Journal of Molecular Structure 1304 (2024) 137639

Contents lists available at ScienceDirect

Journal of Molecular Structure

journal homepage: www.elsevier.com/locate/molstr

# Enhanced molecular docking: Novel algorithm for identifying highest weight k-cliques in weighted general and protein-ligand graphs

Kati Rozman [a], An Ghysels [b], Bogdan Zavalnij [c], Tanja Kunej [d], Urban Bren [e], Dušanka Janežič [a,**], Janez Konc [a,e,f,g,*]

[a] University of Primorska, Faculty of Mathematics, Natural Sciences and Information Technologies, Glagoljaška ulica 8, Koper SI-6000, Slovenia
[b] Ghent University, IBiTech – BioMMedA group, Corneel Heymanslaan 10, entrance 36, 9000 Gent, Belgium
[c] Rényi Institute of Mathematics, 1053 Budapest, Hungary
[d] University of Ljubljana, Department of Animal Science, Biotechnical Faculty, SI-1230 Domžale, Slovenia
[e] University of Maribor, Faculty of Chemistry and Chemical Technology, Smetanova 17, SI-2000 Maribor, Slovenia
[f] National Institute of Chemistry, Theory Department, Hajdrihova 19, SI-1000 Ljubljana, Slovenia
[g] University of Ljubljana, Faculty of Pharmacy, Aškerčeva 7, SI-1000 Ljubljana, Slovenia

## ARTICLE INFO

*Keywords:*
Highest weight k-cliques algorithm
Weighted graphs
Graph coloring
Graph theory
Molecular docking
ProBiS-Dock algorithm

## ABSTRACT

Molecular docking, a key process in drug discovery, is often used in the discovery of new bioactive compounds. In this technique, small molecules are systematically placed at a protein binding site to identify the ligands with the highest binding affinity. Here we have developed a new graph-theoretical algorithm called K-CliqueWeight. This algorithm efficiently identifies the top N highest weight k-cliques in different types of vertex-weighted graphs and can serve as a building block for various algorithms addressing different problems, including molecular docking. K-CliqueWeight and its variant K-CliqueDynWeight are extensions of our established and widely used maximum clique algorithm. Our new algorithm uses a novel approach to approximate graph coloring and provides efficient upper bounds on the size and weight of a k-clique within the branch-and-bound algorithm. It outperforms alternative methods and often shows a speedup of several orders of magnitude. Rigorous tests with general random graphs and those specifically designed for docking confirm its exceptional performance. K-CliqueWeight has been integrated into the existing ProBiS-Dock algorithm for molecular docking. The algorithm is freely available to the academic community at http://insilab.org/kcliqueweight.

## 1. Introduction

The k-clique problem is a problem of identifying completely connected subgraphs or cliques within a graph, each consisting of exactly k vertices. The highest weight k-clique problem extends the concept to vertex-weighted graphs and its goal is to find the k-clique with the highest sum of vertex weights within the graph.

Another challenging problem is the N highest weight k-cliques problem, where the goal is to find a set of up to N weighted cliques whose weights are among the highest of all k-cliques in a given graph. In cases where the graph contains less than N k-cliques, the algorithm returns all k-cliques within a graph.

Several algorithms have been developed to find k-cliques in unweighted graphs and also to detect highest weight k-cliques in vertex-weighted graphs [1,2]. These algorithms have been used in a variety of important contexts, in both research and industry [3–6].

In this work, we introduce a new algorithm for identifying up to N highest-weight k-cliques in a vertex-weighted graph, where N is given as a parameter. To the best of our knowledge, this is the first algorithm of its kind. Our newly developed algorithm is a versatile graph-theoretical algorithm suitable for various types of vertex-weighted graphs and universal problem solving.

