

Online Library of Jeff Vitter's Papers

Jeffrey S. Vitter
Texas A&M University

November 9, 2009

This file is an index to an online library of several of my more recent papers and in some cases the overhead transparencies for talks. Most of the papers and talks deal with the design and analysis of algorithms and data structures. They are grouped according to specific topic area, roughly in chronological order. The first section lists some general papers, such as my new monograph on *Algorithms and Data Structures for External Memory* 2.71 and my book *Efficient Algorithms for MPEG Video Compression* 3.21. Click on whatever titles or topic areas interest you to get the papers you want. Some papers are listed in more than one topic area. For example, papers on I/O-efficient algorithms for geometric problems are listed in both the External Memory Algorithms section and the Computational Geometry section. And my newer work on entropy-compressed data structures is listed in both the Database section and the Data Compression section.

I encourage you to copy and distribute any of these papers for any noncommercial use, at no charge to anyone. However, if any money (beyond the actual cost of reproduction) is going to change hands, you need my written permission first.

My publications are stored in both gzip-compressed postscript format and in Adobe pdf format. Most web browsers (at least on UNIX machines) will display these formats automatically. If your browser doesn't, you may need to download the image tool Adobe Acrobat to view the documents in pdf format. For the gzip-compressed postscript files, you can download gunzip (UNIX, MacOS, or Windows) or Stuffit Expander (MacOS or Windows) to unpack them, and ghostview to display them. Alternatively, you can also access these files via anonymous ftp at ftp.cs.duke.edu in directory pub/jsv/Papers.

The document you're reading now, plus in addition the titles, bibliography references, and abstracts of the publications, is available in gzip-compressed postscript format (roughly 150 Kbytes) at <http://faculty.cse.tamu.edu/jsv/Papers/catalog.ps.gz> and Adobe pdf format (roughly 300 Kbytes) at <http://faculty.cse.tamu.edu/jsv/Papers/catalog.pdf>.

My full curriculum vitae listing all publications is available online in various formats. Many publications in my CV are not included below, but are available in hardcopy by email request to me at jsv@tamu.edu. If you find any errors or have any problems retrieving any items, please let me know by email.

Contents

1 SURVEYS AND MANUSCRIPTS	9
1.1 Average-Case Analysis of Algorithms and Data Structures	9
1.2 Arithmetic Coding for Data Compression	9
1.3 Report of the Working Group on Storage I/O for Large-Scale Computing	9
1.4 Efficient Algorithms for MPEG Video Compression	9

1.5	Miscellaneous Tips for Writing and Formatting	9
1.6	Algorithms and Data Structures for External Memory— <i>main reference!</i>	9
2	EXTERNAL MEMORY ALGORITHMS, I/O EFFICIENCY, AND DATABASES	9
2.1	The Input/Output Complexity of Sorting and Related Problems	9
2.2	I/O Overhead and Parallel VLSI Architectures for Lattice Computations	10
2.3	Algorithms for Parallel Memory I: Two-Level Memories	10
2.4	Algorithms for Parallel Memory II: Hierarchical Multilevel Memories	11
2.5	Large-Scale Sorting in Uniform Memory Hierarchies	11
2.6	Optimal Deterministic Sorting on Parallel Disks	11
2.7	Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies	12
2.8	Blocking for External Graph Searching	12
2.9	Indexing for Data Models with Constraints and Classes	12
2.10	External-Memory Computational Geometry	13
2.11	External-Memory Graph Algorithms	13
2.12	External-Memory Algorithms for Processing Line Segments in Geographic Informa- tion Systems	14
2.13	TPIE: Transparent Parallel I/O Programming Environment	15
2.14	I/O-Efficient Scientific Computation using TPIE	15
2.15	Efficient 3-D Range Searching in External Memory	15
2.16	Optimal External Memory Interval Management	16
2.17	Simple Randomized Mergesort on Parallel Disks	16
2.18	Report of the Working Group on Storage I/O for Large-Scale Computing	17
2.19	I/O-Efficient Algorithms and Environments	17
2.20	On Sorting Strings in External Memory	18
2.21	I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking . .	18
2.22	Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Search- ing Problems	19
2.23	Scalable Sweep-Based Spatial Join	19
2.24	Competitive Analysis of Buffer Management Algorithms for Parallel I/O Systems . .	20
2.25	Modeling and Optimizing I/O Throughput of Multiple Disks on a Bus	20
2.26	Scalable Mining for Classification Rules in Relational Databases	20
2.27	Data Cube Approximation and Histograms via Wavelets	21
2.28	Efficient Searching with Linear Constraints	21
2.29	Wavelet-Based Histograms for Selectivity Estimation	21
2.30	External Memory Algorithms and Data Structures: Dealing with Massive Data . . .	21
2.31	Efficient Bulk Operations on Dynamic R-trees	21
2.32	I/O-Efficient Dynamic Point Location in Monotone Subdivisions	22
2.33	Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets	22
2.34	On Two-Dimensional Indexability and Optimal Range Search Indexing	22
2.35	A Simple and Efficient Parallel Disk Mergesort	23
2.36	Online Data Structures in External Memory	23
2.37	External Memory Algorithms with Dynamically Changing Memory Allocations . . .	24
2.38	A Unified Approach for Indexed and Non-Indexed Spatial Joins	25
2.39	I/O-Efficient Algorithms for Problems on Grid-Based Terrains	25
2.40	Efficient Bundle Sorting	26
2.41	Efficient Flow Computation on Massive Grid Terrains	26

2.42	Distribution Sort with Randomized Cycling	26
2.43	Constrained Querying of Multimedia Databases: Issues and Approaches	27
2.44	CAMEL: Concept Annotated iMagE Libraries	27
2.45	A Framework for Index Bulk Loading and Dynamization	28
2.46	Duality Between Prefetching and Queued Writing with Parallel Disks	28
2.47	Supporting Incremental Join Queries on Ranked Inputs	29
2.48	Aggregate Predicate support in DBMS	29
2.49	XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation	30
2.50	Implementing I/O-Efficient Data Structures Using TPIE	30
2.51	Efficient Update of Indexes for Dynamically Changing Web Documents	31
2.52	SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads	31
2.53	Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching	31
2.54	High-Order Entropy-Compressed Text Indexes	32
2.55	When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications	32
2.56	Fast Compression with a Static Model in High-Order Entropy	32
2.57	CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation	32
2.58	Compressed Data Structures: Dictionaries and the Data-Aware Measures	32
2.59	An Algorithmic Framework for Compression and Text Indexing	32
2.60	Online Algorithms for Prefetching and Caching in Parallel Disks	33
2.61	Bulk Operations for Space-Partitioning Trees	33
2.62	Mining Deviants in Time Series Data Streams	34
2.63	Rank-aware Query Optimization	34
2.64	Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data	35
2.65	Efficient Join Processing over Uncertain-Valued Attributes	35
2.66	A Cache-Oblivious Index for Approximate String Matching	36
2.67	The SBC-tree: An Index for Run-Length Compressed Sequences	36
2.68	Dynamic Rank/Select Dictionaries with Applications to XML Indexing	36
2.69	A Framework for Dynamizing Succinct Data Structures	37
2.70	Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing	37
2.71	Algorithms and Data Structures for External Memory— <i>main reference!</i>	37
2.72	On Searching Compressed String Collections Cache-Obliviously	38
2.73	On Entropy-Compressed Text Indexing in External Memory	38
3	DATA COMPRESSION	38
3.1	Design and Analysis of Dynamic Huffman Codes	38
3.2	ALGORITHM 673 Dynamic Huffman Coding	39
3.3	Analysis of Arithmetic Coding for Data Compression	39
3.4	New Methods for Lossless Image Compression Using Arithmetic Coding	39
3.5	Practical Implementations of Arithmetic Coding	40
3.6	Error Modeling for Hierarchical Lossless Image Compression	40
3.7	Optimal Prefetching via Data Compression	40
3.8	Optimal Prediction for Prefetching in the Worst Case	40
3.9	Parallel Lossless Image Compression Using Huffman and Arithmetic Coding	40
3.10	Nearly Optimal Vector Quantization via Linear Programming	41

3.11	Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding	41
3.12	Fast and Efficient Lossless Image Compression	41
3.13	Fast Progressive Lossless Image Compression	42
3.14	Arithmetic Coding for Data Compression	42
3.15	Explicit Bit Minimization for Motion-Compensated Video Coding	42
3.16	Dictionary Selection using Partial Matching	43
3.17	Rate-Distortion Optimizations for Motion Estimation in Low-Bitrate Video Coding	43
3.18	Efficient Cost Measures for Motion Compensation at Low Bit Rates	43
3.19	Lexicographic Bit Allocation for MPEG Video	44
3.20	Text Compression via Alphabet Re-representation	44
3.21	Efficient Algorithms for MPEG Video Compression	44
3.22	Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching	45
3.23	High-Order Entropy-Compressed Text Indexes	46
3.24	Indexing Equals Compression: Experiments on Suffix Arrays and Trees	46
3.25	Fast Compression with a Static Model in High-Order Entropy	47
3.26	Compressed Data Structures: Dictionaries and the Data-Aware Measures	47
3.27	Compressed Dictionaries: Space Measures, Data Sets, and Experiments	47
3.28	An Algorithmic Framework for Compression and Text Indexing	48
3.29	SBC-tree: Efficient Indexing for RLE-Compressed Strings	48
3.30	Dynamic Rank/Select Dictionaries with Applications to XML Indexing	48
3.31	A Framework for Dynamizing Succinct Data Structures	48
3.32	Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing	49
3.33	Compressed Index for Dictionary Matching	49
3.34	On Searching Compressed String Collections Cache-Obliviously	50
3.35	On Entropy-Compressed Text Indexing in External Memory	50
3.36	Space-Efficient Framework for Top-k String Retrieval Problems	50
4	LEARNING, PREDICTION, ESTIMATION, CACHING, AND PREFETCHING	51
4.1	Complexity Results on Learning by Neural Nets	51
4.2	Learning in Parallel	51
4.3	Optimal Prefetching via Data Compression	52
4.4	Practical Prefetching via Data Compression	52
4.5	Optimal Prediction for Prefetching in the Worst Case	52
4.6	Using Vapnik-Chervonenkis Dimension to Analyze the Testing Complexity of Program Segments	53
4.7	A Theory for Memory-Based Learning	53
4.8	Coping with Uncertainty in Map Learning	54
4.9	Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments	54
4.10	Application-Controlled Paging for a Shared Cache	55
4.11	Estimating Alphanumeric Selectivity in the Presence of Wildcards	56
4.12	Competitive Analysis of Buffer Management Algorithms for Parallel I/O Systems	56
4.13	Wavelet-Based Histograms for Selectivity Estimation	57
4.14	Scalable Mining for Classification Rules in Relational Databases	57
4.15	Data Cube Approximation and Histograms via Wavelets	58
4.16	Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets	58

4.17	Dynamic Maintenance of Wavelet-Based Histograms	59
4.18	XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation	59
4.19	Lexicographically Optimal Smoothing for Broadband Traffic Multiplexing	59
4.20	SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads	60
4.21	Online Algorithms for Prefetching and Caching in Parallel Disks	60
4.22	CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation	60
5	SAMPLING, HISTOGRAMS, AND RANDOM VARIATE GENERATION	60
5.1	Faster Methods for Random Sampling	60
5.2	Random Sampling with a Reservoir	61
5.3	An Efficient Algorithm for Sequential Random Sampling	61
5.4	Dynamic Generation of Discrete Random Variates	61
5.5	Approximate Data Structures with Applications	62
5.6	Wavelet-Based Histograms for Selectivity Estimation	62
5.7	Data Cube Approximation and Histograms via Wavelets	62
5.8	Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets	62
5.9	Dynamic Maintenance of Wavelet-Based Histograms	62
5.10	XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation	62
5.11	SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads	62
5.12	CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation	62
6	COMPUTATIONAL GEOMETRY	62
6.1	Parallel Transitive Closure and Point Location in Planar Structures	62
6.2	Optimal Cooperative Search in Fractional Cascaded Data Structures	63
6.3	Approximation Algorithms for Geometric Median Problems	63
6.4	A Simplified Technique for Hidden-Line Elimination in Terrains	63
6.5	External-Memory Computational Geometry	64
6.6	Indexing for Data Models with Constraints and Classes	64
6.7	The Object Complexity Model for Hidden-Surface Elimination	64
6.8	Binary Space Partitions for Fat Rectangles	64
6.9	Efficient 3-D Range Searching in External Memory	64
6.10	Optimal External Memory Interval Management	65
6.11	Communication Issues in Large-Scale Geometric Computation	65
6.12	Practical Techniques for Constructing Binary Space Partitions for Orthogonal Rectangles	65
6.13	Cylindrical Static and Kinetic Binary Space Partitions	65
6.14	I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking	66
6.15	Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Searching Problems	66
6.16	Efficient Searching with Linear Constraints	66
6.17	Scalable Sweep-Based Spatial Join	66
6.18	Constructing Binary Space Partitions for Orthogonal Rectangles in Practice	66
6.19	Efficient Bulk Operations on Dynamic R-trees	66

6.20	I/O-Efficient Dynamic Point Location in Monotone Subdivisions	66
6.21	A Parallel Algorithm for Planar Orthogonal Grid Drawings	67
6.22	On Two-Dimensional Indexability and Optimal Range Search Indexing	67
6.23	Online Data Structures in External Memory	67
6.24	A Unified Approach for Indexed and Non-Indexed Spatial Joins	67
6.25	I/O-Efficient Algorithms for Problems on Grid-Based Terrains	67
6.26	CAMEL: Concept Annotated iMagE Libraries	67
6.27	External Memory Algorithms and Data Structures: Dealing with Massive Data . . .	67
6.28	Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data	67
6.29	Efficient Join Processing over Uncertain-Valued Attributes	67
6.30	Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing	68
6.31	Algorithms and Data Structures for External Memory— <i>main reference!</i>	68
7	PARALLEL ALGORITHMS AND SCIENTIFIC COMPUTING	68
7.1	The Input/Output Complexity of Sorting and Related Problems	68
7.2	I/O Overhead and Parallel VLSI Architectures for Lattice Computations	68
7.3	Parallel Transitive Closure and Point Location in Planar Structures	68
7.4	Optimal Algorithms for Parallel Memory I: Two-Level Memories	68
7.5	Optimal Algorithms for Parallel Memory I: Two-Level Memories	68
7.6	Large-Scale Sorting in Uniform Memory Hierarchies	68
7.7	Optimal Deterministic Sorting on Parallel Disks	69
7.8	Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies	69
7.9	Learning in Parallel	69
7.10	Parallel Lossless Image Compression Using Huffman and Arithmetic Coding	69
7.11	Optimal Cooperative Search in Fractional Cascaded Data Structures	69
7.12	A Divide and Conquer Approach to Shortest Paths in Planar Layered Graphs	69
7.13	TPIE: Transparent Parallel I/O Programming Environment	69
7.14	I/O-Efficient Scientific Computation using TPIE	69
7.15	Simple Randomized Mergesort on Parallel Disks	70
7.16	Modeling and optimizing I/O throughput of multiple disks on a bus	70
7.17	Online Data Structures in External Memory	70
7.18	Lower Bounds and Parallel Algorithms for Planar Orthogonal Grid Drawings	70
7.19	External Memory Algorithms and Data Structures: Dealing with Massive Data . . .	70
7.20	Distribution Sort with Randomized Cycling	70
7.21	Duality Between Prefetching and Queued Writing with Parallel Disks	70
8	COMBINATORIAL ALGORITHMS AND COMBINATORIAL OPTIMIZA-	
	TION	70
8.1	Average-Case Analysis of Algorithms and Data Structures	70
8.2	The Input/Output Complexity of Sorting and Related Problems	70
8.3	ϵ -Approximations with Small Packing Constraint Violation	71
8.4	Approximation Algorithms for Geometric Median Problems	71
8.5	Nearly Optimal Vector Quantization via Linear Programming	71
8.6	A Theory for Memory-Based Learning	71
8.7	Optimal Algorithms for Parallel Memory I: Two-Level Memories	71
8.8	Optimal Algorithms for Parallel Memory I: Two-Level Memories	71
8.9	Large-Scale Sorting in Uniform Memory Hierarchies	71
8.10	Optimal Deterministic Sorting on Parallel Disks	71

8.11	Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies	72
8.12	Approximate Data Structures with Applications	72
8.13	Blocking for External Graph Searching	72
8.14	External-Memory Graph Algorithms	72
8.15	External-Memory Algorithms for Processing Line Segments in Geographic Information Systems	72
8.16	Simple Randomized Mergesort on Parallel Disks	72
8.17	On Sorting Strings in External Memory	72
8.18	Simple Randomized Mergesort on Parallel Disks	73
8.19	Online Data Structures in External Memory	73
8.20	External Memory Algorithms with Dynamically Changing Memory Allocations	73
8.21	Efficient Bundle Sorting	73
8.22	External Memory Algorithms and Data Structures: Dealing with Massive Data	73
8.23	Distribution Sort with Randomized Cycling	73
8.24	Efficient Sorting using Registers and Caches	73
8.25	Duality Between Prefetching and Queued Writing with Parallel Disks	73
8.26	Constrained Querying of Multimedia Databases: Issues and Approaches	74
8.27	Supporting Incremental Join Queries on Ranked Inputs	74
8.28	Aggregate Predicate support in DBMS	74
8.29	Online Algorithms for Prefetching and Caching in Parallel Disks	74
8.30	Bulk Operations for Space-Partitioning Trees	74
8.31	Mining Deviants in Time Series Data Streams	74
8.32	Rank-aware Query Optimization	74
8.33	Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data	74
8.34	Efficient Join Processing over Uncertain-Valued Attributes	74
8.35	A Framework for Dynamizing Succinct Data Structures	74
8.36	Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing	74
8.37	SBC-tree: Efficient Indexing for RLE-Compressed Strings	74
8.38	Algorithms and Data Structures for External Memory— <i>main reference!</i>	75
8.39	On Searching Compressed String Collections Cache-Obliviously	75
8.40	On Entropy-Compressed Text Indexing in External Memory	75
9	ONLINE ALGORITHMS AND DYNAMIC DATA STRUCTURES	75
9.1	A Complexity Theoretic Approach to Incremental Computation	75
9.2	Optimal Prefetching via Data Compression	75
9.3	Dynamic Generation of Discrete Random Variates	75
9.4	Approximate Data Structures with Applications	76
9.5	Online Perfect Matching and Mobile Computing	76
9.6	Load Balancing in the L_p Norm	76
9.7	Efficient 3-D Range Searching in External Memory	77
9.8	Optimal External Memory Interval Management	77
9.9	Competitive Parallel Disk Prefetching and Buffer Management	77
9.10	Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments	77
9.11	A Competitive Application-Controlled Paging Algorithm for a Shared Cache	77
9.12	Cylindrical Static and Kinetic Binary Space Partitions	77
9.13	I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking	77
9.14	Efficient Searching with Linear Constraints	77
9.15	Efficient Bulk Operations on Dynamic R-trees	77

9.16	I/O-Efficient Dynamic Point Location in Monotone Subdivisions	77
9.17	On Two-Dimensional Indexability and Optimal Range Search Indexing	77
9.18	Online Data Structures in External Memory	78
9.19	External Memory Algorithms with Dynamically Changing Memory Allocations . . .	78
9.20	Dynamic Maintenance of Wavelet-Based Histograms	78
9.21	External Memory Algorithms and Data Structures: Dealing with Massive Data . . .	78
9.22	Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching	78
9.23	Constrained Querying of Multimedia Databases: Issues and Approaches	78
9.24	CAMEL: Concept Annotated iMagE Libraries	78
9.25	XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation	78
9.26	Efficient Update of Indexes for Dynamically Changing Web Documents	78
9.27	SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads	78
9.28	CXHist: An On-line Classification-based Histogram for XML String Selectivity Es- timation	78
9.29	Online Algorithms for Prefetching and Caching in Parallel Disks	79
9.30	Mining Deviants in Time Series Data Streams	79
9.31	Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data	79
9.32	Efficient Join Processing over Uncertain-Valued Attributes	79
9.33	SBC-tree: Efficient Indexing for RLE-Compressed Strings	79
9.34	Dynamic Rank/Select Dictionaries with Applications to XML Indexing	79
9.35	A Framework for Dynamizing Succinct Data Structures	79
9.36	Algorithms and Data Structures for External Memory— <i>main reference!</i>	79

1 SURVEYS AND MANUSCRIPTS

1.1 Average-Case Analysis of Algorithms and Data Structures

Knuthian analysis techniques. See paper 8.1.

1.2 Arithmetic Coding for Data Compression

A good overview of arithmetic coding and implementation considerations. See paper 3.14.

1.3 Report of the Working Group on Storage I/O for Large-Scale Computing

A position paper of the Working Group on Storage I/O Issues in Large-Scale at the ACM Workshop on Strategic Directions in Computing Research. See paper 2.18.

1.4 Efficient Algorithms for MPEG Video Compression

The interplay between compression and buffer control algorithms in order to maximize network performance and achieve high visual clarity has shown great results, and Efficient Algorithms for MPEG Video Compression is the first book dedicated to the subject. See paper 3.21.

1.5 Miscellaneous Tips for Writing and Formatting

J. S. Vitter. “Miscellaneous Tips for Writing and Formatting,” updated from time to time, December 2005.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Jeff’s LaTeX macros and a LaTeX template file

This report is an ongoing compendium of my writing and formatting rules that I got tired of repeating over and over again (often to the same students!). Now that it’s written in plain English, there’s no longer any excuse for not following these rules to the letter!

1.6 Algorithms and Data Structures for External Memory—*main reference!*

See paper 2.71 for my book, which gives a good general introduction to algorithms and data structures for use in external memory, when I/O can be a bottleneck.

2 EXTERNAL MEMORY ALGORITHMS, I/O EFFICIENCY, AND DATABASES

A good introduction on external memory algorithms and data structures is my book on the subject (see 2.71).

2.1 The Input/Output Complexity of Sorting and Related Problems

A. Aggarwal and J. S. Vitter. “The Input/Output Complexity of Sorting and Related Problems,” *Communications of the ACM*, **31**(9), September 1988, 1116–1127. A shortened version appears in “The I/O Complexity of Sorting and Related Problems,” *Proceedings of the 14th Annual International Colloquium on Automata, Languages, and Programming (ICALP ’87)*, Karlsruhe, West Germany, July 1987, published in *Lecture Notes in Computer Science*, **267**, Springer-Verlag, Berlin.

Full text (Adobe pdf format)

We provide tight upper and lower bounds, up to a constant factor, for the number of inputs and outputs (I/Os) between internal memory and secondary storage required for five sorting-related problems: sorting, the fast Fourier transform (FFT), permutation networks, permuting, and matrix transposition. The bounds hold both in the worst case and in the average case, and in several situations the constant factors match.

Secondary storage is modeled as a magnetic disk capable of transferring P blocks each containing B records in a single time unit; the records in each block must be input from or output to B contiguous locations on the disk. We give two optimal algorithms for the problems, which are variants of merge sorting and distribution sorting. In particular we show for $P = 1$ that the standard merge sorting algorithm is an optimal external sorting method, up to a constant factor in the number of I/Os. Our sorting algorithms use the same number of I/Os as does the permutation phase of key sorting, except when the internal memory size is extremely small, thus affirming the popular adage that key sorting is not faster. We also give a simpler and more direct derivation of Hong and Kung's lower bound for the FFT for the special case $B = P = O(1)$.

2.2 I/O Overhead and Parallel VLSI Architectures for Lattice Computations

M. H. Nodine, D. P. Lopresti and J. S. Vitter. "I/O Overhead and Parallel VLSI Architectures for Lattice Computations," *IEEE Transactions on Computers*, **40** (7), July 1991, 843–852.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we introduce input/output (I/O) overhead ψ as a complexity measure for VLSI implementations of two-dimensional lattice computations of the type arising in the simulation of physical systems. We show by pebbling arguments that $\psi = \Omega(n^{-1})$ when there are n^2 processing elements available. If the results are required to be observed at every generation, and no on-chip storage is allowed, we show the lower bound is the constant 2. We then examine four VLSI architectures and show that one of them, the multi-generation sweep architecture, also has I/O overhead proportional to n^{-1} . We compare the constants of proportionality between the lower bound and the architecture. Finally, we prove a closed-form for the discrete minimization equation giving the optimal number of generations to compute for the multi-generation sweep architecture.

2.3 Algorithms for Parallel Memory I: Two-Level Memories

J. S. Vitter and E. A. M. Shriver. "Algorithms for Parallel Memory I: Two-Level Memories," double special issue on Large-Scale Memories in *Algorithmica*, **12**(2–3), 1994, 110–147. A shortened version appears in "Optimal Disk I/O with Parallel Block Transfer," *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, Baltimore, MD, May 1990, 159–169.

