




Research Process
Practical Hints and Clues



Dr. Thorsten Papenbrock
Information Systems Group, HPI



Solution 1

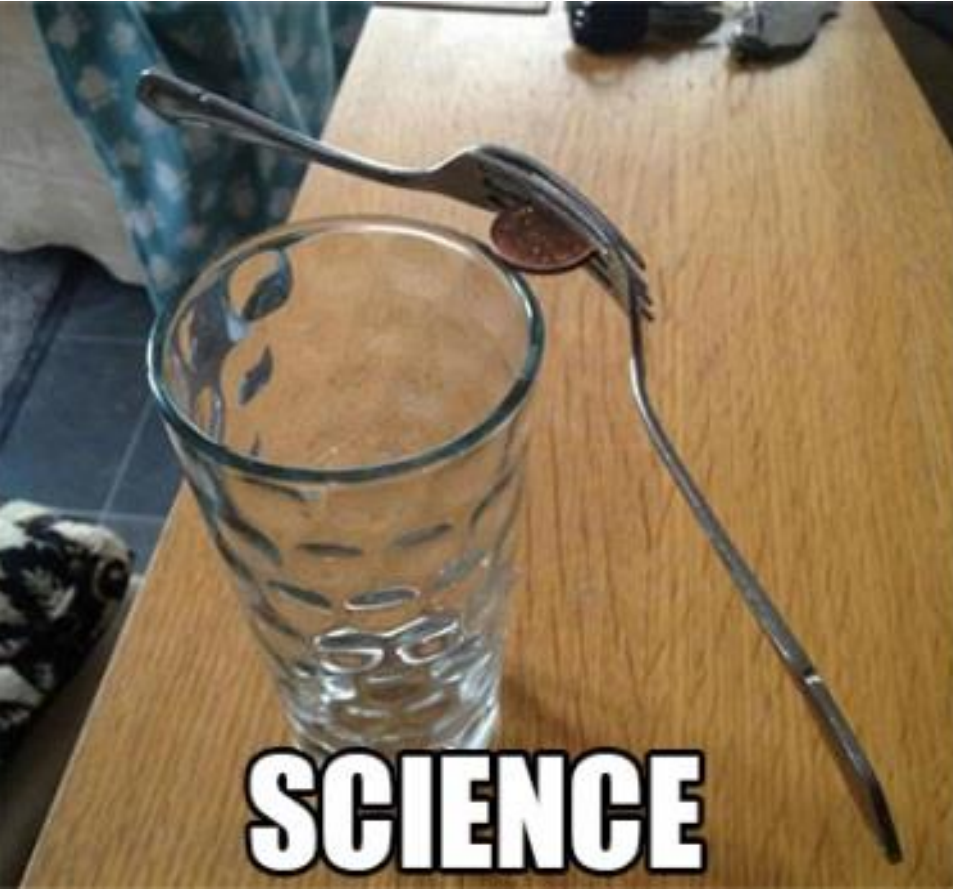


Solution 2



Innovation!

Science vs. Engineering



Science vs. Engineering

Science

Science is concerned with **understanding fundamental laws** of nature and the behavior of materials and living things.

Engineering

Engineering is the **application of science and technology to create useful products and services** for the whole community, within economic, environmental and resource constraints.

- What does this mean for us?
 - **Solve general problems!**
 - Do not optimize for very specific data, situations, use cases, ... (e.g. key discovery for only the ncvoter dataset)

Science vs. Engineering

Science

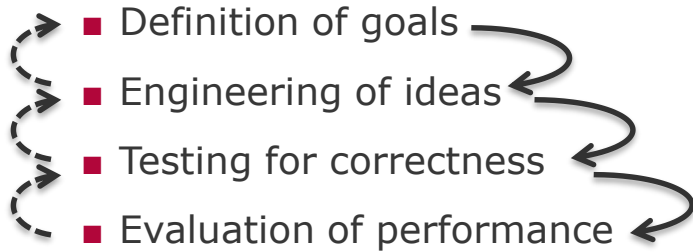
Science is concerned with **understanding fundamental laws** of nature and the behavior of materials and living things.

Engineering

Engineering is the **application of science and technology to create useful products and services** for the whole community, within economic, environmental and resource constraints.

- To **understand**, we need good **software** that correctly/efficiently analyses nature/materials/things
 - For good **software**, we need good **engineering**
 - **Good science implicitly requires good engineering** (although we rarely talk about the engineering efforts)

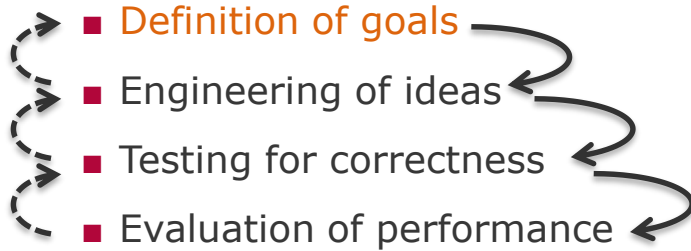
Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

Definition of goals

- **Identify the problem**

- Performance?
- Scalability?
- Precision?
- Completeness?
- ...

