



RESEARCH ARTICLE

OPEN ACCESS

OPTIMAL TRIANGLE DETECTION PREVENTING DUPLICATES IN LARGE-SCALE GRAPHS

Sarah Balbouzi^{1*}, Lougmiri Zekri² and Kheira Mechach³

^{1, 2, 3}Department of computer science, Faculty of exact and applied sciences, Oran 1 University Ahmed Ben Bella, Oran, Algeria

¹<http://orcid.org/0009-0009-0606-9037>, ²<http://orcid.org/0000-0002-7563-9787>, ³<http://orcid.org/0009-0009-4447-990X>

E-mail: *balbouzi.sarah@edu.univ-oran1.dz, zekri.lougmiri@univ-oran1.dz, mechach.kheira@univ-oran1.dz

ARTICLE INFO

Article History

Received on: March 10, 2026

Reviewed on: April 10, 2026

Accepted on: May 01, 2026

Published on: July 31, 2026

Keywords:

Communities,
Complex systems,
Social Networks,
Optimized sorted lists
intersection,
Triangle detection.

ABSTRACT

During the past decades, the world shifted towards online transactions. From data exchange, service provision to business, users and providers rely on the internet and social networks as a common platform for sharing and trading. In order to ensure an effective and reliable management of these networks and the flow of information within them, it is essential to detect user profiles and communities. In that context, triangle detection is a fundamental task in network analysis, as triangles are the smallest structure in a community. However, identifying triangles in large-scale and dense networks is a heavy and highly time-consuming treatment, due to costly neighborhood intersection computations. Our research aims to optimize intersection computation by minimizing the number of comparisons performed, avoid triangle duplication throughout the detection process and thus reducing the overall computation time. The method we propose, Test With Jumps for triangle detection without duplicates (TWJtr-D), is a refined version of Test With Jumps for triangle detection (TWJtr) algorithm. It optimizes intersection computing between neighbors sorted lists while avoiding redundant comparisons and duplicated triangles. Experimental results, on real-world datasets, demonstrate that TWJtr-D significantly reduces computation time and outperforms state-of-the-art methods which process in a similar pattern. This highlights the efficiency of TWJtr-D for triangle detection in large-scale networks. Future work will upgrade this method to parallel processing to rank its efficiency among algorithms in the same field.



Copyright ©2026 by authors and Galileo Institute of Technology and Education of the Amazon (ITEGAM). This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).

I. INTRODUCTION

By nature, human beings are curious about their surroundings and other's behaviors such as social interactions. Ever since scholars started studying and searching for logical explanations to various living beings' actions and resolving daily life issues, expressing these situations through a simplified representation was an intuitive first step on the resolution path. In 1735, Euler put the first stepping stone towards graph theory while resolving Königsberg' seven bridges problem [1]. Taking the city's map as a starting point, Euler simplified it by eliminating all unnecessary figures keeping only the seven bridges and the four areas they connect. This new representation made it easier to find a solution that allows citizens to move across the city using each bridge only once. Later on, in 1930, Moreno conducted a social study on school pupils [2]. The results obtained from the proposed survey were represented using a graph, in which Moreno considered each respondent to the survey as a node, and each person they select as their answers established a connection between the two individuals. Given the complexity of the interactions expressed and the large number of actors involved in these interactions, the study field evolved from simple graphic networks to complex systems. Furthermore, large graphs, like social networks, are seen as complex networks [3] where actors interact individually [4]. In order to understand these complex systems, numerous approaches have been proposed. The continuous progress achieved by researchers resulted in the development of multiple versions of complex systems, each with specific properties. In 1959, the first random graph saw light [5].

Expressing synthetic data, this graph included n nodes connected randomly through $\frac{n(n-1)}{2}$ links. The probability of an arc connecting two nodes is p , following a Poisson distribution, and $(1 - p)$ defines its absence probability. Milgram introduced in 1967 the small world model [6]. Inspired by “the six steps apart” notion [7], which speculates that each couple of nodes are connected, at most, through a sequence of five intermediary nodes forming six linked steps. For further analyzes of complex systems, specifically small world models’ properties, the Watts-Strogatz was a key model. Starting from an initially regular system, where all nodes are connected to their k closest neighbors on a circular or grid topology, evolving into a random system where each initial link is replaced by a new edge with p probability [8]. Other systems were also constructed, from the preferential attachment models where Simon introduced the “rich get richer” concept [9] applied on linguistic networks based on words frequency. This model was, later on, extended by adding the cumulative benefit notion and applied over scientific citations’ networks [10]. Although these systems were revolutionary, they still had limits expressing real world characteristics. The Barabasi-Albert model (BA), came as a solution [3] based on a scale-free network using the power-law for degree distribution keeping the preferential attachment concept. To enhance the BA and Dorogovtsev-Mendes-Samkhin (DMS) [11] models, the Bianconi-Barabasi model added a new parameter to each node called fitness or aptitude allowing competitiveness among entities through their own efforts [12].

In 2014, the multiplex systems expressed another side of real-world interactions [13]. These representations express different kinds of interactions in the same graph through multiple layers. They give to each node the possibility to belong to more than one layer. The temporal aspect was also considered to evaluate interactions’ lifespan [14], and shared interactions among groups of entities were expressed through hyperlinks in hypergraphs [15]. With the widespread adoption of the internet, which has significantly interconnected the world, it nowadays serves as the backbone for nearly all modern activities. The continuous transmission of messages, emails and the submission of millions of queries every second have resulted in a tremendous increase in the volume of data flowing through networks. Efficiently processing this vast amount of information, identifying user profiles, and delivering query results pose significant computational challenges. To mitigate these issues, the development of highly efficient indexing systems became essential. These systems establish links between user profiles and documents, while also accounting for their interactions across the network. The indexes, derived from document intersections, enable precise tracking of term locations and facilitate community detection by identifying triangles and, by extension, clusters, of individuals with common interests or behaviors. In this context, graph theory has emerged as an indispensable tool for analyzing such complex networks. The detection of structural patterns, such as triangles and cycles [16], is critical for uncovering key network properties.