Full text with FFT figure missing (gzip-compressed postscript)

Full text with FFT figure missing (Adobe pdf format)

We provide the first optimal algorithms in terms of the number of input/outputs (I/Os) required between internal memory and multiple secondary storage devices for the problems of sorting, FFT, matrix transposition, standard matrix multiplication, and related problems. Our two-level memory model is new and gives a realistic treatment of *parallel block transfer*, in which during a single I/O each of the P secondary storage devices can simultaneously transfer a contiguous block of B records. The model pertains to a large-scale uniprocessor system or parallel multiprocessor system with P disks. In addition, the sorting, FFT, permutation network, and standard matrix multiplication algorithms are typically optimal in terms of the amount of internal processing time. The difficulty

in developing optimal algorithms is to cope with the partitioning of memory into P separate physical devices. Our algorithms' performance can be significantly better than those obtained by the well-known but nonoptimal technique of disk striping. Our optimal sorting algorithm is randomized, but practical; the probability of using more than ℓ times the optimal number of I/Os is exponentially small in $\ell(\log \ell) \log(M/B)$, where M is the internal memory size.

2.4 Algorithms for Parallel Memory II: Hierarchical Multilevel Memories

J. S. Vitter and E. A. M. Shriver. "Algorithms for Parallel Memory II: Hierarchical Multilevel Memories," double special issue on Large-Scale Memories in *Algorithmica*, **12**(2-3), 1994, 148-169. A shortened version appears in "Optimal Disk I/O with Parallel Block Transfer," *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90)*, Baltimore, MD, May 1990, 159-169.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we introduce parallel versions of two hierarchical memory models and give optimal algorithms in these models for sorting, FFT, and matrix multiplication. In our parallel models, there are P memory hierarchies operating simultaneously; communication among the hierarchies takes place at a base memory level. Our optimal sorting algorithm is randomized and is based upon the probabilistic partitioning technique developed in the companion paper for optimal disk sorting in a two-level memory with parallel block transfer. The probability of using ℓ times the optimal running time is exponentially small in $\ell(\log \ell) \log P$.

2.5 Large-Scale Sorting in Uniform Memory Hierarchies

J. S. Vitter and M. H. Nodine. "Large-Scale Sorting in Uniform Memory Hierarchies," special issue on parallel I/O systems in *Journal of Parallel and Distributed Computing*, **17**, January 1993, 107-114. A shortened version appears in "Large-Scale Sorting in Parallel Memories," *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '91)*, Hilton Head, SC, July 1991, 29-39.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

2.6 Optimal Deterministic Sorting on Parallel Disks

M. H. Nodine and J. S. Vitter. "Optimal Deterministic Sorting on Parallel Disks." An earlier and shortened version appears in *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, Velen, Germany, June-July 1993, 120-129.

Full text (Adobe pdf format)

We present a load balancing technique that leads to an optimal deterministic algorithm called Balance Sort for external sorting on multiple disks. Our measure of performance is the number of input/output (I/O) operations. In each I/O, each of the D disks can simultaneously transfer a block of data. Our algorithm improves upon the randomized optimal algorithm of Vitter and Shriver as well as the (non-optimal) commonly-used technique of disk striping. It also improves upon our earlier merge-based sorting algorithm in that it has smaller constants hidden in the big-oh notation, and it is possible to implement using only striped writes (but independent reads). In a companion paper, we show how to modify the algorithm to achieve optimal CPU time, even on parallel processors and parallel memory hierarchies.

2.7 Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies

M. H. Nodine and J. S. Vitter. “Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies,” submitted for publication. An earlier and shortened version appears in *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '93)*, Velen, Germany, June–July 1993, 120–129.

Full text (Adobe pdf format)

We present a practical deterministic load balancing strategy for distribution sort that is applicable to parallel disks and parallel memory hierarchies with both single and parallel processors. The simplest application of the strategy is an optimal deterministic algorithm called Balance Sort for external sorting on multiple disks with a single CPU, as described in the companion paper. However, the internal processing of Balance Sort does not seem parallelizable. In this paper, we develop an elegant variation that achieves full parallel speedup. The algorithms so derived are optimal for all parallel memory hierarchies with any type of a PRAM base-level interconnection and are either optimal or best-known for a hypercube interconnection. We show how to achieve optimal internal processing time as well as optimal number of I/Os in parallel two-level memories.

2.8 Blocking for External Graph Searching

M. H. Nodine, M. T. Goodrich, and J. S. Vitter. “Blocking for External Graph Searching,” *Algorithmica*, **16**(2), August 1996, 181–214. A shortened version appears in *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems (PODS '93)*, Washington, D. C., May 1993.

Full text (Adobe pdf format)

In this paper, we consider the problem of using disk blocks efficiently in searching graphs that are too large to fit in internal memory. Our model allows a vertex to be represented any number of times on the disk in order to take advantage of redundancy. We give matching upper and lower bounds for complete d -ary trees and d -dimensional grid graphs, as well as for classes of general graphs that intuitively speaking have a close to uniform number of neighbors around each vertex.

2.9 Indexing for Data Models with Constraints and Classes

P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter. “Indexing for Data Models with Constraints and Classes,” *Journal of Computer and System Sciences*, **52**(3), 1996, 589–612. An earlier version appeared in *Proceedings of the 12th Annual ACM Symposium on Principles of Database Systems (PODS '93)*, Washington, D. C., May 1993.

Full text (Adobe pdf format)

We examine I/O-efficient data structures that provide indexing support for new data models. The database languages of these models include concepts from constraint programming (e.g., relational tuples are generalized to conjunctions of constraints) and from object-oriented programming (e.g., objects are organized in class hierarchies). Let n be the size of the database, c the number of classes, B the page size on secondary storage, and t the size of the output of a query. (1) Indexing by one attribute in many constraint data models is equivalent to external dynamic interval management, which is a special case of external dynamic 2-dimensional range searching. We present a semi-dynamic data structure for this problem that has worst-case space $O(n/B)$ pages, query I/O time $O(\log_B n + t/B)$ and $O(\log_B n + (\log_B n)^2/B)$ amortized insert I/O time. Note that, for the static version of this problem, this is the first worst-case optimal solution. (2) Indexing by one attribute and by class name in an object-oriented model, where objects are organized as a forest

hierarchy of classes, is also a special case of external dynamic 2-dimensional range searching. Based on this observation, we first identify a simple algorithm with good worst-case performance, query I/O time $O(\log_2 c \log_B n + t/B)$, update I/O time $O(\log_2 c \log_B n)$ and space $O((n/B) \log_2 c)$ pages for the class indexing problem. Using the forest structure of the class hierarchy and techniques from the constraint indexing problem, we improve its query I/O time to $O(\log_B n + t/B + \log_2 B)$.

2.10 External-Memory Computational Geometry

M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. “External-Memory Computational Geometry,” *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science (FOCS '93)*, Palo Alto, CA, November 1993.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper, we give new techniques for designing efficient algorithms for computational geometry problems that are too large to be solved in internal memory, and we use these techniques to develop optimal and practical algorithms for a number of important large-scale problems in computational geometry. Our algorithms are optimal for a wide range of two-level and hierarchical multilevel memory models, including parallel models. The algorithms are optimal in terms of both I/O cost and internal computation.

Our results are built on four fundamental techniques: *distribution sweeping*, a generic method for externalizing plane-sweep algorithms; *persistent B-trees*, for which we have both on-line and off-line methods; *batch filtering*, a general method for performing K simultaneous external-memory searches in any data structure that can be modeled as a planar layered dag; and *external marriage-before-conquest*, an external-memory analog of the well-known technique of Kirkpatrick and Seidel. Using these techniques we are able to solve a very large number of problems in computational geometry, including batched range queries, 2-d and 3-d convex hull construction, planar point location, range queries, finding all nearest neighbors for a set of planar points, rectangle intersection/union reporting, computing the visibility of segments from a point, performing ray-shooting queries in constructive solid geometry (CSG) models, as well as several geometric dominance problems.

These results are significant because large-scale problems involving geometric data are ubiquitous in spatial databases, geographic information systems (GIS), constraint logic programming, object oriented databases, statistics, virtual reality systems, and graphics. This work makes a big step, both theoretically and in practice, towards the effective management and manipulation of geometric data in external memory, which is an essential component of these applications.

2.11 External-Memory Graph Algorithms

Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. “External-Memory Graph Algorithms,” *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '95)*, San Francisco, CA, January 1995.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a collection of new techniques for designing and analyzing efficient external-memory algorithms for graph problems and illustrate how these techniques can be applied to a wide variety of specific problems. Our results include:

- *Proximate-neighboring*. We present a simple method for deriving external-memory lower bounds via reductions from a problem we call the “proximate neighbors” problem. We use

this technique to derive non-trivial lower bounds for such problems as list ranking, expression tree evaluation, and connected components.

- *PRAM simulation.* We give methods for efficiently simulating PRAM computations in external memory, even for some cases in which the PRAM algorithm is not work-optimal. We apply this to derive a number of optimal (and simple) external-memory graph algorithms.
- *Time-forward processing.* We present a general technique for evaluating circuits (or “circuit-like” computations) in external memory. We also use this in a deterministic list ranking algorithm.
- *Deterministic 3-coloring of a cycle.* We give several optimal methods for 3-coloring a cycle, which can be used as a subroutine for finding large independent sets for list ranking. Our ideas go beyond a straightforward PRAM simulation, and may be of independent interest.
- *External depth-first search.* We discuss a method for performing depth first search and solving related problems efficiently in external memory. Our technique can be used in conjunction with ideas due to Ullman and Yannakakis in order to solve graph problems involving closed semi-ring computations even when their assumption that vertices fit in main memory does not hold.

Our techniques apply to a number of problems, including list ranking, which we discuss in detail, finding Euler tours, expression-tree evaluation, centroid decomposition of a tree, least-common ancestors, minimum spanning tree verification, connected and biconnected components, minimum spanning forest, ear decomposition, topological sorting, reachability, graph drawing, and visibility representation.

2.12 External-Memory Algorithms for Processing Line Segments in Geographic Information Systems

L. Arge, D. E. Vengroff, and J. S. Vitter. “External-Memory Algorithms for Processing Line Segments in Geographic Information Systems,” *Algorithmica*, **47**(1), 2007, 1–25. A shortened version appears in *Proceedings of the 3rd Annual European Symposium on Algorithms (ESA '95)*, September 1995, published in *Lecture Notes in Computer Science*, **979**, Springer-Verlag, Berlin, 295–310.

Full text (Adobe pdf format)

In the design of algorithms for large-scale applications it is essential to consider the problem of minimizing I/O communication. Geographical information systems (GIS) are good examples of such large-scale applications as they frequently handle huge amounts of spatial data. In this paper we develop efficient new external-memory algorithms for a number of important problems involving line segments in the plane, including trapezoid decomposition, batched planar point location, triangulation, red-blue line segment intersection reporting, and general line segment intersection reporting. In GIS systems, the first three problems are useful for rendering and modeling, and the latter two are frequently used for overlaying maps and extracting information from them.

To solve these problems, we combine and modify in novel ways several of the previously known techniques for designing efficient algorithms for external memory. We also develop a powerful new technique that can be regarded as a practical external memory version of fractional cascading. Except for the batched planar point location problem, no algorithms specifically designed for external memory were previously known for these problems. Our algorithms for triangulation and line segment intersection partially answer previously posed open problems, while the batched planar

point location algorithm improves on the previously known solution, which applied only to monotone decompositions. Our algorithm for the red-blue line segment intersection problem is provably optimal.

2.13 TPIE: Transparent Parallel I/O Programming Environment

The TPIE software project, initially begun by Darren Vengroff as part of his PhD dissertation work, is being carried on at Duke University. TPIE provides a high-level implementation platform for applications that require efficient external memory access. The manual and distribution information can be found on the TPIE group home page. A new release is coming out Fall 1998.

2.14 I/O-Efficient Scientific Computation using TPIE

D. E. Vengroff and J. S. Vitter. "I/O-Efficient Scientific Computation using TPIE," *Proceedings of the Goddard Conference on Mass Storage Systems and Technologies*, College Park, MD, September 1996, published in NASA Conference Publication 3340, Volume II, 553-570.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In recent years, I/O-efficient algorithms for a wide variety of problems have appeared in the literature. Thus far, however, systems specifically designed to assist programmers in implementing such algorithms have remained scarce. TPIE is a system designed to fill this void. It supports I/O-efficient paradigms for problems from a variety of domains, including computational geometry, graph algorithms, and scientific computation. The TPIE interface frees programmers from having to deal not only of explicit read and write calls, but also the complex memory management that must be performed for I/O-efficient computation.

In this paper, we discuss applications of TPIE to problems in scientific computation. We discuss algorithmic issues underlying the design and implementation of the relevant components of TPIE and present performance results of programs written to solve a series of benchmark problems using our current TPIE prototype. Some of the benchmarks we present are based on the NAS parallel benchmarks, while others are of our own creation.

We demonstrate that the CPU overhead required to manage I/O is small and that even with just a single disk the I/O overhead of I/O-efficient computation ranges from negligible to the same order of magnitude as CPU time. We conjecture that if we use a number of disks in parallel this overhead can be all but eliminated.

2.15 Efficient 3-D Range Searching in External Memory

D. E. Vengroff, and J. S. Vitter. "Efficient 3-D Range Searching in External Memory," *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC '96)*, Philadelphia, PA, May 1996.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a new approach to designing data structures for the important problem of external-memory range searching in two and three dimensions. We construct data structures for answering range queries in $O((\log \log \log_B N) \log_B N + K/B)$ I/O operations, where N is the number of points in the data structure, B is the I/O block size, and K is the number of points in the answer to the query. We base our data structures on the novel concept of B -approximate boundaries, which are manifolds that partition space into regions based on the output size of queries at points within the space.

Our data structures answer a longstanding open problem by providing three dimensional results comparable to those provided by Sairam and Ramaswamy for the two dimensional case, though completely new techniques are used. Ours is the first 3-D range search data structure that simultaneously achieves both a base- B logarithmic search overhead (namely, $(\log \log \log_B N) \log_B N$) and a fully blocked output component (namely, K/B). This gives us an overall I/O complexity extremely close to the well-known lower bound of $\Omega(\log_B N + K/B)$. The space usage is more than linear by a logarithmic or polylogarithmic factor, depending on type of range search.

2.16 Optimal External Memory Interval Management

L. Arge and J. S. Vitter. “Optimal External Memory Interval Management,” *SIAM Journal on Computing*, **32**(6), 2003, 1488–1508. An extended abstract appears in “Optimal Dynamic Interval Management in External Memory,” *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS '96)*, Burlington, VT, October 1996, 560–569. Also appeared in Abstracts of the 1st CGC Workshop on Computational Geometry, Center for Geometric Computing, Johns Hopkins University, Baltimore, MD, October 1996.

Full text (Adobe pdf format)

We present a space- and I/O-optimal external-memory data structure for answering stabbing queries on a set of dynamically maintained intervals. Our data structure settles an open problem in databases and I/O algorithms by providing the first optimal external-memory solution to the dynamic interval management problem, which is a special case of 2-dimensional range searching and a central problem for object-oriented and temporal databases and for constraint logic programming. Our data structure simultaneously uses optimal linear space (that is, $O(N/B)$ blocks of disk space) and achieves the optimal $O(\log_B N + T/B)$ I/O query bound and $O(\log_B N)$ I/O update bound, where B is the I/O block size and T the number of elements in the answer to a query. Our structure is also the first optimal external data structure for a 2-dimensional range searching problem that has worst-case as opposed to amortized update bounds. Part of the data structure uses a novel balancing technique for efficient worst-case manipulation of balanced trees, which is of independent interest.

2.17 Simple Randomized Mergesort on Parallel Disks

R. D. Barve, E. F. Grove and J. S. Vitter. “Simple Randomized Mergesort on Parallel Disks,” special issue on parallel I/O in *Parallel Computing*, **23**(4), 1997. An earlier version appears in *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '96)*, Padua, Italy, June 1996, 109–118.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We consider the problem of sorting a file of N records on the D -disk model of parallel I/O in which there are two sources of parallelism. Records are transferred to and from disk concurrently in blocks of B contiguous records. In each I/O operation, up to one block can be transferred to or from each of the D disks in parallel. We propose a simple, efficient, randomized mergesort algorithm called SRM that uses a forecast-and-flush approach to overcome the inherent difficulties of simple merging on parallel disks. SRM exhibits a limited use of randomization and also has a useful deterministic version. Generalizing the technique of forecasting, our algorithm is able to read in, at any time, the “right” block from any disk, and using the technique of flushing, our algorithm evicts, without any I/O overhead, just the “right” blocks from memory to make space for new ones to be read in. The disk layout of SRM is such that it enjoys perfect write parallelism, avoiding

fundamental inefficiencies of previous mergesort algorithms. By analysis of generalized maximum occupancy problems we are able to derive an analytical upper bound on SRM's expected overhead valid for arbitrary inputs.

The upper bound derived on expected I/O performance of SRM indicates that SRM is provably better than disk-striped mergesort (DSM) for realistic parameter values D , M , and B . Average-case simulations show further improvement on the analytical upper bound. Unlike previously proposed optimal sorting algorithms, SRM outperforms DSM even when the number D of parallel disks is small.

2.18 Report of the Working Group on Storage I/O for Large-Scale Computing

G. Gibson, J. S. Vitter, and J. Wilkes, editors. "Report of the Working Group on Storage I/O for Large-Scale Computing," Strategic Directions in Computing Research, *ACM Computing Surveys*, **28**(4), December 1996.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We discuss the strategic directions and challenges in the management and use of *storage systems*—those components of computer systems responsible for the storage and retrieval of data. The performance gap between main and secondary memories shows no imminent sign of vanishing, and thus continuing research into storage I/O will be essential to reap the full benefit from the advances occurring in many other areas of computer science. In this report we identify a few strategic research goals and possible thrusts to meet those goals.

2.19 I/O-Efficient Algorithms and Environments

D. E. Vengroff and J. S. Vitter. "I/O-Efficient Algorithms and Environments," *ACM Computing Surveys*, **28**(4es), December 1996.

Text (html)

There has recently been much productive work in the algorithms community on techniques for efficient use of external memory in large-scale applications. In order to implement I/O-optimal algorithms efficiently, the machines they run on must support fundamental external-memory operations. Unfortunately, existing file systems generally do not support the necessary semantics or provide useful tools. There are three basic approaches to supporting development of I/O-efficient code: array-oriented systems (such as PASSION and ViC*), access-oriented systems (such as the UNIX file system and Panda), and framework-oriented systems (such as TPIE, a Transparent Parallel I/O Programming Environment). In this position statement, we discuss the advantages and potential of the TPIE approach in enabling I/O-efficient computation.

Categories and Subject Descriptors: D.4.2 [Operating Systems]: Storage Management - Secondary Storage; D.4.4 [Operating Systems]: Communications Management - Input/Output; E.2 [Data Storage Representations]: Contiguous representations; E.5 [Files]: Sorting/searching; F.2.1 [Analysis of Algorithms and Problem Complexity: Numerical Algorithms and Problems - Computations on matrices; F.2.2 [Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems - Computations on discrete structures, geometrical problems and computations, sorting and searching; B.4.4 [Input/Output and Data Communications: Performance Analysis and Design Aids] - Formal models, Worst-case analysis;

General Terms: Algorithms, Design, Languages, Performance, Theory. Additional Key Words and Phrases: I/O, external memory, secondary memory, communication, disk drive, parallel disks, sorting.

2.20 On Sorting Strings in External Memory

L. Arge, P. Ferragina, R. Grossi, and J. S. Vitter. “On Sorting Strings in External Memory,” *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC '97)*, El Paso, TX, May 1997, 540–548, and in “Sequence Sorting in Secondary Storage,” *Proceedings of the 1997 International Conference on Compression and Complexity of Sequences (SEQUENCES '97)*, Positano, Italy, June 1997.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we address for the first time the I/O complexity of the problem of sorting strings in external memory, which is a fundamental component of many large-scale text applications. In the standard unit-cost RAM comparison model, the complexity of sorting K strings of total length N is $\Theta(K \log_2 K + N)$. By analogy, in the external memory (or I/O) model, where the internal memory has size M and the block transfer size is B , it would be natural to guess that the I/O complexity of sorting strings is $\Theta(\frac{K}{B} \log_{M/B} \frac{K}{B} + \frac{N}{B})$, but the known algorithms do not come even close to achieving this bound. Our results show, somewhat counterintuitively, that the I/O complexity of string sorting depends upon the length of the strings relative to the block size. We first consider a simple comparison I/O model, where one is not allowed to break the strings into their characters, and we show that the I/O complexity of string sorting in this model is $\Theta(\frac{N_1}{B} \log_{M/B} \frac{N_1}{B} + K_2 \log_{M/B} K_2 + \frac{N}{B})$, where N_1 is the total length of all strings shorter than B and K_2 is the number of strings longer than B . We then consider two more general I/O comparison models in which string breaking is allowed. We obtain improved algorithms and in several cases lower bounds that match their I/O bounds. Finally, we develop more practical algorithms without assuming the comparison model.

2.21 I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking

P. K. Agarwal, L. Arge, T. M. Murali, K. R. Varadarajan, and J. S. Vitter. “I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking,” *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, San Francisco, CA, January 1998.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

For a polyhedral terrain, the contour at z -coordinate h is defined to be the intersection of the plane $z = h$ with the terrain. In this paper, we study the contour-line extraction problem, where we want to preprocess the terrain into a data structure so that given a query z -coordinate h , we can report the h -contour quickly. This problem is central to geographic information systems (GIS), where terrains are often stored as Triangular Irregular Networks (TINs). We present an I/O-optimal algorithm for this problem which stores a terrain with N vertices using $O(N/B)$ blocks, where B is the size of a disk block, so that for any query h , the h -contour can be computed using $O(\log_B N + |C|/B)$ I/O operations, where $|C|$ denotes the size of the h -contour.

We also present an improved algorithm for a more general problem of blocking bounded-degree planar graphs such as TINs (i.e., storing them on disk so that any graph traversal algorithm can traverse the graph in an I/O-efficient manner). We apply it to two problems that arise in GIS.

2.22 Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Searching Problems

L. A. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. “Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Searching Problems,” *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98)*, San Francisco, CA, January 1998.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We describe a powerful framework for designing efficient batch algorithms for certain large-scale dynamic problems that must be solved using external memory. The class of problems we consider, which we call *colorable external-decomposable problems*, include rectangle intersection, orthogonal line segment intersection, range searching, and point location. We are particularly interested in these problems in two and higher dimensions. They have numerous applications in geographic information systems (GIS), spatial databases, and VLSI and CAD design. We present simplified algorithms for problems previously solved by more complicated approaches (such as rectangle intersection), and we present efficient algorithms for problems not previously solved in an efficient way (such as point location and higher-dimensional versions of range searching and rectangle intersection).

We give experimental results concerning the running time for our approach applied to the red-blue rectangle intersection problem, which is a key component of the extremely important database operation spatial join. Our algorithm scales well with the problem size, and for large problems sizes it greatly outperforms the well-known sweepline approach.

2.23 Scalable Sweep-Based Spatial Join

L. A. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. S. Vitter. “Scalable Sweep-Based Spatial Join,” *Proceedings of the 1998 International Conference on Very Large Databases (VLDB '98)*, New York, August 1998, 570–581. This work forms the basis of two patent applications currently filed and pending with the patent office.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper, we examine the spatial join problem. In particular, we focus on the case when neither of the inputs is indexed. We present a new algorithm, Scalable Sweep-based Spatial Join (SSSJ), that is based on the distribution-sweeping technique recently proposed in computational geometry, and that is the first to achieve theoretically optimal bounds on internal computation time as well as I/O transfers. We present experimental results based on an efficient implementation of the SSSJ algorithm, and compare it to the state-of-the-art Partition-Based Spatial-Merge (PBSM) algorithm of Patel and DeWitt.

Our SSSJ algorithm performs an initial sorting step along the vertical axis, after which we use the distribution-sweeping technique to partition the input into a number of vertical strips, such that the data in each strip can be efficiently processed by an internal-memory sweepline algorithm. A key observation that allowed us to greatly improve the practical performance of our algorithm is that in most sweepline algorithms not all input data is needed in main memory at the same time. In our initial experiments, we observed that on real-life two-dimensional spatial data sets of size N , the internal-memory sweepline algorithm requires only $O(\sqrt{N})$ memory space. This behavior (also known as the square-root rule in the VLSI literature) implies that for real-life two-dimensional data sets, we can bypass the vertical partitioning step and directly perform the sweepline algorithm after

the initial external sorting step. We implemented SSSJ such that partitioning is only done when it is detected that the sweepline algorithm exhausts the internal memory. This results in an algorithm that not only is extremely efficient for real-life data but also offers guaranteed worst-case bounds and predictable behavior on skewed and/or bad input data: Our experiments show that SSSJ performs at least 25% better than PBSM on real-life data sets, and that it robustly handles skewed data on which PBSM suffers a serious performance degeneration.

