

# Using Java Method Patterns to Predict Runtime Behaviour

Jeremy Singer

# Overview

- Machine Learning
- Features of Java Methods
- Thread-Level Speculation
- Results

Animal	# legs	# horns	texture	Will it kill me?
	0	0	smooth	yes
	4	1	rough	yes
	8	0	slimy	no

Animal	# legs	# horns	texture	Will it kill me?
 A centipede with many pairs of legs, orange and white, crawling on a dark, textured rock surface.	100	2	smooth	yes
 A black and white dolphin leaping out of the water, creating a splash.	0	0	smooth	no
 A grey koala clinging to a tree branch, looking towards the camera.	2	0	furry	no

# Some rules

**if (#horns > 0) then killer**

**if (#legs > 2) then killer**

*(problems with confidence)*

**if (texture == rough) then killer**

*(problems with support)*

# Applying rules 'in the wild'

- Having *trained* a predictor ...
- now I need to *test* it
- Enter wild animal ...

# Features for Java Methods

- Simple to detect
- Static (analyse source code)
- Binary
- Combine to give concise, abstract summaries
- Stolen some features from earlier work
- Added many new features
- *Fundamental nano-patterns*

# Method Call Patterns

NoParams	Takes no arguments
NoReturn	Returns <code>void</code>
Recursive	Calls itself recursively
SameName	Calls a different method with same name
Leaf	Does not issue any method calls

# Object-Oriented Patterns

ObjectCreator	Creates <b>new</b> objects
{Static,Instance} FieldReader	Reads field values from {class,object}
{Static,Instance} FieldWriter	Writes field values to {class,object}
TypeManipulator	Uses typecasts or <b>instanceof</b> operators

# Control Flow Patterns

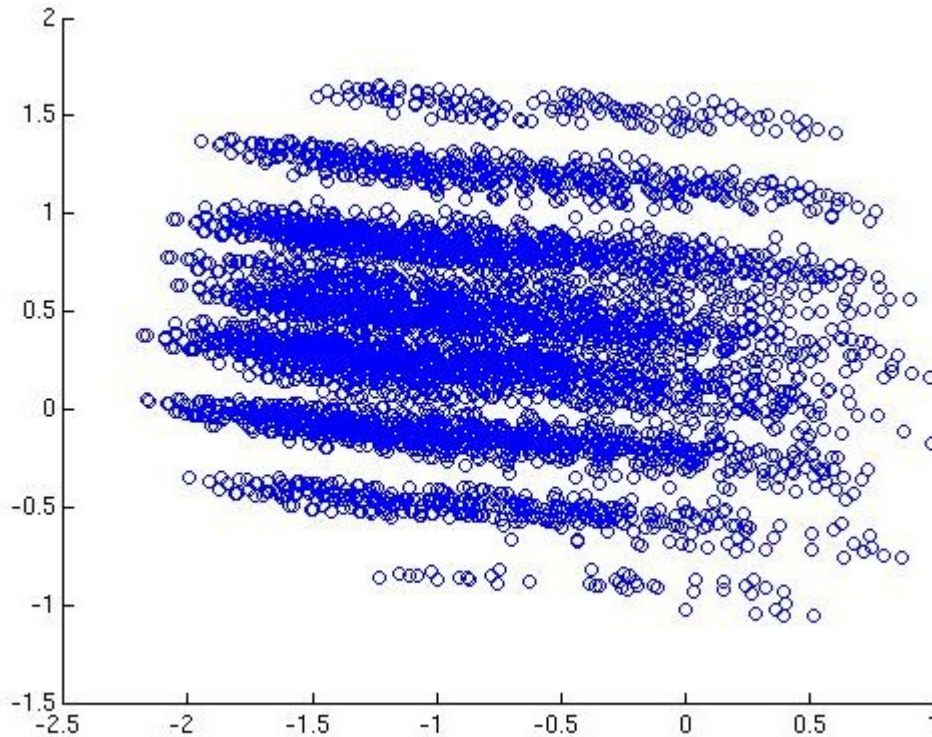
Straightline	No branches in method body
Looping	Control flow loops in method body
Exceptions	May throw an unhandled exception

# Data Flow Patterns

LocalReader	Reads local vars from stack frame
LocalWriter	Writes local vars on stack frame
ArrayCreator	Creates a new array in heap
ArrayReader	Reads values from an array
ArrayWriter	Writes values to an array

# Applications of Nano-Patterns

- Clustering similar Java methods



# Applications of Nano-Patterns

- Quantitative studies
  - Comparing different Java programs
  - Measuring evolution of single Java program
- Machine learning
  - Object lifetime prediction
  - *Squash prediction in thread-level speculation*

# Thread-Level Speculation

- Run code in parallel, even if we cannot guarantee correctness
- (Automatic rollback mechanisms to recover from mistakes)
- Speculative method-level parallelism
- At method calls, spawn speculative thread to run caller continuation in || with callee

# SMLP Execution



time

# Read-after-Write Dependence

```
void charlie() {  
    // do some work ...  
    luis();  
    // do some more work ...  
    read value of x;  
}
```

```
void luis() {  
    x = 42;  
}
```