We show its application in molecular docking [7–11], where our new algorithm, integrated with the ProBiS-Dock algorithm, is used to determine the conformation of the small molecule in the protein with the lowest binding energy (see Fig. 1). Interestingly, a similar strategy

* Corresponding author at: National Institute of Chemistry, Theory Department, Hajdrihova 19, SI-1000 Ljubljana, Slovenia.
** Corresponding author at: University of Primorska, Faculty of Mathematics, Natural Sciences and Information Technologies, Glagoljaška ulica 8, Koper SI-6000, Slovenia.
*E-mail addresses:* dusanka.janezic@upr.si (D. Janežič), konc@cmm.ki.si (J. Konc).

https://doi.org/10.1016/j.molstruc.2024.137639
Received 7 December 2023; Received in revised form 22 January 2024; Accepted 24 January 2024
Available online 28 January 2024

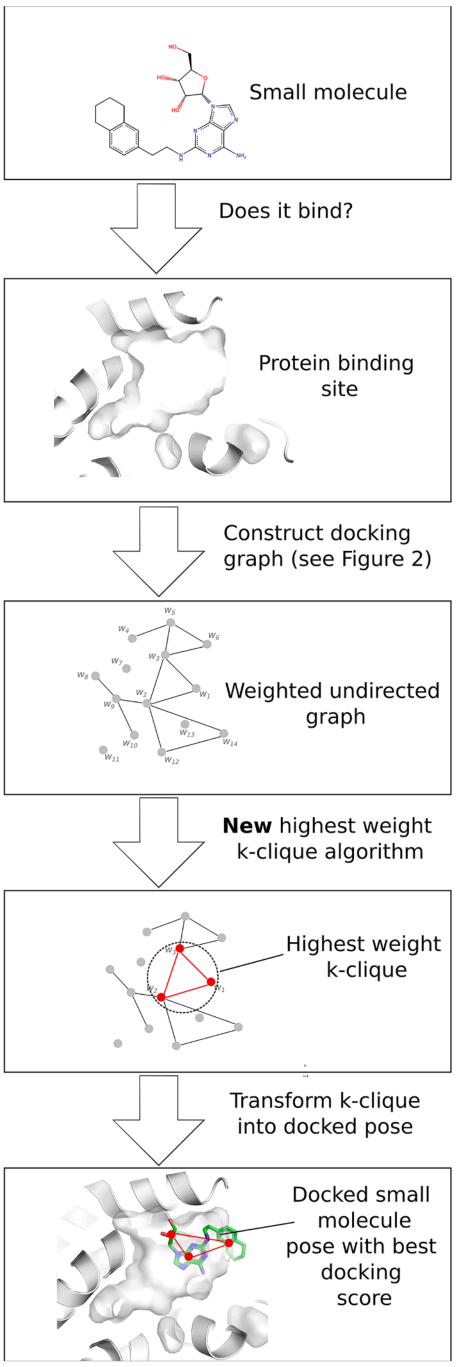
Small molecule
Does it bind?
Protein binding site
Construct docking graph (see Figure 2)
Weighted undirected graph
New highest weight k-clique algorithm
Highest weight k-clique
Transform k-clique into docked pose
Docked small molecule pose with best docking score

**Fig. 1.** Overview of molecular docking with the new N highest weight k-cliques algorithm.

using a maximum weight clique search on a quantum device has been explored by others [12].

ProBiS-Dock enables rapid docking of small molecules to proteins and has been successfully validated *in silico* using standard benchmarks as well as *in vitro* by the discovery of new inhibitors in cancer therapy [7]. Internally, it uses specially constructed vertex-weighted graphs, so-called docking graphs, in which a k-clique with the highest weight corresponds to the conformation with the lowest energy of a small molecule bound to the protein.

We construct a comprehensive new test set of docking graphs. We then evaluate the performance of our new graph-theoretic algorithm on these graphs as well as on random weighted graphs. Our algorithm shows superior performance on both types of graphs compared to another competing algorithm in the field. The docking graphs constructed in this work can also serve as a valuable, real world test set for other researchers to evaluate the effectiveness of their graph-theoretic algorithms operating on vertex-weighted graphs.

## 2. Methods

### 2.1. Generation of weighted graphs for testing

#### 2.1.1. Random weighted graphs

The first test set consists of 88 random weighted graphs. We generated the random weighted graphs with 100, 200, 300, 500, 700, 1000, 5000 and 10,000 vertices. We also varied the edge density within each graph size category, where the edge density of a graph is the probability p of an edge between two graph vertices. These probabilities took discrete values of 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, and 0.99. For each graph of a given size, for each pair of vertices vi and vj, we have connected the vertices vi and vj by an edge if a uniform continuous distribution u(vi, vj) for random numbers in the range [0, 1) is less than the probability p with equal probability in the entire range.