As part of our experimental work we experimented with a number of different techniques for performing the internal sweepline. By using an efficient partitioning heuristic, we were able to speed up the internal sweeping used by PBSM by a factor of over 4 on the average for real-life data sets. The resulting improved PBSM then performs approximately 10% better than SSSJ on the real-life data we used, and it is thus a good choice of algorithm when the data is known not to be too skewed.

2.24 Competitive Analysis of Buffer Management Algorithms for Parallel I/O Systems

See paper 4.12.

2.25 Modeling and Optimizing I/O Throughput of Multiple Disks on a Bus

R. D. Barve, E. A. M. Shriver, P. Gibbons, B. Hillyer, Y. Matias, and J. S. Vitter. “Modeling and optimizing I/O throughput of multiple disks on a bus,” Appeared as a short paper in *Joint International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '98/Performance '98)*, 264–265. A longer version is currently under submission. Our modeling and optimization work forms the basis of two patent applications currently filed and pending with the patent office.

Full text (gzip-compressed postscript)

Slides for talk (gzip-compressed postscript)

For a wide variety of computational tasks, disk I/O continues to be a serious obstacle to high performance. To meet demanding I/O requirements, systems are designed to use multiple disk drives that share one or more I/O ports to form a disk farm or RAID array. The focus of the present paper is on systems that use multiple disks per SCSI bus. We measured the performance of concurrent random I/Os for three types of SCSI disk drives and three types of computers. The measurements enable us to study bus-related phenomena that impair performance. We describe these phenomena, and present a new I/O performance model that incorporates bus effects to predict the average throughput achieved by concurrent random I/Os that share a SCSI bus. This model, although relatively simple, predicts performance on these platforms to within 11% for fixed I/O sizes in the range 16–128 KB/s. We then describe a technique to improve the I/O throughput. This technique increases the percentage of disk head positioning time that is overlapped with data transfers, and increases the percentage of transfers that occur at bus bandwidth, rather than at disk-head bandwidth. Our technique is most effective for large I/Os and high concurrency—an important performance region for large-scale computing—our improvements are 10–20% better than the naive method for random workloads.

2.26 Scalable Mining for Classification Rules in Relational Databases

See paper 4.14.

2.27 Data Cube Approximation and Histograms via Wavelets

See paper 4.15.

2.28 Efficient Searching with Linear Constraints

P. K. Agarwal, L. A. Arge, J. Erickson, P. G. Franciosa, and J. S. Vitter. “Efficient Searching with Linear Constraints,” *Journal of Computer and System Sciences*, **61**(2), October 2000, 194–216. An extended abstract appears in *Proceedings of the 17th Annual ACM Symposium on Principles of Database Systems (PODS '98)*, Seattle, WA, June 1998, 169–178.

Full text (Adobe pdf format)

We show how to preprocess a set S of points in d -dimensional Euclidean space to get an external memory data structure that efficiently supports linear-constraint queries. Each query is in the form of a linear constraint $\mathbf{a} \cdot \mathbf{x} \leq \mathbf{b}$; the data structure must report all the points of S that satisfy the query. (This problem is called halfspace range searching in the computational geometry literature.) Our goal is to minimize the number of disk blocks required to store the data structure and the number of disk accesses (I/Os) required to answer a query. For $d = 2$, we present the first near-linear size data structures that can answer linear-constraint queries using an optimal number of I/Os. We also present a linear-size data structure that can answer queries efficiently in the worst case. We combine these two approaches to obtain tradeoffs between space and query time. Finally, we show that some of our techniques extend to higher dimensions.

2.29 Wavelet-Based Histograms for Selectivity Estimation

See paper 4.13.

2.30 External Memory Algorithms and Data Structures: Dealing with Massive Data

J. S. Vitter. “External Memory Algorithms and Data Structures: Dealing with Massive Data,” *ACM Computing Surveys*, **33**(2), June 2001, 209–271.

This survey article is superseded by a more comprehensive book 2.71. The book is available online and is recommended as the preferable reference.

ACM Computing Surveys link to original 2001 survey article (Adobe pdf format)

Slides for a talk (Adobe pdf format)

2.31 Efficient Bulk Operations on Dynamic R-trees

L. Arge, K. H. Hinrichs, J. Vahrenhold, and J. S. Vitter. “Efficient Bulk Operations on Dynamic R-trees,” special issue on experimental algorithmics in *Algorithmica*, **33**(1), May 2002, 104–128. A shorter and earlier version appears in *Proceedings of the 1st Workshop on Algorithm Engineering and Experimentation (ALENEX '99)*, Baltimore, MD, January 1999.

Full text (Adobe pdf format)

In recent years there has been an upsurge of interest in spatial databases. A major issue is how to efficiently manipulate massive amounts of spatial data stored on disk in multidimensional *spatial indexes* (data structures). Construction of spatial indexes (*bulk loading*) has been researched intensively in the database community. The continuous arrival of massive amounts of new data make it important to efficiently update existing indexes (*bulk updating*).

In this article we present a simple technique for performing bulk update and query operations on multidimensional indexes. We present our technique in terms of the so-called R-tree and its variants, as they have emerged as practically efficient indexing methods for spatial data. Our method uses ideas from the *buffer tree* lazy buffering technique and fully utilizes the available internal memory and the page size of the operating system. We give a theoretical analysis of our technique, showing that it is efficient both in terms of I/O communication, disk storage, and internal computation time. We also present the results of an extensive set of experiments showing that in practice our approach performs better than the previously best known bulk update methods with respect to update time, and that it produces a better quality index in terms of query performance. One important novel feature of our technique is that in most cases it allows us to perform a batch of updates and queries simultaneously. To be able to do so is essential in environments where queries have to be answered even while the index is being updated and reorganized.

2.32 I/O-Efficient Dynamic Point Location in Monotone Subdivisions

P. K. Agarwal, L. Arge, G. Brodal, and J. S. Vitter. “I/O-Efficient Dynamic Point Location in Monotone Subdivisions,” *Proceedings of the 10th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '99)*, Baltimore, MD, January 1999.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present an efficient external-memory dynamic data structure for point location in monotone planar subdivisions. Our data structure uses $O(N/B)$ disk blocks to store a monotone subdivision of size N , where B is the size of a disk block. It supports queries in $O(\log_B^2 N)$ I/Os (worst-case) and updates in $O(\log_B^2 N)$ I/Os (amortized).

We also propose a new variant of B -trees, called *level-balanced B -trees*, which allow insert, delete, merge, and split operations in $O((1 + \frac{b}{B} \log_{M/B} \frac{N}{B}) \log_b N)$ I/Os (amortized), $2 \leq b \leq B/2$, even if each node stores a pointer to its parent. Here M is the size of main memory. Besides being essential to our point-location data structure, we believe that *level-balanced B -trees* are of significant independent interest. They can, for example, be used to dynamically maintain a planar st-graph using $O((1 + \frac{b}{B} \log_{M/B} \frac{N}{B}) \log_b N) = O(\log_B^2 N)$ I/Os (amortized) per update, so that reachability queries can be answered in $O(\log_B N)$ I/Os (worst case).

2.33 Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets

See paper 4.16.

2.34 On Two-Dimensional Indexability and Optimal Range Search Indexing

L. Arge, V. Samoladas, and J. S. Vitter. “Two-Dimensional Indexability and Optimal Range Search Indexing,” *Proceedings of the 18th Annual ACM Symposium on Principles of Database Systems (PODS '99)*, Philadelphia, PA, May–June 1999.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we settle several longstanding open problems in theory of indexability and external orthogonal range searching. In the first part of the paper, we apply the theory of indexability to the problem of two-dimensional range searching. We show that the special case of 3-sided querying can be solved with constant redundancy and access overhead. From this, we derive indexing schemes

for general 4-sided range queries that exhibit an optimal tradeoff between redundancy and access overhead.

In the second part of the paper, we develop dynamic external memory data structures for the two query types. Our structure for 3-sided queries occupies $O(N/B)$ disk blocks, and it supports insertions and deletions in $O(\log_B N)$ I/Os and queries in $O(\log_B N + T/B)$ I/Os, where B is the disk block size, N is the number of points, and T is the query output size. These bounds are optimal. Our structure for general (4-sided) range searching occupies $O((N/B)(\log(N/B))/\log \log_B N)$ disk blocks and answers queries in $O(\log_B N + T/B)$ I/Os, which are optimal. It also supports updates in $O((\log_B N)(\log(N/B))/\log \log_B N)$ I/Os.

2.35 A Simple and Efficient Parallel Disk Mergesort

R. D. Barve and J. S. Vitter. “A Simple and Efficient Parallel Disk Mergesort,” invited submission to a special issue of *Theory of Computing Systems*, **35**(2), March/April 2002, 189–215. A shortened version appears in *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, St. Malo, France, June 1999.

Full text (Adobe pdf format)

Slides for talk plus extra foils on dynamic memory allocation (gzip-compressed postscript)

External sorting is a fundamental operation in many large scale data processing systems not only for producing sorted output but also as a core subroutine in many operations. Technology trends indicate that developing techniques that effectively use multiple disks in parallel in order to speed up the performance of external sorting is of prime importance. The *simple randomized merging (SRM)* mergesort algorithm proposed in our earlier work is the first parallel disk sorting algorithm that requires a provably optimal number of passes and that is fast in practice. Knuth (in the new edition of *The Art of Computer Programming*, Vol. 3: *Sorting and Searching*) recently identified SRM (which he calls “randomized striping”) as the method of choice for sorting with parallel disks.

In this paper, we present an efficient implementation of SRM, based upon novel data structures. We give a new implementation for SRM’s *lookahead forecasting* technique for parallel prefetching and its *forecast and flush* technique for buffer management. Our techniques amount to a significant improvement in the way SRM carries out the *parallel, independent* disk accesses necessary to efficiently read blocks of input runs during external merging.

We present the performance of SRM over a wide range of input sizes and compare its performance with that of *disk-striped mergesort (DSM)*, the commonly used technique to implement external mergesort on D parallel disks. DSM consists of using a standard mergesort algorithm in conjunction with striped I/O for parallel disk access. SRM merges together significantly more runs at a time compared with DSM, and thus it requires fewer merge passes. We demonstrate in practical scenarios that even though the streaming speeds for merging with DSM are a little higher than those for SRM (since DSM merges fewer runs at a time), sorting using SRM is significantly faster than with DSM, since SRM requires fewer passes.

The techniques in this paper can be generalized to meet the load-balancing requirements of other applications using parallel disks, including distribution sort, multiway partitioning of a file into several other files. and some potential multimedia streaming applications.

2.36 Online Data Structures in External Memory

J. S. Vitter. “Online Data Structures in External Memory,” *Proceedings of the 26th Annual International Colloquium on Automata, Languages, and Programming (ICALP '99)*, Prague, Czech

Republic, July 1999, published in Lecture Notes in Computer Science, **1644**, Springer-Verlag, Berlin, 119–133.

This survey article is superseded by a more comprehensive book 2.71. The book is available online and is recommended as the preferable reference.

Full text of ICALP '99 survey (Adobe pdf format)

Slides for ICALP '99 talk (gzip-compressed postscript)

The data sets for many of today's computer applications are too large to fit within the computer's internal memory and must instead be stored on external storage devices such as disks. A major performance bottleneck can be the input/output communication (or I/O) between the external and internal memories. In this paper we discuss a variety of online data structures for external memory, some very old and some very new, such as hashing (for dictionaries), B-trees (for dictionaries and 1-D range search), buffer trees (for batched dynamic problems), interval trees with weight-balanced B-trees (for stabbing queries), priority search trees (for 3-sided 2-D range search), and R-trees and other spatial structures. We also discuss several open problems along the way.

2.37 External Memory Algorithms with Dynamically Changing Memory Allocations

R. D. Barve and J. S. Vitter. "External Memory Algorithms with Dynamically Changing Memory Allocations," Duke University Technical Report CS-1998-09, June 1998. A shorter version of the paper appeared as "A Theoretical Framework for Memory-Adaptive Algorithms," *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS '99)*, New York, NY, October 1999, 273–284.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We consider the problem of devising external memory algorithms whose memory allocations can change dynamically and unpredictably at run-time. The investigation of "memory-adaptive" algorithms, which are designed to adapt to dynamically changing memory allocations, can be considered a natural extension of the investigation of traditional, non-adaptive external memory algorithms. Our study is motivated by high performance database systems and operating systems in which applications are prioritized and internal memory is dynamically allocated in accordance with the priorities. In such situations, external memory applications are expected to perform as well as possible for the current memory allocation. The computation must be reorganized to adapt to the sequence of memory allocations in an online manner.

In this paper we present a simple and natural dynamic memory allocation model. We define memory-adaptive external memory algorithms and specify what is needed for them to be dynamically optimal. Using novel techniques, we design and analyze dynamically optimal memory-adaptive algorithms for the problems of sorting, permuting, FFT, permutation networks, (standard) matrix multiplication and LU decomposition. We also present a dynamically optimal (in an amortized sense) memory-adaptive version of the buffer tree, a generic external memory data structure for a large number of batched dynamic applications. We show that a previously devised approach to memory-adaptive external mergesort is provably nonoptimal because of fundamental drawbacks. The lower bound proof techniques for sorting and matrix multiplication are fundamentally distinct techniques, and they are invoked by most other external memory lower bounds; hence we anticipate that the techniques presented here will apply to many external memory problems.

2.38 A Unified Approach for Indexed and Non-Indexed Spatial Joins

L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, J. Vahrenhold, and J. S. Vitter. “A Unified Approach for Indexed and Non-Indexed Spatial Joins,” *Proceedings of the 7th International Conference on Extending Database Technology (EDBT '00)*, Konstanz, Germany, March 2000.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Most spatial join algorithms either assume the existence of a spatial index structure that is traversed during the join process, or solve the problem by sorting, partitioning, or on-the-fly index construction. In this paper, we develop a simple plane-sweeping algorithm that unifies the index-based and non-index based approaches. This algorithm processes indexed as well as non-indexed inputs, extends naturally to multi-way joins, and can be built easily from a few standard operations. We present the results of a comparative study of the new algorithm with several index-based and non-index based spatial join algorithms. We consider a number of factors, including the relative performance of CPU and disk, the quality of the spatial indexes, and the sizes of the input relations. An important conclusion from our work is that using an index-based approach whenever indexes are available does not always lead to the best execution time, and hence we propose the use of a simple cost model to decide when to follow an index-based approach.

2.39 I/O-Efficient Algorithms for Problems on Grid-Based Terrains

L. Arge, L. Toma, and J. S. Vitter. “I/O-Efficient Algorithms for Problems on Grid-Based Terrains”, *Proceedings of the 2nd Workshop on Algorithm Engineering and Experimentation (ALENEX '00)*, San Francisco, CA, January 2000.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

The potential and use of Geographic Information Systems (GIS) is rapidly increasing due to the increasing availability of massive amounts of geospatial data from projects like NASA’s Mission to Planet Earth. However, the use of these massive datasets also exposes scalability problems with existing GIS algorithms. These scalability problems are mainly due to the fact that most GIS algorithms have been designed to minimize internal computation time, while I/O communication often is the bottleneck when processing massive amounts of data.

In this paper, we consider I/O-efficient algorithms for problems on grid-based terrains. Detailed grid-based terrain data is rapidly becoming available for much of the earth’s surface. We describe $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/O algorithms for several problems on \sqrt{N} by \sqrt{N} grids for which only $O(N)$ algorithms were previously known. Here M denotes the size of the main memory and B the size of a disk block.

We demonstrate the practical merits of our work by comparing the empirical performance of our new algorithm for the *flow accumulation* problem with that of the previously best known algorithm. Flow accumulation, which models flow of water through a terrain, is one of the most basic hydrologic attributes of a terrain. We present the results of an extensive set of experiments on real-life terrain datasets of different sizes and characteristics. Our experiments show that while our new algorithm scales nicely with dataset size, the previously known algorithm “breaks down” once the size of the dataset becomes bigger than the available main memory. For example, while our algorithm computes the flow accumulation for the Appalachian Mountains in about three hours, the previously known algorithm takes several weeks.

2.40 Efficient Bundle Sorting

Y. Matias, E. Segal, and J. S. Vitter. “Efficient Bundle Sorting,” *SIAM Journal on Computing*, **36**(2), 2006, 394–410. An earlier version appeared in *Proceedings of the 11th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '00)*, San Francisco, CA, January 2000, 839–848.

Full text (Adobe pdf format)

Many data sets to be sorted consist of a limited number of distinct keys. Sorting such data sets can be thought of as bundling together identical keys and having the bundles placed in order; we therefore denote this as *bundle sorting*. We describe an efficient algorithm for bundle sorting in external memory that requires at most $c(N/B)\log_{M/B}k$ disk accesses, where N is the number of keys, M is the size of internal memory, k is the number of distinct keys, B is the transfer block size, and $2 < c < 4$. For moderately sized k , this bound circumvents the $\Theta((N/B)\log_{M/B}(N/B))$ I/O lower bound known for general sorting. We show that our algorithm is optimal by proving a matching lower bound for bundle sorting. The improved running time of bundle sorting over general sorting can be significant in practice, as demonstrated by experimentation. An important feature of the new algorithm is that it is executed “in-place”, requiring no additional disk space.

2.41 Efficient Flow Computation on Massive Grid Terrains

L. Arge, J. S. Chase, L. Toma, J. S. Vitter, R. Wickremesinghe, P. Halpin, and D. Urban, “Efficient Flow Computation on Massive Grid Terrain Datasets,” *Geoinformatica*, **7**(4), December 2003, 283–313. An extended abstract appears in “Flow Computation on Massive Grids,” *Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS '01)* Atlanta, GA, November 2001, 82–87.

Full text (Adobe pdf format)

Web pages for Terraflow

As detailed terrain data becomes available, GIS terrain applications target larger geographic areas at finer resolutions. Processing the massive data involved in such applications presents significant challenges to GIS systems and demands algorithms that are optimized both for data movement and computation.

In this paper we develop efficient algorithms for flow routing on massive terrains, extending our previous work on flow accumulation. We have implemented these algorithms in the Terraflow system, which is the first comprehensive terrain flow software system designed and optimized for massive data. We compare the performance of Terraflow with that of state of the art commercial and open-source GIS systems. On large terrains, Terraflow outperforms existing systems by a factor of 2 to 1000, and is capable of solving problems no system was previously able to solve.

2.42 Distribution Sort with Randomized Cycling

J. S. Vitter and D. A. Hutchinson. “Distribution Sort with Randomized Cycling,” *Journal of the ACM*, to appear. An earlier version appeared in *Proceedings of the 12th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '01)*, Washington, DC, January 2001.

Full text (Adobe pdf format)

Parallel independent disks can enhance the performance of external memory (EM) algorithms, but the programming task is often difficult. In this paper we develop randomized variants of distribution sort for use with parallel independent disks. We propose a simple variant called *randomized cycling distribution sort* (RCD) and prove that it has optimal expected I/O complexity. The analysis uses a novel reduction to a model with significantly fewer probabilistic interdependencies. Experimental evidence is provided to support its practicality. Other simple variants are also

examined experimentally and appear to offer similar advantages to RCD. Based upon ideas in RCD we propose general techniques that transparently simulate algorithms developed for the unrealistic multihead disk model so that they can be run on the realistic parallel disk model. The simulation is optimal for two important classes of algorithms: the class of *multipass* algorithms, which make a complete pass through their data before accessing any element a second time, and the algorithms based upon the well-known distribution paradigm of EM computation.

2.43 Constrained Querying of Multimedia Databases: Issues and Approaches

A. Natsev, J. R. Smith, Y. C. Chang, C.-S. Li, and J. S. Vitter. “Constrained Querying of Multimedia Databases: Issues and Approaches,” *Proceedings of SPIE Electronic Imaging 2001: Storage and Retrieval for Image and Video Databases*, San Jose, CA, January 2001.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

This paper investigates the problem of high-level querying of multimedia data by imposing arbitrary domain-specific constraints among multimedia objects. We argue that the current structured query model, and the query-by-content model, are insufficient for many important applications, and we propose an alternative query framework that unifies and extends the previous two models. The proposed framework is based on the querying-by-concept paradigm, where the query is expressed simply in terms of concepts, regardless of the complexity of the underlying multimedia search engines. The query-by-concept paradigm was previously illustrated by the CAMEL system. The present paper builds upon and extends that work by adding arbitrary constraints and multiple levels of hierarchy in the concept representation model.

We consider queries simply as descriptions of virtual data sets, and that allows us to use the same unifying concept representation for query specification, as well as for data annotation purposes. We also identify some key issues and challenges presented by the new framework, and we outline possible approaches for overcoming them. In particular, we study the problems of concept representation, extraction, refinement, storage, and matching.

2.44 CAMEL: Concept Annotated iMagE Libraries

A. Natsev, A. Chadha, B. Soetarman, and J. S. Vitter. “CAMEL: Concept Annotated iMagE Libraries,” *Proceedings of SPIE Electronic Imaging 2001: Storage and Retrieval for Image and Video Databases*, San Jose, CA, January 2001.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

The problem of content-based image searching has received considerable attention in the last few years. Thousands of images are now available on the internet, and many important applications require searching of images in domains such as E-commerce, medical imaging, weather prediction, satellite imagery, and so on. Yet, content-based image querying is still largely unestablished as a mainstream field, nor is it widely used by search engines. We believe that two of the major hurdles for this poor acceptance are poor retrieval quality and usability.

In this paper, we introduce the CAMEL system—an acronym for Concept Annotated iMagE Libraries—as an effort to address both of the above problems. The CAMEL system provides an easy-to-use, and yet powerful, text-only query interface, which allows users to search for images based on *visual concepts*, identified by specifying relevant keywords. Conceptually, CAMEL annotates images with the visual concepts that are relevant to them. In practice, CAMEL defines visual concepts by looking at sample images off-line and extracting their relevant visual features. Once

defined, such visual concepts can be used to search for relevant images on the fly, using content-based search methods. The visual concepts are stored in a Concept Library and are represented by an associated set of wavelet features, which in our implementation were extracted by the WALRUS image querying system. Even though the CAMEL framework applies independently of the underlying query engine, for our prototype we have chosen WALRUS as a back-end, due to its ability to extract and query with image region features.

CAMEL improves retrieval quality because it allows experts to build very accurate representations of visual concepts that can be used even by novice users. At the same time, CAMEL improves usability by supporting the familiar text-only interface currently used by most search engines on the web. Both improvements represent a departure from traditional approaches to improving image query systems—instead of focusing on **query execution**, we emphasize **query specification** by allowing simpler and yet more precise query specification.

2.45 A Framework for Index Bulk Loading and Dynamization

P. K. Agarwal, L. Arge, O. Procopiuc, and J. S. Vitter. “A Framework for Index Bulk Loading and Dynamization,” *Proceedings of the 28th Annual International Colloquium on Automata, Languages, and Programming (ICALP '01)*, Crete, Greece, July 2001, published in Lecture Notes in Computer Science, **2076**, Springer-Verlag, Berlin, Germany.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We investigate automated methods for externalizing internal memory data structures. We consider a class of balanced trees that we call weight-balanced partitioning trees (or wp-trees) for indexing a set of points in d -dimensional space. Well-known examples of wp-trees include kd -trees, BBD-trees, psuedo quad trees, and BAR trees. These trees are defined with fixed degree and are thus suited for internal memory implementations. Given an efficient wp-tree construction algorithm, we present a general framework for automatically obtaining a new dynamic external tree data structure. Using this framework together with a new general construction (bulk loading) technique of independent interest, we obtain data structures with guaranteed good update performance in terms of I/O transfers. Our approach gives considerably improved construction and update I/O bounds of kd -trees and BBD trees.

2.46 Duality Between Prefetching and Queued Writing with Parallel Disks

D. A. Hutchinson, P. Sanders, and J. S. Vitter. “Duality Between Prefetching and Queued Writing with Parallel Disks,” *SIAM Journal on Computing*, **34**(6), 1443–1463, June 2005. An extended abstract appears in *Proceedings of the 9th Annual European Symposium on Algorithms (ESA '01)*, Århus, Denmark, August 2001, published in Lecture Notes in Computer Science, **2161**, Springer-Verlag, Berlin, Germany.