Triangles, for instance, offer valuable insights into clustering coefficients and community structures, with applications spanning from social network analysis to network reliability. Several methods have been proposed for triangles’ efficient detection and enumeration in large-scale graphs. Among the most notable contributions we cite Schank and Wagner, who provided both an experimental study [17] and a technical report [18] detailing comprehensive approach for identifying and counting triangles in large graphs. Latapy extended this work by proposing specialized methods for handling triangle computation in massive, sparse graphs characterized by power-law degree distributions [4-5]. Cycle detection, particularly cycles of specific lengths, represents another fundamental problem in graph theory. Seminal contributions from Itai and Rodeh on the identification of minimum circuits in graphs [19], alongside the work of Alon, Yuster, and Zwick (AYZ) on detecting and counting cycles of specific lengths [20], [21], have led the groundwork for the development of efficient cycle-detection algorithms. These techniques have wide-ranging applications, including circuit design and network security. Beyond structural detection, optimizing graph algorithms is imperative for managing the substantial computational and memory challenges posed by large networks. Recent advances in intersection algorithms for sorted lists [22], [23] are particularly relevant for constructing indexes between user and document profiles. These optimized intersection methods accelerate the identification of shared connections and patterns.

Thereby, improving the efficiency of community detection and enhancing our understanding of the structural dynamics within large networks. This paper extends our previous work [24], where we tackled the problem of triangle detection by adjusting the Test With Jumps (TWJ) algorithm proposed in [22], [23]. We refine in this paper Test With Jumps for Triangle detection (TWJtr) [24], by addressing the duplication issue known to be a major flaw; this new method is called Test With Jumps for Triangle detection without Duplicates (TWJtr-D). We have shown that TWJtr accelerates the computation of intersections between sorted neighbors’ lists of the graph’s nodes, thereby enabling efficient triangle detection and counting. The results obtained will then be compared to those of other selected state-of-the-art methods. The rest of the paper is structured as follows. We first go through the most relevant methods previously developed dealing with triangle detection issue in section 2. Considering the lists intersection and sequential processing methodology as core concept, we present, in section 3, two main and largely used methods. Our approach will be detailed in section 4, where we present TWJ the basic method and TWJtr its extended version. The fact that duplicated triangles occur during the detection process, we adapt the later algorithm, TWJtr, and describe TWJtr-D, a refined version which avoids duplications while exploiting the TWJtr’s strengths. The results derived from experiments applied to a collection of datasets are presented and discussed in section 5. Finally, we conclude in section 6 highlighting the most effective method for addressing our problem, and define future perspectives and objectives.

II. RELATED WORKS

The fast growth of the internet with the invasion of social networks and the enormous amount of data circulating online, detecting communities and user profiles became an important task to manage. As a matter of fact, knowing which profiles are more receptive or likely to be interested in a certain information, or which ones are in possession of a requested information so it can be extracted, minimizes the time and data loss; a gain that cannot be ignored in many fields. To achieve this goal, many works have been developed over the years. In 1977, Freeman [25], [26] introduced the formula for betweenness centrality, a measure indicating the importance of a vertex “ i ” of the network in forwarding information. It is calculated by counting the number of shortest paths connecting each pair of vertices that pass through “ i ”, the higher this value, the more relevant the node is. Later, in 2002, the Girvan & Newman method (GN) [27] extended the concept of betweenness centrality to edges, introducing edge betweenness.

Similar to node betweenness, it measures an edge's importance in the network's information flow by calculating the number of shortest paths connecting each pair of vertices that pass through the edge "e". The authors used this measure as criterion for a reverse hierarchical partitioning. Another work was published in 1998 [28] introducing a multilevel partitioning method based on the k-way. The authors aimed to improve speed and partitioning quality. Their method includes three steps that transformed the original graph into a smaller one coarsening it, dividing it to finally refining it back to its original form saving the obtained communities. In 2006, Newman [29], [24] introduced a new formula which became one of the most used ones in this field [30], [31]. Based on the community definition, the modularity was a measure used to evaluate the quality of a community acquired from a specific partitioning. Alongside community detection, another aspect of graph theory emerged as a research spotlight. Being the smallest structure of a community, triangles detection became an extremely active research field attracting the attention of a large number of researchers. Depending on the definition of a triangle, several methods have been developed. The most intuitive method was the brute-force procedure [4] where, for every possible triplet of the graph, a triangle was returned if the three arcs connecting each pair of nodes exist. Nevertheless, this method was the less efficient being extremely slow and considering nodes that are not even each other's neighbors.

In [4] an algorithm was developed based on arboricity, which is a sparsity measure computed by the minimum number of forests needed to cover all edges. The idea was to sort the graph's nodes decreasingly according to their degrees, and check if each pair of a node's neighbors is linked to count a triangle. An additional step was included to avoid duplications where a neighbor must be among the remaining nodes to ensure it wasn't treated beforehand. [4-17] cite the matrix product $A^3(G)$ of the graph's adjacency matrix $A(G)$, the diagonal of the resulting matrix would count twice the number of triangles to which each vertex contributes. However, this method has some limitations, given the dynamic characteristics and width of the graphs, the creation of the adjacency matrix $A(G)$, which is a hollow structure that needs to be constantly updated, takes quite a lot of computation time, as well as the computation of the product $A^3(G)$, which has $O(n^3)$ complexity that cannot be disregarded. The node-iterator (or vertex listing) [4-19] is a procedure that detects a triangle for each node v having two nodes $(u, w) \in \text{Adj}(v)$ connected by an arc. Being quite convenient, the node-iterator has a flaw. We have noticed that the same triangle is detected several times which leads to a duplication problem, a major issue in terms of computing time and memory when applied on large-scale graphs. Similarly for the edge-iterator (or edge listing) method [4-19], the replication of detected triangles is also present where, for each common neighboring node of the vertices of an arc (u, v) , a triangle is identified.

In [4-21] the AYZ algorithm counts the number of triangles in a graph by dividing it into two sub-graphs according to $k = \sqrt{m}$, where m is the number of edges, then applying the node-iterator without duplications on the sub-graph having a degree $d(v) \leq k$, and the edge-iterator on the remaining sub-graph [4]. [17] introduces the Forward algorithm where, for each node v , an initially empty structure $A(v)$ is associated to it. A triangle is detected for each element of $A(v) \cap A(t)$ where $v < t \wedge t \in \text{Adj}(v)$. In [21] the authors use a method based on marking the nodes neighboring a vertex v to detect the triangles formed by the latter, which they later, unmark in order to eliminate duplications. Ding et al [32] present a differential privacy approach for triangle counting in large graphs, introducing a novel algorithm that balances privacy and accuracy. Their method effectively counts triangles while maintaining differential privacy, showing competitive performance on large datasets. However, the approach has limitations, including potential high computational overhead and limited exploration of parameter impact on privacy-accuracy trade-offs. Notably, methods ensuring differential privacy may introduce noise or perturbations to guarantee confidentiality, which can reduce the precision of the results.