- **A concrete use case or application can help to ...**

- Find a problem
- Identify your contribution
- State why your contribution is relevant
- See what aspects of your contribution need to be evaluated

Definition of goals

■ SMART goals

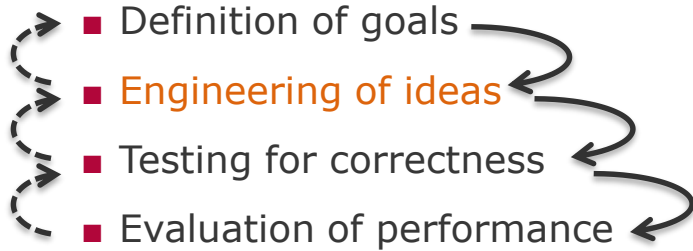
- **Specific:** What shall be the contribution of your algorithm?
 - Properties: efficiency, robustness, scalability, security, ...
 - Solve a new problem or enter a new dimension of a problem
- **Measurable:** How can you evaluate your progress?
 - Even better: can you monitor your progress?
- **Assignable:** Formulate sub-goals and solve them iteratively.
 - E.g. 1. correctness, 2. efficiency, 3. scalability, 4. robustness
- **Realistic:** Have a certain confidence in your ideas before taking them up.
- **Time-boxed:** Answers give rise to new questions, so consider time limitations.

Definition of goals

■ Write an exposé

- **Abstract:** 5 sentences
 - Motivation, Problem Statement, Approach, Results, Conclusion
 - E.g. <http://users.ece.cmu.edu/~koopman/essays/abstract.html>
- **Introduction:** Basically an extended abstract
- **Related Work:** What is already done?
Why is the problem not solved yet?
What we do as well?
What we do differently?
- **Approach:** The new idea that may solve the problem
- **Evaluation:** How to evaluate the idea (experiments and datasets)

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

■ Good research requires good engineering

- We sell new ideas, not the engineering of these ideas
- But even good ideas cannot be sold, if they are not properly engineered

■ “Premature optimization is the root of all evil.” – Donald E. Knuth

- It is likely to obfuscate your code
- Measure, then optimize where it pays off (pareto principle)
- Know your bottlenecks!

■ Design for performance

- **Performance Patterns:** Smith, C. U., & Williams, L. G. (2001). Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software. ISBN 0201722291
- **Performance Antipatterns:** Smith, C. U., & Williams, L. G. (2000, September). Software performance antipatterns. In *Workshop on Software and Performance* (pp. 127-136).

- **Know your programming language!**

- Know when costly operations happen
 - Call by reference/value
 - Autoboxing
 - (Tail-)Recursions
 - ...
- Java vs. Scala vs. Python vs. C

■ Pitfalls

□ Mind your I/O operations: latency vs. throughput

- $10\,000 * (t_{latency} + t_{write}) > t_{latency} + 10\,000 * t_{write}$
- Bundle I/O operations into batches, employ latency hiding when necessary

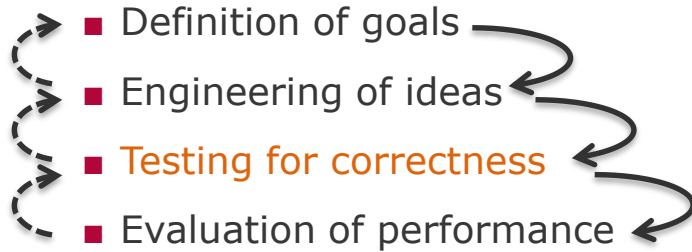
□ File and network operations are expensive

- Reduce the number of file/connection handles
(number of simultaneously open file handles is also limited)

□ If I/O is a problem, there are some optimization opportunities

- Avoid standard frameworks like Java serialization or Python pickle
- Employ binary serialization instead of JSON/XML
- Store text as UTF-8
- Got free CPU cycles? Compression can pay off...

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

Testing for correctness

■ Test your theory/hypothesis

- Write test cases!
 - TDD, tests first, unit testing, continuous testing, ...
- Compare results to some naïve baseline or some other reference implementation
 - In the best case: reference implementation of a different programmer and language
- The ultimate verification: proof correctness of your algorithm
 - If formal proof is hard, make an informal proof to yourself

■ Component tests

- Write unit tests for your utilities (e.g., parsers, helpers, calculators, data structures ...)