Full text (Adobe pdf format)

Parallel disks promise to be a cost effective means for achieving high bandwidth in applications involving massive data sets, but algorithms for parallel disks can be difficult to devise. To combat this problem, we define a useful and natural duality between writing to parallel disks and the seemingly more difficult problem of prefetching. We first explore this duality for applications involving read-once accesses using parallel disks. We get a simple linear time algorithm for computing optimal prefetch schedules and analyze the efficiency of the resulting schedules for randomly placed data and for arbitrary interleaved accesses to striped sequences. Duality also provides an optimal schedule for the integrated caching and prefetching problem, in which blocks can be accessed multi-

ple times. Another application of this duality gives us the first parallel disk sorting algorithms that are provably optimal up to lower order terms. One of these algorithms is a simple and practical variant of multiway merge sort, addressing a question that has been open for some time.

2.47 Supporting Incremental Join Queries on Ranked Inputs

A. Natsev, Y. C. Chang, J. R. Smith, C.-S. Li, and J. S. Vitter, “Supporting Incremental Join Queries on Ranked Inputs,” *Proceedings of the 27th International Conference on Very Large Databases (VLDB '01)*, Rome, Italy, September 2001.

Full text (Adobe pdf format)

This paper investigates the problem of incremental joins of multiple ranked data sets when the join condition is a list of arbitrary user-defined predicates on the input tuples. This problem arises in many important applications dealing with ordered inputs and multiple ranked data sets, and requiring the top k solutions. We use multimedia applications as the motivating examples but the problem is equally applicable to traditional database applications involving optimal resource allocation, scheduling, decision making, ranking, etc.

We propose an algorithm J^* that enables querying of ordered data sets by imposing arbitrary *user-defined join predicates*. The basic version of the algorithm does not use any random access but a J_{PA}^* variation can exploit available indexes for efficient random access based on the join predicates. A special case includes the join scenario considered by Fagin for joins based on identical keys, and in that case, our algorithms perform as efficiently as Fagin’s. Our main contribution, however, is the generalization to join scenarios that were previously unsupported, including cases where random access in the algorithm is not possible due to lack of unique keys. In addition, J^* can support *multiple join levels*, or nested join hierarchies, which are the norm for modeling multimedia data. We also give ϵ -approximation versions of both of the above algorithms. Finally, we give strong optimality results for some of the proposed algorithms, and we study their performance empirically.

2.48 Aggregate Predicate support in DBMS

A. Natsev, G. Fuh, W. Chen, C.-H. Chiu, and J. S. Vitter. “Aggregate Predicate Support in DBMS,” *Proceedings of the 13th Australasian Database Conference (ADC '02)*, Melbourne, Australia, January 2002, published in *Conferences in Research and Practice in Information Technology*, Vol. 5, Xiaofang Zhou, Ed.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we consider aggregate predicates and their support in database systems. Aggregate predicates are the predicate equivalent to aggregate functions in that they can be used to search for tuples that satisfy some aggregate property over a set of tuples (as opposed to simply computing an aggregate property over a set of tuples). The importance of aggregate predicates is exemplified by many modern applications that require ranked search, or top- k queries. Such queries are the norm in multimedia and spatial databases.

In order to support the concept of aggregate predicates in DBMS, we introduce several extensions in the query language and the database engine. Specifically, we extend the SQL syntax to handle aggregate predicates and work out the semantics of such extensions so that they behave correctly in the existing database model. We also propose a new `rk.SORT` operator into the database engine, and study relevant indexing and query optimization issues.

Our approach provides several advantages, including enhanced usability and improved performance. By supporting aggregate predicates natively in the database engine, we are able to reuse

existing indexing and query optimization techniques, without sacrificing generality or incurring the runtime overhead of database-external approaches. To the best of our knowledge, the proposed framework is the first to support user-defined indexing with aggregate predicates and search based upon user-defined ranking. We also provide empirical results from a simulation study that validates the effectiveness of our approach.

2.49 XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation

L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Parr. Submitted. An earlier version appears as “XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation,” *Proceedings of the 28th International Conference on Very Large Databases (VLDB '02)*, Hong Kong, China, August 2002.

Full text (Adobe pdf format)

The extensible mark-up language (XML) is gaining widespread use as a format for data exchange and storage on the World Wide Web. Queries over XML data require accurate selectivity estimation of path expressions to optimize query execution plans. Selectivity estimation of XML path expression is usually done based on summary statistics about the structure of the underlying XML repository. All previous methods require an off-line scan of the XML repository to collect the statistics.

In this paper, we propose XPathLearner, a method for estimating selectivity of the most commonly used types of path expressions without looking at the XML data. XPathLearner gathers and refines the statistics using query feedback in an on-line manner and is especially suited to queries in Internet scale applications since the underlying XML repositories are likely to be inaccessible or too large to be scanned entirely. Besides the on-line property, our method also has two other novel features: (a) XPathLearner is workload aware in collecting the statistics and thus can be dramatically more accurate than the more costly off-line method under tight memory constraints, and (b) XPathLearner automatically adjusts the statistics using query feedback when the underlying XML data change. We show empirically the estimation accuracy of our method using several real data sets.

2.50 Implementing I/O-Efficient Data Structures Using TPIE

L. Arge, O. Procopiuc, and J. S. Vitter. “Implementing I/O-Efficient Data Structures Using TPIE,” *Proceedings of the 10th Annual European Symposium on Algorithms (ESA '02)*, Rome, Italy, September 2002, published in Lecture Notes in Computer Science, Springer-Verlag, **2461**, Berlin, Germany, 88–100.

Full text (Adobe pdf format)

In recent years, many theoretically I/O-efficient algorithms and data structures have been developed. The TPIE project at Duke University was started to investigate the practical importance of these theoretical results. The goal of this ongoing project is to provide a portable, extensible, flexible, and easy to use C++ programming environment for efficiently implementing I/O-algorithms and data structures. The TPIE library has been developed in two phases. The first phase focused on supporting algorithms with a sequential I/O pattern, while the recently developed second phase has focused on supporting on-line I/O-efficient data structures, which exhibit a more random I/O pattern. This paper describes the design and implementation of the second phase of TPIE.

2.51 Efficient Update of Indexes for Dynamically Changing Web Documents

L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal. “Efficient Update of Indexes for Dynamically Changing Web Documents,” *World Wide Web*, **10**(1), March 2007, 37–69. An extended abstract appears in “Dynamic Maintenance of Web Indexes Using Landmarks,” *Proceedings of the 12th International World Wide Web Conference (WWW '03)*, Budapest, May 2003, 102–111.

Full text (Adobe pdf format)

Recent work on incremental crawling has enabled the indexed document collection of a search engine to be more synchronized with the changing World Wide Web. However, this synchronized collection is not immediately searchable, because the keyword index is rebuilt from scratch less frequently than the collection can be refreshed. An inverted index is usually used to index documents crawled from the web. Complete index rebuild at high frequency is expensive. Previous work on incremental inverted index updates have been restricted to adding and removing documents. Updating the inverted index for previously indexed documents that have changed has not been addressed.

In this paper, we propose an efficient method to update the inverted index for previously indexed documents whose contents have changed. Our method uses the idea of landmarks together with the `diff` algorithm to significantly reduce the number of postings in the inverted index that need to be updated. Our experiments verify that our landmark-diff method results in significant savings in the number of update operations on the inverted index.

2.52 SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads

L. Lim, M. Wang, and J. S. Vitter. “SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads,” *Proceedings of the 29th International Conference on Very Large Databases (VLDB '03)*, Berlin, Germany, September 2003.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Most RDBMSs maintain a set of histograms for estimating the selectivities of given queries. These selectivities are typically used for cost-based query optimization. While the problem of building an accurate histogram for a given attribute or attribute set has been well-studied, little attention has been given to the problem of building and tuning a set of histograms collectively for multidimensional queries in a self-managed manner based only on query feedback.

In this paper, we present SASH, a Self-Adaptive Set of Histograms that addresses the problem of building and maintaining a set of histograms. SASH uses a novel two-phase method to automatically build and maintain itself using query feedback information only. In the online tuning phase, the current set of histograms is tuned in response to the estimation error of each query in an online manner. In the restructuring phase, a new and more accurate set of histograms replaces the current set of histograms. The new set of histograms (attribute sets and memory distribution) is found using information from a batch of query feedback. We present experimental results that show the effectiveness and accuracy of our approach.

2.53 Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching

See paper 3.22.

2.54 High-Order Entropy-Compressed Text Indexes

See paper 3.23.

2.55 When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications

See paper 3.24.

2.56 Fast Compression with a Static Model in High-Order Entropy

See paper 3.25.

2.57 CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation

L. Lim, M. Wang, and J. S. Vitter. “CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation,” *Proceedings of the 31st International Conference on Very Large Databases (VLDB '05)*, Trondheim, Norway, August–September 2005.

Full text (Adobe pdf format)

Query optimization in IBM’s System RX, the first truly hybrid relational-XML data management system, requires accurate selectivity estimation of path-value pairs, i.e., the number of nodes in the XML tree reachable by a given path with the given text value. Previous techniques have been inadequate, because they have focused mainly on the tag-labeled paths (tree structure) of the XML data. For most real XML data, the number of distinct string values at the leaf nodes is orders of magnitude larger than the set of distinct rooted tag paths. Hence, the real challenge lies in accurate selectivity estimation of the string predicates on the leaf values reachable via a given path.

In this paper, we present CXHist, a novel workload-aware histogram technique that provides accurate selectivity estimation on a broad class of XML string-based queries. CXHist builds a histogram in an on-line manner by grouping queries into buckets using their true selectivity obtained from query feedback. The set of queries associated with each bucket is summarized into feature distributions. These feature distributions mimic a Bayesian classifier that is used to route a query to its associated bucket during selectivity estimation. We show how CXHist can be used for two general types of (path,string) queries: exact match queries and substring match queries. Experiments using a prototype show that CXHist provides accurate selectivity estimation for both exact match queries and substring match queries.

2.58 Compressed Data Structures: Dictionaries and the Data-Aware Measures

See paper 3.26.

2.59 An Algorithmic Framework for Compression and Text Indexing

See paper 3.28.

2.60 Online Algorithms for Prefetching and Caching in Parallel Disks

R. Shah, P. J. Varman, and J. S. Vitter. “Online Algorithms for Prefetching and Caching in Parallel Disks,” *Proceedings of the 16th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '04)*, Barcelona, Spain, June 2004.

Full text (Adobe pdf format)

Parallel disks provide a cost effective way of speeding up I/Os in applications that work with large amounts of data. The main challenge is to achieve as much parallelism as possible, using prefetching to avoid bottlenecks in disk access. Efficient algorithms have been developed for some particular patterns of accessing the disk blocks. In this paper, we consider general request sequences. When the request sequence consists of unique block requests, the problem is called prefetching and is a well-solved problem for arbitrary request sequences. When the reference sequence can have repeated references to the same block, we need to devise an effective caching policy as well. While optimum offline algorithms have been recently designed for the problem, in the online case, no effective algorithm was previously known. Our main contribution is a deterministic online algorithm threshold-LRU which achieves $O((MD/L)^{2/3})$ competitive ratio and a randomized online algorithm threshold-MARK which achieves $O(\sqrt{(MD/L) \log(MD/L)})$ competitive ratio for the caching/prefetching problem on the parallel disk model (PDM), where D is the number of disks, M is the size of fast memory buffer, and $M + L$ is the amount of lookahead available in the request sequence. The best-known lower bound on the competitive ratio is $O(\sqrt{MD/L})$ for lookahead $L \geq M$ in both models. We also show that if the deterministic online algorithm is allowed to have twice the memory of the offline then a tight competitive ratio of $O(\sqrt{MD/L})$ can be achieved. This problem generalizes the well-known paging problem on a single disk to the parallel disk model.

2.61 Bulk Operations for Space-Partitioning Trees

T. M. Ghanem, R. Shah, M. F. Mokbel, W. G. Aref, and J. S. Vitter. “Bulk Operations for Space-Partitioning Trees,” *Proceedings of the 20th Annual IEEE International Conference on Data Engineering (ICDE '04)*, Boston, March–April 2004.

Full text (Adobe pdf format)

The emergence of extensible index structures, e.g., GiST (Generalized Search Tree) and SP-GiST (Space-Partitioning Generalized Search Tree), calls for a set of extensible algorithms to support different operations (e.g., insertion, deletion, and search). Extensible bulk operations (e.g., bulk loading and bulk insertion) are of the same importance and need to be supported in these index engines. In this paper, we propose two extensible buffer-based algorithms for bulk operations in the class of space-partitioning trees; a class of hierarchical data structures that recursively decompose the space into disjoint partitions. The main idea of these algorithms is to build an in-memory tree of the target space-partitioning index. Then, data items are recursively partitioned into disk-based buffers using the in-memory tree. Although the second algorithm is designed for bulk insertion, it can be used in bulk loading as well. The proposed extensible algorithms are implemented inside SP-GiST; a framework for supporting the class of space-partitioning trees. Both algorithms have I/O bound $O(NH/B)$, where N is the number of data items to be bulk loaded/inserted, B is the number of tree nodes that can fit in one disk page, H is the tree height in terms of pages after applying a clustering algorithm. Experimental results are provided to show the scalability and applicability of the proposed algorithms for the class of space-partitioning trees. A comparison of the two proposed algorithms shows that the first algorithm performs better in case of bulk loading. However the second algorithm is more general and can be used for efficient bulk insertion.

2.62 Mining Deviants in Time Series Data Streams

S. Muthukrishnan, R. Shah, and J. S. Vitter. “Mining Deviants in Time Series Data Streams,” *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM '04)*, Santorini Island, Greece, June 2004.

Full text (Adobe pdf format)

One of the central tasks in managing, monitoring and mining data streams is that of identifying outliers. There is a long history of study of various outliers in statistics and databases, and a recent focus on mining outliers in data streams. Here, we adopt the notion of deviants from Jagadish et al as outliers. Deviants are based on one of the most fundamental statistical concept of standard deviation (or variance). Formally, deviants are defined based on a representation sparsity metric, i.e., deviants are values whose removal from the dataset leads to an improved compressed representation of the remaining items. Thus, deviants are not global maxima/minima, but rather these are appropriate local aberrations. Deviants are known to be of great mining value in time series databases. We present first-known algorithms for identifying deviants on massive data streams. Our algorithms monitor streams using very small space (polylogarithmic in data size) and are able to quickly find deviants at any instant, as the data stream evolves over time. For all versions of this problem—univariate vs multivariate time series, optimal vs nearoptimal vs heuristic solutions, offline vs streaming—our algorithms have the same framework of maintaining a hierarchical set of candidate deviants that are updated as the time series data gets progressively revealed. We show experimentally using real network traffic data (SNMP aggregate time series) as well as synthetic data that our algorithm is remarkably accurate in determining the deviants.

2.63 Rank-aware Query Optimization

I. Ilyas, R. Shah, W. G. Aref, J. S. Vitter, and A. Elmagarmid. “Rank-aware Query Optimization,” *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, Paris, France, June 2004.

Full text (Adobe pdf format)

Ranking is an important property that needs to be fully supported by current relational query engines. Recently, several rank-join query operators have been proposed based on rank aggregation algorithms. Rank-join operators progressively rank the join results while performing the join operation. The new operators have a direct impact on traditional query processing and optimization. We introduce a rank-aware query optimization framework that fully integrates rank-join operators into relational query engines. The framework is based on extending the System R dynamic programming algorithm in both enumeration and pruning. We define ranking as an interesting property that triggers the generation of rank-aware query plans. Unlike traditional join operators, optimizing for rank-join operators depends on estimating the input cardinality of these operators. We introduce a probabilistic model for estimating the input cardinality, and hence the cost of a rank-join operator. To our knowledge, this paper is the first effort in estimating the needed input size for optimal rank aggregation algorithms. Costing ranking plans, although challenging, is key to the full integration of rank-join operators in real-world query processing engines. We experimentally evaluate our framework by modifying the query optimizer of an open-source database management system. The experiments show the validity of our framework and the accuracy of the proposed estimation model.

2.64 Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data

R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. “Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data,” *Proceedings of the 30th International Conference on Very Large Databases (VLDB '04)*, Toronto, CA, August 2004.

Full text (Adobe pdf format)

It is infeasible for a sensor database to contain the exact value of each sensor at all points in time. This uncertainty is inherent in these systems due to measurement and sampling errors, and resource limitations. In order to avoid drawing erroneous conclusions based upon stale data, the use of uncertainty intervals that model each data item as a range and associated probability density function (pdf) rather than a single value has recently been proposed. Querying these uncertain data introduces imprecision into answers, in the form of probability values that specify the likelihood the answer satisfies the query. These queries are more expensive to evaluate than their traditional counterparts but are guaranteed to be correct and more informative due to the probabilities accompanying the answers. Although the answer probabilities are useful, for many applications, it is only necessary to know whether the probability exceeds a given threshold; we term these Probabilistic Threshold Queries (PTQ). In this paper we address the efficient computation of these types of queries. In particular, we develop two index structures and associated algorithms to efficiently answer PTQs. The first index scheme is based on the idea of augmenting uncertainty information to an R-tree. We establish the difficulty of this problem by mapping one-dimensional intervals to a two-dimensional space, and show that the problem of interval indexing with probabilities is significantly harder than interval indexing which is considered a well-studied problem. To overcome the limitations of this R-tree based structure, we apply a technique we call variance-based clustering, where data points with similar degrees of uncertainty are clustered together. Our extensive index structure can answer the queries for various kinds of uncertainty pdfs, in an almost optimal sense. We conduct experiments to validate the superior performance of both indexing schemes.

2.65 Efficient Join Processing over Uncertain-Valued Attributes

R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. “Efficient Join Processing over Uncertain-Valued Attributes,” *Proceedings of the 2006 ACM Conference on Information and Knowledge Management (CIKM '06)*, Arlington, VA, November 2006.

Full text (Adobe pdf format)

In an uncertain database, each data item is modeled as a range associated with a probability density function. Previous works for this kind of data have focussed on simple queries such as range and nearest-neighbor queries. Queries that join multiple relations have not been addressed in earlier work despite the significance of joins in databases. In this paper, we address probabilistic join over uncertain data, essentially a query that augments the results with probability guarantees to indicate the likelihood of each join tuple being part of the result. We extend the notion of join operators, such as equality and inequality, for uncertain data. We also study the performance of probabilistic join. We observe that a user may only need to know whether the probability of the results exceeds a given threshold, instead of the precise probability value. By incorporating this constraint, it is possible to achieve much better performance. In particular, we develop three sets of optimization techniques, namely item-level, page-level and index-level pruning, for different join operators. These techniques facilitate pruning with little space and time overhead, and are easily adapted to most join algorithms. We verify the performance of these techniques experimentally.

2.66 A Cache-Oblivious Index for Approximate String Matching

W.-K. Hon, T.-W. Lam, R. Shah, S.-L. Tam, and J. S. Vitter. “A Cache-Oblivious Index for Approximate String Matching,” *Proceedings of the 16th Annual Conference on Combinatorial Pattern Matching (CPM '07)*, London, Ontario, Canada, July 2007, published in Lecture Notes in Computer Science, **4580** Springer-Verlag, Berlin, Germany, 40–51.

Full text (Adobe pdf format)

This paper revisits the problem of indexing a text for approximate string matching. Specifically, given a text T of length n and a positive integer k , we want to construct an index of T such that for any input pattern P , we can find all its k -error matches in T efficiently. This problem is well-studied in the internal-memory setting. Here, we extend some of these recent results to external-memory solutions, which are also cache-oblivious. Our first index occupies $O((n \log kn)/B)$ disk pages and finds all k -error matches with I/Os, where B denotes the number of words in a disk page. To the best of our knowledge, this index is the first external-memory data structure that does not require I/Os. The second index reduces the space to $O((n \log n)/B)$ disk pages, and the I/O complexity is $O((|P| + occ)/B + \log^{k(k+1)} n \log \log n)$.

2.67 The SBC-tree: An Index for Run-Length Compressed Sequences

M. Y. Eltabakh, W.-K. Hon, R. Shah, W. Aref, and J. S. Vitter. “The SBC-tree: An Index for Run-Length Compressed Sequences,” *Proceedings of the 11th International Conference on Extending Database Technology (EDBT '08)*, Nantes, France, March 2008, 523–534.

Full text (Adobe pdf format)

Run-Length-Encoding (RLE) is a data compression technique that is used in various applications, e.g., biological sequence databases, multimedia, and facsimile transmission. One of the main challenges is how to operate, e.g., indexing, searching, and retrieval, on the compressed data without decompressing it. In this paper, we present the String B-tree for Compressed sequences, termed the SBC-tree, for indexing and searching RLE-compressed sequences of arbitrary length. The SBC-tree is a two-level index structure based on the well-known String B-tree and a 3-sided range query structure. The SBC-tree supports substring as well as prefix matching, and range search operations over RLE-compressed sequences. The SBC-tree has an optimal external-memory space complexity of $O(N/B)$ pages, where N is the total length of the compressed sequences, and B is the disk page size. The insertion and deletion of all suffixes of a compressed sequence of length m takes $O(m \log_B(N + m))$ I/O operations. Substring matching, prefix matching, and range search execute in an optimal $O(\log_B N + (|p| + T)/B)$ I/O operations, where $|p|$ is the length of the compressed query pattern and T is the query output size. We present also two variants of the SBC-tree: the SBC-tree that is based on an R-tree instead of the 3-sided structure, and the one-level SBC-tree that does not use a two-dimensional index. These variants do not have provable worst-case theoretical bounds for search operations, but perform well in practice. The SBC-tree index is realized inside PostgreSQL in the context of a biological protein database application. Performance results illustrate that using the SBC-tree to index RLE-compressed sequences achieves up to an order of magnitude reduction in storage, up to 30% reduction in I/Os for the insertion operations, and retains the optimal search performance achieved by the String B-tree over the uncompressed sequences.

2.68 Dynamic Rank/Select Dictionaries with Applications to XML Indexing

A. Gupta, W.-K. Hon, R. Shah, and J. S. Vitter. “Dynamic Rank/Select Dictionaries with Applications to XML Indexing,” submitted.

Full text (Adobe pdf format)

We consider a central problem in text indexing: Given a text T over an alphabet Σ , construct a compressed data structure answering the queries $access(i)$, $ranks(i)$, and $selects(i)$ for a symbol $s \in \Sigma$. Many data structures consider these queries for static text T . We consider the dynamic version of the problem, where we are allowed to insert and delete symbols at arbitrary positions of T . This problem is a key challenge in compressed text indexing and has direct application to dynamic XML indexing structures that answer subpath queries [XBW].

We build on the results of [RRR, GMR] and give the best known query bounds for the dynamic version of this problem, supporting arbitrary insertions and deletions of symbols in T . Specifically, with an amortized update time of $O((1/\epsilon)n^\epsilon)$, we suggest how to support $ranks(i)$, $selects(i)$, and $access(i)$ queries in $O((1/\epsilon)\log\log n)$ time, for any $\epsilon < 1$. The best previous query times for this problem were $O(\log n \log |\Sigma|)$, given by [Makinen Navarro]. Our bounds are competitive with state-of-the-art static structures [GMR]. Some applicable lower bounds for the partial sums problem [PD] show that our update/query tradeoff is also nearly optimal. In addition, our space bound is competitive with the corresponding static structures. For the special case of bitvectors (i.e., $|\Sigma| = 2$), we also show the best tradeoffs for query/update time, improving upon the results of [Makinen Navarro, Hon, RRR].

Our focus on fast query/slower update is well-suited for a query-intensive XML indexing environment. Using the XBW transform [XBW], we also present a dynamic data structure that succinctly maintains an ordered labeled tree T and supports a powerful set of queries on T .

2.69 A Framework for Dynamizing Succinct Data Structures

See paper 3.31.

2.70 Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing

See paper 3.32.

2.71 Algorithms and Data Structures for External Memory—*main reference!*

J. S. Vitter. *Algorithms and Data Structures for External Memory*, Series on Foundations and Trends in Theoretical Computer Science, now Publishers, Hanover, MA, 2008. Also published as Volume 2, Issue 4 of *Foundations and Trends in Theoretical Computer Science*.

Book (Adobe pdf format)

Slides for a talk (Adobe pdf format)

Data sets in large applications are often too massive to fit completely inside the computer's internal memory. The resulting input/output communication (or I/O) between fast internal memory and slower external memory (such as disks) can be a major performance bottleneck. In this book we discuss the state of the art in the design and analysis of *external memory* (or *EM*) *algorithms and data structures*, where the goal is to exploit locality in order to reduce the I/O costs. We consider a variety of EM paradigms for solving batched and online problems efficiently in external memory.

