NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
Karachi Campus

Project Report
Section: BCS-5J

DESIGN AND ANALYSIS OF ALGORITHMS (CS-2009)

Instructor Name: Miss Anaum Hamid

Group Members

Sufiyaan Usmani (21K-3195)
Yousuf Ahmed Siddiqui (21K-4594)
Muhammad Shahmir Raza (21K-3158)

Contents

1 Abstract

This project focuses on the Design and Analysis of Geometric Algorithms, implementing various algorithms with different time complexities (O(h)) for solving geometric problems. The algorithms included in the project are Line Segment Intersection, Graham Scan, Jarvis March, Quick Hull, among others. The study involves a comprehensive analysis of these algorithms, exploring their theoretical foundations, practical implementations, and experimental results.
Project Link: https://glare.netlify.app/

2 Introduction

The field of geometric algorithms plays a crucial role in various applications, ranging from computer graphics to geographical information systems. This project delves into the design and analysis of several geometric algorithms, each tailored to address specific problems related to geometric shapes. The objective is to implement these algorithms and analyze their time complexities under different scenarios.
The project is divided into two sections:

- Line Segment Intersection: Detects intersections among a set of line segments, essential for applications in computer graphics and computational geometry.
- Convex Hull: The convex hull of a set of points is the smallest convex polygon that encompasses all the given points. It is a fundamental concept in computational geometry and finds applications in areas such as pattern recognition, computer graphics, and geographic information systems. Constructing the convex hull involves determining the vertices that form the outer boundary of the convex polygon, providing a compact representation of the spatial distribution of points and facilitating efficient geometric computations. Following algorithms are implemented in this project:
  - Brute Force
  - Jarvis March
  - Graham Scan
  - Quick Elimination
  - Andrew Scan

3 Technologies and Programming Design

3.1 Front-End

In the frontend development of this project, HTML, CSS, and Bootstrap were utilized to create a visually appealing and responsive user interface. HTML provided the structural foundation for the web pages, defining the layout and organization of content. CSS was employed to enhance the presentation and styling, ensuring a cohesive and aesthetically pleasing design. Additionally, Bootstrap, a front-end framework, was integrated to streamline the development process, offering pre-designed components and responsive grid systems for improved scalability across different devices.


HTML


CSS


Bootstrap

3.2 Back-End

In the implementation of the geometric algorithms for this project, the backend was developed using JavaScript. Leveraging the versatility and asynchronous capabilities of JavaScript, the backend seamlessly handles the computational aspects of the geometric algorithms, including line segment intersection, convex hull construction, and other geometric computations.


Javascript

3.3 Other Tools

VS Code

Git

GitHub

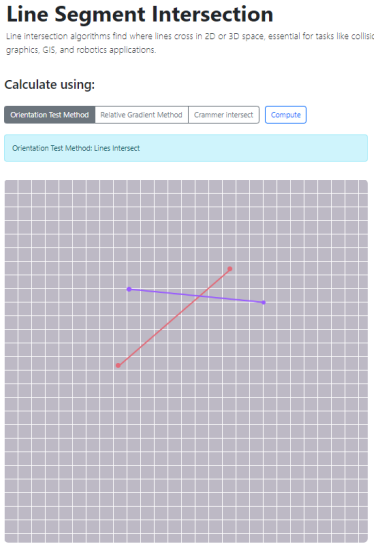
4 Experimental Setup

4.1 Checking Line Segment Intersection

4.1.1 Orientation Test Method

The algorithm relies on the concept of orientation, which involves evaluating the direction of triplets of points. Given three points A, B, and C, the orientation can be categorized as clockwise, counterclockwise, or collinear.
The algorithm operates by examining the orientations of two pairs of points from each line segment. If the orientations of these pairs differ, it implies that the corresponding line segments intersect. This method leverages the cross-product of vectors formed by the points to establish the relative orientation, allowing for a straightforward and efficient intersection check.

