Dagstuhl Seminar

No. 02371, Report No. 353

08.-13.09.2002

Experimental Algorithmics

J. Bentley (Avaya Labs, USA)

R. Fleischer (Hong Kong U of Science and Technology)

B. Moret (U New Mexico, USA)

E. M. Schmidt (Aarhus U, Denmark)

Contents

1 Summary **4**

2 Program **5**

Monday, September 11 . . . 5
Tuesday, September 12 . . . 5
Wednesday, September 13 . . . 6
Thursday, September 14 . . . 6
Friday, September 15 . . . 7

3 Abstracts **8**

John Bentley, Both Sides Now: Tales from Three Decades of Experiments on Algorithms . . . 8
Tetsuo Asano, Distribute Points Uniformly: Applications to Digital Halftoning . . . 8
Matthias Stallmann, The Importance of Instance Classes in Experimental Algorithm Evaluation . . . 9
Irene Finocchi, Trading off colors and rounds in distributed vertex coloring algorithms . . . 9
Peter Sanders, A Practical Minimum Spanning Tree Algorithm Using the Cycle Property . . . 10
Stefan Droste, Complexity of Black-Box Optimization . . . 10
Michael Jünger, Simple and Efficient Bilayer Cross Counting . . . 11
Mike Fellows, Parameterized Complexity: the Main Ideas and Connections to Practical Computing . . . 11
Ulrike Stege, Efficient implementations via combining tractable parameterizations . . . 12
Andrew V. Goldberg, A Practical Shortest Path Algorithm with Linear Expected Time . . . 12
Adam L. Buchsbaum, Fast Prefix Matching of Bounded Strings . . . 13
Karen Aardal, Reformulation and algorithm performance . . . 13
Tandy Warnow, Phylogenetic Reconstruction . . . 13
Knut Reinert, Algorithms for High-throughput Comparison of Peptide Expression using Liquid Chromatography and Mass Spectrometry 14
Olaf Delgado-Friedrichs, Algorithms and Experiments in Theoretical Crystal Chemistry . . . 14
J. Ian Munro, Experimenting with Grammar Based Compression for Data Warehousing Applications . . . 15
Robert E. Bixby, Solving Linear and Integer Programs . . . 16
Rolf Fagerberg, Dealing with the Memory Hierarchy the Cache-Oblivious Way . . . 16
Robert Sedgewick, A few thoughts on the analysis of graph algorithms 17

Jörg-Rüdiger Sack, Parallel implemention of geometric shortest path algorithms 17
Christos Zaroliagis, Using Multi-Level Graphs for Timetable Information in Railway Systems 18
Matthias Müller-Hannemann, Constructing Inverter Trees in VLSI Design 18
Cynthia Phillips, Running Experiments on Parallel Machines 19
Maciej Liśkiewicz, On some Experiments with the LINPACK Benchmark on the SunFire 15K 19
Rudolf Fleischer, AAR — The Algorithm Animation Repository 20
Collected by E. D. Demaine, Open Problems Session 21

4 List of Participants 25

1 Summary

In September 2000, the Dagstuhl Seminar on *Experimental Algorithmics* brought together researchers from both worlds of algorithmics, theoreticians and practitioners. The main question of that seminar was whether and how theoretical and experimental research can co-exist as equal partners under the big roof of *algorithmics*. At the end, the nearly 50 participants agreed that the seminar had been very successful in bridging the two worlds, and they decided to summarize their findings in a Springer Lecture Notes volume *Experimental Algorithmics — The State of the Art*, which was published in 2002. They also agreed that they were still far away from their main goal, namely to characterize the different roles of theory and practice in the field of algorithmics, and that there should be another seminar on this topic in the future.

Therefore, another Seminar was held in September 2002 to further discuss the fundamental question of the value of experiments as opposed to purely theoretical analysis of algorithms. It was also discussed what happens when computer scientists (theoretical or practical) venture out in the world of real systems building and testing (networks, bioinformatics, natural language systems,...) where they usually meet non-CS engineers or physicists with their own methodological framework of experimental evaluation. Is there a fruitful interaction between CS and non-CS? Can we (the experimental algorithmicists) learn from them? Or they from us?

The aim of this workshop was to bring together three groups, more theoretical oriented researchers, more practical oriented researchers, and people working on real systems. In all, 44 researchers with affiliations in Australia, Austria, Canada, Denmark, Germany, Greece, Hong Kong, Italy, Japan, Spain, and the USA participated in the meeting. Four invited keynote speakers, Jon Bentley, Robert Bixby, Mike Fellows, and Tandy Warnow, gave one-hour position talks. The remaining 21 presentations given by participants of the meeting covered a wide range of topics in experimental algorithmics. The abstracts of most of these presentations are contained in this seminar report. One evening was reserved for an open problems session, also included at the end of this report.

