Research in Experimental Computer Science

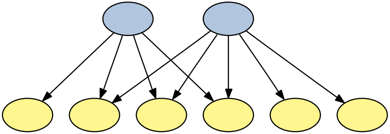
Suvarshi Bhadra, Alexander Conrad, Adam Smith, and Gregory M. Kapfhammer

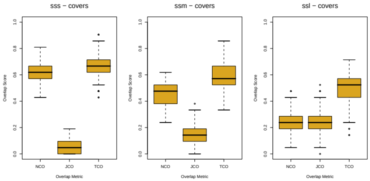
Department of Computer Science
Allegheny College, Pennsylvania, USA
http://www.cs.allegheny.edu/~gkapfham/

Allegheny College summeR reSearch Series (ACRoSS), July 2008

Featuring images from www.campusbicycle.com and www.pdclipart.org

What is Experimental Computer Science?



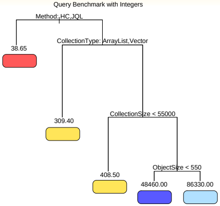


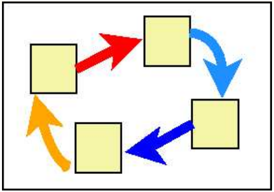
Solve Problems with Algorithms

Detailed Empirical Results

Implement and empirically **evaluate** the efficiency and effectiveness of **algorithms** that solve real-world **problems**

What is Experimental Computer Science?





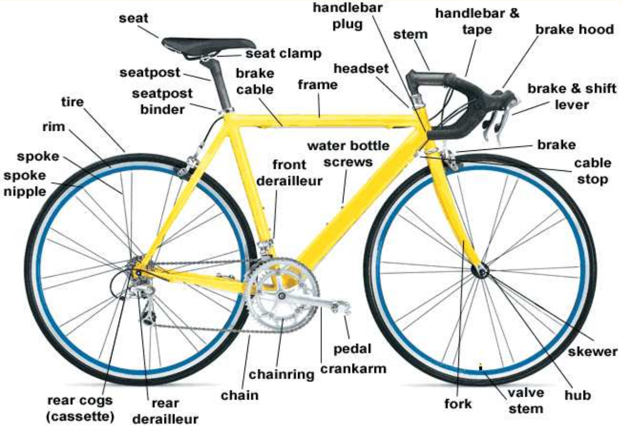
Statistical Analysis Techniques

Working Computational Artifacts

After analyzing **gigabytes** of data, **publish** results and **release** software **tools** that are useful to **academics** and **industrialists**

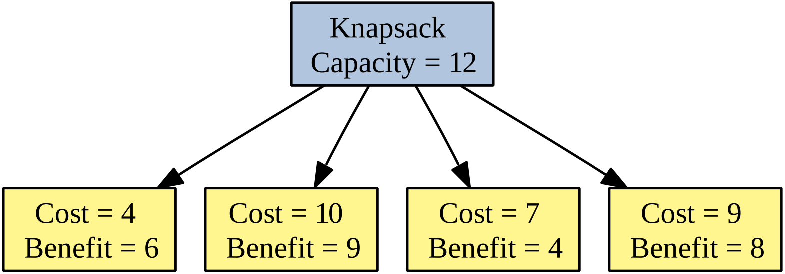
Computer software as **community service**

Backpacks and Bicycles



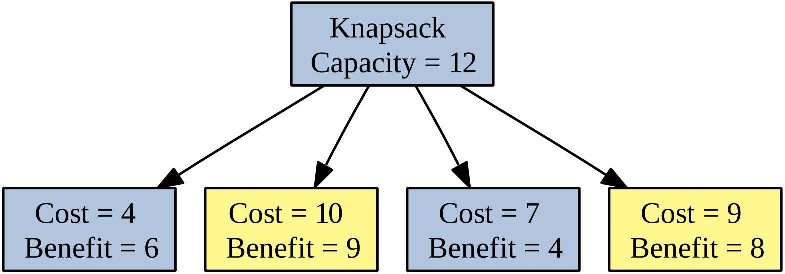
Goal: Find a backpack that will support your commute to work or school

