MAT-70906 Software Science Project

Lecture 2:

About publishing, reviewing an article, and proposed subjects for projects

Tampere University of Technology
periods III–IV, 2013-2014

Antero Kangas
antero.kangas@tut.fi
Experiences of the most common publishing forums

Workshops, conferences, and journals.

- Workshops
  - The fastest but least valued.
  - Selected by the abstract (a bad sign), or by the full-length articles.
  - Acceptance ratio 30 – 70 %.
  - One review cycle.
  - Presentations take typically $\sim 1/2$ hour, sometimes even more.
  - E.g. FMICS (Formal Methods for Industrial Critical Systems), DoSTA (Domain-Specific Approaches to Software Test Automation).
• Conferences
  – Selected usually by the full-length articles.
  – Acceptance ratio smaller than in workshops.
  – Presentation time at most half an hour.
  – Usually more audience than in workshops.
  – Examples: ICSE (International Conference on Software Engineering), CAV (Computer Aided Verification).

• Journals
  – The slowest and most valued.
  –Selected by the full-length articles.
  – Several repair cycles (possibility to limited communication with anonymous reviewers).
Cheating conferences and cheating articles/1

- The purpose of a cheating conference?

“SCIgen is a program that generates random Computer Science research papers, including graphs, figures, and citations. It uses a hand-written context-free grammar to form all elements of the papers. Our aim here is to maximize amusement, rather than coherence.” [from SCIgen p.]
Cheating conferences and cheating articles/2

“One useful purpose for such a program is to auto-generate submissions to conferences that you suspect might have very low submission standards. A prime example, which you may recognize from spam in your inbox, is SCI/IIIS and its dozens of co-located conferences (check out the very broad conference description on the WMSCI 2005 website). There’s also a list of known bogus conferences. Using SCIgen to generate submissions for conferences like this gives us pleasure to no end. In fact, one of our papers was accepted to SCI 2005! See Examples for more details.” [from SCIgen p.]
Experiences of Reviewing an Article

• Those who read reviews are
  – other reviewers (usually members of the program committee)
  – writers of the article

• Typically in a review
  1. the article is evaluated
  2. the core of the article is summarised in a few sentences
  3. the pros and cons of the article are brought out
  4. improvements are suggested
Reviewing an Article: Evaluation

A workshop:

*** OVERALL EVALUATION:
--- 3 (strong accept)
--- 2 (accept)
--- 1 (weak accept)
--- 0 (borderline paper)
--- -1 (weak reject)
--- -2 (reject)
--- -3 (strong reject)

*** REVIEWER’S CONFIDENCE:
--- 4 (expert)
--- 3 (high)
--- 2 (medium)
--- 1 (low)
--- 0 (null)

A conference:

Confidence level: 1 2 3 (highest)

Numerical Ranking

Please give scores from 1 to 10 for the following categories:

Appropriateness to the CONCUR
Originality
Technical Strength
Presentation
Overall Evaluation

The scores should follow the following ranking criteria:

1 = Absolutely unacceptable, incorrect, out of scope, etc.
2 = Very weak, definitely below conference quality.
3 = Paper may have some merits, but doesn’t meet conference standards.
4 = Paper has some merits, but perhaps doesn’t meet CONCUR standards.
5 = Neutral score, could go either way.
6 = Marginal paper, maybe above threshold.
7 = Typical CONCUR paper.
8 = Very good, clear accept.
9 = Outstanding paper, maybe among the 5 best accepted.
10 = Breakthrough, best paper.
Reviewing an Article: Core of the Article

This article introduces a debugger for web applications running on top of Ruby on Rails and TurboGears (Python). The debugger is able to follow sessions (between a web browser and a server) even if the same session contains several threads and processes, and even if the same process is involved in multiple sessions.

The debugger enables testing of otherwise very hard-to-test concurrent behaviour in a certain domain. Authors also claim (the first paragraph in Section 6) that testing can be automated. Thus, the article is in the scope of this workshop.

...
Reviewing an Article: Suggested Improvements

... 

However, there are some issues that should be fixed in the article. I will list them here in the order of importance.

