The Space Race: Progress in Algorithm Space Complexity

by

Hayden Rome

S.B., Computer Science and Engineering and Mathematics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored by:	Hayden Rome Department of Electrical Engineering and Computer Science May 12, 2023
Certified by:	Neil Thompson Research Scientist, MIT CSAIL Thesis Supervisor
Certified by:	Jayson Lynch Research Scientist, MIT CSAIL Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

The Space Race: Progress in Algorithm Space Complexity

by

Hayden Rome

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This paper presents the first broad survey of the space complexities of algorithms for important problems in computer science, analyzing more than 800 algorithms for different problem families, and comparing the different algorithms for each of these problem families. The survey reveals the increasing importance of space complexity in recent years and discusses its relationship with time complexity. Our findings reveal an increasing trend in the percentage of algorithm papers that include space complexity analysis. We identify an increasing trend in the percentage of problem families with asymptotic time-space tradeoffs. Additionally, we find that the few problem families that see improvements in space complexity have typically improved at rates faster than the improvement rates of DRAM access speed and DRAM capacity. Under the right conditions, these algorithmic improvements to space complexity can be much more important than hardware improvements when considering computational speedups related to data accesses. This study sheds light on the space complexity of algorithms and contributes to a better understanding of the relationship between time and space complexities. We have also uploaded the space complexity work for this paper to our website, The Algorithm Wiki[1], to serve as a useful resource for theorists and practitioners alike.

Thesis Supervisor: Neil Thompson
Title: Research Scientist, MIT CSAIL

Thesis Supervisor: Jayson Lynch
Title: Research Scientist, MIT CSAIL

[1] https://algorithm-wiki.csail.mit.edu

Acknowledgments

Fellow MIT students Jeffrey Li, Chirag Falor, and Khaleel al-Adhami worked on the gathering and derivation of space complexities with me. For this work, my roles included doing the analysis, helping gather and derive space complexities, and writing this paper. Jayson Lynch has provided invaluable oversight and help in all aspects of this work, from checking space complexity proofs to looking over my writing, as well as acting as a mentor. Chirag and Jayson also provided help in conducting a search for similar space complexity survey work for the introduction of this paper. Jayson, Jeffrey, Chirag, and I also plan to prepare a jointly-authored research paper on this work as well as the additional document containing all of our space complexity proofs in the near future, under the supervision of Neil Thompson.

I would also like to acknowledge Matrin Farach-Colton and Erik Demaine for useful discussion and feedback regarding different models of computation. In particular, Martin pointed out space complexity can be larger than time complexity, and with Erik we worked out the standard models in which that holds. Also, Michael Coulombe, one of Erik's students, helped us find a specific type of Turing machine, the log-space transducer, that we use in our discussion of constant auxiliary space in Word RAM.

I'd also like to thank Ryan Williams and Virginia Vassilevska Williams for their insight into specific problems' space complexities, especially APSP and matrix multiplication.

The Director of FutureTech Neil Thompson provided plentiful guidance throughout the entireity of this work. I would especially like to thank Neil for providing me with the opportunity to do this work and for supporting my MEng this past year.

Contents

List of Figures

Chapter 1

Introduction

In this work, we aim to provide a comprehensive survey of the space complexities of algorithms for important problems in computer science. Previous work from Sherry and Thompson [10] gave a similar survey on time complexities. In their work, they identified the most important problems in computer science and analyzed how fast algorithms improve, comparing that rate with the rate of improvement of hardware processing speed. The scope of this paper is the analysis of the space complexities of algorithms from Sherry and Thompson's work [10] as a whole, rather than discussing the specific individual space complexities of algorithms. In addition to this paper, we will provide an addendum that contains our space complexity notes (brief proofs) for each of the more than 800 algorithms that we studied. In addition to gathering and deriving all of these space complexities, we analyze trends in literature, noting how many algorithm papers actually include space complexity derivations, as well as comparing the rates of improvement in space complexity with other rates of improvement in computing, such as DRAM and time complexity. Furthermore, we have populated the Algorithm Wiki website[1] with all the algorithms in our database as well as their respective space complexities.

We also comment on the history and progress of algorithms and algorithmic research, to understand what areas of algorithms researchers care about, and in what areas there are opportunities for progress.

[1] https://algorithm-wiki.csail.mit.edu

Furthermore, we present the rate at which space complexity has been improving, and what this means for “memory wall" issues. In hardware, the processing speed of CPUs is improving faster than memory access speed. The difference in speed has become large enough to where memory access speed is becoming a significant bottleneck in modern computing [14]. This is known as the “memory wall", and in this work, we address a similar idea in software–comparing the improvement rates of space complexity and time complexity. We find that the rate of improvement in space complexity is slower than both the rate of improvement in memory hardware as well as the rate of improvement in time complexity. Space complexity improvements do not necessarily alleviate memory wall issues, since even if you store less data, you still may have the same number of total data accesses. However, storing less data means that you can potentially move more of the data up the memory hierarchy, speeding up the accesses for that data.

This paper is organized as follows. In Section 1.1, we discuss previous algorithmic surveys. In Section 1.2, we provide a brief overview of algorithmic problems, algorithms, and asymptotic time and space complexity. In Section 2, we present the main results of the analysis of space complexities. In Section 3, we describe the methodology we used to derived space complexities. In Section 4, we discuss some future work planned in the lab regarding other aspects of algorithmic development.

1.1 Related Work

The concept of space complexity was introduced in 1965 by Stearns [12]. Since then, space complexity has been an important area of research in algorithm analysis, alongside time complexity. While there have been some surveys of space complexity, such as Michel’s work [8], there are no space complexity surveys that analyze the field of algorithms as a whole like we do here. Some textbooks, such as Alfred [1] and CLRS [3], have provided analyses of space complexity for fundamental algorithms, but they are limited in scope. In this work, we consider not just the most pedagogically important algorithms, but all algorithms for a given problem that appear in any

of the 57 textbooks and course descriptions from highly regarded algorithms courses, surveyed by [10].