0/1 Knapsack Problem



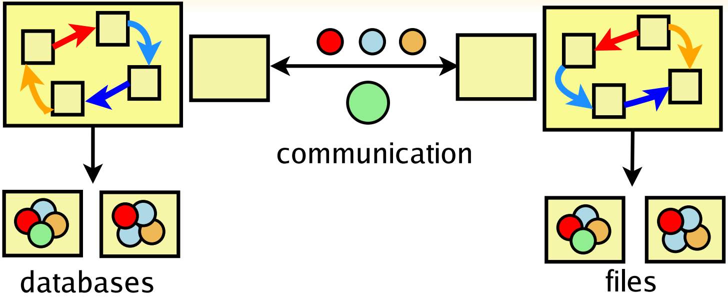
- **Question**: Can you select items so that you **maximize** the benefit while ensuring that the cost does not **exceed** the capacity?
- This problem is **NP-complete** (see Garey and Johnson) and yet it also has many practical applications in both **software** and **finance**

0/1 Knapsack Problem



- **Question**: Can you select items so that you **maximize** the benefit while ensuring that the cost does not **exceed** the capacity?
- This problem is **NP-complete** (see Garey and Johnson) and yet it also has many practical applications in both **software** and **finance**

Complexity and the Software Crisis



"Software entities are more complex for their size than perhaps any other human construct"

- **Frederick Brooks** (Professor of Computer Science at the University of North Carolina – Chapel Hill)

Regression Testing Techniques

Reduction Prunes the Test Suite

Prioritization Reorders the Tests

It is **expensive** to run a test suite $T = \langle T_1, \ldots, T_n \rangle$. **Prioritization** searches through the $n! = n \times n - 1 \times \ldots \times 1$ orderings for those that **maximize** an objective function like **coverage** or **fault detection**.

Prioritizing When Memory is Constrained

Frequent Memory Rewrites

High Testing Costs

Frequent **reads** and **writes** to memory may **increase** execution time by as much as **600%** when a Java application executes on a virtual machine with a **small heap**.

Solution: maximize memory **reuse** between test cases

The Impact of Test Ordering

	m_1 30	m_2 30	m_3 30	m_4 30	m_5 30	m_6 30	Test Size
T_1	•	•	•				90
T_2				•	•	•	90
T_3	•	•	•				90
T_4				•	•	•	90
T_5	•	•					60

- $T = \langle T_1, T_2, T_3, T_4, T_5 \rangle$ transfers **750** units to and from memory
- $T' = \langle T_2, T_4, T_1, T_3, T_5 \rangle$ only loads and unloads **180** units

Challenges of Test Prioritization

Real-World Suites Have Hundreds of Tests

- $n!$ possible solutions
- For 100 test cases, $9.33262154 \times 10^{157}$ possible solutions
- It takes 2.22 seconds to evaluate a solution permutation
- Finding the answer would take $6.52846694 \times 10^{149}$ years

Problem Formulation

- **Intelligently** comb the search space for an **effective** ordering
- Can **efficient** techniques identify **good** prioritizations?
- Will the prioritizers work properly in **real-world** software development environments?

Challenges of Test Prioritization

Real-World Suites Have Hundreds of Tests

- n! possible solutions
- For 100 test cases, $9.33262154 \times 10^{157}$ possible solutions
- It takes 2.22 seconds to evaluate a solution permutation
- Finding the answer would take $6.52846694 \times 10^{149}$ years

