EPSRC

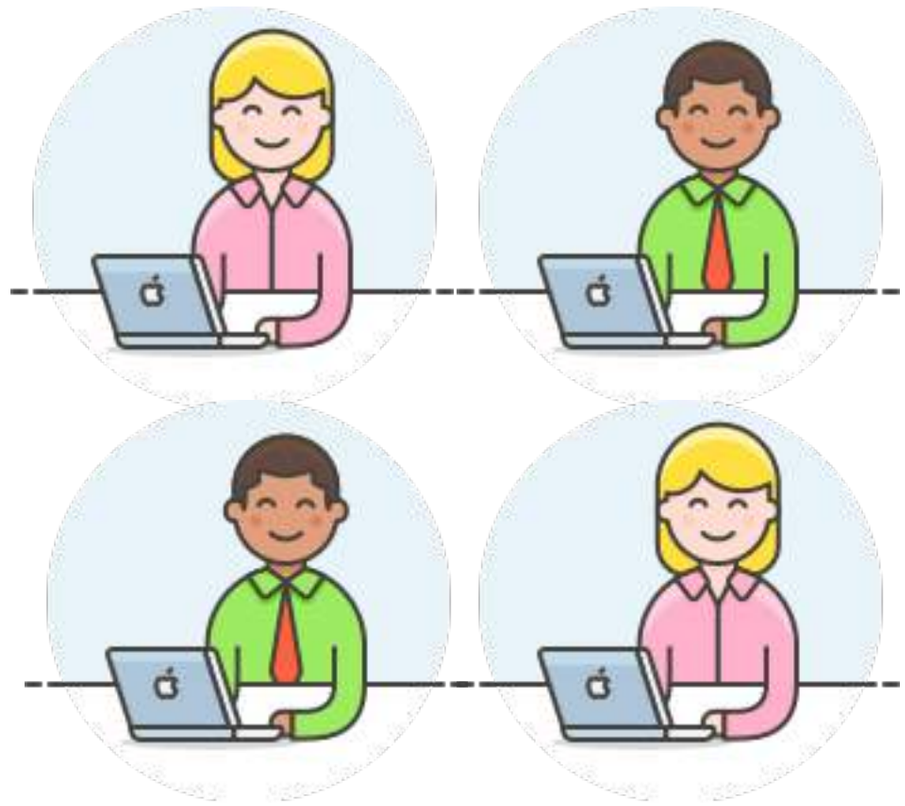
Genetic Improvement of Software

Justyna Petke

CREST UCL

EPSRC

Human Programming



Programmer icons by Streamline Icons http://www.streamlineicons.com License: CC Attribution 4.0

