IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)
An International Open Access, Peer-reviewed, Refereed Journal

SPACE-EFFICIENT IN-PLACE ALGORITHMS FOR COMPUTATIONAL GEOMETRY ON UNIT DISK GRAPHS

[1]Asla M. P
[1]Assistant Professor
[1]Department of Artificial Intelligence and Cyber Security,
[1]Ilahia College of Engineering and Technology, Muvattupuzha, Kerala, India

Abstract—Memory-constrained computing environments necessitate algorithms that minimize auxiliary space usage while maintaining computational efficiency. This paper presents novel in-place implementations of fundamental geometric algorithms on Unit Disk Graphs (UDGs), a critical structure in wireless network modeling and computational geometry. We develop and analyze three space-efficient algorithms: (1) maximum clique detection using bipartite matching with O $(n^2 + m (n + K^3))$ time complexity, (2) minimum enclosing circle computation, and (3) closest/farthest pair identification. All algorithms operate within O (1) auxiliary space constraints while processing element datasets. Experimental results demonstrate that our implementations achieve correct solutions with significantly reduced memory footprint compared to conventional approaches, making them suitable for embedded systems, IoT devices, and largescale geometric data processing. The maximum clique algorithm successfully identifies cliques in wireless network topologies, while the MEC algorithm provides optimal facility placement solutions. Our work bridges theoretical computational geometry with practical resource-constrained implementations.

Index Terms—Unit Disk Graph, In-Place Algorithms, Space Complexity, Bipartite Matching, Minimum Enclosing Circle, Maximum Clique, Computational Geometry, Wireless Networks.

1 INTRODUCTION

The proliferation of resource-constrained computing devices from IoT sensors to embedded systems demands algorithms that minimize memory consumption without sacrificing correctness or excessive computational overhead. Traditional algorithm design often prioritizes time complexity, accepting linear or even quadratic auxiliary space requirements. However, in memory-limited environments, such space usage becomes prohibitive.

In-place algorithms offer an elegant solution by restricting auxiliary space usage to O (1) while allowing modification of the input array. Formally, an in-place algorithm must satisfy four constraints: input objects reside in array elements, element swapping is permissible during execution, all input elements must remain present post-execution (possibly reordered), and only constant auxiliary space may be utilized. This model contrasts with the more restrictive read-only model, which prohibits any input modification.

1.1 Unit Disk Graphs: Theory and Applications

A Unit Disk Graph (UDG) G = (V, E) is the intersection graph of equal-radius disks in Euclidean space. Each vertex $v_i \in V$ corresponds to a disk center, and an edge $(v_i, v_j) \in E$ exists if and only if the Euclidean distance between centers satisfies $d(v_i, v_j) \le 2r$, where r is the unit radius. UDGs naturally model wireless communication networks where each node has uniform transmission range. This geometric structure captures fundamental properties of ad-hoc networks, sensor deployments, and cellular systems. Despite their geometric simplicity, many computational problems on UDGs including coloring, independent set, and clique detection remain NP-hard in the general case.

Application domains include wireless networks for topology modeling and range assignment, facility location for optimal placement problems such as hospitals and emergency services, clustering analysis for identifying dense regions in spatial datasets, and collision detection for proximity queries in robotics and computer graphics.

1.2 Problem Formulation

This paper addresses three fundamental problems. The first problem concerns maximum clique detection: given a set $C = \{C_1, C_2, \ldots, C_n\}$ of unit disks with centers $c_i = (\alpha_i, \beta_i)$, compute the size of the maximum clique in the corresponding UDG using O(1) auxiliary space. The second problem addresses minimum enclosing circle: given n points $P = \{p_1, \ldots, p_n\} \subset R^2$, compute the radius and center of the minimum-radius circle containing all points using O(1) auxiliary space. The third problem

identifies extreme pairs: for a point set P, identify the closest pair (p_i, p_j) minimizing $d(p_i, p_j)$ and the farthest pair maximizing this distance, using O (1) auxiliary space.

