Algorithm Engineering for Sorting and Searching, and All That

Stefan Edelkamp
University of Koblenz, Postfach 201 602, 56016 Koblenz, Germany
edelkamp@uni-koblenz.de

Abstract

We look at several proposals to engineer the set of fundamental searching and sorting algorithms. Aspects are improving locality of disk access and cache access, the efficiency tuning by reducing the number of branch mispredictions, and reducing at leading factors hidden in the Big-Oh notation. These studies in algorithm engineering, in turn, lead to exiting new algorithm designs. On the practical side, we will establish that efficient sorting and searching algorithms are in tight collaboration, as sorting is used for finding duplicates in disk-based search, and heap structures designed for efficient graph search can be exploited in classical and adaptive sorting. We indicate the effects of engineered sorting and searching for combined task and motion planning.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Sorting, Searching, Algorithm Engineering

Digital Object Identifier 10.4230/LIPIcs.SEA.2020.2

Category Invited Talk

Acknowledgements I want to thank all co-authors of my papers.

1 Introduction

Several decades ago, in the early days of computer science Donald E. Knuth and Kurt Mehlhorn both dedicated a book of their monographs to the topic of sorting and searching. Since then, driven by the advances in computer hardware technology there have been several proposals to engineer the set of fundamental algorithms. One aspect we look at is improving locality of disk access and cache access, another one efficiency tuning by reducing the number of branch mispredictions. We also will look at leading factors hidden in Landau's Big-Oh notation to study how far the algorithms are from their respective lower bounds. We highlight one result in sorting and one in searching, as well as one possible application area.

2 Sorting

QuickXsort is a highly efficient in-place sequential sorting scheme that mixes Hoare's Quicksort algorithm with X, where X can be chosen from a wider range of other known sorting algorithms, like Heapsort, Insertionsort and Mergesort. Its major advantage is that QuickXsort can be in-place even if X is not. We provide general transfer theorems expressing the number of comparisons of QuickXsort in terms of the number of comparisons of X. More specifically, if pivots are chosen as medians of (not too fast) growing size samples, the average number of comparisons of QuickXsort and X differ only by $o(n)$-terms. For median-of-k pivot selection for some constant k, the difference is a linear term whose coefficient we compute precisely. For instance, median-of-three QuickMergesort uses at most $n \lg n - 0.8358n + O(\log n)$ comparisons. Furthermore, we examine the possibility of sorting base cases with some other algorithm using even less comparisons. By doing so the average-case number of comparisons can be reduced down to $n \lg n - 1.4112n + o(n)$ for a remaining gap of only $0.0315n$ comparisons

to the known lower bound (while using only $O(\log n)$ additional space and $O(n \log n)$ time overall). Implementations of these sorting strategies show that the algorithms challenge well-established library implementations like Musser's Introsort.

3 Searching

A priority queue – a data structure supporting, the operations minimum (top), insert (push), and extract-min (pop) – is said to operate in-place if it uses O(1) extra space in addition to the n elements stored at the beginning of an array. Prior to this work, no in-place priority queue was known to provide worst-case guarantees on the number of element comparisons that are optimal up to additive constant terms for both insert and extract-min. In particular, for the standard implementation of binary heaps, insert and extract-min operate in logarithmic time while involving at most $\lceil \lg n \rceil$ and $2 \lg n$ [could possibly be reduced to $\lg \lg n + O(1)$ and $\lg n + \log^* n + O(1)$] element comparisons, respectively. We propose a variant of a binary heap that operates in-place, executes minimum and insert in $O(1)$ worst-case time, and extract-min in $O(\lg n)$ worst-case time while involving at most $\lg n + O(1)$ element comparisons. These efficiencies surpass lower bounds known for binary heaps, thereby resolving a long-standing theoretical debate.

4 Application

Logistics operations often require a robot to pickup and deliver objects from multiple locations within certain time frames. This is a challenging task-and-motion planning problem as it intertwines logical and temporal constraints about the operations with geometric and differential constraints related to obstacle avoidance and robot dynamics. To address these challenges, we couple vehicle routing over a discrete abstraction with sampling-based motion planning. On the one hand, vehicle routing provides plans to effectively guide sampling-based motion planning as it explores the vast space of feasible motions. On the other hand, motion planning provides feasibility estimates which vehicle routing uses to refine its plans. This coupling makes it possible to extend the state-of-the-art in multi-goal motion planning by also incorporating capacities, pickups, and deliveries in addition to time windows. When not all pickups and deliveries can be completed in time, the approach seeks to minimize the violations and maximize the profit.

5 Conclusion

We passed by various priority queue designs for pimping up shortest path search and recent mixtures of sorting algorithms that show both outstanding theoretical and practical performances. On the theoretical side we introduced an in-place heap, for which minimum and insert take $O(1)$ worst-case time, and extract-min takes $O(\lg n)$ worst-case time and involves at most $\lg n + O(1)$ element comparisons. We established an algorithm which in a sequence of n min-deletes and m decrease-keys requires $2m + 1.5n \lg n + o(n)$ comparisons. In sorting, the average-case number of comparisons can be reduced down to $n \lg n - 1.4112n + o(n)$ for a remaining gap of only $0.0315n$ comparisons to the known lower bound, while using only $O(\lg n)$ additional space and $O(n \lg n)$ time overall.

