Using Genetic Improvement to Optimise Optimisation Algorithm Implementations

Aymeric Blot    Justyna Petke

University College London

ROADEF 2022 (25 February 2022)

http://www0.cs.ucl.ac.uk/staff/a.blot/publis#blot:2022:roadef
Automated Software Improvement

Software synthesis:

\[ \min_{s \in S} f(s, T) \]

With:
- \( s \) a software
- \( S \) the set of all software
- \( f \) the fitness function
- \( T \) the software specification

Genetic improvement:

\[ \min_{p(s_0) \in S} f(p(s_0), T) \]

With:
- \( s_0 \) a given software
- \( p(s_0) \) a patched version of \( s_0 \)

Hypothesis:
- \( s_0 \) is already very good
Genetic Improvement (GI)

**Applications:**
- Functional properties
  - Program repair / bug fixing
  - Feature transplantation
- Non-functional properties
  - Execution time
  - Energy / memory usage
  - Solution quality

**As an optimisation problem:**
- Very expensive
  - Compilation time
  - Fitness uncertainty
  - Fitness approximation
- Inconvenient search space
  - Huge neighbourhoods
  - Deceiving plateaus
  - *Fractal* nature

**Motivation:**

*Evolve software (source code) to improve performance*

---

Petke et al., IEEE Transactions on Evolutionary Computation, 2018
Source Code Representation

Example C++ code:

```cpp
... if (j > i) {
    x = j;
} ...
```

Example XML code:

```xml
...<stmt>if <condition>(j &gt; i)</condition> <block>{
    <stmt> x = j;</stmt>
} </block></stmt> ...
```

Software evolution:

- Convert source code to XML (SrcML)
- Focus on selected tags
- Mutate the AST
- Scrub XML tags
Genetic Improvement (GI)

In a nutshell:
- Start from original software
- Create software mutations
- Apply, recompile, evaluate, accept
- Accumulate sequences of edits
- Show final patch

Software edits:
- Statement deletion
- Statement insertion
- Statement replacement
- Data structure replacement
- Literal mutation
Case Study

**Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II (TEVC 2009)**

- Simple C++ implementation
- Nine hardcoded “complicated” problems
- Inverted generational distance (IGD)

**Selected files:**

- DMOEA/dmoeafunc.h.xml
- NSGA2/nsga2func.h.xml
- common/recombination.h.xml

Li and Zhang, IEEE Transactions on Evolutionary Computation, 2009
Experimental Setup

Simple local search:

- First improvement
- Mutation:
  - 50% create/append edit
  - 50% delete edit
- Fitness:
  - CPU instructions (perf)
  - Reject if solution quality > 110%
- Budget:
  - Wallclock time
  - ≈ 1000 evaluations
Experimental Protocol

Training: To find improved software variants
  ▶ Using the search process (local search)
  ▶ Until budget exhaustion (≈ 3 hours 45 minutes)
  ▶ Three runs on one problem

Validation: To avoid overfitting
  ▶ Filter out potentially harmful mutations
  ▶ Three runs on one unseen problem

Test: To assess generalisation
  ▶ Three runs on one (new) unseen problem

Sanity check:
  ▶ Three runs on all nine problems
Cross-validation \((k = 5)\)

Data is separated into \(k\) disjoint “folds”
Then labelled in \(k\) different ways:

Test: \((X)\)
- Single fold
- Sequentially

Validation: \((V)\)
- Single fold
- Uniform at random

Training: \((T)\)
- \(k - 2\) folds
- All remaining
Results

MOEA/D

CPU instructions

NSGA-II

Training Validation Test All instances

131.6% 267.4% 231.7% 133.7% 100% 100%

474.5% 2925.5% 14912.7%

474.5% 2925.5% 14912.7%

329% 116.6%

127.4% 106.6% 116.6%

110.2% 104.4% 110.2%

150.7%
Results

MOEA/D

Observations

- Consistent −7 to −12% improvement
- Major speedups (up to −60%) fail to generalise
- Various negative impact on solution quality
Removing IGD computation: (−12% execution time at validation)

```cpp
+++ after: DMOEA/dmoeafunc.h
void CMOEAD::calc_distance() {
  distance = 0;
  for(int i=0; i<ps.size(); i++) {
    double min_d = 1.0e+10;
    for(int j=0; j<population.size(); j++) {
      double d = dist_vector(ps[i].y.obj,
                              population[j].indiv.y.obj);
      if (d<min_d) min_d = d;
    }
    distance += min_d;
  }
  distance /= ps.size();
}
```
Patch Examples

Removing IGD computation: \((-12\% \text{ execution time at validation})\)

```c
+++ after: DMOEA/dmoeafunc.h
   // load the representative Pareto-optimal solutions
   sprintf(filename,"PF/pf_%s.dat",strTestInstance);
-   loadpfront(filename,ps);

+++ after: DMOEA/dmoeafunc.h
   // load the representative Pareto-optimal solutions
-   sprintf(filename,"PF/pf_%s.dat",strTestInstance);
   loadpfront(filename,ps);
```

Note:

► Final population was captured and externally reassessed
Hidden parameter tuning: (−48% execution time at validation)

```c
+++ after: DMOEA/dmoeafunc.h
   // mating selection based on probability
   if (rnd<realb) {type = 1;} // neighborhood
-   else {type = 2;} // whole population
+   else {} // whole population
```

Notes:

- Brackets added automatically thanks to SrcML
- realb = 0.9
- Failed to generalise on third problem (test)
Patch Examples

**New strategy:** (−27% execution time at validation)

```c
+++ after: DMOEA/dmoeafunc.h
    // produce a child solution
    CMOEADInd child;
    diff_evo_xover2(population[n].indiv,
                     population[p[0]].indiv,
                     population[p[1]].indiv,
                     child);
    +    type = 1;
    // apply polynomial mutation
    realmutation(child, 1.0/nvar);
```

**Notes:**

▶ type is used twice (matingselection(...) and update_problem(...))
▶ Insertion happens between both uses
▶ Fail to generalise on third problem (test)
Patch Examples

New strategy: (−9% execution time at validation)

```c
+++ after: NSGA2/nsga2func.h.xml
    bool flag = true;
    int size = offspring.size();
-   for (int i=0; i<size; i++) {
-       if (ind==offspring[i]) {
-           flag = false;
-           break;
-       }
-   }
+   nfes = 0;
    if(flag) offspring.push_back(ind);
```

Notes:

- Remove duplicity check (reset debug variable)
- Generalises, but worse fitness (+50%) during sanity check
Conclusion

Findings:
▶ “Free” 10% speedup
▶ Algorithmic changes
  ▶ Some “known”
  ▶ Some “new”
▶ Overfitting issues

What’s next?
▶ Better multi-objective setup
▶ New targets for edits
▶ Transplantation from optimisation frameworks
▶ Guidance process
Take Away

To err is human
- Practice ≠ theory
- Software bugs and defects

Automated performance improvement
- Compiler/parameter tuning
- Source code evolution (with GI)

Genetic improvement
- Evolution applied to software
- Functional properties
  - Bug fixing
  - Functionality transplantation
- Non-functional properties
  - Execution time
  - Solution quality
  - Energy/memory usage
Selected References

Aymeric Blot and Justyna Petke.  
Empirical comparison of search heuristics for genetic improvement of software.  

Hui Li and Qingfu Zhang.  
Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II.  

Genetic improvement of software: A comprehensive survey.  
Complicated Pareto Sets (MOEA/D)

Li and Zhang, IEEE Transactions on Evolutionary Computation, 2009