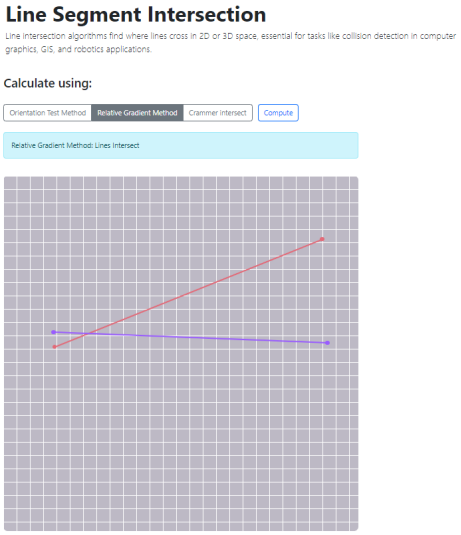
- Time Complexity: $\mathcal{O}(n \log n)$
- Space Complexity: $\mathcal{O}(n)$



4.1.2 Relative Gradient Method

The Relative Gradient Method, employed in the programming design for line segment intersection, is a computationally efficient algorithm that assesses whether two line segments intersect. By calculating and comparing the relative gradients (slopes) of the line segments, the algorithm identifies potential intersections. If the gradients differ, implying an intersection, the method calculates the intersection point and verifies its existence within the bounds of both line segments. Special considerations are given to handle cases such as vertical lines with undefined gradients

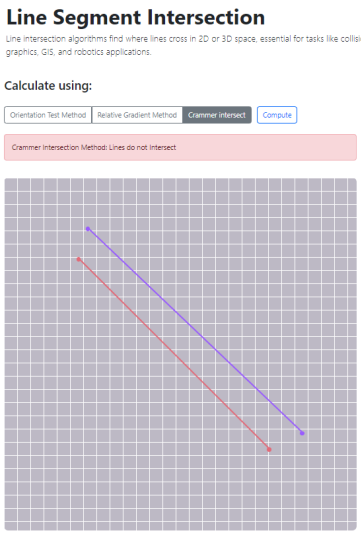
- Time Complexity: $\mathcal{O}(1)$
- Space Complexity: $\mathcal{O}(1)$



4.1.3 Cramer Intersect

[1] The provided algorithm, utilizing Cramer's Rule, determines the intersection of two line segments defined by given points. It begins by computing vectors s1 and s2 representing the direction of each line segment. Cramer's Rule is then applied to solve a system of linear equations, determining parameters s and t to express the potential intersection point. The algorithm checks if s and t fall within the range [0, 1], indicating a valid intersection within the bounds of both line segments. If an intersection is found, the algorithm computes and returns the corresponding coordinates

- Time Complexity: $\mathcal{O}(1)$
- Space Complexity: $\mathcal{O}(1)$



4.2 Convex Hull

4.2.1 Brute Force

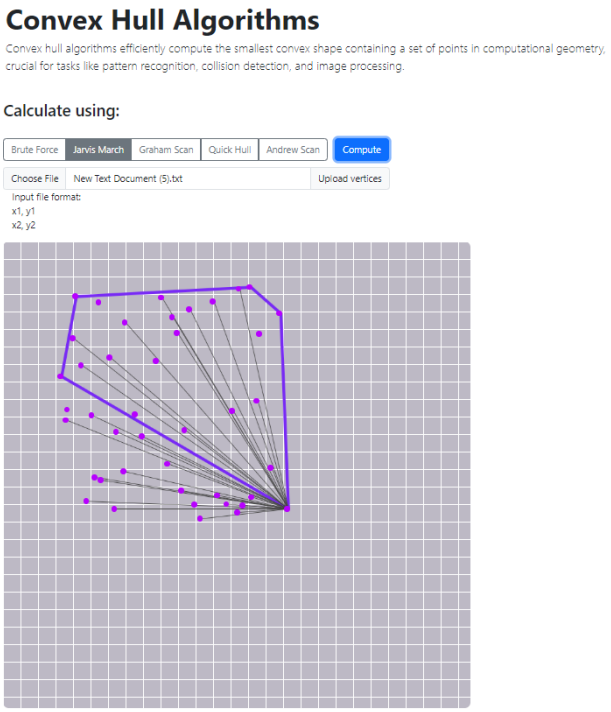
The approach involves evaluating all possible combinations of points to determine the subset that forms the convex hull. For each point, the algorithm checks its inclusion in the convex hull by verifying that all other points lie either to its left or right. This exhaustive examination of all possible subsets ensures the identification of the outermost vertices, forming the convex hull

- Time Complexity: $\mathcal{O}(n^3)$
- Space Complexity: $\mathcal{O}(n)$

4.2.2 Jarvis March

Also known as the Gift Wrapping Algorithm, it iteratively selects the point with the smallest polar angle relative to the current point as part of the convex hull. Starting with the point of the lowest y-coordinate as the initial pivot, the algorithm systematically extends the convex hull by selecting the next point based on its angular relationship with the current hull vertex. This process continues until the convex hull is fully enclosed.