2 RELATED WORK

2.1 Geometric Intersection Graphs

Clark and Colbourn established foundational complexity results for UDGs, proving NP-completeness of various optimization problems. Their work demonstrated that despite geometric constraints, UDGs retain computational hardness properties of general graphs. Subsequent research has explored approximation algorithms and parameterized complexity. Das et al. developed streaming algorithms for UDG problems, though with logarithmic space requirements. Our work extends this line by achieving constant space bounds through careful algorithm engineering.

2.2 In-Place Algorithm Design

The in-place computational model has received increasing attention in big data contexts. Heap sort exemplifies classic in-place design: O (n log n) time with O (1) auxiliary space. Recent work has extended in-place techniques to geometric problems, though comprehensive treatment of UDG algorithms remained unexplored. De et al. pioneered in-place geometric algorithms, introducing techniques for convex hull computation and range searching. Our maximum clique algorithm builds upon their bipartite matching framework, adapting it specifically for UDG topology.

2.3 Facility Location and MEC

The minimum enclosing circle problem dates to Sylvester (1857) and has practical importance in facility location theory, known as the 1-center problem in optimization literature. Welzl's algorithm achieves O(n) expected time using randomization and recursion with implicit O (n) stack space. Our contribution provides a deterministic O (n) algorithm with O (1) auxiliary space, trading time for memory, an acceptable tradeoff in memory-critical applications where n remains moderate (less than 1000 points).

2.4 Closest Pair Algorithms

The closest pair problem admits O (n log n) solutions via divide-and-conquer or plane-sweep techniques. However, these require O (n) auxiliary storage for recursion or sorting. Our in-place approach using heap sort maintains O (1) space while accepting O (n^2 log n) time, appropriate when space is the dominant constraint.

3 METHODOLOGY

3.1 Algorithm 1: Maximum Clique via Bipartite Matching

The key insight exploits the complementary relationship between cliques and independent sets: a clique in graph G corresponds to an independent ^{ij}set in complement graph. For UDGs, we can efficiently construct relevant subgraphs. Let C_i, C_j be two intersecting disks. Any clique containing both must have all member centers in region R_{ij}, the intersection of circles of radius 2 centered at c_i and c_j. We partition R_{ij} into half-planes R^1 and R^2 by line segment $[c_i, c_j]$. The bipartite graph $G^B = (C^1, C^2, E_{ij})$ has edge ij $(C_p, C_q) \in E_{ij}$ if disks C_p and C_q do not intersect, that is, $d(c_p, c_q) > 2$. The maximum clique in subgraph induced by R_{ij} has size $|R_{ij}| - |M_{ij}|$, where M_{ij} is a maximum matching in G^B.

3.1.1 In-Place Implementation Strategy

We maintain $_{ij}$implicit bipartite structure through$_{ij}$ careful index management. The array organization maintains C[0] and C[1] as the current disk pair under consideration, C [2 to] as disks in C^1 with matched disks from 2 to and exposed disks from +1 to , and C [+1 to m] as disks in C^2 with matched disks from +1 to + and exposed disks from ++1 to m.

The matching construction employs an augmenting path approach. We start with empty matching with equals zero. For each exposed vertex $w \in B^1$, we seek augmenting path to B^2 . If direct edge (w, v) exists with v exposed, we match them and increment Otherwise, we search for alternating path through matched vertices. If no augmenting path exists, we mark was useless and decrement.

The complexity analysis shows outer loop of $O(n^2)$ pairs, per pair of $O(m)$ intersection tests plus $O(K^3)$ matching, giving overall $O(n^2 + m(n + K^3))$ time with $O(1)$ space.

