The Educational Insights and Opportunities Afforded by the Nuances of Prim's and Kruskal's MST Algorithms

Ali Erkan
Ithaca College
Ithaca, NY, USA
aerkan@ithaca.edu

ABSTRACT

The computation of a minimum spanning tree (MST) is a fundamental topic in any algorithms course. In this paper, we outline a series of projects based on a thorough exploration of the performances of two well-known MST algorithms: Prim's and Kruskal's. For a graph of n nodes and e edges, both run in $O(e \lg n)$ time but these results are based on picking the right data structures. Prim's relies on a priority queue that supports priority modification in $O(\lg n)$ time while Kruskal's relies on a disjoint-set that supports find/union operations in $O(\lg n)$ time. The performance ramifications of using simpler data structures allow students to thoroughly understand the operations of these two algorithms, get a deeper comprehension of asymptotic complexity analysis, negotiate empirical results with analytical predictions, and illustrate the value of "programming to the interface". Our study also includes the performance consequences of the qualities of the graphs on which MSTs are computed: structure (random vs scale-free), density (sparse vs dense), and edge cost distribution. Finally, as an additional inquiry to engage students in a somewhat unexplored direction, we focus on the average pairwise-distance of the MSTs produced by Prim's vs Kruskal's.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; • **Theory of computation** → **Graph algorithms analysis**; **Data structures design and analysis**; *Shortest paths*; • **Software and its engineering** → **Abstract data types**;

KEYWORDS

Minimum Spanning Trees, Prim's algorithm, Kruskal's algorithm, Data Structures, Empirical results, Asymptotic complexity

ACM Reference Format:
Ali Erkan. 2018. The Educational Insights and Opportunities Afforded by the Nuances of Prim's and Kruskal's MST Algorithms. In *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3197091.3197129

ITiCSE'18, July 2–4, 2018, Larnaca, Cyprus

ACM ISBN 978-1-4503-5707-4/18/07…$15.00
https://doi.org/10.1145/3197091.3197129

1 INTRODUCTION

This paper is based on the academic value of an extensive evaluation of the two most well-known solutions to the minimum spanning tree (MST) problem in a conventional "Algorithms" or "Data Structures and Algorithms" course. Our work is based on engaging students with an empirical performance comparison of Prim's and Kruskal's MST algorithms (referred to simply as "Prim's" and "Kruskal's" from here on) so that students (i) witness a significant validation of asymptotic complexity analysis, (ii) understand aspects of performance that are filtered away when using an analytical lens, (iii) advance their empirical experimentation skills, and (iv) engage in a modest yet non-trivial guided inquiry on a characterization of the resulting MSTs (average pairwise distance). This Prim's/Kruskal's comparison takes the form of a culminating project at the end of the semester but students are continuously engaged in weekly self-contained scaffolding labs to construct and test building blocks to be eventually integrated for the study. Our approach has been refined over a two-year period at two institutions and has helped students acquire a deeper and more holistic comprehension of algorithmic concepts and performance analysis. The primary institution is the author's academic home while the secondary institution is an R1 university. In both contexts, class size was capped at 25 and incoming students had thorough knowledge of linear data structures as well as binary trees.

The importance of empirical analysis in Computer Science education has been an ongoing thread during the past 15 years [2, 5, 8]. Our pedagogical philosophy, on the other hand, is rooted in *inductive learning* as it provides operational specifics for students to construct their own comprehension and high-level principles [4]. In particular, given the conventional challenges associated with teaching algorithms, we are trying to make a non-trivial portion of the course to be *learner-centered* where students take responsibility for building their own version of reality. Our attempts to have students negotiate different forms of assessment (such as asymptotic vs empirical) is based on *inquiry-based learning* since they actively participate in the formulation of the main questions, the explanation of their own observations, and the hypothesizing of resolutions to negotiate (seemingly) conflicting results [1]. According to the categories provided by Staver and Bay [7], we also follow *structured inquiry* where students are given a problem as well as the outline of a potential solution for them to complete the missing pieces.

The use of programming is a substantial intervention in an Algorithms course so we start by providing a justification for it. We then move to an outline of the project and show the anticipated results of a Prim's/Kruskal's performance comparison as well as the structural differences to be discovered in the MSTs they construct. The paper concludes with a few self-reflective remarks.