Leiserson et al. [6] argue that post-Moore's-Law progress in computing will come in three main areas–hardware, software, and algorithms. Algorithms are the problem-solving techniques implemented by this software. If we can solve the same problem with a faster algorithm, then we are improving our effective computing power. This algorithmic progress has been studied previously by Sherry and Thompson [10]. They look at improvements in time complexity for 113 different *problem families*. Each problem family has a specific problem that defines the family. An algorithm belongs to a problem family if the algorithm solves a variation of the problem associated with that problem family. An example of a problem family is 'Comparison Sorting', and an algorithm that would belong to that problem family is 'Merge Sort'.

The work by Sherry and Thompson [10], and related work by Liu [7], focused on the time complexity of algorithms and the publication of new, faster algorithms within the different problem families. In this paper, we turn the focus from time complexity to space complexity. How space-efficient are the different algorithms for a problem family, and how quickly do more space-efficient algorithms arise?

In this paper, we study the same sampling of algorithms and problem families as Sherry and Thompson [10] and Liu [7], but we extend that analysis to include space complexity–something that these papers do not address, nor do any other large algorithmic surveys. As part of this paper's contribution, we have derived and gathered the space complexities of more than 800 algorithms from this sample–many of which the original papers did not even mention their algorithm's space complexity.

1.2 Background

In computer science, an *algorithmic problem* is defined as a relation from a set of values called *inputs* to a set of values called *outputs*, and an *algorithm* is defined as a (finite) sequence of instructions that are used to solve an algorithmic problem. For example, integer addition can be thought of as an algorithmic problem, where the input consists

of a set of two integers, and the output is another integer. In many cases, an input will correspond to exactly one or even multiple correct outputs. There are cases, however, where the output of a problem is not well defined. For example, with the problem 'texture synthesis', which is the problem of trying to generate an image that looks like it has a given texture, there is not an agreed upon way to determine exactly whether the output image is correct or not. We consider problems like this to be "inexact problems", and they are not included in our analysis. Algorithms can be deterministic or randomized, and in the case of randomized algorithms, we often consider the probability that inputs get mapped to a correct output. In this work, we exclude randomized algorithms from our analysis since it is not straightforward to compare them with deterministic algorithms in a way that also incorporates the probability that the randomized algorithm produces a correct output.

In the analysis of algorithms, two important metrics are *time complexity* and *space complexity*. These consider how long it takes an algorithm to produce an output, and how much information an algorithm needs to store at a time while performing its set of instructions. To make our algorithms more efficient, we want to reduce the amount of time and storage our algorithm uses while solving a problem.

We usually discuss these metrics with respect to the size of the input, and focus on when the size of the input is large or tends to infinity. As such, we use *asymptotic notation* when discussing time and space complexities, ignoring any constant factors (i.e. factors that don't change when the size of the input changes) and lower-order terms. For example, if an algorithm takes at most $\frac{n^2-n}{2}$ steps and uses $2n$ units of space to solve a problem, we say that the time complexity is $O(n^2)$, since we ignore the constant factor of $\frac{1}{2}$ and the lower-order term $-n$, and the space complexity is $O(n)$, since we ignore the constant factor of 2.

Chapter 2

Results

In this work, we catalog the space complexities of over 800 algorithms, 638 of which we derived ourselves, since they were not derived in the literature. In this chapter, we examine the best algorithms' time and space complexity for each problem family and note the differences between those two distributions as a whole in Section 2.1. We also compare the best space complexity and best time complexity of each problem individually in Section 2.3.

Most notably, we calculate the annualized rates of improvement in space complexity for each problem family at multiple input sizes (10^3, 10^6, and 10^9) in Section 2.2. We compare these rates with the rates of improvement of DRAM access speed and DRAM capacity, drawing conclusions about whether algorithmic space complexity improvement or these hardware improvements are more important in computational speedup.

We also explore the idea of time-space tradeoffs for problem families in Section 2.4, looking at how many problem families have such tradeoffs among their best algorithms over the years. We can also examine how a problem families tradeoffs have evolved over time, visualizing the Pareto frontier of time complexity vs space complexity for each problem family.

Overall, we provide a comprehensive overview of the current best algorithms for computer science problems with respect to both time and space, and we highlight the progress made over time.

We examine the state of space analysis in literature, noting how many algorithms have space complexity analysis included in their original papers, as well as those where the analysis was done in later papers.

2.1 A Look at the Algorithm Research Landscape

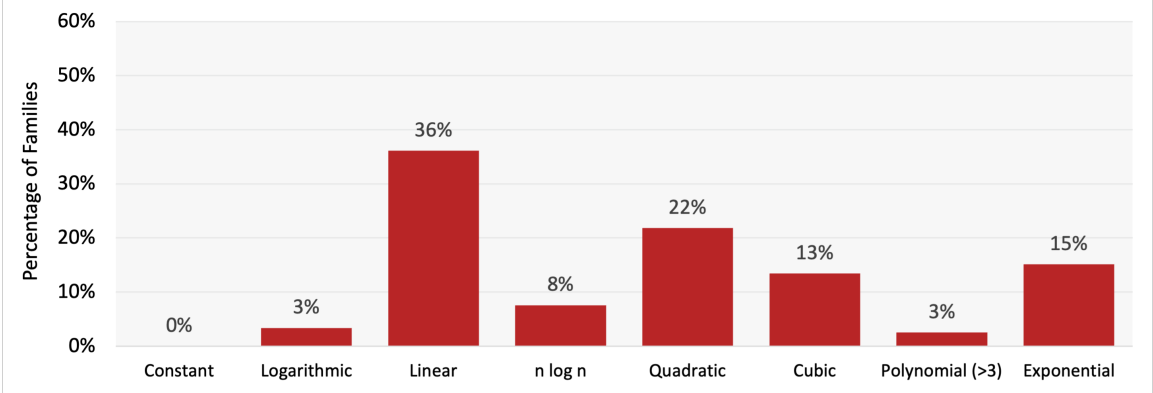
With the space complexities gathered of over 800 prominent algorithms in computer science, we analyze the overall space complexity landscape. For the 118[1] problem families that we have algorithms for, what is the distribution of best space complexities? Is that distribution similar to the distribution of best time complexities?

