Report from Dagstuhl Seminar 21071

Scalable Data Structures

Edited by

Gerth Stølting Brodal[1], John Iacono[2], Markus E. Nebel[3], and Vijaya Ramachandran[4]

1 **Aarhus University, DK,** gerth@cs.au.dk
2 **UL – Brussels, BE,** jiacono@ulb.ac.be
3 **Universität Bielefeld, DE,** nebel@techfak.uni-bielefeld.de
4 **University of Texas – Austin, US,** vlr@cs.utexas.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 21071 "Scalable Data Structure". Even if the field of data structures is quite mature, new trends and limitations in computer hardware together with the ever-increasing amounts of data that need to be processed raise new questions with respect to efficiency and continuously challenge the existing models of computation. Thermal and electrical power constraints have caused technology to reach "the power wall" with stagnating single processor performance, meaning that all nontrivial applications need to address scalability with multiple processors, a memory hierarchy and other communication challenges. Scalable data structures are pivotal to this process since they form the backbone of the algorithms driving these applications. The extended abstracts included in this report contain both recent state of the art advances and lay the foundation for new directions within data structures research.

Seminar February 14–19, 2021 – http://www.dagstuhl.de/21071
2012 ACM Subject Classification Theory of computation → Data structures design and analysis; Theory of computation → Design and analysis of algorithms
Keywords and phrases algorithms, big data, data structures, GPU computing, large data sets, models of computation, parallel algorithms
Digital Object Identifier 10.4230/DagRep.11.1.1
Edited in cooperation with Jesper Steensgaard

1 Executive summary

Gerth Stølting Brodal (Aarhus University, DK)
John Iacono (UL – Brussels, BE)
Markus E. Nebel (Universität Bielefeld, DE)
Vijaya Ramachandran (University of Texas – Austin, US)

About the seminar

Scalable data structures form the backbone for computing: Computing is about processing, exchanging, and storing data. The organization of data profoundly influences the performance of accessing and manipulating data. By optimizing the way data is stored, performance can be improved by several orders of magnitude when data scales. This Dagstuhl seminar brought together researchers from several research directions to illuminate solutions to the

scalability challenge of data structures. The seminar was the 14th in a series of loosely related Dagstuhl seminars on data structures. Due to the ongoing Covid-19 pandemic the seminar was purely virtual.

Topics

The presentations covered both advances in classic data structure fields, as well as insights addressing the scalability of computing for different models of computation.

Classic data structure questions on dictionaries, hashing, filters, and heaps were the topic of several talks. Wild (Section 4.8) considered new pointer based search trees, Kozma (Section 4.24) discussed algorithms related to self-adjusting trees and heaps, Bercea (Section 4.4) presented results for dictionaries and filters, Even (Section 4.21) discussed dynamic stable perfect hashing, and Farach-Colton (Section 4.2) presented results for succinct stable hash tables. Johnson (Section 4.25) presented the vector quotient filter based on Robin Hood hashing and designed to exploit SIMD instructions. An external memory dictionary was presented by Conway (Section 4.20) who presented the SplinterDB key-value store for NVMe solid state drives.

For data structure problems on strings, Gørtz (Section 4.9) discussed the support for random access in compact representations of strings, and Starikovskaya (Section 4.22) dictionary look-ups with mismatches.

Data structures for storing and querying static and dynamic graphs were the topic of a sequence of talks. Pettie (Section 4.1) considered trade-offs between space usage and query time for supporting exact distance queries in planar graphs. Rotenberg (Section 4.5) considered planarity testing of dynamic graphs under the insertion and deletion of edges with polylogarithmic update time. Kopelowitz (Section 4.6) considered maintaining the orientation of edges in dynamic forests under the insertion of edges, guaranteeing low out degree of all nodes. Henzinger (Section 4.16) presented an algorithm for maintaining a $(\Delta + 1)$-vertex coloring of a graph with maximal degree Δ with constant time edge insertions and deletions. Bast (Section 4.27) gave a demonstration of an implementation of algorithms for real-time searching knowledge graphs with billions of edges.

Parallel algorithms for problems on graphs were addressed in multiple talks. Liu (Section 4.10) considered a parallel algorithm for counting triangles (cliques of size three) in graphs under batched updates of edge insertions and deletions, and Blelloch (Section 4.15) considered parallel batched dynamic algorithms for the minimum spanning tree and minimum cut problems. Sun (Section 4.13) considered a parallel algorithm for the single source shortest path problem using lazy batched priority queues. Shun (Section 4.3) considered a parallel index-based algorithm for graph clustering and an approximation algorithm using locality-sensitive hashing.

Computational models supporting massive parallelism, like GPUs and TCUs, were addressed in talks by Owens (Section 4.11) who considered open-addressing hashing on GPUs, by Geil (Section 4.18) who consider how to solve the maximum clique problem on GPUs, and by Silvestri (Section 4.14) who addressed similarity search with tensor core units. Sitchinava & Jacob (Section 4.19) in their joint talk, considered the power of the atomic and non-atomic versions of the parallel fork-join model. Sanders (Section 4.12) considered how to execute MapReduce computations robustly and efficiently on realistic distributed-memory parallel machines.

Ellen (Section 4.17) considered labelling schemes for networks supporting distributed deterministic radio broadcast using labels of constant-length at the nodes of the network. Lincoln (Section 4.23) presented new techniques for proving fine-grained average-case hardness results, and Fagerberg (Section 4.26) considered the fragile complexity of adaptive algorithms. Finally, Driemel (Section 4.7) considered approximate-near-neighbor data structures for time series under the continuous Fréchet distance.

Final Thoughts

The organizers would like to thank the Dagstuhl team for their continuous support and allowing this seminar to happen as a purely virtual Dagstuhl seminar. They also thank all participants for their contributions to this seminar.

Previous seminars in the series had few female participants. A focus for this seminar was to significantly increase the female attendance. 50 % of the invited participants were female, resulting in a 38 % female attendance.

Even though the seminar was challenged by the different time zones of the participants, on average 37 of the 48 participants participants attended the talks, and all talks were attended by at least 30 partcipants. In the post-seminar survey it was appreciated that the seminar took place as a virtual seminar instead of being cancelled, but it was also stated that the virtual format can never be as productive as an in-person seminar and showed how much we should appreciate the possibilities Dagstuhl offers under regular circumstances.