Justyna Petke, UCL

EPSRC

Automated 🏆 Programming

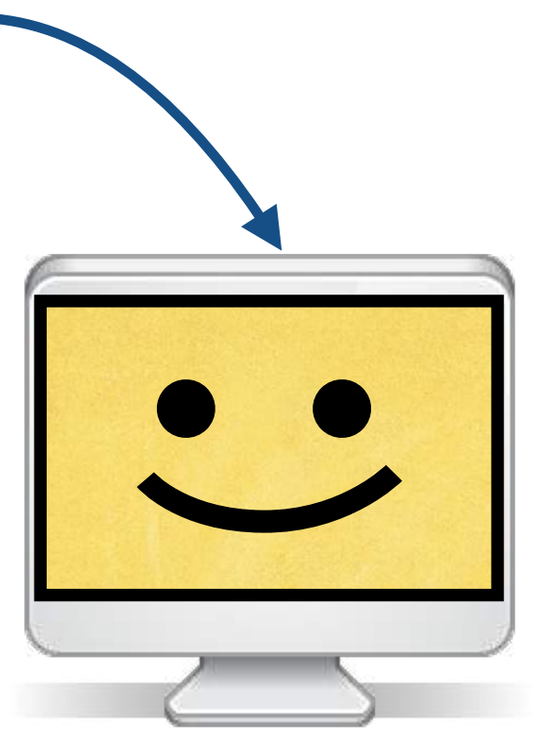
Justyna Petke, UCL

EPSRC

Automated Software Improvement

human writes code

computer improves it



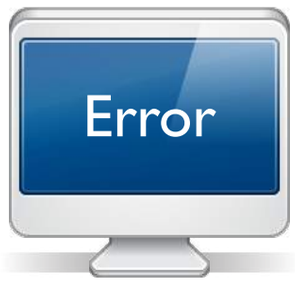
Justyna Petke, UCL

EPSRC

Functional Properties

New Feature

Functionality Improvement

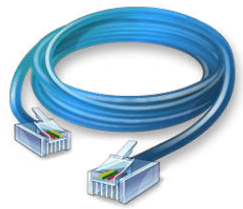


Bug Repair

Non-functional Properties

Execution Time

Memory



Bandwidth

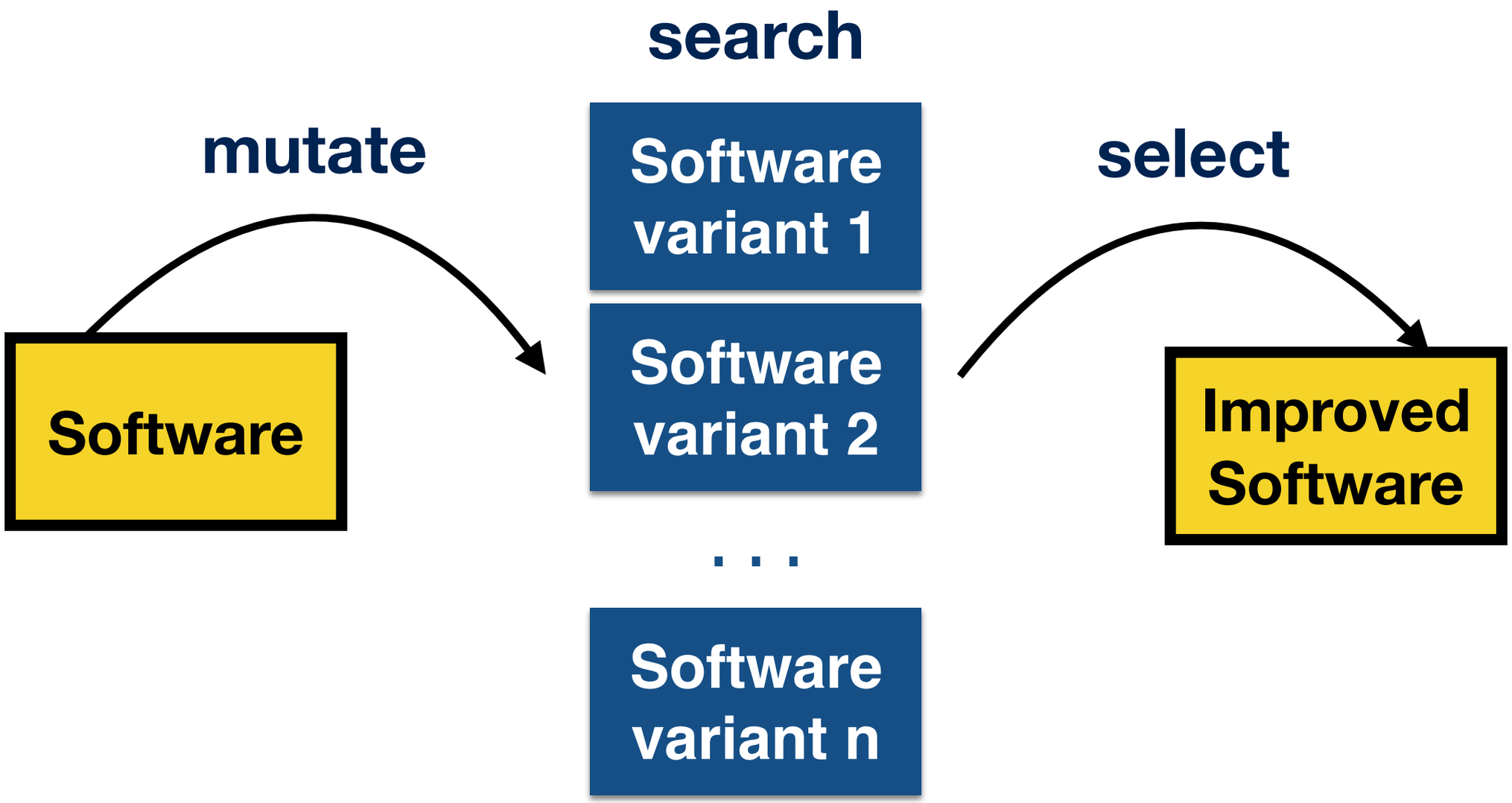
Energy

Size

EPSRC

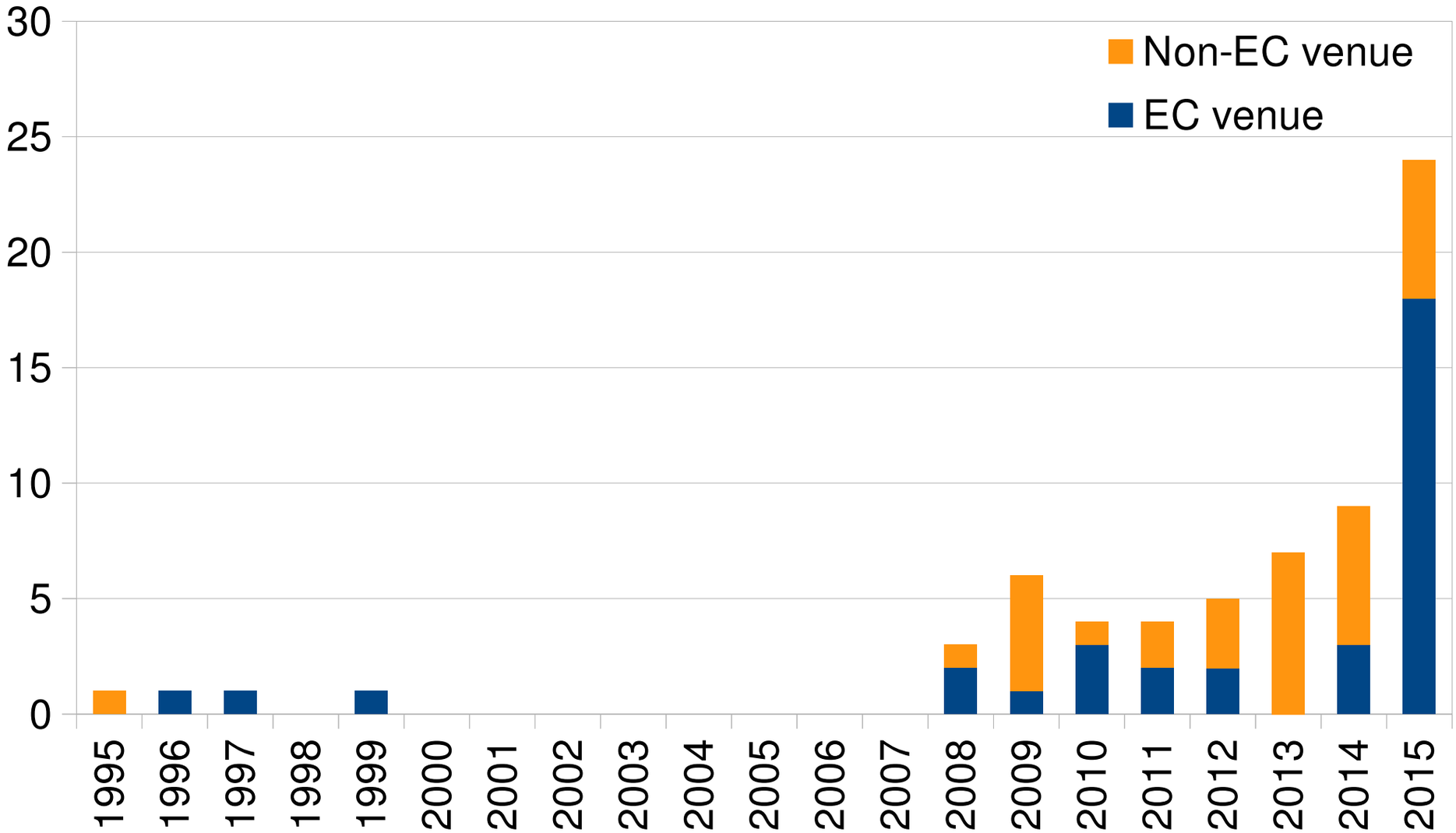
Genetic Improvement

mutates software to find an improved software version



Genetic Improvement of Software: A Comprehensive Survey

Petke et al., Transactions on Evolutionary Computation 2017

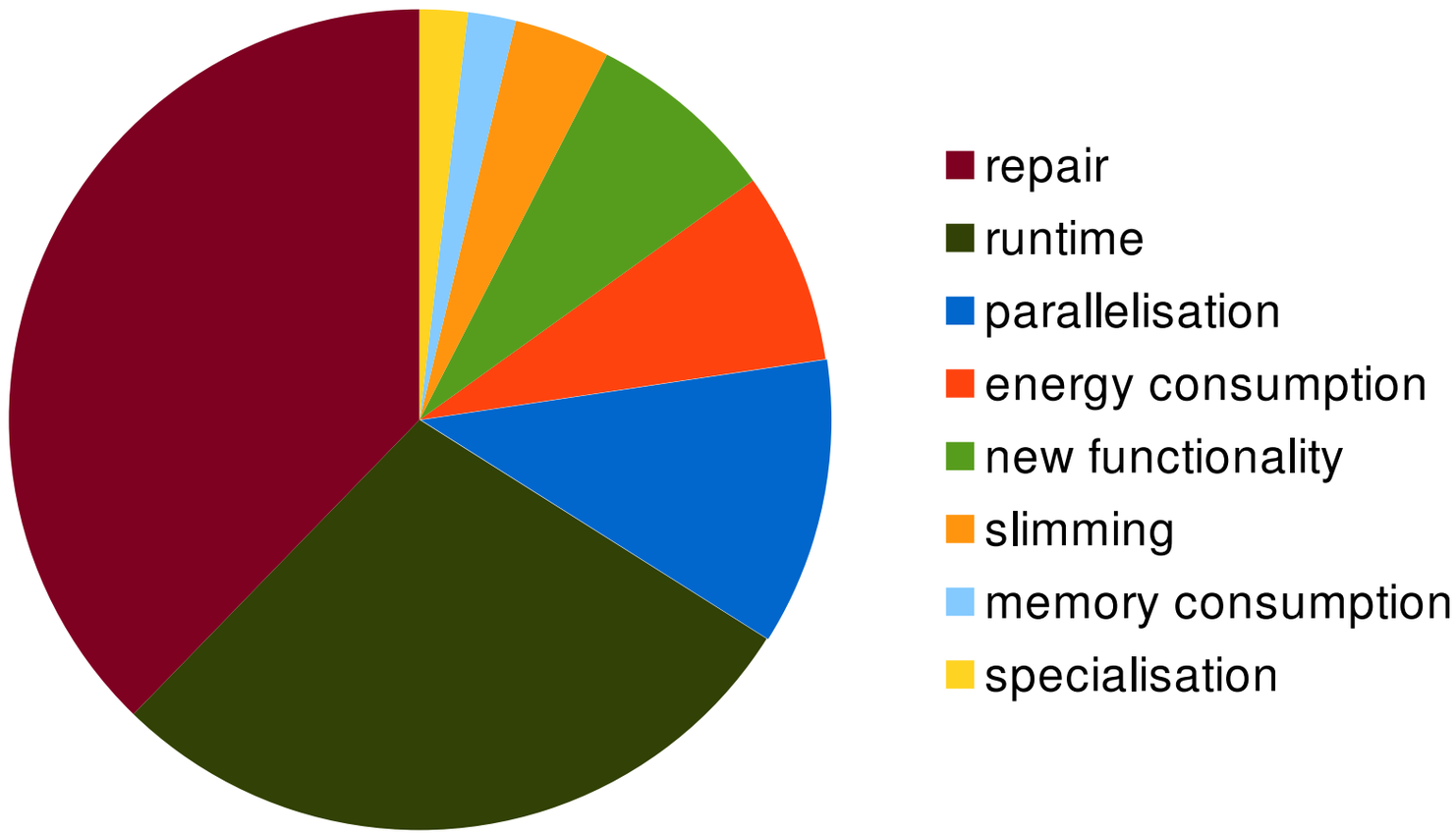


Justyna Petke, UCL

EPSRC

Genetic Improvement of Software: A Comprehensive Survey

Petke et al., Transactions on Evolutionary Computation 2017



Justyna Petke, UCL

EPSRC

The Genetic Programming Bibliography

http://gpbib.cs.ucl.ac.uk/

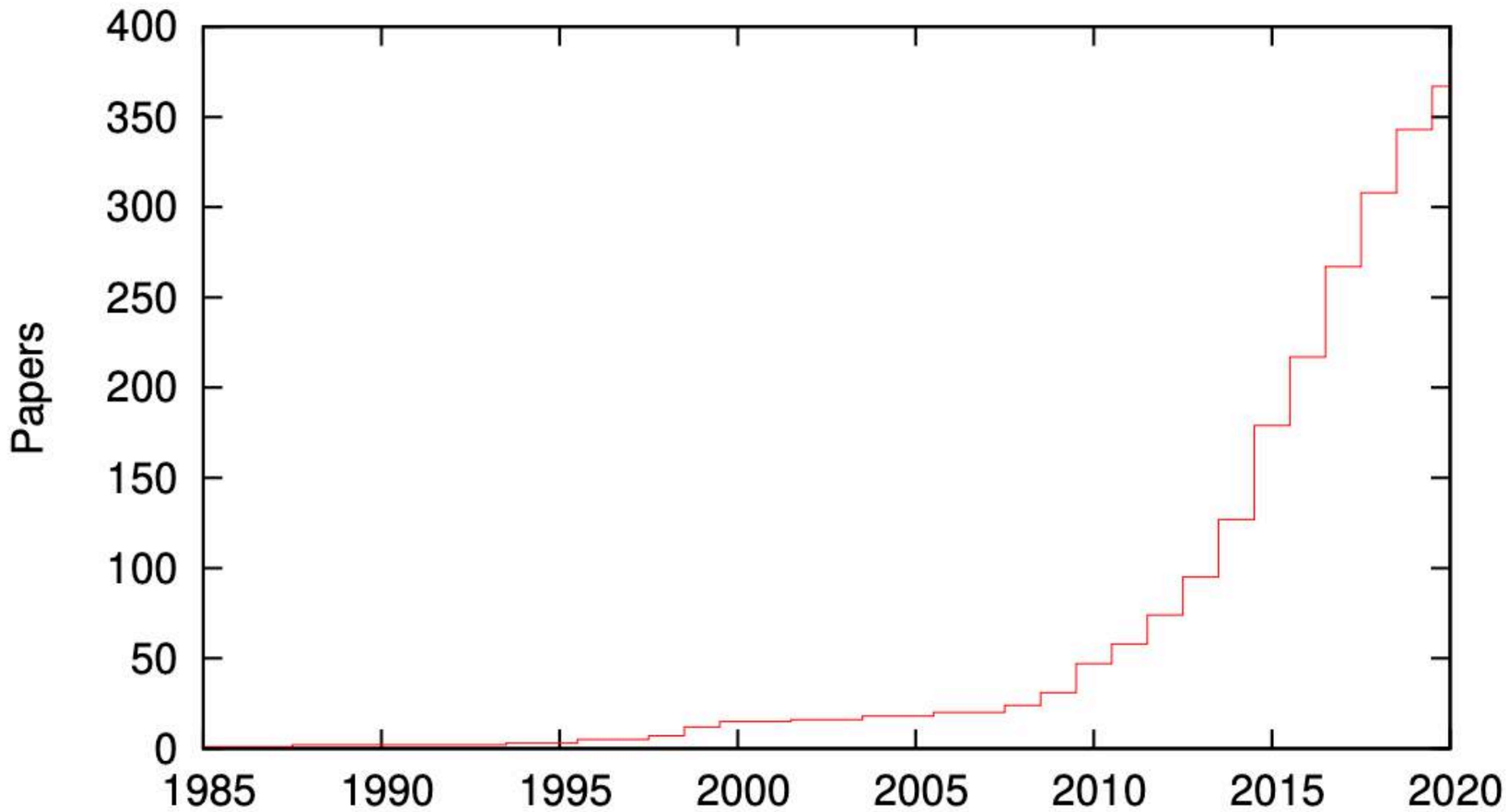


Fig. 1. No. of core papers on genetic improvement by year.