For the batched problem of sorting and related problems like permuting and fast Fourier transform, the key paradigms include distribution and merging. The paradigm of disk striping offers an elegant way to use multiple disks in parallel. For sorting, however, disk striping can be nonoptimal with respect to I/O, so to gain further improvements we discuss prefetching, distribution, and merging techniques for using the disks independently. We also consider useful techniques for batched

EM problems involving matrices (such as matrix multiplication and transposition), geometric data (such as finding intersections and constructing convex hulls) and graphs (such as list ranking, connected components, topological sorting, and shortest paths). In the online domain, canonical EM applications include dictionary lookup and range searching. The two important classes of indexed data structures are based upon extendible hashing and B-trees. The paradigms of filtering and bootstrapping provide a convenient means in online data structures to make effective use of the data accessed from disk. We also reexamine some of the above EM problems in slightly different settings, such as when the data items are moving, when the data items are variable-length (e.g., text strings), when the internal data representations are compressed, or when the allocated amount of internal memory can change dynamically.

Programming tools and environments are available for simplifying the EM programming task. During the course of the book, we report on some experiments in the domain of spatial databases using the TPIE system (Transparent Parallel I/O programming Environment). The newly developed EM algorithms and data structures that incorporate the paradigms we discuss are significantly faster than methods currently used in practice.

This book is an expanded version of an earlier survey article 2.30.

2.72 On Searching Compressed String Collections Cache-Obliviously

P. Ferragina, R. Grossi, A. Gupta, R. Shah, and J. S. Vitter. “On Searching Compressed String Collections Cache-Obliviously,” *Proceedings of the 27th Annual ACM Symposium on Principles of Database Systems (PODS '08)*, Vancouver, Canada, June 2008.

Full text (Adobe pdf format)

Current data structures for searching large string collections are limited in that they either fail to achieve minimum space or they cause too many cache misses. In this paper, we discuss some edge linearizations of the classic trie data structure that are simultaneously cache-friendly and storable in compressed space. The widely known frontcoding scheme is one example of linearization; it is at the core of Prefix B-trees and many other disk-conscious compressed indexes for string collections. However, it is largely thought of as a space-effective heuristic without efficient search support.

In this paper, we introduce new insights on front-coding and other novel linearizations, and study how close their space occupancy is to the information-theoretic minimum. The moral is that they are not just heuristics. The second contribution of this paper engineers these linearizations to design a novel dictionary encoding scheme that achieves nearly optimal space, offers competitive I/O-search time, and is also conscious of the query distribution. Finally, we combine those data structures with cache-oblivious tries and obtain a succinct variant, whose space is close to the information-theoretic minimum.

2.73 On Entropy-Compressed Text Indexing in External Memory

See paper 3.35.

3 DATA COMPRESSION

3.1 Design and Analysis of Dynamic Huffman Codes

J. S. Vitter. “Design and Analysis of Dynamic Huffman Codes,” *Journal of the ACM*, 34(4), October 1987, 825–845. A shortened version appears in “The Design and Analysis of Dynamic

Huffman Coding,” *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS '85)*, Portland, OR, October 1985, 293–302.

Publication version (Adobe pdf format)

Technical report version (gzip-compressed postscript)

We introduce and analyze a new one-pass algorithm for constructing dynamic Huffman codes and also analyze the one-pass algorithm due to Faller, Gallager, and Knuth. In each algorithm, both the sender and the receiver maintain equivalent dynamically varying Huffman trees, and the coding is done in real time. We show that the number of bits used by the new algorithm to encode a message containing t letters is $< t$ bits more than that used by the conventional two-pass Huffman scheme, independent of the alphabet size. This is best possible in the worst case, for any one-pass Huffman method. Tight upper and lower bounds are derived. Empirical tests show that the encodings produced by the new algorithm are shorter than those of the other one-pass algorithm and, except for long messages, are shorter than those of the two-pass method. The new algorithm is well-suited for online encoding/decoding in data networks and for file compression.

3.2 ALGORITHM 673 Dynamic Huffman Coding

J. S. Vitter. “ALGORITHM 673 Dynamic Huffman Coding,” *ACM Transactions on Mathematical Software*, 15(2), June 1989, 158–167. Also appears in *Collected Algorithms of ACM*.

Publication version (Adobe pdf format)

Tech report version (gzip-compressed postscript)

We present a Pascal implementation of the one-pass algorithm for constructing dynamic Huffman codes that is described and analyzed in a companion paper [Vitter, 1987]. The program runs in real time; that is, the processing time for each letter of the message is proportional to the length of its codeword. The number of bits used to encode a message of t letters is less than t bits more than that used by the well-known two-pass algorithm. This is best possible for any one-pass Huffman scheme. In practice it uses fewer bits than all other Huffman schemes. The algorithm has applications in file compression and network transmission.

3.3 Analysis of Arithmetic Coding for Data Compression

P. G. Howard and J. S. Vitter. “Analysis of Arithmetic Coding for Data Compression,” invited paper in special issue on data compression for images and texts in *Information Processing and Management*, 28(6), 1992, 749–763. A shortened version appears in an invited paper in *Proceedings of the 1991 IEEE Data Compression Conference (DCC '91)*, Snowbird, UT, April 1991, 3–12.

Full text (Adobe pdf format)

3.4 New Methods for Lossless Image Compression Using Arithmetic Coding

P. G. Howard and J. S. Vitter. “New Methods for Lossless Image Compression Using Arithmetic Coding,” invited paper in special issue on data compression for image and text in *Journal of Information Processing and Management*, 28(6), 1992, 765–779. A shorter and earlier version appears in *Proceedings of the 1991 IEEE Data Compression Conference (DCC '91)*, Snowbird, UT, April 1991, 257–266.

Not yet available.

3.5 Practical Implementations of Arithmetic Coding

P. G. Howard and J. S. Vitter. “Practical Implementations of Arithmetic Coding,” invited paper in *Images and Text Compression*, Kluwer Academic Publishers, 1992. A shortened version appears as an invited paper in *Proceedings of the 3rd International Conference on Advances in Communication and Control Systems (COMCON '91)*, Victoria, Canada, October 1991.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

3.6 Error Modeling for Hierarchical Lossless Image Compression

P. G. Howard and J. S. Vitter. “Error Modeling for Hierarchical Lossless Image Compression,” *Proceedings of the 1992 IEEE Data Compression Conference (DCC '92)*, Snowbird, UT, March 1992, 269–278.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a new method for error modeling applicable to the MLP algorithm for hierarchical lossless image compression. This method, based on a concept called the *variability index*, provides accurate models for pixel prediction errors without requiring explicit transmission of the models. We also use the variability index to show that prediction errors do not always follow the Laplace distribution, as is commonly assumed; replacing the Laplace distribution with a more general symmetric exponential distribution further improves compression. We describe a new compression measurement called *compression gain*, and we give experimental results showing that the MLP method using the variability index technique for error modeling gives significantly more compression gain than other methods in the literature.

3.7 Optimal Prefetching via Data Compression

See paper 4.3.

3.8 Optimal Prediction for Prefetching in the Worst Case

See paper 4.5.

3.9 Parallel Lossless Image Compression Using Huffman and Arithmetic Coding

P. G. Howard and J. S. Vitter. “Parallel Lossless Image Compression Using Huffman and Arithmetic Coding,” *Information Processing Letters*, **59**, 1996, 65–73. A shortened version appears in *Proceedings of the 1992 IEEE Data Compression Conference (DCC '92)*, Snowbird, UT, March 1992, 299–308.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We show that high-resolution images can be encoded and decoded efficiently in parallel. We present an algorithm based on the hierarchical MLP method, used either with Huffman coding or with a new variant of arithmetic coding called *quasi-arithmetic coding*. The coding step can be parallelized, even though the codes for different pixels are of different lengths; parallelization of the prediction and error modeling components is straightforward.

3.10 Nearly Optimal Vector Quantization via Linear Programming

J.-H. Lin and J. S. Vitter. “Nearly Optimal Vector Quantization via Linear Programming,” *Proceedings of the 1992 IEEE Data Compression Conference (DCC '92)*, Snowbird, UT, March 1992, 22–31.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present new vector quantization algorithms based on the theory developed by the authors. The new approach is to formulate a vector quantization problem as a 0-1 integer linear program. We first solve its relaxed linear program by linear programming techniques. Then we transform the linear program solution into a provably good solution for the vector quantization problem. These methods lead to the first known polynomial-time full-search vector quantization codebook design algorithm and tree pruning algorithm with provable worst-case performance guarantees. We also introduce the notion of *pseudo-random pruned tree-structured vector quantizers*. Initial experimental results on image compression are very encouraging.

3.11 Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding

P. G. Howard and J. S. Vitter. “Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding,” *Journal of Information Processing and Management*, **30**(6), 1994, 777–790. A shortened version appears in *Proceedings of the 1993 IEEE Data Compression Conference (DCC '93)*, Snowbird, UT, April 1993.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We give a detailed algorithm for fast text compression. Our algorithm, related to the PPM method, simplifies the modeling phase by eliminating the *escape* mechanism, and speeds up coding by using a combination of *quasi-arithmetic coding* and Rice coding. We provide details of the use of quasi-arithmetic code tables, and analyze their compression performance. Our *Fast PPM* method is shown experimentally to be almost twice as fast as the PPMC method, while giving comparable compression.

3.12 Fast and Efficient Lossless Image Compression

P. G. Howard and J. S. Vitter. “Fast and Efficient Lossless Image Compression,” *Proceedings of the 1993 IEEE Data Compression Conference (DCC '93)*, Snowbird, UT, April 1993.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a new method for lossless image compression that gives compression comparable to JPEG lossless mode with about five times the speed. Our method, called *FELICS*, is based on a novel use of two neighboring pixels for both prediction and error modeling. For coding we use single bits, adjusted binary codes, and Golomb or Rice codes. For the latter we present and analyze a provably good method for estimating the single coding parameter. (Note: This method is the foundation for the subsequently developed state-of-the-art methods now used for lossless image compression.)

3.13 Fast Progressive Lossless Image Compression

P. G. Howard and J. S. Vitter. “Fast Progressive Lossless Image Compression,” *Proceedings of the 1994 IST/SPIE International Symposium on Electronic Imaging Science and Technology*, San Jose, CA, February 1994.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a method for progressive lossless compression of still grayscale images that combines the speed of our earlier FELICS method with the progressivity of our earlier MLP method. We use MLP’s pyramid-based pixel sequence, and image and error modeling and coding based on that of FELICS. In addition, we introduce a new prefix code with some advantages over the previously used Golomb and Rice codes. Our new progressive method gives compression ratios and speeds similar to those of non-progressive FELICS and those of JPEG lossless mode, also a non-progressive method.

The image model in Progressive FELICS is based on a simple function of four nearby pixels. We select two of the four nearest known pixels, using the two with the middle (non-extreme) values. Then we code the pixel’s intensity relative to the selected pixels, using single bits, adjusted binary codes, and simple prefix codes like Golomb codes, Rice codes, or the new family of prefix codes introduced here. We estimate the coding parameter adaptively for each context, the context being the absolute value of the difference of the predicting pixels; we adjust the adaptation statistics at the beginning of each level in the progressive pixel sequence.

3.14 Arithmetic Coding for Data Compression

P. G. Howard and J. S. Vitter. “Arithmetic Coding for Data Compression,” *Proceedings of the IEEE*, 82(6), June 1994, 857–865.

Full text (Adobe pdf format)

Arithmetic coding provides an effective mechanism for removing redundancy in the encoding of data. We show how arithmetic coding works and describe an efficient implementation that uses table lookup as a fast alternative to arithmetic operations. The reduced-precision arithmetic has a provably negligible effect on the amount of compression achieved. We can speed up the implementation further by use of parallel processing. We discuss the role of probability models and how they provide probability information to the arithmetic coder. We conclude with perspectives on the comparative advantages and disadvantages of arithmetic coding.

3.15 Explicit Bit Minimization for Motion-Compensated Video Coding

D. T. Hoang, P. M. Long, and J. S. Vitter, “Explicit Bit Minimization for Motion-Compensated Video Coding,” *Proceedings of the 1994 IEEE Data Compression Conference (DCC '94)*, Snowbird, UT, March 1994, 175–184.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We compare methods for choosing motion vectors for motion-compensated video compression. Our primary focus is on videophone and videoconferencing applications, where very low bit rates are necessary, where the motion is usually limited, and where the frames must be coded in the order they are generated. We provide evidence, using established benchmark videos of this type, that choosing motion vectors to minimize codelength subject to (implicit) constraints on quality yields substantially better rate-distortion tradeoffs than minimizing notions of prediction error. We illustrate this point using an algorithm within the $p \times 64$ standard. We show that using quadrees

to code the motion vectors in conjunction with explicit codelength minimization yields further improvement. We describe a dynamic-programming algorithm for choosing a quadtree to minimize the codelength.

3.16 Dictionary Selection using Partial Matching

D. T. Hoang, P. M. Long and J. S. Vitter. “Dictionary Selection using Partial Matching,” *Information Sciences*, **119**(1–2), 57–72, 1999. An earlier version appeared in “Multiple-Dictionary Compression using Partial Matching,” *Proceedings of the 1995 IEEE Data Compression Conference (DCC '95)*, Snowbird, UT, March 1995, 272–281.

Full text (Adobe pdf format)

Motivated by the desire to find text compressors that compress better than existing dictionary methods, but run faster than PPM implementations, we describe methods for text compression using multiple dictionaries, one for each context of preceding characters, where the contexts have varying lengths. The context to be used is determined using an escape mechanism similar to that of PPM methods. We describe modifications of three popular dictionary coders along these lines and experiments evaluating their efficacy using the text files in the Calgary corpus. Our results suggest that modifying LZ77 along these lines yields an improvement in compression of about 4%, that modifying LZFG yields a compression improvement of about 8%, and that modifying LZW in this manner yields an average improvement on the order of 12%.

Keywords: Text compression, prediction by partial matching, Ziv-Lempel coding, Lempel-Ziv coding, LZ coding.

3.17 Rate-Distortion Optimizations for Motion Estimation in Low-Bitrate Video Coding

D. T. Hoang, P. M. Long, and J. S. Vitter. “Rate-Distortion Optimizations for Motion Estimation in Low-Bitrate Video Coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, **8**(4), August 1998, 488–500. A shorter version appears in *Proceedings of the Digital Video Compression Conference, IS&T/SPIE 1996 Symposium on Electronic Imaging: Science & Technology*, **2668**, San Jose, CA, January–February 1996, 18–27.

Full text (Adobe pdf format)

We present and compare methods for choosing motion vectors for motion-compensated video coding. Our primary focus is on videophone and videoconferencing applications, where very low bit rates are necessary, where motion is usually limited, and where frames must be coded in the order they are generated. We provide evidence, using established benchmark videos typical of these applications, that choosing motion vectors explicitly to minimize rate, subject to implicit constraints on distortion, yields better rate-distortion tradeoffs than minimizing notions of prediction error. Minimizing a linear combination of rate and distortion results in further rate-distortion improvements. Using a heuristic function of the prediction error and the motion vector codelength results in compression performance comparable to the more computationally intensive coders while running much faster. We incorporate these ideas into coders that operate within the $p \times 64$ standard.

3.18 Efficient Cost Measures for Motion Compensation at Low Bit Rates

D. T. Hoang, P. M. Long, and J. S. Vitter. “Efficient Cost Measures for Motion Compensation at Low Bit Rates,” *Proceedings of the 1996 IEEE Data Compression Conference (DCC '96)*, Snowbird, UT, April 1996, 102–111.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present and compare methods for choosing motion vectors for block-based motion-compensated video coding. The primary focus is on videophone and video-conferencing applications, where low bit rates are necessary, where motion is usually limited, and where the amount of computation is also limited. In a typical block-based motion-compensated video coding system, motion vectors are transmitted along with a lossy encoding of the residuals. As the bit rate decreases, the proportion required to transmit the motion vectors increases. We provide experimental evidence that choosing motion vectors explicitly to minimize rate (including motion vector coding), subject to implicit constraints on distortion, yields better rate-distortion tradeoffs than minimizing some measure of prediction error. Minimizing a combination of rate and distortion yields further improvements. Although these explicit-minimization schemes are computationally intensive, they provide invaluable insight which we use to develop practical algorithms. We show that minimizing a simple heuristic function of the prediction error and the motion vector code-length results in rate-distortion performance comparable to explicit-minimization schemes while being computationally feasible. Experimental results are provided for coders that operate within the H.261 standard.

3.19 Lexicographic Bit Allocation for MPEG Video

D. T. Hoang, E. Linzer, and J. S. Vitter, “Lexicographic Bit Allocation for MPEG Video,” Special issue on high-fidelity media processing in *Journal of Visual Communication and Image Representation*, **8**(4), December 1997, 384–404. An extended abstract appears in *Proceedings of the 1997 IEEE International Conference on Image Processing (ICIP’97)*, Santa Barbara, CA, October 1997. An earlier version appears in “A Lexicographic Framework for MPEG Rate Control,” *Proceedings of the 1997 IEEE Data Compression Conference (DCC ’97)*, Snowbird, UT, March 1997, 101–110.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

3.20 Text Compression via Alphabet Re-representation

P. M. Long, A. I. Natsev, and J. S. Vitter, “Text Compression via Alphabet Re-representation,” *Proc. Data Compression Conference (DCC ’97)*, Snowbird, Utah, March 1997.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We consider re-representing the alphabet so that a representation of a character reflects its properties as a predictor of future text. This enables us to use an estimator from a restricted class to map contexts to predictions of upcoming characters. We describe an algorithm that uses this idea in conjunction with neural networks. The performance of this implementation is compared to other compression methods, such as UNIX compress, gzip, PPMC, and an alternative neural network approach.

3.21 Efficient Algorithms for MPEG Video Compression

D. T. Hoang and J. S. Vitter *Efficient Algorithms fo MPEG Video Compression*, John Wiley & Sons, New York, NY, 2002.

Wiley web site

Discount price at bookpool.com

Unofficial version (Adobe pdf format)

Video belongs to a class of information called *continuous media*. Continuous media is characterized by the essentially continuous manner in which the information is presented. This is in contrast to *discrete media*, in which there is no essential temporal component. Text, images, and graphics are examples of discrete media, while movies, sound, and computer animation are examples of continuous media. Even though a slide show is a time-based presentation of images, it is not a continuous medium since each image is viewed as an individual item. On the other hand, a video clip, while also consisting of a sequence of images, is a continuous medium since each image is perceived in the context of past and future images.

With continuous media, therefore, the temporal dimension becomes important. For example, a video sequence compressed with a constant image quality for every frame is often more desirable than one in which the image quality varies noticeably over time. However, because the compressibility of individual frames varies over time, maintaining a constant image quality results in a variation in coding rate over time. The process of controlling the coding rate to meet the requirements of a transmissions channel or storage device, while maintaining a desired level of quality, is called *bit rate control*. In this monograph, we focus on the rate control of compressed video. Specifically, we present a new framework for allocating bits to the compression of pictures in a video sequence.

Existing optimal rate control techniques typically regulate the coding rate to minimize a sum-distortion measure. While these techniques can leverage the wealth of tools from least-mean-square optimization theory, they do not guarantee constant-quality video, an objective often mentioned in the literature. In this book, we develop a framework that casts rate control as a resource allocation problem with continuous variables, nonlinear constraints, and a novel lexicographic optimality criterion that is motivated for uniform video quality. With the lexicographic criterion, we propose a new concept of coding efficiency to better reflect the constancy in quality that is generally desired from a video coder.

Rigorous analysis within this framework reveals a set of necessary and sufficient conditions for optimality for coding at both constant and variable bit rates. With these conditions, we are able to construct polynomial-time algorithms for optimal bit rate control. Experimental implementations of these algorithms confirm the theoretical analysis and produce encodings that are more uniform in quality than that achieved with existing rate control methods. As evidence of the generality and flexibility of the framework, we show how to extend the framework to allocate bits among multiple variable bit rate bitstreams that are to be transmitted over a common constant bit rate channel and to encompass the case of discrete variables.

3.22 Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching

R. Grossi and J. S. Vitter. “Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching,” *SIAM Journal on Computing*, **35**(2), 2005, 378–407. An extended abstract of the first appears in *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC '00)*, Portland, OR, May 2000, 397–406.

Full text (Adobe pdf format)

Slides for talk (gzip-compressed postscript)

Slides for talk (Adobe pdf format)

The proliferation of online text, such as on the World Wide Web and in databases, motivates the need for space-efficient index methods that support fast search. Consider a text T of n binary symbols to index. Given any query pattern P of m binary symbols, the goal is to search for P in T quickly, with T being fully scanned only once, namely, when the index is created. All indexing schemes published in the last thirty years support searching in $\Theta(m)$ worst-case time and require

$\Theta(n)$ memory words (or $\Theta(n \log n)$ bits), which is significantly larger than the text itself. In this paper we provide a breakthrough both in searching time and index space under the same model of computation as the one adopted in previous work. Based upon new compressed representations of suffix arrays and suffix trees, we construct an index structure that occupies only $O(n)$ bits and compares favorably with inverted lists in space. We can search any binary pattern P , stored in $O(m/\log n)$ words, in only $o(m)$ time.

Specifically, searching takes $O(1)$ time for $m = o(\log n)$, and $O(m/\log n + \log^\epsilon n) = o(m)$ time for $m = \Omega(\log n)$ and any fixed $0 < \epsilon < 1$. That is, we achieve optimal $O(m/\log n)$ search time for sufficiently large $m = \Omega(\log^{1+\epsilon} n)$. We can list all the *occ* pattern occurrences in optimal $O(\text{occ})$ additional time when $m = \Omega(\text{polylog}(n))$ or when $\text{occ} = \Omega(n^\epsilon)$; otherwise, listing takes $O(\text{occ} \log^\epsilon n)$ additional time.

3.23 High-Order Entropy-Compressed Text Indexes

R. Grossi, A. Gupta, and J. S. Vitter. “High-Order Entropy-Compressed Text Indexes,” *Proceedings of the 14th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '03)*, Baltimore, MD, January 2003, 841–850.

Full text (Adobe pdf format) of the SODA 2003 conference paper

We present a novel implementation of compressed suffix arrays exhibiting new tradeoffs between search time and space occupancy for a given text (or sequence) of n symbols over an alphabet Σ , where each symbol is encoded by $\log |\Sigma|$ bits. We show that compressed suffix arrays use just $nH_h + O(n \log \log n / \log_{|\Sigma|} n)$ bits, while retaining full text indexing functionalities, such as searching any pattern sequence of length m in $O(m \log |\Sigma| + \text{polylog}(n))$ time. The term $H_h \leq \log |\Sigma|$ denotes the h th-order empirical entropy of the text, which means that our index is nearly optimal in space apart from lower-order terms, achieving asymptotically the empirical entropy of the text (with a multiplicative constant 1). If the text is highly compressible so that $H_n = o(1)$ and the alphabet size is small, we obtain a text index with $o(m)$ search time that requires only $o(n)$ bits. We also report further results and tradeoffs on high-order entropy-compressed text indexes.

3.24 Indexing Equals Compression: Experiments on Suffix Arrays and Trees

L. Foschini, R. Grossi, A. Gupta, and J. S. Vitter. “Indexing Equals Compression: Experiments on Suffix Arrays and Trees,” *ACM Transactions on Algorithms*, **2**(4), 2006, 611–639. An extended abstract appears in R. Grossi, A. Gupta, and J. S. Vitter, “When Indexing Equals Compression: Experiments with Compressing Suffix Arrays and Applications,” *Proceedings of the 15th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '04)*, New Orleans, LA, January 2004, 636–645..

Full text (Adobe pdf format)

We report on a new and improved version of high-order entropy-compressed suffix arrays, which has theoretical performance guarantees comparable to previous work, yet represents an improvement in practice. Our experiments indicate that the resulting text index offers state-of-the-art compression. In particular, we require roughly 20% of the original text size—without requiring a separate instance of the text—and support fast and powerful searches. To our knowledge, this is the best known method in terms of space for fast searching. We can additionally use a simple notion to encode and decode block-sorting transforms (such as the Burrows-Wheeler transform), achieving a slightly better compression ratio than `bzip2`. We also provide a compressed representation of suffix trees (and their associated text) in a total space that is comparable to that of the text alone compressed with `gzip`.

3.25 Fast Compression with a Static Model in High-Order Entropy

L. Foschini, R. Grossi, A. Gupta, and J. S. Vitter. “Fast Compression with a Static Model in High-Order Entropy.” *Proceedings of the 2004 IEEE Data Compression Conference (DCC '04)*, Snowbird, UT, March 2004, 23–25.