# Squashing

- Undoes speculative execution
- Re-executes caller continuation non-speculatively
- Overhead for squashing
- Perhaps, only spawn when we are fairly sure that squashing is unlikely ...
- ***Squash prediction***

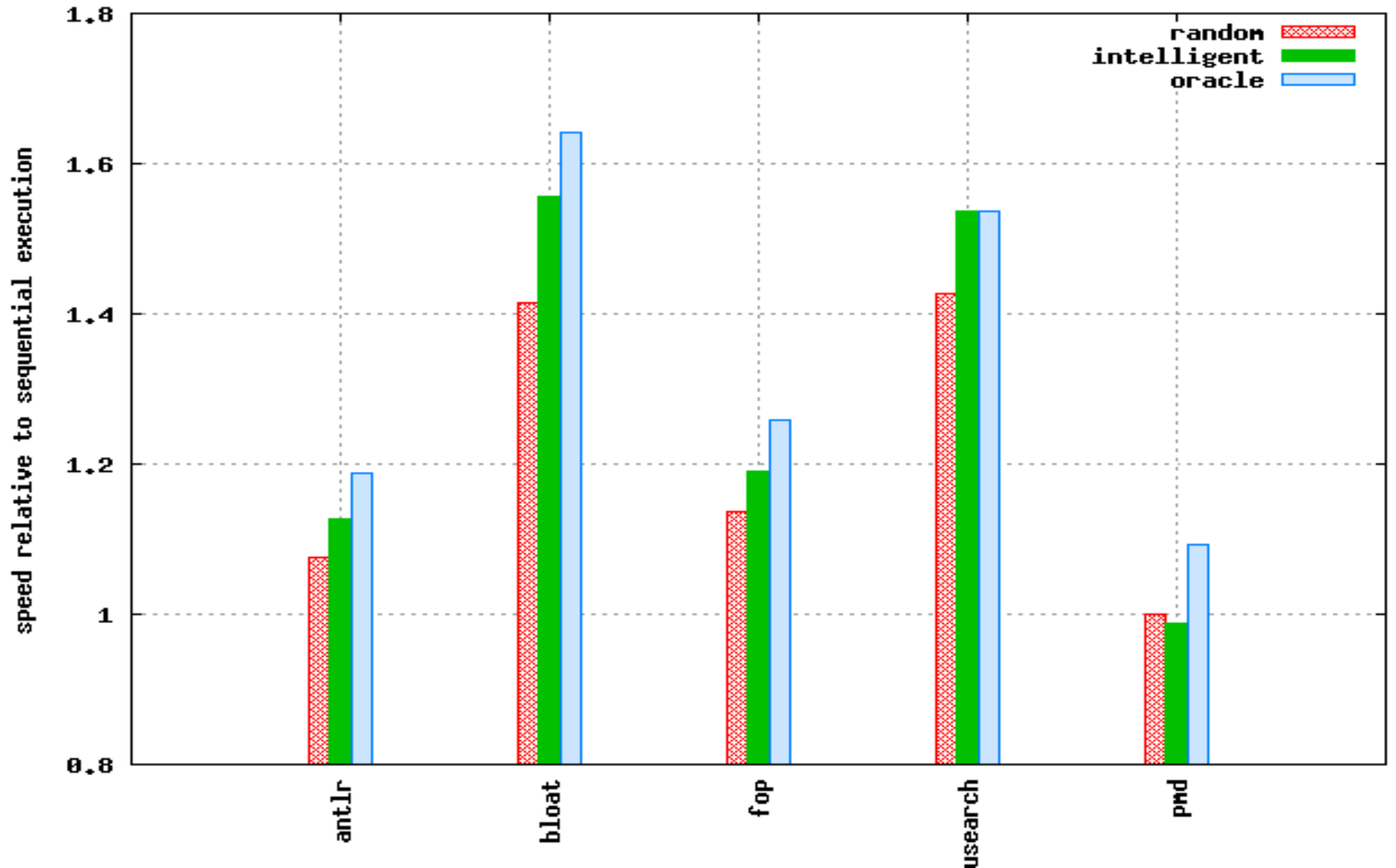
# Squash Prediction using Machine Learning

- Profile some Java applications
  - Spawn at all method calls
  - Determine which spawns commit / squash
- Relate caller / callee method features to squash behaviour
- Throw training data at learning algorithm
  - Association rule mining - CPAR

# Useful Features for Squash Prediction

- Loops in callee method
- \*writer in callee method
- Exceptions in caller method
  
- Rules generated for predicting squashes
  - ~100 for each training set
  - Train on 4 benchmarks, test on 5<sup>th</sup> (LOOCV)

# Speedups



# Conclusions

- Static code features are useful ...
  - To characterise programs for learning
  - Generalise from particular programs
  - In our squash prediction experiment, our rules-based predictor can achieve 80% of speedup due to an oracle predictor
- Future ideas
  - Better learning algorithms
  - More Java method features