


# QClique: Optimizing Performance and Accuracy in Maximum Weighted Clique

Qasim Abbas<sup>1,2</sup><sup>[0009-0008-3034-2825]</sup>,  
Mohsen Koohi Esfahani<sup>3,1</sup><sup>[0000-0002-7465-8003]</sup>,  
Ian Overton<sup>1</sup><sup>[0000-0003-1158-8527]</sup>, and  
Hans Vandierendonck<sup>1</sup><sup>[0000-0001-5868-9259]</sup>

<sup>1</sup>Queen’s University Belfast, UK, <sup>2</sup>University of Westminster, UK  
<sup>3</sup>Burge Systems, Iran  
{qabbas01,mkoochiesfahani01,i.overton,h.vandierendonck}@qub.ac.uk

**Abstract.** The Maximum Weighted Clique (MWC) problem remains challenging due to its unfavourable time complexity. In this paper, we analyze the execution of exact search-based MWC algorithms and show that high-accuracy weighted cliques can be discovered in the early stages of the execution if searching the combinatorial space is performed systematically. Based on this observation, we introduce QClique as an approximate MWC algorithm that processes the search space as long as better cliques are expected. QClique uses a tunable parameter to trade-off between accuracy vs. execution time and delivers 4.7–82.3× speedup in comparison to previous state-of-the-art MWC algorithms while providing 91.4% accuracy and achieves a parallel speedup of up to 56× on 128 threads. Additionally, QClique accelerates the exact MWC computation by replacing the initial clique of the exact algorithm. For WLMC, an exact state-of-the-art MWC algorithm, this results in 3.3× on average.

**Keywords:** Graph algorithms · High-performance computing · Maximum weighted clique · Approximate algorithms.

## 1 Introduction

The Maximum Weighted Clique (MWC) is a graph analytic problem with applications in domains such as social sciences and bioinformatics. To accelerate processing this NP-hard problem, various approximate and exact algorithms have been proposed [1–6], however, by the fast growth of data, faster algorithms are required for better exploration of the search space which grows exponentially as a function of the size of the graph. Maximal Clique Enumeration (MCE) may be used for this purpose. MCE identifies cliques that cannot be extended to a larger clique by adding more vertices. A basic approach to MWC is to generate all maximal cliques and identifying among those the largest-weight clique.

To accelerate the search, pruning rules are designed to avoid accessing unnecessary parts of the search space. The pruning rules are exact when it is mathematically certain that the MWC is not pruned, and they are approximate when the MWC may be pruned. The exact solutions [4–6] have long execution

times as they need to search through the majority of the search space. The approximate algorithms [1–3], on the other hand, introduce specific problems:

- Experimental evaluation shows that approximated searches may neither converge quickly nor reach high-accuracy solutions. The search is often ad hoc or informed by random selection; accordingly, high-accuracy weighted cliques (WC) (i.e., cliques with weights close to MWC) may be missed.
- Approximate algorithms require additional operations (such as graph reduction [4, 7] and estimating the best candidate to join a clique [8]) that increase memory consumption, increase execution time, and limit scalability.
- Heuristic pruning of the search space makes the discovery of high-accuracy WC dependent on the processing order of vertices [9, 10].

To counter the above problems, this work introduces a novel pruning heuristic for efficiently finding MWC in specific regions of interest. The heuristic is based on two key observations: (1) Early in the search, high-accuracy MWC are identified due to the utilisation of degree or degeneracy ordering<sup>1</sup>, leading to an opportunity for reducing the search space by validating no larger-weight cliques exist. (2) While high-degeneracy vertices are visited more frequently during the search, they often do not contribute to discovering WC with higher weights.

We introduce the approximate MWC algorithm **QClique** that dynamically truncates the search by limiting the frequency of processed vertices. QClique flexibly adjusts this limit based on (i) the execution status and (ii) a tunable parameter that controls the trade-off between execution time and accuracy.

Our evaluation shows that QClique is 4.7–82.3 times faster than state-of-the-art approximate and exact algorithms while presenting 91.4% accuracy, on average. While QClique is an approximate MWC, its fast and high-accuracy result allows to compute the exact MWC faster. By replacing the initial seed of WLMC (a state-of-the-art exact MWC algorithm) with QClique result, the exact MWC computation is accelerated by  $3.3\times$  speedup.

The contributions of this paper are:

- Analyzing the discovery process of MWC and the behavior of vertices.
- Introducing the QClique algorithm, a fast and accurate approximate MWC algorithm that dynamically adjusts the exploration frequency of each vertex.
- Evaluation of QClique in comparison to state-of-the-art MWC algorithms and using various graph datasets.

The structure of this article is as follows. Section 2 reviews the terminology and baseline algorithm. Section 3 motivates the design of QClique. The QClique algorithm is described in detail in Section 4 and is evaluated in Section 5. Further related works are reviewed in Section 6 and Section 7 concludes the discussion.

---

<sup>1</sup> Degeneracy ordering is the order of vertices for dynamically degenerating the graph, i.e., the order of vertices with the lowest degrees when they are removed one after another from the graph. By removing a vertex, the degree of the other vertices may be affected and the order of vertices in the list of remained vertices with the lowest degrees may change.