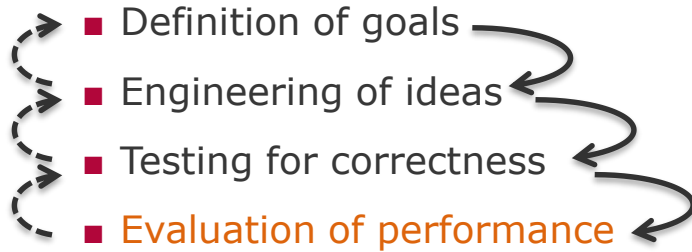
■ Artificial test examples

- Good to get a first running version
- Hardly will contain all the edge cases ← add edge cases to your tests while they occur

■ Reference implementation

- **Automatic** result comparison with reference implementation
- Helpful for debugging: output your delta with reference result
- Do it on as many datasets as possible

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

Evaluation of performance

- **Monitor your progress**

- Measure first, then optimize (cf. Engineering pitfalls)
- Measure often
 - Use many different datasets for testing to avoid optimizing for specific dataset characteristics
 - Explore your parameters

→ **Tip: Write a measurement framework early on!**

- If your goals are SMART, you know when you are done!

Evaluation of performance

- **Evaluation should provide ...**

- empirical evidence for your claims/goals/contributions

- **Evaluation results should therefore be ...**

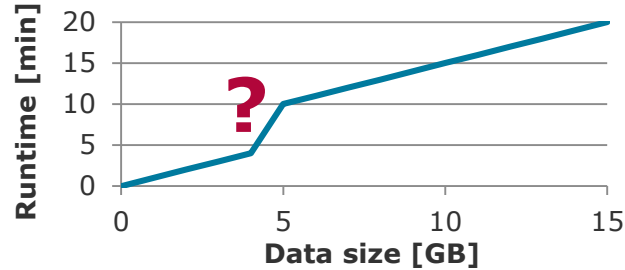
- ... credible, significant, relevant, realistic, general/generalizable, repeatable

■ The fine art of evaluation is to ...

- put performance into **context**:
 - Compare performance measurements to related work
 - Ensure fair comparison settings
 - You might need to re-implement and re-evaluate related work algorithms/approaches
- support your claims with as **few experiments** as possible
- **isolate and show** the effects of your design choices, innovations, and optimizations
- use a **variety** of datasets:
 - different domains and dataset characteristics
 - real-world and synthetic data

Evaluation of performance

■ Result interpretation



Assumption Hypothesis Example Interpretation Conclusion Proof



„Maybe it’s because of the memory consumption.“

„The same happens when I run some other algorithm on large data with too less RAM.“

„At 5 GB the algorithm had to page 4 GB to disk, which costs 5 min that we see in the chart.“

„If the data did not fit into main memory any more, the algorithm might have started paging.“

„The data became larger than main memory capacity at exactly that point; we know the algorithm starts paging then.“

„Given X GB to page. It follows that ... this costs Y min that we see in the chart. qed.“

■ Repeatability Tracks

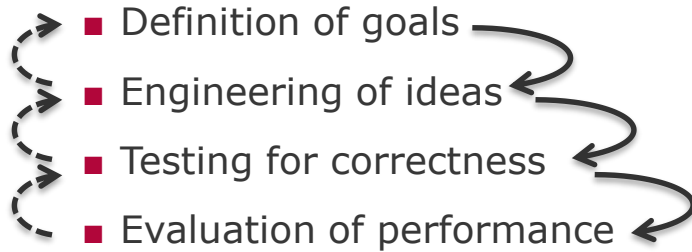
- SIGMOD since 2008: Repeatability
- VLDB since 2008: Experiments and Evaluation
- Consolidation and Validation
 - „Motivated by these surprisingly excellent results, we take a look into the rearview mirror. We have re-implemented the Dwarf index from scratch and make three contributions. First, we successfully repeat several of the experiments of the original paper. Second, we **substantially correct some of the experimental results** reported by the inventors. Some of our results **differ by orders of magnitude.**“

From: Jens Dittrich, Lukas Blunski, Marcos Antonio Vaz Salles. Dwarfs in the rearview mirror: how big are they really? VLDB 2008

- “... Allerdings konnte ebenfalls gezeigt werden, dass die Autoren bei dem Vergleich ihrer Verfahren mit der SNM offensichtlich nicht die transitive Hülle berücksichtigten, denn nur so konnten die großen Unterschiede in den Vergleichen nachvollzogen werden. Unter Berücksichtigung der transitiven Hülle **schneidet die SNM dagegen im Vergleich zu den vorgestellten Verfahren [Adaptive Sorted Neighborhood Method] sehr gut oder sogar besser ab.**“

From: Oliver Wonneberg, HPI, BTW 2009 Studierendenprogramm

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

Acquisition of data

■ Research is usually data driven

- Data is important to ...
 - define the problem and your goals
 - develop ideas for good solutions
 - test for correctness
 - test for performance
- But data owners usually do not give their data away
 - Data is the core capital of companies (strategic resource)
 - Data is private property (sensitive information)
- How to get data?