In [33], a strongly subcubic combinatorial algorithm for triangle detection is introduced, significantly enhancing time complexity compared to existing methods, achieving a runtime of $O(n^{\frac{7}{3}})$. The algorithm offers robust theoretical performance guarantees and is particularly effective for large graphs. However, its practical applicability may be constrained by increased memory usage due to edge partitioning, especially in highly dense graph regions where vertices can appear in multiple subsets. This results in higher memory overhead and potentially longer processing times for dense areas. In their survey [34], authors review distributed graph algorithms for massive graphs, with a focus on triangle detection methods. They highlight that traditional triangle detection approaches, including intersection-based methods, encounter scalability and memory issues in large graphs. Distributed algorithms, however, significantly improve performance by utilizing parallelism and distributed computing resources, effectively addressing these limitations. The authors conclude that distributed techniques are well-suited for handling massive and dense graph data, providing enhanced efficiency and scalability compared to conventional methods.

III. BASELINE METHODS

Amid the long list of the developed algorithms for triangles detection, we will, in our work, focus on two main methods to which we will be comparing our own approach. In the following section, we will detail these two algorithms discussing their limits and complexity.

III.1 NODE-ITERATOR (VERTEX LISTING)

As mentioned in the previous section, the node-iterator is a method that aims to count, for each node of the graph, the number of nodes that are neighbors to it and that are in turn neighbors (Algorithm1). Although this method appears to be very effective, it has a few shortcomings. Indeed, the major problem can be summed up in two points:

- Duplication.
- Computing complexity.

Algorithm 1: Node-iterator (vertex listing).

Input: list and adjacency matrix representing the network $G(V, E)$
Output: the triangles detected in G

```

1: Count = 0
2: Browse  $G$ 
3: for each  $v \in V$  do
4:   for each couple  $(u, w) \in \text{Adj}(v)$  do
5:     if  $(u, w) \in E$  then  $(v, u, w)$  is a triangle
6:       Count++
7:     end if
8:   end for
9: end for
10: return Count

```

Source: Authors, (2026).

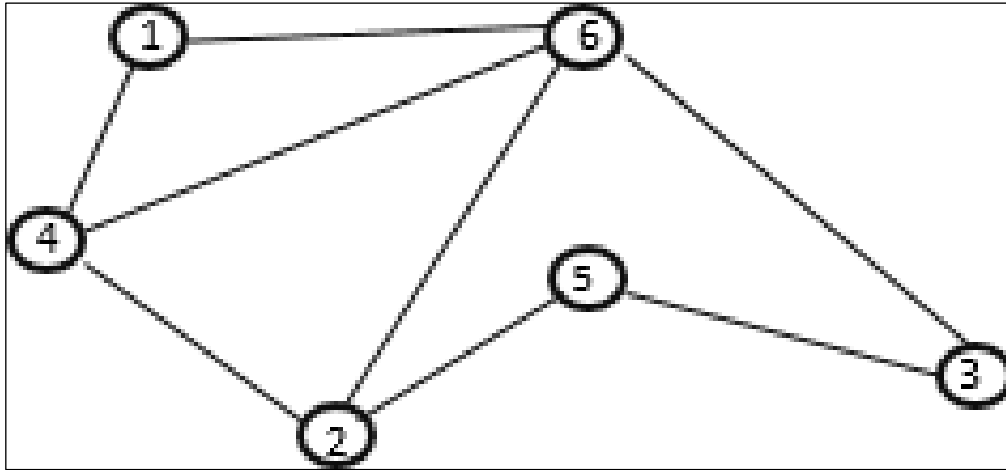


Figure 1: Example of a non-oriented graph.

Source: Authors, (2026).

We will illustrate the node-iterator on the graph sample in figure 1, where nodes 2, 4 and 6 form, visually, a triangle. By iterating through each vertex's adjacency list and verifying connections between every node couple using the adjacency matrix, the same triangle (2, 4, 6) is detected multiple times, once from each vertex involved. This leads to duplication, as the same triangle is reported three times. In large graphs, such redundancy significantly increases computational cost. The node-iterator approach has a time complexity of $O(|V|^3)$ and a space complexity of $O(|V|^2)$. While effective for dense graphs due to adjacency matrix usage, it is inefficient for sparse graphs compared to algorithms with complexity linear in the number of edges $|E|$. Recalling that V is the set of nodes and E is the set of edges.

III.2 EDGE-ITERATOR (EDGE-LISTING OR NAÏVE ALGORITHM)

This algorithm represents the basic idea that defines a triangle. Effectively, it consists in searching, for an arc of the graph, a common neighboring node. In other words, the edge-iterator processes an intersection calculation between the adjacency lists of two nodes forming an arc on the graph and returns a triangle for each common element detected (Algorithm 2).

Algorithm 2: Edge-iterator (Edge listing/ Naïve method).

Input: adjacency list and matrix of the graph $G(V, E)$
Output: detected triangles in G

```

1: Count = 0
2: Browse  $G$ 
3: for each  $(u, v) \in E$  do
4:   if  $\text{Adj}(u) \cap \text{Adj}(v) \neq \emptyset$  then
5:     for each  $w \in \text{Adj}(u) \cap \text{Adj}(v)$  do
6:       Triangle  $(u, v, w)$ 
7:       Count++
8:     end for
9:   end if
10: end for
11: return count

```

Source: Authors, (2026).

Using the same example, Figure 1, the edge-iterator algorithm examines each edge and computes the intersection of the adjacency lists of its endpoints to identify triangles. For instance, starting with edge (2, 6), the intersection $\text{Adj}(2) \cap \text{Adj}(6)$ yields vertex 4, forming the triangle (2, 4, 6). The same triangle is repeatedly found when processing related or reversed edges, thus increasing complexity. The edge-iterator runs in $O(|E| \cdot \Delta)$ time and $O(|V| + |E|)$ space, performing well on sparse graphs but poorly on dense graphs with high maximum vertex degree Δ .

IV. PROPOSED APPROACH

Having noticed that the detection of triangles in graphs can be considered as a problem of intersection calculation between lists of neighbors of a given node of the graph, we adapt the TWJ [22], [23] which is basically dedicated to the fast calculation of the intersection between sorted lists so that it can handle this calculation. Our choice for using adjacency lists was made after concluding that it was better in contrast with adjacency matrix which is an immense hollow structure for real graphs consuming enormous memory space and time computing for their update considering the graphs' dynamic nature.