## 2 Background

### 2.1 Terminology

A graph  $G(V, E)$  has a set of vertices  $V$  and edges  $E$  such that  $E \subseteq V \times V$  (we exclude all self-edges in the product  $V \times V$  as they are irrelevant to clique problems). The set of neighbors of vertex  $v \in V$  is given by  $N(v) = \{u \in V : (u, v) \in E\}$ . A clique  $C(V_c, E_c)$  of the undirected graph  $G(V, E)$  is a fully connected subgraph of  $G$  (i.e.,  $V_c \subseteq V$  and  $E_c \subseteq E$ ) where each two vertices are adjacent, i.e.,  $E_c = V_c \times V_c$ . The Maximum Clique of a graph is the clique with the largest number of vertices. A vertex-weighted undirected graph  $G(V, E, w)$  consists of a set of vertices  $V$ , edges  $E$ , and the weight function  $w : V \rightarrow \mathbb{N}$ . The weight of a clique  $C = (V_c, E_c)$  is given by the weight of its vertices:  $w(C) = w(V_c) = \sum_{v \in V_c} w(v)$ . The Maximum Weighted Clique of a vertex-weighted graph  $G$  is the clique with a maximum weight over all cliques in  $G$ .

The density of a graph is defined as  $d(G) = |E|/|V \times V|$ , where  $|S|$  is the size of set  $S$ . The accuracy of an approximate MWC algorithm for a graph is the proportion of the weight of the largest-found WC to the weight of the MWC of the graph.

### 2.2 Clique Algorithms

There are several clique algorithms, of which MWC is just one. Maximum Clique (MC) searches for the largest-size clique [11]. It is equivalent to MWC with all vertex weights set to one. While similar, MC admits a variety of pruning rules to reduce the search space by considering the size of the clique and the degrees of candidate vertex [12–14]. However, these pruning rules by and large do not apply to MWC, implying that MWC must consider a substantially larger part of the search space. The Maximum Edge-Weighted Clique (MEWC) problem searches for the highest-weight clique where edges carry weights. Maximal Clique Enumeration (MCE) [15, 16] lists all maximal cliques, i.e., the ones that are not a subset of other cliques. MCE serves also as a blue-print for MWC, MWEC and MC algorithms as every largest or highest-weight clique must also be a maximal clique, assuming all weights are non-negative.

### 2.3 Baseline MWC

Algorithm 1 shows the baseline MWC algorithm based on the Bron-Kerbosch MCE algorithm [16] which has been widely used as the template for the state-of-the-art clique algorithms [12, 13, 17–19]. The algorithm recursively grows a clique in the variable  $R$  by selecting one vertex at a time from the candidate set  $P$ . Each time  $R$  grows, the candidate set  $P$  is shrunk by retaining only those candidate vertices that are also a neighbor of the last selected vertex. This ensures that adding any singular vertex from  $P$  to  $R$  will retain the property that  $R$  is a clique. The recursion terminates when the candidate list  $P$  is empty, at which point a clique is found.

The set  $X$  represents excluded vertices and ensures that no maximal clique is reported twice and that no non-maximal cliques are enumerated. A unique maximal clique is found when both  $P$  and  $X$  are empty. If  $P$  is empty and  $X$  is not, the clique held in  $R$  is not maximal or has already been enumerated.

**Algorithm 1** Baseline MWC

---

```

1:  $WC_{max} = \{\}$ 
2: for  $v_i \in V$  do
3:    $P = N(v) \cap \{v_{i+1}, v_{i+2}, \dots, v_{n-1}\}$ 
4:    $X = N(v) \cap \{v_0, v_1, \dots, v_{i-1}\}$ 
5:    $ClqSearch(\{v_i\}, P, X, G)$ 
6: end for
7: return  $WC_{max}$ 

```

---

**Algorithm 2** ClqSearch

---

```

1: function ClqSearch (R, P, X, G)
2: if  $P = \{\}$  then
3:   if  $X = \{\}$  then
4:     if  $w(R) > w(WC_{max})$  then
5:        $WC_{max} = R$ 
6:     end if
7:   end if
8:   return
9: end if
10: for  $v \in P$  do
11:    $P' = P \cap N(v)$ 
12:    $X' = X \cap N(v)$ 
13:    $ClqSearch(R \cup \{v\}, P', X', G)$ 
14:    $P = P \setminus \{v\}$ 
15:    $X = X \cup \{v\}$ 
16: end for
17: return

```

---

It is worth noting that the set  $X$  is not required in solving the MWC problem as a non-maximal clique cannot have a higher weight than the MWC. As such, it is not considered in Lines 4 and 5 of Algorithm 2.

### 3 Motivation

#### 3.1 High-Accuracy Cliques Appear Early in The Enumeration

In this section, we consider the growth in the weight of the largest clique during the execution of MWC search (Section 2.3). We adapted the parallel MCE code of Blanuša et al [18]<sup>2</sup> to track MWC. Figure 1 presents the results for two graphs: keller6, as a denser graph, and p-hat500-01, as a sparser graph. The X-axis shows the relative time passed (in percentage) and the Y-axis shows the weight of the largest-weight clique encountered at that time.

The plots in Figure 1 show that a large MWC growth rate occurs early in the search such that the first WC with weights greater than 80% of the MWC are discovered in less than 10% of the total execution time (weight 61 is found at 9.7% of execution for p-hat500-01, and weight 123 is found at 0.9% of execution for keller6). It is followed by the major portion of the execution to improve the weight of the best-found WC. In the dense keller6 graph, further exploration takes 17,057 times longer than the time it takes to find the first high-accuracy WC. After finding the MWC, further time is spent to establish that no other clique exists with a greater weight. In the keller6 graph, this accounts to 36.5% of the execution time. This demonstrate that **high-weight cliques can be discovered early during search process**.

