

Project Practical Data Structures and Algorithms¹: Final Report

Institution:	Department of Computing, University of Copenhagen (DIKU)
Project duration:	1.1.2003–28.2.2006 (extended with two months)
Principal investigator:	Jyrki Katajainen, Assoc. Prof.
Other investigators:	Claus Jensen, graduate student Andrei Bjarke Moskvitin Josephsen, graduate student Stephan Lynge Herlev Larsen, graduate student Jeppe Nejsum Madsen, M. Sc. Jakob Gregor Pedersen, graduate student Jesper Holm Olsen, M. Sc. Frederik Rønn, M. Sc. Jørgen Havsberg Seland, graduate student Jacob De Fine Skibsted, graduate student Søren Christian Skov, M. Sc. Christian Ulrik Sõttrup, graduate student Anders Thøgersen, graduate student
External collaborators:	Hervé Brönnimann, Ass. Prof. (Polytechnic University) Amr Elmasry, Ass. Prof. (Beirut Arab University) Tomi Armas Pasanen, Lecturer (University of Helsinki) Fabio Vitale, Ph. D. student (University of Insubria)

General picture

The performance engineering laboratory² (PE-lab) in the Department of Computing at the University of Copenhagen³ was founded in 1998. Our mission is to educate elite programmers and to do high-quality research related to any aspect of programming, performance programming in particular. One of the cornerstones of the graduate curriculum offered by our department is research orientation. To make research orientation visible for our students they can become members of the PE-lab. Since the graduate education is supposed to take two years (or four years for part-time students), the contracts between the students and the PE-lab are no longer than one or two years—a normal duration being the period the students write their master's thesis.

¹ Supported by the Danish Natural Science Council under contract 21-02-0501. Original application is available online at

<http://www.cphstl.dk/Report/SNF-application-2002/cphstl-report-2002-8.pdf>.

² <http://www.diku.dk/forskning/performance-engineering/>

³ <http://www.diku.dk/>

In September 2002, at the time when the grant application was written, the PE-lab consisted of five graduate students and the principal investigator. All these students have now completed their master's education: Jesper and Søren in December 2002, Jeppe in March 2003, Frederik in August 2003, and Fabio in March 2006. All others except Fabio, who will continue as a Ph. D. student in Italy, have moved to industry. Of the new students, who entered the PE-lab, Claus and Andrei have already written their master's thesis (May 2006). Claus will continue in the PE-lab as a Ph. D. student.

To keep the research group vital we aim at an annual turnover of 20% of the workers, but as the facts above indicate, the actual turnover is higher. Even if this high turnover is an obstacle for high-profile long-term projects, we have been able to keep the CPH STL project alive. This project is the major initiative of the PE-lab in its whole existence. The goal of the CPH STL project is to develop an enhanced version of the Standard Template Library (STL), which is part of the ISO standard for the C++ programming language⁴. During the project period, 16 CPH STL reports (see the list of publications at the end of this report) were written which detail the design and implementation of various library components. This is a clear indication that the project is still going strong. The major challenge for the coming three years is to make an official release of the CPH STL.

The research done can be classified into three categories: experimental algorithmics, theoretical algorithmics, and software development. In our experimental studies our focus has been on three data structures: priority queues, ordered dictionaries, and unordered dictionaries. In our theoretical studies the efficiency of priority queues and double-ended priority queues has been investigated. Knowing that performance is a non-issue in many applications, we have also investigated problems that are related to programming languages and software development in general. The aim has been to reduce the development time, instead of saving CPU cycles. An example of this kind of research is the Benz⁵ measurement tool which we have developed to save our own time when carrying out the numerous experiments needed to guarantee the high quality of the developed components.

Briefly stated, many of the results obtained have been negative. Our experimental work reveals that most data structures presented in the recent algorithmic literature are only of academic interest, and not economical for practical use. The same observation is backed up by our theoretical work: there are many elegant—but complicated—data structures which can be manipulated within almost optimal bounds, but there is little hope to achieve the same bounds with simple data structures implemented in existing program libraries. In computing, there seems to be a tradition that negative results are not published—or allowed to be published, which is one reason why many of our papers have so far only appeared in our own report series.