1. It is hard to estimate the novelty of the approach, because the authors do not cover related work. I think that McDowell & Helmbold’s survey "Debugging concurrent programs" might give a valuable perspective for the approach presented in this article.

2. The use of the debugger for testing purposes, especially when testing is automated, should be explained more carefully. I could not follow how the automation is supposed to work.

3. The first four sections were easy to read, but Sections 5 and 6 were rather unclearly written. The language in those sections should be checked.
Reviewing an Article: Another Example

This paper presents an algorithm for distributed model checking. The algorithm takes advantage of threading to use the resources (dual-core CPUs) of each computing node more efficiently. The algorithm sends states between nodes in junks. This approach offers a couple of parameters (NumLines and LineSize) with which the communication between nodes can be adjusted so that the algorithm runs efficiently in the computing environment at hand (available memory in the nodes, network and CPU speed).

The paper is well organised and easy to follow. The only problem in organisation is that figures 3 and 4 (worker thread) are on page 12, but page 7 or 8 would be a far more suitable place for them. Now there is Figure 5 on page 8.

In the following, I will first handle one issue in the proposed algorithm and suggest a fix, and finally I give a short list of other notes.
1. An error in the stopping condition of the algorithm

The most crucial problem in the paper is that there seems to be an error in one condition for the termination of the algorithm. On page 13 it is claimed: "In fact, [the root node receives the token and finds global sent and received counter to match] implies that all the nodes are inactive...". It seems to me that the claim does not hold in general.

I think that the problem will be fixed if every time the token arrives to the root node, the node checks whether or not the sum of sends and receives has increased since the last time the node had the token. If there is no increase, the computation is finished. Informal reasoning for this to hold is the following.
2. Other notes

On page 8, left column: using "owner(s)" instead of "p(s)" could be better.

On page 11, left column: the same as above.

On page 15, Fig. 9, line 7: you could point out a possible need for mutual exclusion in the queue handling. Depending on the implementation, this line might interfere with Fig. 10, lines 5-6.

On page 16, Fig. 10: With this font number one and the letter 1 look the same. Perhaps you could use variable name "$ln$", for example.

On page 17, left column, 4th last row: "while (1)" does not exist in the code, there is "while (true)".

On page 17, left column, 3rd last row: to choose the "given number of attempts" wisely, you probably need to know how many cores you have in the node. With a single core processor, the good number might be one (yield always). With a dual core processor without any other calculations running, the thread should never yield, if I understood correctly. I think you could elaborate a bit choosing the number.
About Your Article

• Can be written either in English or in Finnish
• Must contain the Abstract (Tiivistelmä in Finnish), at most 150 words
• Must have at least the following sections: Introduction, Background, Summary or Conclusions, and References
• Length 10–12 pages
• Must be written using LaTeX
• The deadlines are strict but if you have a good reason, you can ask more time, but ask in time! Notice that the next deadlines do not move.
Proposed Subjects for Project

- Measuring the speed of C++ sort
- Comparison indexing and iterator of C++ containers
- Explicit state space
- String matching
- Binary Search Trees
- Selection
- Or…
The C++ sort is an implementation of *introsort* or *introspective sort* algorithm designed by David Musser. It begins with quicksort and switches to heapsort when the recursion depth exceeds a level based on (the logarithm of) the number of elements being sorted.


**Project:**

Measure the speed of C++ sort with at least the following materials:

1. Elements are in a random order, all elements non-equal.
2. The material is in strictly increasing order.
3. The material is in strictly decreasing order.
4. In the material there are many same values: \( k \) different values, each having a quantity of \( \lfloor \frac{n}{k} \rfloor \) or \( \lceil \frac{n}{k} \rceil \). All elements are in random order.
5. The material has one value “plenty” and the rest of values exactly one of
each. All in random order.

C++ Sort/2

The tests must be run in a separate laptop to avoid any network load.

Present the results in \((n, t)\)–coordinate systems, \(n\) must be chosen so that \(t_{\text{max}} \approx 100\) s (at least).

Use the least squares fitting method and fit the results to the following formulas for materials:

1–3: \(a + b \cdot n + c \cdot n \cdot \log n\), and

4–5: \(a + b \cdot k + c \cdot n + d \cdot n \cdot \log k\).