<sup>2</sup> <https://github.com/IBM/parallel-clique-enumeration>, commit 9d7d8ae

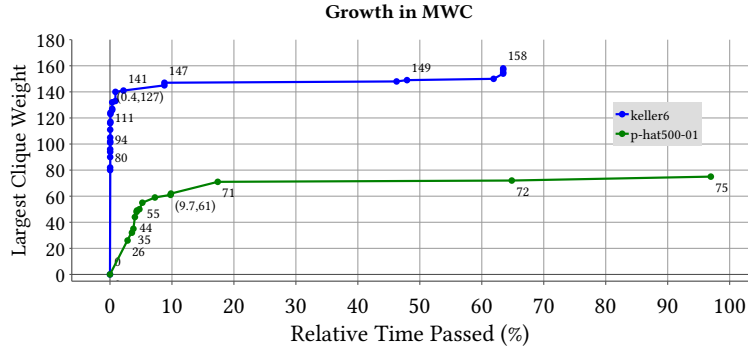


Fig. 1: Growth in MWC during the execution of WLMC

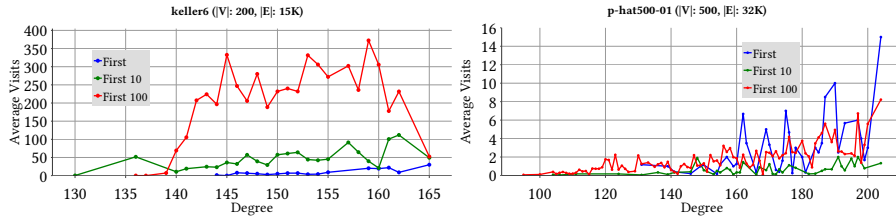


Fig. 2: The number of times vertices are processed before reaching the first, first 10, and first 100 high-accuracy WC (i.e., with weights greater than  $0.8 * \text{MWC}$ ) as a function of their degree. Results for dense (keller6, left) and sparse (p-hat500-01, right) graphs are shown.

### 3.2 High-Accuracy Cliques Require Few Visits of Each Member

A key performance challenge with combinatorial algorithms such as MWC is the consideration of a large number of combinations of vertices during the search; leading to exponential scaling of execution time with graph size. To have a better understanding, we measure the number of times each vertex is processed before a number of high-weight cliques are found. We define high-accuracy WC as WC with weights greater than 0.8 of the weight of the MWC of the graph and we measure the number of visits per vertex (i.e., the number of times the vertex is selected as a member of a clique, see Line 10 of Algorithm 2) during the creation of the first high accuracy WC.

Figure 2 shows the degree distribution of the average number of visits to vertices for the first, first 10, and first 100 high-accuracy WC and for two graphs, keller6 and p-hat500-01. The figure groups an average number of visits by the degree of the vertices. In the sparse p-hat500-01 graph, MWC search has, on average, visited a vertex no more than twice by the time that the first high-accuracy clique has been found. When we consider the execution until the first 10 or the first 100 high-accuracy cliques have been identified, then the average number of visits per vertex increases, and predominantly so for the high-degree vertices. This aspect is logical: high-degree vertices are better connected, hence have better chance of appearing in the candidate list. They are also more likely to appear in multiple maximal cliques.

Similar trends occur for the dense keller6 graph (Figure 2). However, the number of visits per vertex is an order of magnitude higher. The first high-accuracy clique is found for keller6 after visiting a vertex, on average, 30 times. The number of visits, however, balloons to 200-350 visits per vertex when searching for the first 100 high-accuracy cliques. In other words, the dense connectivity implies that many combinations or subsets over vertices are generated during the search, many of which do not materialize in a maximal clique or in a high-accuracy WC.

In conclusion, while it is necessary to consider the whole search space to identify the MWC of the graph, **the first few times that a vertex is included in a clique are sufficient for it to demonstrate its potential impact for being a member of a high-accuracy WC.**

## 4 QCliques

### 4.1 High-Level Description

We observed in Section 3.1 that the high-accuracy WC are found quickly and in Section 3.2 that the first times vertices are added to a clique are enough to discover those high-accuracy WC. In other words, the connection between vertices of the graph facilitates a fast way to reach a close neighborhood of the MWC that contains high-accuracy WC. However, it requires three conditions:

- Performing the search in the neighborhood of all vertices to identify their potential fast routes to high-accuracy WC,
- Systematically searching in contrast to the prevalent methods [7, 9, 20] that randomly search and may miss paths to the MWC neighborhood,
- Steering the search towards favorable WC by preventing it from considering vertices multiple times as they present their impacts in the first attempts.

We design the QClique algorithm as an approximate algorithm that avoids the combinatorial growth by limiting the number of times a vertex is visited. Moreover, adding a vertex to a clique (which is called **vertex visit**) is the major computational component in processing the search space and incurs time and memory costs. To have a high-performance algorithm, we need to restrict the vertex visits to occasions with a good possibility for finding high-accuracy WC.

