Stack-Based Genetic Improvement

Aymeric Blot    Justyna Petke

University College London, UK
UK EPSRC grant EP/P023991/1

GI@ICSE — 3 July 2020

RIP Larry Tesler 1945–2020: inventor of cut/copy & paste (and more)
In a Nutshell

Solution representation in GI:
- Software itself
- Diff patch
- Sequence of edits

Current GI edits:
- Delete(l)
- Replace(l1, l2)
- Insert(l1, l2) (x2)
- ...

Proposed GI edits:
- Cut(l)
- Copy(l)
- Paste(l) (x3)
- ...

Petke et al., IEEE Transactions on Evolutionary Computation, 2018 (literature review)
Why? Are Current Edits Not Good Enough?

Advantages:
- Focus on the changes only
- Easy creation/mutation/crossover
- Close to human understanding

Limitations:
- Complex high granularity recombination
- Type constraints

Oliveira et al., Empirical Software Engineering, 2018 (decoupled representation)
High Granularity Recombination

Example: One Point Across All Subspaces

- Issue: invalid, incomplete genes
- Solution: individual caches

Oliveira et al., Empirical Software Engineering, 2018 (high granularity crossovers)
Ensuring “Type” Validity

Consistency is important!

- Replace([statement], [statement]) will work
- Replace([condition], [condition]) will work
- Replace([condition], [statement]) will fail horribly

Possible solutions?

- Disable high granularity recombination
- Multiple decoupled sub-representations
- Any other complex bespoke mechanism
Equivalent Stack-Based Edits

Initial state: $\text{Cut}(1) \text{ Copy}(2) \text{ Paste}(3) \text{ Paste}(4) \text{ Copy}(5)$
$\leadsto$ empty patch + empty stack: $[]$

Cut: $\text{Cut}(1) \text{ Copy}(2) \text{ Paste}(3) \text{ Paste}(4) \text{ Copy}(5)$
$\leadsto$ $\text{Delete}(1)$ + stack: $[1]$

Copy: $\text{Cut}(1) \text{ Copy}(2) \text{ Paste}(3) \text{ Paste}(4) \text{ Copy}(5)$
$\leadsto$ $\text{Delete}(1)$ + stack: $[1, 2]$

Paste: $\text{Cut}(1) \text{ Copy}(2) \text{ Paste}(3) \text{ Paste}(4) \text{ Copy}(5)$
$\leadsto$ $\text{Delete}(1)$ $\text{Replace}(3, 2)$ + stack: $[1]$

Final patch: $\text{Cut}(1) \text{ Copy}(2) \text{ Paste}(3) \text{ Paste}(4) \text{ Copy}(5)$
$\rightarrow$ $\text{Delete}(1)$ $\text{Replace}(3, 2)$ $\text{Replace}(4, 1)$ + discarded stack
“It Just Works”™

Insertion?
- replace = paste in place
- insert before = paste before
- insert after = paste after

High granularity recombination?
- Simple “non-decoupled” crossover
- Full decoupling with Target(1) (x3), Copy(1), Cut(1), Paste(1)

Type validity?
- Pop and push to type-specific stacks

Spector and Robinson, Genetic Programming Evolvable Machines, 2002 (Push language)
Conclusion

Idea:

- Replacement set of edits
- Equivalent, backward compatible

Advantages:

- Same features but simpler
- Built-in memorisation mechanism
- Automatic type separation (multiple stacks)
Selected References

Vinicius Paulo L. Oliveira, Eduardo Faria de Souza, Claire Le Goues, and Celso G. Camilo-Junior.
Improved representation and genetic operators for linear genetic programming for automated program repair.

Genetic improvement of software: A comprehensive survey.

Lee Spector and Alan J. Robinson.
Genetic programming and autoconstructive evolution with the push programming language.