Keeping DRY

A Comprehensive Measure of Modularity Within a Computer Program

J. Hassler Thurston

By Pete Souza (White House (P071412PS-0469)) [Public domain], via Wikimedia Commons

DRY vs WET

- DRY = “Don’t Repeat Yourself”
  - “Don’t do anything twice”
  - Makes code more efficient and readable
  - First formulated in “The Pragmatic Programmer” by Hunt and Thomas, 1999
- WET = opposite of DRY
  - “Write Everything Twice”
  - “We Like Typing”

Importance of DRYness

- Makes code more readable
- Separation of Concerns (modularity)
- Easier to modify code

Can we measure DRYness?

- Program “volume” as measure of modularity -- Baker et. al. (1979)
- Can measure similarity between two programs
  - Count similarity of operators/operands (lexical)
  - Count parse tree differences (structural)
  - Application: Plagiarism detection (esp. for college programming projects)
- Have not found any measure of DRYness in the literature
- How could we measure it?

DRYness Score

- Input = a Java program
- Output = a number between 0 and 1
  - 0 = completely DRY/completely modular
  - 1 = completely WET (soaked)
  - -1 = program does not compile

Difficulties

- Modularity/DRYness scale not well-defined
- Must revert to heuristic methods
- Cannot test our heuristics against a benchmark, because there is no benchmark
- Approach:
  - Write heuristics using Javaparser library (performs lexical analysis and parsing on Java files)
  - Evaluate them on sample Java files
  - See whether results conform to what should be expected from a DRYness heuristic

Heuristics -- Modified Baker Heuristic

- Adapted from Baker 1979
- Measure modified version of “program volume”
- V = 1 - n/N
  - n = number of distinct operators and operands
  - N = total number of operators and operands
  - Simplification: n = number of distinct *types* of operators and operands that appear
- “Comparison to previous work”

Heuristics -- Iteration 1 Heuristic

- Recursive algorithm
- DRYness score of leaves = 0 or 1
  - 1 for literal expressions (boolean, int, String values)
  - 0 for everything else
- DRYness score of node = average DRYness score of subtrees, EXCEPT ...

Heuristics -- Iteration 1 Heuristic

- DRYness of block statements
  - For every block statement (sequence of statements in a method, loop, …) ...
  - Count number of statements that are equivalent
  - Count total number of pairs of statements
  - return (# equivalent pairs) / (# total pairs)

Heuristics -- All Pairs Naive Heuristic

- More robust version of Iteration 1 Heuristic
- For each subtree of type T:
  - add subtree to ArrayList for all type T subtrees
- For each ArrayList of type T subtrees:
  - DRYness score of ArrayList =
    (# equivalent pairs) / (# total pairs)
- DRYness of program:
  - average DRYness of each ArrayList
- Meant to account for equivalent expressions across the whole Java program

