Research proposal: Practical data structures and algorithms

Institution:	Department of Computing, University of Copenhagen (DIKU)
Project duration:	1.1.2003–31.12.2005
Principal investigator:	Jyrki Katajainen, Assoc. Prof.
Other investigators:	Jeppe Nejsum Madsen, B. Sc., Array Technology A/S
	Jesper Holm Olsen, B. Sc.
	Frederik Rønn, B. Sc.
	Søren Christian Skov, B. Sc.
	Fabio Vitale, exchange student, University of Insubria

Background

The research done on algorithmics in the Department of Computing at the University of Copenhagen[1] can be divided into three overlapping categories: optimization, theoretical algorithmics, and experimental algorithmics. Most algorithmists at the department, including the principal investigator, are active in all these areas. At the end of 1998, the principal investigator and his graduate students received a grant from the Danish Natural Science Council (under contract 9801749, project Performance Engineering[2]), which was used to found the performance engineering laboratory[3] (PE-lab) at the department. With this initiative we wanted to stimulate the research in the area of experimental algorithmics. The foundation of the laboratory was an immediate reaction to opinions — similar to ours — presented at various discussion forums[4] that the empirical approach should be pursued more consciously and more rigorously in algorithmics as well as in computing in general.

In our undergraduate education on algorithmics we use the book by Cormen et al.[5], which is an excellent piece of work, presenting many important algorithms, proving their correctness, and analysing their theoretical efficiency in a rigorous manner. The analysis of algorithms is based on the

[1] http://www.diku.dk/

[2] http://www.cphstl.dk/Report/SNF-slutrapport-2002/cphstl-report-2002-5.pdf

[3] http://www.diku.dk/research-groups/performance-engineering/

[4] See, e.g., [J. N. Hooker, Needed: An empirical science of algorithms, *Operations Research* **42** (1994), 201–212], [Walter F. Tichy, Paul Lukowicz, Lutz Prechelt, and Ernst A. Heinz, Experimental evaluation in computer science: A quantitative study, *The Journal of Systems and Software* **28**,1 (1995), 1–18], or [Walter F. Tichy, Should computer scientists experiment more? 16 excuses to avoid experimentation, *Computer* **31**,5 (1998), 32–40].

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 2nd Edition, The MIT Press (2001).

random-access-machine (RAM) model popularized by Aho et al.[6] The instruction set of a RAM resembles a lot the instruction set of the EDSAC computer[7] built in England at the end of the 1940s. However, the computer architecture has changed a lot during the last 50 years — today memories are hierarchical and processors are semiparallel and pipelined. In contemporary computers one inevitably faces the limit of the speed of light: it takes longer time to access information that is far away from the processor(s) than to access information already stored in registers.

When all this is said, it is easy to understand that not all algorithms presented, for example, in the book by Cormen et al., or in the computing literature in general, are practical. As a concrete example let us take the task of sorting n integers. It has been shown that this task can be carried out using $O(n \log_2 \log_2 n)$ instructions on a RAM[8], whereas all classical sorting algorithms require $\Omega(n \log_2 n)$ instructions. The new method was even claimed to be practical[9], but the experiments carried out in our research group[10] show the opposite: the new sorting algorithm relies heavily on random-access capabilities of the memory system resulting in a poor cache behaviour. Our best implementation of the new method was a factor of 7–8 times slower than a quicksort-based method[11] available in the Silicon Graphics Inc. (SGI) implementation[12] of the Standard Template Library.

The goal in the research, for which we apply support, is

1) to find new practical algorithms,
2) to make old impractical algorithms practical if possible,
3) to improve the performance of old practical algorithms, and
4) to make the benchmark results publicly available so that the boundary between practical and theoretical algorithms becomes better known among practitioners.

The ultimate goal is to make the field of experimental algorithmics more like a science, what it is not at the moment.

The total funding applied is 480 000 DKK, to be distributed over the years 2003, 2004, and 2005. Among the most important budget items are the costs associated with participation in conferences and meetings, the

[6] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Company (1974).

[7] M. V. Wilkes and W. Renwick, The EDSAC, *Report of a Conference on High Speed Automatic Calculating Machines*, University Mathematical Laboratory, Cambridge (1950), 9–11. Reprinted in Brian Randell (ed.), *The Origins of Digital Computers: Selected Papers*, 3rd Edition, Springer-Verlag, Berlin/Heidelberg (1982), 417–421.

