1 Introduction

Broadly speaking, I am interested in the design and analysis of data structures and graph algorithms. More specifically, my work focuses on simple, core problems in those areas, seeking to understand them on a fundamental level and explain how the theory and practice relate. To that end I have engaged in broad, exploratory experimentation regarding priority queues [8]; proposed simple algorithms with better theoretical guarantees for diameter approximation [2]; and explored the design space of compressed tree algorithms in my thesis, providing theoretical backing for experimental findings [5], solving a newly proposed problem [9], and offering a unified analysis framework based on identifying key properties common to several algorithms. The border between theory and practice offers an incredibly rich environment for research—gaining a better understanding of one often leads to advances in the other—and yet it is often overlooked in favor of working strictly in one of the two paradigms.

Focusing on gaining a deeper understanding of core problems has beneficial side effects. With little specialized skill required, it lowers the barrier to entry and makes it easy to discuss with individuals of varying research backgrounds. It can take only a few minutes to explain a problem to students or colleagues, and they can realistically begin collaborating on the spot. This offers opportunities for interested students to get involved, even without a strong theoretical background. Furthermore, we can incorporate these sort of insights into mainstream pedagogy rather than relegate them to conferences and graduate-level special topics courses.

2 Previous and Ongoing Work

Priority queues. I began my research career as an undergraduate working on an experimental study of priority queues, and this work continued into my graduate studies. I recall being confused by the gap between theory and practice for this fundamental data structure. On one hand, it seemed that just about every application in practice used binary heaps [16] or some derivative thereof. On the other hand, theoretical applications typically used Fibonacci heaps [4], which outperform binary heaps in the worst case. The conventional wisdom said that Fibonacci heaps are simply too slow in practice, and a line of recent papers [1, 3, 6, 7, 14] sought to provide priority queues that matched the Fibonacci bounds in theory while being
more efficient in practice. The experimental studies backing the conventional wisdom [10, 13] were nearly two decades old.

I implemented several priority queues, both old and new, in a straight-forward fashion from their original theoretical publications. Gathering a wide variety of data sets, including real workloads from production environments, we uncovered some interesting results. The conventional wisdom certainly did not tell the whole story. The relative performance of different priority queues was highly input-dependent. The Fibonacci heap is not slower than the binary heap as a rule, but merely on certain classes of input. On others, it is faster by a wide margin. The Fibonacci heap, in fact, also outperformed the recent, supposedly more practical priority queues on almost every input. Furthermore, low-level caching patterns were found to be the best predictor of relative performance. This work may provide guidance to future researchers, both theoreticians and practitioners alike, about how to design and implement priority queues for varying applications.

**Diameter approximation.** The diameter of a graph is a fundamental parameter, and computing it has many applications. The most efficient known methods for finding the exact answer amount to solving the all-pairs shortest paths problem. It is natural to relax the requirement to an approximate answer in search of a more efficient algorithm, and indeed, prior work [12] provided an $O(m\sqrt{n})$ algorithm which guaranteed an optimal $3/2$-approximation for the case of unweighted graphs. Our work extends the previous algorithm to achieve the same approximation guarantee and a running time of $O\left(m^{3/2}\right)$. We proposed a second algorithm which runs in $O(mn^{2/3})$ time, covering the case of dense graphs. While we have simple, subquadratic algorithms for approximating the diameter of weighted, directed graphs, the asymptotic complexity of the problem remains an open question.

**Compressed trees.** The compressed tree framework is essential to many algorithms that provide efficient computation on trees and other graphs. The classic example is the disjoint set union (union-find) problem, which maintains a partition of $n$ elements into disjoint sets, subject to $m$ intermixed unite and find operations. In this particular setting, the goal is to link trees together in a way to minimize their height during unite operations while simultaneously traversing and flattening paths during find operations.

Using sufficient linking rules and compaction methods, one can achieve an amortized $O(\alpha(n, m/n))$ bound per operation [15]. More naïve methods, such as linking by index, only achieve an $O(\log n)$ bound; yet, in certain experiments [11], linking by index was found to perform best. The input sequences were randomized so that the node indices were essentially a uniformly random permutation. Formally, we call this randomized linking. We showed that randomized linking achieves the $O(\alpha(n, m/n))$ bound in expectation.

We also studied the case where two partitions are maintained on the same set of elements subject to the constraint that one partition is a refinement of the other. We call this the nested set union problem. It was natural to wonder whether the nested structure of the sets could be exploited to save space. We answered in the affirmative, matching the $O(\alpha(n, m/n))$ bound using only a few extra bits of storage per node. Our solution scales to handle any constant number of nested partitions with little overhead.

Our work on the nested set union problem led me to a few key insights about the nature
of optimal compressed tree algorithms. In particular, I proved a meta-theorem which applies to any algorithm satisfying two key properties common to virtually all known optimal algorithms. There is hope that this will make it considerably easier to analyze algorithms for the path evaluation problem. In this setting we perform path compaction, but we are not allowed to use linking rules. The best known algorithm achieves a $O(\log^* n)$ bound per operation, and is very complicated and not intuitive. Either improving the bound or providing a simpler algorithm matching the same bound would make considerable progress on the problem. I am actively pursuing this line of research.

3 Future Research

Though I have already spoken generally about the type of problems that interest me and touched lightly on avenues leading from my past work, I should discuss my plans for future research more explicitly. Here are a few areas which have yet to receive my full attention.

Shortest path and distance oracle problems have broad applications in both theory and practice. There has been consistent incremental improvement on both fronts. With research groups consistently trying to improve map search products, practically efficient algorithms are often not analyzed theoretically, or at best given loose, poor bounds. Some effort has been made to rectify this, but there is still a large gap between theory and practice.

The minimum spanning tree problem is one of the oldest and most studied problems in computer science. The best algorithm with known complexity is complicated, and the analysis is essentially baked into the algorithm itself. One could hope to offer a simpler algorithm and perhaps even improve the running time by teasing this structure out of the algorithm, making it a more natural process.

Planar network simplification via series-parallel and Y-Δ transformations is a simple tool often used and taught in electrical engineering. The best known algorithm for this process achieves a quadratic upper bound, while the best known lower bound is only linear. Making progress toward resolving the complexity could have pedagogical implications.

References