Justyna Petke, UCL

Example Real-world Applications

Bug fixing:

Janus Manager: “Fixing bugs in your sleep: how genetic improvement became an overnight success” Haraldsson et al. (GI@GECCO 2017)

SapFix (partially): “SapFix: automated end-to-end repair at scale.” Marginean et al. (ICSE-SEIP 2019)

Runtime improvement:

BarraCUDA: “Improving CUDA DNA Analysis Software with Genetic Programming” Langdon et al. (GECCO 2015)

EPSRC

Automated Software Specialisation: Preliminary Result




“Using Genetic Improvement & Code Transplants to Specialise a C++ Program to a Problem Class”
Justyna Petke, Mark Harman, William B. Langdon & Westley Weimer
17th European Conference on Genetic Programming (EuroGP 2014)

EPSRC

Human Software Improvement

Satisfiability (SAT) solvers e.g. **MiniSAT**

Used in several application domains:
planning, testing, verification and many other

EPSRC

Human Software Improvement

Satisfiability (SAT) solvers e.g. **MiniSAT**

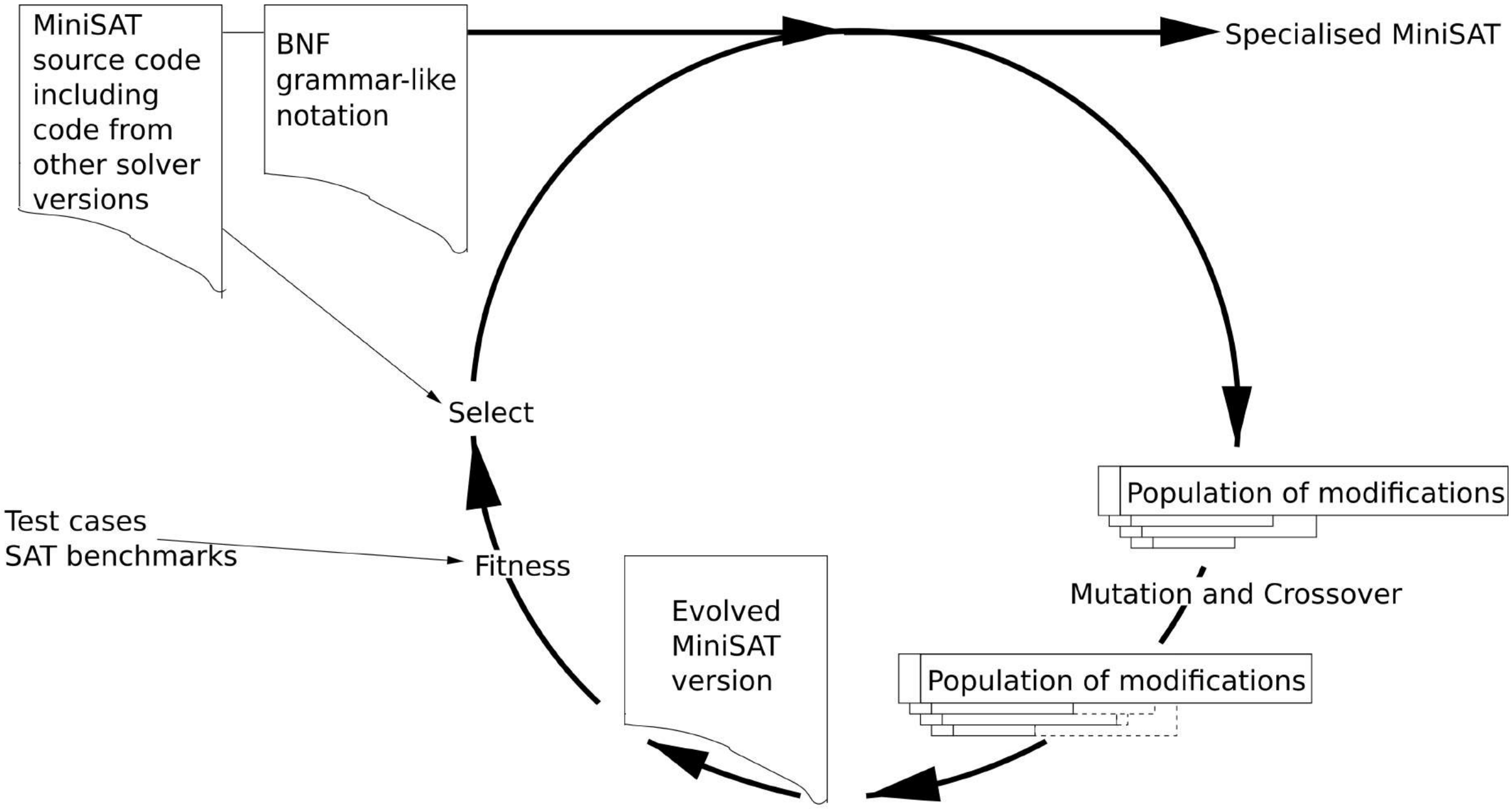
Used in several application domains:
planning, testing, verification and many other