We have assigned a weight to each vertex v, which is a positive integer, by proceeding as follows For each vertex v, we have generated a real random number according to the normal random number distribution, which is defined as follows: f(x;μ,σ)=1/(σ·sqrt(2π))eˆ(−1/2-((*x*−μ)/σ)ˆ2), where μ is the mean and σ is the standard deviation; we set μ to 100.0 and σ to 20.0. We checked that the resulting random values were positive numbers. We then multiplied the real number by 10,000 and rounded the resulting value to the nearest integer so that we obtained a positive integer random number that was used as the weight of vertex v.

#### 2.1.2. Docking protein-ligand weighted graphs

The second test set consists of 447 weighted docking graph, which are based on actual protein-ligand structures. Docking protein-ligand graphs were generated using the procedure introduced and described in detail in Ref. [7] and are based on the Directory of Useful Decoys (DUD-E) benchmark set [13]. The DUD-E set contains 22,886 active compounds and their affinities against 102 protein targets, an average of 224 ligands per protein target and was developed for benchmarking molecular docking programs. We used up to nine active compounds for each of the 93 protein targets we were able to prepare. For each target, only active compounds with different CHEMBL identifiers [14] and different fragment numbers were considered. This resulted in 447 docking graphs. The following describes how a docking graph representing a combination of an active compound and a protein target was constructed (see Fig. 2).

First, each active compound to be docked was broken down into rigid fragments and rotatable bonds (Fig. 2a). The fragments are rigid substructures of the active compound, while the rotatable bonds correspond to the atoms between the rigid fragments, where the relative orientation of the fragments to each other can change, i.e. rotatable bonds allow a molecule to adopt different conformations. The number of possible rotations and conformations therefore increases exponentially with the number of rotatable bonds. The active compounds in our set consisted of one to thirteen fragments.

Second, the resulting rigid fragments were each inserted individually into the protein binding site in different positions and orientations (Fig. 2b). Each fragment was moved across a grid within the binding site in steps of approximately 0.75 Å. At each position on the grid, a fragment was systematically rotated around its geometric center in all three dimensions. For each position and rotation of a fragment obtained, a docking score, the so-called ProBiS-score, was calculated [7]. This score corresponds approximately to the strength of the binding of the fragment to the protein. Several positions were determined for each fragment and the most favorable ones, i.e. those with a ProBiS score of less than zero, were selected for further analysis. All fragments of an active