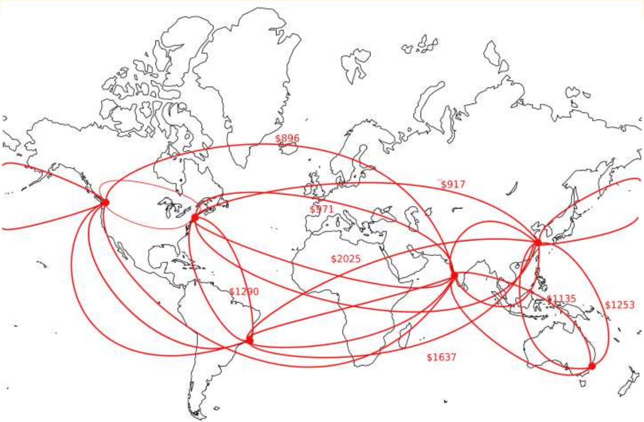
Problem Formulation

- **Intelligently** comb the search space for an **effective** ordering
- Can **efficient** techniques identify **good** prioritizations?
- Will the prioritizers work properly in **real-world** software development environments?

Cheapest Way to Travel the World?

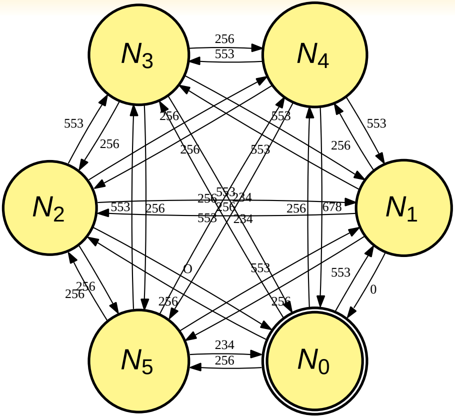
Cheapest way to see Seattle, Rio, Shanghai, Mumbai, and Sydney.

Cheapest Way to Travel the World?



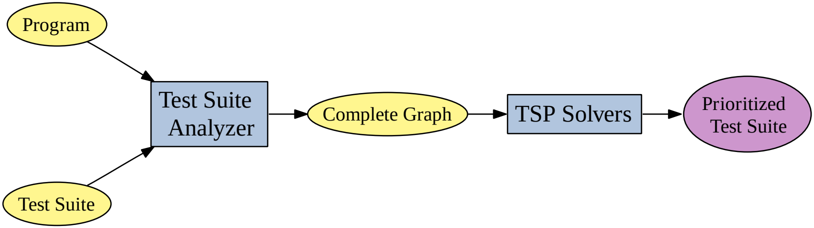
5! = 120 possible solutions

Classic Problem in Graph Theory



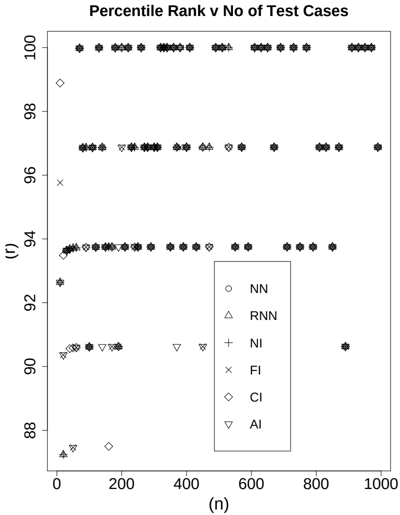
Goal: Find the least weight Hamiltonian path through a complete graph

Prioritizing with Hamiltonian Paths



- Is it possible to **efficiently** create test prioritizations by **solving** the “find the cheapest way to travel the world” problem?
- If yes, then are the orderings **effective**?

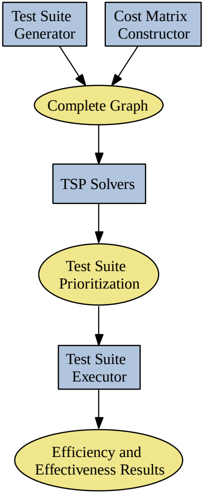
How Good are TSP Solvers?