Optimised for efficiency by expert human developers

EPSRC

Specialising MiniSAT Using GI

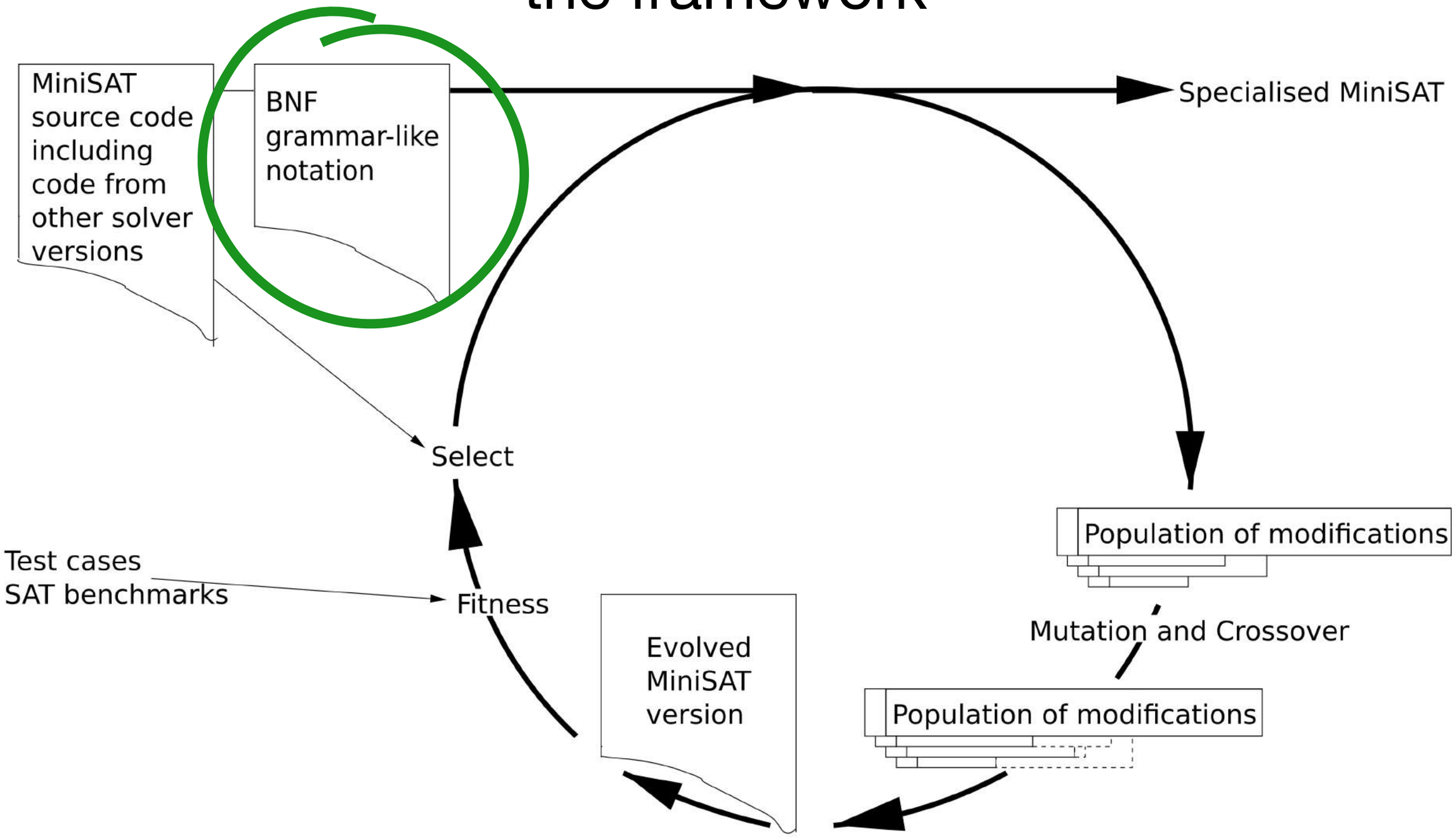
the framework



EPSRC

Specialising MiniSAT Using GI

the framework



EPSRC

Specialising MiniSAT Using GI

software representation

Software representation:

Changes at the level of lines of source code

Each individual is composed of a list of changes

Specialised notation used to preserve syntax

Justyna Petke, UCL

Specialising MiniSAT Using GI

software representation: example

<Solver_135> ::= " if" <IF_Solver_135> " return false;\n"

<IF_Solver_135> ::= "(!ok)"

<Solver_138> ::= "" <_Solver_138> "{Log_count64++;/*138*/}\n"

<_Solver_138> ::= "sort(ps);"

<Solver_139> ::= "Lit p; int i, j;\n"

<Solver_140> ::= "for(" <for1_Solver_140> ";" <for2_Solver_140> ";" <for3_Solver_140> ") {\n"

<for1_Solver_140> ::= "i=j=,p=lit_Undef"

<for2_Solver_140> ::= "i < ps.size()"

<for3_Solver_140> ::= “i++"

Justyna Petke, UCL

EPSRC

Specialising MiniSAT Using GI

code transplants

Search (Genetic Programming) has access to both:

the host program to be evolved

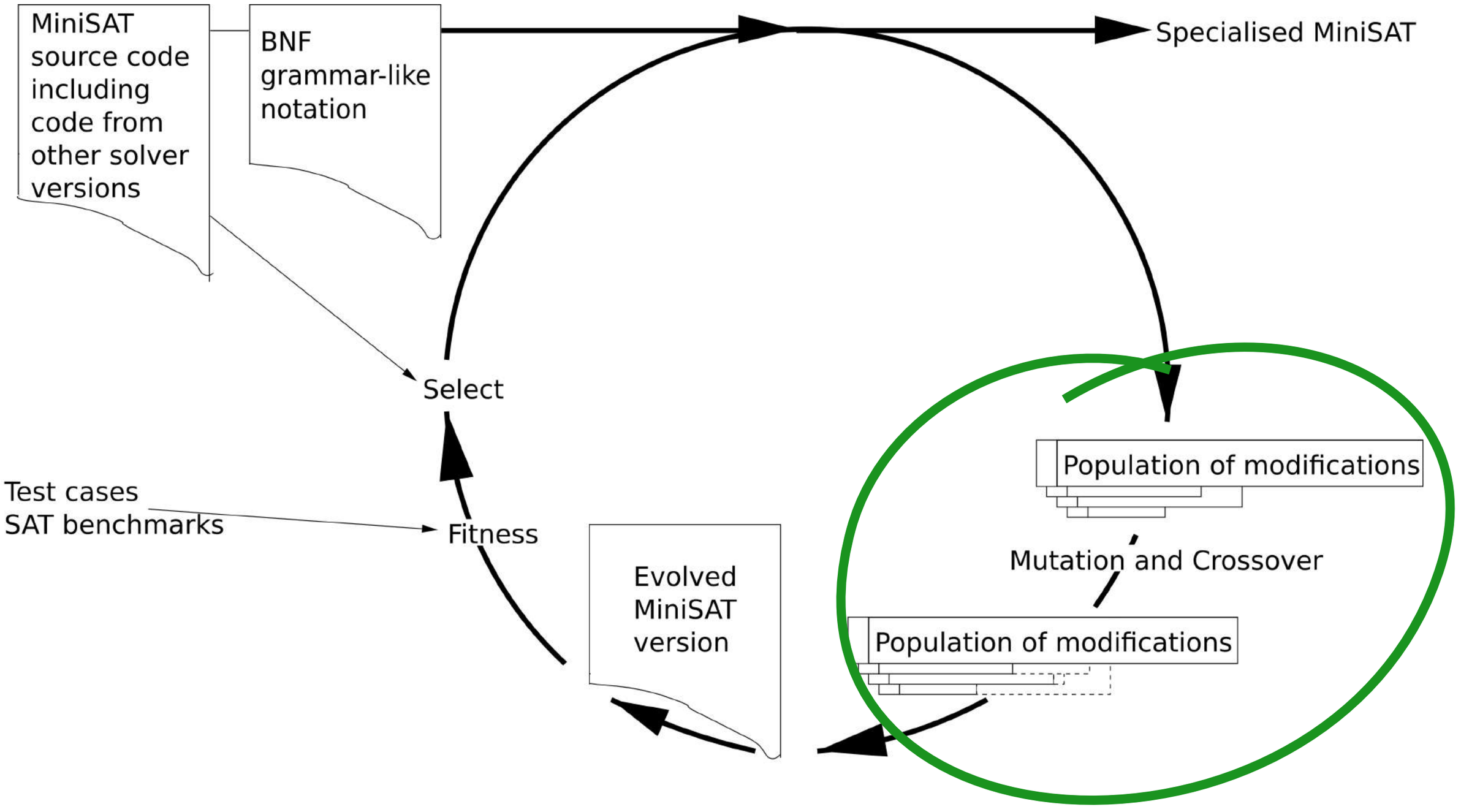
the donor program(s)

Justyna Petke, UCL

EPSRC

Specialising MiniSAT Using GI

the framework



EPSRC

Specialising MiniSAT Using GI

mutation

Addition of one of the following operations:

DELETE, COPY or REPLACE

Examples:

<_Solver_135>

<_Solver_138>+<_Solver_140>

<for3_Solver_140><for3_Solver_836>

EPSRC

Specialising MiniSAT Using GI

crossover

Concatenation of two individuals by appending two lists of mutations

<_Solver_135>

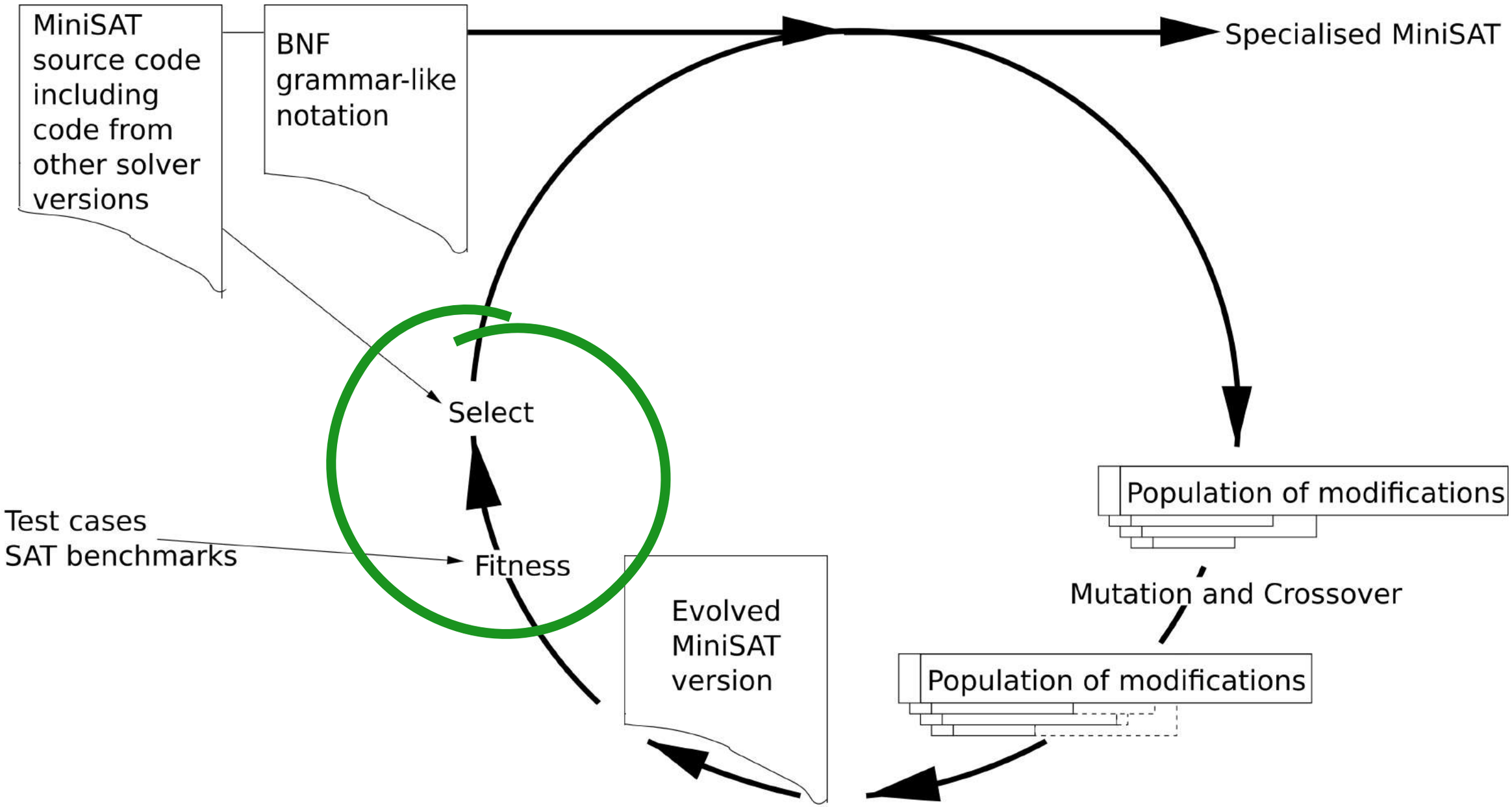
<_Solver_138>+<_Solver_140>

<_Solver_135> <_Solver_138>+<_Solver_140>

EPSRC

Specialising MiniSAT Using GI

the framework



EPSRC

Specialising MiniSAT Using GI

fitness

Fitness of each software variant is evaluated by:

Test case result

Runtime on the set of test cases

EPSRC

Specialising MiniSAT Using GI

selection

Fixed number of generations & fixed population size

Initial population contains single-mutation individuals

Top-half of population selected

Based on a threshold fitness value

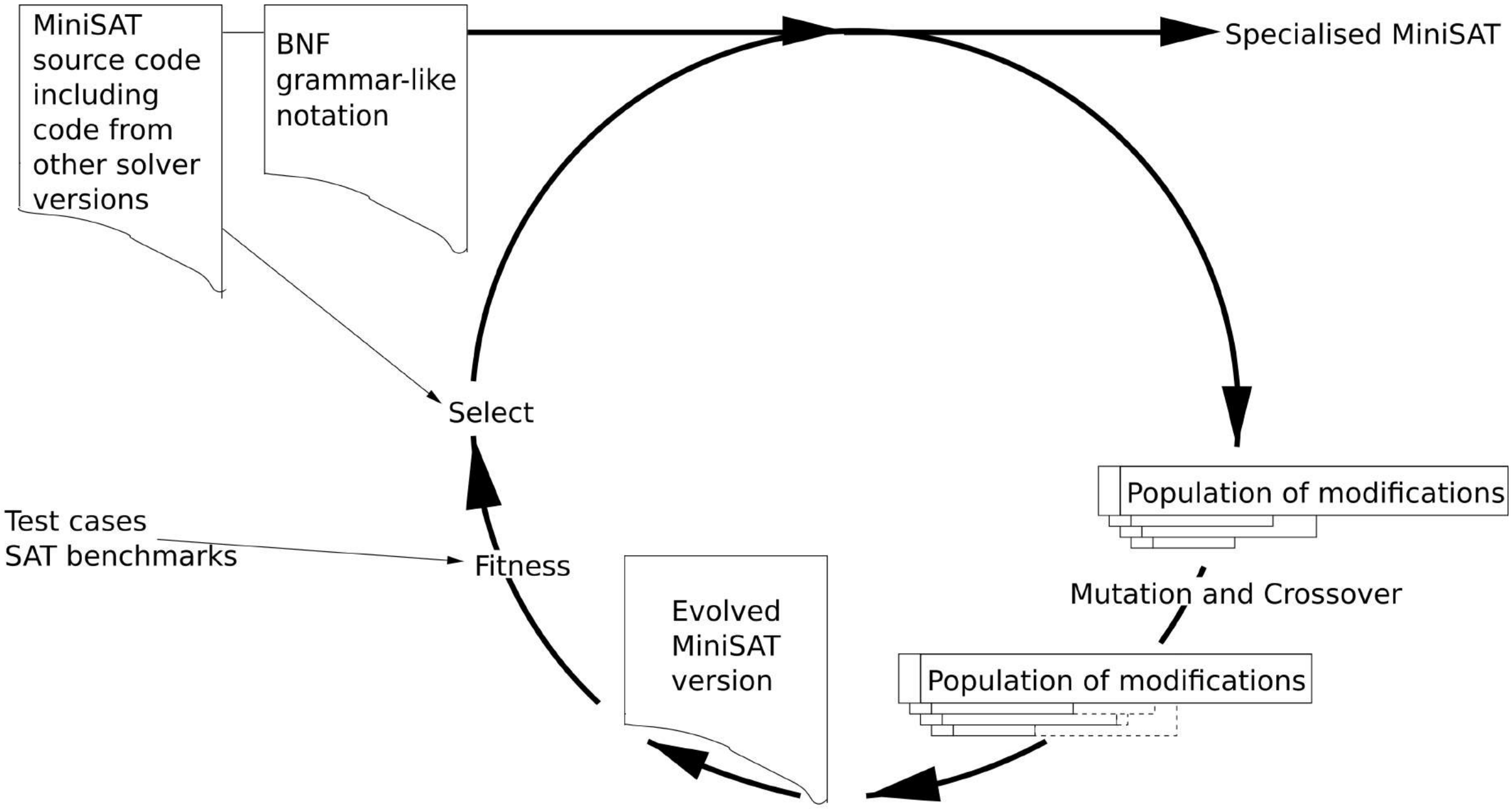
Mutation applied with 50% probability

Crossover applied with 50% probability

EPSRC

Specialising MiniSAT Using GI

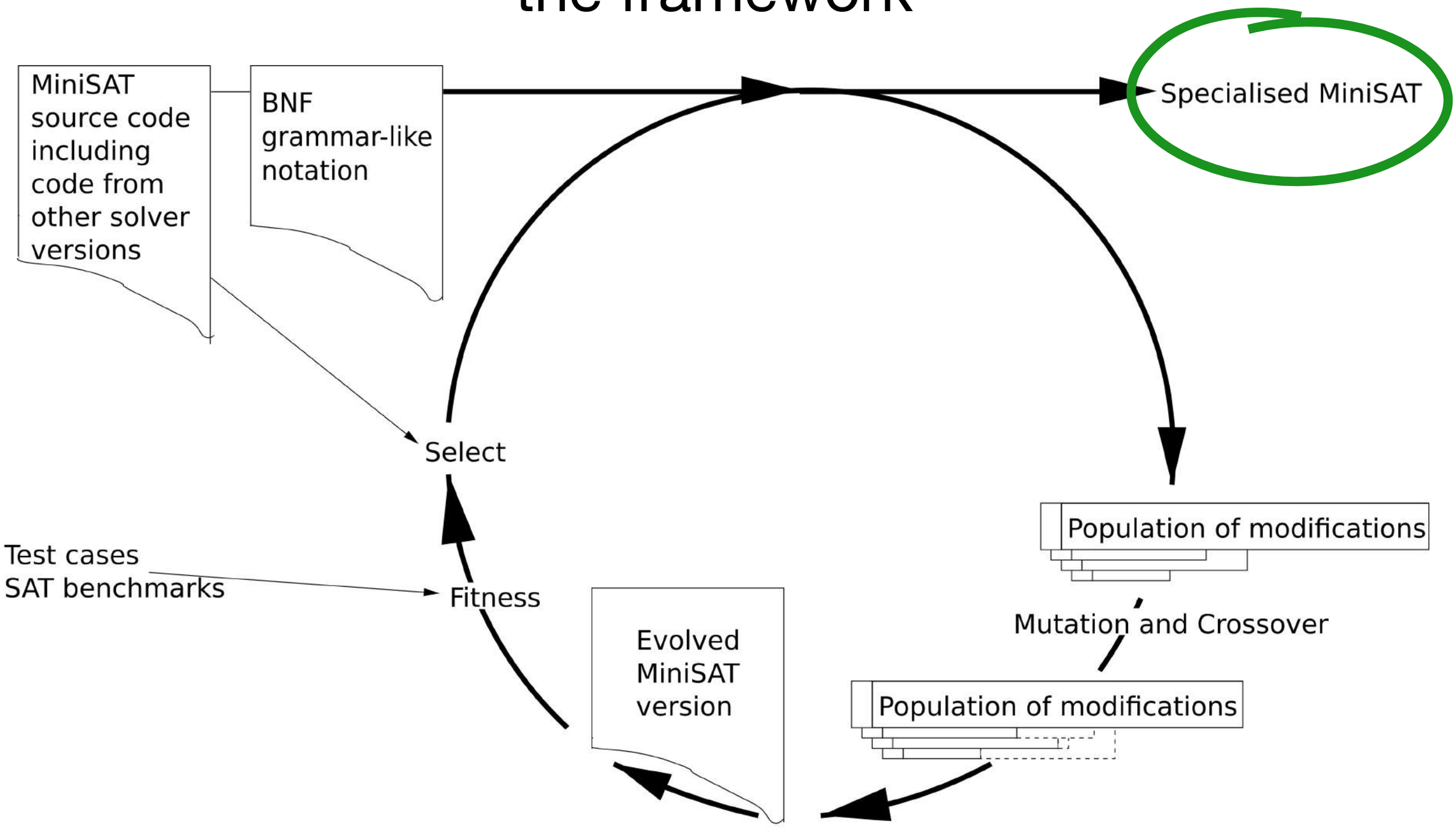
the framework



EPSRC

Specialising MiniSAT Using GI

the framework



EPSRC

Specialising MiniSAT Using GI

filtering step

Mutations in best individuals are often independent

Greedy approach used to combine best individuals

Justyna Petke, UCL

EPSRC

Results

Improved MiniSAT's performance by 17% by specialising to the set of benchmarks from the Combinatorial Interaction Testing domain

EPSRC

Reducing Energy Consumption Using Genetic Improvement

“Reducing Energy Consumption Using Genetic Improvement”; Bobby R. Bruce, Justyna Petke & Mark Harman; 27th International Genetic and Evolutionary Computation Conference (GECCO 2015)