As usual, Schloß Dagstuhl proved to be an excellent place to hold a great meeting, so we would not only like to thank the participants of the seminar for making this a very successful event but also the Dagstuhl staff for providing a friendly and stimulating working environment.

Jon Bentley
Rudolf Fleischer
Bernard Moret
Erik Meineche Schmidt

2 Program

Monday, September 11

9:00 Rudolf Fleischer: Introduction

9:15 Jon Bentley: Both Sides Now: Tales from Three Decades of Experiments on Algorithms

10:15 **Coffee Break**

10:45 Tetsuo Asano: Distribute points uniformly: applications to digital halftoning

11:15 Matt Stallmann: The importance of instance classes in experimental evaluation of algorithms

12:15 **Lunch**

15:30 **Coffee Break**

16:00 Irene Finocchi: Trading off colors and rounds in distributed vertex coloring algorithms

16:30 Peter Sanders: A practical minimum spanning tree algorithm using the cycle property

17:00 Stefan Droste: Complexity of black-box optimization

17:30 Michael Jünger: Simple and efficient bilayer cross counting

18:00 **Dinner**

Tuesday, September 12

9:00 Mike Fellows: Parameterized complexity

10:00 Ulrike Stege: Efficient implementations via combining tractable parameterizations

10:30 **Coffee Break**

11:00 Andrew Goldberg: A practical shortest path algorithm with linear expected time

11:30 Adam Buchsbaum: Fast prefix matching of bounded strings

12:15 **Lunch**

15:30 **Coffee Break**

16:00 Karen Aardal: Reformulation and algorithm performance

16:30 Bentley, Sedgewick, Moret, Bixby, Sanders, Demaine: Environemnts for experiments

18:00 **Dinner**

20:00 **Special Session**: Erik Demaine: Open Problems Session

Wednesday, September 13

9:00 Tandy Warnow: Phylogeny reconstruction

10:00 Knut Reinert: From peaks to peptids

10:30 **Coffee Break**

11:00 Olaf Delgado Friedrichs: Algorithms and experiments in theoretical crystal chemistry

11:30 Ian Munro: Grammar based compression of database information

12:15 **Lunch**

13:45 Hike to a nice place with cake, coffee, tea, ...

18:00 **Dinner**

Thursday, September 14

9:00 Robert Bixby: Solving linear and integer programs

10:00 Rolf Fagerberg: Dealing with the memory hierarchy — the cache-oblivious way

10:30 **Coffee Break**

11:00 Robert Sedgewick: Analysis of graph algorithms

11:30 Jörg-Rüdiger Sack: Geometric shortest paths

12:15 **Lunch**

15:30 **Coffee Break**

16:00 Stefan Näher: Automatic visualization of algorithms in `C++`

16:30 Christos Zaroliagis: Using multi-level graphs for time-table information in railway systems

17:00 Matthias Müller-Hannemann: Construction of inverter trees in VLSI design

17:30 Cynthia Phillips: Running experiments on parallel machines

18:00 **Dinner**

Friday, September 15

9:00 Maciej Liskiewicz: On some experiments with performance benchmark programs for parallel computers

9:30 Rudolf Fleischer: AAR — the Algorithm Animation Repository

10:00 Discussion: Useful tools

10:30 **Coffee Break**

11:00 Conclusions

12:15 **Lunch**

18:00 **Dinner**

3 Abstracts

Both Sides Now: Tales from Three Decades of Experiments on Algorithms

John Bentley

Some algorithmic discussions use the words “applied” and “experimental” as interchangeable. This talk argues that they are extremes on two (almost) independent dimensions. One’s *motivation* for solving a problem can range from “pure” (addressing the problem for its intrinsic merit) to “applied” (applying the solution to an external problem). Similarly, the *tools* for solving a problem range from “theoretical” (symbolic deduction) to “experimental” (computational induction). I illustrate these general ideas by describing two ten-year research programs: one dealt with with Travelling Salesman Problem (TSP), the other dealt with sorting. The motivations ranged over a continuum from very pure to very applied, with strong interaction between the two. The tools ranged from very theoretical to very experimental, also with strong interaction. Summary:

> Pure and applied, theory and experiment:
> all are worthy, all are fun.