- **“At some point, every company writes its own data generator”**
 - HPI: dbtesma, hanaGenerator, ...
 - IBM, SAP, ...
- **Public data generators**
 - www.tpc.org/tpch/
 - www.generatedata.com/
 - www.red-gate.com/products/sql-development/sql-data-generator/
- **Advantage:**
 - As much data as you want!
 - Data fits your needs! (?)

- **Challenge: Generate data with real world characteristics**

- Benford Law Frequency, UCCs, INDs, FDs, ...

- **Chicken-Egg-Problem**

Use **profiling algorithms** to generate data with natural meta data characteristics!



Use data with natural meta data characteristics to build and test **profiling algorithms**!

- **Usual problems**

- Less variety in INDs, UCCs, and FDs (e.g. fd_reduced dataset)
- Data replication (at some point) due to the use of seed data
- Specific optimization for only a few meta data characteristics

Research Process

Acquisition of data

Enigma

<https://public.enigma.com/>

Data.Gov

<https://www.data.gov/>

Web Data Commons – Web Table Corpora

<http://webdatacommons.org/webtables/index.html>

MusicBrainz @BitBucket

<https://bitbucket.org/metabrainz/musicbrainz-server>

The GDELT Project

<https://www.gdeltproject.org/>

UCI Machine Learning Repository

<https://archive.ics.uci.edu/ml/index.php>

Kaggle

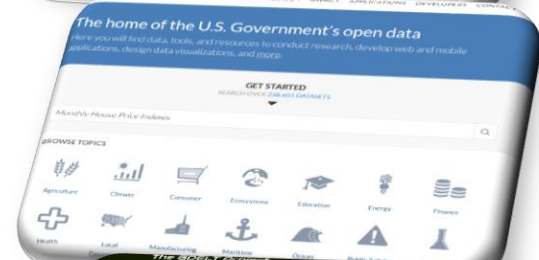
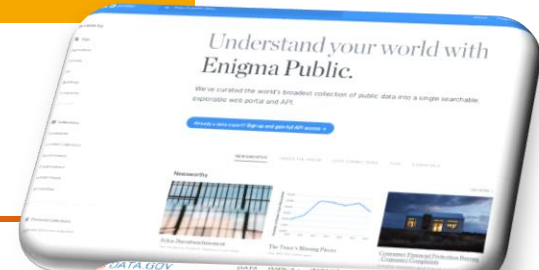
<https://www.kaggle.com/competitions>

Awesome Data @GitHub

<https://github.com/awesomedata/awesome-public-datasets>

Quora

<https://www.quora.com/Data/Where-can-I-find-large-datasets-open-to-the-public>



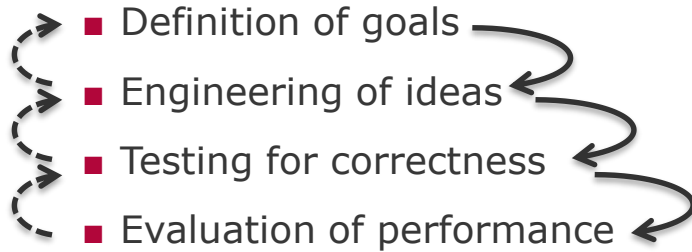
■ Good sources:

- WDC web tables project: <http://webdatacommons.org/webtables/index.html>
- UCI machine learning repository: <http://archive.ics.uci.edu/ml/>
- Public competitions: www.kaggle.com/competitions
- Collection: www.quora.com/Where-can-I-find-large-datasets-open-to-the-public

■ Which data to consider?

- Use datasets from different **domains**
- Use datasets of different **size**
- Use datasets in different **formats**
- Use datasets from different **sources**

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

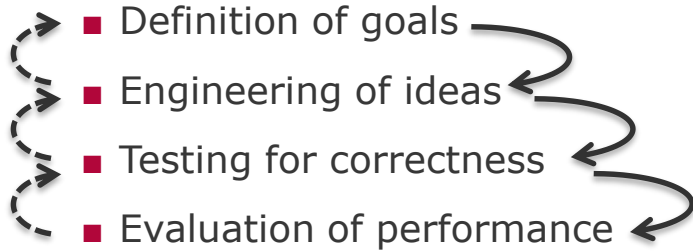
■ We often optimize for efficiency

- New complexity class:
 - e.g. $O(n^3) \rightarrow O(n^2)$
 - great, but do not try if complexity of problem is proven!
- Factorial improvement:
 - e.g. $O(3n) \rightarrow O(2n)$
 - great, but factor should be significant!
(otherwise the performance gain could be due to engineering)