We use (i) an array called *NumVisits* to track the number of times each vertex is visited, (ii) the tunable parameter  $k$ , and (iii) the size of the current clique ( $WC$ ) to limit the vertex visits. If  $NumVisits_v \leq k \times |WC|$ , we allow adding vertex  $v$  to  $WC$  as  $v$  has not been processed in the current search space enough and high-accuracy WC may be discovered by adding  $v$ . By increasing  $k$ , QClique explores a greater portion of the search space and finds more accurate results while requires more execution time. In this way, QClique makes a trade-off between accuracy and performance.

### 4.2 QClique Pseudo-Code

**Preprocessing.** We use the Compressed Sparse Row (CSR) format [21] with sorted vertices based on the degrees and in descending order.

**QClique**<sup>3</sup>. Algorithm 3 shows QClique that receives  $k$  and returns  $WC_{max}$  (initialized in Line 1). For each vertex  $v$ , WC stores the current clique `CanList` is

<sup>3</sup> QClique is available on <https://github.com/QasimAbbas28/QClique-Euro-PAR>

---

**Algorithm 3** QClique

---

**Input:** Graph  $G(V, E, W)$ , int  $k$   
**Output:**  $WC_{max}$

```

1:  $WC_{max} = \{\}$ 
2: // Parallel loop
3: for  $v = 1; v \leq |V|; v++$  do
4:    $WC \leftarrow \{\}$ 
5:    $NumVisits[1..|V|] \leftarrow 0$ 
6:    $CanList \leftarrow N(v)$ 
7:    $SearchWC(v, WC, NumVisits,$ 
      $CanList)$ 
8: end for
9: return  $WC_{max}$ 

```

---



---

**Algorithm 4** SearchWC()

---

**Input:** int  $v$ , Array  $WC$ , NumVisits, CanList

```

1:  $WC.insert(v)$ 
2:  $NumVisits[v]++$ 
3: if  $weight(WC) > weight(WC_{max})$ 
   then
4:   // Atomic compare & swap
5:    $WC_{max} \leftarrow WC$ 
6: end if
7:  $NewCanList \leftarrow CanList \cap N(v)$ 
8: for  $u \in NewCanList$  do
9:   if  $NumVisits_u \leq k \times |WC|$  then
10:     $SearchWC(u, WC,$ 
       $NumVisits, NewCanList)$ 
11:   end if
12: end for
13:  $WC.remove(v)$ 
14: return

```

---

the set of potential vertices to extend the clique (equivalent to  $P$  in Alg. 2) and  $SearchWC()$  is called to explore the neighborhood of  $v$ .

$SearchWC()$ . Algorithm 4 shows the  $SearchWC()$  that performs recursive tree search to consider different combinations of vertices to be added to the current clique ( $WC$ ). After adding the vertex to  $WC$  (Line 1), the weight of the new clique is compared against  $WC_{max}$  (Lines 3–5). Then, to explore if  $WC$  can be extended, in Lines 7–12 the members of the intersection of  $N_v$  and  $CanList$  are examined based on the number of times they have been added to a  $WC$  using  $NumVisits$  and  $SearchWC()$  is called. After considering all members of  $NewCanList$ ,  $v$  is removed from  $WC$  (Line 13) as extension of the clique in the neighborhood of  $v$  is not further examined.

### 4.3 Parallelism

QClique parallelizes the outer loop over vertices (the **for** loop in Line 3, Algorithm 3). By limiting the search using  $k$  and  $NumVisits$ , QClique limits the total work performed per vertex, which inherently reduces load imbalance between vertices.

We have used a separate  $NumVisits$  for each vertex which is accessed by one thread. The other solution was to consider a global  $NumVisits$  which is accessed by concurrent threads. However, this solution suffers from three major problems: (i) it does not guarantee that the search space around each top-level vertex is explored to the same extent, i.e., one vertex may consume most allowed visits and leave little for other vertices, (ii) concurrent accesses to indices of  $NumVisits$  requires mutual exclusion or atomics that degrade performance, and (iii) the visits would be accounted for “out-of-order” in comparison to a

sequential execution, which would lead to each parallel execution potentially exploring a different part of the search space, depending on interleaving of counter updates, which may prohibit the algorithm of finding high-quality WC and make the quality potentially vary between executions.

To resolve these issues, we create a separate set of visit counters for each top-level vertex and only track visits within the part of the search space that corresponds to one top-level vertex. This defines the parallelism in the algorithm whilst maintaining good accuracy.

#### 4.4 Optimizing Performance in Exact MWC Algorithms

Using larger values for the  $k$  parameter covers a larger portion of the search space and a larger WC is discovered. Ultimately, by setting  $k$  to infinity, QClique is converted to an exact MWC algorithm, but with a great overhead on the execution time. Using WLMC algorithm [4] which uses the graph’s maximum clique as its initial WC seed and starts pruning the search space using its weight as the lower bound. We modify WLMC to use the output of QClique as its initial WC. This modification (i) reduces the time passes in producing the maximum clique of the graph and (ii) allows WLMC to prune the search space more effectively. As such, WLMC produces the exact solution in a shorter time. Section 5.5 shows that this replacement of initial seed facilitates  $3.3\times$  speedup.

## 5 Evaluation

### 5.1 Experimental Setup

**Machine.** We used a machine with  $2\times$  AMD EPYC 7702 CPUs, in 128 cores and threads (i.e., without hyper-threading) and 768 GB memory, CentOS 7.9.