⁴ British Standards Institute, *The C++ Standard: Incorporating Technical Corrigendum No. 1*, 2nd Edition, John Wiley & Sons. Ltd (2003).

⁵ For further details, see <http://www.cphstl.dk/WWW/tools.html>.

During the project period the PE-lab organized the following workshops, summer schools, and doctoral courses:

- 4th STL Workshop⁶, 26 May 2003
- Summer School on Experimental Algorithmics⁷, 5-7 July 2004
- 9th Scandinavian Workshop on Algorithm Theory⁸, 8-10 July 2004
- 5th STL Workshop⁹, 21 June 2005
- Course on generative software development¹⁰, 30 Jan. – 3 Feb. 2006.

In 2004, Scandinavian Algorithm Week was organized in collaboration with three Danish universities, the University of Copenhagen being the main organizer. The 9th Scandinavian Workshop on Algorithm Theory and its satellite events attracted more than 120 algorithmists and computer scientists to Denmark, and about 80 talks were given during the six days. A longer account about this event can be found from associated web pages¹¹.

Main results

The organization of the laboratory is project-oriented. In this section we give an overview of the projects and the results obtained in these projects. In connection with each project, the investigators involved are mentioned and the scientific output is given in the form of references to the list of publications provided at the end of this report. The publications are divided into three categories: P) scientific papers, R) technical reports, and T) theses supervised by the principal investigator.

Benz measurement tool

The first version of Benz measurement tool was released at the beginning of 2003 on the internet. The tool was developed to make it simpler to measure the execution time of programs. This has been a great success; before it took days to make an experiment but now it can be done in a few hours. After the first release the tool has been improved and as a result it can now also measure the number of instructions, number of cache misses, and the number of page faults.

[Investigators: Christian, Jacob, Jyrki; Output: R1]

Cache-oblivious algorithms in practice

The ideal cache model was introduced in 1999, and since then several cache-oblivious algorithms have been developed. In the previous theoretical work

⁶ <http://www.diku.dk/forskning/performance-engineering/News/All-news/2003.05.25-19:22:24.html>

⁷ <http://www.diku.dk/forskning/performance-engineering/Sommerskole/>

⁸ <http://swat.diku.dk/>

⁹ <http://www.diku.dk/forskning/performance-engineering/News/All-news/2005.05.31-14:59:00.html>

¹⁰ <http://www.diku.dk/forskning/performance-engineering/Generative-software-development/>

¹¹ <http://swat.diku.dk/press-release.html>

<http://www.diku.dk/forskning/performance-engineering/News/All-news/2004.09.10-13:13:57.html>

the goal has been to develop algorithms that perform optimally on all memory levels. We have tested the newly developed methods for classical searching and sorting problems. Our results have predominantly been negative: The new methods are often far slower than methods developed for the RAM model, if the problem instances are being solved in main memory, and slower than methods developed for the external-memory model, if the problem instances are massive thereby making disk access necessary. A considerable research effort will be necessary to improve these methods before they can be used in practical applications.

[Investigators: Andrei, Frederik, Jesper, Søren; Output: [R3](#), [T1](#), [T3](#)]

The truth about heaps

As far as we know, d -ary heaps ($d \in \{2, 3, 4\}$) are the only data structures that fulfil the strict complexity bounds given for heap operations in the C++ standard⁴. As to the practical efficiency of various forms of heaps, recent articles give conflicting results. In our experiments we have compared almost every known variant of multiary heaps published after Williams' article from 1964. Our experiments show that it is very difficult to compete with Williams' binary heaps. Our conclusion is that there has not been any progress within this field for the last 40 years.

[Investigators: Claus, Fabio, Jyrki; Output: [R5](#)]

Interactive constraint satisfaction

Many practical problems can be modelled as constraint satisfaction problems, where the task is to assign values to variables subject to a set of constraints. The problem is NP-hard and the interactive variant, where the problem needs to be solved online after modifications of the existing constraints, is computationally even more demanding. Because of these theoretical obstacles it has been surprising to see how large problem instances one can solve interactively. The work has been carried out in cooperation with Array Technology A/S.