■ Consider edge cases

- worst case, best case, and average case complexity

Algorithm Research

- Definition of goals
 - Engineering of ideas
 - Testing for correctness
 - Evaluation of performance
- 

Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

- **Good and well-engineered ideas go unnoticed without good presentation**
 - Talks & Write-ups
 - True for seminars, master theses, and scientific publications



■ Bugs in writing

- Personal Pronoun: We vs. I
 - We

- Time: Present vs. Past
 - Present
 - Past only if it is not avoidable

- Abbreviations:
 - Progressive Sorted Neighborhood Method (PSNM) → is OK

Presentation of results

■ **Introducing a new topic**

1. Motivate the new topic and describe the problem
2. Place the new topic in the overall picture of your work
3. Describe your solution
4. Consider and discuss alternative solutions/approaches
5. Justify your decisions
6. Conclude how your work solves the problem

■ Evaluating an experiment

1. Explain why and what you are going to test
2. Describe the test setup und environment
3. Describe what the test results look like and what can been seen
4. Explain expected results
5. Point out the reasons for significant, unexpected, surprising results
6. Analyse your findings and draw a conclusion for them

■ Abstract Writing (5 sentences)

- **Motivation:** *Why do we care* about the problem and the results? This section should include the importance of your work, the difficulty of the area, and the impact it might have if successful.
- **Problem statement:** *What problem* are you trying to solve? What is the *scope* of your work (a generalized approach, or for a specific situation)? Be careful not to use too much jargon.
- **Approach:** *How did you go about solving* or making progress on the problem? Did you use simulation, analytic models, prototype construction, or analysis of field data for an actual product? What was the *extent* of your work (did you look at one application program or a hundred programs in twenty different programming languages?) What important *variables* did you control, ignore, or measure?
- **Results:** *What's the answer?* Specifically, most good computer architecture papers conclude that something is so many percent faster, cheaper, smaller, or otherwise better than something else. Put the result there, in numbers. Avoid vague, hand-waving results such as "very", "small", or "significant." If you must be vague, you are only given license to do so when you can talk about orders-of-magnitude improvement.
- **Conclusions:** *What are the implications* of your answer? Is it going to change the world (unlikely), be a significant "win", be a nice hack, or simply serve as a road sign indicating that this path is a waste of time (all of the previous results are useful). Are your results *general*, potentially generalizable, or specific to a particular case?

■ Related Work

- Name **all** related work
- Describe it as much as it **relevant** for your approaches
- **Differentiate** it from your work (or explain commonalities)
- Name the **weaknesses** (if possible)

■ Avoid passive !!!

- Always write who did what!
→ avoid passive!
- Example: **The sentence is converted to lower-case letters.**
→ **We convert the sentence to lowe-case letters.**
→ **The algorithm/function/actor/thread/process/worker/...**

■ Informal speech

- “as” and “since” in the meaning of “because” are informal speech

■ Be precise and put the adjectives, adverbs, ... where they belong

- Example: I worked on my thesis hardly.
→ I hardly worked on my thesis.
- Example: I worked on my thesis only.
→ I only worked on my thesis.
→ I worked only on my thesis.

■ Mind the correct ordering of sentences (Yoda speech)

- Example: I on my own wrote a letter.
→ I wrote a letter on my own.

■ The Oxford-Comma

- He collected books, papers, and documents.
- He collected books, wrote papers and read documents.

■ The Introducer-Comma

- However, ...
- Additionally, ...
- To achieve a goal, ...
- As we saw earlier, ...

■ **which vs. that**

□ "that"

- There is never a "," in front of "that"
- Introduces a relative sentence that is necessary for the understanding of the main sentence

□ "which"

- There is always a "," in front of "which"
- Introduces a relative sentence that gives additional information and is not necessary for the understanding of the main sentence

Presentation of results

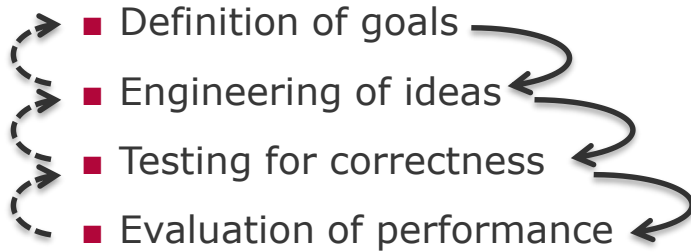
■ Reference formatting

- Most importantly: Make the references consistent!
- If you use one attribute for a “book”, then use it for all books
- Use the same abbreviation pattern for all conference names (not “Com. of the ACM” **and** “Communication of the ACM”)