We show the distribution of the best time complexities of problem families in Figure 1a and the best space complexities in Figure 1b. Generally, we expect and see that the distribution of space complexities is concentrated towards lower asymptotic complexities, more so than are the time complexities. The space complexity of an algorithm is often upper bounded by the time complexity because whenever something is stored, it takes at least one time step to store it[2]. For example, if we need to store a size $O(n)$ array, we need to perform at least $O(n)$ steps–at least one step for each element. All of these storage time steps count towards the time complexity of the algorithm.

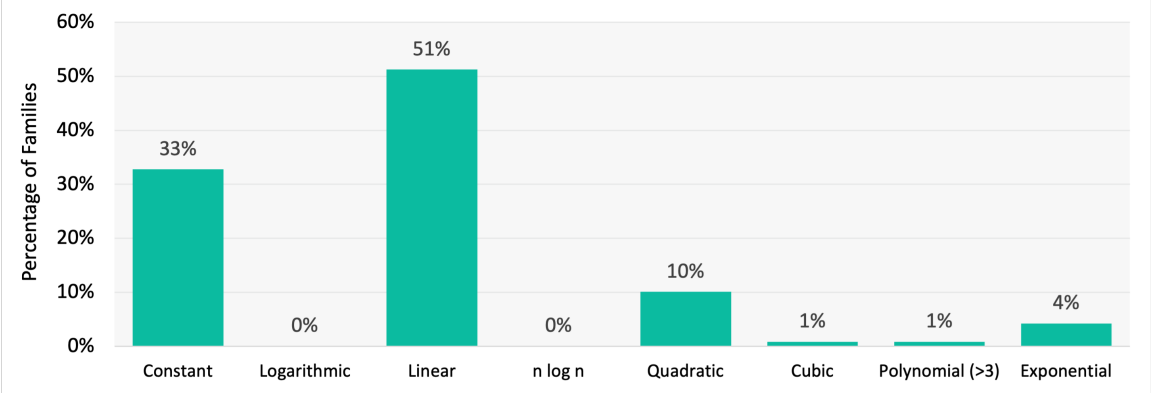
Interestingly, though, we do not see any problems in which the best space complexity is logarithmic or $O(n \log^c n)$ for any constant $c > 0$, which is not the case for time complexity. This may be a result of the model in which we are considering being the word RAM model with $O(\log n)$-size words, since then there are "bit tricks" to store logarithmic amounts of data into some constant number of words [2]. Even so, in all of these algorithms, we simply do not see any data structures that dominate an algorithm's space complexity that have a logarithmic (or poly-logarithmic) size compared to the input size. It is, however, very common to see problems where the

[1]Sherry and Thompson [10] and Liu [7] include 113 problem families. We extend this by 5 by digging deeper into literature and resolving some classification ambiguities.

[2]Some algorithms actually need to allocate a larger segment of memory than is actually written to, in which case it is possible for space complexity to actually be larger than time complexity.



(a) Problem families' current best time complexity



(b) Problem families' current best space complexity

Figure 1: Overview of Algorithm Landscape

dominating data structure used is linear or polynomial in the size of the input. It is also interesting to note how many problems have constant and linear best auxiliary space, meaning that for many of these problems, we are either at or very close to the best asymptotic auxiliary space complexity possible.

When discussing the state of algorithms, and in particular the space complexity of algorithms, it is also interesting to note how rarely space complexity is derived in the literature. Figure 2 shows us that many algorithm papers in the past did not include any sort of space complexity analysis, which isn't surprising because the notion was only introduced in 1965. The percentage of algorithm papers that include space analysis is increasing, growing by 3.6% per decade. A plausible explanation for this is that space complexity is becoming more important as the size of real-world inputs grows.

