instances

Run

- Run on larger instances

`Solver.cc.xml` **was obtained using SrcML**

Parameter Configuration

Scenario file

```
[software]
target_files = minisat.params
[search]
possible_edits = ParamSetting
```

Parameter file

```
CLI_PREFIX = "-"
CLI_GLUE = "="
TIMING = "run"

lbd-cut [3, 10] [5]
lbd-cut-max [4, 30] [10]
cp-increase g[5000, 50000] [15000]
core-tolerance (0.0, 1.0) [0.02]
...
```

Configuration Files

```
# magic constants
CLI_...
TIMING="setup compile"

# categorical parameters
name {value1, value2, value3} [value1]
name {True, False, None} [None]

# numerical parameters
name (0, 1.0) [0] # uniform continuous
name e(0, 1.0) [0] # exponential
name [0, 100] [0] # uniform discrete
name g[-999999999, 999999999] [0] # geometric

# and more!
name1 | name2 == True # conditional
{name1 == True, name2 == True} # forbidden combination
@name {True, False} [True] # invisible parameter
name$suffix {True, False} [True] # invisible suffix
```

Without Batch Processing

Scenario file

```
[software]
run_cmd = bash run.sh
```

Run script

```
#!/bin/sh
./simp/minisat data/uf50-01.cnf $@
./simp/minisat data/uf50-02.cnf $@
./simp/minisat data/uf100-01.cnf $@
./simp/minisat data/uf100-02.cnf $@
...

./simp/minisat data/uuf50-01.cnf $@
./simp/minisat data/uuf50-02.cnf $@
./simp/minisat data/uuf100-01.cnf $@
./simp/minisat data/uuf100-02.cnf $@
...
```

Adding Batch Processing

Scenario file

```
[software]
run_cmd = bash run.sh {INST}

[search]
batch_instances =
    file:data/inst_sat.txt
    ___
    file:data/inst_unsat.txt
batch_sample_size = 4
```

Run script

```
#!/bin/sh
./simp/minisat $@
```

Combining Models

Scenario file

```
[software]
target_files =
    minisat.params
    simp/Solver.cc.xml
[search]
possible_edits =
    ParamSetting
    SrcmlStmtDeletion
    SrcmlStmtReplacement
    SrcmlStmtInsertion
```

Order of operations

1. Pick a possible type of edit
2. Find a suitable location to apply it
3. Generate the rest of the ingredients

Types of XML Edits

Node deletion, replacement

```
class SrcmlStmtDeletion(AbstractXmlNodeDeletion):
    NODE_TAG = ’stmt’
```

Node insertion

```
class SrcmlStmtInsertion(AbstractXmlNodeInsertion):
    NODE_PARENT_TAG = ’block’
    NODE_TAG = ’stmt’
```

Text setting

```
class SrcmlNumericSetting(AbstractXmlTextSetting):
    NODE_TAG = ’number’
    CHOICES = [’-1’, ’0’, ’1’]
```

Text wrapping

```
class SrcmlRelativeNumericSetting(AbstractXmlTextWrapping):
    NODE_TAG = ’number’
    CHOICES = [(’(’, ’+1)’), (’(’, ’-1)’), (’(’, ’/2)’), (’(’
```

Adding (Scenario-Specific) Custom Code to Magpie

custom.py

```
import magpie

class TypeReplacement(magpie.models.xml.NodeReplacement):
    NODE_TYPE = ’type’

class TypeSetting(magpie.models.xml.TextSetting):
    NODE_TYPE = ’type’
    CHOICES = [’float’, ’double’ , ’int’]

magpie.utils.known_edits.append(TypeReplacement)
magpie.utils.known_edits.append(TypeSetting)
```

Scenario file

```
[magpie]
import = custom.py
[search]
possible_edits = TypeReplacement TypeSetting
```

Modifying the XML Tree

Scenario file

```
[srcml]
rename =
    stmt: break continue decl_stmt do expr_stmt for goto if r
focus = block stmt
internodes = block
process_pseudo_blocks = True
process_literals = False
process_operators = False
```

Modifying per-model

```
[software]
model_config =
    *.params : [paramconfig]
    *.xml : [srcml]
```

Outline

Introduction

GI as Seen by Magpie

The Magpie Framework

Magpie in Practice

Future Directions and Challenges

The Road Ahead

New functionalities

- New performance indicators (e.g., memory, energy)
- User-friendly multi-criteria optimisation
- Tree-sitter support

Quality of life improvements

- Better OS/container/cluster support
- Better XML/srcML-related tooling

House keeping

- Better documentation
- More tutorials

🔗 `https://tree-sitter.github.io/tree-sitter/`

The Magpie Wish List

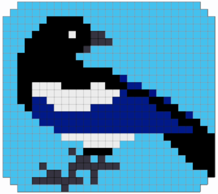
From a tooling perspective

- Interactive search visualisation
- Interactive model visualisation
- Fine-grained user interaction

From a research perspective

- Fault localisation
- Semantic search guidance
- Performance prediction
- Two-way explanations

Final Words



Magpie, One Framework to Rule Them All!

- Modular Python framework for GI/MT/AC/CO
- Hack-friendly, for researchers!
- User-friendly, for developers!

Use it now!

```
> git clone https://github.com/bloa/magpie.git
> cd magpie
> python3 magpie local_search \
         --scenario examples/triangle-c/_magpie/scenario_slow.txt
```

Please do not hesitate to reach out for support!