IV.1 TWJ AND DATA STRUCTURE

Before discussing the TWJ's functioning for the detection of triangles, we must first go through the data structure it handles, which is quite particular as it appears in figure 2. Indeed, the TWJ is based on the processing of lists transformed into sequences, where each sequence is composed of three elements as follows:

- SeqID: identifier of the sequence obtained by calculating the modulo of the i^{th} element of the list by a sequencing value chosen beforehand (if the elements of the list are integers).
- Nseq: represents the number of elements that belong to the sequence seqID or the length of the sequence. For a list of integer type, the Nseq is the number of elements belonging to the interval between $\text{seqID} * \text{sequencing value}$, and, $(\text{seqID} + 1) * \text{sequencing value}$.
- Elements of the sequence: is none other than the N elements counted in the NSeq which are ordered and listed until the appearance of a (possible) next sequence.

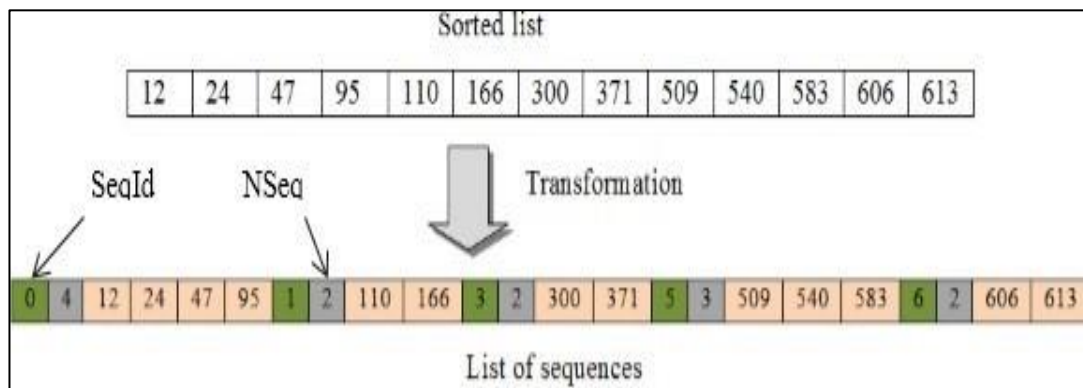


Figure 2: Sequence list creation for a sequencing value= 100.

Source: Authors, (2026).

Once the sequence lists created, the TWJ is applicable

IV.2 TWJ ALGORITHM

In [22-23] the TWJ is described as a method which takes as input two sorted lists transformed into sequences according to a precise and identical sequencing value, the TWJ compares, starting from the first box, sequentially the elements of the two lists, having equal seqIDs, and returns each common value detected until having reached the size of this sequence, applying the naive algorithm (Algorithm 4) for the calculation of the intersection, and passes to the next one for the two lists. On the other hand, if the current seqIDs of the two lists are different, one passes to the next sequence of the list whose seqID is lower and compares it again with that of the other list that has not changed and so on until two equal seqIDs are obtained, or, that one of the two lists has reached its size (exhausted all its seqIDs).

Algorithm 3: Test With Jumps (TWJ).

Input: Integer intersection (Integer[] SeqA, Integer[] SeqB, Integer sizeA, Integer sizeB, short* C)

```

1: Integer i = 0, j = 0
2: while i < sizeA ∧ j < sizeB do
3:   if the two compared seqID are equal then
4:     Apply the naïve intersection algorithm between the two sequences (Algorithm 4)
5:   else
6:     Move on to the next sequence in the table with the smallest seqID
7:   end if
8: end while

```

Source: Authors, (2026).

Algorithm 4: Naive intersection computation

Input: integer intersection(integer[] A, integer[] B, integer sizeA, integer sizeB, short[] C)

Output: common elements

```

1: integer count = 0, i = 0, j = 0;
2: while i < sizeA and j < sizeB do
3:   if A[i] == B[j] then
4:     C[count++] = A[i];
5:     i++; j++;
6:   else if A[i] > B[j] then
7:     j++;
8:   else
9:     i++;
10:  end if
11: end while
12: return count;

```

Source: Authors, (2026).

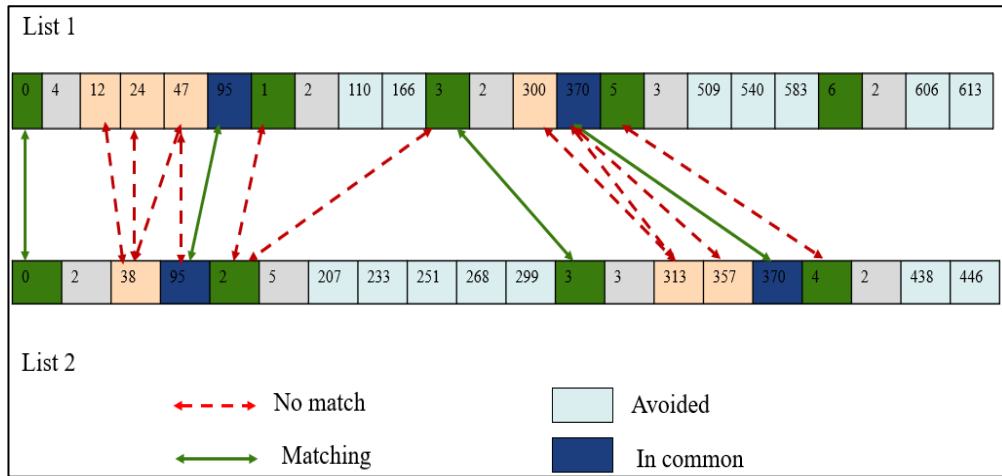


Figure 3: TWJ algorithm process.

Source: Authors, (2026).

Figure 3 shows an example of TWJ processing over two sequence lists. It is clearly noticeable that the TWJ's logic permits avoiding unnecessary comparisons highly gaining in terms of computation time. Thus, applying TWJ over large scale datasets improves significantly time complexity.

IV.3 TWJ AND TRIANGLE DETECTION

Returning to our primary case study, which is triangle detection, the approach we propose involves extending the TWJ method. For each node $u \in \text{Adj}(v)$ where $v \in G$, we apply it to the adjacency lists of nodes v and u , which have been previously transformed into sequences. Consequently, each newly identified element x signifies the presence of an existing triangle (uvx) within the graph (see Algorithm 5).

Algorithm 5: TWJ triangle detection (TWJtr)

Input: adjacency list of the network G

Output: detected triangles

```

1: Count = 0;
2: for each  $v \in G$  do
3:   for every  $u \in \text{Adj}(v)$  do
4:     Compute the intersection  $\text{Adj}(v) \cap \text{dj}(u)$  by applying the TWJ algorithm
5:     for each element returned do
6:       Count++;
7:     end for
8:   end for
9: end for
10: return Count; {The list of triangles if desired}

```

Source: Authors, (2026).