(Joint with Bob Sedgewick, Doug McIlroy, David Johnson, Cathy McGeoch and a cast of dozens; the opinions are my own.)

Distribute Points Uniformly: Applications to Digital Halftoning

Tetsuo Asano

The problem of distributing points as uniformly as possible in a unit square has been studied for many years. In this talk we are interested in the discrete version of the problem, that is, given a lattice plane of size $N \times N$ and an integer $n < N^2$, choose n lattice points so that they are distributed uniformly. Among various criteria for measuring the uniformity we take the one minimizing the ratio of the maximum gap over the minimum gap, where the maximum gap is defined by the radius of the largest empty circles touching three points and the minimum gap is defined by the minimum pairwise distance. A naive algorithm for the problem is the one called ”incremental Voronoi insertion” in which points are inserted one by one by choosing appropriate lattice points near Voronoi vertices in the Voronoi diagram for the already selected points. For the application to digital halftoning we are required to achieve uniformity everytime when we insert points. We show this problem is closely related to the notions of discrepancy and dispersion studied in computational geometry. We present efficient heuristic algorithms together with some experimental results.

The Importance of Instance Classes in Experimental Algorithm Evaluation

Matthias Stallmann

Traditional experimental evaluation of algorithm performance relies heavily on random problem instances or, in some problem domains, on specific benchmarks. Random instances, though useful in analysis of average-case behavior, are not necessarily typical of practical instances, nor are they likely to present the range of challenges the algorithms need to overcome. On the other hand, benchmarks can be useful in pinpointing specific issues with which algorithms must contend, but they encourage designers to fine-tune algorithms for target instances sometimes to the detriment of better general techniques.
We propose the concept of an *instance class*, a set of instances on which, with respect to a given performance measure (execution time, solution quality, number of occurrences of some event, etc), "reasonable algorithms" should exhibit low variance. An instance class, when carefully defined for a problem domain, offers a way to obtain statistically significant results about algorithm performance. We demonstrate the use of naturally-defined and enlightening instance classes for problems as disparate as sorting, bigraph crossing minimization, and satisfiability. These classes have led to thorough evaluation of existing algorithms with sometimes surprising results, and to the development of significantly better algorithms.
(Joint work with Franc Brglez, Debu Ghosh, Robert Hochberg, and Xiao Yu Li.)

Trading off colors and rounds in distributed vertex coloring algorithms

Irene Finocchi

We report on the results of an extensive experimental evaluation of very simple, distributed, randomized algorithms for vertex coloring. We consider variants of

algorithms known from the literature, boosting them with a distributed independent set computation, and we show that some of them are extremely fast and very effective, thus being amenable to be used in practice.

(Joint work with Alessandro Panconesi and Riccardo Silvestri.)

A Practical Minimum Spanning Tree Algorithm Using the Cycle Property

Peter Sanders

We present a simple new algorithm for computing minimum spanning trees that is more than two times faster than the best previously known algorithms (for dense, "difficult" inputs). It is of conceptual interest that the algorithm uses the property that the heaviest edge in a cycle can be discarded. Previously this has only been exploited in asymptotically optimal algorithms that are considered to be impractical. An additional advantage is that the algorithm can greatly profit from pipelined memory access. Hence, an implementation on a vector machine is up to 13 times faster than previous algorithms. We outline additional refinements for MSTs of implicitly defined graphs and the use of the central data structure for querying the heaviest edge between two nodes in the MST. The latter result is also interesting for sparse graphs.

(Joint work with Irit Katriel and Jesper Träff.)

Complexity of Black-Box Optimization

Stefan Droste

In many engineering applications the objective function to be optimized can only be accessed by experiments or simulations. Furthermore, there is often no time or expertise to analyse the function and make a mathematical model. Hence, the objective function is in a black-box, i. e. can only be evaluated, but its parameters are not known. In this case black-box algorithms, general heuristics that do not use any parameters of the objective function, must be used. As a black-box contains less information than the input of a classical algorithm, the number of evaluations of the objective function, until an optimum is found, is a sensible quality measure for black-box algorithms.

Defining the black-box complexity of a function set according to this measure as the minimal number of evaluations every black-box algorithm must do in order to find an optimum of any function of the set leads to a new complexity measure.

It is uncomparable to classical algorithmic run-time complexity, as problems of high algorithmic complexity can be represented by function sets having low black-box complexity and vice versa. For the class of unimodal functions over $\{0,1\}^n$ (every non-optimal point has a Hamming neighbour with better function value) the black-box complexity can be shown to be exponential in n. Hence, no black-box algorithm can optimize every unimodal function with a polynomial number of queries.