Heuristics -- All Pairs Weighted Heuristic

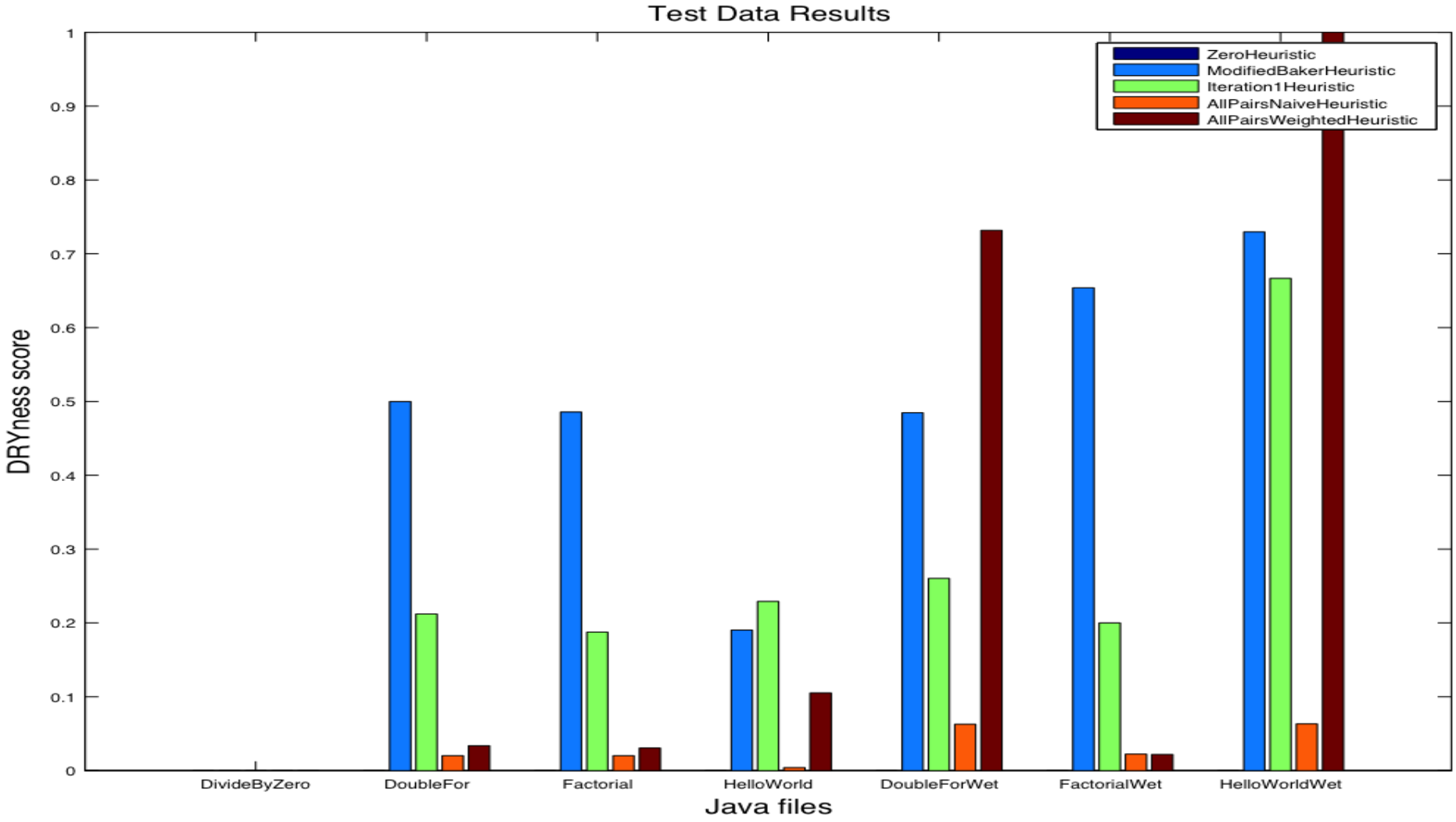
- Same as All Pairs Naive Heuristic, EXCEPT…
- DRYness of program = *weighted* average of DRYness score for each ArrayList
- Want equivalent Block Statements to have greater weight than equivalent one-line comments