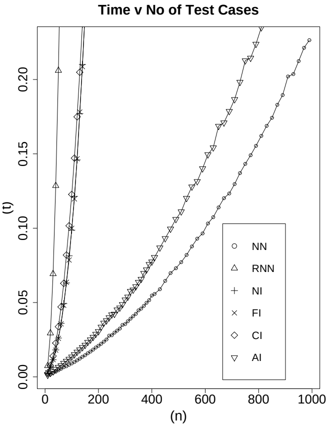
- Implemented by researchers at the University of Vienna
- Six key algorithmic techniques
- All algorithms produce 80+ percentile rankings for the path cost metric
- Most expensive algorithm takes 200s for $n = 1000$

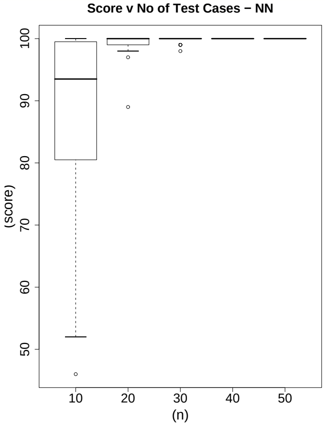
Experiment Design

- **Project Metrics**:
  - Lines of Code: 5,274
  - Data Files: 15,180
  - Data Files Size: 4.5 GB
- **Key Implementations**:
  - Synthetic Test Suite Generator
  - Cost Matrix Constructor
  - Test Suite Executor



Empirical Results





Efficient Prioritizers

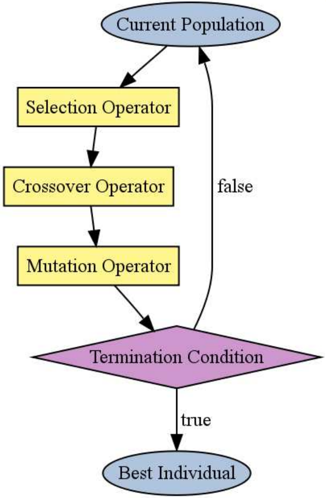
High Percentile Rankings

What is a Genetic Algorithm?

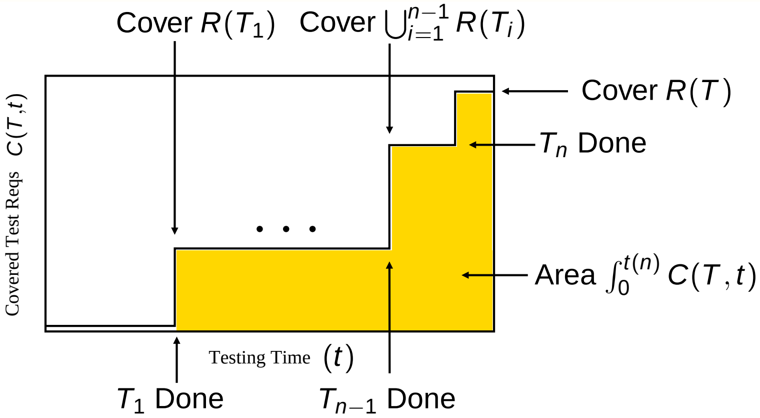
A **search-based** approach to test suite prioritization

Parts of a Genetic Algorithm:

- **Data Structures**:
  - Chromosome
  - Individual
  - Population
- **Functions**:
  - Selection Operator
  - Crossover Operator
  - Mutation Operator
  - Termination Condition
  - Fitness Function



What is Coverage Effectiveness?



- Prioritize to **increase** the CE of a test suite $CE = \frac{\text{Actual}}{\text{Ideal}} \in [0, 1]$

Genetic Test Suite Prioritizer

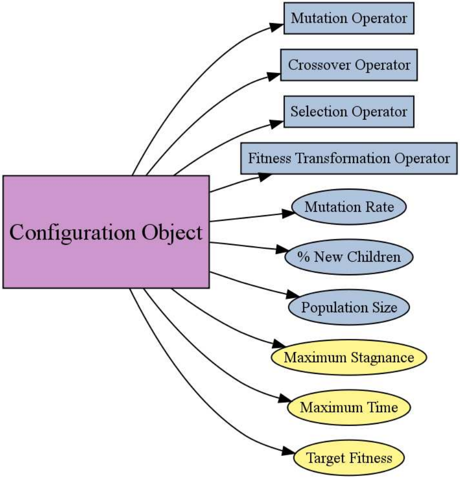
- **Motivation**: There are known instances where greedy techniques always yield sub-optimal orderings. Few experiments have studied the efficiency and effectiveness of search-based techniques.
- **Goals**: Identify configurations of the genetic algorithm that produce desirable results. Outperform random search.