(Joint work with Thomas Jansen, Karsten Tinnefeld, and Ingo Wegener.)

Simple and Efficient Bilayer Cross Counting

Michael Jünger

We consider the problem of counting the interior edge crossings when a bipartite graph $G = (V, E)$ with node set V and edge set E is drawn such that the nodes of the two shores of the bipartition are drawn as distinct points on two parallel lines and the edges as straight line segments. The efficient solution of this problem is important in layered graph drawing. Our main observation is that it can be reduced to counting the inversions of a certain sequence. This leads to an $O(|E|+|C|)$ algorithm, where C denotes the set of pairwise interior edge crossings, as well as to a simple $O(|E| \log |V_{\text{small}}|)$ algorithm, where V_{small} is the smaller cardinality node set in the bipartition of the node set V of the graph. We present the algorithms and the results of computational experiments with these and other algorithms on a large collection of instances.

(Joint work with Wilhelm Barth and Petra Mutzel.)

Parameterized Complexity: the Main Ideas and Connections to Practical Computing

Mike Fellows

The talk and the paper have two purposes:

1. To give an exposition of the main ideas of parameterized complexity, and
2. To discuss the connections of parameterized complexity and FPT techniques to the systematic design of heuristics and approximation algorithms.

Efficient implementations via combining tractable parameterizations

Ulrike Stege

We introduce the problem Profit Cover. For a given graph $G = (V, E)$ and an integer $p > 0$, the goal is to determine $PC \subseteq V$ such that the profit, $|E'| - |PC|$, is at least p, where E' are the by PC covered edges. We show that p-Profit Cover is a parameterization of Vertex Cover. We present a fixed-parameter-tractable (fpt) algorithm for p-Profit Cover that runs in $O(p|V| + 1.150964^p)$. We combine our algorithm for p-Profit Cover with an fpt-algorithm for k-Vertex Cover. We show that this results in a more efficient implementation to solve Minimum Vertex Cover than each of the algorithms independently. We also demonstrate the generality of our approach on the example of Planar Dominating Set.

(Joint work with Iris van Rooij, Alex Hertel, and Philipp Hertel.)

A Practical Shortest Path Algorithm with Linear Expected Time

Andrew V. Goldberg

The shortest path problem with nonnegative arc lengths is one of the fundamental optimization problems that is very important in practice. Algorithms for this problem have been studied since 1950's.

We present an algorithm for the problem based on the multi-level bucket data structure. Our algorithm has a linear expected time for a uniform arc length distribution. The worst-case running time of the algorithm is good, although not the best.

We also describe an efficient implementation of the algorithm and an experimental study evaluating performance of our implementation. For 32-bit input arc lengths, our implementation always takes time that is below 2.5 times the time to perform breadth-first search on the input graph. On non-pathological inputs, the time for our implementation is less than a factor of two away from the time for breadth-first search. This suggests that there is limited room for practical efficiency improvement.

Fast Prefix Matching of Bounded Strings

Adam L. Buchsbaum

Longest Prefix Matching (LPM) is the problem of finding which string from a given set is the longest prefix of another, given string. LPM is a core problem in many applications, including IP routing, network data clustering, and telephone network management. These applications typically require very fast matching of bounded strings, i.e., strings that are short and based on small alphabets. We note a simple correspondence between bounded strings and natural numbers that maps prefixes to nested intervals so that computing the longest prefix matching a string is equivalent to finding the shortest interval containing its corresponding integer value. We then present *retries*, a fast and compact data structure for LPM on general alphabets. Performance results show that retries outperform previously published data structures for IP look-up. By extending LPM to general alphabets, retries admit new applications that could not exploit prior LPM solutions designed for IP look-ups.

(Joint work with Glenn S. Fowler, Balachander Krishnamurthy, Kiem-Phong Vo, and Jia Wang.)

Reformulation and algorithm performance

Karen Aardal

We discuss a reformulation of an integer equality contstrained knapsack problem in terms of a lattice describing the integer null space. We give a sufficient condition under which this reformulation creates good directions for a tree search. Using the reformulation makes it possible to solve the problem several orders of magnitude faster than by brach-and-bound. This is illustrated by a computational study.