3.2 Algorithm 2: Minimum Enclosing Circle

The MEC satisfies crucial geometric properties. The MEC is uniquely determined by 2 or 3 points on its boundary. If 2

Given three non-collinear points, slopes of perpendicular bisectors are computed as $m_1 = \frac{y_i - y_j}{x_i - x_j}$ with perpendicular slope $m^{\perp} = -\frac{1}{\ }$, and similarly for m_2 and $m^{\perp}$. Circumcenter coordinates are obtained by solving bisector intersection. Only scalar variables including current radius, best radius, and center coordinates are maintained with no auxiliary arrays. Complexity is O(n) time for $O(n^3)$ triples times O (n) verification, with O (1) space.

3.3 Algorithm 3: Extreme Point Pairs

Both closest and farthest pair algorithms begin with sorting via heap sort. Heap construction builds max-heap in-place with O (n log n) time, maintaining heap property during extraction with no auxiliary arrays. The sorting phase repeatedly swaps root with last element, decreases heap size and re-heapifies, resulting in sorted array in ascending order.

After sorting by x-coordinate, we iterate through all pairs computing distance as:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (1)$$

For closest pair, we initialize $d_{min} = \infty$ and update when smaller distance found. For farthest pair, we initialize $d_{max} = 0$ and update when larger distance found. Complexity is $O(n^2 \log n)$ time for sort plus all pairs, with O (1) space.

4 IMPLEMENTATION DETAILS

4.1 Data Structures

Point representation uses a structure containing float x and float y coordinates. Disk and circle are represented implicitly by center Point and unit radius assumption.

4.2 Key Functions

Distance computation is implemented as a function taking two points and returning the square root of the sum of squared coordinate differences. Intersection test returns true if distance between two centers is less than or equal to 2.0. Bipartite edge check returns true if distance between two centers is greater than 2.0, indicating non-intersection.

4.3 Memory Management

All algorithms operate on input array A[1 to n] with only scalar variables including loop indices i, j, k, temporary swap variables, matching counters, , distance accumulators d_{min}, d_{max}, and best solution trackers for radius and center coordinates. Total auxiliary space is O(1) regardless of input size.

5 EXPERIMENTAL RESULTS

5.1 Experimental Setup

Experiments were conducted on Intel Core i7-9700K processor with 16GB RAM running Ubuntu 20.04 and GCC 9.3.0 compiler. Datasets included random uniform points uniformly distributed in $[0, 1000]^2$, clustered data with Gaussian mixture with 5 centers, and grid-aligned regular lattice with perturbations. Metrics measured included execution time as wall-clock averaged over 10 runs, peak memory usage via Valgrind massif, and solution quality verified against brute-force reference.

5.2 Maximum Clique Results

Performance results for maximum clique detection show that for n equals 50, execution time is 12.3 milliseconds with memory usage 0.8 KB and clique size 7. For n equals 100, time is 89.7 ms with 1.6 KB memory and clique size 12. For n equals 200, time is 634.2 ms with 3.2 KB memory and clique size 18. For n equals 500, time is 8234.5 ms with 8.0 KB memory and clique size 29.

Observations indicate that memory usage scales linearly with input size containing only array storage. Time complexity matches theoretical $O(n^2 + mK^3)$ prediction. Clique sizes are consistent with random UDG theory of approximately square root of n.

5.3 Minimum Enclosing Circle Results

MEC performance shows that for n equals 10, execution time is 0.8 ms with 0.2 KB memory and radius error 0.00 percent. For n equals 20, time is 12.4 ms with 0.3 KB memory and error 0.00 percent. For n equals 50, time is 487.3 ms with 0.8 KB memory.

Closest and Farthest Pair Results

Extreme pairs performance demonstrates that for n equals 100, closest pair time is 15.2 ms and farthest pair time is 15.4 ms. For n equals 500, times are 342.8 ms and 345.1 ms respectively. For n equals 1000, times are 1387.4 ms and 1392.6 ms. For n equals 5000, times are 34821.3 ms and 34976.8 ms. Heap sort preprocessing takes O (n log n) time, dominated by O (n^2) pairwise distance computation.