Suitable reading: The least squares fitting method, and how to generate a random material (a random permutation).
C++ Containers

Vector (impl. as a dynamic array) and deque (double ended queue) are a kind of sequence containers. Their elements are ordered following a strict linear sequence. Map (impl. as a binary tree) is a kind of associative container that stores elements formed by the combination of a key value and a mapped value.

Revieavable article: Read some article on (these) data structures, or Valmari, A.: “Käytännön kokemus algoritmikirjastojen autuudesta”. Tietojenkäsittelytiede 29, Lokakuu 2009, sivut 49–70. (http://www.tkts.fi/lehti/a29/valmari.pdf).

Project: Make a plan for an experiment for comparison of the used time when elements are searched by indexing or using the iterator of the container. Make the comparisons for C++ containers vector, deque, and map.

Suitable reading: data structures.
Explicit state space

A very tight hash table – data structure is described in the reviewable article. It was inspired by a need to construct the state space of the $2 \times 2 \times 2$ Rubik’s cube.


**Project:** Design, implement, test and measure the memory consumption of a set of $10^7$ random 35-bit elements. The entries have no accessory bits but each block has four extra bytes for the size of the block and the overflow link. Compare the memory consumption to theoretical minimum and ordinary array.

**Suitable reading:** Data structures, especially hash tables.
String Matching/1

Finding all occurrences of a pattern in a text is an important problem that arises frequently in text-editing programs. Several algorithms have been developed, e.g. Rabin-Karp (RP) has same worst-case running time than the naive brute force algorithm but works better in practice, similar, but much better, is Knuth-Morris-Pratt (KMP), and Boyer-Moore is often the best practical choice.

Revievable article:

String Matching/2

Project: Implement, test, and compare the efficiency of naive brute force, RP, KMP, and BM algorithms. Use different test materials: short text, long text, small alphabet, large alphabet, and the string to be searched appears zero time, once, a few times or many times in the text.

Binary Search Trees/1

Search trees are data structures that support many dynamic-set operations in $O(h)$ time, thus, the set operations are fast if the height $h$ of the tree is small; but if its height is large, their performance may be no better than with a linked list. Red-black trees are balanced in order to guarantee that basic dynamic-set operations take $O(\log n)$ in the worst case.

Binary Search Trees/2

**Project:** Implement (or take from C++ standard library) Binary Search Tree and Red-Black Tree algorithms, measure and compare their time consumption. Do the tests with natural language (e.g. a book) and with generated material of only four character-alphabet (simulating DNA-chains).

**Suitable reading:** Chapter of binary search trees and red-black trees in Corman, Leiserson and Rivest: *Introduction to Algorithms.*
Selection/1

The selection problem can be specified formally as follows:
Input: A set $A$ of $n$ (distinct) numbers and a number $i$, with $1 \leq i \leq n$.
Output: The element $x \in A$ is larger than exactly $i - 1$ other elements of $A$.
A median is the half-way point of the set.

Selection/2

Project: In Corman, Leiserson and Rivest: *Introduction to Algorithms*. there are introduced two algorithms for solving the selection problem: *Selection in expected linear time* (SEL) and *Selection in worst-case linear time* (SWL) compare selection on worst-case in linear time. Implement both and use the for finding median. Test and compare their time consumption.

Or..

Suggest your own subject.

A typical project could be an analysis of some algorithm or comparison of 2-3 algorithms that solve the same problem.

I.e. explain the problem, tell about the background theory of the algorithms, decide and justify your criterions for comparison, implement the algorithm(s), design suitable test cases, run tests, and present and analyse the results...

Find such an algorithm or problem and propose it!

**Suitable reading:** Chapter of Corman, Leiserson and Rivest: *Introduction to Algorithms.*

From its bibliography you can also find a suitable article for the first task.
Next

- Choose or propose the subject for your project. The subject must be accepted by the lecturer.
- Find an article of the topic of your project and write a review, one page, of it
- The next deadline is 29.1.2013 (returning the review of the article)
- Start learning Latex
- Remember to start your email subjects by \textit{MAT-70906}