Phylogenetic Reconstruction

Tandy Warnow

The problem of reconstructing phylogenies (also known as evolutionary trees) is major in biology. There are many methods, and we examine their performance via simulation, with topological accuracy the most important criterion. Within the class of polynomial time methods we specifically compare three methods: Neighbor joining (NJ), greedy maximum parsimony (GMP), and a new method

we developed, called “$\mathrm{DCM_{NJ}} + \mathrm{MP}$”. This is the disk-covering method (DCM) applied to NJ on subproblems, and with MP as the selection criterion. We show how experimental performance studies led to the design of this method, although theoretical results gave us its basic structure. Our simulation study shows how each of these three methods is affected by parameters of the model tree: the evolutionary diameter, the deviation from from a molecular clock, and the number of taxa. All methods require longer sequences to achieve good performance, as these parameters increase, but GMP is the worst, then NJ, then $\mathrm{DCM_{NJ}} + \mathrm{MP}$. See http://www.cs.utexas.edu/users/tandy.

(Joint work with Bernard Moret of the Univ. of New Mexico.)

Algorithms for High-throughput Comparison of Peptide Expression using Liquid Chromatography and Mass Spectrometry

Knut Reinert

A stated goal of Proteomics is to provide a snapshot of the active proteins and their expression levels, in specific tissues under specific conditions and to differentiate this from other tissues. We describe algorithms for measuring and comparing expression levels of proteins that have been digested into peptides and separated and analyzed using liquid chromatography coupled with mass spectrometry in a high-throughput environment. The key step lies in the quick construction of *LC-MS maps*, i.e. in the reliable detection of all the peaks belonging to a peptide, the prediction of mono-isotopic peak m/z, and the prediction of the charge of the feature. This task is confounded by peaks that have low signal to noise, presence of chemical noise, and *interleaved* peaks of co-eluting peptides.

Algorithms and Experiments in Theoretical Crystal Chemistry

Olaf Delgado-Friedrichs

Many crystal structures can be conveniently interpreted as 1-skeleta of 3-dimensional periodic tilings. Given a tiling, the corresponding network or 3-periodic graph consisting of its vertices and edges is easy to extract. As it turns out, a large class of crystal networks can be derived from a surprisingly simple class of tilings. But is there any natural way to construct a unique tiling for each network? The mathematical theory of Delaney symbols has led to the development of data structures and algorithms which make it possible to investigate this and related questions experimentally.

Experimenting with Grammar Based Compression for Data Warehousing Applications

J. Ian Munro

In a commercial data warehousing system, it was necessary to compress both relations and indices. While it was quite acceptable to retain a reasonably large (several megabyte) dictionary in main memory, decompression had to be both fast and operate on page sized chunks.
A grammar based technique was explored. A simplistic view of this approach is that of extending a Huffman code to extra characters by replacing instances of the substring "*ab*" with the new symbol $< ab >$. A grammar would thus grow as the data string shrank. Clearly the process can continue until we are left with a single character for the entire string, or, more likely, until either the size of the grammar together with the (Huffman compressed) string length is minimized or the grammar has grown to a permitted bound. Finding the best grammar under any of these criteria is NP-Hard, but heuristics are reasonably effective.
The approach of repeatedly adding the most frequently occurring pair was implemented and runs reasonably quickly. A more intuitively attractive heuristic is to choose the pair whose replacement will most reduce the expected length of the compressed string. This expected value is the length of the string (in the current alphabet) times the entropy of the current alphabet. Of course, as one adds new characters, the length of the (uncompressed) string length will decrease while the entropy will increase. The technical problem is that with r characters in the current alphabet, we choose the next character to add from a set of up to r^2 elements, each of whose effects change at each step. $\Theta(r)$ will definitely change relative to others, and many others may change. $\Theta(r)$ time to add the $r + 1$st character leads to a linear time algorithm overall, but re-examining all symbols at each step is unacceptable. The former takes a few minutes on a 3 megabyte sample, while the latter takes several hours and would seem to be much worse on the target range for input of 100 gigabytes. On sample data, the compression rates were respectively, 10% and 20% better than gzip for the grammar plus file and substantially better if we ignore the memory resident dictionary. A method performing partial updates to entropy changes was found to be almost as fast as the replacing the most frequent pair and almost as effective as the true entropy based approach. More importantly, page level decompression is easy under this scheme though not possible under a Lempel-Ziv approach on large sections.
The work is still exploratory, but seems promising, both for effective/efficient compression and for interesting work on the necessary data structures.

(Joint work with J-P Pretti, T. Snider and D. Toman and supported by Aruna PLC and Communications and Information Technology Ontario.)

