Heuristic Search and Software Engineering

Joshua Charles Campbell

Department of Computing Science
University of Alberta

April 14, 2015

Me

- I am not an expert in search...
- I research software engineering
- My research seems to keep touching search

Overview

Search-Based Software Engineering

UA Research

Opportunities for Heuristic Search

Section 1

Search-Based Software Engineering

Trends...[7]

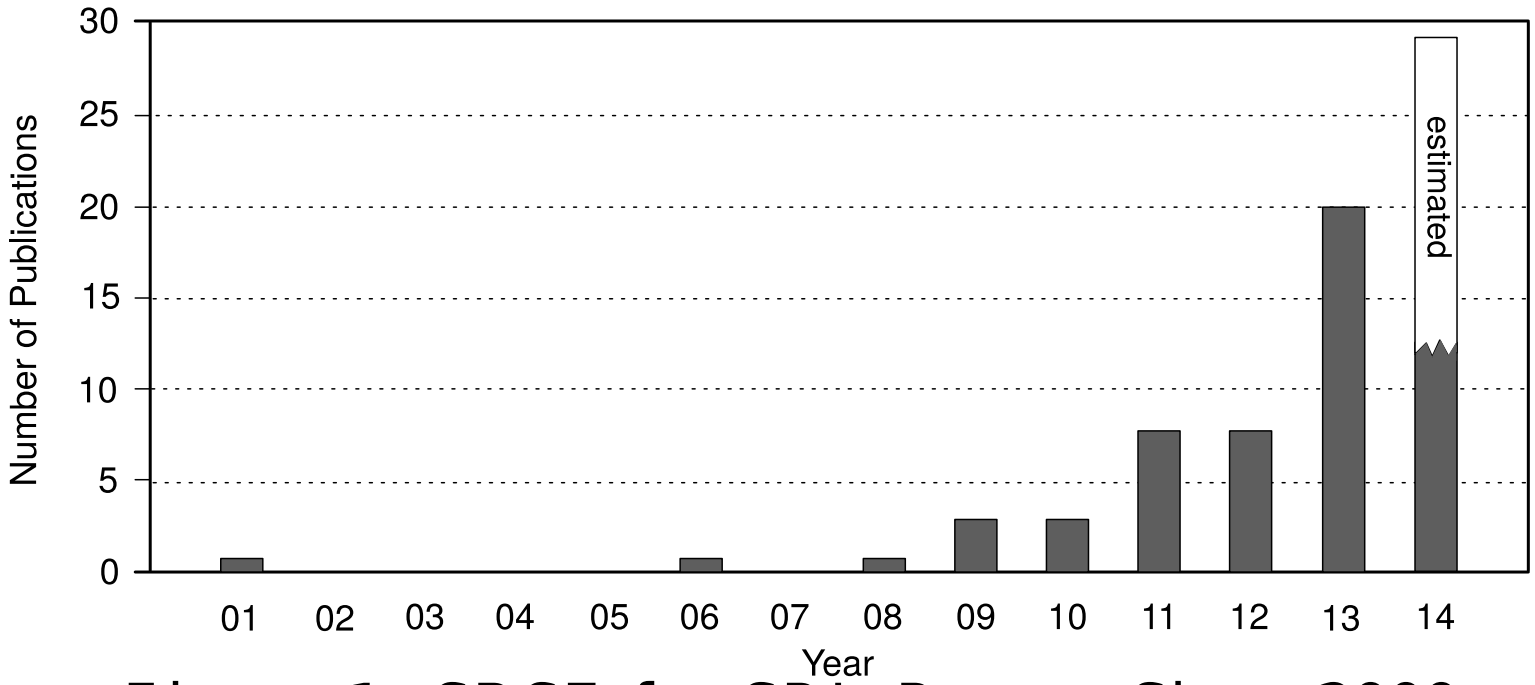

Figure 1: SBSE for SPL Papers Since 2000

- This presentation cannot be comprehensive!