2 Table of Contents

3 Seminar program

Since the seminar was converted into a purely virtual seminar, the program had to accommodate participants from many time zones – primarily from Europe and US. The program consisted of two blocks 16:00–18:00 and 20:00–22:00 Dagstuhl time (CET) from Monday to Thursday, i.e. during 17:00–23:00 in Israel, 10:00–16:00 on the US East Coast, 7:00–13:00 on the US West Coast, and 5:00–11:00 in Hawaii. Most talks were 30 minutes. All talks took place using the Zoom platform. The Wonder.me platform was used for social interaction, to replace the otherwise informal interactions happening during coffee breaks, meals, evenings etc. when at a physical Dagstuhl seminar. Participants were encouraged to visit Wonder.me 15 minutes before session start and after talks ended.

Monday February 15, 2021

16:00 *One slide presentations & Group picture*

17:00 *Planar Distance Oracles with Better Time-Space Tradeoffs*
Seth Pettie

17:30 *Succinct, Stable Hash Tables*
Martin Farach-Colton

20:00 *Parallel Index-Based Structural Graph Clustering and Its Approximation*
Julian Shun

20:30 *Dictionaries et al.*
Ioana Bercea

21:00 *Fully-dynamic Planarity Testing in Polylogarithmic Time*
Eva Rotenberg

21:15 *Orientations in Incremental Forests*
Tsvi Kopelowitz

Tuesday February 16, 2021

16:00 *Approximate Near-Neighbor under the (continuous) Fréchet distance*
Anne Driemel

16:30 *Lazy Search Trees*
Sebastian Wild

17:00 *Random Access in Persistent Strings*
Inge Li Gørtz

17:30 *Parallel Batch-Dynamic Triangle Counting*
Quanquan Liu

20:00 *Open-Addressing Hashing on GPUs*
John Owens

20:30 *Connecting MapReduce Computations to Realistic Machine Models*
Peter Sanders

21:00 *Parallel SSSP using Lazy Batch Priority Queues*
Yihan Sun

21:30 *Similarity Search with Tensor Core Units*
Francesco Silvestri

Wednesday February 17, 2021

16:00 *Parallel Batch Dynamic Algorithms*
Guy Blelloch

16:30 *Fully Dynamic* $(\Delta + 1)$*-Vertex Coloring in constant time per operation*
Monika Henzinger

17:00 *Open problem session*

20:00 *Constant-length Labelling Schemes for Deterministic Radio Broadcast*
Faith Ellen

20:30 *Using GPUs to Solve the Maximum Clique Problem*
Afton Noelle Geil

21:00 *Atomic Power in Forks*
Nodari Sitchinava & Riko Jacob

21:30 *SplinterDB: A NVMe Key-Value Store on the Iacono-Patrascu Lower Bound*
Alex Conway

Thursday February 18, 2021

16:00 *Dynamic Stable Perfect Hashing, Extendable?!*
Guy Even

16:30 *Dictionary look-up with mismatches*
Tatiana Starikovskaya

17:00 *New Techniques for Proving Fine-Grained Average-Case Hardness*
Andrea Lincoln

17:30 *Self-adjusting Trees and Heaps*
László Kozma

20:00 *Vector Quotient Filters: Overcoming the Time/Space Trade-Off in Filter Design*
Rob Johnson

20:30 *Fragile Complexity of Adaptive Algorithms*
Rolf Fagerberg

21:00 *Searching Knowledge Graphs with Billions of Edges*
Hannah Bast

4 Overview of Talks

4.1 Planar Distance Oracles with Better Time-Space Tradeoffs

Seth Pettie (University of Michigan – Ann Arbor, US)

Joint work of Yaowei Long, Seth Pettie
Main reference Yaowei Long, Seth Pettie: "Planar Distance Oracles with Better Time-Space Tradeoffs", in Proc. of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021, pp. 2517–2537, SIAM, 2021.
URL https://doi.org/10.1137/1.9781611976465.149

In a recent breakthrough, Charalampopoulos, Gawrychowski, Mozes, and Weimann (2019) showed that exact distance queries on planar graphs could be answered in $n^{o(1)}$ time by a data structure occupying $n^{1+o(1)}$ space, i.e., up to $o(1)$ terms, optimal exponents in time (0) and space (1) can be achieved simultaneously. Their distance query algorithm is recursive: it makes successive calls to a point-location algorithm for planar Voronoi diagrams, which involves many recursive distance queries. The depth of this recursion is non-constant and the branching factor logarithmic, leading to $(\log n)^{\omega(1)} = n^{o(1)}$ query times.

In this paper we present a new way to do point-location in planar Voronoi diagrams, which leads to a new exact distance oracle. At the two extremes of our space-time tradeoff curve we can achieve either $n^{1+o(1)}$ space and $\log^{2+o(1)} n$ query time, or $n \log^{2+o(1)} n$ space and $n^{o(1)}$ query time.

4.2 Succinct, Stable Hash Tables

Martin Farach-Colton (Rutgers University – Piscataway, US)

Joint work of Michael Bender, Abhishek Bhattacharjee, Alex Conway, Martin Farach-Colton, Rob Johnson, Sudarsun Kannan, William Kuszmaul, Nirjhar Mukherjee, Don Porter, Guido Tagliavini, Janet Vorobyeva, Evan West
Main reference Michael Bender, Abhishek Bhattacharjee, Alex Conway, Martin Farach-Colton, Rob Johnson, Sudarsun Kannan, William Kuszmaul, Nirjhar Mukherjee, Don Porter, Guido Tagliavini, Janet Vorobyeva, Evan West: "Parallel Index-Based Structural Graph Clustering and Its Approximation". To appear in *Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

Hash tables is one of the most extensively studied data structures, and yet surprising basic questions have yet to be answered. For example, is it possible to be stable and cache efficient? Or stable and succinct? Stability is a property that has been heavily exploited in practice but has received little theoretical treatment.

We present a hash table that is stable and meets or exceeds the state of the art in CPU cost, External Memory cost, succinctness, failure probability and resizability. We then show to use this hash table to improve address translation, a critical component of virtual memory systems.

4.3 Parallel Index-Based Structural Graph Clustering and Its Approximation

Julian Shun (MIT – Cambridge, US)

Joint work of Julian Shun, Tom Tseng, Laxman Dhulipala
Main reference Tom Tseng, Laxman Dhulipala, Julian Shun: "Parallel Index-Based Structural Graph Clustering and Its Approximation", CoRR, Vol. abs/2012.11188, 2020.
URL https://arxiv.org/abs/2012.11188

SCAN (Structural Clustering Algorithm for Networks) is a well-studied, widely used graph clustering algorithm. For large graphs, however, sequential SCAN variants are prohibitively slow, and parallel SCAN variants do not effectively share work among queries with different SCAN parameter settings. Since users of SCAN often explore many parameter settings to find good clusterings, it is worthwhile to precompute an index that speeds up queries.