**Code.** We implemented QClique in C++ with OpenMP [22]. We compare QClique to WLMC [4]<sup>4</sup>, FastWClq [7]<sup>5</sup>, PTC [9], and OTClique [10]<sup>6</sup>. The first three ones have single-threaded implementations and we compare them with single-threaded executions of QClique. OTClique has a parallel implementation that is compared with the parallel execution of the QClique. WLMC and OTClique are exact algorithms and PTC and FastWClq are approximate algorithms.

**Datasets.** We use graphs from DIMACS [23], NR [24], SNAP [25], and other resources [26–28] with random numbers between 0–50 as vertex weights.

**Presentation.** To prevent confusion with numbers, in this section, we present the relative results, e.g., speedup and relative accuracy. Numeric values are shown on Appendix A, Table 1. Average speedup has been calculated using arithmetic mean. In Figures 3, 4, 5, and 8 each dot presents a dataset.

### 5.2 Evaluation of Performance and Accuracy

Figure 3 compares performance of single-threaded execution of QClique with  $k = 2$  to WLMC as an exact MWC algorithm. In Figure 3 the Y-axis shows the percentage of accuracy provided by QClique in comparison to exact result in WLMC. The figure shows that QClique provides 4.7 times speedup and its

<sup>4</sup> <http://home.mis.u-picardie.fr/~cli/EnglishPage.html>

<sup>5</sup> <https://lcs.ios.ac.cn/~caisw/CLQ.html>

<sup>6</sup> <http://www2.kobe-u.ac.jp/~ky/otclique/otclique.html>



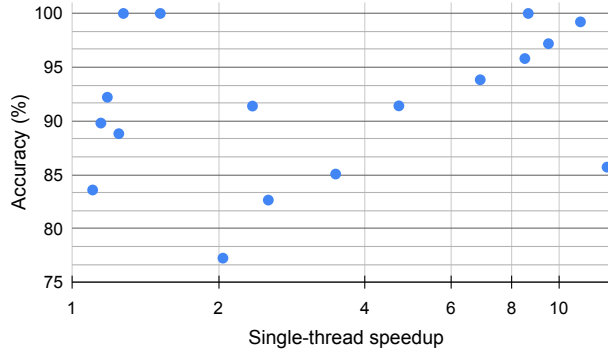


Fig. 3: Speedup and accuracy of single-threaded execution of QClique ( $k = 2$ ) in comparison to WLMC (exact single-threaded MWC)

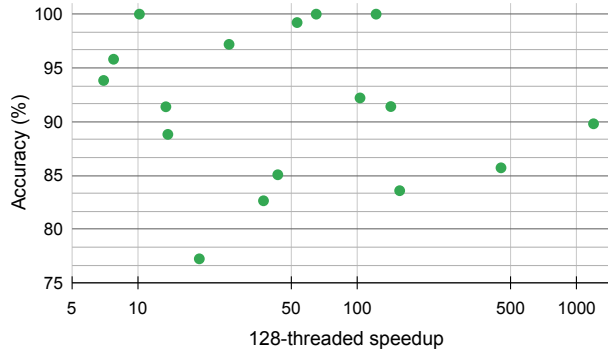


Fig. 4: Speedup and accuracy of 128-threaded execution of QClique ( $k = 2$ ) in comparison to OTClique (exact parallel MWC)

accuracy is always greater than 75% which on average is 91.4% . Table 1 shows that single-threaded QClique (Column 8) processes larger graphs in less than 270 seconds, while WLMC (Column 11) cannot process them in 9600 seconds.

Figure 4 shows speedup and accuracy of QClique with  $k = 2$  in comparison to OTClique as a parallel MWC. QClique is 45.9 times faster while the average accuracy of QClique is 91.4%.

Figure 5 compares QClique ( $k = 2$ ) vs PTC as a single-threaded approximate algorithm. In this figure, X-axis shows speedup of QClique compared to PTC and the Y-axis shows the accuracy of the algorithms. The figure shows that QClique usually provides accuracy close to 100% while it is 5.3 times faster than PTC. Table 1 shows that for the two largest graphs, PTC cannot complete processing in 1800 seconds while QClique execution is finished in less than 270 seconds.

### 5.3 Accuracy vs. Performance

Figure 6 shows the trade-off between the accuracy and execution time in two graphs: “p-hat700-1” as the sparser graph and “p-hat-300-3” as the denser one. For the sparser graph (Figure 6a), when the value of  $k$  is 1, the accuracy is 86.8%. When  $k$  is increased, the accuracy on the sparse graph is improved to 100% but this increases the execution time.