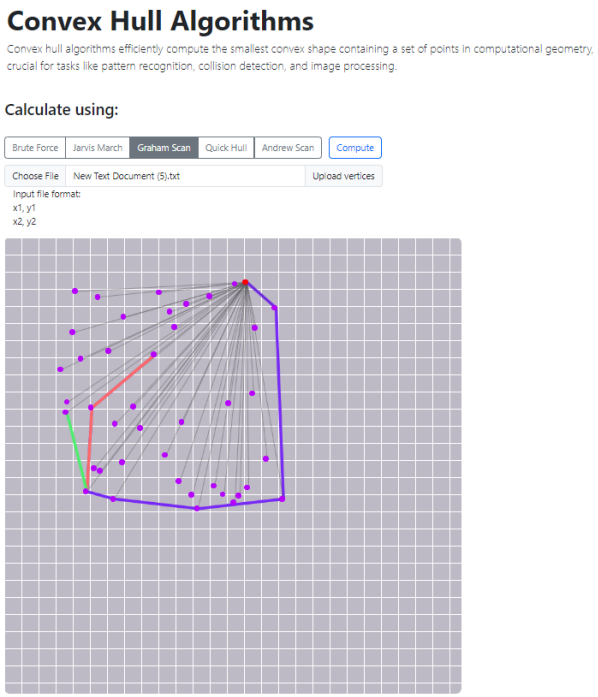
- Time Complexity: $\mathcal{O}(nh)$
- Space Complexity: $\mathcal{O}(h)$



4.2.3 Graham Scan

Employing a stack to streamline the process, the algorithm begins by selecting the point with the lowest y-coordinate (and the leftmost point in the case of a tie) as the pivot. It then sorts the remaining points based on their polar angles with respect to the pivot. By systematically considering each point in the sorted order, the algorithm efficiently identifies the vertices of the convex hull, eliminating interior points. The stack plays a crucial role in maintaining the convex hull's current state, allowing for the dynamic addition and removal of points as the algorithm progresses.

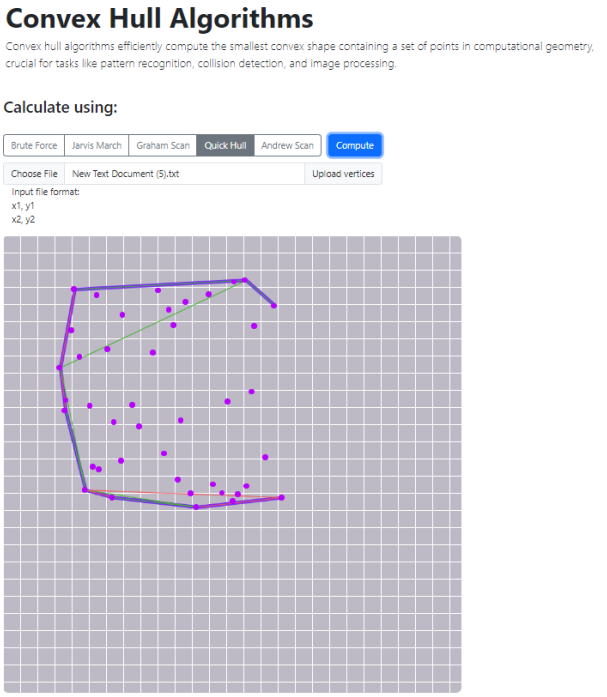
- Time Complexity: $\mathcal{O}(n \log n)$
- Space Complexity: $\mathcal{O}(n)$



4.2.4 Quick Elimination

Rooted in a divide-and-conquer strategy, this algorithm begins by sorting the points based on their x-coordinates. Subsequently, it identifies the extreme points, namely the leftmost and rightmost, which must be part of the convex hull. By dividing the set of points into two subsets on either side of the line connecting these extreme points, the algorithm recursively applies itself to each subset. This recursive division reduces the problem size and accelerates the identification of hull vertices.