2 JUSTIFICATION

For a set N of nodes and a set E of undirected edges over those nodes that carry positive weights/costs, a MST for a graph $G = (N, E)$ is a selection of $|N| - 1$ edges in the configuration of a tree that directly/indirectly makes every node reachable from every other node in the cheapest possible way for the whole graph. Prim's is a greedy algorithm that selects these edges by a single "wave" of consideration starting from a designated node. Kruskal's is another greedy algorithm that independently grows many minimum spanning "saplings" until they all converge into a single tree. Prim's asymptotic complexity is $O(|E| \lg |N|)$ since it has to do a $O(\lg |N|)$-cost lookup for each of the $|E|$ edges. Kruskal's asymptotic complexity is $O(|E| \lg |N|)$ since it has to do a $O(\lg |N|)$-cost disjoint-set union/find operation on $|E|$ edges. The $O(|E| \lg |E|)$-cost sorting of the edges required by Kruskal's does not make an asymptotic complexity difference because $|E| \leq |N|^2$ and thus $\lg |E| \leq \lg |N|^2 = 2 \lg |N| = O(\lg |N|)$. Therefore, our expectations are that if we are to see an empirical performance difference, it might be with Prim's over Kruskal's in the context of dense graphs [3, 6].

Once the graph data structure is covered and greedy algorithms are outlined, pseudocode comprehension and complexity analysis of both algorithms typically take one lecture; proof of correctness (if also done) requires another. So what would motivate us to "stretch" this comparison to a semester-long theme? We believe that if the asymptotic complexity analysis of these two algorithms is the tip of the learning iceberg, then the educational value of their thorough evaluation corresponds to the submerged part due to comprehension opportunities that go beyond MSTs. The formation of this hypothesis[1] was based on a multi-year period of observation and self-reflection on our own Algorithms course that resulted in three tiers of justification for change:

• The primary and most overarching one is based on whether there should even be a programming component in an algorithms course. It can be said that while programming is the central tool of our discipline, the details that unavoidably come with it pollute the elegance of algorithmic ideas. This is a timeless invariant within CS education, one in which the author wholeheartedly believes. However, it hinges on a hidden assumption: what might be a "detail" to a computer scientist may not yet be a detail to a computer science *student*. When compared to the CS curricula designed to serve simpler computational landscapes of previous decades, we are now navigating our students through a more technologically-complex academic and professional world. For example, web development and mobile development (neither of which is simple enough to be sufficiently addressed in one or two courses) are now seen as parts of basic competence. Since the number of credits to major in CS has remained somewhat constant, students are more likely to have seen a wider range of topics just once than a more limited range of topics multiple times by the time they enroll in an algorithms course. That is, they may not yet have had sufficient repetition of central ideas to be able to distinguish the fundamental from the detail. Therefore, instead of clouding elegance, programming (in the context of algorithms course) could be helping to develop skills to *recognize* "detail" and to *reiterate* the fundamental.

• The secondary justification is based on the academic value of empirical analysis. In the context of an algorithms course, time becomes an abstraction based on counting the number of times the most frequently encountered atomic operation is executed; runtime complexity is then expressed as a mathematical way to produce this number as a function of input size. At a time when a single-semester discrete math course is potentially the sole mathematical preparation for many students, the abstract notions of $O(N)$, $\Theta(N)$, $\Omega(N)$ as well as their Θ-exclusive versions of $o(N)$ and $\omega(N)$ take time to digest. Empirical justification of performance based on the conventional notion of time helps students with this abstraction. Second, for students who do not take a research-methods course (a group not trivial in numbers), the basics of empirical analysis is a necessary area of growth since analytical methods cannot be employed on all real-world systems. Third, empirical analysis allows students to get a wider sense of average case behavior, a perspective that asymptotic complexity affords only in a few select cases (e.g. binary search, quick-sort). Finally, empirical analysis allows us to contextualize concepts that students may otherwise remember only as definitions from a discrete math course: independent variable (factors that influence runtime duration), dependent variable (time), Cartesian product (all possible independent variable value combinations as performance impactful subsets), etc.