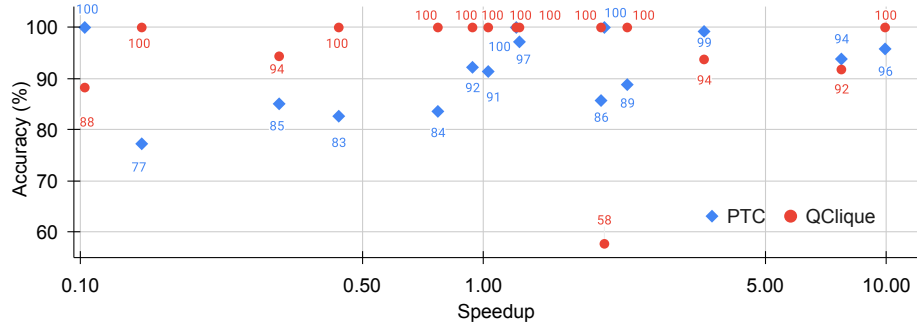
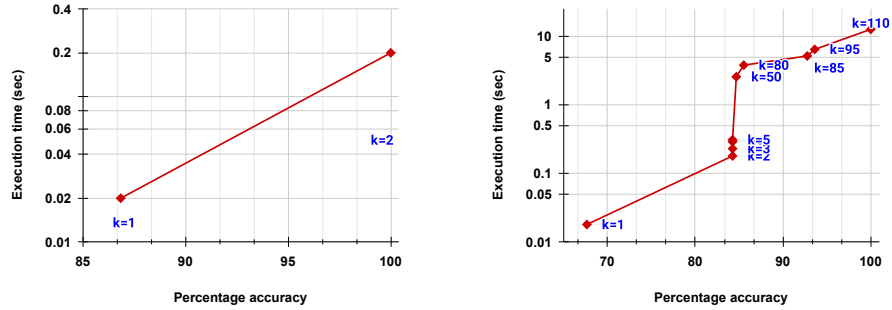


Fig. 5: Speedup and accuracy of single-threaded execution of QClique ( $k = 2$ ) in comparison to PTC (approximate single-threaded MWC) - Each graph has a speedup (the execution time of the PTC divided by QClique) on the X-axis and its accuracy for these algorithms is shown on the Y-axis.



(a) Graph “p-hat700-1” (density: 0.2)      (b) Graph “p-hat-300-3” (density: 0.7)  
 Fig. 6: The effects of increasing parameter  $k$  in QClique.

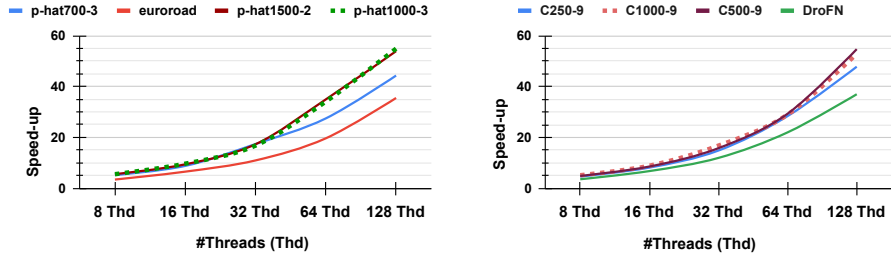
For the denser graphs, QClique requires a greater value of  $k$  to reach 100% accuracy and this increases the execution time. The number of neighbors per vertex is often greater in the dense graph which results in a larger search space and QClique needs more attempts per vertex to thoroughly explore the search space and deliver 100% accuracy.

**5.4 Scalability**

Figure 7 compares the speedup of multi-threaded execution of QClique in comparison to single-threaded execution. for graphs with lower density values in Figure 7a and higher ones in Figure 7b. It shows that QClique is scalable on graphs with higher density.

**5.5 Optimizing Exact Algorithms**

In Section 4.4, we explained improving the performance of exact algorithms by deploying QClique to create a high-accuracy WC and to use it as the initial seed. The Figure 8 shows the speedup provided by this seeding technique. The last column of Table 1 (named “Q-W”) also includes its performance. As WLMC is a sequential algorithm, we have used the single-threaded execution of QClique. Each cell in Column “Q-W” shows the sum of the single-threaded QClique exe-



(a) Low-density graphs (b) High-density graphs  
 Fig. 7: Self-relative parallel speedup of QClique where Thd is the threads

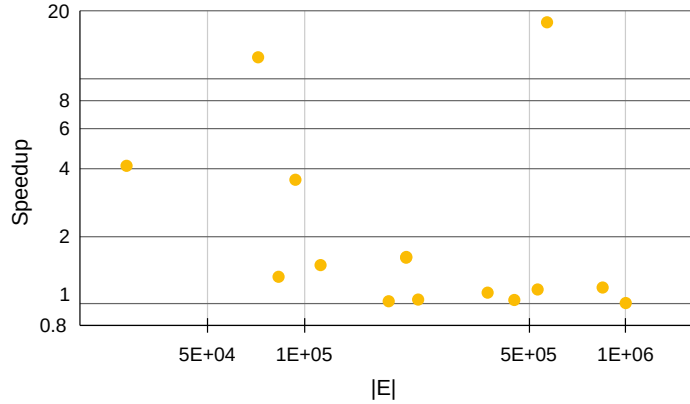


Fig. 8: Speedup of single-threaded execution of QClique and WLMC (after receiving the seed from QClique) as a function of the number of edges in the graph

cution time and the execution time of modified WLMC for the related dataset. The results show that single-threaded QClique provides 3.3 times speedup, on average, to WLMC.

### 6 Further Related Work

Markov Chain Monte Carlo approach [29] and local search approach [30] are approximate MWC algorithms but they cannot improve the best-found WC if getting stuck in local optima [31]. Other approximate algorithms use heuristics such as tabu search [8, 32], ant colony [33], and genetic algorithm [34]. Kiziloz and Dokeroglu [9] proposed a heuristic-based technique called the Parallel Tabu Clique (PTC) algorithm which uses vertex operators to search for possible neighborhood extensions. FastWClq [7, 35] interleaves graph reduction and clique finding. The OTClique [10] algorithm is an exact algorithm that constructs an optimal table by dividing the vertices into subsets and calculating the upper bound using the sum of the optimal weights of the subgraphs. The Weighted Large Maximum Clique (WLMC) [4] is a branch-and-bound exact algorithm that examines the search space of all possible cliques in the graph and based on the best-founded WC, prunes sub-optimal branches in a recursive manner. Wang

et al [3] introduced SCCWalk that eliminates the Hamiltonian cycle problem during the local search.