Justyna Petke, UCL

EPSRC

Reducing Energy Consumption Using Genetic Improvement

up to 25% energy consumption reduction achieved when specialising for different application domains

MiniSAT Satisfiability Solver

"Reducing Energy Consumption Using Genetic Improvement"; Bobby R. Bruce, Justyna Petke & Mark Harman; 27th International Genetic and Evolutionary Computation Conference (GECCO 2015)

EPSRC

Specialising Software Using GI

Petke et al.,Transactions on Software Engineering (TSE) 2017

up to 3% improvement
achieved when
specialising for different
application domains

Image License: https://www.imagemagick.org/script/license.php

EPSRC

A few findings

Both problem-specific and general changes were produced by GI

Modifying the crossover/mutation rates has had no significant impact on the results

No obvious increase in the fitness value between generations has been observed

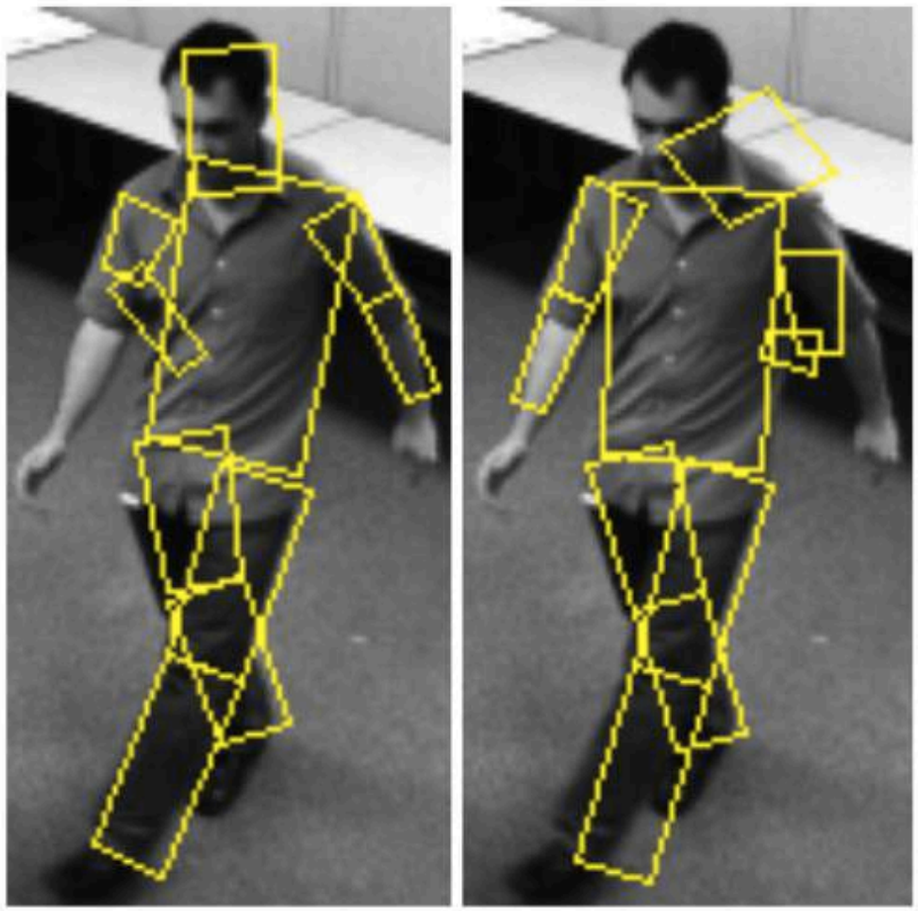
In some cases allowing GI to let slightly faulty versions to go to the next generation helped GI in finding an improved software version

Interactions between various mutations have been observed

EPSRC

Reducing Energy Consumption Using Genetic Improvement

original picture produced by body-tracking software



picture produced by software variant with 33% energy consumption reduction

Bobby R. Bruce, Justyna Petke, Mark Harman, and Earl T. Barr “Approximate Oracles and Synergy in Software Energy Search Spaces”, IEEE Transactions on Software Engineering (TSE 2017)

Justyna Petke, UCL

EPSRC

Reducing Energy Consumption Using Genetic Improvement

A few findings:

The number of effective modifications in the GI search space is scarce, but increases 15-fold if approximation is allowed

Pairings of effective modifications often do not produce a positive effect

Justyna Petke, UCL

EPSRC

Automated Software Transplantation

Idea: add a new feature to the software by transplanting it from a different piece of software

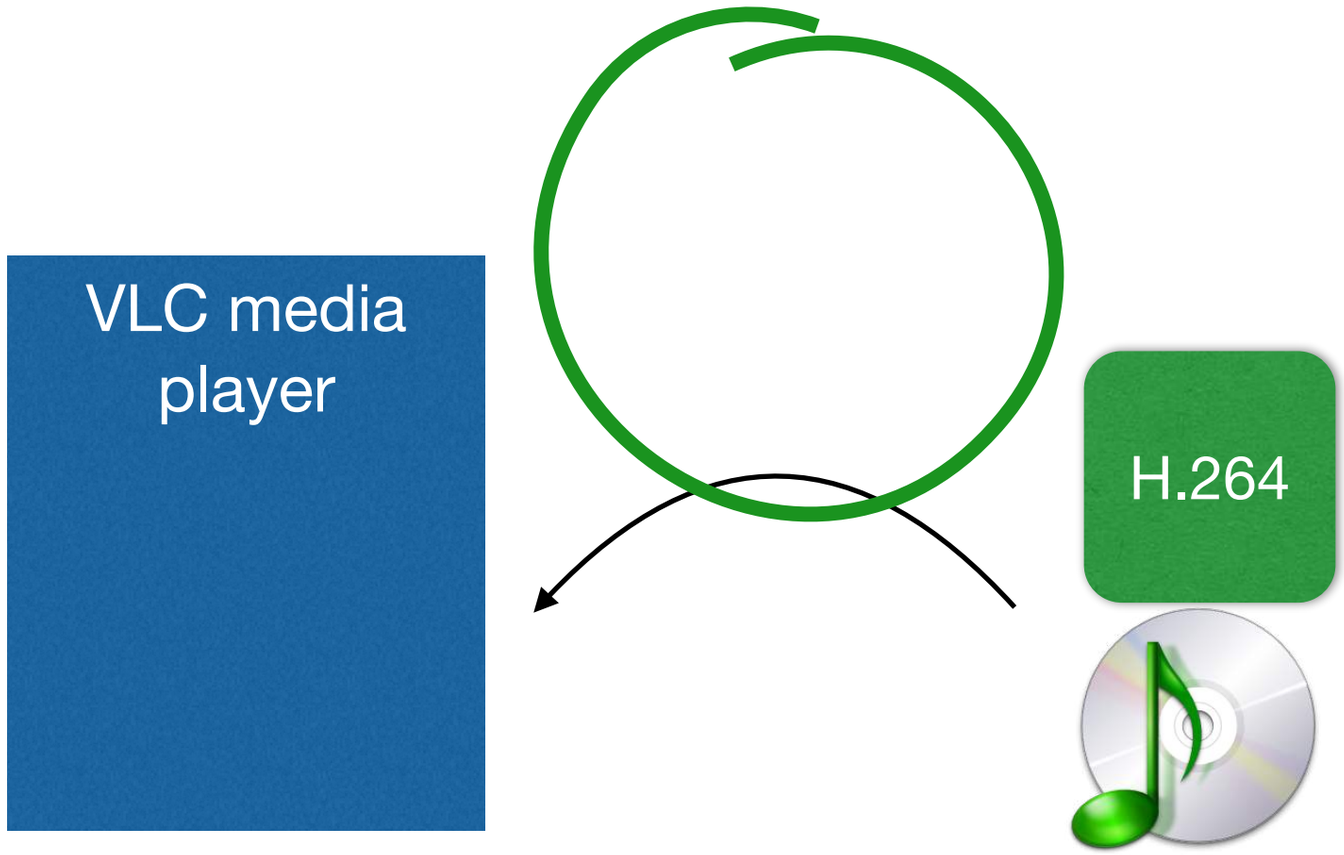
“Automated Software Transplantation”; Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean & Justyna Petke; 24th International Symposium on Software Testing and Analysis (ISSTA 2015)