• The third justification has to do with choosing the MST problem (among other options) as the basis of *the* programming thread to interweave throughout an algorithms course. The MST problem is naturally associated with a multiplicity of algorithms. Prim's, being the asymptotically favored one, can either be implemented with a simple binary heap (albeit in $\omega(|E| \lg |N|)$ time for non-sparse graphs) or with one that supports the more non-trivial '`decreaseKey()`' operation for a uniform $O(|E| \lg |N|)$ performance. The disjoint-set used by Kruskal's, on the other hand, can be based on either an array (with a costly '`union()`' operation) or a tree (with an amortized logarithmic cost for '`union()`') with performance ramifications. The graph data structure used by both algorithms can utilize array- or linked-lists for maintaining adjacency lists. Finally, assessing average pairwise distance of the resulting MST (a factor that's not typically covered in theoretical treatments of the topic) creates a natural use of Floyd's algorithm; this gives us a conceptual thread that starts with the basics of the graph data structure and spans all the way to dynamic programming. MSTs also have a wealth of applications to motivate students. These include printed circuit board design, cancer imaging, large-scale cosmological structure identification, feature extraction from remotely-sensed images, weather data interpretation, clustering, approximation algorithms for NP problems in addition to the more frequently cited uses such as route discovery and cycle avoidance in network design[2].

3 PROJECT DESCRIPTION

As implied earlier, there are two components to the overall structure of the project. First, there is the series of weekly labs designed to guide students build components to be (eventually) integrated to conduct the empirical study; these include the implementation

[1] We use the word "hypothesis" simply because small class size and the inevitable curricular refinements consequential to other changes in the major make it very difficult to conduct a long-term investigation expected of robust large-scale learning-science experiments.

[2] Geometry in Action: http://www.ics.uci.edu/~eppstein/gina/mst.html

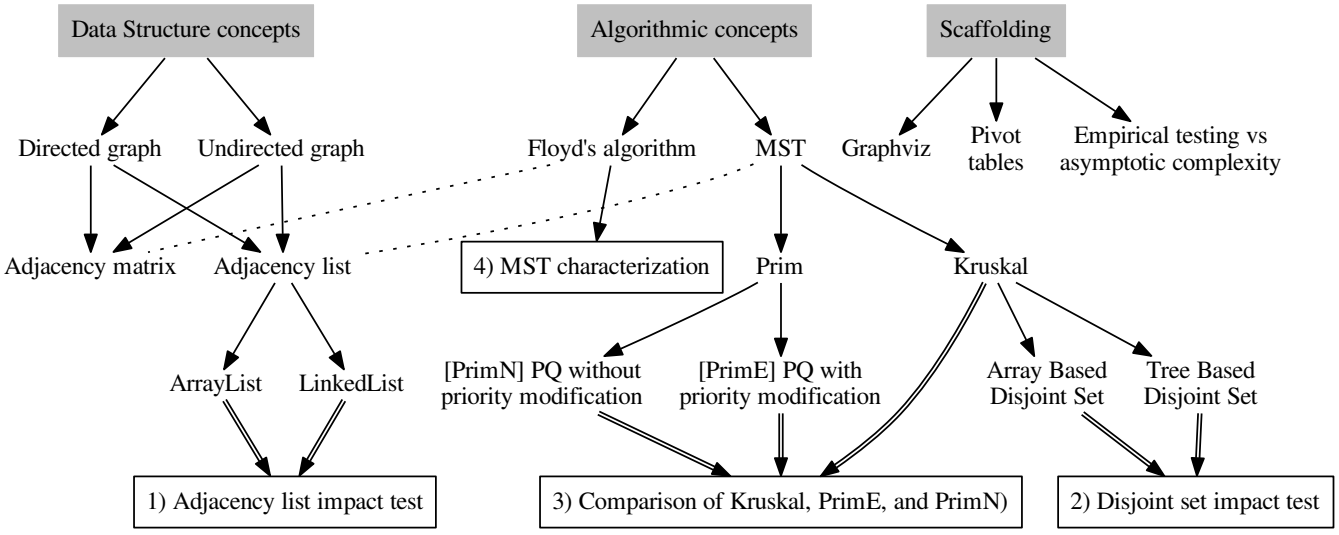


Figure 1: A big-picture view of the MST project: The underlying data-structures are on the left while the associated algorithms are on the center and right. Single-line arrows represent topic hierarchies (which imply dependencies) and double-line arrows to the boxes represent a subset of the performance comparisons tests done throughout the semester (the numbers loosely indicating their order). The dotted lines associate the two central algorithms (minimum spanning trees and all-to-all minimum edge costs) with the graph data structures with which they naturally couple. Finally, three scaffolding topics are shown in the top-right because in addition to being associated with three labs, they are extremely consequential in making this project beneficial, meaningful, and enjoyable for students.

