Genetic Improvement using Grammars

Michal Wiglasz

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole

iwiglasz@fit.vutbr.cz

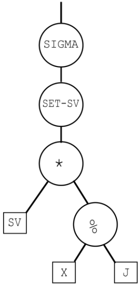
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

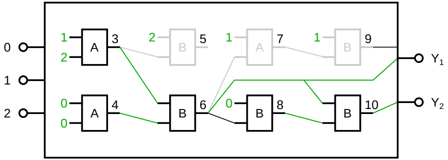
December 17, 2015

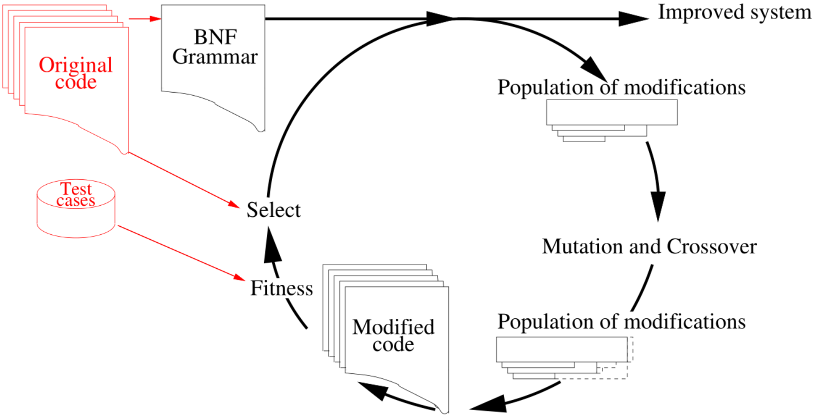
Genetic improvement

- Process of automatically improving a system's behaviour using genetic programming (GP).
- Typically improvement of non-functional properties:
  - Execution time
  - Power consumption
  - Memory footprint
- But it can also improve functional properties.
- Focus on automatization of the improvement process – **evolutionary process**.

- Evolution of computer programs.
- John Koza (1992) – LISP expressions (tree-based).
- Many other representations:
  - Machine code
  - Graph-based (e.g. Cartesian GP)
  - Grammar-based (e.g. grammatical evolution)






Original code
BNF Grammar
Improved system
Population of modifications
Test cases
Select
Mutation and Crossover
Fitness
Modified code
Population of modifications

Grammar construction

```
args.push_back(string(argv[0]));
```

⟨*sample.c_89*⟩ ::= ⟨*_sample.c_89*⟩

⟨*_sample.c_89*⟩ ::= ‘`args.push_back(string(argv[0]));`’

Grammar construction

```c
int counter = 0;
```

⟨*sample.c_112*⟩ ::= `int counter = 0;'

Grammar construction

```
while (in.getline(buffer, 4095)) {
```

$$\langle \textit{sample.c\_47} \rangle ::= \text{`}\texttt{while }\text{'} \langle \textit{WHILE\_sample.c\_47} \rangle \text{`}\texttt{ \{\textbackslash n}\text{'}$$

$$\langle \textit{WHILE\_sample.c\_47} \rangle ::= \text{`}\texttt{(in.getline(buffer, 4095))}\text{'}$$

Grammar construction

```
for (i = 0; i < counter; i++)
```

⟨*sample.c_214*⟩ ::= `for ` ⟨*FOR1_sample.c_214*⟩ ⟨*FOR2_sample.c_214*⟩ ⟨*FOR3_sample.c_214*⟩ ` {\n`

⟨*FOR1_sample.c_214*⟩ ::= `(i = 0;`

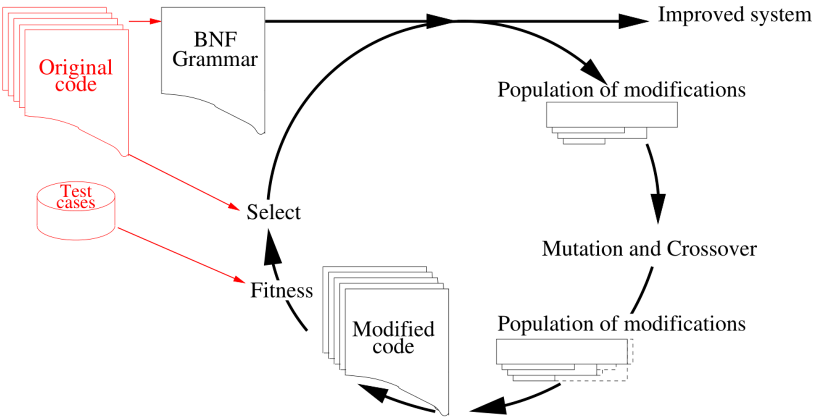
⟨*FOR2_sample.c_214*⟩ ::= `i < counter;`