Full text (Adobe pdf format)

We report on a simple encoding format called `wzip` for decompressing block-sorting transforms, such as the Burrows-Wheeler Transform (BWT). Our compressor uses the simple notions of gamma encoding and RLE organized with a wavelet tree to achieve a slightly better compression ration than `bzip2` in less time. In fact, our compression/decompression time is dependent upon H_h , the empirical h th order entropy. Another key contribution of our compressor is its simplicity. Our compressor can also operate as a full-text index with a small amount of data, while still preserving backward compatibility with just the compressor.

3.26 Compressed Data Structures: Dictionaries and the Data-Aware Measures

A. Gupta, W. Hon, R. Shah, and J. S. Vitter. “Compressed Data Structures: Dictionaries and the Data-Aware Measures,” *Theoretical Computer Science*, **387**(3), November 2007, 313–331.

Full text (Adobe pdf format)

We propose measures for compressed data structures, in which space usage is measured in a data-aware manner. In particular, we consider the fundamental *dictionary problem on set data*, where the task is to construct a data structure to represent a set S of n items out of a universe $U = \{0, 1, \dots, u - 1\}$ and support various queries on S . We use a well-known data-aware measure for set data called *gap* to bound the space of our data structures. We describe a novel dictionary structure taking $gap + O(n \log(u/n) / \log n) + O(n \log \log(u/n))$ bits. Under the RAM model, our dictionary supports membership, rank, select, and predecessor queries in nearly optimal time, matching the time bound of Andersson and Thorup’s predecessor structure, while simultaneously improving upon their space usage. Our dictionary structure uses exactly *gap* bits in the leading term (i.e., the constant factor is 1) and answers queries in near-optimal time. When seen from the worst case perspective, we present the first $O(n \log(u/n))$ -bit dictionary structure which supports these queries in near-optimal time under RAM model. We also build a dictionary which requires the same space and supports membership, select, and partial rank queries even more quickly in $O(\log \log n)$ time. To the best of our knowledge, this is the first of a kind result which achieves data-aware space usage and retains near-optimal time.

3.27 Compressed Dictionaries: Space Measures, Data Sets, and Experiments

A. Gupta, W.-K. Hon, R. Shah, and J. S. Vitter. “Compressed Dictionaries: Space Measures, Data Sets, and Experiments,” in *Proceedings of the 5th International Workshop on Experimental Algorithms (WEA '06)*, Menorca, Spain, May 2006, 158–169.

Full text (Adobe pdf format)

We present an experimental study of the space-time tradeoffs for the dictionary problem. Our primary goal is to reduce the space requirement for storing a dictionary data structure. Many compression schemes have been developed for dictionaries, which fall generally in the categories of combinatorial encodings and data-aware methods and still support queries efficiently. We show that for many real-world datasets, data-aware methods lead to a worthwhile compression over combinatorial methods. Additionally, we design a new data-aware building block structure called BSGAP that presents improvements over other data-aware methods.

3.28 An Algorithmic Framework for Compression and Text Indexing

R. Grossi, A. Gupta, and J. S. Vitter. “An Algorithmic Framework for Compression and Text Indexing,” being submitted to journal. This work is an extension of two separate pieces of research in conference form: “High-Order Entropy-Compressed Text Indexes,” *Proceedings of the 14th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '03)*, Baltimore, MD, January 2003, 841–850; and “A Nearly Tight Analysis of the Burrows Wheeler Transform,” *Proceedings of the 5th Workshop on Analytical Algorithmics and Combinatorics (ANALCO '08)*, San Francisco, CA, January 2008.

Full text (Adobe pdf format)

We present a unified algorithmic framework to obtain nearly optimal space bounds for text compression and compressed text indexing, apart from lower-order terms. For a text T of n symbols drawn from an alphabet Σ , our bounds are stated in terms of the h th-order empirical entropy of the text, H_h . In particular, we provide a tight analysis of the Burrows-Wheeler transform (BWT) establishing a bound of $nH_h + M(T, \Sigma, h)$ bits, where $M(T, \Sigma, h)$ denotes the asymptotical number of bits required to store the empirical statistical model for contexts of order up to h appearing in T . Using the same framework, we also obtain an implementation of the compressed suffix array (CSA) which achieves $nH_h + M(T, \Sigma, h) + O(n \lg \lg n / \lg_{|\Sigma|} n)$ bits of space while still retaining competitive full-text indexing functionality.

The novelty of the proposed framework lies in its use of the finite set model instead of the empirical probability model (as in previous work), giving us new insight into the design and analysis of our algorithms. For example, we show that our analysis gives improved bounds since $M(T, \Sigma, h) \leq \min\{g'_h \lg(n/g'_h + 1), H_h^* n + \lg n + g''_h\}$, where $g'_h = O(|\Sigma|^{h+1})$ and $g''_h = O(|\Sigma|^{h+1} \lg |\Sigma|^{h+1})$ do not depend on the text length n , while $H_h^* \geq H_h$ is the modified h th-order empirical entropy of T . Moreover, we show a strong relationship between a compressed full-text index and the succinct dictionary problem. We also examine the importance of lower-order terms, as these can dwarf any savings achieved by high-order entropy. We report further results and tradeoffs on high-order entropy-compressed text indexes in the paper.

3.29 SBC-tree: Efficient Indexing for RLE-Compressed Strings

See paper 2.67.

3.30 Dynamic Rank/Select Dictionaries with Applications to XML Indexing

See paper 2.68.

3.31 A Framework for Dynamizing Succinct Data Structures

A. Gupta, W.-K. Hon, R. Shah, and J. S. Vitter. “A Framework for Dynamizing Succinct Data Structures,” *Proceedings of the 34th Annual International Colloquium on Automata, Languages, and Programming (ICALP '07)*, Wrocław, Poland, July 2007, published in *Lecture Notes in Computer Science*, **4596** Springer-Verlag, Berlin, Germany, 521-532.

Full text (Adobe pdf format)

We present a framework to dynamize succinct data structures, to encourage their use over non-succinct versions in a wide variety of important application areas. Our framework can dynamize most state-of-the-art succinct data structures for dictionaries, ordinal trees, labeled trees, and text collections. Of particular note is its direct application to XML indexing structures that answer

subpath queries. Our framework focuses on achieving information-theoretically optimal space along with near-optimal update/query bounds.

As the main part of our work, we consider the following problem central to text indexing: Given a text T over an alphabet Σ , construct a compressed data structure answering the queries $access(i)$, $ranks(i)$, and $selects(i)$ for a symbol $s \in \Sigma$. Many data structures consider these queries for static text T . We build on these results and give the best known query bounds for the dynamic version of this problem, supporting arbitrary insertions and deletions of symbols in T .

Specifically, with an amortized update time of $O(n^\epsilon)$, any static succinct data structure D for T , taking $t(n)$ time for queries, can be converted by our framework into a dynamic succinct data structure that supports $ranks(i)$, $mathopselects(i)$, and $access(i)$ queries in $O(t(n) + \log \log n)$ time, for any constant $\epsilon > 0$. When $|\Sigma| = \text{polylog}(n)$, we achieve $O(1)$ query times. Our update/query bounds are near-optimal with respect to the lower bounds.

3.32 Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing

Y.-F. Chien, W.-K. Hon, R. Shah, and J. S. Vitter. “Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing,” *Proceedings of the 2008 IEEE Data Compression Conference (DCC '08)*, Snowbird, UT, March 2008.

Full text (Adobe pdf format)

We introduce a new variant of the popular Burrows-Wheeler transform (BWT) called Geometric Burrows-Wheeler Transform (GBWT). Unlike BWT, which merely permutes the text, GBWT converts the text into a set of points in 2-dimensional geometry. Using this transform, we can answer to many open questions in compressed text indexing: (1) Can compressed data structures be designed in external memory with similar performance as the uncompressed counterparts? (2) Can compressed data structures be designed for position restricted pattern matching? We also introduce a reverse transform, called Points2Text, which converts a set of points into text. This transform allows us to derive the best known lower bounds in compressed text indexing. We show strong equivalence between data structural problems in geometric range searching and text pattern matching. This provides a way to derive new results in compressed text indexing by translating the results from range searching.

3.33 Compressed Index for Dictionary Matching

W.-K. Hon, T.-W. Lam, R. Shah, S.-L. Tam, and J. S. Vitter. “Compressed Index for Dictionary Matching,” *Proceedings of the 2008 IEEE Data Compression Conference (DCC '08)*, Snowbird, UT, March 2008.

Full text (Adobe pdf format)

The past few years have witnessed several exciting results on compressed representation of a string T that supports efficient pattern matching, and the space complexity has been reduced to $|T|H_k(T) + o(|T| \log \sigma)$ bits, where $H_k(T)$ denotes the k th-order empirical entropy of T , and σ is the size of the alphabet. In this paper we study compressed representation for another classical problem of string indexing, which is called dictionary matching in the literature. Precisely, a collection D of strings (called patterns) of total length n is to be indexed so that given a text T , the occurrences of the patterns in T can be found efficiently. In this paper we show how to exploit a sampling technique to compress the existing $O(n)$ -word index to an $(nH_k(D) + o(n \log \sigma))$ -bit index with only a small sacrifice in search time.

3.34 On Searching Compressed String Collections Cache-Obliviously

See paper 2.72.

3.35 On Entropy-Compressed Text Indexing in External Memory

W.-K. Hon, R. Shah, S. V. Thankachan, and J. S. Vitter. “On Entropy-Compressed Text Indexing in External Memory,” *Proceedings of the 16th International Conference on String Processing and Information Retrieval (SPIRE '09)*, Saariselkä, Finland, August 2009, published in Lecture Notes in Computer Science, **5721** Springer-Verlag, Berlin, Germany, 75–89.

Full text (Adobe pdf format)

A new trend in the field of pattern matching is to design indexing data structures which take the space very close to that required by the indexed text (in entropy-compressed form) and also simultaneously achieve good query performance. Two popular indexes, namely the FM-index of Ferragina and Manzini and the CSA of Grossi and Vitter, achieve this goal by exploiting the Burrows-Wheeler transform (BWT). However, due to the intricate permutation structure of BWT, no locality of reference can be guaranteed when we perform pattern matching with these indexes. Chien et al. gave an alternative text index which is based on sparsifying the traditional suffix tree and maintaining an auxiliary 2-D range query structure. Given a text T of length n drawn from a σ -sized alphabet set, they achieved an $O(n \log \sigma)$ -bit index for T and showed that this index can preserve locality in pattern matching and hence is amenable to be used in external-memory settings. We improve upon this index and show how to apply entropy compression to reduce index space. Our index takes $O(n(H_k + 1)) + o(n \log \sigma)$ bits of space where H_k is the k th-order empirical entropy of the text. This is achieved by creating variable length blocks of text using arithmetic coding.

3.36 Space-Efficient Framework for Top- k String Retrieval Problems

W.-K. Hon, R. Shah, and J. S. Vitter. “Space-Efficient Framework for Top- k String Retrieval Problems,” *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS '09)*, Atlanta, GA, October 2009.

Full text (Adobe pdf format)

Given a set $\mathcal{D} = \{d_1, d_2, \dots, d_D\}$ of D strings of total length n , our task is to report the “most relevant” strings for a given query pattern P . This involves somewhat more advanced query functionality than the usual pattern matching, as some notion of “most relevant” is involved. In information retrieval literature, this task is best achieved by using inverted indexes. However, inverted indexes work only for some predefined set of patterns. In the pattern matching community, the most popular pattern-matching data structures are suffix trees and suffix arrays. However, a typical suffix tree search involves going through all the occurrences of the pattern over the entire string collection, which might be a lot more than the required relevant documents.

The first formal framework to study such kind of retrieval problems was given by Muthukrishnan. He considered two metrics for relevance: frequency and proximity. He took a threshold-based approach on these metrics and gave data structures taking $O(n \log n)$ words of space. We study this problem in a slightly different framework of reporting the top k most relevant documents (in sorted order) under similar and more general relevance metrics. Our framework gives linear space data structure with optimal query times for arbitrary score functions. As a corollary, it improves the space utilization for the problems considered by Muthukrishnan while maintaining optimal query performance. We also develop compressed variants of these data structures for several specific relevance metrics.

4 LEARNING, PREDICTION, ESTIMATION, CACHING, AND PREFETCHING

4.1 Complexity Results on Learning by Neural Nets

J.-H. Lin and J. S. Vitter. “Complexity Results on Learning by Neural Nets,” *Machine Learning*, **6**, 1991, 211–230. A shortened version appears in *Proceedings of the 2nd Annual ACM Workshop on Computational Learning Theory (COLT '89)*, Santa Cruz, CA, July–August 1989, published by Morgan Kaufmann, San Mateo, CA, 118–133.

Full text but no figures (gzip-compressed postscript)

Full text but no figures (Adobe pdf format)

We consider the computational complexity of learning by neural nets. We are interested in how hard it is to design appropriate neural net architectures and to train neural nets for general and specialized learning tasks. Our main result shows that the training problem for 2-cascade neural nets (which have only two non-input nodes, one of which is hidden) is \mathcal{NP} -complete, which implies that finding an optimal net (in terms of the number of non-input units) that is consistent with a set of examples is also \mathcal{NP} -complete. This result also demonstrates a surprising gap between the computational complexities of one-node (perceptron) and two-node neural net training problems, since the perceptron training problem can be solved in polynomial time by linear programming techniques. We conjecture that training a k -cascade neural net, which is a classical threshold network training problem, is also \mathcal{NP} -complete, for each fixed $k \geq 2$. We also show that the problem of finding an optimal perceptron (in terms of the number of non-zero weights) consistent with a set of training examples is \mathcal{NP} -hard.

Our neural net learning model encapsulates the idea of modular neural nets, which is a popular approach to overcoming the scaling problem in training neural nets. We investigate how much easier the training problem becomes if the class of concepts to be learned is known *a priori* and the net architecture is allowed to be sufficiently non-optimal. Finally, we classify several neural net optimization problems within the polynomial-time hierarchy.

4.2 Learning in Parallel

J. S. Vitter and J.-H. Lin. “Learning in Parallel,” *Information and Computation*, **92**(2), February 1992, 179–202. A shortened version appears in *Proceedings of the 1st Annual ACM Workshop on Computational Learning Theory (COLT '88)*, Cambridge, MA, August 1988, published by Morgan Kaufmann, San Mateo, CA, 106–124.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper, we extend Valiant’s sequential model of concept learning from examples [Valiant 1984] and introduce models for the efficient learning of concept classes from examples *in parallel*. We say that a concept class is \mathcal{NC} -learnable if it can be learned in polylog time with a polynomial number of processors. We show that several concept classes which are polynomial-time learnable are \mathcal{NC} -learnable in constant time. Some other classes can be shown to be \mathcal{NC} -learnable in logarithmic time, but not in constant time. Our main result shows that other classes, such as s -fold unions of geometrical objects in Euclidean space, which are polynomial-time learnable by a greedy set cover technique, are \mathcal{NC} -learnable using a non-greedy technique. We also show that (unless $\mathcal{P} \subseteq \mathcal{RNC}$) several polynomial-time learnable concept classes related to linear programming are not \mathcal{NC} -learnable. Equivalence of various parallel learning models and issues of fault-tolerance are also discussed.

4.3 Optimal Prefetching via Data Compression

J. S. Vitter and P. Krishnan. “Optimal Prefetching via Data Compression,” *Journal of the ACM*, **43**(5) September 1996, 771–793. A shortened version appears in *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science (FOCS '91)*, San Juan, Puerto Rico, October 1991, 121–130.

Full text (Adobe pdf format)

Caching and prefetching are important mechanisms for speeding up access time to data on secondary storage. Recent work in competitive online algorithms has uncovered several promising new algorithms for caching. In this paper, we apply a form of the competitive philosophy for the first time to the problem of prefetching to develop an optimal universal prefetcher in terms of fault ratio, with particular applications to large-scale databases and hypertext systems. Our algorithms for prefetching are novel in that they are based on data compression techniques that are both theoretically optimal and good in practice. Intuitively, in order to compress data effectively, you have to be able to predict future data well, and thus good data compressors should be able to predict well for purposes of prefetching. We show for powerful models such as Markov sources and m th order Markov sources that the page fault rates incurred by our prefetching algorithms are optimal in the limit for almost all sequences of page accesses.

4.4 Practical Prefetching via Data Compression

K. M. Curewitz, P. Krishnan, and J. S. Vitter. “Practical Prefetching via Data Compression,” *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93)*, Washington, D. C, May 1993, 257–266. This work is the basis of a patent by the authors, “Online Background Predictors and Prefetchers,” United States Patent No. 5,485,609, Duke University, January 16, 1996.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

An important issue that affects response time performance in current OODB and hypertext systems is the I/O involved in moving objects from slow memory to cache. A promising way to tackle this problem is to use *prefetching*, in which we predict the user’s next page requests and get those pages into cache in the background. Current databases perform limited prefetching using techniques derived from older virtual memory systems. A novel idea of using data compression techniques for prefetching was recently advocated by Vitter in Krishnan in which the prefetchers based on the Lempel-Ziv data compressor (the UNIX *compress* command) were shown theoretically to be optimal in the limit. In this paper we analyze the *practical* aspects of using data compression techniques for prefetching. We adapt three well-known data compressors to get three simple, deterministic, and universal prefetchers. We simulate our prefetchers on sequences of page accesses derived from the OO1 and OO7 benchmarks and from CAD applications, and demonstrate significant reductions in fault-rate. We examine the important issues of cache replacement, size of the data structure used by the prefetcher, and problems arising from bursts of “fast” page requests (that leave virtually no time between adjacent requests for prefetching and book keeping). We conclude that prediction for prefetching based on data compression techniques holds great promise.

4.5 Optimal Prediction for Prefetching in the Worst Case

P. Krishnan and J. S. Vitter. “Optimal Prediction for Prefetching in the Worst Case,” *SIAM Journal on Computing*, **27**(6), December 1998, 1617–1636. An earlier version appears in *Proceedings*

of the 5th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '94), Alexandria, VA, January 1994, 392–401.

Full text (Adobe pdf format)

Response time delays caused by I/O is a major problem in many systems and database applications. Prefetching and cache-replacement methods are attracting renewed attention because of their success in avoiding costly I/Os. Prefetching can be looked upon as a type of online sequential prediction, where the predictions must be accurate as well as made in a computationally efficient way. Unlike other online problems, prefetching cannot admit a competitive analysis, since the optimal offline prefetcher incurs no cost when it knows the future page requests. Previous analytical work on prefetching by Vitter and Krishnan consisted of modeling the user as a probabilistic Markov source.

In this paper, we look at the much stronger form of worst-case analysis and derive a randomized algorithm that we prove analytically *converges almost surely to the optimal fault rate in the worst case for every sequence of page request* with respect to the important class of finite state prefetchers. In particular, we make no assumption about how the sequence of page requests is generated. This analysis model can be looked upon as a generalization of the competitive framework, in that it compares an online algorithm in a worst-case manner over all sequences against a powerful yet non-clairvoyant opponent. We simultaneously achieve the computational goal of implementing our prefetcher in optimal constant expected time per prefetched page, using the optimal dynamic discrete random variate generator of Matias, Vitter, and Ni.

4.6 Using Vapnik-Chervonenkis Dimension to Analyze the Testing Complexity of Program Segments

K. Romanik and J.S. Vitter. “Using Vapnik-Chervonenkis Dimension to Analyze the Testing Complexity of Program Segments,” *Information and Computation*, **128**(2), August 1, 1996, 87–108. A shortened version appears in “Using Computational Learning Theory to Analyze the Testing Complexity of Program Segments,” *Proceedings of the 17th Annual IEEE International Computer Software and Applications Conference (COMpostscriptAC '93)*, Phoenix, 1993, 367–373.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We examine the complexity of testing different program constructs. We do this by defining a measure of testing complexity known as VCP-dimension, which is similar to the Vapnik-Chervonenkis dimension, and applying it to classes of programs, where all programs in a class share the same syntactic structure. VCP-dimension gives bounds on the number of test points needed to determine that a program is approximately correct, so by studying it for a class of programs we gain insight into the difficulty of testing the program construct represented by the class. We investigate the VCP-dimension of straight line code, if-then-else statements, and for loops. We also compare the VCP-dimension of nested and sequential if-then-else statements as well as that of two types of for loops with embedded if-then-else statements. Finally, we perform an empirical study to estimate the expected complexity of straight line code.

4.7 A Theory for Memory-Based Learning

J.-H. Lin and J. S. Vitter. “A Theory for Memory-Based Learning,” special issue of *Machine Learning*, **17**(2/3), November/December 1994, 143–167. A shortened version appears in *Proceedings of the 5th Annual ACM Conference on Computational Learning Theory (COLT '92)*, Pittsburgh, PA, July 1992, 103–115.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

A memory-based learning system is an extended memory management system that decomposes the input space either statically or dynamically into subregions for the purpose of storing and retrieving functional information. The main generalization techniques employed by memory-based learning systems are the nearest-neighbor search, space decomposition techniques, and clustering. Research on memory-based learning is still in its early stage. In particular, there are very few rigorous theoretical results regarding memory requirement, sample size, expected performance, and computational complexity. In this paper, we propose a model for memory-based learning and use it to analyze several methods— ϵ -covering, hashing, clustering, tree-structured clustering, and receptive-fields—for learning smooth functions. The sample size and system complexity are derived for each method. Our model is built upon the generalized PAC learning model of Haussler and is closely related to the method of vector quantization in data compression. Our main result is that we can build memory-based learning systems using new clustering algorithms of Lin and Vitter to PAC-learn in polynomial time using only polynomial storage in typical situations.

Keywords: Memory-based learning, PAC learning, clustering, approximation, linear programming, relaxation, covering, hashing.

4.8 Coping with Uncertainty in Map Learning

K. J. Basye, T. L. Dean, and J. S. Vitter. “Coping with Uncertainty in Map Learning,” *Machine Learning*, **29**(1), 1997, 65–88. A shortened version appears in *Proceedings of the 11th Joint Conference on Artificial Intelligence (IJCAI '89)*, Detroit, MI, August 1989, 663–668, and in *Autonomous Mobile Robots: Perception, Mapping, and Navigation*, (edited by S. S. Iyengar and Alberto Elfes), IEEE Computer Society Press.

Full text (Adobe pdf format)

In many applications in mobile robotics, it is important for a robot to explore its environment in order to construct a representation of space useful for guiding movement. We refer to such a representation as a map, and the process of constructing a map from a set of measurements as map learning. In this paper, we develop a framework for describing map-learning problems in which the measurements taken by the robot are subject to known errors. We investigate approaches to learning maps under such conditions based on Valiant’s probably approximately correct learning model. We focus on the problem of coping with accumulated error in combining local measurements to make global inferences. In one approach, the effects of accumulated error are eliminated by the use of local sensing methods that never mislead but occasionally fail to produce an answer. In another approach, the effects of accumulated error are reduced to acceptable levels by repeated exploration of the area to be learned. We also suggest some insights into why certain existing techniques for map learning perform as well as they do. The learning problems explored in this paper are quite different from most of the classification and boolean-function learning problems appearing in the literature. The methods described, while specific to map learning, suggest directions to take in tackling other learning problems.

4.9 Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments

P. Krishnan, P. M. Long, and J. S. Vitter. “Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments,” *Algorithmica*, **23**(1), January 1999, 31–56. An extended abstract also appears in *Machine Learning: Proceedings of the Twelfth International Conference*, Armand

Prieditis and Stuart Russell, eds., Morgan Kaufmann Publishers, San Francisco, CA, 1995, under the title “Learning to Make Rent-to-Buy Decisions with Systems Applications.”

Full text (Adobe pdf format)

In the single rent-to-buy decision problem, without a priori knowledge of the amount of time a resource will be used we need to decide when to buy the resource, given that we can rent the resource for \$1 per unit time or buy it once and for all for \$ c . In this paper we study algorithms that make a sequence of single rent-to-buy decisions, using the assumption that the resource use times are independently drawn from an unknown probability distribution. Our study of this rent-to-buy problem is motivated by important systems applications, specifically, problems arising from deciding when to spindown disks to conserve energy in mobile computers [DKM, LKH, MDK], thread blocking decisions during lock acquisition in multiprocessor applications [KLM], and virtual circuit holding times in IP-over-ATM networks [KLP, SaK].

We develop a provably optimal and computationally efficient algorithm for the rent-to-buy problem and evaluate its practical merit for the disk spindown scenario via simulation studies. Our algorithm uses $O(\sqrt{t})$ time and space, and its expected cost for the t th resource use converges to optimal as $O(\sqrt{\log t/t})$, for any bounded probability distribution on the resource use times. Alternatively, using $O(1)$ time and space, the algorithm almost converges to optimal.