This talk presents a practical and provably efficient parallel index-based SCAN algorithm based on GS*-Index, a recent sequential algorithm. Our parallel algorithm improves upon the asymptotic work of the sequential algorithm by using integer sorting. It is also highly parallel; it achieves logarithmic span for both index construction and clustering queries. Furthermore, we apply locality-sensitive hashing (LSH) to design a novel approximate SCAN algorithm and prove guarantees for its clustering quality.

We present an experimental evaluation of our parallel algorithms on large real-world graphs. On a 48-core machine with two-way hyper-threading, our parallel index construction achieves 50–151× speedup over the construction of GS*-Index. In fact, even on a single thread, our index construction algorithm is faster than GS*-Index. Our parallel index query implementation achieves 5–32× speedup over GS*-Index queries across a range of SCAN parameter values, and our implementation is always faster than ppSCAN, a state-of-the-art parallel SCAN algorithm. Moreover, our experiments show that applying LSH results in much faster index construction on denser graphs without large sacrifices in clustering quality.

4.4 Dictionaries et al.

Ioana Oriana Bercea (Tel Aviv University, IL)

Joint work of Ioana Oriana Bercea, Guy Even
Main reference Ioana O. Bercea, Guy Even: "A Dynamic Space-Efficient Filter with Constant Time Operations", in Proc. of the 17th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2020, June 22-24, 2020, Tórshavn, Faroe Islands, LIPIcs, Vol. 162, pp. 11:1–11:17, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
URL https://doi.org/10.4230/LIPIcs.SWAT.2020.11

In this talk, we discuss recent advancements in dictionary and filter design. A dynamic dictionary is a data structure that maintains sets under insertions and deletions and supports membership queries of the form "is an element x in the set?". A filter performs approximate membership in the sense in which it must always answer "yes" if the element is in the set but otherwise, it can make an error with probability at most ε. Both dictionaries and filters are fundamental data structures that are employed in data management projects which require a space-efficient representation of and fast access to large datasets. In the first part of the talk, we review some of the most recent designs for dictionaries and propose some open problems on how to improve them. We then focus on two solutions to an open problem of Arbitman, Naor, and Segev [FOCS 2010] on designing dynamic dictionaries on multisets and dynamic filters. The presentation is based on two recent papers by Bercea and Even.

While a straightforward reduction turns every incremental (insertions only) dictionary into an incremental filter with similar performance guarantees, the reduction is known to fail in the dynamic setting. Conventional wisdom holds that one should instead reduce from a dynamic dictionary on multisets, in which every element can have arbitrary multiplicity. In the first result we discuss, we show that such a strong dictionary is not required. Instead, we show that it is enough to employ a dynamic dictionary for random multisets, in which each element is chosen independently and uniformly at random from the universe. We then give the first dynamic dictionary for random multisets, and subsequently, the first dynamic filter, that is space-efficient and performs all operations in constant time in the worst case. In the second result we discuss, we show how to get a dynamic dictionary for multisets via a different balls-into-bins experiment that allows us to efficiently store binary counters at the expense of only an additive linear term increase in space. We conclude with the open problem of designing a dynamic filter that performs all operations in worst-case constant time and is succinct when ε is a constant.

4.5 Fully-dynamic Planarity Testing in Polylogarithmic Time

Eva Rotenberg (Technical University of Denmark – Lyngby, DK)

Joint work of Eva Rotenberg, Jacob Holm
Main reference Jacob Holm, Eva Rotenberg: "Fully-dynamic planarity testing in polylogarithmic time", in Proc. of the Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020, pp. 167–180, ACM, 2020.
URL https://doi.org/10.1145/3357713.3384249

Given a dynamic graph subject to insertions and deletions of edges, a natural question is whether the graph presently admits a planar embedding. We give a deterministic fully-dynamic algorithm for general graphs, running in amortized $O(\log^3 n)$ time per edge insertion or deletion, that maintains a bit indicating whether or not the graph is presently planar.

4.6 Orientations in Incremental Forests

Tsvi Kopelowitz (Bar-Ilan University – Ramat Gan, IL)

Joint work of Michael Bender, Tsvi Kopelowitz, William Kuszmaul, Ely Porat, Clifford Stein

For any forest $G = (V, E)$ it is possible to orient the edges E so that no vertex in V has out-degree greater than 1. This paper considers the incremental edge-orientation problem, in which the edges E arrive over time and the algorithm must maintain a low-out-degree edge orientation at all times. We give an algorithm that maintains a maximum out-degree of 3 while flipping at most $O(\log \log n)$ edge orientations per edge insertion, with high probability in n. The algorithm requires worst-case time $O(\log n \log \log n)$ per insertion, and takes amortized time $O(1)$. The previous state of the art required up to $O(\log n / \log \log n)$ edge flips per insertion.

4.7 Approximate Near-Neighbor under the (Continuous) Fréchet Distance

Anne Driemel (Universität Bonn, DE)

Joint work of Anne Driemel, Ioannis Psarros

We study approximate-near-neighbor data structures for time series under the continuous Fréchet distance. For an attainable approximation factor $c > 1$ and a query radius r, an approximate-near-neighbor data structure can be used to preprocess n curves in $\mathbb{R}$ (aka time series), each of complexity m, to answer queries with a curve of complexity k by either returning a curve that lies within Fréchet distance cr, or answering that there exists no curve in the input within distance r. In both cases, the answer is correct. Our first data structure achieves a $(5 + \varepsilon)$ approximation factor, uses space in $n \cdot \mathcal{O}\left(\varepsilon^{-1}\right)^k + \mathcal{O}(nm)$ and has query time in $\mathcal{O}(k)$. Our second data structure achieves a $(2 + \varepsilon)$ approximation factor, uses space in $n \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k + \mathcal{O}(nm)$ and has query time in $\mathcal{O}\left(k \cdot 2^k\right)$. Our third positive

result is a probabilistic data structure based on locality-sensitive hashing, which achieves space in $\mathcal{O}(nm)$ and query time in $\mathcal{O}(k)$, and which answers queries with an approximation factor in $\mathcal{O}(k)$. All of our data structures make use of the concept of signatures, which were originally introduced for the problem of clustering time series under the Fréchet distance. In addition, we show lower bounds for this problem. Consider any data structure which achieves an approximation factor less than 2 and which supports curves of arclength up to L and answers the query using only a constant number of probes. We show that under reasonable assumptions on the word size any such data structure needs space in $L^{\Omega(k)}$.