[8] Arne Andersson, Torben Hagerup, Stefan Nilsson, and Rajeev Raman, Sorting in linear time?, *Journal of Computer System Sciences* **57** (1998), 74–93

[9] Stefan Nilsson, The fastest sorting algorithm?, *Dr. Dobb's Journal* **311** (2000), 38–45

[10] Morten Nicolaj Pedersen, A study of the practical significance of word RAM algorithms for internal integer sorting, M. Sc. Thesis, Department of Computing, University of Copenhagen (1999), ii+74 pp. 77 app. pp.

[11] David R. Musser, Introspective sorting and selection algorithms, *Software—Practice and Experience*, **27** (1997), 983–993.

[12] Silicon Graphics, Inc., Standard template library programmer's guide, Website accessible at `http://www.sgi.com/tech/stl/` (1993–2002).

reimbursement of costs borne by our guests, the organization of workshops, the purchase of relevant books and journals, and the acquisition of two workstations together with some supporting software.

Planned research

The main body of the research done in the PE-lab is basic research — both practical and theoretical, but recently some applied research has been done, too. The organization of the laboratory is project-oriented. In this section we describe three practical-minded projects already started. These should give a flavour of the research planned for the next three years.

Algorithmic issues in program library development

The Standard Template Library (STL) is an integrated part of the standard library for the C++ programming language[13]. In the CPH STL project, which was started in autumn 2000, the goal is to

- study and analyse existing specifications for and implementations of the STL to determine the best approaches to optimization,
- provide an enhanced edition of the STL and make it freely available on the Internet,
- provide cross-platform benchmark results to give library users better grounds for assessing the quality of different STL components,
- develop software tools that can be used in the development of component libraries, and
- carry out experimental algorithmic research.

For further information about the project, see the CPH STL website[14].

This development project is an important source of interesting research problems. We would like to mention three recent results achieved in our research group.

1) In 1968 (as written by Woodall[15]) Robson[16] proved that no online algorithm for dynamic storage allocation can be competitive. After since the problem has widely been considered an unsolvable one. By relaxing some of the assumptions, on which Robson's lower bound argument relies, Katajainen[17] was able to prove that a competitive online algorithm actually exists. After 40 years of standstill this result opens new avenues for dynamic storage allocation.

---

[13] International Organization for Standardization (ISO), *ISO/IEC 14882: Standard for the C++ Programming Language* (1998).

[14] `http://www.cphstl.dk/`

[15] D. R. Woodall, The bay restaurant — a linear storage problem, *American Mathematical Monthly* **81** (1974), 240–246.

[16] J. M. Robson, An estimate of the store size necessary for dynamic storage allocation, *Journal of the ACM* **18** (1971), 416–423.

[17] Jyrki Katajainen, A competitive online algorithm for dynamic storage allocation, in preparation (2002).

2) In many places of the C++ standard the iterator validity is mentioned, but the requirements stated for various data structures are quite arbitrary. For a programmer — to whom the iterator validity is important — it would be easier to remember if the rule was simply: an operation that modifies the contents of a container does not invalidate the iterators of elements that are not involved in this operation. Up to until recently it was unknown what is the cost incurred by this rule for various container classes. Katajainen[18] proved tight bounds for the cost of iterator validity. Actually, this is surprisingly cheap, so the results provide interesting implementation challenges for the developers of the CPH STL.
3) Sorting is one of the most studied problems in algorithmics[19]. Most general-purpose sorting algorithms require $\Theta(n \log_2 n)$ element comparisons and $\Theta(n \log_2 n)$ element moves to sort a sequence of n elements. Katajainen and Vitale[20] proved that there exists a sorting method that performs at most $n \log_2 n + 0.59n$ element comparisons and only at most $2.5n$ element moves if we allow the usage of $4n$ extra bits in addition to the elements being sorted. Their techniques can also be applied in the implementation of priority queues and priority deques.

Since the start of the project, 24 CPH STL reports have been written, which detail the design of library components and supporting software. These reports are available via the CPH STL website. The main concern in the CPH STL project is the efficiency of the library components and the extension of the ideas to other domains. We will not mention any concrete research problems here but refer to the CPH STL publications and presentations, where many of the problems have been described in detail. The tool building is also important, as we wrote in an earlier application[21] but, since a grant from the Danish Natural Science Council cannot be used for the salaries of research assistants, support for this activity will be applied via other sources.