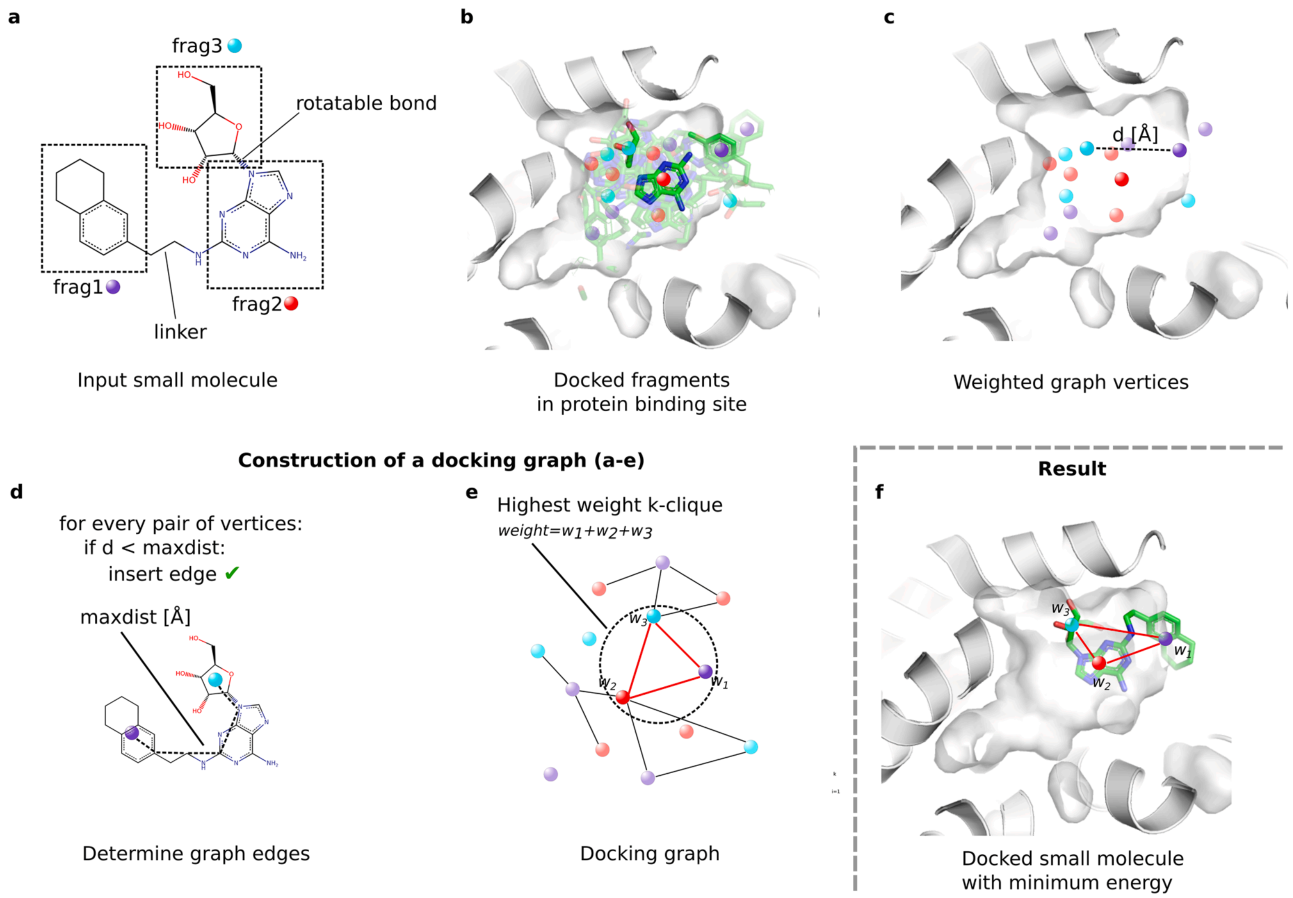

**Fig. 2.** Construction of docking graphs used in the ProBiS-Dock molecular docking algorithm [7]. (a) The input small molecule is divided into rigid fragments separated by rotatable bonds and linkers (sets of atoms connected by rotatable bonds); (b) each fragment is docked separately to the protein binding site, resulting in multiple possible docking positions for each fragment; (c) graph vertices are determined as geometric centers of the fragments, each vertex is assigned a weight representing its docked score, and a distance d between each pair of vertices is calculated; (d) graph edges are determined so that if the distance between two vertices (each representing a different fragment) is less than the maximum possible distance, an edge is inserted between these two vertices; (e) a highest weight k-clique ($k =$ 3 in this case, since there are three different fragments in the small molecule) in the constructed docking graph represents a possible energetically favorable configuration with the highest weight (minimum docking score) of the three fragments in the protein binding site; (f) for each highest weight k-clique, a docked small molecule is reconstructed from the three fragments forming the clique by inserting missing rotatable bonds and linkers between them.

compound were treated equally using the procedure described above, so that we obtained several possible positions and orientations within the binding site for each fragment.

Third, the collection of fragment positions was considered as a set of vertices (Fig. 2c). The docking graph was constructed such that each of its vertices corresponds to a position of each rigidly docked fragment, i.e., its geometric center. The vertices thus represent different fragments and their different positions. The ProBiS-Score of the fragment is used as the weight of the vertex. Since docking scores are negative real numbers, we converted them to weights, which are positive integers, by multiplying each docking score by −10,000 and rounding the resulting value to the nearest integer.

Fourth, the edges between the vertices were constructed (Fig. 2d). Two vertices, each representing a different fragment of the docked ligand, were connected by an edge if the corresponding docked fragments could be reconnected to the original small molecule ligand, where in most cases the docked small molecule is in a different conformation than the conformation used as input. For this purpose, a linker was inserted between each fragment pair, where a linker can be a chain of rotatable bonds or a combination of rotatable bonds and other rigid fragments. Whether two fragments can be linked together is determined by a) checking that the fragments do not collide with each other, and b) comparing the actual distance between a pair of docked fragments in the binding site with the maximum possible distance between these fragments in the docked small molecule. The latter distance is calculated as the length of the shortest path, measured in atom-atom bonds, between the two fragments in a molecule. If the actual distance is less than the maximum possible distance, the two vertices have been connected by an edge.