6 DISCUSSION

6.1 Space-Time Tradeoffs

Our algorithms exemplify fundamental tradeoffs in algorithm design. The maximum clique algorithm achieves near-optimal time complexity while maintaining O (1) space through a matching-based approach that avoids explicit graph storage. The MEC algorithm accepts O(n) time for O (1) space, where faster algorithms like Welzl and Megiddo require O(n) recursion stack or randomization unsuitable for deterministic embedded contexts. Extreme pairs algorithm with in-place sorting incurs O (n^2 log n) cost versus O (n log n) with auxiliary storage, which is acceptable when n is less than 10,000 and memory is scarce.

6.2 Practical Implications

Our algorithms suit embedded systems and microcontrollers with limited RAM of less than 64KB. Sensor networks often process hundreds of nodes, well within our scalability range. While not fully streaming, algorithms can process data in fixed-size windows maintaining constant space per window. Reduced memory access translates to lower energy consumption, critical for battery- powered devices.

6.3 Limitations

The MEC algorithm is impractical beyond n equals 100 due to O(n) complexity. The closest pair O(n^2) approach is slower than divide-and-conquer for large n. Maximum clique remains exponential in worst-case clique size K.

6.4 Future Directions

Algorithmic improvements could explore incremental MEC achieving O(n^3) with O(1) space, approximate clique algorithms trading solution quality for speed, and parallelization for multi-core in-place algorithms. Application extensions include 3D UDGs for volumetric wireless networks, dynamic UDGs with node insertion and deletion, and integration with real sensor network deployments.

7 CONCLUSION

We have demonstrated that fundamental geometric problems on Unit Disk Graphs admit efficient in-place implementations using only O (1) auxiliary space. Our algorithms provide practical solutions for memory-constrained environments including embedded systems, IoT networks, and large-scale spatial data processing.

The maximum clique algorithm achieves O (n^2 + m(n+K^3)) time complexity through careful exploitation of UDG geometry and bipartite matching. The MEC and extreme pair algorithms, while accepting higher time complexity, maintain strict constant-space constraints suitable for resource-limited scenarios.

Experimental validation confirms theoretical predictions and demonstrates practical feasibility. Memory usage remains constant regardless of input size, enabling processing of datasets that would exhaust available memory in traditional approaches. This work bridges theoretical computational geometry with pragmatic algorithm engineering, proving that space-efficient implementations can maintain correctness and reasonable performance for moderately-sized problems. As edge computing and IoT proliferate, such memory-conscious algorithms become increasingly essential.

References

[1] M. De, S. C. Nandy, and S. Roy, "In-place algorithms for computing a largest clique in geometric intersection graph "Discrete Applied Mathematics, vol. 178, pp. 58 – 70, 2014.

[2] B. N. Clark, C. J. Colbourn, and D. S. Johnson, "Unit disk graphs," Discrete Mathematics, vol. 86, no. 1 – 3, pp. 165 – 177, 1990.

[3] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in New Results and New Trends in Computer Science, Springer, 1991, pp. 359–370.

[4] S. Xu, R. M. Freund, and J. Sun, "Solution methodologies for the smallest enclosing circle problem," Computational Optimization and Applications, vol. 25, no. 1–3, pp. 283–292, 2003.

[5] P. Gupta and P. R. Kumar, "The capacity of wireless networks," IEEE Transactions on Information Theory, vol. 46, no. 2, pp. 388–404, Mar. 2000.

[6] J. L. Bentley and M. I. Shamos, "Divide-and-conquer in multidimensional space," in Proceedings of the 8th Annual ACM Symposium on Theory of Computing, 1976, pp. 220–230.

[7] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," SIAM Journal on Computing, vol. 12, no. 4, pp. 759–776, 1983.

[8] J. I. Munro and H. Suwanda, "Implicit data structures for fast search and update," Journal of Computer and System Sciences, vol. 21, no. 2, pp. 236–250, 1980.

[9] F. P. Preparata and M. I. Shamos, Computational Geometry: An Introduction. New York: Springer-Verlag, 1985.