Project Statistics:

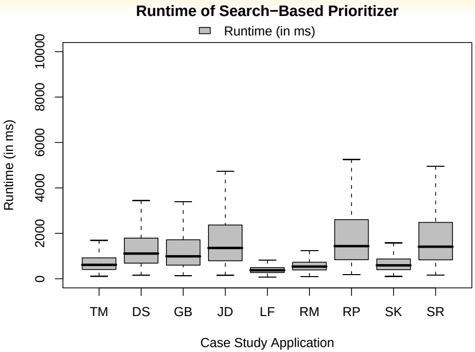
- 6,369 lines of code
- 6 mutation operators
- 7 crossover operators
- 3 selection operators
- 3 fitness transformation operators

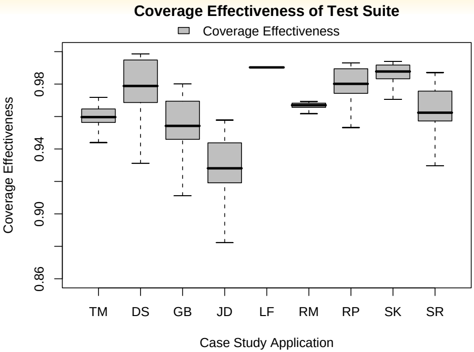
Experiment Design

- **Metrics**:
  - Runtime of prioritization technique
  - Coverage effectiveness of fi nal test ordering
- **Data Sets**:
  - 9 real-world
  - 54 synthetic
- **Configurations**:
  - 10,206 confi gurations
  - 91,854 real-world experiments
  - 551,124 synthetic experiments



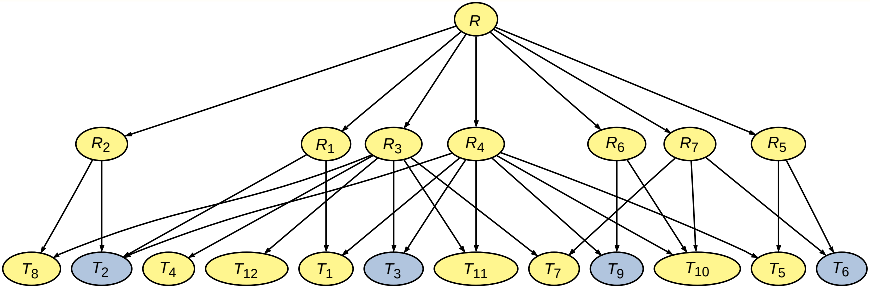
Empirical Results





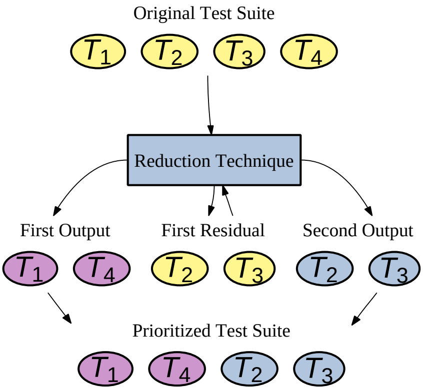
Preliminary investigation indicates that the genetic algorithm can produce **better** results in **less time** than random search

Finding the Overlap in Coverage



- $R_j \rightarrow T_i$ means that requirement R_j is **covered by** test T_i
- $T = \langle T_2, T_3, T_6, T_9 \rangle$ covers **all** of the test requirements
- Test suite **reduction** discards the test cases that **redundantly** cover the test requirements