This procedure leads to a docking graph (Fig. 2e), a vertex-weighted graph in which a highest weight k-clique, where k is the number of fragments of the small molecule, corresponds to the lowest energy configuration of the k fragments representing the complete small molecule. Completeness is meant here in the sense that the k-clique contains all the constituent fragments of a small molecule. Linkers, which are inserted between the docked fragments later in the process, are still missing at this stage. The fragments that form a highest weight k-clique are at a relative distance from each other, which makes it possible to reconnect them by inserting linkers to form a complete active compound. After insertion of the linkers, a docked conformation of the given small molecule bound to the target protein is obtained (Fig. 2f).

### 2.2. *Molecular docking with the ProBiS-Dock algorithm enabled by the N highest weight k-cliques algorithm*

It is important to distinguish the search for highest weight k-cliques from the search for maximum weight cliques in the context of molecular docking. In the latter type of algorithms, the goal is to find the maximum weight clique, a clique with the highest sum of vertex weights. The number of vertices in this clique is not limited to k as is in the search for k-cliques.

Assuming that a small molecule to be docked consists of k fragments, where k can be different for different small molecules (see Fig. 2, where $k = 3$), then a weighted clique represents a configuration of fragments in three-dimensional binding site space so they are at distances relative to each other which allow them to be bonded back into the original small molecule. The vertex weights here indicate the fragment docking scores. There can be multiple fragment configurations that fulfil these criteria. A maximum weight clique algorithm or a highest weight k-clique algorithm can be used to identify a configuration with the highest docking score. However, a maximum weight clique search could lead to a solution with less than k vertices, which as such does not represent the entire molecule. In constrast, the highest weight k-clique represents the configuration that includes all k fragments and thus represents the most favorable docked conformation of the entire small molecule within a protein binding site. This is the desired result in molecular docking. The search for the k-clique with the highest weight is thus an important and still open problem to be solved for these types of calculations.

### 2.3. Graph notations

This paper introduces a new N highest weight k-cliques algorithm and its variant. The algorithm builds on and extends the MaxClique maximum clique algorithm developed earlier [15]. The new algorithm, K-CliqueWeight (KCQW), and its variant K-CliqueDynWeight (KCQDW) both search for up to N highest weight k-cliques in a vertex-weighted graph. The KCQDW variant uses dynamically varying upper bounds as introduced in [15], which in general increase efficiency of the algorithm. Table 1 gives an overview of the developed algorithms, and Table 2 gives an overview of the variables used in the developed algorithms and their meaning.

### 2.4. A maximum clique algorithm for unweighted undirected graphs – the core algorithm

We outline the MaxClique (MCQ) algorithm for finding maximum cliques in an unweighted undirected graph, here referred to as the core algorithm, which was first described in [15]. In the present work, we have extended this core algorithm with the ability to find N highest weight k-cliques in a vertex-weighted graph, as described in Section 5.

The core algorithm consists of two procedures, the Expand procedure, a recursive depth-first search procedure that uses the branch-and-bound technique to explore possible maximum cliques in a graph, and the ColorSort procedure, an improved approximate graph coloring algorithm that provides upper bounds on the size of the maximum clique that can be found at each step of the Expand procedure. In graph coloring, each graph vertex is assigned a color using the smallest possible number of colors so that no two adjacent vertices have the same color. This is an NP-hard problem, and performing it exactly on a graph would take exponential time as the number of graph vertices increases. Therefore, we use an approximate coloring algorithm that determines a suboptimal number of colors. This approximate algorithm is fast and still provides reasonable upper bounds on the size of a clique in a graph.

**Table 1**
Overview of the developed maximum clique and highest weight k-cliques algorithms.

| Name* | Definition | Graph type | Searches for |
|---|---|---|---|
| MCQ [15] | MaxClique | unweighted graph | maximum clique |
| MCQD [15] | MaxCliqueDyn | unweighted graph | maximum clique |
| KCQW | K-CliqueWeight | weighted graph | at most N k-cliques |
| KCQDW | K-CliqueDynWeight | weighted graph | at most N k-cliques |

* New and old algorithms are separated by horizontal line.