12. REFERENCES

- [1] Z. Abedjan, P. Schulze, and F. Naumann. DFD: Efficient functional dependency discovery. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 949–958, 2014.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 487–499, 1994.
- [3] P. Bohannon, W. Fan, and F. Geerts. Conditional functional dependencies for data cleaning. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 746–755, 2007.
- [4] C. R. Carlson, A. K. Arora, and M. M. Carlson. The application of functional dependency theory to relational databases. *Computer Journal*, 25(1):68–73, 1982.
- [5] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.
- [6] E. F. Codd. Further normalization of the data base relational model. *IBM Research Report, San Jose, California*, RJ909, 1971.
- [7] G. Cormode, L. Golab, K. Flip, A. McGregor, D. Srivastava, and X. Zhang. Estimating the confidence of conditional functional dependencies. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 469–482, 2009.
- [8] S. S. Cosmadakis, P. C. Kanellakis, and N. Spyrtatos. Partition semantics for relations. *Journal of Computer and System Sciences*, 33(2):203–233, 1986.
- [9] P. A. Flach and I. Savnik. Database dependency discovery: a machine learning approach. *AI Communications*, 12(3):139–160, 1999.
- [10] E. Garnaud, N. Hanusse, S. Maabout, and N. Novelli. Parallel mining of dependencies. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*, pages 491–498, 2014.

Algorithm Research



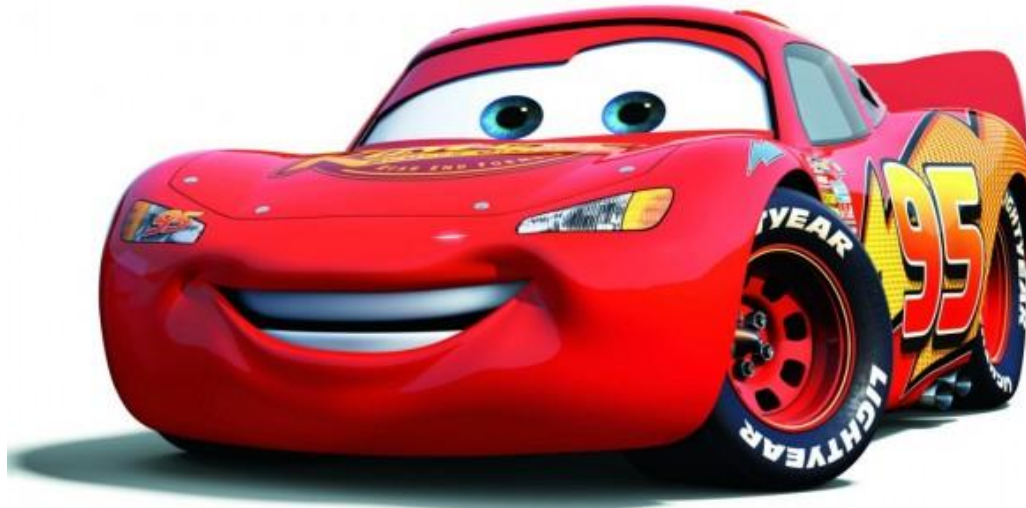
Cross-cutting concerns

- Acquisition of data
- Analysis of complexity
- Presentation of results

■ Hints and Clues

- Research → Reading
- Research → Innovating
- Research → Engineering
- Research → Structured Working, Writing, and Presentation

Efficient Java Code



Algorithm design comes first

- Utmost efficient code will not compensate for a bad algorithm
- Think about data structures: *trees, tries, hash tables, linked lists, arrays, heaps, ...* and algorithmic strategies: *sorting, hashing, search strategies, ...* first, including their complexities

Avoid premature optimization

- Create a working algorithm first
- Measure and detect bottlenecks to optimize the hot spots of your code

Code efficiency can go along with good algorithm design

- Study patterns/anti-patterns, coding idioms and best-practices
- Research performant libraries
- Stick to the two above points
- That is: Achieve performance by design, not by hacking!

Java is not slow

- But it is easy to write inefficient code
(stressing garbage collection, autoboxing, thoughtless String-handling, ...)

Before tweaking

- Make sure you have a good algorithm
- Detect bottlenecks (are you CPU, memory, or I/O bound?)
- Create (micro-)benchmarks to measure the effects
- Use benchmarks to pinpoint the problem

While tweaking

- Continuously verify correctness with unit/integration tests

- Inefficient algorithm
- Inefficient loops
- Garbage collector
- Unoptimizable code
- (Un-)Boxing
- Inefficient string handling
- Parallelization issues: starvation, too much synchronization etc.

- Inefficient algorithm
- Caching unnecessary objects
- Objects too large
- Overallocated strings, collections, maps
- Oversized data types

- Inefficient algorithm
- Too many file/network accesses
- Sequential vs. random access
- Serialization inefficient
- Serialized objects too large
- Inefficient caching

- Immutable objects are needed for good API design
- Easy to use in defensive API design
- Address immutable

```
class Person {  
    private final String name;  
    private final Address address;  
    public Person(final String name, final Address address) {  
        this.name = name;  
        this.address = address;  
    }  
}
```

- Person also immutable

Mutable vs. Immutable #2

- Mutable objects are better for fast code
- Harder to use in defensive API design
- Address mutable

```
class Person {  
    private final String name;  
    private final Address address;  
    public Person(final String name, final Address address) {  
        this.name = name;  
        this.address = new Address(address);  
    }  
}
```

- What happens if we don't copy the address?