In terms of complexity, the TWJtr has a worst-case time complexity of $(|V|^3)$ for dense graphs and $O(|E| \cdot \Delta)$ for sparse graphs, and $(|V| + |E|)$ space complexity.

IV.4 RESOLVING THE DUPLICATION ISSUE IN TRIANGLE DETECTION

After analyzing the field's existing methods and experiencing them multiple times, we were able to see firsthand the number of duplicated triangles that were detected and its heavy impact on the algorithms' performances. Therefore, resolving the duplication issue became an essential objective our study aimed for. To achieve this goal, we considered both the dataset's structure and the TWJtr process. Giving the graph's non-oriented nature, we modified the basic TWJtr by including an additional treatment which allows marking nodes that have been processed. In other words, every time a node v is treated and all triangles involving it are detected, v would be marked as an explored node. Thereafter, only the unmarked neighbors of the remaining nodes would be considered for intersection computing thus detecting triangles formed by these couples whereas triangles formed by couples with marked nodes were already detected when the marked node was processed. Thus, returning duplicated triangles is avoided and time complexity drops significantly. The pseudocode of algorithm 6 presents the algorithm TWJtr-D for eliminating duplications.

Algorithm 6: TWJtr without duplicates (TWJtr-D).

Output: detected triangles <pre> 1: Count = 0; 2: for each $v \in G$ do 3: for every $u \in \text{Adj}(v)$ do 4: Compute the intersection $\text{Adj}(v) \cap \text{Adj}(u)$ by applying the TWJ algorithm if neither u nor v were explored (unmarked) 5: for each element returned do 6: Count++; 7: end for 8: end for 9: mark(v) 10: end for 11: return Count; {The list of triangles if desired}</pre>	Input: adjacency list of the network G
--	---

Source: Authors, (2026).

V. EXPERIMENTS AND RESULTS

The experimentations were conducted on an Intel(R) Core™ i7-8700K CPU @3.70 Ghz, 64 GB of RAM and two Hard discs: 500GB VME and 6TB HDD. We used Java Netbeans as programming environment.

V.1 TEST DATA

We performed our tests over the below twelve real graphs downloaded in (.txt) format that we load in memory.

Table 1: Experimented datasets.

Graph name	$ V $ = number of vertices	$ E $ = number of edges
New sites	1040	313869
Hr	1099	498202
Hu	1054	222887
Ro	1035	125826
Tv shows	588	26527
Company	1033	131723
Politicians	660	53621
CAH	9876	51969
Public figures	1012	133535
Ego Fb	3662	172776
Artists	1046	570398
Athletes	1088	186013
Government	793	88005

Source: Authors, (2026).

V.2 COMPARATIVE STUDY

We compare our approach, the TWJtr, to the node-iterator and edge-iterator algorithms. This comparison is based on:

- The number of tests performed by each method when detecting triangles excluding the search for elements in the adjacency list.
- The number of triangles detected (including repetitions).
- Calculation time.

The experimentations conducted, over the twelve datasets mentioned above, have all expressed the same behavior in response to the algorithms we applied. In order to present graphically these results for each comparison criterion, the number of graphics would be inconvenient to expose in this section of our article. Therefore, among the acquired results, we selected those of the Hr, Hu and Ro datasets to be demonstrated. These datasets, the Hr, Hu and Ro, were collected from the music streaming service Deezer.

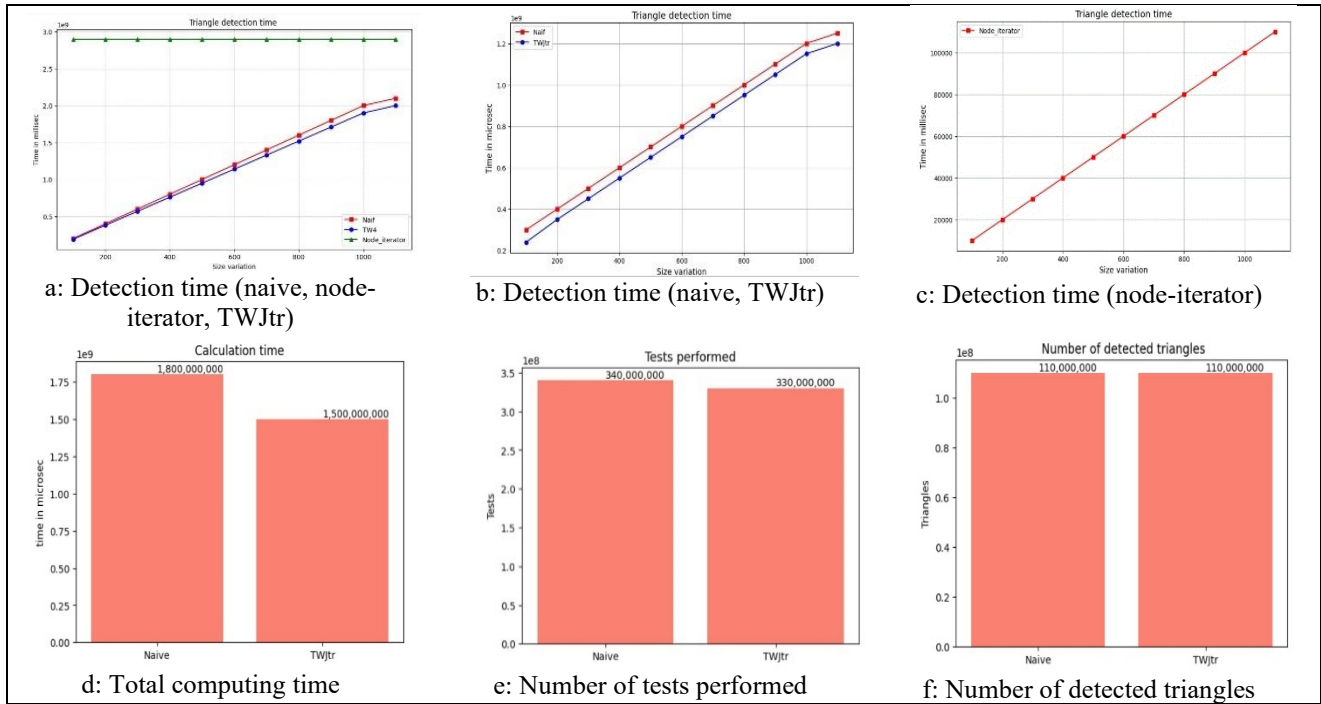


Figure 4: RO dataset's statistics.
Source: Authors, (2026).