[Investigator: Jeppe; Output: [T2](#)]

Generation of random data structures

Algorithms for generating random binary search trees and random binary heaps have been investigated. These algorithms work correctly even if the input provided for the generation is arbitrary. The motivation for this research has been to transform old algorithms, which are efficient if the input is random, to randomized algorithms where the randomization is made within the algorithms themselves and the efficiency is guaranteed without making any assumptions on the distribution of input elements.

[Investigators: Jyrki, Tomi; Output: none yet]

The cost of iterator validity

An iterator can be used to iterate over a collection of elements stored in a data structure, or simply to access the element pointed to. An iterator and the element pointed to live in a close symbiosis; when the element is moved, the iterator may become invalid if it is not updated accordingly. A data structure is said to provide iterator validity if the iterators to its elements are kept valid at all times independent of the movements of elements. As to the cost of keeping iterators valid, it can be shown that for all fundamental data structures (linked lists, resizable arrays, unordered dictionaries, ordered dictionaries, and priority queues) an implementation exists that supports 1) bidirectional iterators, 2) keeps the iterators valid under modifications, 3) executes all iterator operations in constant time in the worst case, and 4) uses linear space on the number of elements stored. For random-access iterators the problem appears to be more difficult. In particular, for vectors the general modification operations `insert()` and `erase()`, to be supported by the C++ standard, are too strong. Efficient solutions under these strong modification operations can be provided too, though most of them are mainly of theoretical interest.

[Investigator: Jyrki; Output: [R9](#)]

An experimental evaluation of navigation piles

A navigation pile is a priority-queue structure proposed recently by Katajainen and Vitale [[P1](#)]. It has many nice theoretical features, i.e. in many cases it can be used as a substitute for a binomial queue even if it is only a binary tree. We have evaluated the practical efficiency of navigation piles and made several interesting findings like the two mentioned below:

- 1) Pointer-based navigation piles have difficulties in matching the performance of classical heaps that are implicit data structures. However, the difference is not that big, and when the elements being manipulated are large, piles are faster than heaps.
- 2) Compact navigation piles that rely on bit packing can be hopelessly slow on contemporary computers. This made us to conclude that most packed data structures and algorithms relying on packing have very little practical relevance.

[Investigators: Claus, Jyrki; Output: [R16](#), [T6](#)]

The comparison complexity of priority-queue operations

A priority queue is an abstract data type that should be provided as a standard library component. In spite of its practical importance, it has long been an open problem how to achieve the best possible bounds—taking into account constant factors—for various priority-queue operations. We have been able to devise a realization of a priority queue which has optimal performance when the operations to be supported are `insert()`, `find-min()`,

delete-min(), and delete(); and almost optimal performance if in addition decrease() is to be supported.

[Investigators: Amr, Claus, Jyrki; Output: [R8](#), [T6](#)]

A critical analysis of randomized data structures

The main virtue of randomized algorithms is that they often offer good average-case performance without any assumptions about the distribution of the elements manipulated. In spite of this, randomized data structures—in our study randomized search trees—are normally analysed under the assumption that all elements are accessed with equal probability. This gives an overly optimistic picture of their performance. According to our theoretical and experimental analysis, the expected worst-case behaviour of randomized search trees is much worse than that of their best deterministic counterparts.

[Investigators: Hervé, Jyrki, Tomi; Output: none yet]

Building heaps faster

The construction of a heap is a classic data-structuring problem. For all practical purposes the problem has been solved satisfactorily, but as to its comparison complexity there is still a gap between the best known lower bounds and upper bounds. Several conjectures have been made about the optimal behaviour in the worst case and in the average case. We have tried hard to disprove the stated conjectures, and have been able to make some progress in the area of average-case algorithms.