Incorporating the Costs of a Test Case



- Harrold, Gupta, Soffa (HGS)
- Delayed Greedy (DGR)
- Traditional Greedy (GRD)
- 2-Optimal Greedy (2OPT)

Hypothesis: Using the execution **time** of a test case can **improve** the reduced and prioritized test suites

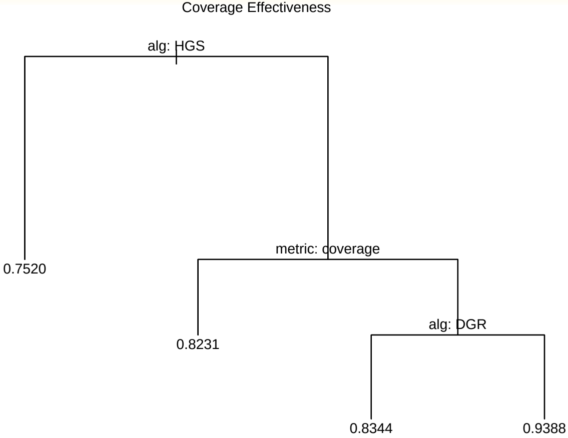
Compare (i) **greedy choices** (cost, coverage, and ratio) and (ii) **algorithms**

Greedy Choices Impact Effectiveness

	R_1	R_2	R_3	R_4	R_5	Execution Time
T_1	✓	✓	✓	✓		4
T_2			✓	✓		1
T_3		✓				1
T_4	✓				✓	1

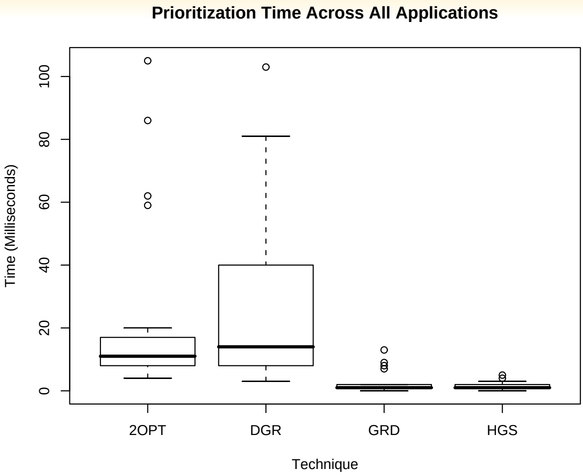
Greedy-by	T_r	$time(T_r)$	T_p	CE
coverage	$\langle T_1, T_4 \rangle$	5	$\langle T_1, T_4, T_2, T_3 \rangle$	0.400
time	$\langle T_2, T_3, T_4 \rangle$	3	$\langle T_2, T_3, T_4, T_1 \rangle$	0.714
ratio	$\langle T_2, T_4, T_3 \rangle$	3	$\langle T_2, T_4, T_3, T_1 \rangle$	0.743

Empirical Results: Effectiveness



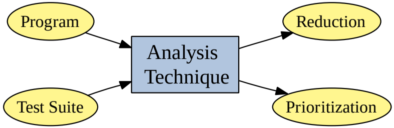
Using **ratio** and **time** improves the CE of the prioritized test suite

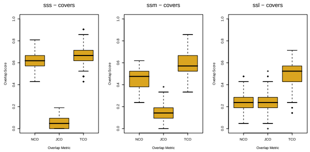
Empirical Results: Efficiency



Prioritizers are **useful** in practice because they incur **low time** overheads

Concluding Remarks





Algorithms and Software Tools

Detailed Empirical Evaluations

- **Experimental** computer science involves the **implementation** and **evaluation** of algorithms that handle **real-world** problems
- Solving the **software crisis** with freely available **data sets** and free/open source **computational artifacts**

http://www.cs.allegheny.edu/~gkapfham/research/