The Role of Artificial Intelligence in Software Engineering

By: Mark Harman
Presented by: Jacob Lear

About the Author

- Mark Harman is a Professor of Software Engineering at University College London
- Director of CREST (Centre for Research on Evolution Search and Testing)
- Director for Research Council Funding
- Head of Software Systems Engineering Group

Brief History of AI

- Artificial Intelligence (AI) can trace its roots back to the seminal work of Turing and McCarthy.
- Alan M. Turing (1912-1954) wrote “Computing machinery and intelligence” in 1950 and proposed the “Turing Test”.

Introduction

- Science Fiction or Commonplace?
- “Computational intelligence regularly provides examples of specific areas of intelligent behavior for which machines comfortably surpass the performance of even the best humans.”
- Since the 1950's, AI has stimulated philosophical and technological debates, as well as interest and a little concern.
AI and SE

- AI is about making machines intelligent.
- Software Engineering (SE) is the activity of defining, designing, and deploying complex and challenging systems.
- SE is one of the most challenging of all engineering disciplines, but it is not recognized as such because software is generally well concealed.

Al and SE

- Not static fields of activity.
- More to come!
- During past 5 yrs. there have been important breakthroughs in AI.

SE Difficulty Example

- Consider a tiny smartphone, which may contain 5 – 10 MLOC.
- The failure of just about any line could lead to total system failure.
- The space of inputs to even the smallest app on the phone is likely to exceed $10^{80}$, yet only one of these inputs may reveal such a critical fault.

Possible Solutions

- Software engineers have one critical advantage over other engineers:
- Software engineer’s own material, software, and tools can be used to attack challenges posed by production of systems.
- AI algorithms are well suited to such complex SE problems because they can replicate intelligent behavior.
AI in SE

- SE community has used many algorithms, methods, and techniques that have emerged from the AI community in almost every area of SE activity.
- SE community has used 3 broad areas of AI techniques:
  - Search Based Software Engineering (SBSE)
  - Fuzzy and probabilistic methods for reasoning in the presence of uncertainty.
  - Classification, learning, and prediction.

When does AI for SE work well?

- 3 areas in which AI techniques are useful:
  - Probabilistic Software Engineering
  - Classification Learning and Prediction for Software Engineering
  - Search Based Software Engineering (SBSE)

Probabilistic Software Engineering

- Aim is to apply to SE, AI techniques developed to handle real world problems which are fuzzy and probabilistic.
- Natural fit because, increasingly, SE needs to accommodate fuzzy, ill-defined, noisy, and incomplete information.
- Processes by which software systems are built are often based on estimates.

Examples of Probabilistic Software Engineering

- Bayesian probabilistic reasoning used to model software reliability.
- Analysis of users requires probabilistic reasoning due to the stochastic nature of human behavior.
Classification Learning and Prediction for Software Engineering

- Great interest in modeling and predicting software costs during project planning.
- For example, a wide variety of machine learning techniques such as artificial neural networks, case based reasoning, and rule induction have been used for software project prediction, ontology learning, and defect prediction.

Search Based Software Engineering

- Goal is to re-formulate SE problems as optimization problems that can be attacked with computational search.
- Proven to be an applicable and successful approach, with applications from requirements and design to maintenance and testing.
- The “virtual character” of software makes it an engineering material ideally suited to computational search.

Relationship Between Approaches to AI for SE

- AI techniques for SE reveal considerable overlaps.
- All ways in which AI has been applied to SE can be thought of as ways to optimize either the engineering process or its products.
- The optimizations can usually be formulated as measurable objectives and constraints, the solutions to which reside in large spaces, making them ideal for computation search.

Machine Learning

- Machine learning – Essentially the study of approaches to computation that improve with use.
- To improve, need a way to measure improvement.
- With such a measure, can use SBSE to optimize according to it.
- SE situations typically have many possible measurements against which improvement can be sought.
Genetic Programming

- Genetic Programming – Widely used computational search technique in SBSE and also a machine learning approach.
- Algorithm for learning models of software behavior.
- Exciting recent breakthroughs in automatic bug fixing, porting between platforms, languages and programming paradigms, and trading functional and non-functional properties.

Improvement is Learning

- Close connections between machine learning for SE and SBSE.
- One way of learning is to optimize!
- Progress that takes place during optimization = learning process!

Steps to Application of AI in SE

1. Find a suitable formulation of the SE problem so that AI techniques become applicable.
2. This opens a technological window of opportunity through which many AI techniques may pass.
   - Example: SBSE can be used to optimize the performance of predictive models and case-based reasoners.