4.8 Lazy Search Trees

Sebastian Wild (University of Liverpool, GB)

Joint work of Sebastian Wild, Bryce Sandlund
Main reference Bryce Sandlund, Sebastian Wild: "Lazy Search Trees", in Proc. of the 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020, pp. 704–715, IEEE, 2020.
URL https://doi.org/10.1109/FOCS46700.2020.00071

We introduce the lazy search tree data structure. The lazy search tree is a comparison-based data structure on the pointer machine that supports order-based operations such as rank, select, membership, predecessor, successor, minimum, and maximum while providing dynamic operations insert, delete, change-key, split, and merge. We analyze the performance of our data structure based on a partition of current elements into a set of *gaps* $\{\Delta_i\}$ based on rank. A query falls into a particular gap and *splits* the gap into two new gaps at a rank r associated with the query operation. If we define $B = \sum_i |\Delta_i| \log_2(n/|\Delta_i|)$, our performance over a sequence of n insertions and q distinct queries is $O(B + \min(n \log\log n, n \log q))$. We show B is a lower bound.

Effectively, we reduce the insertion time of binary search trees from $\Theta(\log n)$ to $O(\min(\log(n/|\Delta_i|) + \log\log|\Delta_i|,\ \log q))$, where Δ_i is the gap in which the inserted element falls. Over a sequence of n insertions and q queries, a time bound of $O(n \log q + q \log n)$ holds; better bounds are possible when queries are non-uniformly distributed. As an extreme case of non-uniformity, if all queries are for the minimum element, the lazy search tree performs as a priority queue with $O(\log\log n)$ time insert and decrease-key operations. The same data structure supports queries for *any* rank, interpolating between binary search trees and efficient priority queues.

Lazy search trees can be implemented to operate mostly on arrays, requiring only $O(\min(q, n))$ pointers, suggesting smaller memory footprint, better constant factors, and better cache performance compared to many existing efficient priority queues or binary search trees. Via direct reduction, our data structure also supports the efficient access theorems of the splay tree, providing a powerful data structure for non-uniform element access, both when the number of accesses is small and large.

4.9 Random Access in Persistent Strings

Inge Li Gørtz (Technical University of Denmark – Lyngby, DK)

Joint work of Inge Li Gørtz, Philip Bille
Main reference Philip Bille, Inge Li Gørtz: "Random Access in Persistent Strings", in Proc. of the 31st International Symposium on Algorithms and Computation, ISAAC 2020, December 14-18, 2020, Hong Kong, China (Virtual Conference), LIPIcs, Vol. 181, pp. 48:1–48:16, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
URL https://doi.org/10.4230/LIPIcs.ISAAC.2020.48
Main reference Philip Bille, Inge Li Gørtz: "Random Access in Persistent Strings", CoRR, Vol. abs/2006.15575, 2020.
URL https://arxiv.org/abs/2006.15575

We consider compact representations of collections of similar strings that support random access queries. The collection of strings is given by a rooted tree where edges are labeled by an edit operation (inserting, deleting, or replacing a character) and a node represents the string obtained by applying the sequence of edit operations on the path from the root to the node. The goal is to compactly represent the entire collection while supporting fast random access to any part of a string in the collection. This problem captures natural scenarios such as representing the past history of an edited document or representing highly-repetitive collections. Given a tree with n nodes, we show how to represent the corresponding collection in $O(n)$ space and $O(\log n/\log\log n)$ query time. This improves the previous time-space trade-offs for the problem. Additionally, we show a lower bound proving that the query time is optimal for any solution using near-linear space. To achieve our bounds for random access in persistent strings we show how to reduce the problem to the following natural geometric selection problem on line segments. Consider a set of horizontal line segments in the plane. Given parameters i and j, a segment selection query returns the jth smallest segment (the segment with the jth smallest y-coordinate) among the segments crossing the vertical line through x-coordinate i. The segment selection problem is to preprocess a set of horizontal line segments into a compact data structure that supports fast segment selection queries. We present a solution that uses $O(n)$ space and support segment selection queries in $O(\log n/\log\log n)$ time, where n is the number of segments. Furthermore, we prove that that this query time is also optimal for any solution using near-linear space.

4.10 Parallel Batch-Dynamic Triangle Counting

Quanquan C. Liu (MIT – Cambridge, US)

Joint work of Laxman Dhulipala, Quanquan C. Liu, Julian Shun, Shangdi Yu
Main reference Laxman Dhulipala, Quanquan C. Liu, Julian Shun: "Parallel Batch-Dynamic k-Clique Counting", CoRR, Vol. abs/2003.13585, 2020.
URL https://arxiv.org/abs/2003.13585

In this talk, we present new batch-dynamic algorithms for the triangle counting problem, which are dynamic algorithms where the updates are batches of edge insertions and deletions. We study this problem in the parallel setting, where the goal is to obtain algorithms with low (polylogarithmic) depth. The result we provide in this talk is a new parallel batch-dynamic triangle counting algorithm with $O(\sqrt{m})$ amortized work per update and $O(\log(m))$ depth with high probability, and $O(B+m)$ space for a batch of B edge insertions or deletions. We also present a multicore CPU implementation of our parallel batch-dynamic triangle counting

algorithm. On a 72-core machine with two-way hyper-threading, our implementation achieves 36.54–74.73× parallel speedup, and in certain cases achieves significant speedups over existing parallel algorithms for the problem, which are not theoretically-efficient.

4.11 Open-Addressing Hashing on GPUs

John Owens (University of California, Davis, US)

Joint work of Muhammad Awad, Saman Ashkiani, Serban Porumbescu, Martín Farach-Colton, John Owens

We describe the implementation and performance of several open-addressing-based hash table designs on GPUs: cuckoo hashing; bucketed cuckoo hashing; power-of-two-choices hashing; and iceberg hashing. We target query rate, build rate, and load factor as our metrics of choice, and static-sized workloads with large batches. In general, a three-hash-function bucketed cuckoo hash implementation with a bucket size of 16 was the best performer overall.