On the practical side, we established that efficient sorting and searching algorithms are in tight collaboration, as sorting is used for finding duplicates in disk-based search, and new heap structures designed for efficient graph search can be exploited in classical and adaptive sorting. We indicated the effects engineered sorting and searching for combined task and motion planning in robotics.

References

1 Stefan Edelkamp. External-memory state space search. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering - Selected Results and Surveys*, volume 9220 of *LNCS*, pages 185–225. Springer, 2016. doi:10.1007/978-3-319-49487-6_6.

2 Stefan Edelkamp. Improving the cache-efficiency of shortest path search. In Gabriele Kern-Isberner, Johannes Fürnkranz, and Matthias Thimm, editors, *KI 2017*, volume 10505 of *LNCS*, pages 99–113. Springer, 2017. doi:10.1007/978-3-319-67190-1_8.

3 Stefan Edelkamp and Tristan Cazenave. Improved diversity in nested rollout policy adaptation. In Gerhard Friedrich, Malte Helmert, and Franz Wotawa, editors, *KI 2016*, volume 9904 of *LNCS*, pages 43–55. Springer, 2016. doi:10.1007/978-3-319-46073-4_4.

4 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. Two constant-factor-optimal realizations of adaptive heapsort. In Costas S. Iliopoulos and William F. Smyth, editors, *IWOCA 2011*, volume 7056 of *LNCS*, pages 195–208. Springer, 2011. doi:10.1007/978-3-642-25011-8_16.

5 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. The weak-heap data structure: Variants and applications. *J. Discrete Algorithms*, 16:187–205, 2012. doi:10.1016/j.jda.2012.04.010.

6 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. Weak heaps engineered. *J. Discrete Algorithms*, 23:83–97, 2013. doi:10.1016/j.jda.2013.07.002.

7 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. An in-place priority queue with $o(1)$ time for push and $\lg n + O(1)$ comparisons for pop. In Lev D. Beklemishev and Daniil V. Musatov, editors, *CSR 2015*, volume 9139 of *LNCS*, pages 204–218. Springer, 2015. doi:10.1007/978-3-319-20297-6_14.

8 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. Heap construction - 50 years later. *Comput. J.*, 60(5):657–674, 2017. doi:10.1093/comjnl/bxw085.

9 Stefan Edelkamp, Amr Elmasry, and Jyrki Katajainen. Optimizing binary heaps. *Theory Comput. Syst.*, 61(2):606–636, 2017. doi:10.1007/s00224-017-9760-2.

10 Stefan Edelkamp, Max Gath, Tristan Cazenave, and Fabien Teytaud. Algorithm and knowledge engineering for the TSPTW problem. In *CISched 2013*, pages 44–51. IEEE, 2013. doi:10.1109/SCIS.2013.6613251.

11 Stefan Edelkamp, Peter Kissmann, and Martha Rohte. Symbolic and explicit search hybrid through perfect hash functions - A case study in connect four. In Steve A. Chien, Minh Binh Do, Alan Fern, and Wheeler Ruml, editors, *ICAPS 2014*. AAAI, 2014. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7921.

12 Stefan Edelkamp, Morteza Lahijanian, Daniele Magazzeni, and Erion Plaku. Integrating temporal reasoning and sampling-based motion planning for multigoal problems with dynamics and time windows. *Rob. Autom. Lett.*, 3(4):3473–3480, 2018. doi:10.1109/LRA.2018.2853642.

13 Stefan Edelkamp, Erion Plaku, and Yassin Warsame. Monte-carlo search for prize-collecting robot motion planning with time windows, capacities, pickups, and deliveries. In Christoph Benzmüller and Heiner Stuckenschmidt, editors, *KI 2019*, volume 11793 of *LNCS*, pages 154–167. Springer, 2019. doi:10.1007/978-3-030-30179-8_13.

14 Stefan Edelkamp and Armin Weiß. Blockquicksort: Avoiding branch mispredictions in quicksort. *ACM Journal of Experimental Algorithmics*, 24(1):1.4:1–1.4:22, 2019. doi:10.1145/3274660.

15 Stefan Edelkamp and Armin Weiß. Worst-case efficient sorting with quickmergesort. In Stephen G. Kobourov and Henning Meyerhenke, editors, *ALENEX 2019*, pages 1–14, 2019. doi:10.1137/1.9781611975499.1.

16 Stefan Edelkamp, Armin Weiß, and Sebastian Wild. Quickxsort: A fast sorting scheme in theory and practice. *Algorithmica*, 82(3):509–588, 2020. doi:10.1007/s00453-019-00634-0.

17 Erion Plaku, Sara Rashidian, and Stefan Edelkamp. Multi-group motion planning in virtual environments. *J. of Visualization and Comp. Animation*, 29(6), 2018. doi:10.1002/cav.1688.

18 Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp. Efficient symbolic search for cost-optimal planning. *Artif. Intell.*, 242:52–79, 2017. doi:10.1016/j.artint.2016.10.001.