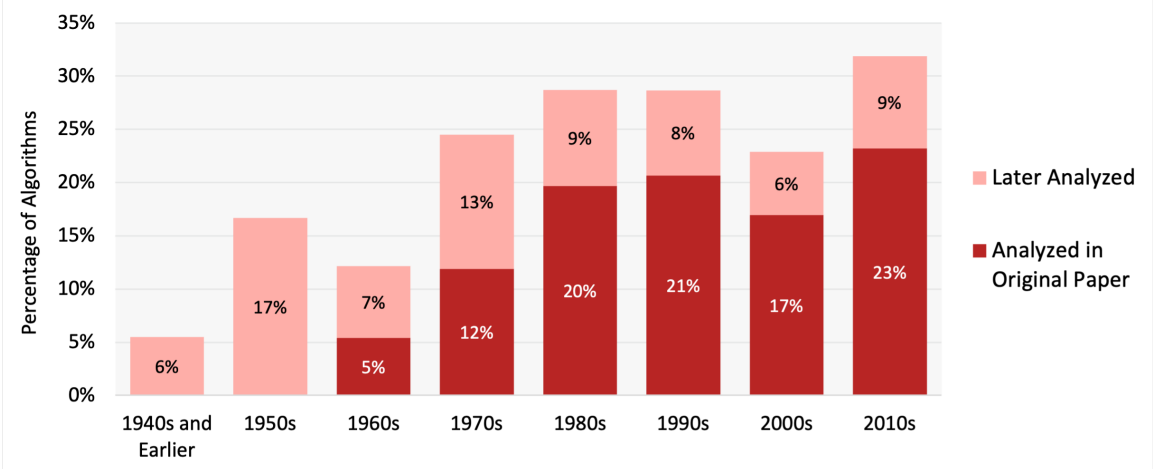


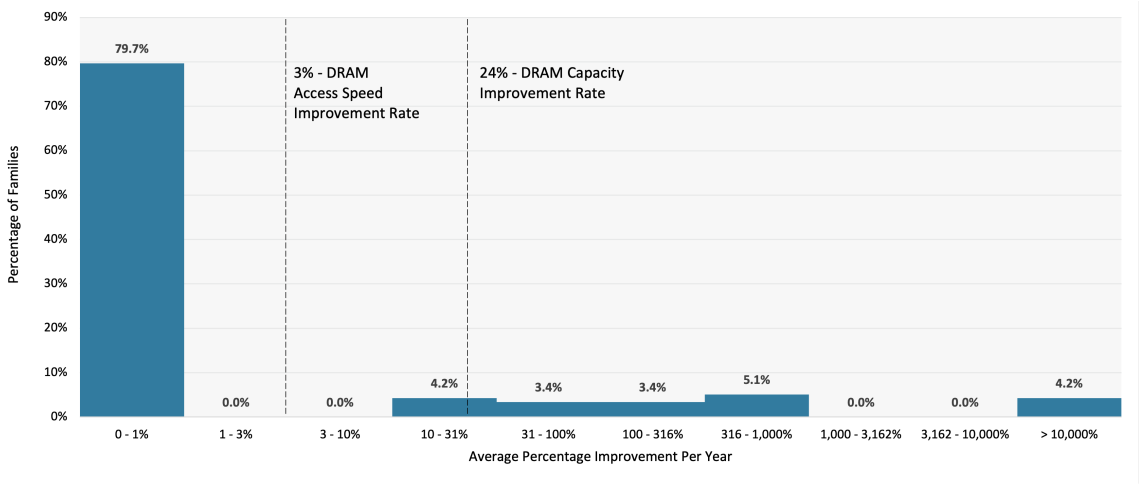
Figure 2: Percentage of algorithms published in each decade with space analysis in the original paper or later.

2.2 The Rate of Space Complexity Improvement

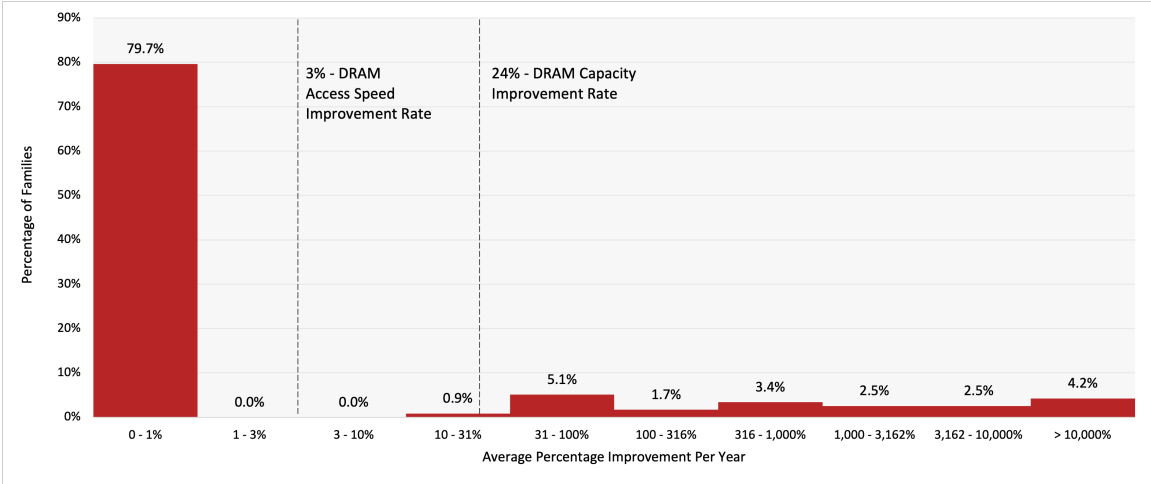
To analyze the improvement rate of space complexity, we look at the first algorithm published in an problem family, and for each subsequent algorithm, we note whether it improves the problem's best space complexity or not. If it is an improvement, we calculate the annualized rate of improvement from the previous best space algorithm to the improving algorithm. At the end, we take the average of these annualized rates of improvement for each of a problem's space improvements. This average annualized improvement rate is what is plotted in Figure 3, for multiple different input sizes.

The vast majority of problems saw no asymptotic improvements in space complexity. However, all of the problem families with space improvements improved faster than DRAM access speed has, and most of them have improved faster than DRAM capacity has as well. In scenarios where all of your data can be stored in memory, then improvements in space complexity may allow you to move more of your data accesses further up the memory hierarchy (e.g. from the L2 to L1 cache), giving larger computational speedups than simply increasing DRAM access speed. This is particularly relevant for the problems that see space complexity improve faster than DRAM access speed. For the problems that improve space complexity faster than DRAM capacity, these space complexity improvements are more important than the

(a) Problem size: $n = 1$ thousand (10^3)



(b) Problem size: $n = 1$ million (10^6)



(c) Problem size: $n = 1$ billion (10^9)

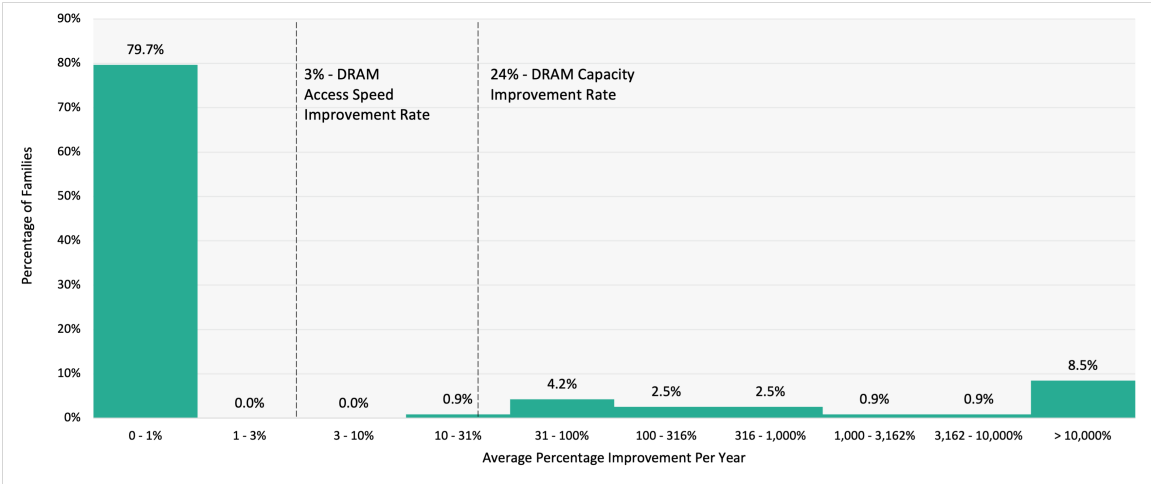


Figure 3: Distribution of average yearly improvement rates for 118 problem families, as calculated based on asymptotic space complexity, for problems of size: (a) $n = 1$ thousand, (b) $n = 1$ million, and (c) $n = 1$ billion. The DRAM access speed and DRAM capacity yearly improvement rates were calculated using DRAM data from Hennessy and Patterson [5].