## 7 Conclusion

We analyzed the discovery of high-accuracy weighted cliques (WC) and introduced QClique, a novel approximate MWC algorithm that truncates the search space by dynamically limiting the number of times a vertex is processed as a member of high-accuracy cliques.

QClique is designed to effectively steer the search to the neighborhood of high-accuracy WC. In this way, QClique limits the use of set intersections to a minimum and does not utilise weight-based heuristics and memory-and compute-intensive operations such as graph reduction. Our evaluation on a wide range of graphs shows that QClique delivers a high-level scalability and  $4.7\text{--}82.3\times$  speedup in comparison to state-of-the-art MWC algorithms while providing 91.4% accuracy, on average. The evaluation also shows that QClique accelerates WLMC, an exact MWC algorithm, by 3.3 times, on average.

## A Numeric Results

Table 1 shows the numeric values of evaluation.

## Acknowledgments

We are grateful to the Euro-Par anonymous reviewers for the detailed and constructive comments and to Northern Ireland High-Performance Computing (NI-HPC), funded by EPSRC (EP/T022175), for computing resources.

This project has received funding from the European Union’s Horizon 2020 Marie Skłodowska-Curie grant agreement No 945231 and the Engineering and Physical Sciences Research Council under grant agreements EP/X01794X/1, EP/Z531054/1 and EP/T022175/1.

## References

1. Y. Wang et al. Two efficient local search algorithms for maximum weight clique problem. In *Proc. AAAI Conf. Artif. Intell.*, volume 30, 2016.
2. SP. Fekete and H. Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38:501–511, 2004.
3. Y. Wang et al. Scwalk: An efficient local search algorithm and its improvements for maximum weight clique problem. *Artifi. Intelli.*, 280:103230, 2020.
4. H. Jiang et al. An exact algorithm for the maximum weight clique problem. In *Proc. AAAI Conf. Artif. Intell.*, volume 31, 2017.
5. S. Shimizu et al. Fast maximum weight clique extraction algorithm: Optimal tables for branch-and-bound. *Discrete Applied Mathematics*, 223:120–134, 2017.
6. Z. Fang et al. An exact algorithm based on maxsat reasoning for the maximum weight clique problem. *J. Artif. Intell. Res.*, 55:799–833, 2016.
7. S. Cai and J. Lin. Fast solving maximum weight clique problem in massive graphs. In *IJCAI*, pages 568–574, 2016.
8. Michel Gendreau. *An introduction to tabu search*. Springer, 2003.
9. HE. Kiziloz and T. Dokeroglu. A robust and cooperative parallel tabu search. *Computers & Industrial Engineering*, 118:54–66, 2018.
10. S. Shimizu et al. Parallelization of a branch-and-bound algorithm for the maximum weight clique problem. *Discrete Optimization*, 41:100646, 2021.

Graphs	V  (K)	E  (K)	Den	QClique (k=1)		QClique (k=2)		WLMC		FastWClique		PTC		Q-W				
				MW	TS	TP	MW	TS	TP	MW	TS	MW	TS	MW	TS			
san400-0-7-2	0.4	56	0.7	125	1.9	0.03	141	7.9	1.5	157	9.1	>1800	117	38.5	154	437.4	157	6.7
p-hat1000-3	1.0	372	0.7	261	17.5	0.4	343	346.5	48.2	415	877.2	>1800	49	49.6	415	151.6	415	781.7
p-hat500-03	0.5	94	0.8	232	2.4	0.1	302	52.4	5.8	340	65.4	78.7	239	0.1	340	119.1	340	18.4
p-hat700-3	0.7	183	0.7	281	6.4	0.1	367	129.1	17.5	398	152.7	>1800	25	0.0	398	121.2	398	148.5
p-hat1500-2	1.5	569	0.5	270	29.0	0.8	415	259.6	69.2	427	2,471.0	>1800	190	273.0	427	318.8	427	138.6
p-hat1500-3	1.5	847	0.8	390	58.9	1.4	468	1,981.9	177.2	468	2,528.8	>1800	116	562.1	413	203.1	468	2,137.6
p-hat300-3	0.3	33	0.7	159	2.4	0.01	199	3.8	10.4	235	1.14	0.7	161	0.2	235	85.7	235	0.2
p-hat700-1	0.7	60	0.2	66	0.7	0.02	76	0.79	0.2	76	0.18	0.2	72	29.4	76	469.4	76	0.5
brock800-3	0.8	207	0.6	108	5.4	0.1	160	29.6	5.9	167	251.6	45.9	85	26.1	167	293.2	167	156.8
brock800-4	0.8	208	0.6	112	5.4	0.1	137	36.7	6.5	146	253.1	45.5	146	40.7	134	283.6	146	156.5
c250-9	0.3	28	0.9	200	0.3	0.01	276	66.5	1.4	302	156.2	18.5	98	29.3	302	68.3	302	38.0
c500-9	0.5	112	0.9	275	3.4	0.1	326	127.4	11.6	390	140.5	>1800	159	3.3	390	98.1	390	94.5
ci1000-9	1.0	450	0.9	332	25.2	0.5	377	1,025.5	94.5	488	2,094.7	>1800	275	175.9	488	145.5	488	2,011.9
c2000-05	2.0	1,000	0.5	94	39.8	0.9	127	276.1	33.9	128	3,056.9	>1800	79	757.2	120	973.6	128	3,029.2
keller5	0.8	226	0.8	152	9.7	0.2	182	117.9	14.8	182	1,019.9	>1800	139	3.3	105	235.3	182	975.5
gen400	0.4	72	0.9	233	2.4	0.05	302	303.1	7.0	355	1,055.0	302.0	45	10.3	335	94.3	355	84.6
frb30-15-2	0.5	83	0.8	161	4.3	0.1	192	68.8	4.0	224	864.0	>1800	94	31.1	224	134.7	224	654.5
pwr-bcs	1.7	4.1	0.003	81	88.6	0.03	92	91.7	3.7	>9600	>1800	>1800	71	148.2	81	96.3	112	186.6
scc fb-msgs	1.9	532	0.3	3,911	41.7	24.8	3,911	38.1	27.7	3,911	57.8	>1800	3,911	57.8	3,911	45.9	3,911	49.9
DroFn	11.4	788	0.012	10	71.6	0.8	13	266.9	18.8	>9600	>1800	>1800	>1800	>1800	>1800	>1800	13	162.2
cit-DBLP	12.6	50	0.001	60	68.9	1.4	78	179.4	6.2	>9600	>1800	>1800	52	183.8	>1800	>1800	88	76.6