Solving Linear and Integer Programs

Robert E. Bixby

Computational results where shown for linear programming claiming that LP algorithms had achieved over a 2000 fold speed improvement in the last 15 years. Combined with machine improvments over that period, the result was a total speed improvement of a factor of nearly 2 million!
The main focus of the talk was integer programming. Over the last several decades, from the early 1970s to as recently as 1998, the underlying solution technology in commercial mixed-integer programming codes remained essentially unchanged. This in spite of important advances in the theory, many of these advances having clear computational value. In the last several years, that situation has changed. The talked discussed some of the specific ways in which it had changed. Singnificant computational results were presented.

Dealing with the Memory Hierarchy the Cache-Oblivious Way

Rolf Fagerberg

Cache-obliviousness is a particular elegant way of designing algorithms using the multi-level memory hierarchy of modern computers efficiently. It was introduced in 1999 by Frigo, Leiserson, Prokop, and Ramachandran. We give examples of cache-oblivious algorithms, including double for-loops, dynamic programming, search trees, and sorting, and report on practical experiences with these. Our main message is that for a number of basic algorithmic problems there exist solutions which

- are simple,
- are theoretically I/0-efficient,
- adapts automatically to the specifics of the memory hierarchy,
- compete well with explicit (cache-aware) I/O algorithms.

(Joint work with Gerth S. Brodal, Riko Jacob, and Kristoffer Vinther.)

A few thoughts on the analysis of graph algorithms

Robert Sedgewick

Worst-case upper bounds are often not useful for predicting performance or for comparing graph-processing algorithms. By randomizing the order of input edges, we set up a situation where we can talk about expected performance of algorithms and particular graphs, which leads to a number of interesting research questions.

Parallel implemention of geometric shortest path algorithms

Jörg-Rüdiger Sack

Shortest path problems are among the fundamental problems studied in computational geometry and graph algorithms. They arise in application areas such as Geographic Information Systems and Robotics. In such applications, frequently, weighted metrics capturing the varying nature of a terrain (e.g., water, rock, forest) are more meaningful than the Euclidean metric. Considering weighted metric increases significantly the time algorithms execute and it increases the design complexity of algorithms solving such problems. Our approach has been to first design novel more efficient sequential algorithms and then to parallelize one of our practical algorithms.
Here we present the results of this parallelization effort. Our computational model is the distributed MIMD. Factors influencing the run-time include: algorithmic issues, data (size, partition, clustering, ...), machine related factors (topology, processors, interconnection strategies), and programming factors (programming styles, geometric primitives, type and implementation of communication libraries). Novel data partitioning and processor allocation schemes are introduced which maintain spatial proximity. Our experiments include tests varying the data partitioning strategy, data sizes, tile sizes, data distribution (such as clustering), and communication speed. To the best of our knowledge, this is the first parallel implementation of shortest path problems in these metric. Our experiments show that we achieve a good speedup on standard architectures with different communication/computation charateristics, incl. PCs interconnected by a cross-bar switch using fast ethernet and a state-of-the-art beowulf cluster with gigabit interconnect.

(Joint work with Mark Lanthier and Doron Nussbaum; testing by T. Wen and T. Guo.)

Using Multi-Level Graphs for Timetable Information in Railway Systems

Christos Zaroliagis

In many fields of application, shortest path finding problems in very large graphs arise. Scenarios where large numbers of on-line queries for shortest paths have to be processed in real-time appear for example in traffic information systems. In such systems, the techniques considered to speed up the shortest path computation are usually based on precomputed information. One approach proposed often in this context is a space reduction, where precomputed shortest paths are replaced by single edges with weight equal to the length of the corresponding shortest path. In this paper, we give a first systematic experimental study of such a space reduction approach. We introduce the concept of multi-level graph decomposition. For one specific application scenario from the field of timetable information in public transport, we perform a detailed analysis and experimental evaluation of shortest path computations based on multi-level graph decomposition. Finally, we discuss a different graph modeling of the timetabling problem and present preliminary experimental results.

(Joint work with Frank Schulz and Dorothea Wagner.)

Constructing Inverter Trees in VLSI Design

Matthias Müller-Hannemann

The construction of inverter trees becomes more and more important for timing optimization and reduced power consumption in physical design of modern chips. Inverter trees are signal trees where a signal has to be distributed from a source (an output pin of some circuit) to a number of sinks (input pins of other circuits). Feasible inverter trees have to respect several constraints (load constraints, parity constraints, and timing constraints). Moreover, these trees have to be embedded in a rectilinear fashion on the chip image avoiding blockages (preplaced macros and other circuits). Optimization goals are to maximize the slack and to minimize power consumption.