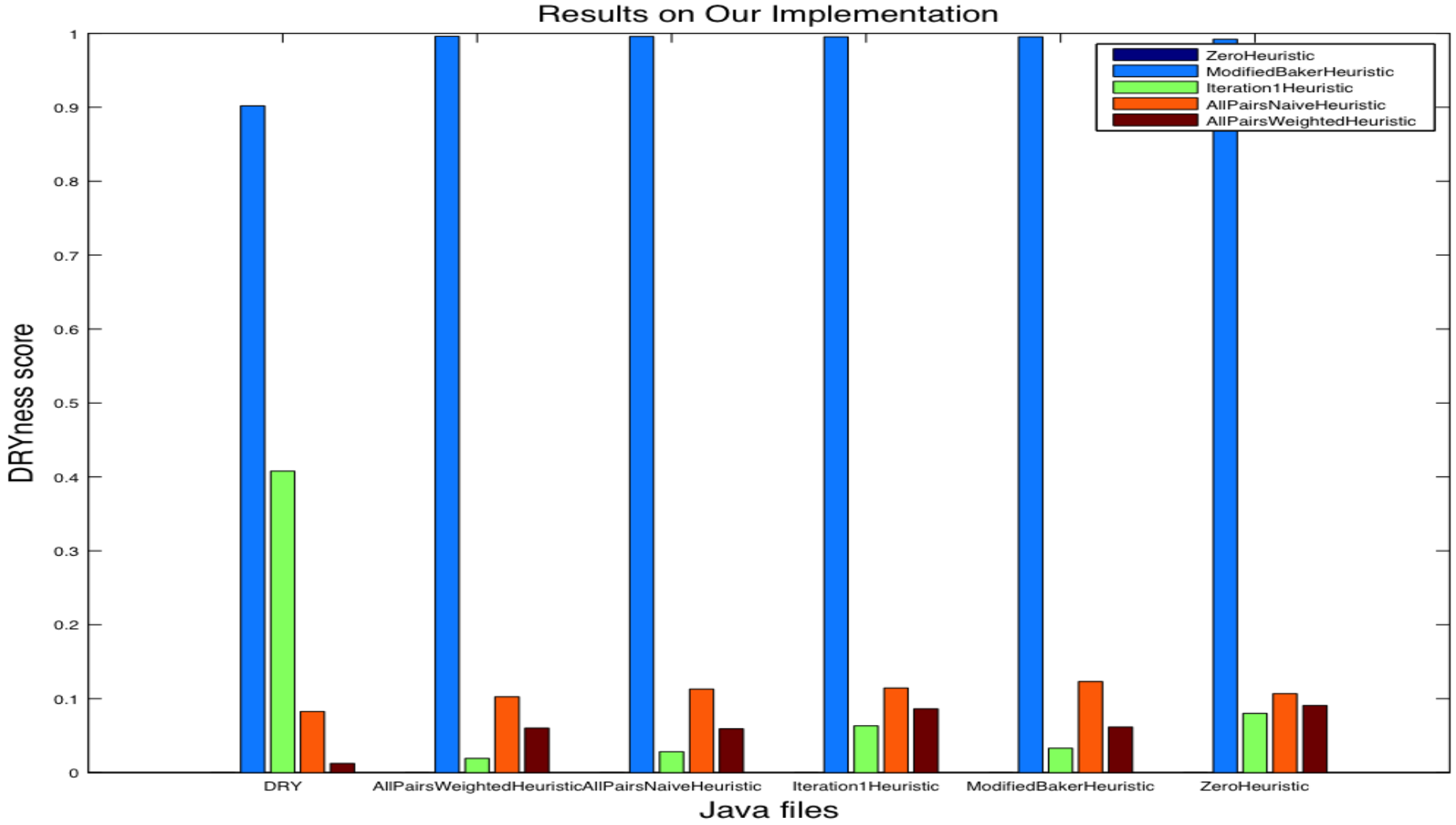
Test programs

- 7 Java programs
  - 1 empty class (helps check whether our heuristics are well-defined)
  - 3 pairs of Java classes
    - HelloWorld.java, HelloWorldWet.java
    - Factorial.java, FactorialWet.java
    - DoubleFor.java, DoubleForWet.java

Test programs

- Our own implementation!
  - Written in Java
  - Implementation files are much larger than the 7 test files
  - Is our own code itself modular?