and testing of (i) a binary heap, (ii) a disjoint-set, (iii) a directed graph, (iv) an undirected graph, (v) Kruskal's algorithm, (vi) Prim's without priority modification, and (vii) Prim's with priority modification[3]. Then there is the empirical study itself which typically occurs as an culmination exercise towards the end of semester. The primary deliverable we expect is a technical report that shows students' findings in which they identify their random variables, show their empirical results, outline how they negotiate these results with the predictions of asymptotic complexity analysis, and their characterization of the structure of the resulting MSTs.

It is important to note that labs are not readily declared to be pieces of the project. In fact, each is intentionally designed and framed as stand-alone entities to support the algorithmic topics of the corresponding week. By making sure that the weekly implementations produced by students '`implement`' interfaces issued by the instructor (in the literal programming sense), their eventual integration is ensured to be simple. This loose coupling between the building blocks and the final project is an important consideration because we want all students (even those who do not do well with a few of the labs) to be able to attempt the final project.

Figure 1 is a high-level view of the components threaded together throughout semester; due to space constraints, we focus only on a few of the pertinent components:

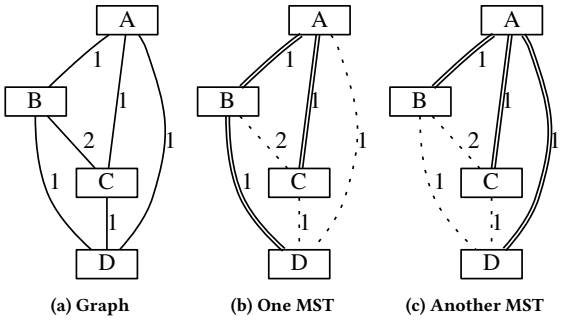
• Kruskal's Evaluation: Kruskal's operation requires the use of a disjoint-set to keep track of sets of node-subsets, each containing a "local" MST. There are two common implementations of disjoint-sets, each with different asymptotic complexity costs for set-membership tests and subset union operations. Since this is an under-the-hood matter, the implementation of Kruskal's doesn't change based on the data structure. But the performance ramifications can be mathematically expressed and empirically detected.

• Prim's Evaluation: Prim's operation requires the use of a priority queue in which "known" (but not "done") nodes are maintained; however, during the course of its traversal, encountering cheaper edges to known nodes requires the priority of enqueued elements to be modified. If this feature cannot be supported, then Prim's can be implemented in a simpler manner by maintaining edges (instead of nodes) in the queue. However, this modification not only increases Prim's time complexity, but also requires the algorithm to be implemented differently. In our project, we refer to the former as PrimN and the latter as PrimE since the letters 'N' and 'E' indicate which set of $G = (N, E)$ is being used to populate the priority queue. This also means that students end up comparing three MST algorithms: Kruskal, PrimE, and PrimN.

• Determination of Independent Variables: While graph size is the most relevant independent variable, out project also considers edge density (i.e. the average number of edges per node), edge cost distribution (i.e. the range of values from which edge costs are drawn), and graph structure (random graphs vs scale-free graphs, as explained below). This means that when coupled with algorithm choice, our exploration is over five independent variables.

• Determination of Dependent Variables: While time-to-completion is the most relevant variable, we also explore the diameter and the average pairwise distance of the MSTs since different

[3]In a semester arrangement of 14 weeks, these seven labs along with the three that we use for scaffolding matters leave four labs for other topics to be covered.



(a) Graph (b) One MST (c) Another MST

Figure 2: A simple undirected graph and two MSTs. Since the diameter of the second MST is less than the diameter of the first, the second is more compact and (potentially) "better".