In this talk we present a computational study with instances from ASICS of IBM, present a slack-bound-based greedy-like clustering heuristic to build up the tree topology, and discuss several embedding and legalization strategies. Our computational results demonstrate that our code usually builds up inverter trees which are relatively close to the bounds and which compare favorably with a standard code currently used by IBM.

Running Experiments on Parallel Machines

Cynthia Phillips

We discuss a number of topics related to (parallel) tools for experimental analysis of algorithms and analyzing parallel codes and parallel system software. We begin with an example showing that even in pure application analysis, the obvious question may be the wrong question. We then summarize some features of the PICO (Parallel Integer and Combinatorial Optimizer) massively-parallel mixed-integer programming (MIP) code. In particular, we summarize using the AMPL-PICO interface for rapid development of 1) linear-programming-based approximation-algorithm codes (e.g. for a "one day" test) and 2) MIP codes that exploit combinatorial structure. We describe two open problems in the analysis of PICO: handling multiple levels and types of nondeterminism and computing certificates.
We describe the difficulties inherent in using actual hardware to compare algorithms for scheduling and node allocation. We explore the interaction between simulation and hardware-based experimentation.
Finally, we invite discussion on components of a full (parallel) experimental algorithmics workbench. We plan to build such a system for analysis of multiple types of optimization algorithms: combinatorial, stochastic global (e.g. evolutionary algorithms) and simulaton-based optimization algorithms.

On some Experiments with the LINPACK Benchmark on the SunFire 15K

Maciej Liśkiewicz

We report on some experiments with the standard LINPACK Benchmark on the SunFire 15K shared-memory parallel computer that we performed in May and June this year at the University of Lübeck. For the configuration: 72 UltraSPARC III CPUs and 72 GBytes of memory, we achieved the best machine performance with the *Sun Performance Library* $R_{max} = 59.17$ Gflop/s for $N_{max} = 91000$. To explain the poor efficiency $R_{max}/R_{peak} = 0.46$ (Sun announced the efficiency 0.80) we performed a sequence of tests to measure the scalability of the computer and we obtained astonishing results concerning the correctness of the solutions given by the program for some small numbers of threads. In these cases the *norm. resid* exceeded the value 10^{12}, thereby showing that the solutions are wrong.

AAR — The Algorithm Animation Repository

Rudolf Fleischer

We introduce the Algorithm Animation Repository (AAR), a refereed collection of algorithm animations, animation tools, and other animation related material that is currently developed at the HKUST. The goal of the AAR is to provide a platform where good algorithm animations can be published and thus be made accessible to a wider audience, in particular for teaching purposes. The AAR was founded at last year's Dagstuhl Seminar on Software Visualization, with the above mentioned co-authors serving as the founding Editorial Board, under the chairmanship of R. Fleischer.

(Joint work with Pierluigi Crescenzi, Nils Faltin, Chris Hundhausen , Stefan Näher, Guido Rössling, John Stasko, and Erkki Sutinen.)

Open Problems Session

Collected by E. D. Demaine

The following is a list of the problems presented on September 10, 2002 at the open-problem session of the 2nd Dagstuhl Seminar on Experimental Algorithmics held in Wadern, Germany.

Estimating Running Time: Easy Cases?
Robert Sedgewick
Princeton University
rs@cs.princeton.edu

How should we design an experiment to estimate the running time of a program as a function of n? In general, of course, this problem is unsolvable (cf. the halting problem). The idea here is to focus on a very restricted class of programs, and to focus on just estimating the coefficient of the (known) lead term, possibly with knowledge of the entire asymptotic expansion of the running time. One of the main questions here is whether it makes sense to run the program on several instances of the same (large) size, or to run the program on several instances all of different sizes, or with what distribution of sizes, etc.

Determining exactly which restricted class of programs makes sense is part of the open problem. An example of something that *should* be easy is insertion sort; there are many other natural candidates. By making some progress on problems with known solutions computed analytically by hand, we would hope to obtain techniques for estimating the solution for similar unknown problems. In particular, when we make a slight modification to an algorithm whose performance is well-understood, we might not be able to redo the analysis easily, but we can easily run empirical studies.

A few issues that arose in discussion: The entire functional form of the asymptotic running time might be necessary to get a good estimate even for the lead term; at least it may help eliminate noise. A particularly tricky aspect is when lower-order terms oscillate; in this case, we might bound the term by e.g. proving a theorem, and use this bound to estimate the lead term.