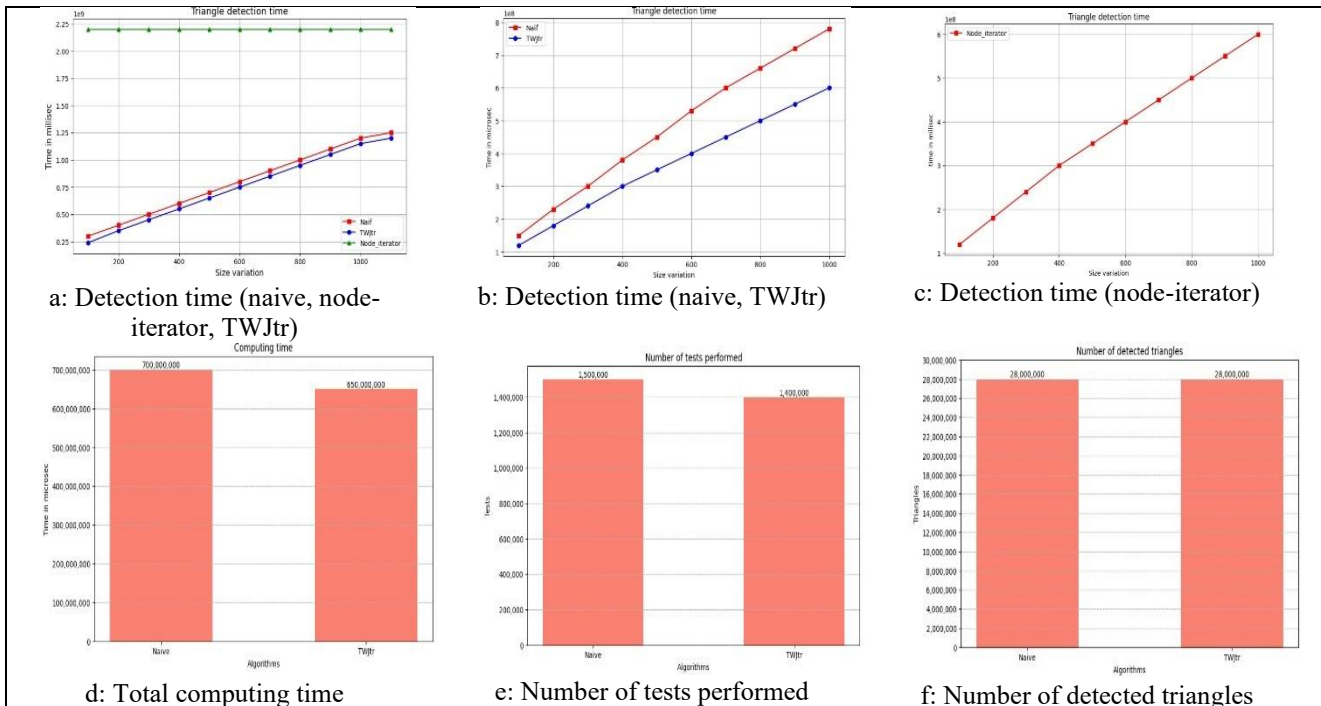


Figure 5: HU dataset's statistics
Source: Authors, (2024).

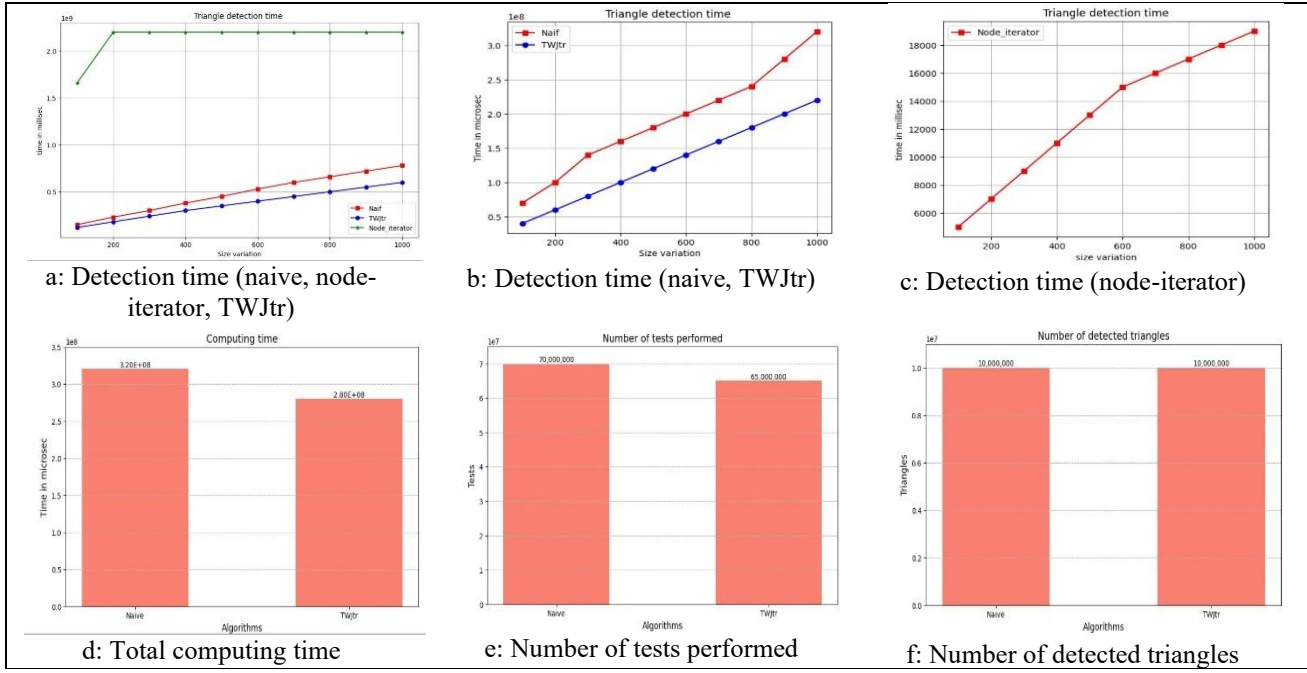


Figure 6: HR dataset's statistics

Source: Authors, (2026).