DRAM capacity improvements when considering if we can even store all of the data for a problem in memory rather than having to store some on disk.

Furthermore, the improvement rate of space complexity is generally less than the improvement rate of time complexity found in [10].

2.3 Comparing Individual Problems' Space and Time Complexities

In addition to looking at the broad distributions of problems' best space and time complexities like in Figure 1, we can look at which specific space and time complexities occur together. Figure 4 is a heat map of the best space and time complexities for each problem family.

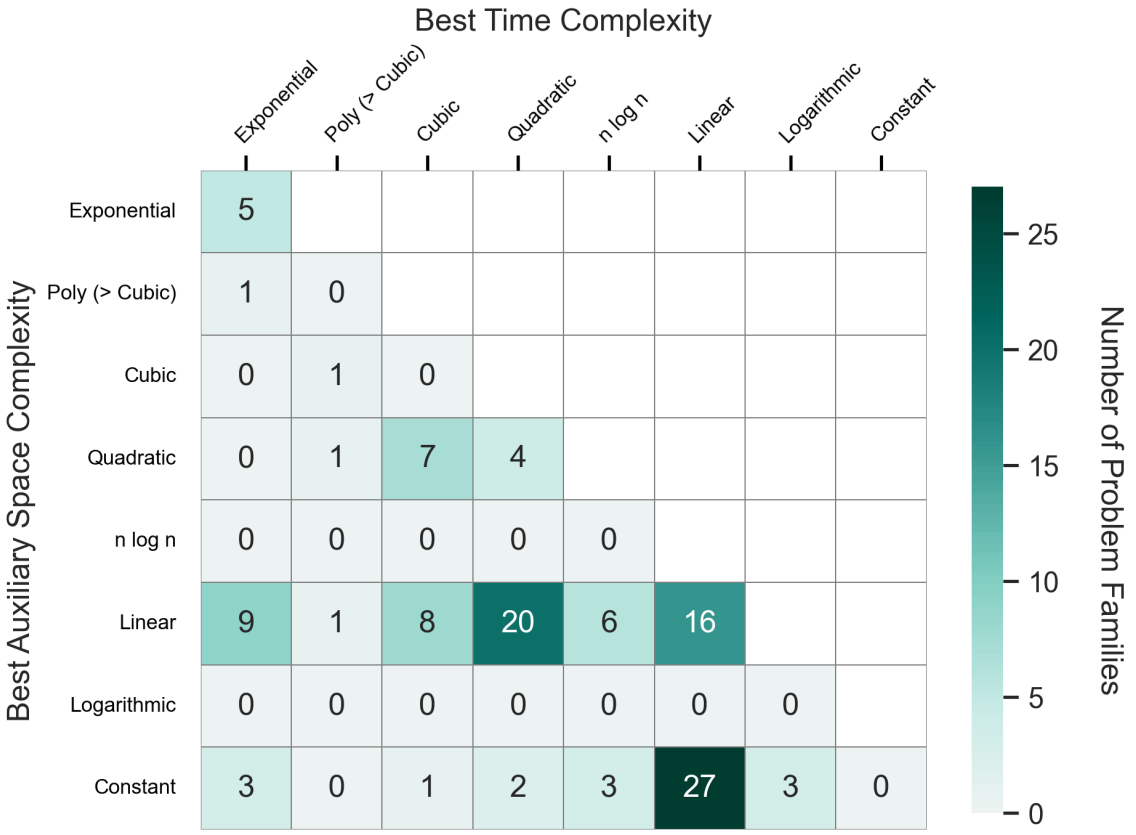


Figure 4: Heat map of the distribution of the best space and best time complexities for 118 problem families.

Of note, like mentioned in Section 2.1, we see that the two rows for “logarithmic" space and “$n \log n$" space are empty, meaning that no problem families have either of those as their best space complexity. In terms of bits (rather than words), these rows correspond to $O(\log^c n)$ bits and $O(n \log^c n)$ bits where $c > 1$, which are complexities we also rarely see in time complexity. We can also see that there tends to be more constant and linear space families as the best time complexity gets smaller, which makes sense because algorithms with smaller time complexities tend to optimize performance by minimizing unnecessary memory usage and utilizing simple data structures or even modifying the input in-place. It is interesting to note that the two largest clusters (27 and 20) are cases where the time complexity is a linear factor larger than the space complexity.

Some more interesting cases are the 10 families that we see in the top left corner, not in the diagonal (the squares with values 1, 1, 1, 7). These families have best space complexity that is asymptotically less than their best time complexity rather than equal, while also having best space complexity that is greater than linear in their problem size. In other words, they have slightly better space complexity than their upper bound (equal to their time complexity), yet they are still very space-inefficient. These are likely to be problems that we could see improvements in space in the near future, as well as all of the other problems with best auxiliary space greater than linear. For a list of all of these, refer to Appendix A.

2.4 Time-Space Tradeoffs

When choosing an algorithm to solve a given problem, the obvious choice is to choose the algorithm with the best asymptotic time and space complexity. However, some problems may not have one single “best" algorithm for both. Instead, there may be multiple algorithms with a tradeoff in time and space complexity. Thus, there is a Pareto frontier of algorithms that are all reasonable, depending on how much you care about space efficiency or speed.