**Table 2**
Graph variables and definitions used in the clique algorithms.

| Name | Definitions | Algorithm* |
|---|---|---|
| G | An input graph G represented by a set of vertices R and a set of adjacent vertices Γ(v) for each vertex v∈R. | All |
| R | A set of input graph vertices represented by numbers 0… n. | |
| Γ(v) | A set of vertices adjacent (connected by an edge) to each vertex v∈R in the input graph G. | |
| \|Γ(v)\| | Degree of vertex v, the number of adjacent vertices of vertex v∈R, i.e., connected by an edge to vertex v. | |
| Δ(R) | Maximum degree of any vertex in R. | |
| C | A set of color classes C[$k$], $k = 1…\|R\| + 1$, where each contains non-adjacent vertices only; k represents the color of all vertices in the k-th color class. | |
| Q | A global variable, a set of graph vertices that represent a (weighted) clique being constructed. | |
| W | A set of weights for graph vertices of the input graph (e.g., W[R[i]] denotes the weight of vertex with index i in R). | KCQ(D)W |
| k | Number of vertices k in a k-clique. | |
| N | The maximum number of highest weight k-cliques to be outputted by the algorithm. | |
| P | A set of k-cliques and their weights, containing N or less highest weight k-cliques found in the input graph G. | |

* »All« denotes that variables in this section (between horizontal lines) are used by all the developed algorithms (maximum clique and k-clique); KCQ(D)W: only by K-CliqueWeight and K-CliqueDynWeight.

**Input.** The algorithm takes as input an unweighted graph G represented by a set of vertices R and a set of adjacent vertices Γ(v) for each vertex v∈R.

**Output.** The maximum clique that was found in the input graph represented by the set Qmax consisting of its vertices. The algorithm outputs one of possibly several alternative maximum cliques that are in the input graph.

**Initialization.** The vertices in set R (see line 1 in Table 3) are initially sorted by their degrees in decreasing order, so that the first vertex has the highest degree of all vertices in R and the last one has the lowest degree. The initial upper bound (color) is then determined for each vertex v∈R, so that the first Δ(R) vertices in R have their colors set to 0, 1, 2, … Δ(R), respectively, and the remaining vertices have their colors set to Δ(R) (see line 2 in Table 3). This initialization of colors was found to be efficient in [15].

**Table 3**
Initialization of graph vertices and the first call of the maximum (weight) clique algorithms.

| Line | Pseudocode | Description | Algorithm* |
|---|---|---|---|
| 1. | SortDecDeg (R) | Sort vertices in R by decreasing degrees (from the highest to the lowest degree). | MCQ |
| 2. | InitializeColors (R) | Determine initial upper bounds (colors). | |
| 3. | SortDecDegIncWeigh (R, W) | Sort vertices in R by decreasing degrees, then sort vertices with equal degrees by increasing weights. | KCQ(D)W |
| 4. | ColorSortKWeight (R) | Set initial colors and wcolors of the input vertices in set R. These are the upper bounds to the size and the weight of a highest weight k-clique, respectively. | |
| 5. | Expand (R) | Start the recursive branch-and-bound tree search for (weight) cliques. | All |

* »All« denotes that pseudocode in this section (between horizontal lines) is used by all the developed algorithms (weighted and unweighted); MCQ: only by unweighted MaxClique and MaxCliqueDyn algorithms; KCQ(D)W: only by K-CliqueWeight and K-CliqueDynWeight.

**ColorSort procedure: determination of upper bounds to clique size.** ColorSort efficiently colors the input graph's vertices, where the number of colors represents the upper bound for the size of the maximum clique that can be found in the graph (Table 4). The procedure also sorts (reorders) the vertices. It takes the candidate set of graph vertices, R, as input, and partitions it into color classes represented by the set C, where each color class C[k] contains vertices with color k in set R. In a loop over vertices in R, each vertex $p \in R$ is assigned a color (a positive number k) based on its connectivity with other vertices, so that no two adjacent vertices have the same color. Therefore, each two adjacent vertices p, $r \in R$ are always assigned into two different color classes $C[k_1]$ and $C[k_2]$ where $k_1 \neq k_2$. The algorithm continues until all vertices in R are colored.