⟨*FOR3_sample.c_214*⟩ ::= `i++)`

Grammar construction – example

⟨*start*⟩	::=	⟨*main_1*⟩ ⟨*main_2*⟩ ⟨*main_3*⟩ ⟨*main_4*⟩ ⟨*main_5*⟩ ⟨*main_6*⟩ ⟨*main_7*⟩ ⟨*main_8*⟩ ...
⟨*main_1*⟩	::=	`` `int main(int argc, char **argv) {\n' ``
⟨*main_2*⟩	::=	`` `if ' `` ⟨*IF_main_2*⟩ `` ` {\n' ``
⟨*IF_main_2*⟩	::=	`` `(argc > 2 && !strcmp(argv[1], "-A"))' ``
⟨*main_3*⟩	::=	`` `const char *file = argv[2];' ``
⟨*main_4*⟩	::=	`` `ifstream in;' ``
⟨*main_5*⟩	::=	⟨*_main_5*⟩
⟨*_main_5*⟩	::=	`` `in.open(file);' ``
⟨*main_6*⟩	::=	`` `char buf[4096];' ``
⟨*main_7*⟩	::=	`` `int lastret = -1;' ``
⟨*main_8*⟩	::=	`` `while ' `` ⟨*WHILE_main_8*⟩ `` ` {\n' ``
⟨*WHILE_main_8*⟩	::=	`` `(in.getline(bug, 4095))' ``

(...)


Original code
BNF Grammar
Improved system
Population of modifications
Test cases
Select
Mutation and Crossover
Fitness
Modified code
Population of modifications

Evolving improved programs

- GP can add, remove or change lines of code.
- No new code is generated – only existing code is used.
- GP can exchange rules of the same type:
  - single-line statements – `<_*>`
  - `if` conditions – `<IF_*>`
  - `while` conditions – `<WHILE_*>`
  - `for` loop initialization parts – `<FOR1_*>`
  - `for` loop conditions – `<FOR2_*>`
  - `for` loop increment parts – `<FOR3_*>`

Evolving improved programs

- It is impractical to evolve the whole program.
- GP individual = ordered list of changes made to the grammar, for example:

```
<FOR3_sample.c_11><FOR3_sample.c_66>  # replace
<_sample.c_74>                        # remove line
<_sample.c_84>+<_sample.c_14>         # insert line
```

Mutation operator

Mutation appends a new grammar modification to the list:

```
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_74>
<_sample.c_84>+<_sample.c_14>
```

↓

```
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_74>
<_sample.c_84>+<_sample.c_14>
<FOR2_sample.c_11><FOR2_sample.c_147>
```

Crossover operator

Crossover concatenates two individuals:

```
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_84>+<_sample.c_14>
```

```
<_sample.c_74>
<WHILE_sample.c_77><WHILE_sample.c_124>
```

```
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_84>+<_sample.c_14>
<_sample.c_74>
<WHILE_sample.c_77><WHILE_sample.c_124>
```

- After a mutation and crossover, a genotype can include several changes to the same line.
- Only the last change is relevant – the rest is removed.

```
<_sample.c_84><_sample.c_14>
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_84><_sample.c_65>
<_sample.c_84><_sample.c_11>
```

```
<FOR3_sample.c_11><FOR3_sample.c_66>
<_sample.c_84><_sample.c_11>
```

Parent selection

- Up to half of the population become parents.
- If necessary, children are generated randomly.
- Programs which fail to compile cannot be selected .

Not all generated programs are valid

- Representation ensures no parse errors can occur.
- But there are other compile errors, often caused by variables out of scope.
- Partial solution:
    - Restrict moves to be within the same source file.

Not all generated programs are valid

- Modifications of `while` and `for` conditions might create infinite loops.
- Halting problem.
- Possible solutions:
  - Extract fitness from running program – no need to wait for termination.
  - Limit execution time and abort slow programs.

How to test modified programs?

- Sometimes the task specification is not available.
- We can always use the original program as an oracle, but:
  - bugs may be replicated in improved versions,
  - may not be reliable for every possible input.
- It is advisable to also seek other oracles.

Bowtie2

- GI used to improve speed of Bowtie2.
- Tool for aligning DNA sequencing reads to long reference sequences.
- 50 000 lines of C++ code
- The search focused only on about 3000 lines.
- The result had only 7 changes in 3 source files.
- More than 70 times faster with slightly improved results.

Langdon, W., Harman, M.: Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation* 19(1), 118–135 (Feb 2015). ISSN 1089-778X.

Thank You For Your Attention!

References

Miller, Julian F. *Cartesian genetic programming*. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-17309-7.

Langdon, W., Harman, M., Jia Y.: Efficient multi-objective higher order mutation testing with genetic programming. *Journal of systems and Software* 83(12), 2416-2430 (Dec 2010). ISSN 0164-1212.

Langdon, W., Harman, M.: Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation* 19(1), 118–135 (Feb 2015). ISSN 1089-778X.