We describe the results of simulating our algorithm for the disk spindown problem using disk access traces obtained from an HP workstation environment. We introduce the natural notion of *effective cost* which merges the effects of energy conservation and response time performance into one metric based on a user specified parameter a , the relative importance of response time to energy conservation. (The buy cost c varies linearly with a .) We observe that by varying a , we can model the tradeoff between power and response time well. We also show that our algorithm is best in terms of effective cost for almost all values of a , saving effective cost by 6–25% over the optimal online algorithm in the competitive model (i.e., the 2-competitive algorithm that spins down the disk after waiting c seconds). In addition, for small values of a (corresponding to when saving energy is critical), our algorithm when compared against the optimal online algorithm in the competitive model reduces excess energy by 17–60%, and when compared against the 5 second threshold reduces excess energy by 6–42%.

4.10 Application-Controlled Paging for a Shared Cache

R. D. Barve, E. F. Grove, and J. S. Vitter. “Application-Controlled Paging for a Shared Cache,” *SIAM Journal on Computing*, **29**(4), 2000, 1290–1303. An extended abstract also appears in *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS '95)*, Milwaukee, WI, October 1995.

Full text (Adobe pdf format)

We consider a cache shared by several concurrently running application processes and propose a provably efficient application-controlled global strategy for the shared cache. Using future information implicitly in the form of good decisions by application processes, we are able to break through the H_k lower bound on competitive ratio proved for classical paging for a k -sized cache. For a size- k cache shared by P application processes that always make good cache replacement decisions, we develop an online application-controlled paging algorithm with a competitive ratio of $2H_{P-1} + 2$. Typically, P is much smaller than k , perhaps by several orders of magnitude. Our competitive ratio improves upon the $2P + 2$ competitive ratio achieved by Cao et al. We show for this problem that no online algorithm A can have a competitive ratio better than H_{P-1} even if the application processes aiding A have perfect knowledge of individual request sequences. Our results are with respect to a worst-case interleaving of the individual request sequences of the P

applications.

We introduce a notion of fairness in the more realistic situation when application processes do not always make good cache replacement decisions. We show that our algorithm ensures that no application process needs to evict one of its cached pages to service some page fault caused by a mistake of some other application. Our algorithm is not only fair, but remains efficient; the global paging performance can be bounded in terms of the number of mistakes that application processes make.

4.11 Estimating Alphanumeric Selectivity in the Presence of Wildcards

P. Krishnan, J. S. Vitter, and B. Iyer. “Estimating Alphanumeric Selectivity in the Presence of Wildcards,” *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, Montreal, Canada, May 1996, 282–293.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Success of commercial query optimizers and database management systems (object-oriented or relational) depend on accurate cost estimation of various query reorderings [BGI]. Estimating predicate selectivity, or the fraction of rows in a database that satisfy a selection predicate, is key to determining the optimal join order. Previous work has concentrated on estimating selectivity for numeric fields [ASW, HaSa, IoP, LNS, SAC, WVT]. With the popularity of textual data being stored in databases, it has become important to estimate selectivity accurately for alphanumeric fields. A particularly problematic predicate used against alphanumeric fields is the SQL LIKE predicate [Dat]. Techniques used for estimating numeric selectivity are not suited for estimating alphanumeric selectivity.

In this paper, we study for the first time the problem of estimating alphanumeric selectivity in the presence of wildcards. Based on the intuition that the model built by a data compressor on an input text encapsulates information about common substrings in the text, we develop a technique based on the suffix tree data structure to estimate alphanumeric selectivity. In a statistics generation pass over the database, we construct a compact suffix tree-based structure from the columns of the database. We then look at three families of methods that utilize this structure to estimate selectivity during query plan costing, when a query with predicates on alphanumeric attributes contains wildcards in the predicate.

We evaluate our methods empirically in the context of the TPC-D benchmark. We study our methods experimentally against a variety of query patterns and identify five techniques that hold promise.

4.12 Competitive Analysis of Buffer Management Algorithms for Parallel I/O Systems

R. D. Barve, M. Kallahalla, P. J. Varman, and J. S. Vitter. R. D. Barve, M. Kallahalla, P. Varman, and J. S. Vitter. “Competitive Analysis of Buffer Management Algorithms for Parallel I/O Systems,” *Journal of Algorithms*, **36**, August 2000. An earlier version appeared as “Competitive Parallel Disk Prefetching and Buffer Management,” *Proceedings of the Fifth Annual Workshop on I/O in Parallel and Distributed Systems (IOPADS)*, 1998, San Jose, California.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We provide a competitive analysis framework for online prefetching and buffer management algorithms in parallel I/O systems, using a read-once model of block references. This has widespread

applicability to key I/O-bound applications such as external merging and concurrent playback of multiple video streams. Two realistic lookahead models, global lookahead and local lookahead, are defined. Algorithms NOM and GREED based on these two forms of lookahead are analyzed for shared buffer and distributed buffer configurations, both of which occur frequently in existing systems. An important aspect of our work is that we show how to implement both the models of lookahead in practice using the simple techniques of forecasting and flushing.

Given a D -disk parallel I/O system and a globally shared I/O buffer that can hold up to M disk blocks, we derive a lower bound of $\Omega(\sqrt{D})$ on the competitive ratio of *any* deterministic online prefetching algorithm with $O(M)$ lookahead. NOM is shown to match the lower bound using global M -block lookahead. In contrast, using only local lookahead results in an $\Omega(D)$ competitive ratio. When the buffer is distributed into D portions of M/D blocks each, the algorithm GREED based on local lookahead is shown to be optimal, and NOM is within a constant factor of optimal. Thus we provide a theoretical basis for the intuition that global lookahead is more valuable for prefetching in the case of a shared buffer configuration whereas it is enough to provide local lookahead in case of the distributed configuration. Finally, we analyze the performance of these algorithms for reference strings generated by a uniformly-random stochastic process and we show that they achieve the minimal expected number of I/Os. These results also give bounds on the worst-case expected performance of algorithms which employ randomization in the data layout.

4.13 Wavelet-Based Histograms for Selectivity Estimation

Y. Matias, J. S. Vitter, and M. Wang. “Wavelet-Based Histograms for Selectivity Estimation,” submitted to journal. An earlier version appears in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data (SIGMOD’98)*, Seattle, Washington, June 1998.

Full text (Adobe pdf format)

Query optimization is an integral part of relational database management systems. One important task in query optimization is selectivity estimation, that is, given a query P , we need to estimate the fraction of records in the database that satisfy P . Many commercial database systems maintain histograms to approximate the frequency distribution of values in the attributes of relations.

In this paper, we present a technique based upon a multiresolution wavelet decomposition for building histograms on the underlying data distributions, with applications to databases, statistics, and simulation. Histograms built on the cumulative data distributions give very good approximations with limited space usage. We give fast algorithms for constructing histograms and using them in an on-line fashion for selectivity estimation. Our histograms also provide quick approximate answers to OLAP queries when the exact answers are not required. Our method captures the joint distribution of multiple attributes effectively, even when the attributes are correlated. Experiments confirm that our histograms offer substantial improvements in accuracy over random sampling and other previous approaches.

4.14 Scalable Mining for Classification Rules in Relational Databases

s M. Wang, B. Iyer, and J. S. Vitter. “Scalable Mining for Classification Rules in Relational Databases,” *Herman Rubin Festschrift*, Lecture Notes Monograph Series, **45**, Institute of Mathematical Statistics, Hayward, CA, Fall 2004.

Full text (Adobe pdf format)

Classification is a key function of many “business intelligence” toolkits and a fundamental building block in data mining. Immense data may be needed to train a classifier for good accuracy.

The state-of-art classifiers need an in-memory data structure of size $O(N)$, where N is the size of the training data, to achieve efficiency. For large data sets, such a data structure will not fit in the internal memory. The best previously known classifier does a quadratic number of I/Os for large N .

In this paper, we propose a novel classification algorithm (classifier) called MIND (MINing in Databases). MIND can be phrased in such a way that its implementation is very easy using the extended relational calculus SQL, and this in turn allows the classifier to be built into a relational database system directly. MIND is truly scalable with respect to I/O efficiency, which is important since scalability is a key requirement for any data mining algorithm.

We built a prototype of MIND in the relational database manager DB2 and benchmarked its performance. We describe the working prototype and report the measured performance with respect to the previous method of choice. MIND scales not only with the size of the datasets but also with the number of processors on an IBM SP2 computer system. Even on uniprocessors, MIND scales well beyond the dataset sizes previously published for classifiers. We also give some insights that may have an impact on the evolution of the extended relational calculus SQL.

4.15 Data Cube Approximation and Histograms via Wavelets

J. S. Vitter, M. Wang, and B. Iyer. "Data Cube Approximation and Histograms via Wavelets," *Proceedings of Seventh International Conference on Information and Knowledge Management (CIKM'98)*, Washington D.C., November 1998.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

There has recently been an explosion of interest in the analysis of data in data warehouses in the field of On-Line Analytical Processing (OLAP). Data warehouses can be extremely large, yet obtaining quick answers to queries is important. In many situations, obtaining the exact answer to an OLAP query is prohibitively expensive in terms of time and/or storage space. It can be advantageous to have fast, approximate answers to queries.

In this paper, we present an I/O-efficient technique based upon a multiresolution wavelet decomposition that yields an approximate and space-efficient representation of the data cube, which is one of the core OLAP operators. We build our compact data cube on the logarithms of the partial sums of the raw data values of a multidimensional array. We get excellent approximations for on-line range-sum queries with limited space usage and computational cost. Multiple data cubes can be handled simultaneously. Each query can generally be answered, depending upon the accuracy supported, in one I/O or a small number of I/Os. Experiments show that our method performs significantly better than other approximation techniques such as histograms and random sampling.

4.16 Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets

J. S. Vitter and M. Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets," *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data (SIGMOD'99)*, Philadelphia, PA, June 1999.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Computing multidimensional aggregates in high dimensions is a performance bottleneck for many OLAP applications. Obtaining the exact answer to an aggregation query can be prohibitively

expensive in terms of time and/or storage space in a data warehouse environment. It is advantageous to have fast, approximate answers to OLAP aggregation queries.

In this paper, we present a novel method that provides approximate answers to high-dimensional OLAP aggregation queries in massive sparse data sets in a time-efficient and space-efficient manner. We construct a *compact data cube*, which is an approximate and space-efficient representation of the underlying multidimensional array, based upon a multiresolution wavelet decomposition. In the on-line phase, each aggregation query can generally be answered using the compact data cube in one I/O or a small number of I/Os, depending upon the desired accuracy.

We present two I/O-efficient algorithms to construct the compact data cube for the important case of *sparse high-dimensional arrays*, which often arise in practice. The traditional histogram methods are infeasible for the massive high-dimensional data sets in OLAP applications. Previously developed wavelet techniques are efficient only for dense data. Our on-line query processing algorithm is very fast and capable of refining answers as the user demands more accuracy. Experiments on real data show that our method provides significantly more accurate results for typical OLAP aggregation queries than other efficient approximation techniques such as random sampling.

4.17 Dynamic Maintenance of Wavelet-Based Histograms

Y. Matias, J. S. Vitter, and M. Wang. “Dynamic Maintenance of Wavelet-Based Histograms,” *Proceedings of the 26th International Conference on Very Large Databases (VLDB '00)*, Cairo, Egypt, September 2000.

Full text (Adobe pdf format)

In this paper, we introduce an efficient method for the dynamic maintenance of wavelet-based histograms (and other transform-based histograms). Previous work has shown that wavelet-based histograms provide more accurate selectivity estimation than traditional histograms, such as equi-depth histograms. But since wavelet-based histograms are built by a nontrivial mathematical procedure, namely, wavelet transform decomposition, it is hard to maintain the accuracy of the histogram when the underlying data distribution changes over time. In particular, simple techniques, such as split and merge, which works well for equi-depth histograms, and updating a fixed set of wavelet coefficients, are not suitable here.

We propose a novel approach based upon probabilistic counting and sampling to maintain wavelet-based histograms with very little online time and space costs. The accuracy of our method is robust to changing data distributions, and we get a considerable improvement over previous methods for updating transform-based histograms. A very nice feature of our method is that it can be extended naturally to maintain multidimensional wavelet-based histograms, while traditional multidimensional histograms can be less accurate and prohibitively expensive to build and maintain.

4.18 XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation

See paper 2.49.

4.19 Lexicographically Optimal Smoothing for Broadband Traffic Multiplexing

S. Anastasiadis, P. Varman, J. S. Vitter, and K. Yi. “Optimal Lexicographic Shaping of Aggregate Streaming Data,” *IEEE Transactions on Computers*, **54**(4), 398–408, April 2005. An earlier version appeared in S. Anastasiadis, P. Varman, and J. S. Vitter. “Lexicographically Optimal Smoothing for Broadband Traffic Multiplexing,” *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC '02)*, July 2002, Monterey, CA.

Full text (Adobe pdf format)

We investigate the problem of smoothing multiplexed network traffic, when either a streaming server *transmits data* to multiple clients, or a server *accesses data* from multiple storage devices or other servers. We introduce efficient algorithms for lexicographically optimally smoothing the aggregate bandwidth requirements over a shared network link. In the data transmission problem, we consider the case in which the clients have different buffer capacities but no bandwidth constraints, or no buffer capacities but different bandwidth constraints. For the data access problem, we handle the general case of a shared buffer capacity and individual network bandwidth constraints. Previous approaches in the literature for the data access problem handled either the case of only a single stream or did not compute the lexicographically optimal schedule.

Lexicographically optimal smoothing (*lexopt* smoothing) has several advantages. By provably minimizing the variance of the required aggregate bandwidth, maximum resource requirements within the network become more predictable, and useful resource utilization increases. Fairness in sharing a network link by multiple users can be improved, and new requests from future clients are more likely to be successfully admitted without the need for frequently rescheduling previously accepted traffic. Efficient resource management at the network edges can better meet quality of service requirements without restricting the scalability of the system.

4.20 SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads

See paper 2.52.

4.21 Online Algorithms for Prefetching and Caching in Parallel Disks

See paper 2.60.

4.22 CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation

See paper 2.57.

5 SAMPLING, HISTOGRAMS, AND RANDOM VARIATE GENERATION

5.1 Faster Methods for Random Sampling

J. S. Vitter. “Faster Methods for Random Sampling,” *Communications of the ACM*, **27**(7), July 1984, 703–718.

Several new methods are presented for selecting n records at random without replacement from a file containing N records. Each algorithm selects the records for the sample in a sequential manner—in the same order the records appear in the file. The algorithms are online in that the records for the sample are selected iteratively with no preprocessing. The algorithms require a constant amount of space and are short and easy to implement. The main result of this paper is the design and analysis of Algorithm D, which does the sampling in $O(n)$ time, on the average; roughly n uniform random variates are generated, and approximately n exponentiation operations (of the form a^b , for real numbers a and b) are performed during the sampling. This solves an open

problem in the literature. CPU timings on a large mainframe computer indicate that Algorithm D is significantly faster than the sampling algorithms in use today.

For an improved and optimized version of the random sampling method, see paper 5.3. For reservoir methods, where n is not known in advance, see paper 5.2.

Full text (Adobe pdf format)

5.2 Random Sampling with a Reservoir

J. S. Vitter. “Random Sampling with a Reservoir,” *ACM Transactions on Mathematical Software*, **11**(1), March 1985, 37–57.

We introduce fast algorithms for selecting a random sample of n records without replacement from a pool of N records, where the value of N is unknown beforehand. The main result of the paper is the design and analysis of Algorithm Z; it does the sampling in one pass using constant space and in $O(n(1 + \log(N/n)))$ expected time, which is optimum, up to a constant factor. Several optimizations are studied that collectively improve the speed of the naive version of the algorithm by an order of magnitude. We give an efficient Pascal-like implementation that incorporates these modifications and that is suitable for general use. Theoretical and empirical results indicate that Algorithm Z outperforms current methods by a significant margin.

For sampling methods where n is known in advance, see paper 5.3.

Full text (Adobe pdf format)

5.3 An Efficient Algorithm for Sequential Random Sampling

J. S. Vitter. “An Efficient Algorithm for Sequential Random Sampling,” *ACM Transactions on Mathematical Software*, **13**(1), March 1987, 58–67.

This paper presents an improved and optimized version of the random sampling method from J. S. Vitter, “Faster Methods for Random Sampling,” *Communications of the ACM*, **27**(7), July 1984, 703–718. The object is to choose in sequential online fashion a random sample of size n from a universe of size N . For reservoir methods, where n is not known in advance, see paper 5.2.

Full text (Adobe pdf format)

5.4 Dynamic Generation of Discrete Random Variates

Y. Matias, J. S. Vitter and W.-C. Ni. “Dynamic Generation of Discrete Random Variates,” *Theory of Computing Systems*, **36**(4), 2003, 329–358. A shortened earlier version appears in *Proceedings of the 4th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '93)*, Austin, TX, January 1993, 361–370.

Full text (Adobe pdf format)

We present and analyze efficient new algorithms for generating a random variate distributed according to a dynamically changing set of N weights. The base version of each algorithm generates the discrete random variate in $O(\log^* N)$ expected time and updates a weight in $O(2^{\log^* N})$ expected time in the worst case. We then show how to reduce the update time to $O(\log^* N)$ amortized expected time. We show how to apply our techniques to a recent lookup table technique in order to obtain an expected constant time in the worst case for generation and update. The algorithms are conceptually simple. We give parallel algorithms for parallel generation and update having optimal processors-time product. We also give an efficient dynamic algorithm for maintaining approximate heaps of N elements; each query is required to return an element whose value is within an ϵ factor of the maximal element value. For $\epsilon = 1/\text{polylog}(N)$, each query, insertion, or deletion takes $O(\log \log N)$ time.

Keywords: random number generator, random variate, alias, bucket, rejection, dynamic data structure, update, approximate priority queue.

5.5 Approximate Data Structures with Applications

See paper 8.12.

5.6 Wavelet-Based Histograms for Selectivity Estimation

See paper 4.13.

5.7 Data Cube Approximation and Histograms via Wavelets

See paper 4.15.

5.8 Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets

See paper 4.16.

5.9 Dynamic Maintenance of Wavelet-Based Histograms

See paper 4.17.

5.10 XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation

See paper 2.49.

5.11 SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads

See paper 2.52.

5.12 CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation

See paper 2.57.

6 COMPUTATIONAL GEOMETRY

6.1 Parallel Transitive Closure and Point Location in Planar Structures

See paper 7.3.

6.2 Optimal Cooperative Search in Fractional Cascaded Data Structures

R. Tamassia and J. S. Vitter. “Optimal Cooperative Search in Fractional Cascaded Data Structures,” invited paper in special issue on parallel computing in *Algorithmica* **15**(2), February 1996, 154–171. A shortened version appears in *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '90)*, Crete, Greece, July 1990, 307–316.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Fractional cascading is a technique designed to allow efficient sequential search in a graph with catalogs of total size n . The search consists of locating a key in the catalogs along a path. In this paper we show how to preprocess a variety of fractional cascaded data structures whose underlying graph is a tree so that searching can be done efficiently in parallel. The preprocessing takes $O(\log n)$ time with $n/\log n$ processors on an EREW PRAM. For a balanced binary tree cooperative search along root-to-leaf paths can be done in $O((\log n)/\log p)$ time using p processors on a CREW PRAM. Both of these time/processor constraints are optimal. The searching in the fractional cascaded data structure can be either explicit, in which the search path is specified before the search starts, or implicit, in which the branching is determined at each node. We apply this technique to a variety of geometric problems, including point location, range search, and segment intersection search.

6.3 Approximation Algorithms for Geometric Median Problems

J.-H. Lin and J. S. Vitter. “Approximation Algorithms for Geometric Median Problems,” *Information Processing Letters*, **44**, 1992, 245–249.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we present approximation algorithms for median problems in metric spaces and fixed-dimensional Euclidean space. Our algorithms use a new method for transforming an optimal solution of the linear program relaxation of the s -median problem into a provably good integral solution. This transformation technique is fundamentally different from the methods of randomized and deterministic rounding by Raghavan and the methods proposed the authors’ earlier work in the following way: Previous techniques never set variables with zero values in the fractional solution to the value of 1. This departure from previous methods is crucial for the success of our algorithms.

6.4 A Simplified Technique for Hidden-Line Elimination in Terrains

F. P. Preparata and J. S. Vitter. “A Simplified Technique for Hidden-Line Elimination in Terrains,” *International Journal of Computational Geometry & Applications*, **3**(2), 1993, 167–181. A shortened version appears in *Proceedings of the 1992 Symposium on Theoretical Aspects of Computer Science (STACS '92)*, Paris, France, February 1992, published in *Lecture Notes in Computer Science*, **577**, Springer-Verlag, Berlin, 135–146.

Full text but no figures (gzip-compressed postscript)

Full text but no figures (Adobe pdf format)

In this paper we give a practical and efficient output-sensitive algorithm for constructing the display of a polyhedral terrain. It runs in $O((d+n)\log^2 n)$ time and uses $O(n\alpha(n))$ space, where d is the size of the final display, and $\alpha(n)$ is a (very slowly growing) functional inverse of Ackermann’s function. Our implementation is especially simple and practical, because we try to take full advantage of the specific geometrical properties of the terrain. The asymptotic speed of our algorithm has been improved upon theoretically by other authors, but at the cost of higher space usage and/or high overhead and complicated code. Our main data structure maintains an implicit

representation of the convex hull of a set of points that can be dynamically updated in $O(\log^2 n)$ time. It is especially simple and fast in our application since there are no rebalancing operations required in the tree.

Keywords: display, hidden-line elimination, polyhedral terrain, output-sensitive, convex hull.

6.5 External-Memory Computational Geometry

See paper 2.10.

6.6 Indexing for Data Models with Constraints and Classes

See paper 2.9.

6.7 The Object Complexity Model for Hidden-Surface Elimination

E. F. Grove, T. M. Murali, and J. S. Vitter. “The Object Complexity Model for Hidden-Surface Elimination,” *International Journal of Computational Geometry and Applications*, **9**, 1999, 207–217. A preliminary version appeared in the *Proceedings of the Seventh Canadian Conference on Computational Geometry (CCCG '95)*, Québec City, Québec, Canada, August 1995.

Full text (Adobe pdf format)

We define a new complexity measure, called *object complexity*, for hidden-surface elimination algorithms. This model is more appropriate than the standard scene complexity measure used in computational geometry for predicting the performance of these algorithms on current graphics rendering systems.

We also present an algorithm to determine the set of visible windows in 3-D scenes consisting of n isothetic windows. It takes time $O(n \log n)$, which is optimal. The algorithm solves in the object complexity model the same problem that Bern addressed for the standard scene complexity model.

6.8 Binary Space Partitions for Fat Rectangles

P. K. Agarwal, E. F. Grove, T. M. Murali, and J. S. Vitter. “Binary Space Partitions for Fat Rectangles,” *SIAM Journal on Computing*, **29**(5), 2000, 1422–1448. A shortened version appeared earlier in the *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS'96)*, Burlington, Vermont, October 1996.

Full text (Adobe pdf format)

We consider the practical problem of constructing binary space partitions (BSPs) for a set S of n orthogonal, non-intersecting, two-dimensional rectangles in three-dimensional Euclidean space such that the aspect ratio of each rectangle in S is at most α , for some constant $\alpha \geq 1$. We present an $O(n2^{\sqrt{\log n}})$ -time algorithm to build a binary space partition of size $O(n2^{\sqrt{\log n}})$ for S . We also show that if m of the n rectangles in S have aspect ratios greater than α , we can construct a BSP of size $O(n\sqrt{m}2^{\sqrt{\log n}})$ for S in $O(n\sqrt{m}2^{\sqrt{\log n}})$ time. The constants of proportionality in the big-oh terms are linear in $\log \alpha$. We extend these results to cases in which the input contains non-orthogonal or intersecting objects.

6.9 Efficient 3-D Range Searching in External Memory

See paper 2.15.

6.10 Optimal External Memory Interval Management

See paper 2.16.

6.11 Communication Issues in Large-Scale Geometric Computation

J. S. Vitter. “Communication Issues in Large-Scale Geometric Computation,” *ACM Computing Surveys*, **28**(4es), December 1996.

Full text (html)

Large-scale problems involving geometric data arise in numerous settings, and severe communication bottlenecks can arise in solving them. Work is needed in the development of I/O-efficient algorithms, as well as those that effectively utilize hierarchical memory. In order for new algorithms to be implemented efficiently in practice, the machines they run on must support fundamental external-memory operations. We discuss several advantages offered by TPIE (Transparent Parallel I/O Programming Environment) to enable I/O-efficient implementations.