ACM SIGSOFT Distinguished Paper Award Winner

Gold `Humie’ Winner at GECCO 2016

EPSRC

Automated Software Transplantation



Transplanted the H.264 video encoding functionality in 26 h of computation time
(compared with estimated 20 days of human effort)

Justyna Petke, UCL

Appeared at ISSTA 2015

- Overview
- Preprint
- MuScalpel
- Subjects
- Test Suites
- Artifact Evaluation
- SSBSE 2015

muScalpel

Implemented in TXL and C, muScalpel realizes μTrans and comprises 28k SLoCs, of which 16k is TXL, and 12k is C. muScalpel implements a custom version of GP. Unlike conventional GP, which creates an initial population from individuals that contain multiple statements, muScalpel generates an initial population of individuals with just 1 statement, uniformly selected. muScalpel's underlying assumption is that our organs need very few of the statements in their donor. Starting from one LOC gives muScalpel the possibility to find small solutions quickly. muScalpel focuses on evolving the organ's vein. muScalpel also inherits the limitations of TXL, such as its stack limit which precludes parsing large programs and its default C grammar's inability to properly handle preprocessor directives.

As we all know, software is often difficult to build and run, due to dependencies on its development environment and target platform. muScalpel is no exception. Please keep in mind that we built and ran muScalpel only on 64-bit Ubuntu 14.04 LTS machine, with 16 GB RAM, SSD and 8 physical cores, with its TXL v10.6a-64 (14.7.13), gcc-4.8, cflow (GNU cflow) 1.4 installed. Any other configurations may have affect on the results of the replication of our experiments.

This website contains the source for muScalpel, muScalpel in binary form, and the data sets, including test suites, that underlie our experiments. To facilitate replicating our results, we have written a sequence of **scripts** that run a *single* run of each of our experiments. The name of the script identifies the experiment. We have worked hard to make each script bullet-proof and have it thoroughly check your environment for its dependencies and tell you what, if anything, is missing. Despite our best efforts, you may still encounter problems. If that happens, please contact us so we can work with you to resolve them.

Experiment Scripts

- Link to a script that runs all our experiments, as submitted to ISSTA 2015 artifact evaluation track. Here we also provide a dockerized version of our experiments.
  - **All experiments:** : Download

http://crest.cs.ucl.ac.uk/autotransplantation/MuScalpel.html

EPSRC

Genetic Improvement Frameworks

Gin https://github.com/gintool/gin

PyGGI https://github.com/coinse/pyggi

EPSRC

Gin 

An extensible and modifiable toolbox for GI experimentation

Automatically transforms, builds, and tests Java projects

Supports automated test-generation & source code profiling

Scales to large real-world projects

Alexander E. I. Brownlee, Justyna Petke, Brad Alexander, Earl T. Barr, Markus Wagner, David R. White: “Gin: genetic improvement research made easy”, Genetic and Evolutionary Computation Conference (GECCO 2019)

EPSRC

Gin 

Automated profiling to link tests with methods

Dynamic class loading and test case execution

Compilation & test execution performed entirely within memory to improve performance

Support for Maven & Gradle projects

Automated test case generation & runtime manipulation

Detailed reports on JUnit test results

EPSRC

Gin 

Changes at the line & statement level

Supports expression-level changes

Supports constrained edits to nodes of the same type

Changes: Delete, Replace, Copy, Swap, Move

Gin 

First results:

case study on Gson (with added GP implementation)

Gin fixed injected bugs

Gin found an edit that improved runtime

Justyna Petke, Alexander E. I. Brownlee: Software Improvement with Gin: A Case Study. Symposium on Search Based Software Engineering (SSBSE 2019)

EPSRC

PyGGI 

Python General Genetic Improvement Framework

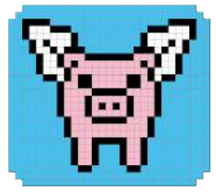
Supports line- & statement-level changes

Copy/delete/replace supported

Simple, short scripts needed to run

Gabin An, Aymeric Blot, Justyna Petke, Shin Yoo: "PyGGI 2.0: Language Independent Genetic Improvement Framework”, European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)
demo: https://www.youtube.com/watch?v=PxRUdlRDS40&feature=youtu.be

EPSRC

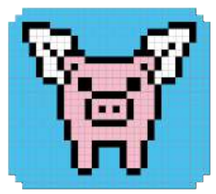
PyGGI 

XML tree representation

Supports multiple programming languages (inc. Python)

Build-in support for srcML (C++, C, C#, Java)

EPSRC

PyGGI 

Bug fixes found

Previous results on runtime improvement replicated

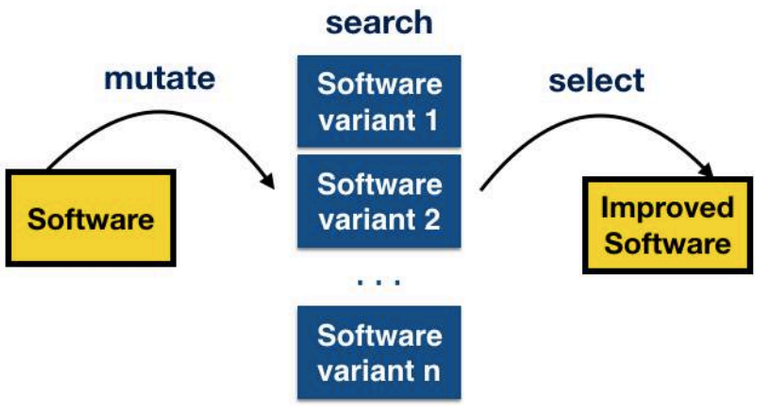
Gabin An, Aymeric Blot, Justyna Petke, Shin Yoo: "PyGGI 2.0: Language Independent Genetic Improvement Framework”, European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)
demo: https://www.youtube.com/watch?v=PxRUdlRDS40&feature=youtu.be

Genetic Improvement of Software

EPSRC

Genetic Improvement

mutates software to find an improved software version

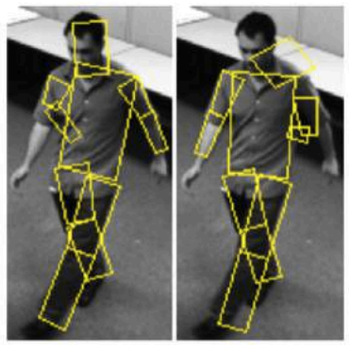


Justyna Petke, UCL

EPSRC

Reducing Energy Consumption Using Genetic Improvement

original picture produced by body-tracking software



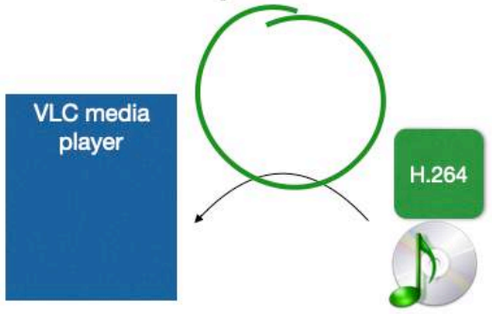
picture produced by software variant with 33% energy consumption reduction

Bobby R. Bruce, Justyna Petke, Mark Harman, and Earl T. Barr "Approximate Oracles and Synergy in Software Energy Search Spaces", IEEE Transactions on Software Engineering (TSE 2017)

Justyna Petke, UCL

EPSRC

Automated Software Transplantation



Transplanted the H.264 video encoding functionality in 26 h of computation time
(compared with estimated 20 days of human effort)

Media icon by Oxygen Team https://iconarchive.com/show/oxygen-icons-by-oxygen-icons.org.html
License: GNU Lesser General Public License

Justyna Petke, UCL

EPSRC

Genetic Improvement Frameworks

Gin https://github.com/gintool/gin

PyGGI https://github.com/coinse/pyggi

Justyna Petke, UCL

Justyna Petke, UCL