4.12 Connecting MapReduce Computations to Realistic Machine Models

Peter Sanders (KIT – Karlsruher Institut für Technologie, DE)

Main reference Peter Sanders: "Connecting MapReduce Computations to Realistic Machine Models", CoRR, Vol. abs/2002.07553, 2020.
URL https://arxiv.org/abs/2002.07553

We explain how the popular, highly abstract MapReduce model of parallel computation (MRC/MPC) can be rooted in reality by showing how to execute MapReduce computations robustly and efficiently on realistic distributed-memory parallel machines. First, a refined model MRC+ is introduced that includes parameters for total work w, bottleneck work $\hat{w}$, data volume m, and maximum object sizes $\hat{m}$. Then matching upper and lower bounds are established for executing a MapReduce calculation on distributed-memory machines – $\Theta(w/p + \hat{w} + \log p)$ work and $\Theta(m/p + \hat{m} + \log p)$ bottleneck communication volume using p processing elements. The theorem is formulated in such a way that multiple MapReduce steps can be chained. The result is obtained using a careful combination of several load balancing algorithms some of which may be of independent interest.

This also appeared at IEEE BigData 2020.

4.13 Parallel SSSP using Lazy Batch Priority Queues

Yihan Sun (University of California – Riverside, US)

Joint work of Xiaojun Dong, Yan Gu, Yihan Sun, Yunming Zhang
Main reference Xiaojun Dong, Yan Gu, Yihan Sun, Yunming Zhang: "Efficient Stepping Algorithms and Implementations for Parallel Shortest Paths", CoRR, Vol. abs/2105.06145, 2021.
URL https://arxiv.org/abs/2105.06145

In this paper, we study the single-source shortest-path (SSSP) problem with nonnegative edge weights, which is a notoriously hard problem in the parallel context. In practice, the Δ-stepping algorithm proposed by Meyer and Sanders has been widely adopted.

However, Δ-stepping has no known worst-case bounds for general graphs. The performance of Δ-stepping also highly relies on the parameter Δ, which requires exhaustive tuning. There have also been lots of theoretical algorithms, such as Radius-stepping, but they either have no implementations available or are much slower than Δ-stepping in practice.

In this paper, we propose a stepping algorithm framework that generalizes existing algorithms such as Δ-stepping and Radius-stepping, and a new abstract data type, lazy-batched priority queue, or LaB-PQ, that abstracts the semantics of the priority queue needed by the stepping algorithms. The framework allows all stepping algorithms, whether theoretical or practical, to be analyzed similarly and implemented similarly. We provide two data structures to support LaB-PQ, with the goal of theoretical and practical efficiency, respectively.

Based on the new framework and the new LaB-PQ, we show a new stepping algorithm, ρ-stepping, that is simple, supporting worst-case cost bounds, fast in practice, and preprocessing-free. Meanwhile, we also show new bounds for a list of existing algorithms, including Radius-Stepping, Δ^*-stepping (our new variant of Δ-stepping), and others.

The stepping algorithm framework also provides almost identical implementations for three algorithms: Bellman-Ford, Δ^*-stepping, and ρ-stepping. We compare our performance with four state-of-the-art implementations. On the five social and web graphs, our ρ-stepping is at least 20% faster than all the existing implementations. On the two road graphs, our Δ^*-stepping is at least 20% faster than existing implementations, while our ρ-stepping is also competitive. The almost identical implementations also allow for in-depth analyses and comparisons among the stepping algorithms in practice.

4.14 Similarity Search with Tensor Core Units

Francesco Silvestri (University of Padova, IT)

Joint work of Thomas D. Ahlem, Francesco Silvestri
Main reference Thomas D. Ahle, Francesco Silvestri: "Similarity Search with Tensor Core Units", in Proc. of the Similarity Search and Applications – 13th International Conference, SISAP 2020, Copenhagen, Denmark, September 30 – October 2, 2020, Proceedings, Lecture Notes in Computer Science, Vol. 12440, pp. 76–84, Springer, 2020.
URL https://doi.org/10.1007/978-3-030-60936-8_6

Tensor Core Units (TCUs) are hardware accelerators developed for deep neural networks, which efficiently support the multiplication of two dense $\sqrt{m} \times \sqrt{m}$ matrices, where m is a given hardware parameter. In this talk, we show that TCUs can speed up similarity search problems as well. We propose algorithms for the Johnson-Lindenstrauss dimensionality reduction and for similarity join that, by leveraging TCUs, achieve a $\sqrt{m}$ speedup up with respect to traditional approaches.

4.15 Parallel Batch Dynamic Algorithms

Guy E. Blelloch (Carnegie Mellon University – Pittsburgh, US)

Joint work of Daniel Anderson, Guy E. Blelloch
Main reference Daniel Anderson, Guy E. Blelloch: "Parallel Minimum Cuts in O(m $\log^2$(n)) Work and Low Depth", CoRR, Vol. abs/2102.05301, 2021.
URL https://arxiv.org/abs/2102.05301

In the talk I present work on dynamic algorithms that process batches of operations as well as individual operations. The advantage being that it allows for parallelism. I first present a batch incremental algorithm for MST, and describe some applications to sliding window updates. I then describe an approach to apply mixed batches of queries and updates on trees in parallel. This is used as part of recent results on a parallel algorithm for min-cut.

4.16 Fully Dynamic $(\Delta + 1)$-Vertex Coloring in Constant Time per Operation

Monika Henzinger (Universität Wien, AT)

Joint work of Monika Henzinger, Pan Peng
Main reference Monika Henzinger, Pan Peng: "Constant-Time Dynamic (Δ+1)-Coloring", in Proc. of the 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France, LIPIcs, Vol. 154, pp. 53:1–53:18, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
URL https://doi.org/10.4230/LIPIcs.STACS.2020.53

We give a fully dynamic (Las-Vegas style) algorithm with *constant expected amortized* time per update that maintains a proper $(\Delta + 1)$-vertex coloring of a graph with maximum degree at most Δ. This improves upon the previous $O(\log \Delta)$-time algorithm by Bhattacharya et al. (SODA 2018). Our algorithm uses an approach based on assigning random ranks to vertices and does not need to maintain a hierarchical graph decomposition. We show that our result does not only have optimal running time, but is also optimal in the sense that already deciding whether a Δ-coloring exists in a dynamically changing graph with maximum degree at most Δ takes $\Omega(\log n)$ time per operation.