Intrinsically Hard Instances: How to Find?
Michael Fellows
University of Newcastle, Australia
mfellows@cs.newcastle.edu.au

How do we find intrinsically hard instances for NP-hard problems that defy all algorithms? What is the value of finding such instances for evaluating the performance of heuristics? In particular, the restricted domain of parameterized complexity may make this task easier, because of the tighter constraints it places on instances.

Three natural suggestions that came up during discussion:

1. Take a random example, kernelize (reduce while preserving the answer, a notion standard in parameterized complexity), and see how much of the instance is left. (Is a large kernel always "hard"?)
2. Internet-based competition ("gambling"). The idea is to run a "hard-instance stock market" which people (even kids) invest a small amount of money to have their examples considered; this is a sort of random parallel search driven by humans.
3. Reduction from hard 3SAT instances. A fair amount is known about hard 3SAT instances, and the reduction from 3SAT to graph 3-coloring doesn't blow up the size much.

Can intrinsically hard instances help us compare multiple implementations, as well as determine whether an implementation is "good enough"? One example discussed was the problem of graph 3-coloring. In this context, is the following conjecture true?

Hard puzzle conjecture: There exists an infinite sequence of 3-colorable graphs such that *every algorithm* (of constant size) performs poorly on *all* sufficiently large instances in the sequence.

Make LEDA Look Bad

Peter Sanders
Max-Planck-Institut für Informatik
sanders@mpi-sb.mpg.de

The 'Make-LEDA-Look-Bad' Contest challenges you to find difficult worst-case instances for two polynomial-time graph algorithms: general weighted matching and max-flow. Even more difficult is to develop a worst-case instance generator that creates an infinite family of difficult instances. The idea is to collect a good set of instances for benchmarking implementations of these algorithms.

See `http://www.mpi-sb.mpg.de/~schaefer/MLLB/` for more details.

Algorithm Sets

Jon Bentley
Avaya Labs Research
jbentley@avaya.com

Let's build algorithm sets analogous to chemistry sets, which allow kids to play and experiment with algorithms instead of chemicals. The idea is to have a classic set of experiments on algorithms, each of which has the following components:

1. Problem statement
2. Application it came from (for the really juicy problems)
3. Environment for kids to work with
   (a) Code for the algorithms
   (b) Testbed for exercising the algorithms
   (c) Animation so that they could see it work
   (d) Inputs
   (e) Generators to make more inputs
4. Classic form of the experiment
5. Discussion about the design of the experiment: why it was set up this way as opposed to various other ways, and how it was implemented.
6. Interaction between theory and experiments

Some candidates arose during the discussion:

1. Sorting (insertion sort, quicksort, etc.)
2. Binary search trees (random inserts, and then random inserts and deletes, an actual set of experiments that was active for over 10 years)
3. Longest common subsequence for DNA sequences (easily motivated to most age groups)
4. Bin packing
5. Traveling Salesman Problem
6. Minimum spanning tree
7. 2-coloring (for a younger audience)

"Little kids" might mean first-year graduate students, or undergraduates, or indeed little kids.

Why Are Solution Spaces So Lumpy?

Michael Fellows
University of Newcastle, Australia
mfellows@cs.newcastle.edu.au

There are several examples of problems whose solution spaces tend to be (but aren't universally) "lumpy" in practice, in the sense that many desired solutions are clustered together instead of being evenly distributed. Can we prove anything giving insight into why solution spaces are lumpy?

For example, with k-leaf spanning tree (is there a spanning tree with at least k leaves?), solutions seem to be clustered among the leaves of the height-k search tree. This solution structure has been exploited by Frank Dehne in some experiments where, by partitioning the search space into pieces and searching each in parallel, he seems to obtain solutions much faster. (Here the problem has already been kernel-reduced.)

Another example is Bill Cook's code for the Traveling Salesman Problem which picks 7 candidate tours out of a soup, takes their union, solves TSP exactly on that union, and adds the result to the soup. The union tends to be a graph with treewidth around 10, which makes TSP solvable exactly in a reasonable amount of time. But theoretically the treewidth is unbounded; perhaps the low treewidth is caused by lumpyness.

A few issues arose in discussion: Some insight might come from problems engineered to have unique solutions, because then there are "no lumps" (in an exploitable way—from another point of view, all solutions are lumped together). Additional light may be shed from the extensive study of 3SAT instances.

4 List of Participants - Dagstuhl Seminar 00371 (08.04.99; 17:37)

Date: 10.09.2000 - 15.09.2000
Title: Experimental Algorithms
webpage: `http://www.dagstuhl.de/DATA/Participants/00371.html`