Categories and Subject Descriptors: D.4.2 [Operating Systems]: Storage Management - Secondary Storage; D.4.4 [Operating Systems]: Communications Management - Input/Output; E.2 [Data Storage Representations]: Contiguous representations; F.2.2 [Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems - Computations on discrete structures, geometrical problems and computations; B.4.4 [Input/Output and Data Communications: Performance Analysis and Design Aids] - Formal models, Worst-case analysis;

General Terms: Algorithms, Design, Languages, Performance, Theory. Additional Key Words and Phrases: computational geometry, I/O, external memory, secondary memory, communication, disk drive, parallel disks.

6.12 Practical Techniques for Constructing Binary Space Partitions for Orthogonal Rectangles

P. K. Agarwal, T. M. Murali, and J. S. Vitter. “Practical Techniques for Constructing Binary Space Partitions for Orthogonal Rectangles,” *Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG’97)*, Nice, France, June 1997, 382–384.

Full text (Adobe pdf format)

We present the first systematic comparison of the performance of algorithms that construct Binary Space Partitions for orthogonal rectangles in three-dimensional Euclidean space. We compare known algorithms with our implementation of a variant of a recent algorithm of Agarwal et al. We show via an empirical study that their algorithm constructs BSPs of near-linear size in practice and performs better than most of the other algorithms in the literature.

6.13 Cylindrical Static and Kinetic Binary Space Partitions

P. K. Agarwal, L. J. Guibas, T. M. Murali, and J. S. Vitter. “Cylindrical Static and Kinetic Binary Space Partitions,” *Proceedings of the 13th Annual Symposium on Computational Geometry (SoCG’97)*, Nice, France, June 1997, 39–48.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We describe the first known algorithm for efficiently maintaining a Binary Space Partition (BSP) for n continuously moving segments in the plane. Under reasonable assumptions on the motion, we show that the total number of times the BSP changes is $O(n^2)$, and that we can update the BSP in $O(\log n)$ expected time per change. We also consider the problem of constructing a BSP for n

triangles in three-dimensional Euclidean space. We present a randomized algorithm that constructs a BSP of expected size $O(n^2)$ in $O(n^2 \log^2 n)$ expected time. We also describe a deterministic algorithm that constructs a BSP of size $O((n+k) \log n)$ and height $O(\log n)$ in $O((n+k) \log^2 n)$ time, where k is the number of intersection points between the edges of the projections of the triangles onto the xy -plane.

6.14 I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking

See paper 2.21.

6.15 Theory and Practice of I/O-Efficient Algorithms for Multidimensional Batched Searching Problems

See paper 2.22.

6.16 Efficient Searching with Linear Constraints

See paper 2.28.

6.17 Scalable Sweep-Based Spatial Join

See paper 2.23.

6.18 Constructing Binary Space Partitions for Orthogonal Rectangles in Practice

T. M. Murali, P. K. Agarwal, and J. S. Vitter. “Constructing Binary Space Partitions for Orthogonal Rectangles in Practice,” *Proceedings of the 6th Annual European Symposium on Algorithms (ESA '98)*, Venice, August 1998, published in Lecture Notes in Computer Science, **1461**, Springer-Verlag, Berlin, 211–222.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper, we develop a simple technique for constructing a Binary Space Partition (BSP) for a set of orthogonal rectangles in three-dimensions. Our algorithm has the novel feature that it tunes its performance to the geometric properties of the rectangles, e.g., their aspect ratios.

We have implemented our algorithm and tested its performance on real data sets. We have also systematically compared the performance of our algorithm with that of other techniques presented in the literature. Our studies show that our algorithm constructs BSPs of near-linear size and small height in practice, has fast running times, and answers queries efficiently. It is a method of choice for constructing BSPs for orthogonal rectangles.

6.19 Efficient Bulk Operations on Dynamic R-trees

See paper 2.31.

6.20 I/O-Efficient Dynamic Point Location in Monotone Subdivisions

See paper 2.32.

6.21 A Parallel Algorithm for Planar Orthogonal Grid Drawings

R. Tamassia, I. G. Tollis, and J. S. Vitter. “A Parallel Algorithm for Planar Orthogonal Grid Drawings,” *Parallel Processing Letters*, March 2000. An extended abstract appears in “Lower Bounds and Parallel Algorithms for Planar Orthogonal Grid Drawings,” *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing (SPDP '91)*, Dallas, TX, December 1991, 386–393.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we consider the problem of constructing *planar orthogonal grid drawings* (or more simply, *layouts*) of graphs, with the goal of minimizing the number of bends along the edges. We present optimal parallel algorithms that construct graph layouts with $O(n)$ maximum edge length, $O(n^2)$ area, and at most $2n + 4$ bends (for biconnected graphs) and $2.4n + 2$ bends (for simply connected graphs). All three of these quality measures for the layouts are optimal in the worst case for biconnected graphs. The algorithm runs on a CREW PRAM in $O(\log n)$ time with $n/\log n$ processors, thus achieving optimal time and processor utilization. Applications include VLSI layout, graph drawing, and wireless communication.

6.22 On Two-Dimensional Indexability and Optimal Range Search Indexing

See paper 2.34.

6.23 Online Data Structures in External Memory

See paper 2.36.

6.24 A Unified Approach for Indexed and Non-Indexed Spatial Joins

See paper 2.38.

6.25 I/O-Efficient Algorithms for Problems on Grid-Based Terrains

See paper 2.39.

6.26 CAMEL: Concept Annotated iMagE Libraries

See paper 2.44.

6.27 External Memory Algorithms and Data Structures: Dealing with Massive Data

See paper 2.30.

6.28 Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data

See paper 2.64.

6.29 Efficient Join Processing over Uncertain-Valued Attributes

See paper 2.65.

6.30 Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing

See paper 3.32.

6.31 Algorithms and Data Structures for External Memory—*main reference!*

See paper 2.71.

7 PARALLEL ALGORITHMS AND SCIENTIFIC COMPUTING

7.1 The Input/Output Complexity of Sorting and Related Problems

See paper 2.1.

7.2 I/O Overhead and Parallel VLSI Architectures for Lattice Computations

See paper 2.2.

7.3 Parallel Transitive Closure and Point Location in Planar Structures

R. Tamassia and J. S. Vitter. “Parallel Transitive Closure and Point Location in Planar Structures,” *SIAM Journal on Computing*, 20(4), August 1991, 708–725. A shortened version appears in “Optimal Parallel Algorithms for Transitive Closure and Point Location in Planar Structures,” *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '89)*, Sante Fe, N. M., June 1989, 399–408. Also appears as an invited paper in *Proceedings of the International Workshop on Discrete Algorithms and Complexity*, Fukuoka, Japan, November 1989, 169–178.

Full text but no figures (gzip-compressed postscript)

Full text but no figures (Adobe pdf format)

Parallel algorithms for several graph and geometric problems are presented, including transitive closure and topological sorting in planar *st*-graphs, preprocessing planar subdivisions for point location queries, and construction of visibility representations and drawings of planar graphs. Most of these algorithms achieve optimal $O(\log n)$ running time using $n/\log n$ processors in the EREW PRAM model, n being the number of vertices.

7.4 Optimal Algorithms for Parallel Memory I: Two-Level Memories

See paper 2.3.

7.5 Optimal Algorithms for Parallel Memory I: Two-Level Memories

See paper 2.4.

7.6 Large-Scale Sorting in Uniform Memory Hierarchies

See paper 2.5.

7.7 Optimal Deterministic Sorting on Parallel Disks

See paper 2.6.

7.8 Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies

See paper 2.7.

7.9 Learning in Parallel

See paper 4.2.

7.10 Parallel Lossless Image Compression Using Huffman and Arithmetic Coding

See paper 3.9.

7.11 Optimal Cooperative Search in Fractional Cascaded Data Structures

See paper 6.2.

7.12 A Divide and Conquer Approach to Shortest Paths in Planar Layered Graphs

S. Subramanian, R. Tamassia, and J. S. Vitter. “An Efficient Parallel Algorithm for Shortest Paths in Planar Layered Digraphs,” *Algorithmica*, **14**, 1995, 322–339. A shortened earlier version appears in “A Divide and Conquer Approach to Shortest Paths in Planar Layered Graphs,” *Proceedings of the Fourth IEEE Symposium on Parallel and Distributed Processing*, Dallas, TX, December 1992, 176–183.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We give efficient parallel algorithms to compute shortest-paths in planar layered digraphs. We show that these digraphs admit special kinds of separators, called *one-way* separators, which allow paths in the graph to cross them only once. We use these separators to give divide-and-conquer solutions to the problem of finding the shortest paths. We first give a simple algorithm that works on the CREW PRAM model and computes the shortest path between any two vertices of an n -node planar layered digraph in time $O(\log^3 n)$ using $n/\log n$ processors. A CRCW version of this algorithm runs in $O(\log^2 n \log \log n)$ time and uses $O(n/\log \log n)$ processors. We then improve the time bound to $O(\log^2 n)$ on the CREW model and $O(\log n \log \log n)$ on the CRCW model. The processor bounds still remain $n/\log n$ for the CREW model and $n/\log \log n$ for the CRCW model.

7.13 TPIE: Transparent Parallel I/O Programming Environment

See paper 2.13.

7.14 I/O-Efficient Scientific Computation using TPIE

See paper 2.14.

7.15 Simple Randomized Mergesort on Parallel Disks

See paper 2.17.

7.16 Modeling and optimizing I/O throughput of multiple disks on a bus

See paper 2.25.

7.17 Online Data Structures in External Memory

See paper 2.36.

7.18 Lower Bounds and Parallel Algorithms for Planar Orthogonal Grid Drawings

See paper 6.21.

7.19 External Memory Algorithms and Data Structures: Dealing with Massive Data

See paper 2.30.

7.20 Distribution Sort with Randomized Cycling

See paper 2.42.

7.21 Duality Between Prefetching and Queued Writing with Parallel Disks

See paper 2.46.

8 COMBINATORIAL ALGORITHMS AND COMBINATORIAL OPTIMIZATION

8.1 Average-Case Analysis of Algorithms and Data Structures

J. S. Vitter and Ph. Flajolet. “Average-Case Analysis of Algorithms and Data Structures,” Chapter 9 in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity* (edited by J. van Leeuwen), Elsevier and M.I.T. Press, 1990, 431–524.

Full text but no figures (Adobe pdf format)

The aim of this chapter to describe the main mathematical methods and applications in the average-case analysis of algorithms and data structures. It comprises two parts: First, we present basic combinatorial enumerations based on symbolic methods and asymptotic methods with emphasis on complex analysis techniques (such as singularity analysis, saddle point, Mellin transforms). Next, we show how to apply these general methods to the analysis of sorting, searching, tree data structures, hashing, and dynamic algorithms. The emphasis is on algorithms for which exact “analytic models” can be derived.

8.2 The Input/Output Complexity of Sorting and Related Problems

See paper 2.1.

8.3 ϵ -Approximations with Small Packing Constraint Violation

J.-H. Lin and J. S. Vitter. “ ϵ -Approximations with Small Packing Constraint Violation,” *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC '92)*, Victoria, Canada, May 1992, 771–782.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present efficient new randomized and deterministic methods for transforming optimal solutions for a type of relaxed integer linear program into provably good solutions for the corresponding \mathcal{NP} -hard discrete optimization problem. Without any constraint violation, the ϵ -approximation problem for many problems of this type is itself \mathcal{NP} -hard. Our methods provide polynomial-time ϵ -approximations while attempting to minimize the packing constraint violation.

Our methods lead to the first known approximation algorithms with provable performance guarantees for the *s-median problem*, the *tree pruning problem*, and the *generalized assignment problem*. These important problems have numerous applications to data compression, vector quantization, memory-based learning, computer graphics, image processing, clustering, regression, network location, scheduling, and communication. We provide evidence via reductions that our approximation algorithms are nearly optimal in terms of the packing constraint violation. We also discuss some recent applications of our techniques to scheduling problems.

8.4 Approximation Algorithms for Geometric Median Problems

See paper 6.3.

8.5 Nearly Optimal Vector Quantization via Linear Programming

See paper 3.10.

8.6 A Theory for Memory-Based Learning

See paper 4.7.

8.7 Optimal Algorithms for Parallel Memory I: Two-Level Memories

See paper 2.3.

8.8 Optimal Algorithms for Parallel Memory I: Two-Level Memories

See paper 2.4.

8.9 Large-Scale Sorting in Uniform Memory Hierarchies

See paper 2.5.

8.10 Optimal Deterministic Sorting on Parallel Disks

See paper 2.6.

8.11 Optimal Deterministic Sorting on Parallel Processors and Parallel Memory Hierarchies

See paper 2.7.

8.12 Approximate Data Structures with Applications

Y. Matias, J. S. Vitter, and N. Young. “Approximate Data Structures with Applications,” *Proceedings of the 5th Annual SIAM/ACM Symposium on Discrete Algorithms (SODA '94)*, Alexandria, VA, January 1994. The work is the basis of a patent by the authors, “Implementation of Approximate Data Structures,” United States Patent No. 5,519,840, Bell Laboratories, May 21, 1996.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In this paper we introduce the notion of *approximate data structures*, in which a small amount of error is tolerated in the output. Approximate data structures trade error of approximation for faster operation, leading to theoretical and practical speedups for a wide variety of algorithms. We give approximate variants of the van Emde Boas data structure, which support the same dynamic operations as the standard van Emde Boas data structure, except that answers to queries are approximate. The variants support all operations in constant time provided the error of approximation is $1/\text{polylog}(n)$, and in $O(\log \log n)$ time provided the error is $1/\text{polynomial}(n)$, for n elements in the data structure.

We consider the tolerance of prototypical algorithms to approximate data structures. We study in particular Prim’s minimum spanning tree algorithm, Dijkstra’s single-source shortest paths algorithm, and an on-line variant of Graham’s convex hull algorithm. To obtain output which approximates the desired output with the error of approximation tending to zero, Prim’s algorithm requires only linear time, Dijkstra’s algorithm requires $O(m \log \log n)$ time, and the on-line variant of Graham’s algorithm requires constant amortized time per operation.

8.13 Blocking for External Graph Searching

See paper 2.8.

8.14 External-Memory Graph Algorithms

See paper 2.11.

8.15 External-Memory Algorithms for Processing Line Segments in Geographic Information Systems

See paper 2.12.

8.16 Simple Randomized Mergesort on Parallel Disks

See paper 2.17.

8.17 On Sorting Strings in External Memory

See paper 2.20.

8.18 Simple Randomized Mergesort on Parallel Disks

See paper 2.35.

8.19 Online Data Structures in External Memory

See paper 2.36.

8.20 External Memory Algorithms with Dynamically Changing Memory Allocations

See paper 2.37.

8.21 Efficient Bundle Sorting

See paper 2.40.

8.22 External Memory Algorithms and Data Structures: Dealing with Massive Data

See paper 2.30.

8.23 Distribution Sort with Randomized Cycling

See paper 2.42.

8.24 Efficient Sorting using Registers and Caches

L. Arge, J. S. Chase, and J. S. Vitter. “Efficient Sorting using Registers and Caches,” invited paper in special issue of *ACM Journal of Experimental Algorithmics*, **7**(9), 2002. An earlier and shorter version appears in L. Arge, R. Barve, J. S. Chase, J. S. Vitter, and R. Wickremesinghe. “Efficient Sorting using Registers and Caches,” *Proceedings of the 4rd Workshop on Algorithm Engineering (WAE '00)*, Saarbrücken, Germany, September 2000, published in *Lecture Notes in Computer Science*, **1982** Springer-Verlag, Berlin, Germany, 51–62.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

Modern computer systems have increasingly complex memory systems. Common machine models for algorithm analysis do not reflect many of the features of these systems, e.g., large register sets, lockup-free caches, cache hierarchies, associativity, cache line fetching, and streaming behavior. Inadequate models lead to poor algorithmic choices and an incomplete understanding of algorithm behavior on real machines.

A key step toward developing better models is to quantify the performance effects of features not reflected in the models. This paper explores the effect of memory system features on sorting performance. We introduce a new cache-conscious sorting algorithm, R-MERGE, which achieves better performance in practice over algorithms that are superior in the theoretical models. R-MERGE is designed to minimize memory stall cycles rather than cache misses by considering features common to many system designs.

8.25 Duality Between Prefetching and Queued Writing with Parallel Disks

See paper 2.46.

8.26 Constrained Querying of Multimedia Databases: Issues and Approaches

See paper 2.43.

8.27 Supporting Incremental Join Queries on Ranked Inputs

See paper 2.47.

8.28 Aggregate Predicate support in DBMS

See paper 2.48.

8.29 Online Algorithms for Prefetching and Caching in Parallel Disks

See paper 2.60.

8.30 Bulk Operations for Space-Partitioning Trees

See paper 2.61.

8.31 Mining Deviants in Time Series Data Streams

See paper 2.62.

8.32 Rank-aware Query Optimization

See paper 2.63.

8.33 Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data

See paper 2.64.

8.34 Efficient Join Processing over Uncertain-Valued Attributes

See paper 2.65.

8.35 A Framework for Dynamizing Succinct Data Structures

See paper 3.31.

8.36 Geometric Burrows-Wheeler Transform: Linking Range Searching and Text Indexing

See paper 3.32.

8.37 SBC-tree: Efficient Indexing for RLE-Compressed Strings

See paper 2.67.

8.38 Algorithms and Data Structures for External Memory—*main reference!*

See paper 2.71.

8.39 On Searching Compressed String Collections Cache-Obliviously

See paper 2.72.

8.40 On Entropy-Compressed Text Indexing in External Memory

See paper 3.35.

9 ONLINE ALGORITHMS AND DYNAMIC DATA STRUCTURES

9.1 A Complexity Theoretic Approach to Incremental Computation

P. B. Miltersen, S. Subramanian, J. S. Vitter, and R. Tamassia “Complexity Models for Incremental Computation,” special issue of *Theoretical Computer Science*, **130**, 1994, 203–236. A shortened version appears in *Proceedings of the 1993 Symposium on Theoretical Aspects of Computer Science (STACS '93)*, Wurzburg, Germany, February 1993, published in Lecture Notes in Computer Science, Springer-Verlag, Berlin, 640–649.

Full text (Adobe pdf format)

We present a new complexity theoretic approach to incremental computation. We define complexity classes that capture the intuitive notion of incremental efficiency and study their relation to existing complexity classes. We show that problems that have small sequential space complexity also have small incremental time complexity.

We show that all common LOGSPACE-complete problems for P are also incr-POLYLOGTIME-complete for P. We introduce a restricted notion of completeness called NRP-completeness and show that problems which are NRP-complete for P are also incr-POLYLOGTIME-complete for P. We also give incrementally complete problems for NLOGSPACE, LOGSPACE, and non-uniform NC1. We show that under certain restrictions problems which have efficient dynamic solutions also have efficient parallel solutions. We also consider a non-uniform model of incremental computation and show that in this model most problems have almost linear complexity. In addition, we present some techniques for lower bounding the complexity of explicitly defined problems.

We also look at the time complexity of circuit value and network stability problems restricted to comparator gates. We show that the comparator-circuit value problem and the “Lex-First Maximal Matching” problem are in incr-LOGSPACE while the comparator-network stability and the “Man-Optimal Stable Marriage Problem” are in rincr-LOGSPACE. This shows that the dynamic versions of these problems are solvable quickly in parallel even though there are no known NC algorithms to solve them from scratch.

9.2 Optimal Prefetching via Data Compression

See paper 4.3.

9.3 Dynamic Generation of Discrete Random Variates

See paper 5.4.

9.4 Approximate Data Structures with Applications

See paper 8.12.

9.5 Online Perfect Matching and Mobile Computing

E. F. Grove, M.-Y. Kao, P. Krishnan, and J. S. Vitter. “Online Perfect Matching and Mobile Computing,” *Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS '95)*, Kingston, Ontario, August 1995.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

We present a natural online perfect matching problem motivated by problems in mobile computing. A total of n customers connect and disconnect sequentially, and each customer has an associated set of stations to which it may connect. Each station has a capacity limit. We allow the network to preemptively switch a customer between allowed stations to make room for a new arrival. We wish to minimize the total number of switches required to provide service to every customer. Equivalently, we wish to maintain a perfect matching between customers and stations and minimize the lengths of the augmenting paths. We measure performance by the worst case ratio of the number of switches made to the minimum number required. When each customer can be connected to at most two stations:

- Some intuitive algorithms have lower bounds of $\Omega(n)$ and $\Omega(n/\log n)$.
- When the station capacities are 1, there is an upper bound of $O(\sqrt{n})$.
- When customers do not disconnect and the station capacity is 1, we achieve a competitive ratio of $O(\log n)$.
- There is a lower bound of $\Omega(\sqrt{n})$ when the station capacities are 2.
- We present optimal algorithms when the station capacity is arbitrary in special cases.

9.6 Load Balancing in the L_p Norm

A. Awerbuch, A. Azar, E. F. Grove, P. Krishnan, M. Y. Kao, and J. S. Vitter. “Load Balancing in the L_p Norm,” *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS '95)*, Milwaukee, WI, October 1995.

Full text (gzip-compressed postscript)

Full text (Adobe pdf format)

In the load balancing problem, there is a set of servers, and jobs arrive sequentially. Each job can be run on some subset of the servers, and must be assigned to one of them in an online fashion. Traditionally, the assignment of jobs to servers is measured by the L_∞ norm; in other words, an assignment of jobs to servers is quantified by the maximum load assigned to any server. In this measure the performance of the greedy load balancing algorithm may be a logarithmic factor higher than the offline optimal. In many applications, the L_∞ norm is not a suitable way to measure how well the jobs are balanced. If each job sees a delay that is proportional to the number of jobs on its server, then the *average* delay among all jobs is proportional to the sum of the squares of the numbers of jobs assigned to the servers. Minimizing the average delay is equivalent to minimizing the Euclidean (or L_2) norm. For any fixed p , $1 \leq p < \infty$, we show that the greedy algorithm performs within a constant factor of the offline optimal with respect to the L_p norm. The constant

grows linearly with p , which is best possible, but does not depend on the number of servers and jobs.

9.7 Efficient 3-D Range Searching in External Memory

See paper 2.15.

9.8 Optimal External Memory Interval Management

See paper 2.16.

9.9 Competitive Parallel Disk Prefetching and Buffer Management

See paper 4.12.

9.10 Adaptive Disk Spindown via Optimal Rent-to-Buy in Probabilistic Environments

See paper 4.9.

9.11 A Competitive Application-Controlled Paging Algorithm for a Shared Cache

See paper 4.10.

9.12 Cylindrical Static and Kinetic Binary Space Partitions

See paper 6.13.

9.13 I/O-Efficient Algorithms for Contour-line Extraction and Planar Graph Blocking

See paper 2.21.

9.14 Efficient Searching with Linear Constraints

See paper 2.28.

9.15 Efficient Bulk Operations on Dynamic R-trees

See paper 2.31.

9.16 I/O-Efficient Dynamic Point Location in Monotone Subdivisions

See paper 2.32.

9.17 On Two-Dimensional Indexability and Optimal Range Search Indexing

See paper 2.34.

9.18 Online Data Structures in External Memory

See paper 2.36.

9.19 External Memory Algorithms with Dynamically Changing Memory Allocations

See paper 2.37.

9.20 Dynamic Maintenance of Wavelet-Based Histograms

See paper 4.17.

9.21 External Memory Algorithms and Data Structures: Dealing with Massive Data

See paper 2.30.

9.22 Compressed Suffix Arrays and Suffix Trees, with Applications to Text Indexing and String Matching

See paper 3.22.

9.23 Constrained Querying of Multimedia Databases: Issues and Approaches

See paper 2.43.

9.24 CAMEL: Concept Annotated iMagE Libraries

See paper 2.44.

9.25 XPathLearner: An On-Line Self-Tuning Markov Histogram for XML Path Selectivity Estimation

See paper 2.49.

9.26 Efficient Update of Indexes for Dynamically Changing Web Documents

See paper 2.51.

9.27 SASH: A Self-Adaptive Histogram Set for Dynamically Changing Workloads

See paper 2.52.

9.28 CXHist: An On-line Classification-based Histogram for XML String Selectivity Estimation

See paper 2.57.

9.29 Online Algorithms for Prefetching and Caching in Parallel Disks

See paper 2.60.

9.30 Mining Deviants in Time Series Data Streams

See paper 2.62.

9.31 Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data

See paper 2.64.

9.32 Efficient Join Processing over Uncertain-Valued Attributes

See paper 2.65.

9.33 SBC-tree: Efficient Indexing for RLE-Compressed Strings

See paper 2.67.

9.34 Dynamic Rank/Select Dictionaries with Applications to XML Indexing

See paper 2.68.

9.35 A Framework for Dynamizing Succinct Data Structures

See paper 3.31.

9.36 Algorithms and Data Structures for External Memory—*main reference!*

See paper 2.71.