4.17 Constant-length Labelling Schemes for Deterministic Radio Broadcast

Faith Ellen (University of Toronto, CA)

Joint work of Faith Ellen, Barun Gorain, Avery Miller, Andrzej Pelc
Main reference Faith Ellen, Barun Gorain, Avery Miller, Andrzej Pelc: "Constant-Length Labeling Schemes for Deterministic Radio Broadcast", in Proc. of the The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019, pp. 171–178, ACM, 2019.
URL https://doi.org/10.1145/3323165.3323194
Joint work of Faith Ellen, Seth Gilbert
Main reference Faith Ellen, Seth Gilbert: "Constant-Length Labelling Schemes for Faster Deterministic Radio Broadcast", in Proc. of the SPAA '20: 32nd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, July 15-17, 2020, pp. 213–222, ACM, 2020.
URL https://doi.org/10.1145/3350755.3400238

Broadcast is a fundamental network communication primitive, in which a source node has a message that has to be received by all other nodes. In synchronous radio networks, this problem is non-trivial. This follows from the fact that, if two or more neighbours of a node transmit at the same time, it hears nothing. In fact, if nodes do not store any information, broadcast is impossible deterministically, even in a four-cycle. If the nodes have distinct identifiers from a small name space, then a round-robin strategy suffices, but it takes a long time.

This talk will show that every radio network can be labelled using a small constant number of bits so that broadcast can be accomplished by a fixed deterministic algorithm that does not know the network topology nor any bound on its size. Specifically, there is a labelling scheme that stores 2 (carefully chosen) bits per nodes that allows broadcast to be performed in $O(n)$ rounds, where n is the size of the network. There is a variant of this algorithm using 4 bits per node that completes broadcast in $O(\sqrt{Dn})$ rounds, where D is the source eccentricity of the network. This number of rounds is shown to be optimal for a class of algorithms that includes both.

Then, using ideas from some old algorithms, which assume nodes have distinct identifiers, a deterministic algorithm is constructed that uses 3 bits per node and completes in $O(D \log^2 n)$ rounds. A randomized construction of a labelling scheme with 3 bits per node for a broadcast algorithm that completes in $O(D \log n + \log^2 n)$ rounds is also presented.

This talk is based on a SPAA 2019 paper with Barun Gorain, Avery Miller, and Andrzej Pelc and a SPAA 2020 paper with Seth Gilbert.

4.18 Using GPUs to Solve the Maximum Clique Problem

Afton Noelle Geil (University of California, Davis, US)

In this talk, I discuss different maximum clique algorithms and their suitability for implementation on GPUs. First, I consider depth-first branch and bound strategies, which have been used for previous parallel implementations on CPUs, but present significant challenges for efficient implementation on GPUs. I then discuss my implementation of a breadth-first maximum clique algorithm, which exposes much more parallel work for the thousands of GPU threads. I describe methods for utilizing bounds for pruning in the breadth-first

traversal to eliminate many candidate cliques; however, even with this pruning, these memory requirements are still excessive for computing breadth-first maximum clique on many graphs. Finally, I propose using a hybrid, tunable, breadth-/depth-first traversal of the search tree as a method of balancing the available parallelism and memory requirements. This choice enables us to choose a strategic ordering of vertices and to discover new lower bounds for the maximum clique size more quickly than with the breadth-first implementation, thereby increasing the effectiveness of the search tree pruning.

4.19 Atomic Power in Forks

Nodari Sitchinava (University of Hawaii at Manoa – Honolulu, US)
Riko Jacob (IT University of Copenhagen, DK)

Joint work of Michael Goodrich, Riko Jacob, Nodari Sitchinava
Main reference Michael T. Goodrich, Riko Jacob, Nodari Sitchinava: "Atomic Power in Forks: A Super-Logarithmic Lower Bound for Implementing Butterfly Networks in the Nonatomic Binary Fork-Join Model", in Proc. of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 – 13, 2021, pp. 2141–2153, SIAM, 2021.
URL https://doi.org/10.1137/1.9781611976465.128

We prove an $\Omega(\log n \log\log n)$ lower bound for the span of implementing the n input, $\log n$-depth FFT circuit (also known as butterfly network) in the nonatomic binary fork-join model. In this model, memory-access synchronizations occur only through fork operations, which spawn two child threads, and join operations, which resume a parent thread when its child threads terminate. Our bound is asymptotically tight for the nonatomic binary fork-join model, which has been of interest of late, due to its conceptual elegance and ability to capture asynchrony. Our bound implies super-logarithmic lower bound in the nonatomic binary fork-join model for implementing the butterfly merging networks used, e.g., in Batcher's bitonic and odd-even mergesort networks. This lower bound also implies an asymptotic separation result for the atomic and nonatomic versions of the fork-join model, since, as we point out, FFT circuits can be implemented in the atomic binary fork-join model with span equal to their circuit depth.

4.20 SplinterDB: A NVMe Key-Value Store on the Iacono-Patrascu Lower Bound

Alex Conway (VMware Research – Palo Alto, US)

Joint work of Vijay Chidambaram, Martin Farach-Colton, Abhishek Gupta, Rob Johnson, Richard Spillane, Amy Tai

NVMe and NVRAM present new challenges and opportunities to the world of storage systems. For example, RocksDB, a state-of-the-art key-value store, may use 30% or less of a single device's bandwidth for insertions, even when using many cores. Because key-value store performance is essential to overall application performance, this can be a critical bottleneck.

In this talk, I discuss how to understand and model performance on NVMe and how to design theoretically optimal data structures which are also fast in practice.

I present SplinterDB, a key-value store which can saturate an NVMe device with low IO amplification under a wide range of parameters. SplinterDB outperforms RocksDB by 6-9x on write-heavy workloads, and by 30-80% on read-heavy workloads.

4.21 Dynamic Stable Perfect Hashing, Extendable?!

Guy Even (Tel Aviv University, IL)

Joint work of Ioana Bercea, Guy Even

A dynamic stable perfect hashing is an injective function that assigns hashcodes in a given range to elements in a dynamically changing set such that the hashcode of an element does not change while the element is continuously in the set. We study the problem of maintaining a perfect hash function in the extendable setting, in which the goal is to design data structures whose space requirements adapt to the current cardinality of the set at all points in time.

It is not possible to support stable perfect hashing of fully dynamic sets (i.e., insertions and deletions). However, we show that it is possible to support insertions (and a few deletions provided that they only slightly reduce the cardinality). We refer to a model in which insertions are allowed (but without deletions) as the "incremental model".

We overview the previous constructions of stable dynamic perfect in the non-extendable setting by Mortensen, Pagh, Patrascu [2005] and Demaine, Mayer auf der Hyde, Pagh, Patrascu [2006]. Our observation is that these constructions are two level constructions in which the first level is, in fact, a filter without duplicates. Each level also contains a repository for free hashcodes.

Since extendable constructions for dictionaries and filters are known, the missing component for incremental stable hashing is extendable hashcode repositories (which we refer to as "motels"). We distinguish between two issues in the design of motels. The first issue is that over-provisioning in motels leads to an increase in the range of the perfect hashing. The second issue is that bigger motels (i.e., bins for hashcode repositories) lead to improved load balancing, thus reducing the probability of an overflow.