When to use mutable objects?

- Fetching data becomes expensive with immutable objects

```
Map<Person, Integer> personOccurrences = new HashMap<>();
public void countOccurrences(DataInput logFiles, int logCount) throws IOException {
    for (int index = 0; index < logCount; index++) {
        String name = logFiles.readUTF();
        String place = logFiles.readUTF();
        Person person = new Person(name, new Address(place));
        final Integer oldValue = this.personOccurrences.get(person);
        this.personOccurrences.put(person, oldValue == null ? 1 : (oldValue + 1));
    }
}
```

- Need to create a new Person and Address for each log entry

When to use mutable objects?

■ Use lookup object

```
Map<Person, Integer> personOccurrences = new HashMap<>();  
public void countOccurrences(DataInput logFiles, int logCount) throws IOException {  
    Person person = new Person();  
    for (int index = 0; index < logCount; index++) {  
        String name = logFiles.readUTF();  
        String place = logFiles.readUTF();  
        person.setName(name);  
        person.getAddress().setPlace(place);  
        final Integer oldValue = this.personOccurrences.get(person);  
        this.personOccurrences.put(person, oldValue == null ? 1 : (oldValue + 1));  
    }  
}
```

- Anti-pattern

```
String name = new String("Peter");
```

- String literals and interned strings are managed by string pool

- Can be tested for equality with `==`

- Similar `Integer.valueOf` maintains small pool

- `[-128, 127]` by default

- `"java.lang.Integer.IntegerCache.high"`

- Maintain pool if few different objects

- That needs to be looked up often

- XML attributes

- EHCACHE provides map-like interface
 - Removes entries if a certain size is reached
 - Different strategies, LRU most often used
- Data is spilled to disk if configured
- Can use third tier caches as well
- Useful if you want to maintain an object pool and you don't know what is needed most

“There are only two hard things in Computer Science: cache invalidation and naming things.”

-- Phil Karlton



■ Anti-pattern

```
String alphabet = "";  
for (char letter = 'a'; letter <= 'z'; letter++)  
    alphabet += letter;
```

■ Use `String +` only when you know what you are doing

- Never use `+=` inside a loop
- Compiler does it on its own for

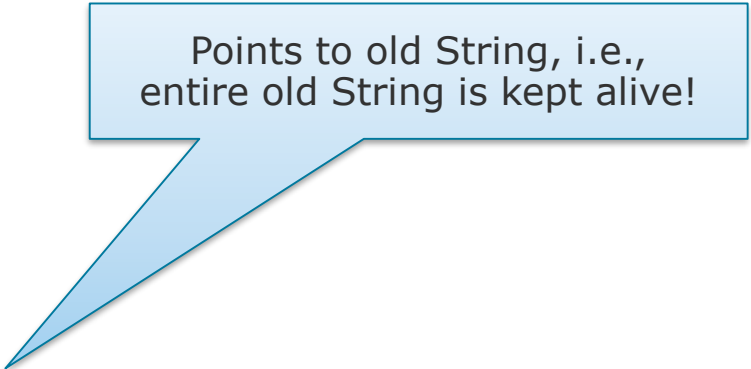
```
String name = firstName + " " + lastName;
```

■ Remember `String` is immutable, needs lots of copying

■ Use `StringBuilder` instead

(`StringBuffer` is the thread-safe, i.e., slower version)

■ Consider using `new String(...)` instead of `...substring(...)`



Points to old `String`, i.e.,
entire old `String` is kept alive!

Get the `int` from an `Integer` without casting:

```
someInteger.intValue()
```

- Beware of boxing and unboxing
 - Strongly degrades performance

```
Map<Person, Integer> personOccurrences = new HashMap<>();  
Person person = new Person(name, new Address(place));  
final Integer oldValue = this.personOccurrences.get(person);  
this.personOccurrences.put(person, oldValue == null ? 1 : (oldValue + 1));
```

- Use `fastutil` or `trove` instead

```
Object2IntMap<Person> personOccurrences = new Object2IntOpenHashMap<>();  
this.personOccurrences.defaultReturnValue(0);  
Person person = new Person(name, new Address(place));  
final int oldValue = this.personOccurrences.getInt(person);  
this.personOccurrences.put(person, oldValue + 1);
```

- Also: very many `Integer` objects (or the like) will eventually effect the GC.
Fastutil & co can reduce the amounts of objects.