	A	B	C	D	E	F	G	H	I	J	K	L
1	*SUM of t*				*ecl*							
2	*N*	*epn*	*A*	*G*	1	2	3	4	5	6	7	N
3	256	log2	K	g	0.56	0.51	0.5	0.49	0.49	0.48	0.5	0.59
4				gp	0.62	0.56	0.55	0.53	0.54	0.54	0.54	0.65
5				r	0.3	0.29	0.29	0.3	0.31	0.32	0.32	0.51
6			PE	g	0.03	0.32	0.35	0.35	0.36	0.36	0.38	0.59
7				gp	0.03	0.32	0.36	0.36	0.36	0.39	0.41	0.64
8				r	0.02	0.3	0.33	0.34	0.34	0.34	0.35	0.51
9			PN	g	0.14	0.12	0.13	0.15	0.17	0.19	0.21	0.59
10				gp	0.14	0.13	0.14	0.17	0.19	0.21	0.24	0.64
11				r	0.18	0.14	0.14	0.15	0.17	0.19	0.2	0.51

Figure 3: Pivot tables provide convenient ways to explore the interactions between subsets of independent variables and a single dependent variable of interest.

MSTs for a graph may differ in these measures[4]. For example, for the graph shown in figure 2a, the spanning trees in figures 2b in 2c are both minimal. However, while the MST in 2b has a diameter of 3 and an average pairwise distance of $\frac{1+1+1+2+2+3}{6} = \frac{10}{6} \approxeq 1.66$, the one in 2c has a diameter of 2 and an average pairwise distance of $\frac{1+1+1+2+2+2}{6} = \frac{9}{6} = 1.5$. For many applications, the latter one will be considered "better" due to its compactness.

• Use of Pivot Tables: One of the pitfalls of empirical data collection is the necessity of code instrumentation which runs the risk of obfuscating students' work, especially when there is more than one random variable. What we found to be an effective approach is to require students save independent-variable/dependent-variable tuples in a tab delineated text file which can be copied/pasted into a spreadsheet; this data can be conveniently analyzed with a pivot table from a variety of angles[5]. For n independent variables (i.e. the "dials") and a single dependent variable (i.e. the "measurement"), a pivot table can freeze any of the $\binom{n}{n-2}$ dial combinations and use the remaining two to define the rows and columns of a table to display the measurements. The real power of pivot tables comes from the fact that with just a few trivial steps, the data then can be sliced/agreegated/viewed in different arrangements so that the viewer can isolate the most impactful factors observed in the experiment. All of the results we show in section 4 were the results of pivot tables like the one we see in figure 3.

• Factoring in Graph Structure: Even though it is convenient to keep graphs as abstract entities in mathematical analysis, it is interesting to consider how particular structures may (or may not) influence the performance of the algorithms in empirical testing. In our project, we consider three particular structures. The first is constructed by first creating the node set and then by adding the desired number of edges between randomly chosen node pairs; this structure is known as a random graph in the literature. The second is constructed by starting with a small core and by adding nodes one by one, with each node connecting with a fixed number of existing nodes. This means that "old" nodes are more likely to have higher numbers of edges than "young" ones by virtue of having existed for a longer time. The third and final structure is the same as the second one with the difference that the likelihood of an existing node being selected as the destination of a new node edge is proportional to the existing node's edge count; this structure is known as a scale-free graph in the literature. In order to keep our terminology simple for students, we refer to the second and third structures as "grown" networks, disambiguation the two with the phrase "preferential edge selection" when needed.