We present an extendable motel design. Our construction involves splitting of a motel in to two motels as the number of elements increases in a way that retains hashcodes of elements as long as they are continuously in the dataset.

To summarize, we present an extendable incremental stable perfect hashing data structure with the following properties. Let n_t denote the cardinality of the dataset at time t. We assume that $n_{\min} \leq n_t \leq n_{\max}$, where $n_{\max}/n_{\min} = n_{\min}^{O(1)}$. (Thus, we allow a polynomial number of insert operations.) The space of the data structure at time t is $O(n_t \cdot \log\log n_t)$. Each query and insertion is completed in constant time in the worst case. The range of the perfect hashing at time t is $[n_t + r_t]$, where $r_t = n_t/log^{O(1)} n_t$. The probability that an insert operation causes a failure is $1/poly(n_{\max})$. As a corollary, we obtain an extendable data structure for retrieval in the incremental setting.

4.22 Dictionary Look-up with Mismatches

Tatiana Starikovskaya (ENS – Paris, FR)

Main reference Vincent Cohen-Addad, Laurent Feuilloley, Tatiana Starikovskaya: "Lower bounds for text indexing with mismatches and differences", in Proc. of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019, pp. 1146–1164, SIAM, 2019.
URL https://doi.org/10.1137/1.9781611975482.70

Main reference Pawel Gawrychowski, Tatiana Starikovskaya: "Streaming Dictionary Matching with Mismatches", in Proc. of the 30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy, LIPIcs, Vol. 128, pp. 21:1–21:15, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
URL https://doi.org/10.4230/LIPIcs.CPM.2019.21

Main reference Pawel Gawrychowski, Gad M. Landau, Tatiana Starikovskaya: "Fast Entropy-Bounded String Dictionary Look-Up with Mismatches", in Proc. of the 43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK, LIPIcs, Vol. 117, pp. 66:1–66:15, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
URL https://doi.org/10.4230/LIPIcs.MFCS.2018.66

In the problem of dictionary look-ups with k mismatches, we are given a dictionary of strings and must preprocess it into the data structure that supports the following queries: given a string Q, find all strings in the dictionary within Hamming distance k from Q. We discuss recent advances for this problem, both on the upper bound side and the lower bound side.

4.23 New Techniques for Proving Fine-Grained Average-Case Hardness

Andrea Lincoln (MIT – Cambridge, US)

Joint work of Mina Dalirrooyfard, Andrea Lincoln, Virginia Vassilevska Williams.
Main reference Mina Dalirrooyfard, Andrea Lincoln, Virginia Vassilevska Williams: "New Techniques for Proving Fine-Grained Average-Case Hardness", CoRR, Vol. abs/2008.06591, 2020.
URL https://arxiv.org/abs/2008.06591

In this talk I will cover a new technique for worst-case to average-case reductions. There are two primary concepts introduced in this talk: "factored" problems and a framework for worst-case to average-case fine-grained (WCtoACFG) self reductions. We will define new versions of OV, kSUM and zero-k-clique that are both worst-case and average-case fine-grained hard assuming the core hypotheses of fine-grained complexity. We then use these as a basis for fine-grained hardness and average-case hardness of other problems. Our hard factored problems are also simple enough that we can reduce them to many other problems, e.g. to edit distance, k-LCS and versions of Max-Flow. We further consider counting variants of the factored problems and give WCtoACFG reductions for them for a natural distribution. To show hardness for these factored problems we formalize the framework of [Boix-Adsera et al. 2019] that was used to give a WCtoACFG reduction for counting k-cliques. We define an explicit property of problems such that if a problem has that property one can use the framework on the problem to get a WCtoACFG self reduction. In total these factored problems and the framework together give tight fine-grained average-case hardness for various problems including the counting variant of regular expression matching.

4.24 Self-adjusting Trees and Heaps

László Kozma (FU Berlin, DE)

Self-adjusting data structures allow a flexible structure, with little or no bookkeeping, and are usually easy to implement. Their analysis is, however, often difficult, requiring the development of sophisticated potential functions or other techniques. There are several natural self-adjusting strategies, variants of classical algorithms, that seem reasonable, but whose analysis is lacking or incomplete. In my talk I review a number of results and open questions related to such structures.

4.25 Vector Quotient Filters: Overcoming the Time/Space Trade-Off in Filter Design

Rob Johnson (VMware – Palo Alto, US)

4.26 Fragile Complexity of Adaptive Algorithms

Rolf Fagerberg (University of Southern Denmark – Odense, DK)

Joint work of Prosenjit Bose, Pilar Cano, Rolf Fagerberg, John Iacono, Riko Jacob, Stefan Langerman
Main reference Prosenjit Bose, Pilar Cano, Rolf Fagerberg, John Iacono, Riko Jacob, Stefan Langerman: "Fragile Complexity of Adaptive Algorithms", CoRR, Vol. abs/2102.00338, 2021.
URL https://arxiv.org/abs/2102.00338

The fragile complexity of a comparison-based algorithm is $f(n)$ if each input element participates in $O(f(n))$ comparisons. We explore the fragile complexity of algorithms adaptive to various restrictions on the input, i.e., algorithms with a fragile complexity parameterized by a quantity other than the input size n. We show that searching for the predecessor in a sorted array has fragile complexity $\Theta(\log k)$, where k is the rank of the query element, both in a randomized and a deterministic setting. For predecessor searches, we also show how to optimally reduce the amortized fragile complexity of the elements in the array. We also prove the following results: Selecting the kth smallest element has expected fragile complexity $O(\log\log k)$ for the element selected. Deterministically finding the minimum element has fragile complexity $\Theta(\log(\mathrm{Inv}))$ and $\Theta(\log(\mathrm{Runs}))$, where Inv is the number of inversions in a sequence and Runs is the number of increasing runs in a sequence. Deterministically finding the median has fragile complexity $O(\log(\mathrm{Runs}) + \log\log n)$ and $\Theta(\log(\mathrm{Inv}))$. Deterministic sorting has fragile complexity $\Theta(\log(\mathrm{Inv}))$ but it has fragile complexity $\Theta(\log n)$ regardless of the number of runs.