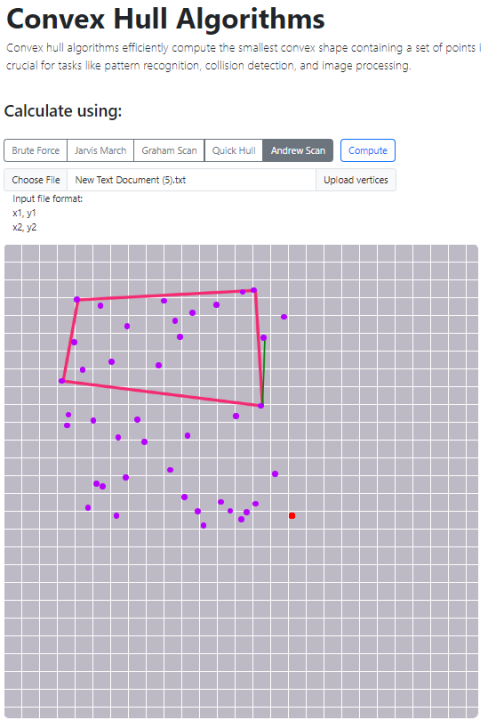
- Time Complexity: $\mathcal{O}(n \log n)$
- Space Complexity: $\mathcal{O}(n)$



4.2.5 Andrew Scan

Andrew Scan[2], also known as the Monotone Chain algorithm, is a widely used method for constructing convex hulls in computational geometry. The algorithm efficiently computes the convex hull of a set of points in the plane by first sorting the points lexicographically. It then processes the sorted points in two phases—upper hull and lower hull—building the convex hull incrementally. During each phase, the algorithm maintains a stack to keep track of the convex hull vertices.
The programming design involves careful consideration of the lexicographical sorting step, stack management, and the monotonicity conditions to facilitate an elegant and efficient implementation of the convex hull construction.

- Time Complexity: $\mathcal{O}(n \log n)$
- Space Complexity: $\mathcal{O}(n)$



5 Results and Discussion

5.1 Line Segment Intersection Algorithms

5.1.1 Orientation Test Method

Time Complexity: $\mathcal{O}(n \log n)$
Space Complexity: $\mathcal{O}(n)$
Advantages

- Deterministic Precision: Provides deterministic and precise results through mathematical orientation calculations for consistent and accurate intersection detection.
- Versatility Across Configurations: Applicable to various geometric configurations, accommodating line segments with diverse slopes and orientations for a broad range of scenarios.
- Consistency Across Platforms: Relies on fundamental geometric principles, ensuring consistent behavior across different platforms and programming languages, promoting platform-independent reliability.

Disadvantages

- Computational Complexity: Exhibits higher computational complexity, especially with larger datasets, due to the calculation of orientations for multiple point triplets, resulting in increased processing time.
- Numerical Precision Sensitivity: Sensitive to numerical precision issues, particularly with collinear or nearly collinear points, introducing potential inaccuracies in intersection detection that require careful handling.
- Handling of Special Cases: Additional considerations and handling are necessary for special cases, such as vertical lines with undefined slopes, adding complexity to the implementation.

5.1.2 Relative Gradient Method

Time Complexity: $\mathcal{O}(1)$
Space Complexity: $\mathcal{O}(1)$
Advantages

- Simplicity and Efficiency: The method is conceptually simple and computationally efficient, involving basic arithmetic operations for gradient calculations, making it accessible and fast for line intersection detection.
- Versatility: The relative gradient approach is applicable to a wide range of scenarios involving non-vertical and non-horizontal lines, providing a versatile solution for geometric computations.
- Numerical Stability: The method typically exhibits good numerical stability, with low susceptibility to precision issues, ensuring accurate and reliable intersection results in various practical applications.

Disadvantages

- Handling Special Cases: While versatile, the method may require additional handling for special cases such as parallel or coincident lines, introducing complexity in the implementation.
- Limited to Finite Slopes: The method assumes finite slopes for the lines, and handling vertical lines may necessitate special treatment, potentially adding complexity to the algorithm.
- Suboptimal for Parallel Processing: In scenarios where parallel processing is crucial, the method may not be the most efficient choice due to its sequential nature and dependency on the relative gradient comparisons.