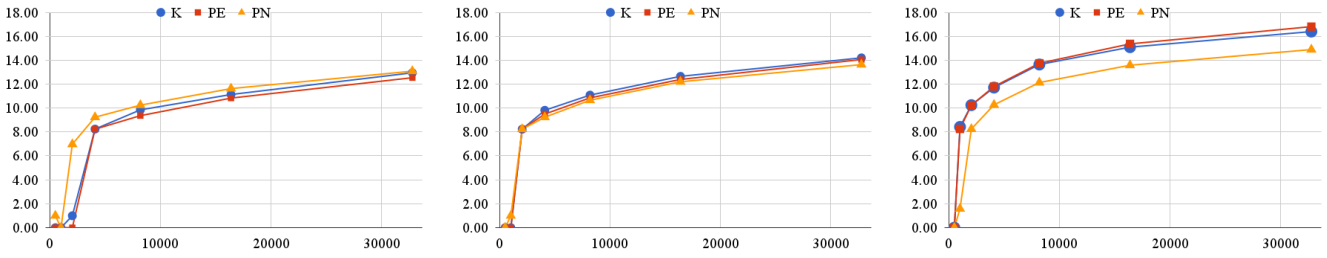
[4]The **diameter** of G is the maximum of the minimum path lengths between all $\binom{|N|}{2}$ possible node pairs while **average pairwise distance** is their average.
[5]The pivot table in Google Sheets works very well.

4 RESULTS

In the preceding sections, we outlined the reasons why we think this project is a worthwhile endeavor in an algorithms course. There is, however, one additional (though somewhat nebulous) factor that should not be forgotten in any consideration or adoption: the magnitude of the revelations drawn from these experiments. Since students will be asked to execute a semester-long plan, it is important for them to consider that the time and effort they spend on the project is worth it. We therefore share some of our findings so that readers can assess how much this can actually happen:

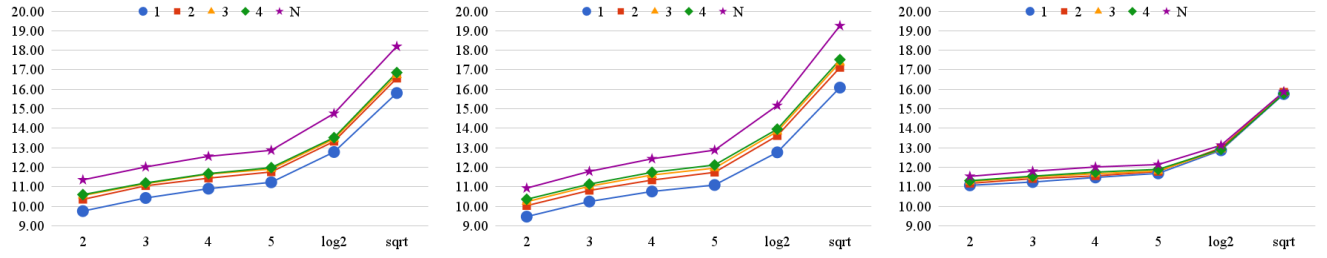
• Figure 4 compares the performance of Kruskal, PrimE, and PrimN as we scale up the sizes of the graphs; please refer to the figure caption for the values of remaining variables. Here we see that for sparse graphs, all three algorithms perform roughly at the same level. However, as we move to more dense graphs (as in, for example, when the number of edges per node is equal to the $\lg_2$ of the number of nodes), the superiority of PrimN becomes apparent by a few orders of magnitude. In the eyes of students, this is a key affirmation of the validity of asymptotic complexity analysis.

• Figure 5 is a member of a suite of plots to explore the performance consequences of the structural qualities of the graphs. In this case, we measure the performance of the three MST algorithms when edges have one, two, three, or four levels of costs, or (at the other extreme) have mostly unique costs. This variation is primarily a dial on the number of different MSTs that can be created for a graph, fewer levels giving us higher numbers of MSTs. We notice that the performance of Kruskal and PrimE are susceptible to this variation while PrimN is stable. However, with few cost levels, Kruskal and PrimE outperform PrimN.



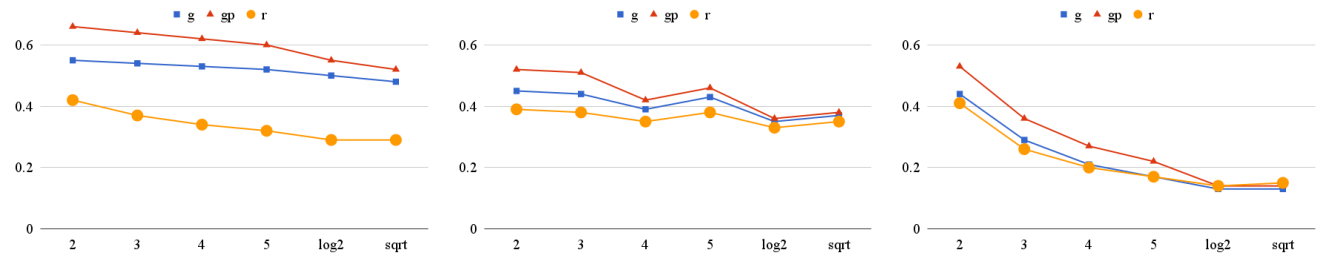
(a) 2 edges per node (b) 4 edges per node (c) $\log_2 N$ edges per node

