Automated Design of Algorithms and Genetic Improvement

Contrast and Commonalities

Saemundur Oskar Haraldsson, PhD student

University of Stirling

July, 2014

John R. Woodward  Co-author
Edmund K. Burke  Supervisor
Mark Harman  DAASE P.I.

http://daase.cs.ucl.ac.uk/
Overview

1. Motivation for Comparison
2. Automated Design of Algorithms
3. Genetic Improvement
4. Similarities & Differences
5. Summary
Researchers in either field should be aware of each others work because:

- Both fields are relatively young
- Driven by similar or same goals
- Equivalent methods
- Often identical objectives
Automated Design of Algorithms.

Methodology for using metaheuristic search methods to improve or discover algorithms for a given computational problem.

3 ways to adjust or alter an already existing algorithm

- **Replacing** components with an alternative.
- **Reordering** the application of the components
- **Parameter tuning**
Adjustment Examples

Component Replacement
- Choosing from a predetermined set of existing components.
  - Selection methods in evolutionary algorithms.
- Generate new replacement components.
  - Evolve mutation operators for genetic algorithms [10].
  - Generate selection heuristics for bin packing [1].

Component Reordering
- Special case of a permutation problem.
  - The order of evaluate, mutate, crossover and select in an evolutionary algorithm [7].
Genetic Improvement

Automatic software adjustment methods that operate directly on the source code, treating it as the genetic material.

- A sub-field of Search Based Software Engineering [2].

4 examples of application domains in current literature:

- Improving **non-functional** properties of existing software.
- **Migration** and **Transplantation**.
- Automatic **bug fixing**.
- **Dynamic Adaptive** software improvement [8].
Genetic Improvement, Examples

Non-functional Improvement

**Speed** is a popular non-functional property to improve.
- Easy evaluation and comparison.
- Used to speed up a 50k+ lines software system by a factor of 70 [5].

Other improved non-functional properties:
- Power consumption [9].
- HashCode uniformity [3].

**Migration & Transplantation** Repurposing software or a part of a software in an environment that it originally was not intended for.
- Migrating software to run on parallel processes [4].
- Transplanting functionality from one program to another [6].
Similarities

- The common general domain is **software**.
- Typically **evolutionary computational** methods.
- Evolve relatively **small** functions or patches.
- **Improve** rather than make from scratch.
- Often same **objectives**.
Automated Design of Algorithms

- Operate on function set and terminals (\textit{ex-situ}).
- Replaces or changes a function call.
- Constrained to where the call is made.
- Often creative, unexpected and/or counterintuitive functions.

Genetic Improvement

- Operate directly on source code (\textit{in-situ}).
- Multiple small changes separated by any number of lines.
- Unconstrained and global to the source code.
- Usually human readable edits.
Historical Differences

Automated Design of Algorithms
- Makes software find **better solution sooner** to increase speed.
- Applied to **algorithms** like heuristics, data mining and machine learning.
- Relatively easy to prevent faulty program variants.

Genetic Improvement
- **Turns off** non-vital functionality to increase speed.
- Applied to software **in general**.
- Often produces faulty program variants.
## Summary

<table>
<thead>
<tr>
<th>Automated Design of Algorithms</th>
<th>Genetic Improvement</th>
<th>ADA ~ GI</th>
<th>Similarities</th>
<th>Differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Examples of adjustment: Replace and Reorder</td>
<td>Operates directly on source code.</td>
<td>Researchers in either field should be aware of each others work.</td>
<td>Domain, methods, objectives, small output, and improve.</td>
<td>in/ex-situ, Un/constrained, human readable, objective approach, sub-domain and faulty variants.</td>
</tr>
</tbody>
</table>
Thanks

Any questions?


