Embedded Dynamic Improvement

Nathan Burles[1]
Jerry Swan[1]
Edward Bowles[1]
Alexander E. I. Brownlee[2]
Zoltan A. Kocsis[2]
Nadarajen Veerapen[2]
1. Department of Computer Science, University of York
2. Computing Science and Mathematics, University of Stirling

The DAASE Project

From `http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=EP/J017515/1`

> *[DAASE] places computational search at the heart of the processes and products it creates and embeds adaptivity into both.*

Well-known that maintenance dominates software lifecycle cost[1].
So a key part of the DAASE manifesto is the creation of *online adaptive* systems, i.e. the improvement process can take place within a system *as it runs*.

[1] Le Goues, Forrest, and Weimer 2013.

Genetic Improvement (GI)

GI can be used to:

- Obtain a multi-objective trade-off between Non-Functional Properties[2] (NFPs).
- Fix bugs[3].
- Optimize/improve functional properties.

Since 'Software is its own substrate'[4], we need only provide a *fitness function*.

[2] Harman, Langdon, et al. 2012.
[3] Le Goues, Forrest, and Weimer 2013.
[4] Harman, McMinn, et al. 2011.

Compilation and Variation

- GI creates one or more *variants* of the seed software and compiles/interprets the variant.
- There's no requirement for the *variation environment* and the *compilation environment* to be the same[5].
- However, an *integrated* environment means that we can be more informed about:
  - Variable scope
  - Types and function signatures
  - Available type conversions etc.

[5] e.g. they are different in Langdon and Harman 2013

Embedded Dynamic Improvement

Here are some ways in which variants can be generated and invoked *within a running program*:

- Create an embeddable wrapper for a GP system which then creates the variants (e.g. TEMPLAR[6]).
- Use *reflection* to manipulate and compile ASTs at runtime (e.g. GEN-O-FIX[7], ANTBOX[8]).

The above systems perform both **GI** *and* **GP**, rather than 'plastic surgery'.

Fitness measures for online-measurable NFPs (e.g. power consumption) can be supplied on a per-platform basis.

[6] Swan and Burles 2015.
[7] Swan, Epitropakis, and Woodward 2014.
[8] Kocsis and Swan 2015.

Taxonomy

Table: Characteristics of GP, Offline GI, and EDI

Characteristic	GP	Offline GI	EDI
Executable Output	Expression tree	Patch for existing system	Updated system state
Language Output	Expression tree	Source/object code	Source/object code[9]
Expressiveness	Arbitrary expressions	Defined by patterns	Arbitrary expressions possible
In situ	No	No	Yes
Scale	Single expression	Entire system	Developer-specified
Processing	Offline	Offline	Online
Execution Frequency	Once	Once	Once or periodically
Demarcation	Researcher-defined	Tool-defined	Developer-defined

[9] Not currently supported by embedded TEMPLAR

Applications

Systems which *train offline* may suffer from *concept drift*.
Online training is therefore particularly useful for:

- **Isolated or Embedded Systems**
  Can be superior to the centralized deployment of software updates, since each system can be optimized to its local environment.
- **Predictive Systems**
  The prediction algorithm can be tuned to be optimal for the particular distribution to which it is exposed.

References I

 Zoltan A. Kocsis and Jerry Swan. *AntBox - A Dependency Injection Container for embedded Automatic Improvement Programming*. Tech. rep. YCS-2015-497. Uni. of York, 2015.

 Jerry Swan and Nathan Burles. "Templar - A Framework for Template-Method Hyper-Heuristics". In: *Genetic Programming, LNCS 9025*. Ed. by Penousal Machado et al. 2015. ISBN: 978-3-319-16500-4.

 Jerry Swan, Michael G. Epitropakis, and John R. Woodward. *Gen-O-Fix: An embeddable framework for Dynamic Adaptive Genetic Improvement Programming*. Tech. rep. CSM-195. Uni. of Stirling, 2014, pp. 1–12.

References II

 William B. Langdon and Mark Harman. “Optimising Existing Software with Genetic Programming”. In: *IEEE TEC* (2013).

 Claire Le Goues, Stephanie Forrest, and Westley Weimer. “Current Challenges in Automatic Software Repair”. In: *Software Quality Journal* 21 (3 2013), pp. 421–443.

 Mark Harman, William B. Langdon, Yue Jia, David Robert White, Andrea Arcuri, and John A. Clark. “The GISMOE challenge: constructing the Pareto program surface using genetic programming to find better programs.” In: *ASE*. Ed. by Michael Goedicke, Tim Menzies, and Motoshi Saeki. ACM, 2012, pp. 1–14. ISBN: 978-1-4503-1204-2.

References III

Mark Harman, Phil McMinn, Jerffeson Teixeira de Souza, and Shin Yoo. “Search-Based Software Engineering: Techniques, Taxonomy, Tutorial”. In: *Empirical Software Engineering and Verification*. Ed. by Bertrand Meyer and Martin Nordio. Vol. 7007. Lecture Notes in Computer Science. Springer, 2011, pp. 1–59. DOI: 10.1007/978-3-642-25231-0_1.