Figure 4: Completion times of Kruskal (circle), PrimE (square), and PrimN (triangle): The x-axis represent the size of the graphs (number of nodes) and the y-axis shows the logarithm (base 2) of the time (milliseconds) it takes to construct an MST. All plots are based on random graphs and edges have unique cost values. Sub-figures 4a, 4b, and 4c distinguish the results based on the edge-density of the graph nodes (shown in the sub-captions).



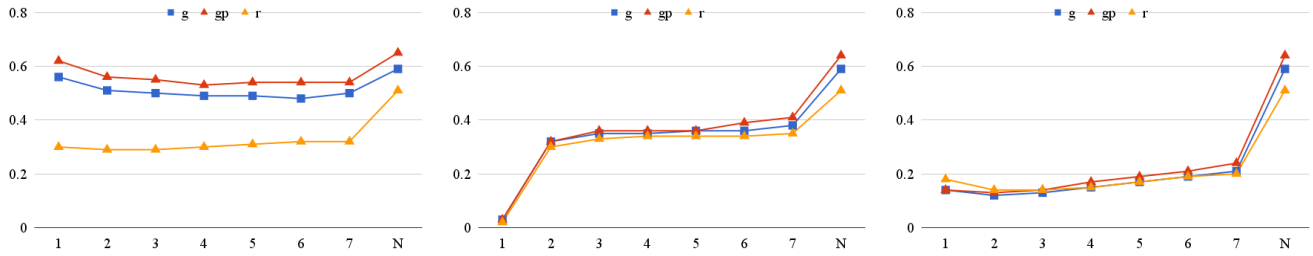
(a) Kruskal (b) PrimE (c) PrimN

Figure 5: Consequences of edge cost distribution on algorithmic performance: The x-axis represent the edge-density of nodes and the y-axis shows the logarithm (base 2) of the time (milliseconds) it takes to construct an MST. Edge density values of 2, 3, 4, and 5 represent sparse graphs where $|E| = O(|N|)$; edge density values of $\log_2(N)$ and $\sqrt{N}$ take us in the "dense graph" direction where $|E| = \omega(|N|)$. All plots are based on grown networks (with preferential attachment) of 32768 nodes. Sub-figures 5a, 5b, and 5c distinguish the results of Kruskal, PrimE, and PrimN. In each subplot, circles represent graphs with no edge cost, squares represent graphs with only two levels of edge costs, etc, and N represents graphs edges costs are mostly unique.



(a) Kruskal (b) PrimE (c) PrimN

Figure 6: Consequences of graph structure on the compactness of the resulting MSTs: The x-axis represents the edge-density of nodes and the y-axis represents the ratio of the average pairwise node-distance of the original graph to the same measure of the corresponding MST. For all plots, the graphs have 256 nodes and edge costs are drawn from three distinct levels. Sub-figures 5a, 5b, and 5c contrast the results of Kruskal, PrimE, and PrimN. In each subplot, circles represent random graphs, squares represent grown networks, and triangles represent grown network with preferential attachment.



(a) Kruskal (b) PrimE (c) PrimN

Figure 7: Consequences of graph structure on the compactness of the resulting MSTs: The x-axis represents the number of levels used for edge costs (where the last value of N represents unique costs for most edges) and the y-axis represents the ratio of the average pairwise node-distance of the graphs to the average pairwise node-distance of the corresponding MSTs (same as figure 6). For all plots, the graphs have 256 node with edge densities set at $\lg(|N|)$. Sub-figures 7a, 7a, and 7a distinguish the results of Kruskal, PrimE, and PrimN. In each subplot, circles represent random graphs, squares represent grown networks, and triangles represent grown network with preferential attachment.