5.1.3 Cramer Intersect

Time Complexity: $\mathcal{O}(1)$
Space Complexity: $\mathcal{O}(1)$
Advantages

- Robustness: The algorithm is robust and applicable to various scenarios, providing accurate intersection results for arbitrary line segments.
- Efficiency: With a time complexity of O(1), the algorithm is efficient for determining line segment intersections, especially in real-time applications.
- Simplicity: Cramer's Rule simplifies the solution of linear equations, making the algorithm straightforward and easy to implement.

Disadvantages

- Numerical Stability: Precision issues may arise due to floating-point arithmetic, impacting the accuracy of intersection results.
- Special Cases: The algorithm may need additional handling for special cases, such as parallel or nearly parallel line segments, where the intersection point might not be well-defined.
- Limited to 2D: The algorithm is designed for 2D space, and extending it to higher dimensions may require significant modifications.

5.2 Convex Hull Algorithm

5.2.1 Brute Force

Time Complexity: $\mathcal{O}(n^3)$
Space Complexity: $\mathcal{O}(n)$
Advantages

- Simplicity and Conceptual Clarity: The Brute Force approach is conceptually straightforward, making it an ideal choice for educational purposes or initial understanding of convex hull concepts.
- Applicability to Small Datasets: In scenarios where the dataset is small, the Brute Force method can provide a viable solution without the need for complex algorithmic optimizations.
- Ease of Implementation: The algorithm is relatively easy to implement, making it accessible for programmers who are new to geometric algorithms and convex hull problems.

Disadvantages

- Inefficiency for Real-world Applications: The method's inefficiency makes it less suitable for real-world applications with substantial datasets, where more optimized algorithms are preferred.
- Limited Scalability: As the size of the dataset increases, the Brute Force method quickly becomes computationally infeasible, limiting its scalability and practical utility in modern computational contexts.

5.2.2 Jarvis March

Time Complexity: $\mathcal{O}(nh)$
Space Complexity: $\mathcal{O}(h)$
Advantages

- Simplicity and Ease of Implementation: The Jarvis March algorithm is conceptually simple and easy to implement, making it an accessible choice for educational purposes and quick prototyping of convex hull solutions.
- Applicability to Arbitrary Point Sets: Jarvis March is versatile and can handle arbitrary point sets, including non-uniform distributions, making it a robust choice for scenarios where the input data's characteristics may vary.
- Minimal Space Complexity: The algorithm typically has low space requirements, utilizing only a constant amount of memory for bookkeeping, contributing to its efficiency in terms of space utilization.

Disadvantages

- High Time Complexity: Jarvis March has a time complexity of O(nh), where 'n' is the number of points and 'h' is the number of vertices in the convex hull. This can make the algorithm inefficient for large datasets or when the convex hull has a significant number of vertices.
- Sensitive to Initial Point Selection: The algorithm's output depends on the selection of the initial point, and in certain cases, poor initial choices can lead to suboptimal or incorrect results, making it less deterministic than some other convex hull algorithms.
- Not Suitable for Parallelization: Jarvis March is inherently sequential, and its iterative nature limits opportunities for parallelization, potentially hindering performance improvements on modern parallel computing architectures.

5.2.3 Graham Scan

Time Complexity: $\mathcal{O}(n \log n)$
Space Complexity: $\mathcal{O}(n)$
Advantages

- Efficiency: Graham Scan typically exhibits a time complexity of O(n log n), making it more efficient than brute-force methods for convex hull construction, particularly for moderate-sized datasets.
- Convexity Guarantee: The algorithm ensures that the output is a convex hull, providing a reliable and predictable solution for geometric problems requiring the identification of the outermost points.
- Simplicity of Implementation: Graham Scan's intuitive approach, involving sorting and a single pass through the sorted points, makes it relatively simple to implement, understand, and integrate into projects.

Disadvantages

- Limited Robustness for Collinear Points: The algorithm may encounter challenges with sets of collinear points, potentially leading to suboptimal performance or the need for additional handling.
- Dependence on Sorting Step: The initial sorting of points based on polar angles introduces an overhead that may impact the algorithm's overall efficiency, particularly for large datasets.
- Non-Adaptability to Dynamic Input: Graham Scan assumes a static set of points, and significant adjustments or recalculations are required when dealing with dynamic input, affecting its adaptability to certain real-time or streaming scenarios.