Trends...[7]

- Product lines
  - e.g. Windows Ultimate, Windows Standard, Windows Pathetic
- Reuse, requirement trace, testing, maintenance

Arcuri, Fraser[1]

- Test suite generation
- Genetic algorithm
- Space: Program Inputs
- Goal: Full Coverage

Harman[8]

- Add new functionality given some starting points
- Gentic Programming
- Space: All type-safe programs
- Goal: Working software
- Starting point: API Calls provided, example plugin

Mkaouer, Kessentini, Bechikh[12]

- Refactoring (move, push down, inline, extract)
- Evolutionary Algorithm
- Space: Refactoring suggestions
- Goal: pareto-optimal quality x robustness
  - based on a underlying weighted code smell count

Mitchell, Mancoridis[11]

- Space: all possible graph partitions
- Goal: Find the "best" way to break a program into modules
- Objective: balance coupling and cohesion
  - ratio of internal to external edges
- Hill Climbing + Simulated Annealing

Zhang, Harman, Mansouri[14]

- What requirements to add for the next release?
- Cost of development / value of release tradeoff
  - Pareto front
- Non-dominated Sorting Genetic Algorithm II
- It's better than random search!

Arcuri, Yao[2]

- Test/code co-evolution “arms race”
- Space: Programs and test suites
- Goal: Correct program
  - Formal specification -> distance fn

Weimer, Nguyen, Goues, Forrest[13]

- Genetic programming
- Space: Programs
- Goal: Passes tests, minimal distance from original
- Starting point: broken program

Kim, Nam, Song, Kim[10]

- Like Weimer, et al. but... those patches are often bogus
- Looked at 60K human writen patches
- Derived fixed rules to guide the GP

Section 2

UA Research

Fault Location[4]

- Do you ever spend hours trying to find a bug...
- But it turns out to be something simple like a missing semicolon or a misspelled variable?

Fault Location[4]

- Large corpus of known-good code
- Code that's the least like known-good code
- Most likely to be bad!

Fault Location Example[4]

- Our program has an error
- The code might contain
  `for itim in collection:`
- If we've never seen the word `itim` before, we can guess that that is causing the error

Fault Location[4]

- Works as well as the compiler does!
- But orthogonally

Fault Location[4]

- What does that have to do with search?
- Nothing, yet...

Fault Location[4]

- Previous work with a similar technique
- Code completion (prediction)

Fault Location[4]

- Completion + fault location
- = code repair!
- Similar goal to previous work in SBSE

Charm[3]

- Estimate a metric that can't be measured directly
- "Charm"
- How much errors are attracted to any part of a program

Charm[3]

- Can't measure code directly
- Measure code near it and average
- Random sampling of neighborhood

Charm[3]

- “Near” -> Levenshtein (edit) distance
- “Average” based on a converged probability estimate after *n* trials
- Put in a box with a bow
  - Correct for variables (line length)

Working w/ Code

- Space of all programs is a super-high-dimensional space!
  - Basically GP...
- Dimensions $\sim$ length
- We can still obtain meaningful measures
  - Chunking
  - Sliding windows

Section 3

Opportunities for Heuristic Search

Planned Research: Code Repair

- Same setup as Weimer: Genetic Programming
- Space: Programs
- Goal: Passes tests, minimal distance from original
- Starting point: broken program

Code Repair

- We already have GP
- And some helpful rules for it
  - But these are labour-intensive and static
- Can we do this automatically and adaptively?
- Can we do this in real time?

Code Repair

- Naive approach: guess-and-test
- Use a language model to locate a fault
- Very narrow search around that location
- Try stuff we've seen people write before

Code Repair Example

- `for itim in collection:`
- itim is probably wrong because we've never seen it before in a working program
- try deleting itim? doesn't work...