• As we illustrated earlier in figure 2, the multiple MSTs of a graph will typically differ in measures such as diameter and average pairwise node distance. In many realizations of graphs in the real world (such as communication networks and social networks), minimal diameter MSTs are desirable because they imply faster communication. Therefore, we also tracked the ratio of

$$\frac{\text{average pairwise distance of } G}{\text{average pairwise distance of } G\text{'s MST}}$$

This value is contained between zero (exclusive) and one (inclusive) where high values represent greater levels of compaction[6]. We plotted the ratio for the three algorithms as a function of edge density in figure 6 and as a function of edge cost distribution in figure 7. PrimN typically produced high-diameter MSTs. In particular, while PrimN dropped down to 0.2, Kruskal's ratios hovered around 0.5, revealing an interesting difference between these two algorithms that cannot be observed in their time complexity analysis.

5 CONCLUSIONS

This project was tested twice at the author's home institution. Because of small class sizes, it was difficult to establish reliable validation results but students were still polled with a brief survey (five point Likert scale, 1 representing the most negative response and 5 representing the most positive response) at the end of both semesters (17 students in the first offering, 21 in the other). What encouraged us to further invest in this direction (and to try it in the summer offering of an algorithms course at an R1 institution, leading to similar positive results) was based on two observations from this survey. First, when students were asked about the learning value of this project, their responses went from 4.2 in one semester to 4.7 in the following semester while the perceived difficulty went down from 4.4 to 4.1. Aside from wordsmithing edits in the instructional material, the primary difference between these semesters were the use of pivot tables for analysis and interfaces (in the programming sense) for the integration of the components. From an instructional point of view, observing students negotiating their empirical results with asymptotic predictions as well as comprehending how the scattered nature of Kruskal's operation (i.e. the merging of many local MSTs into a final single one) correlates with the compactness of the MSTs it produces (as evidenced in figures 6 and 7) proved to be a good return to our investment.

In order to prevent the instructional material (and solutions) from wandering into the wild, the author is happy to share them with the educational community upon request.

[6]We computed these ratios by Floyd's algorithm which is a dynamic programming instance that elegantly connects the project to one of the end topics of the course.

ACKNOWLEDGMENTS

The author would like to thank Keith Schwarz for making available his '`FibonacciHeap`' class that was used in our PrimN to provide an asymptotically efficient '`decreaskey()`' method[7].

REFERENCES

[1] Walter L Bateman. 1990. *Open to Question. The Art of Teaching and Learning by Inquiry.* ERIC.

[2] Ali Erkan, John Barr, Tony Clear, Cruz Izu, Cristian Jose Lopez del Alamo, Hanan Mohammed, and Mahadev Nadimpalli. 2017. Developing a Holistic Understanding of Systems and Algorithms Through Research Papers. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17).* ACM, New York, NY, USA, 390–390. https://doi.org/10.1145/3059009.3081329

[3] Jon Kleinberg and Eva Tardos. 2005. *Algorithm Design.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[4] Michael J Prince and Richard M Felder. 2006. Inductive teaching and learning methods: Definitions, comparisons, and research bases. *Journal of engineering education* 95, 2 (2006), 123–138.

[5] David Reed, Craig Miller, and Grant Braught. 2000. Empirical Investigation Throughout the CS Curriculum. In *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education (SIGCSE '00).* ACM, New York, NY, USA, 202–206. https://doi.org/10.1145/330908.331855

[6] Robert Sedgewick and Kevin Wayne. 2011. *Algorithms* (4th ed.). Addison-Wesley Professional.

[7] John R Staver and Mary Bay. 1987. Analysis of the project synthesis goal cluster orientation and inquiry emphasis of elementary science textbooks. *Journal of Research in Science Teaching* 24, 7 (1987), 629–643.

[8] J. Ángel Velázquez-Iturbide and Antonio Pérez-Carrasco. 2009. Active Learning of Greedy Algorithms by Means of Interactive Experimentation. *SIGCSE Bull.* 41, 3 (July 2009), 119–123. https://doi.org/10.1145/1595496.1562917

[7]http://www.keithschwarz.com/interesting/code/?dir=fibonacci-heap