Double vs. Float

- Often `double` is not needed and `float` is sufficient
- Halves memory consumption
- CPUs usually can perform more floating operation or with less cycles
- Don't ever use one of these types for currencies

- Use `final` as often as possible
- Helps to find programming errors
- Helps compiler/JIT to inline
- IMHO final parameters and variables should work most of the time
- Final classes are also good if you don't devise APIs

- Anti-pattern: Exception Driven Programming

```
int index = 0;
List<String> strings;
try {
    while(true)
        System.out.println(strings.get(index++));
} catch(IndexOutOfBoundsException e) {
}
```

- To show errors, exceptions are essential and good
- Should not be part of normal workflow
- Primitive return times are better if the result is expected
- Most time is spent in creating stack trace

- Always implement `hashCode()`, `equals()`, `toString()`
 - Eclipse and IntelliJ help to implement them
- Use logging, especially in a multithreading environment
 - Popular libraries: SLF4J, log4j, Logback
 - Fine-grained configuration of which logs to display
- Use constant `boolean` expressions for debug statements
 - Changing it to `false` allows compiler to remove all debug branches

```
public final static boolean DEBUG = true;
```

-
- Monitors your application
 - Shows memory consumption
 - Can be used for profiling (install sampler plugins)
 - Very useful to create memory dumps and to query them
 - Finds over-allocated strings and collections
 - Quickly shows you when your data structures are larger than expected
 - Can also be used for remote sessions



de.hpi.isg.sodap.sindy.jobs.Sindy (pid 4696)

Overview Monitor Threads Sampler Profiler

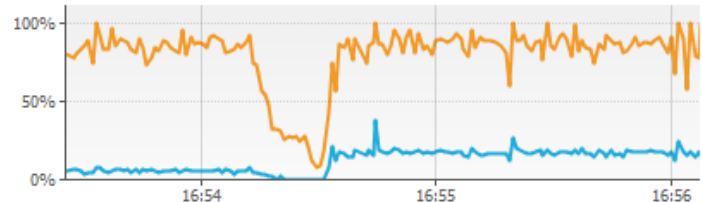
de.hpi.isg.sodap.sindy.jobs.Sindy (pid 4696)

Monitor CPU Memory Classes Threads

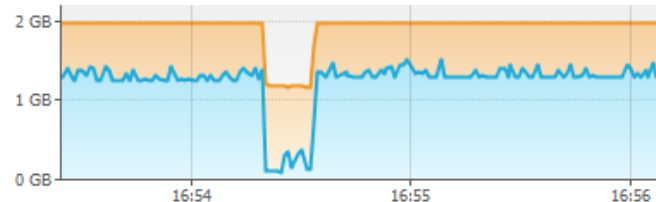
Uptime: 2 min 55 sec

Perform GC

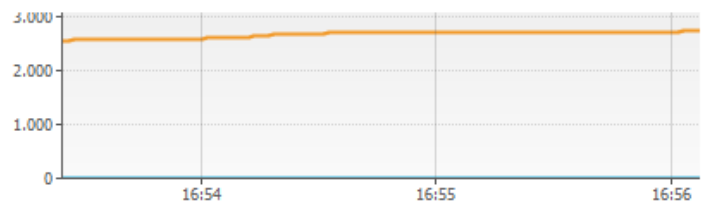
Heap Dump

CPU 

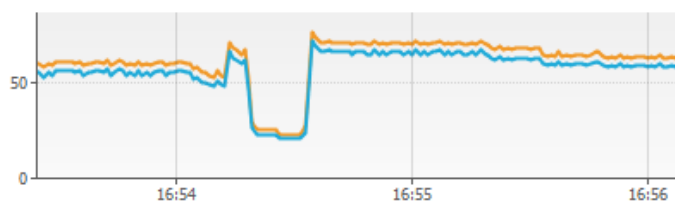
CPU usage GC activity

Heap PermGen 

Heap size Used heap

Classes 

Total loaded classes Shared loaded classes

Threads 

Live threads Daemon threads

- Try `java.util.concurrent` package first before custom solution
- Use lock-free structures
 - `ConcurrentLinkedQueue`, `ConcurrentHashMap`
(Note that `size()` is not constant)
 - `AtomicInteger`, `AtomicReference`
- Never use `Vector`, `Hashtable`
 - Synchronized versions of `ArrayList`, `HashMap`
 - But only for atomic operations
- Never use `volatile` as substitution for synchronized blocks
 - Does not help with write-write conflicts
 - Useful for stop flags

No thread-local
copies of that value

Fully thread safe
critical section

Engineering pitfalls

- General hints ...
 - Use fastutil, TROVE, ... for primitive collections
 - If your heap is full, your garbage collector will eat your CPU cycles
 - Objects need much more space than serialized representations
 - Profile before optimization! → jVisualVM