After conducting experiments on the aforementioned datasets, performance statistics were collected according to the previously defined comparison criteria. The obtained results are illustrated in Figure. 4, 5, and 6. Figure. (4. a), (5. a), and (6. a) depict the execution time evolution of the three evaluated algorithms. The results show that the Node-Iterator algorithm exhibits the highest execution time across all datasets. The Naive algorithm (Edge-iterator) achieves lower execution times, while the TWJtr algorithm consistently outperforms both baseline approaches. For a more detailed analysis, the execution time curves shown in Figure. (4.a), (5.a), and (6.a) are decomposed into Figure. (4.b), (5.b), (6.b) and Figure. (4.c), (5.c), (6.c). Figure. (4.c), (5.c), and (6.c) report the execution time of the Node-Iterator algorithm in milliseconds, indicating that its runtime remains significantly higher than that of the other two methods. In contrast, figure. (4.b), (5.b), and (6.b) compare the Naive and TWJtr algorithms in nanoseconds, where TWJtr achieves lower execution times across all datasets. Figure. (4.d), (5.d), and (6.d) present the total execution time required to complete triangle detection. These results show that TWJtr requires less total computing time than the Naive algorithm across all datasets. Additional efficiency-related statistics are presented in Figure. (e) and (f). Figure. (e) reports the number of intersection tests performed by the Naive and TWJtr algorithms, while Figure. (f) shows the number of triangles detected. The results indicate that both algorithms return the same number of detections, despite performing a different number of intersection tests.

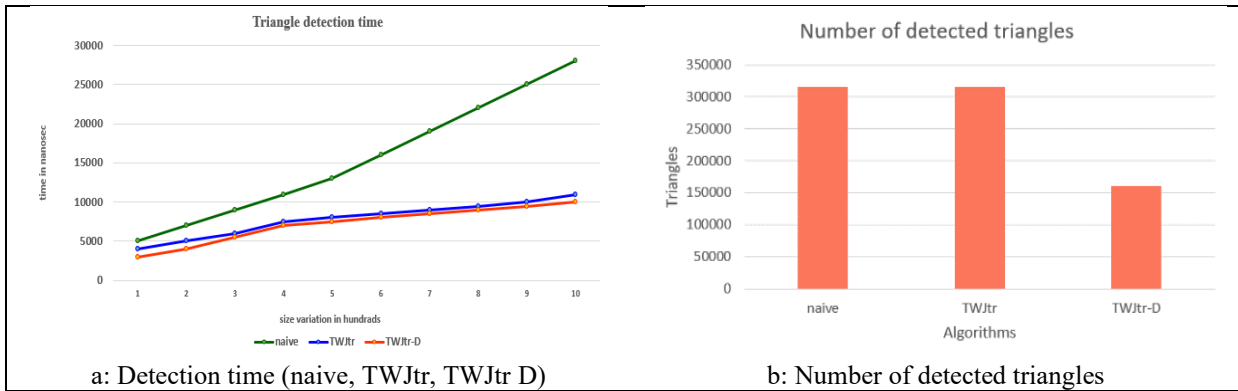


Figure 7. TWJtr-D (TWJtr without duplication) statistics over a graph of 1000 nodes

Source: Authors, (2024).

Figure. 7 presents the results obtained after applying the TWJtr-D algorithm and comparing it with the Naive and TWJtr methods. The execution time comparison illustrated in Figure. (7.a) shows that TWJtr-D achieves the lowest computing time among the three evaluated algorithms. Figure. (7.b) reports the number of triangle detections for each method. The results show that TWJtr-D returns a lower number of detections corresponding to unique triangles, whereas the Naive and TWJtr algorithms produce identical counts due to duplicate detections.

V.3. DISCUSSION

The poor performance of the Node-Iterator algorithm observed in Figure. 4, 5, and 6 can be attributed to its repeated verification process, which checks the existence of edges between all pairs of neighboring nodes. This procedure relies on frequent scans of adjacency lists, resulting in substantial computational overhead.

The Naive algorithm improves execution time by computing intersections between adjacency lists, enabling faster identification of shared neighbors. However, the TWJtr algorithm further optimizes this process by introducing jump-based traversal over unmatched segments of adjacency sequences. This optimization significantly reduces unnecessary comparisons, which explains the consistent performance gains observed for TWJtr in Figure. (4.b), (5.b), and (6.b), as well as its reduced total execution time reported in Figure. (4.d), (5.d), and (6.d). The analysis of Figure. (e) and (f) further confirms this behavior, showing that TWJtr performs fewer intersection tests while detecting the same number of triangles as the Naive algorithm. This demonstrates that the observed performance improvements are achieved without affecting detection completeness. Building upon these results, TWJtr-D introduces an additional duplication-aware mechanism designed to avoid redundant processing of previously examined nodes. This extension explains the further reduction in execution time observed in Figure. (7.a). The lower number of detections reported in Figure. (7.b) reflects the elimination of duplicate triangle detections rather than a loss of correctness. This outcome results from TWJtr-D verifying in each iteration, whether the vertices involved in a test have already been processed which preserves triangle detection completeness by reporting each triangle exactly once. This behavior is consistent with the duplication-aware strategy described in the PROPOSED APPROACH section and demonstrates that the proposed method, TWJtr-D, effectively reduces computational cost while maintaining correct and complete triangle detection. Moreover, the performance gains become more pronounced as graph size increases, indicating improved scalability compared to baseline methods.

VI. CONCLUSION

Considering the immense data traffic flowing on the networks, created by interactions exchanged by a very large number of network users, detecting communities and actor profiles became essential for several reasons. Among them, routing queries for targeted audience, which allows directing information to clusters of individuals sharing the same interests with the said query, a strategy that is, nowadays, the trend of commercial marketing and advertisement through social networks. The detection of communities involves the detection of triangles since triangles represent the basic component of a community. In this context, we built our approach over the TWJ algorithm [22-23] which is defined for the fast calculation of the intersection between sorted lists. We have adapted TWJ to the context of triangle detection that we named TWJtr [24]. Our solution presented a much better performance in terms of time computing, number of preformed tests and number of detected triangles. These results were obtained through comparing our solution with two of this domain's basic algorithms, the node-iterator and the edge-iterator (naive method), chosen after a detailed study of this field's state of the art. Given the considerations above, TWJtr is found to be the best general-purpose algorithm which balances efficiency and practicality, performing well on both sparse and moderately dense graphs, leveraging efficient intersection computations to detect triangles. Additionally, we enhanced the TWJtr to be further optimal. Since the previous methods generated results including a considerable number of duplicated triangles, TWJtr-D, as a refined version of TWJtr, ensures an accurate result eliminating duplicated triangles optimizing time computing, which is an undeniable advantage considering the large-scale networks subjected to triangle detection procedures. In the future, we intend to implement and experiment on platforms like SPARK in order to adapt TWJtr and TWJtr-D for distributed systems. We also, consider comparisons with other algorithms.

VII. AUTHOR'S CONTRIBUTION

Conceptualization: Balbouzi Sarah.

Methodology: Balbouzi Sarah.