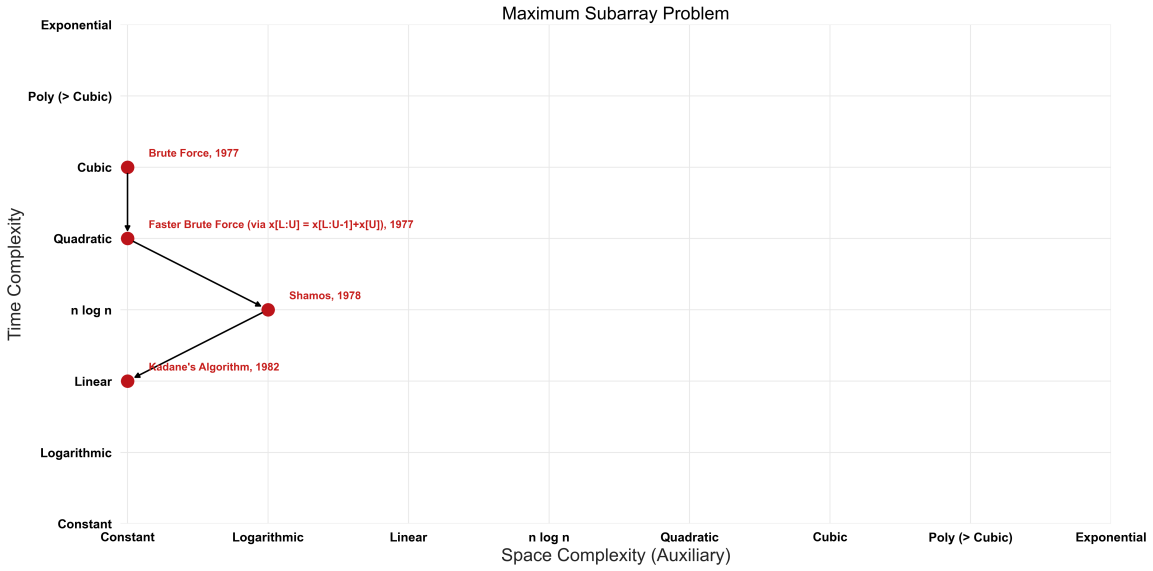
We can visualize the progress of specific problem families over time, like for the

Maximum Subarray Problem family in Figure 5a. The figures we produce like this include all of the algorithms that, at some point, were on the Pareto frontier for the problem family. This figure shows that there was no tradeoff necessary until 1978, when Shamos' algorithm was published with a slightly worse auxiliary space complexity but better time complexity than the best algorithm at the time. Then, the tradeoff disappeared in 1982 with Kadane's algorithm, which is still the single optimal algorithm for this problem. For the rest of the 117 problem families' Pareto frontier graphs, they will be available on our Algorithm Wiki website[3].

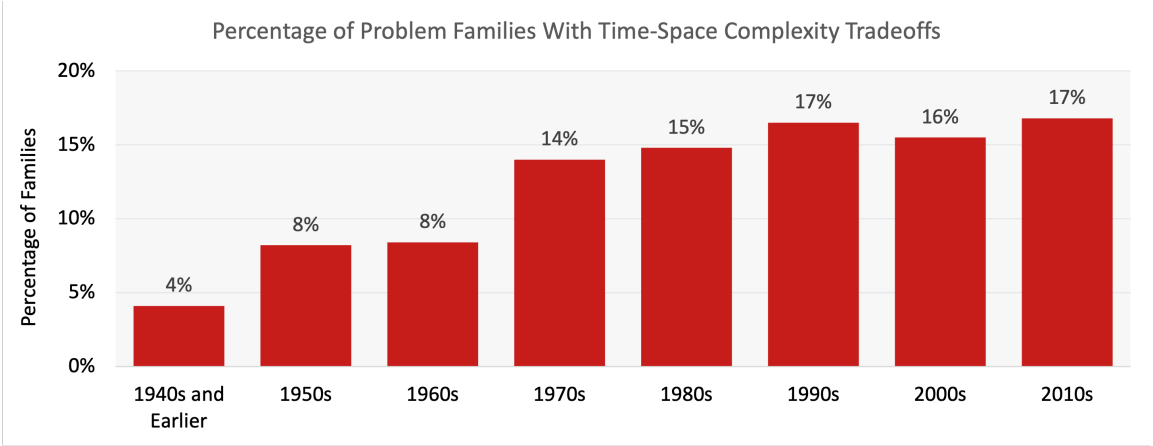
In our analysis, we find that currently 17% of the problem families in our database have time-space tradeoffs. The rest of the problem families have a single algorithm(s) that has a better (or equal) time and space complexity than the rest of the algorithms for that problem.

We also observed that over time, the percentage of problem families with such a tradeoff has been increasing by 1.79 per decade, as new algorithms are published. This historical trend is shown in Figure 5b. This is likely because as algorithm researchers have been working on improving the best space complexity and time complexity for problems, they may find that while it is quite difficult to improve both with a single algorithm, they may be able to improve one of the measures at the expense of the other. In general, though, time-space tradeoffs may exist because for many problems there are multiple natural ways to solve them, and these different strategies may lend themselves to being more efficient in one respect and less so in the other.

[3]https://algorithm-wiki.csail.mit.edu



(a)



(b)

Figure 5: (a) The evolution of the Pareto frontier of time-space tradeoffs for the Maximum Subarray Problem family. (b) The percentage of problem families in each decade that have a time-space tradeoff between algorithms on the Pareto frontier.

Chapter 3

Methods

3.1 Gathering Algorithms

Sherry and Thompson [10] gathered a number of algorithms from 57 textbooks and more than 1,137 research papers. For our analysis here, we omit approximation algorithms, quantum algorithms, parallel algorithms, distributed algorithms, and algorithms for inexact problems (see Section 1.2). We omit these algorithms because they are analyzed with different metrics and they don't provide exact like-to-like comparisons with exact algorithms for exact problems. For example, with quantum algorithms, it would not be particularly useful to treat the number of qubits needed for an algorithm the same way as we treat number of bits for a normal algorithm, especially because of the enormous difference in hardware cost and practicality of making a larger quantum computer versus simply buying more storage for a computer. In addition to the aforementioned algorithm types, we omit a few algorithms which appear in the database yet which have no reference listed and that we could not find with a literature search.

3.2 Grouping Algorithms

Similar to [10], the primary grouping of algorithms is into *problem families*, meaning algorithms that solve the same problem. Different algorithms may be most useful for

different variations of a problem.

For example, all-pairs shortest path (APSP) algorithms have variations for directed or undirected graphs, weighted or unweighted graphs, and dense or sparse graphs. These variations matter because they can have non-trivial differences in time complexities for their best-known algorithms. For example, the best runtime for dense, directed, unweighted graphs is $\tilde{O}(V^{2.575})$, but the best runtime for dense, directed, weighted graphs is $V^3/2^{\Omega(\log V)^{0.5}}$ [15, 13]. Much of the time, however, when applying the different algorithms in a problem family to the different variations of the problem at hand, the space and time complexities remain the same with respect to one another.