4.27 Searching Knowledge Graphs with Billions of Edges

Hannah Bast (Universität Freiburg, DE)

Joint work of Hannah Bast, Johannes Kalmbach
Main reference Hannah Bast, Johannes Kalmbach, Theresa Klumpp, Florian Kramer, Niklas Schnelle: "Efficient SPARQL Autocompletion via SPARQL", CoRR, Vol. abs/2104.14595, 2021.
URL https://arxiv.org/abs/2104.14595

In the talk, I gave a quick introduction to knowledge graphs and their standard query language SPARQL, a variant of SQL. I presented QLever, our SPARQL engine that can efficiently answer queries on knowledge graphs with tens of billions of triples on a single machine. SPARQL queries can be very complex and hard to formulate even for experts. I presented an autocompletion mechanism that allows a user to type SPARQL queries incrementally by obtaining, after each keystroke, context-sensitive suggestions on how to continue the query. The beauty of the mechanism is that the suggestions are themselves SPARQL queries, which are supported by QLever efficiently enough to enable interactive suggestions even for huge knowledge graphs. This feat is achieved by no other SPARQL engine on the market.

5 Open problems

In addition to the many open problems mentioned during the talks, the seminar on Wednesday included an open problem session during which the following problems were discussed.

5.1 Open Problem 1: A Suggestion for the Extension of the Word RAM Model

Guy Even (Tel Aviv University, IL)

The Word RAM Model was formulated by Fredman and Willard in 1990. This formulation was invented as an answer to the sorting algorithm of Paul and Simon [1984] that sorts in linear time but uses operands whose length is $n^2 \log u$ bits long. The Word length (in bits) is denoted by w and it satisfies $w = \log u$, where u denotes the size of the universe. Thus, every element can be represented by a single word. One can access (read or write) a word in memory in constant time. In addition, the following instructions require constant time: addition, subtraction, multiplication, division, shifting, and bitwise operations (AND, OR, XOR). Works based on the Word RAM Model often employ sophisticated manipulations to perform fast computations using only the basic instructions of the Word RAM Model (see Baumann and Hagerup 2018 and references therein). Circuits that implement the constant-time arithmetic instructions of the Word RAM model require logarithmic depth. The size of practical multipliers is quadratic. In light of this fact, we propose the following extension of the Word RAM model.

Extended Word RAM Model

Extend the Word RAM Model so that every Boolean function over $O(w)$ bits that can be computed by a circuit of depth $O(\log w)$ and size $O(w^2)$ is executable in constant time. Why extend the Word RAM model?

1. The set of instructions in the Word RAM model is simply taken from instructions that are available in CPU's (naturally, the execution of these instructions requires a constant number of clock cycles).
2. Spare us from the pain of "bit-games" needed to implement new instructions using old instructions.
3. If a new instruction is useful, it will be added to CPU instruction sets and a special circuit will be used to compute this instruction.

5.2 Open Problem 2: Dynamic All Pairs Shortest Paths

Monika Henzinger (Universität Wien, AT)

Give a fully dynamic algorithm for all pairs shortest paths (APSP) with $\tilde{O}(m)$ time per update operation and $\tilde{O}(1)$ time per query.

5.3 Open Problem 3: Sequences of Heap Operations

László Kozma (FU Berlin, DE)

Consider a simple heap supporting two operations: Insert and DeleteMin. In the comparison model one of the two operations must take $\Omega(\lg n)$ time if the heap has n elements by the lower bound for sorting. What is the complexity if the sequence of operations is known in advance and we must report the remaining elements? E.g. for the sequence

Ins(5), Ins(3), Ins(7), DeleteMin, Ins(9), Ins(6), Ins(10), DeleteMin, DeleteMin

we must report the elements $7, 9, 10$. Can we compute the result in $o(n \lg n)$ or maybe $O(n)$ comparisons?

Observation 1

The number of possible answers is $< 2^n$, which means the simple lower does not apply.

Observation 2

If all Insert operations are before all DeleteMin operations we can report the elements in $O(n)$ time using linear time selection.

Participants

- Kunal Agrawal
Washington University – St. Louis, US
- Elena Arseneva
St. Petersburg State University, RU
- Hannah Bast
Universität Freiburg, DE
- Ioana Oriana Bercea
Tel Aviv University, IL
- Guy E. Blelloch
Carnegie Mellon University – Pittsburgh, US
- Gerth Stølting Brodal
Aarhus University, DK
- Rezaul Chowdhury
Stony Brook University, US
- Alexander Conway
VMware Research – Palo Alto, US
- Martin Dietzfelbinger
TU Ilmenau, DE
- Anne Driemel
Universität Bonn, DE
- Amalia Duch Brown
UPC Barcelona Tech, ES
- Faith Ellen
University of Toronto, CA
- Guy Even
Tel Aviv University, IL
- Rolf Fagerberg
University of Southern Denmark – Odense, DK
- Martin Farach-Colton
Rutgers University – Piscataway, US
- Afton Noelle Geil
University of California – Davis, US
- Inge Li Gørtz
Technical University of Denmark – Lyngby, DK
- Monika Henzinger
Universität Wien, AT
- John Iacono
UL – Brussels, BE
- Riko Jacob
IT University of Copenhagen, DK
- Rob Johnson
VMware – Palo Alto, US
- Valerie King
University of Victoria, CA
- Tsvi Kopelowitz
Bar-Ilan University – Ramat Gan, IL
- László Kozma
FU Berlin, DE
- Moshe Lewenstein
Bar-Ilan University – Ramat Gan, IL
- Andrea Lincoln
MIT – Cambridge, US
- Quanquan C. Liu
MIT – Cambridge, US
- Ulrich Carsten Meyer
Goethe-Universität – Frankfurt am Main, DE
- Ian Munro
University of Waterloo, CA
- Markus E. Nebel
Universität Bielefeld, DE
- Eunjin Oh
POSTECH – Pohang, KR
- John Owens
University of California – Davis, US
- Rotem Oshman
Tel Aviv University, IL
- Seth Pettie
University of Michigan – Ann Arbor, US
- Vijaya Ramachandran
University of Texas – Austin, US
- Rajeev Raman
University of Leicester, GB
- Eva Rotenberg
Technical University of Denmark – Lyngby, DK
- Peter Sanders
KIT – Karlsruher Institut für Technologie, DE
- Robert Sedgewick
Princeton University, US
- Julian Shun
MIT – Cambridge, US
- Francesco Silvestri
University of Padova, IT
- Nodari Sitchinava
University of Hawaii at Manoa – Honolulu, US
- Tatiana Starikovskaya
ENS – Paris, FR
- Jesper Steensgard
Aarhus University, DK
- Yihan Sun
University of California – Riverside, US
- Robert Endre Tarjan
Princeton University, US
- Manoharan Vignesh
University of Texas – Austin, US
- Sebastian Wild
University of Liverpool, GB