Code Repair Example

- `for itim in collection:`
- try replacing itim? with what?
- Start with what we've seen before in the context

  `for ___ in collection:`
  - Start with the most often seen...

Code Repair Example

- `for ____ in collection:`
- Hey we've seen `item` in that context 8 times!
- `for item in collection:`
- Fixed!

Code Repair Q1

- Can we use a better search algorithm?
- GP is too slow! We want no more delay than would annoy a human.
- If we're lucky we'll get ~100 opportunities to "check" our answer...

Code Repair Q2

- Can we guide search based on what we've observed humans do?
- Tailor to each developer, team, project, etc.?
- Not just to make acceptable fixes
  - Make the fixes that I'd make!

Other Areas of SE

- Software Design
- Testing
- Verification
- Maintenance

Software Design[9]

- Ex: Modularisation
- GA, Simulated Annealing, Hill Climbing

Testing[9]

- Coverage, Optimisation
- GA, Simulated Annealing, Ant Colonies

Verification[9]

- Ex: Formal specifications, automated proofs
- Ant Colony

Maintenance[9]

- Refactoring
- GA, Hill Climbing

Section 4

Bibliography

Bibliography (1)


Andrea Arcuri and Gordon Fraser.
On the effectiveness of whole test suite generation.
In *Search-Based Software Engineering*, pages 1–15. Springer, 2014.


Andrea Arcuri and Xin Yao.
A novel co-evolutionary approach to automatic software bug fixing.
In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 162–168. IEEE, 2008.


Joshua Charles Campbell and Abram Hindle.
The charming code that error messages are talking about.

Bibliography (2)

Joshua Charles Campbell, Abram Hindle, and José Nelson Amaral.
Syntax errors just aren't natural: improving error reporting with language models.
In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 252–261. ACM, 2014.

Haitao Dan, Mark Harman, Jens Krinke, Lingbo Li, Alexandru Marginean, and Fan Wu.
Pidgin crasher: Searching for minimised crashing gui event sequences.
In *Search-Based Software Engineering*, pages 253–258. Springer, 2014.

Bibliography (3)

M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. Zhang.
Search based software engineering for software product line engineering: A survey and directions for future work.
In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 5–18, New York, NY, USA, 2014. ACM.

Mark Harman, Yue Jia, Jens Krinke, WB Langdon, Justyna Petke, and Yuanyuan Zhang.
Search based software engineering for software product line engineering: a survey and directions for future work.
In *Proceedings of the 18th International Software Product Line Conference-Volume 1*, pages 5–18. ACM, 2014.

Bibliography (4)

Mark Harman, Yue Jia, and William B Langdon.
Babel pidgin: Sbse can grow and graft entirely new functionality into a real world system.
In *Search-Based Software Engineering*, pages 247–252. Springer, 2014.

Mark Harman, Yue Jia, William B Langdon, Justyna Petke, Iman Hemati Moghadam, Shin Yoo, and Fan Wu.
Genetic improvement for adaptive software engineering (keynote).
In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 1–4. ACM, 2014.

Bibliography (5)

Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim.
Automatic patch generation learned from human-written patches.
In *Proceedings of the 2013 International Conference on Software Engineering*, pages 802–811. IEEE Press, 2013.

Brian S Mitchell and Spiros Mancoridis.
On the automatic modularization of software systems using the bunch tool.
Software Engineering, IEEE Transactions on, 32(3):193–208, 2006.

Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, and Mel Ó Cinnéide.
A robust multi-objective approach for software refactoring under uncertainty.
In *Search-Based Software Engineering*, pages 168–183. Springer, 2014.

Bibliography (6)

Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest.
Automatically finding patches using genetic programming.
In *Proceedings of the 31st International Conference on Software Engineering*, pages 364–374. IEEE Computer Society, 2009.

Yuanyuan Zhang, Mark Harman, and S Afshin Mansouri.
The multi-objective next release problem.
In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1129–1137. ACM, 2007.