Table 1: Comparison of maximum weight (MW), serial execution time (TS), and parallel 128-threaded execution time (TP) in seconds. WLMC and QClique (OTC) are exact algorithms having the same maximum weighted clique values. QClique, FastWClique, and PTC are approximate algorithms. |V| and |E| are the number of vertices and edges in kilo. Density is shown as “Den”, and “Q-W” is QClique (with  $k = 1$ ) accelerating WLMC which produces exact results.

11. Panos M Pardalos and Jue Xue. The maximum clique problem. *Journal of global Optimization*, 4:301–328, 1994.
12. B. Pattabiraman et al. Fast algorithms for the maximum clique problem on massive sparse graphs. In *Algorithms and Models for the Web Graph*, Cham, 2013.
13. Patric R.J. Östergård. A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, 120(1):197–207, 2002.
14. Buchanan A. et al. Solving maximum clique in sparse graphs: an  $\mathcal{O}(nm + n^2 d/4)$  algorithm for  $d$ -degenerate graphs. *Optimization Letters*, 8:1611–1617, 2014.
15. E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2(1):1–6, 1973.
16. C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.
17. H. Vandierendonck. Differentiating set intersections in maximal clique enumeration by function and subproblem size. In *Proceedings of the 38th ACM International Conference on Supercomputing: ICS 2024*, 2024.
18. J. Blanuša et al. Manycore clique enumeration with fast set intersections. *Proc. VLDB Endow.*, 13(12):2676–2690, 2020.
19. Patrick Prosser. Exact algorithms for maximum clique: A computational study. *Algorithms*, 5(4):545–587, 2012.
20. E. Sevinc and T. Dokeroglu. A novel parallel local search algorithm for the maximum weight clique. *Soft Computing*, 24(5):3551–3567, 2020.
21. Y. Saad. Sparskit: a basic tool kit for sparse matrix computations, 1994.
22. L. Dagum and R. Menon. OpenMP: An industry-standard api for shared-memory programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, jan 1998.
23. L. Sanchis. Test case construction for the vertex cover problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 15:315–326, 1994.
24. R. Rossi and N. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.
25. L. Jure and K. Andrej. SNAP Datasets: Stanford large network dataset collection. <http://snap.stanford.edu/data>, June 2014.
26. M. Brockington and JC. Culberson. Camouflaging independent sets in quasi-random graphs. *Cliques, coloring, and satisfiability*, 26:75–88, 1996.
27. L. Sanchis. Generating hard and diverse test sets for np-hard graph problems. *Discrete Applied Mathematics*, 58(1):35–66, 1995.
28. I. Overton et al. Functional transcription factor target networks illuminate control of epithelial remodelling. *Cancers*, 12(10):2823, 2020.
29. D. Achlioptas and F. McSherry. Fast computation of low-rank matrix approximations. *JACM*, 54(2):9–es, 2007.
30. P. Galinier and A. Hertz. A survey of local search methods for graph coloring. *Computers & Operations Research*, 33(9):2547–2562, 2006.
31. D. Achlioptas et al. On the bias of traceroute sampling: or, power-law degree distributions in regular graphs. *JACM*, 56(4):1–28, 2009.
32. A. Hertz et al. A tutorial on tabu search. In *Proc. of Giornate di Lavoro AIRO*, volume 95, pages 13–24, 1995.
33. D. El-Baz et al. A parallel ant colony optimization for the maximum-weight clique problem. In *2016 (IPDPSW)*, pages 796–800, 2016.
34. A. Lambora et al. Genetic algorithm-a literature review. In *2019 (COMITCon)*, pages 380–384. IEEE, 2019.
35. S. Cai et al. A semi-exact algorithm for quickly computing a maximum weight clique in large sparse graphs. *JAIR*, 72:39–67, 2021.