Investigation: Mechach Kheira.

Discussion of results: Balbouzi Sarah and Zekri Lougmiri.

Writing – Original Draft: Balbouzi Sarah.

Writing – Review and Editing: Zekri Lougmiri and Mechach Kheira.

Resources: Zekri Lougmiri.

Supervision: Zekri Lougmiri.

Approval of the final text: Balbouzi Sarah, Zekri Lougmiri and Mechach Kheira.

VIII. ACKNOWLEDGMENTS

This work was funded by the University Oran1 Ahmed Ben Bella, Grant No. C00L07UN310120220006.

IX. REFERENCES

- [1] L. Euler, "Solutio problematis ad geometriam situs pertinentis (The solution of a problem relating to the geometry of position)," *Mém. Acad. Sci. Berlin*, 1735. doi: 10.3931/euler-1
- [2] J. L. Moreno, "Who Shall Survive? A New Approach to the Problem of Human Interrelations," Washington, DC: Nervous and Mental Disease Publ. Co., 1934. doi: 10.1037/h0057750
- [3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, pp. 509–512, 1999. doi: 10.1126/science.286.5439.509
- [4] M. Latapy, "Main-memory triangle computation for very large (sparse (power-law)) graphs," *Theor. Comput. Sci.*, vol. 407, pp. 458–473, 2008. doi: 10.1016/j.tcs.2008.07.009
- [5] P. Erdős and A. Rényi, "On random graphs. I," *Publ. Math.*, vol. 6, pp. 290–297, 1959. doi: 10.1007/BF02781481
- [6] S. Milgram, "The small-world problem," *PsycEXTRA Dataset*, 1967. doi: 10.1037/e300292003-001
- [7] F. Karinthy, "Láncszemek [Chains] (Everything is Different)," Budapest: Atheneum, 1929, pp. 85–90.
- [8] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998. doi: 10.1038/30918

- [9] H. A. Simon, "On a class of skew distribution functions," *Biometrika*, vol. 42, no. 3–4, pp. 425–440, 1955. doi: 10.2307/2333389
- [10] D. J. de S. Price, "A general theory of bibliometric and other cumulative advantage processes," *J. Amer. Soc. Inf. Sci.*, vol. 27, no. 5, pp. 292–306, 1976. doi: 10.1002/asi.4630270505
- [11] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin, "Size-dependent degree distribution of a scale-free growing network," *Phys. Rev. E*, vol. 63, 062101, 2001. doi: 10.1103/PhysRevE.63.062101
- [12] G. Bianconi and A.-L. Barabási, "Competition and multiscaling in evolving networks," *Europhys. Lett*, vol. 54, no. 4, pp. 436–442, 2001. doi: 10.1209/epl/i2001-00455-0
- [13] S. Boccaletti et al., "The structure and dynamics of multilayer networks," *Phys. Rep.*, vol. 544, pp. 1–122, 2014. doi: 10.1016/j.physrep.2014.07.001
- [14] P. Holme and J. Saramäki, Eds., "Temporal Network Theory," Springer, 2019. doi: 10.1007/978-3-030-26047-5
- [15] F. Battiston et al., "The physics of higher-order interactions in complex systems," *Nature Phys.*, vol. 16, pp. 585–594, 2020. doi: 10.1038/s41567-020-0852-7
- [16] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, 1985. doi: 10.1137/0214015
- [17] T. Schank and D. Wagner, "Finding, counting and listing all triangles in large graphs: An experimental study," *Proc. Workshop Exp. Efficient Algorithms (WEA)*, 2005. doi: 10.1007/11427186_16
- [18] T. Schank and D. Wagner, "Finding, counting and listing all triangles in large graphs," *Tech. Rep.*, Universität Karlsruhe, Fakultät für Informatik, 2005.
- [19] A. Itai and M. Rodeh, "Finding a minimum circuit in a graph," *SIAM J. Comput.*, vol. 7, no. 4, pp. 413–423, 1978. doi: 10.1137/0207032
- [20] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Proc. Eur. Symp. Algorithms (ESA)*, 1994. doi: 10.1007/3-540-58484-6_25
- [21] N. Alon, R. Yuster, and U. Zwick, "Finding and counting given length cycles," *Algorithmica*, vol. 17, no. 3, pp. 209–223, 1997. doi: 10.1007/BF02523654
- [22] L. Zekri, F. Manseur, and O. Belhadj, "Calcul de l'intersection entre listes triées à base de sauts (Intersection computation between sorted lists using jumps)," *Proc. EDA, Tanger, Maroc*, 2018, pp. 57–72.
- [23] F. Manseur, L. Zekri, and M. Senouci, "A new fast intersection algorithm for sorted lists on GPU," *J. Inf. Technol. Res. (JITR)*, vol. 15, no. 1, pp. 1–20, 2022. doi: 10.4018/JITR.2022010101
- [24] Our work, "title", yyyyyyyyyyyy, zzzz, doi:xxxxxxxxxxxxxxxx.
- [25] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977. doi: 10.2307/3033543
- [26] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociol.*, vol. 25, no. 2, pp. 163–177, 2001. doi: 10.1080/0022250X.2001.9990249
- [27] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *PNAS*, vol. 99, no. 12, pp. 7821–7826, 2002. doi: 10.1073/pnas.122653799
- [28] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM*, 1998. doi: 10.1137/1.9781611972873.ch11
- [29] M. E. J. Newman, "Modularity and community structure in networks," *Proc. Natl. Acad. Sci. USA (PNAS)*, vol. 103, no. 23, pp. 8577–8582, Jun. 2006. doi: 10.1073/pnas.0601602103
- [30] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *J. Stat. Mech.*, 2008. doi: 10.1088/1742-5468/2008/10/P10008
- [31] P. De Meo, E. Ferrara, G. Fiumara, and A. Proveti, "Generalized Louvain method for community detection in large networks," *Proc. 11th Int. Conf. Intell. Syst. Design Appl.*, 2011. doi: 10.1109/ISDA.2011.6064084
- [32] X. Ding, S. Sheng, H. Zhou et al., "Differentially private triangle counting in large graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5278–5292, 2021. doi: 10.1109/TKDE.2020.3014390
- [33] A. Dumitrescu, "A strongly subcubic combinatorial algorithm for triangle detection with applications," *arXiv preprint*, arXiv:2403.01085, 2024. doi: 10.48550/arXiv.2403.01085
- [34] L. Meng et al., "A survey of distributed graph algorithms on massive graphs," *arXiv preprint*, arXiv:2403.xxxxx, 2024. doi: 10.48550/arXiv.2403.xxxxx