[Investigators: Jyrki; Output: none yet]

Meldable priority queues

The goal in this research was to develop efficient priority queues that support operations find-min(), insert(), decrease(), delete(), and meld(). That is, this work was done with the aim of finding alternatives to Fibonacci heaps and run-relaxed heaps. Our focus was on simplicity as well as on the worst-case comparison complexity of delete() under the assumption that the first three operations have the worst-case cost of $O(1)$. Currently, the best bound achieved by us is $\lg n + O(\lg \lg n)$ element comparisons. Three different data structures guaranteeing this performance have been developed: two-tier relaxed heap, two-tier pruned binomial queue, and relaxed weak queue. These were the first data structures which achieve the bound $\lg n + o(\lg n)$ for the number of element comparisons performed by delete(). Of the three data structures, the last is the most space efficient requiring $4n + O(\lg n)$ words of storage, in addition to the n elements stored.

[Investigators: Amr, Claus, and Jyrki; Output: [R10](#), [R11](#)]

Double-ended priority queues

In this study the comparison complexity of double-ended priority-queue operations was investigated. In addition to the normal priority-queue operations, a double-ended priority queue supports both `find-min()` and `find-max()`. Two general data-structural transformations were described which transform our efficient priority queues into double-ended priority queues. The resulting double-ended priority queues are the fastest among all known double-ended priority queues. Due to their generality, the data-structural transformations applied are of independent interest.

[Investigators: Amr, Claus, and Jyrki; Output: none yet]

Experimental analysis of priority queues

The practical efficiency of various priority-queue structures was studied. The data structures considered include a navigation pile and a local heap which both had been devised by the members of the PE-lab. Several novel aspects about the representation of priority queues were considered:

- 1) How to make priority queues fully dynamic?
- 2) How to provide referential integrity?
- 3) How to improve the cache behaviour?
- 4) What is the relevance of pointer-based data structures?
- 5) What is the relevance of the compaction of data structures?

Previously, these implementation issues had been studied sparsely, so our analyses provide a considerable increase in knowledge in this area. Furthermore, the results obtained with local heaps were encouraging.

[Investigators: Claus, Fabio, and Jyrki; Output: [R14](#), [R16](#), [T5](#), [T6](#)]

Experimental analysis of hash tables

The CPH STL is a high-performance C++ Standard Template Library implementation. In this project, we designed a modular framework for standard-compliant hash tables targeted for inclusion in the CPH STL. Within this framework, hash tables based on three different approaches, two of which were inspired by Per-Åke Larson's dynamic hashing, were implemented. The implementations were compared to existing hash-table implementations in exhaustive synthetic benchmarks. The conclusion was that none of the approaches were universally ideal, but that all had strong points, which made a case for our modular hash-table design. A few common hash-table optimizations were analysed, among them storing the results of hash-function computations, which greatly improved the cache performance of the tables, and the asymptotic running time when variable-length keys were used.

[Investigators: Anders and Jørgen; Output: [P4](#), [R12](#)]

Publications

Scientific papers

- [P1] Jyrki Katajainen and Fabio Vitale, Navigation piles with applications to sorting, priority queues, and priority dequeues, *Nordic Journal of Computing* **10** (2003), 238–262.
- [P2] Hervé Brönnimann, John Iacono, Jyrki Katajainen, Pat Morin, Jason Morrison, and Godfried Toussaint, Space-efficient planar convex hull algorithms, *Theoretical Computer Science* **321** (2004), 25–40.
- [P3] Torben Hagerup and Jyrki Katajainen (Editors), *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory*, *Lecture Notes in Computer Science* **3111**, Springer-Verlag (2004).
- [P4] Jørgen Havsberg Seland and Anders Thøgersen, Custom-built hash tables, submitted (2006).

Technical reports

The ISSN of the CPH STL Report Series is 1602-1150. All CPH STL reports are available online at <http://www.cphstl.dk/>.