Cache-oblivious algorithms in practice

In recent years the processor speed has doubled about every 18 months, whereas memory-access time has not improved so fast. As the speed of processors increases more rapidly than the speed of memories, the optimal use of caches becomes more and more relevant. In particular, the classic measure of the runtime complexity assumes that memory accesses are equal in cost, but this is not true on contemporary computers having multilevel

[18] Jyrki Katajainen, The cost of iterator validity, in preparation (2002).
[19] Donald E. Knuth, *Sorting and Searching*, *The Art of Computer Programming* **3**, 2nd Edition, Addison Wesley Longman (1998).
[20] Jyrki Katajainen and Fabio Vitale, Navigation piles with applications to sorting, priority queues, and priority deques, CPH STL Report **2002-7**, Web document (2002), 22 pp.
[21] Jyrki Katajainen (Editor), Research proposal: Software tools for program library development, CPH STL Report **2001-15**, Web document (2001), 7 pp.

memories. In theory community, there has been some attention towards *cache-aware algorithms* and *cache-oblivious algorithms* which exploit the memory hierarchy in an efficient fashion.

According to Prokop[22] an algorithm is *cache-oblivious* if no program variable dependents on hardware configuration parameters, i.e., the size of caches or the size of cache blocks. This means that a cache-oblivious algorithm that uses a cache optimally will do this regardless of the underlying cache parameters, and this will be the case at each cache level. Therefore, a cache-oblivious algorithm ideally offers the advantages of simplicity and portability over a cache-aware algorithm for which hardware configuration parameters must be adjusted for each computer platform separately.

The STL is a widely used program library and has potential users on a variety of different platforms. Incorporating cache-oblivious techniques in the STL might guarantee a certain (optimal) cache performance and would indeed ease the portability of programs. As part of the CPH STL project[14] the potential use of cache-oblivious algorithms and data structures will be studied. Cache-oblivious versions of most STL algorithms and data structures will be implemented and compared to existing implementations available, e.g., at the SGI STL[12]. The main goal is to explore how alternative implementations of these algorithms perform on contemporary computers to see whether the cache-oblivious approach is viable in practice.

Methods for interactive constraint satisfaction

A large variety of problems in artificial intelligence and in other areas of computing can be viewed as a special case of the constraint satisfaction problem. Examples include machine vision, belief maintenance, scheduling, and circuit design. Constraint satisfaction problems are generally hard to solve. We have proved[23] that the decision version of a CSP is NP-complete, so all known solution methods have exponential worst-case performance.

A great deal of the research in constraint satisfaction has focused on algorithms which, given a constraint network as input, automatically finds a solution. This is useful in applications where, once the problem has been formulated as a constraint network, no user interaction is required. One such example is planning and scheduling (e.g., find a plan/schedule which satisfies the constraints). But in many applications user interaction is required to find a solution. An example is when the user guides the assignment of values to variables. This interactivity restricts the amount of time that can be used for the computations between the choices made by the user. In order to solve a CSP interactively, we therefore have to circumvent the intractability in some way.

---

[22] Harald Prokop, Cache-oblivious algorithms, M. Sc. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (1999).

[23] Jeppe Nejsum Madsen, Algorithms for compressing and joining relations, CPH STL Report **2002-1**, Web document (2002), 101 pp.

The topic of this project is algorithmic methods for solving constraint satisfaction problems interactively. The treatment is restricted to cover a subset of CSPs where the problem can be divided into two parts:

1) a static part, where the constraints do not change between user interactions, and
2) a dynamic part, where the user can influence the solution by adding additional constraints to the problem.

This restricted subset contains still many interesting CSP problems of practical relevance.

Budget

Budget item	First year	Three years
travelling expenses	50 000 DKK	150 000 DKK
visitors	50 000 DKK	150 000 DKK
workshops	5 000 DKK	15 000 DKK
two workstations	50 000 DKK	50 000 DKK
software	20 000 DKK	20 000 DKK
literature	5 000 DKK	15 000 DKK
Sum	180 000 DKK	400 000 DKK
overhead 20%	36 000 DKK	80 000 DKK
In total	216 000 DKK	480 000 DKK

On behalf of the research group

Copenhagen, 30 September 2002

Jyrki Katajainen
Assoc. Prof., Docent, Ph. D.