3.3 Model of Computation

The majority of algorithms analyzed use either the *word RAM* model of computation or the *real RAM* model. The word RAM model is a random-access machine (RAM) that operates on words, or groups, of bits [4]. These words are typically of size $O(\log n)$. The space complexity in this model is typically measured in number of words rather than bits. The real RAM model operates on real numbers exactly instead of using floating point numbers [9]. In the real RAM model, each memory cell contains a real number of any size, with exact precision [9]. We choose these models of computation for analysis because they accurately reflect how many modern computers work, with bitwise operations performed on words being treated as constant time.

It is worth noting that in the Word RAM model, when we say *constant*–or $O(1)$–auxiliary space, this means a constant number of $\log n$-sized words. Thus, this class of constant auxiliary space algorithms using Word RAM is comparable to the complexity class L (logspace). A problem is in L if it is computable by a *log-space transducer*, which is a Turing machine with a read-only input tape, a read/write work tape with size bounded by $O(\log n)$ symbols, and a write-only, write-once output tape [11]. With the Word RAM model, if we can use $O(1)$ auxiliary words which are each $O(\log n)$ size, then counting the symbols instead of words gives us $O(\log n)$ symbols on the

work tape (perhaps with some additional overhead for converting from RAM to TM).

A few algorithms' analysis uses other models of computation, such as Turing machines. We note the model of computation for each algorithm in our data tables.

3.4 Auxiliary Space Complexity

There are multiple different space complexities, or memory requirements, that an algorithm has. These include: input space complexity, output space complexity, auxiliary space complexity, and total space complexity. Input space complexity is the size of the input (typically, the other space complexity measures are with respect to this). Output space complexity is the size of the output. Auxiliary space complexity is the amount of space required during the computation that excludes the input and output. For example, if an algorithm requires constructing a matrix that is not part of the output, then the matrix would count towards the auxiliary space. The total space complexity is the sum of the other three. There may be some exotic models of computation that also have other measures, but we do not consider those.

When discussing space complexity of algorithms, we decided that we will consider only the *auxiliary space* rather than the total space. When considering total space complexity, this can be much larger than the input space or output space; however, there would be a trivial lower bound equal to the max of the input space and the output space. If we consider only the auxiliary space, we can get around this lower bound to give us a more fine-grained view of the space that is actually needed. Furthermore, all algorithms for the same problem will have the same input and output space complexities, since by definition, they are taking the same input to the same output as each other. Thus, focusing on auxiliary space complexity allows us to more easily distinguish space-efficient algorithms from space-inefficient algorithms.

3.5 Classifying Asymptotic Complexities

In the previous work of Sherry and Thompson [10], they grouped asymptotic time complexities in order to analyze the improvements going from one time complexity to another. We do the same with space complexity. Let us define a notion of *complexity classifications*, which are the same classifications used in [10]. We have eight different classifications that we classify algorithms' time and space complexity as. In order, these are:

1. Constant – $O(1)$
2. $\log n$ – $O(\log^c n)$ for some constant $c > 0$
3. Linear – $O(n)$
4. $n \log n$ – $O(n \log^c n)$ for some constant $c > 0$
5. Quadratic – $O(n^2)$
6. Cubic – $O(n^3)$
7. Polynomial ($>$ cubic) – $O(n^c)$ for some constant $c > 3$; also, super-polynomial but sub-exponential
8. Exponential/factorial – $O(c^n)$ for some constant $c > 1$ or $O(n!)$

Furthermore, we also consider intermediate values between these classifications. In Sherry and Thompson's paper [10], they round up these intermediate classifications. For example, if we have an algorithm that runs in $O(n^{2.5})$ time, we would classify this as 5.5 since it is between 5 (quadratic) and 6 (cubic), so they would round this up to cubic. However, for Figures 5b, 5a, we use these intermediate classifications in order to more accurately reflect real asymptotic improvements, rather than only counting tradeoffs or improvements when we see a difference between the classifications that get rounded up to one of the 8 classifications listed above.

3.5.1 Multiple Parameters

Some problems have only one main parameter n that is the focus of the time and space complexities. However, there are other problems where there are multiple parameters that show up in these complexities. For these, we still need a single overall classification for the algorithms that solve these problems. To do this we determine a single asymptotic formula to use as the basis of the classification, which is typically the input size. We do not use this single asymptotic when calculating the actual rates of improvement, though (See Section 3.6).

For the majority of the problems the formula is just a single variable (e.g. n). For matrix problems with n and m being the number of rows and columns respectively, this formula is n^2 for square matrices or mn for rectangular matrices. For graph problems, we typically use the number of vertices V, and we consider dense graphs, plugging in $O(V^2)$ for the number of edges E. Some problems have two separate parameters m, n where the input size is $m + n$, in which case complexities such as $O(mn)$ would be considered "quadratic" when compared to the input size $m + n$.

3.6 Rate of Improvement Calculation

We calculate the improvement rate of space complexity for problem families the same way as in Sherry and Thompson's work [10], except using space complexity instead of time complexity. An improvement from algorithm i to algorithm j with input size n is calculated as

$$\text{Improvement}_{i \to j}(n) = \frac{\text{Words}_i(n)}{\text{Words}_j(n)} \tag{3.1}$$

where n is the problem size and the number of words (units of memory in the Word RAM model) is calculated using the asymptotic auxiliary space complexity. For this work and for Sherry and Thompson's work, the problem sizes considered were $n = 10^3, 10^6, 10^9$.

Again, we use the same formula as in their work, except using space complexity,

to calculate the average per-year percentage improvement rate over t years:

$$\text{YearlyImprovement}_{i \to j}(n) = \left(\frac{\text{Words}_i(n)}{\text{Words}_j(n)} \right)^{1/t} - 1 \tag{3.2}$$

As in their paper, we also only consider years since 1940 to be those where an algorithm was eligible for improvement.