5.2.4 Quick Elimination

Time Complexity: $\mathcal{O}(n \log n)$
Space Complexity: $\mathcal{O}(n)$
Advantages

- Improved Time Complexity: Quick Elimination employs a divide-and-conquer strategy, resulting in a more favorable time complexity (O(n log n)), providing significant efficiency gains compared to brute-force methods, particularly for larger datasets.
- Efficient Identification of Extreme Points: By sorting the points based on their x-coordinates and recursively processing divided subsets, Quick Elimination efficiently identifies extreme points crucial for convex hull construction, streamlining the algorithm's overall execution.
- Scalability for Large Datasets: The divide-and-conquer nature of Quick Elimination allows for efficient handling of large datasets, making it well-suited for real-world applications where scalability is essential for geometric algorithm performance.

Disadvantages

- Potential Sensitivity to Input Order: Quick Elimination relies on sorting the input points, and its efficiency may be impacted by the initial order of the points. In certain scenarios, pre-sorting may be required to mitigate sensitivity to the input order.
- Suboptimal Performance for Certain Distributions: While generally efficient, Quick Elimination may exhibit suboptimal performance for certain point distributions, such as nearly collinear sets, where the divide-and-conquer strategy may not yield significant advantages.
- Recursive Overhead: The recursive nature of the algorithm introduces overhead, and in some cases, it may not be the most optimal choice for small datasets or situations where the inherent structure of the dataset does not lend itself well to the divide-and-conquer approach.

5.2.5 Research Paper Method

Time Complexity: $\mathcal{O}(n \log n)$
Space Complexity: $\mathcal{O}(n)$
Advantages

- Improved Time Complexity: Quick Elimination employs a divide-and-conquer strategy, resulting in a more favorable time complexity, providing significant efficiency gains compared to brute-force methods, particularly for larger datasets.
- Efficient Identification of Extreme Points: By sorting the points based on their x-coordinates and recursively processing divided subsets, Quick Elimination efficiently identifies extreme points crucial for convex hull construction, streamlining the algorithm's overall execution.
- Scalability for Large Datasets: The divide-and-conquer nature of Quick Elimination allows for efficient handling of large datasets, making it well-suited for real-world applications where scalability is essential for geometric algorithm performance.

Disadvantages

- Potential Sensitivity to Input Order: Quick Elimination relies on sorting the input points, and its efficiency may be impacted by the initial order of the points. In certain scenarios, pre-sorting may be required to mitigate sensitivity to the input order.
- Suboptimal Performance for Certain Distributions: While generally efficient, Quick Elimination may exhibit suboptimal performance for certain point distributions, such as nearly collinear sets, where the divide-and-conquer strategy may not yield significant advantages.
- Recursive Overhead: The recursive nature of the algorithm introduces overhead, and in some cases, it may not be the most optimal choice for small datasets or situations where the inherent structure of the dataset does not lend itself well to the divide-and-conquer approach.

6 Conclusion

In conclusion, this project contributes to the understanding of geometric algorithms by implementing and analyzing key algorithms with varying time complexities. The study sheds light on the strengths and weaknesses of each algorithm, providing valuable insights for their practical applications.
This project underscores the significance of geometric algorithms in solving real-world problems and lays the groundwork for future research in this dynamic and evolving field. The lessons learned from this project can be applied to enhance the efficiency of geometric computations in diverse applications, from computer-aided design to robotics and beyond.

7 References

[1] Andre LaMothe. "Tricks of the 3D game programming gurus" Chapter 13 P.840

[2] Min, Hla, and Si-Qing Zheng. "Time-space optimal convex Hull algorithms." Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice. 1993.

[3] Gavrilova, Marina, and Jon G. Rokne. "Reliable line segment intersection testing." Computer-Aided Design 32.12 (2000): 737-745.

[4] Barber, C. Bradford, David P. Dobkin, and Hannu Huhdanpaa. "The quickhull algorithm for convex hulls." ACM Transactions on Mathematical Software (TOMS) 22.4 (1996): 469-483.

[5] Gamby, Ask Neve, and Jyrki Katajainen. "Convex-hull algorithms: Implementation, testing, and experimentation." Algorithms 11.12 (2018): 195.