Test Data Results
ZeroHeuristic
ModifiedBakerHeuristic
Iteration1Heuristic
AllPairsNaiveHeuristic
AllPairsWeightedHeuristic
DRYness score
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
DivideByZero
DoubleFor
Factorial
HelloWorld
DoubleForWet
FactorialWet
HelloWorldWet
Java files


Results on Our Implementation
ZeroHeuristic
ModifiedBakerHeuristic
Iteration1Heuristic
AllPairsNaiveHeuristic
AllPairsWeightedHeuristic
DRYness score
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
DRY
AllPairsWeightedHeuristic
AllPairsNaiveHeuristic
Iteration1Heuristic
ModifiedBakerHeuristic
ZeroHeuristic
Java files

Results -- Modified Baker Heuristic

- Gives significantly poorer ratings than other heuristics (especially larger files)
- Finite number of “types of operators and operands”, so larger files are unfairly rated
- Two possible conclusions:
  - Baker Heuristic needs further modifications to perform well
  - Baker Heuristic is not good for measuring DRYness

Results -- Iteration 1 Heuristic

- On 7 Test Files:
  - Gives poorer ratings than All Pairs heuristics
  - Cannot tell difference between DRY/WET versions of DoubleFor and Factorial programs
- On our implementation:
  - Gives better ratings than All Pairs Heuristics
  - Explanation: Only checks for equivalent expressions in Block Statements
- Conclusion: Does not perform optimally

Results -- All Pairs Heuristics

- Comparison:
  - On 7 Test Files: All Pairs Naive Heuristic gives better ratings
  - On our implementation: All Pairs Weighted Heuristic gives better ratings
- DRYness scores are generally low
- Naive heuristic does not detect significant difference between DRY/WET pairs
- Weighted heuristic gives better rating to FactorialWet than Factorial
- Conclusion: More work needed to make these more objective

Future -- Tree Edit Distance Heuristic

- Instead of measuring equality of subtrees, measure how much you have to change one for them to be equal
- Tree edit distance:
  - Input = two trees
  - Output = number of insertions, deletions, and modifications needed for two trees to be equal
- Difficulties:
  - Not well-defined for different types of nodes (e.g. hard to measure edit distance between an if statement and a comment)
  - Best runtime: $O(n^2)$ for each call (Pawlik and Augsten, 2011)

Conclusions

- Measuring DRYness currently unsolved, not objective
- Can measure DRYness heuristically using lexical and structural techniques
- We are the first to measure DRYness heuristically
- Heuristics can measure DRYness to a somewhat accurate extent
- All Pairs Weighted Heuristic seems to outperform other heuristics
- Can perform poorly on larger Java files

Applications

- Tool for amateur programmers to improve programming style
- Evaluating quality of open-source code
- Comparing various implementations of the same algorithm
- Evaluating student submissions of programming projects
- Future research on program similarity

Future Work

- Modifying Baker’s heuristic
- Tree Edit Distance Heuristic
- Larger sample size of Java files for testing
- Evaluating heuristics on student submissions and open-source code to evaluate how well heuristic methods can work in practice

References

[1] javaparser - java 1.5 parser and ast - google project hosting. `https://code.google.com/p/javaparser/`. Accessed: 2014-12-03.

[2] The wet cart - the daily wtf. `http://thedailywtf.com/articles/The-WET-Cart`. Accessed: 2014-12-02.

[3] A. L. Baker and S. H. Zweben. The use of software science in evaluating modularity concepts. *IEEE Transactions on Software Engineering*, SE-5(2):110–120, 1979.

[4] Brooks. No silver bullet essence and accidents of software engineering. *Computer*, 20(4):10–19, 1987.

[5] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, 1999.

[6] M. Joy, M. Joy, and M. Luck. Plagiarism in programming assignments. *IEEE Transactions on Education*, 42(2):129–133, 1999.

[7] Donald F. McGahn. Copyright infringement of protected computer software: An analytical method to determine substantial similarity. *Rutgers Computer and Technology Law Journal*, 21(1):88, 1995.

[8] Kevin A. Naud, Jean H. Greyling, and Dieter Vogts. Marking student programs using graph similarity. *Computers & Education*, 54(2):545–561, 2010.

[9] Mateusz Pawlik and Nikolaus Augsten. Rted: A robust algorithm for the tree edit distance. 2011.

[10] D. Ratiu, R. Marinescu, and J. Jurjens. The logical modularity of programs. pages 123–127, 2009.

[11] G. WHALE. Identification of program similarity in large populations. *COMPUTER JOURNAL*, 33(2):140–146, 1990.