When calculating the rates using multiple parameters, we use the same ratios of parameters as in Sherry and Thompson's work [10] in order to best draw like-to-like comparisons.

Chapter 4

Future Work

In this work and in the previous work done by our lab [10, 7], there were a small number of algorithms in our database that the analysis leaves out. Namely, those are approximation algorithms, quantum algorithms, parallel algorithms, and all problem families that do not have an exact problem statement (i.e. problems without an exact solution). There is work currently planned to dig deeper into some of these types of algorithms–in particular approximation algorithms, which are often very popular choices in practice.

As in Liu [7], many of the exact problem families are beginning to have theoretically-optimal asymptotic time complexities. Since there is no more room for asymptotic gains for these problems, one approach to keep improving is to solve the problem using approximation algorithms. This opens the door to algorithms that have worst-case time complexities that are less than the theoretically-proven lower bound for exact algorithms for a given problem family.

In addition to lower bounds on the time complexity of algorithms there are also results lower bounding the space complexity for some problems and some results which give a lower bound on some function of both time and space together. We wish to do a survey of these complexity results so we can give a more robust characterization of how far these algorithms are from being optimal.

To back up the claims that the problem families studied in this paper as well as in [10, 7] are the important problems in computer science, we would like to look at

some other measures of importance of the problems. These other measures might include things such as popularity on Wikipedia and number of occurrences in open-source libraries. If we see plenty of overlap between the problem families used here and when using rankings based on these other measures, then we would be able to claim, with more conviction, that the results from this paper and from [10, 7] hold true for the field of computer science as a whole.

Furthermore, as an overarching goal of all of these projects, we wish to organize all computer science research in a manner that is easy to navigate. On our website, The Algorithm Wiki[1], we wish to make it easy for researchers to find open problems. It is currently cumbersome to go through all of the recent papers for a problem, making sure to look at all of the related works to find whether a problem is actually worth working on. We also would like to make it easy to compare different parameterizations of problems, enabling you to easily see whether different assumptions about the parameters for a problem lead to interesting observations.

[1] https://algorithm-wiki.csail.mit.edu

Chapter 5

Conclusion

In past work, the time complexity of algorithms has been studied intensively to measure the speed of computational progress. In this work, we focus on space complexity, as it has been widely neglected in much of algorithm literature in the past. As part of this work, we have gathered the space complexities of over 800 algorithms, of which the vast majority were derived by myself, Jeffrey Li, Chirag Falor, Khaleel al-Adhami, and Jayson Lynch since only a small portion had space complexities that we were able to find in the literature.[1]

Having the space complexity of algorithms readily available via our Algorithm Wiki website[2] will help both computer scientists both in theory and in application. This work should help guide theorists towards problem families which have algorithms with high space complexities so that they may attempt to come up with more space efficient algorithms with better or equal time complexity. It should help computer scientists implementing these algorithms by supplying them with more information that they may use to help decide which algorithm is the most appropriate to solve the specific problem they are working on. This is especially helpful for the problems in which we have found that there are time-space tradeoffs among different algorithms, rather than one single algorithm being optimal for the problem.

Additionally, we showed that to speedup the computation of algorithms from

[1]There is a separate document that contains our notes deriving the space complexities, which will be produced along with another paper that will stem from this paper.

[2]https://algorithm-wiki.csail.mit.edu

the standpoint of having faster data accesses, algorithmic improvements to space complexity can be far more influential than hardware improvements, such as DRAM access speed and DRAM capacity improvements. However, for the vast majority of problems, space complexity has not seen any improvements because either the original algorithms were already space-efficient or it is simply difficult for algorithm researchers to improve on the space complexity. It may also be that researchers simply care much less about improving space complexities than progressing other areas of algorithmic research.

Appendix A

Problem Families with Superlinear Space

The following problem families have no algorithms in our database with auxiliary space complexity that is less than or equal to linear, and so these seem likely to have large potential improvements to their space complexity. They are listed organized by their most space-efficient algorithm's auxiliary space complexity classification. The specific definitions of these problems as well as all of the other problem families we studied can be found on our Algorithm Wiki website.

- Quadratic auxiliary space
  - Maximum Flow
  - All-Pairs Shortest Paths (APSP)
  - Integer Relation
  - Maximum-Weight Matching
  - Constructing Eulerian Trails in a Graph
  - Poisson Problem
  - CFG Problems
  - Finding Frequent Itemsets

  - Graph Isomorphism Problem
  - Determinant of Matrices with Integer Entries
  - Transitive Reduction Problem
  - Secret Sharing
- Cubic auxiliary space
  - Longest Path Problem
- Polynomial (> 3) auxiliary space
  - Traveling-Salesman Problem (TSP)
- Exponential auxiliary space
  - Informed Search
  - Gröbner Bases
  - Coset Enumeration
  - Dependency Inference Problem
  - Frequent Words with Mismatches Problem

Bibliography

[1] Alfred V. Aho and John E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1974.

[2] Ilya Baran, Erik D Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[4] Michael L Fredman and Dan E Willard. Blasting through the information theoretic barrier with fusion trees. In *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*, pages 1–7, 1990.

[5] John L. Hennessy and David A. Patterson. *Computer architecture: a quantitative approach*. Morgan Kaufmann, 6. ed edition, 2019.

[6] Charles E. Leiserson, Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. There's plenty of room at the top: What will drive computer performance after moore's law? *Science*, 368(6495):eaam9744, 2020.

[7] Emily Liu. A metastudy of algorithm lower bounds. Master's thesis, Massachusetts Institute of Technology, 2021.

[8] Pascal Michel. A survey of space complexity. *Theoretical Computer Science*, 101(1):99–132, 1992.

[9] Michael Ian Shamos. *Computational geometry.* Yale University, 1978.

[10] Yash Sherry and Neil C. Thompson. How fast do algorithms improve? [point of view]. *Proceedings of the IEEE*, 109(11):1768–1777, 2021.

[11] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[12] R. E. Stearns, J. Hartmanis, and P. M. Lewis. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 179–190, 1965.

[13] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 664–673, New York, NY, USA, 2014. Association for Computing Machinery.

[14] Wm A Wulf and Sally A McKee. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH computer architecture news*, 23(1):20–24, 1995.

[15] Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, may 2002.