- [R1] Christian Ulrik Søttrup and Jakob Gregor Pedersen, [CPH STL's benchmark værktøj](#), CPH STL Report **2003-1**, Web document (2003), 28 pp.
- [R2] Jakob Sloth, Morten Lemvig, and Mads Kristensen, [Sortering i CPH STL](#), CPH STL Report **2003-2**, Web document (2003), 117 pp.
- [R3] Andrei Josephsen, [Implementation of a cache-oblivious search tree for the CPH STL](#), CPH STL Report **2003-3**, Web document (2003), 35 pp.
- [R4] John Juul Jensen, [Persistent storage framework](#), CPH STL Report **2003-4**, Web document (2003), 74 pp.
- [R5] Claus Jensen, Jyrki Katajainen, and Fabio Vitale, [An extended truth about heaps](#), CPH STL Report **2003-5**, Web document (2003), 16 pp.
- [R6] Stephan Lynge, [Implementing the AVL-trees for the CPH STL](#), CPH STL Report **2004-1**, Web document (2004), 60 pp.
- [R7] Mads D. Kristensen, [Vector implementation for the CPH STL](#), CPH STL Report **2004-2**, Web document (2004), 85 pp.
- [R8] Amr Elmasry, Claus Jensen, and Jyrki Katajainen, [A framework for speeding up priority-queue operations](#), CPH STL Report **2004-3**, Web document (2004), 31 pp.
- [R9] Jyrki Katajainen, [Project proposal: A meldable, iterator-valid priority queue](#), CPH STL Report **2005-1**, Web Document (2005), 37 pp.
- [R10] Amr Elmasry, Claus Jensen, and Jyrki Katajainen, [Relaxed weak queues: an alternative to run-relaxed heaps](#), CPH STL Report **2005-2**, Web Document (2005), 23 pp.
- [R11] Amr Elmasry, Claus Jensen, and Jyrki Katajainen: On the power of structural violations in priority queues, CPH STL Report **2005-3**, Web Document (2005), 19 pp.
- [R12] Anders Thøgersen and Jørgen Havsberg Seland, [Linear hashing based hash tables for the CPH STL](#), CPH STL Report **2005-4**, Web Document (2005), 95 pp.
- [R13] Jyrki Katajainen, [Research proposal: Generic programming—algorithms and tools](#), CPH STL Report **2005-5**, Web Document (2005), 9 pp.

- [R14] Claus Jensen, Jyrki Katajainen, and Fabio Vitale, [Experimental evaluation of local heaps](#), CPH STL Report **2006-1**, Web Document (2006), 25 pp.
- [R15] Hervé Brönnimann and Jyrki Katajainen, [Efficiency of various forms of red-black trees](#), CPH STL Report **2006-2**, Web Document (2006), 13 pp.
- [R16] Claus Jensen and Jyrki Katajainen, [An experimental evaluation of navigation piles](#), CPH STL Report **2006-3**, Web Document (2006), 16 pp.
- [R17] Jyrki Katajainen (Editor), [Project practical data structures and algorithms: Final report](#), CPH STL Report **2006-4**, Web Document (2006), 9 pp.

Theses supervised by the principal investigator

Most of the theses are available online; for further details, see the home page of the PE-lab.

- [T1] Jesper Holm Olsen and Søren Christian Skov, [Cache-oblivious algorithms in practice](#), Department of Computing, University of Copenhagen, Copenhagen (2002), iv+127 pp.
- [T2] Jeppe Nejsum Madsen, [Methods for interactive constraint satisfaction](#), Department of Computing, University of Copenhagen, Copenhagen (2003), vi+125 pp. (This thesis was acknowledged as the best M.Sc. thesis in Denmark in 2003 by *Dansk Selskab for Datalogi*.)
- [T3] Frederik Rønn, [Cache-oblivious searching and sorting](#), Department of Computing, University of Copenhagen, Copenhagen (2003), xi+158 pp.
- [T4] Christopher Derek Curry, [Reengineering a university department](#), Department of Computing, University of Copenhagen, Copenhagen (2004)
- [T5] Fabio Vitale, [An investigation into efficient priority queues](#), Computer Science Department, University of Insubria, Varese (2006), viii+77 pp. (Supervised jointly with Paolo Massazza)
- [T6] Claus Jensen, [Theoretical and practical efficiency of priority queues](#), Department of Computing, University of Copenhagen, Copenhagen (2006), vi+82 pp.
- [T7] Andrei Bjarke Moskvitin Josephsen, [Sekventiel lagringsstruktur og dets anvendelser](#), Department of Computing, University of Copenhagen, Copenhagen (2006), iii+60 pp.

On behalf of the project group

Copenhagen, 31 May 2006

Jyrki Katajainen
Assoc. Prof., Ph. D.