Challenges Ahead in AI for SE

Searching for strategies rather than instances

- Current approaches of AI to SE tend to focus on solving specific problem instances.
- Scope to move up from there to whole classes of problems, and from there, to the provision of strategies for finding solutions.
- Some work has been done on ways of searching for derived probability distributions for statistical testing, inferring strategies from paths in model checking, and on the search for tactics for program transformations.
- Focuses on specific problems!
<table>
<thead>
<tr>
<th>Challenges Ahead in AI for SE</th>
<th>Challenges Ahead in AI for SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching for strategies rather than instances</td>
<td>Exploitation of Multicore Computation</td>
</tr>
<tr>
<td>• How to migrate from searching for solution instances to searching for strategies for finding solutions?</td>
<td>• Many AI techniques used in SE are “embarrassingly parallel” – they naturally decompose into sub-computations that can be carried out in parallel.</td>
</tr>
<tr>
<td>• Search for strategies is a learning process over a training set.</td>
<td>• Parallelization has been exploited in software re-modularization, concept location, and regression testing.</td>
</tr>
<tr>
<td>• Exploit the natural connections between SBSE and machine learning!</td>
<td>• Principal challenge remains: finding ways to translate existing program paradigms into naturally parallelizable versions.</td>
</tr>
<tr>
<td>• Genetic Programming can generalize from solution of problem instances to solution of problem classes.</td>
<td></td>
</tr>
<tr>
<td>• It can characterize the strategies that find the next test input based in the behavior seen so far.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Challenges Ahead in AI for SE</th>
<th>Challenges Ahead in AI for SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploitation of Multicore Computation</td>
<td>Giving Insight to Software Engineers</td>
</tr>
<tr>
<td>• Without translation, execution on multicore systems is slower due to lower clock speeds than single-core CPU’s.</td>
<td>• AI techniques offer ways to yield insight into the nature of SE problems and the spaces in which their solutions reside.</td>
</tr>
<tr>
<td>• Many AI techniques for SE, and almost all for SBSE, are naturally parallelizable.</td>
<td>• Examples: SBSE used to reveal tradeoffs between requirements’ stakeholders and between requirements and their implementations and to bring aesthetic judgments into the software design process.</td>
</tr>
<tr>
<td>• More cores = more scalability!</td>
<td>• Many possible future ways in which AI techniques can be used to gain insight.</td>
</tr>
</tbody>
</table>
Challenges Ahead in AI for SE
Compiling Smart Optimization into Deployed Software

• Most work done so far on AI for SE has been applied off-line to improve the software process or the software itself.
• Why not compile the optimization process into the deployed software so it becomes dynamically adaptive?
• To do so, need to identify parameters that should be optimized, which could be formulated as an optimization problem.
• Work on genetic programming as a means of automatically patching, improving, and porting software may be developed to provide \textit{in situ} optimization.
• Could address long-standing challenges such as autonomic computing and self-adapting systems.

Challenges Ahead in AI for SE
Novel AI-Friendly Software Development and Deployment

• Need to adapt software development processes to better accommodate the increasing use of automated smart AI-inspired tools.
• Example: Release a policy to account for the fact that faults can automatically be fixed. Automated patches may not initially be as trusted as human-generated patches. May need to be used in tandem with original system for ongoing regression testing.

Challenges Ahead in AI for SE
Novel AI-Friendly Software Development and Deployment

• If deployed software takes advantage of dynamic optimization, \textit{in situ}, then this has implications on the design of the software with regards to how it obtains feedback from the user.

Optimizing the Performance of GNU-Chess with a Genetic Algorithm

By: Tomohiko Mitsuta and Lothar M. Schmitt
Presented by: Jacob Lear
Goal

• Use a genetic algorithm to find a chess program which is a slightly modified version of the original GNU-chess program but beats the latter by a measurable margin in an evaluation over several thousand games.

Approach

• Uses parallelization and distributed computing for calculating fitness scores.
• Uses the engineering perspective “learning from a mentor” as a reverse engineering method.
• Optimizes white and black separately from each other.
• Optimizes the coefficients of the evaluation function of the GNU-chess program using the original GNU-chess program itself as mentor.

Genetic Algorithms

• Introduced in the 1960’s.
• Are stochastic search algorithms based on the mechanisms of natural selection and natural genetics.
• Are particularly useful in applications for problem instances with irregular shape of the underlying function for which an optimum is sought.

Mutation

• Mutation Operation – A model of mutation in nature, i.e. random change of genetic information.
• There is a probability for selection for mutation AND a probability that the mutation will be applied to a particular gene.
• Selected genes undergo a change per the randomly selected, uniformly distributed stretching/shrinking factor.
**Crossover**

- Crossover Operation – Model of recombination of genetic material in nature, e.g. during the process of sexual reproduction.
- Creatures in the population are sequentially paired and crossover is applied to every pair with a probability.
- If selected, the lists of genes in the pair are cut at a single, random point and the tails of the two lists are swapped.

**Selection**

- Selection Operation – Model of natural selection: the genetic algorithm selects the individuals in the next generation based upon the evaluation of a given fitness function.
- Creatures with associated greater fitness-value have, in general, a greater probability of selection (survival) into the next generation.

**Fitness Evaluation**

- Fitness evaluation is performed by compiling each family member in the generation into a copy of GNU-chess and then executing a simulation of \( n \) games against the original GNU-chess program.
- The fitness score is determined by the combined result (1 point for win, 0.5 points for a draw) of these \( n \) simulations.

**Results**

- Population becomes quite uniform after about 50 generations, and fitness-gain seems to decrease significantly after 50 generations.
- Increased playing power of both white and black by a moderate amount over the original GNU-chess program.