In comparison to the original maximum clique algorithm [16], on which we based our maximum clique algorithm [15], we introduced an improvement in the ColorSort algorithm. This improvement is based on the observation that a vertex v in the candidate set of graph vertices, R, with its color denoted as color(v) will never be added to the current growing clique Q if $\mathrm{color}(v) < |Qmax| - |Q| + 1$. Such vertices can be kept in their initial non-increasing degrees ordering, which is favorable as input to the coloring algorithm at the next level of the recursion of the Expand procedure, as it enables the graph to be colored with fewer colors and since the number of colors is the upper bound to the size of a maximum clique, this results in tighter upper bounds [17]. Thus, at the start of the algorithm (line 3, Table 4), we calculate a minimum color $\min\_k = |Qmax| - |Q| + 1$. Vertices with colors below min_k are kept in the initial order, whereas vertices with color of min_k and above are reordered based on their colors. Overall, our improved approximate coloring algorithm developed in [15] provides an efficient and effective way to color the vertices of an unweighted graph, providing efficient upper bounds to the size of a maximum clique, and enabling an efficient search for a maximum clique in the input graph.

**Expand procedure: finding a maximum clique.** After the upper bounds of vertices in R are set, the Expand procedure is called. This call is done once during the Initialization phase (see line 5 in Table 3) and at each step during the recursive branch-and bound tree search (line 15 in Table 5). The Expand procedure explores the search tree of possible cliques, starting with the initially empty set Q representing a set of vertices of the currently growing weighted clique. At each step, Expand selects the last vertex $p \in R$ with the highest color, which is the upper bound to the size of the maximum clique (line 3 in Table 5), and removes this vertex p from the set R. If the size of Q plus the color of vertex p is greater than the size of Qmax (line 4 in Table 5), then vertex p is added to Q (line 6 in Table 5).

The subset of vertices $Rp \subset R$, in which each vertex is adjacent to p, is determined (line 7 in Table 5), and if this set Rp is not empty, the ColorSort procedure is called with Rp as an argument. This sets the upper bounds (colors) for vertices in set Rp. The Expand procedure is then called recursively with Rp as argument. The recursive calls continue until Rp is empty. If Rp is empty, and the size of Q is greater than the size of Qmax (line 16 in Table 5), Qmax is updated to be Q. In any case, if the size of the candidate clique Q is greater than the size of the Qmax or not, Expand backtracks removing the added vertex from Q and allowing the search along a different branch of the search tree. The result of the Expand procedure is a set Qmax containing vertices of a maximum clique that was found in the input graph.

**Dynamically varying bounds for greater efficiency.** In [15], we introduce the concept of varying the tightness of upper bounds dynamically during the maximum clique search resulting in the MaxCliqueDyn algorithm. Varying bounds enables to reduce the number of

**Table 4**
Improved approximate coloring algorithm that colors an unweighted undirected graph developed in [15], which serves as the core algorithm in this work for development of its weighted k-clique version. The set Q that contains the currently growing clique and the set Qmax which contains the largest clique currently found, are global variables. For definitions of all variables see Table 2.

| Line | Pseudocode |
|---|---|
| 1. | **procedure** ColorSort (R) |
| 2. | maxno = 1 |
| 3. | min_$k$ = **max** 1, $\lvert Qmax\rvert - \lvert Q\rvert + 1$ |
| 4. | $j = 0$ |
| 5. | $C[1] = \varnothing$, $C[2] = \varnothing$ |
| 6. | **for** $i = 0$ **to** $\lvert R\rvert - 1$ |
| 7. | $p = R[i]$ |
| 8. | $k = 1$ |
| 9. | **while** $C[k] \cap \Gamma(p) \neq \varnothing$ |
| 10. | $k = k + 1$ |
| 11. | **if** $k >$ maxno |
| 12. | maxno $= k$ |
| 13. | $C[\mathrm{maxno} + 1] = \varnothing$ |
| 14. | $C[k] = C[k] \cup \{p\}$ |
| 15. | **if** $k <$ min_k |
| 16. | $R[j] = R[i]$ |
| 17. | $j = j + 1$ |
| 18. | **if** $j > 0$ |
| 19. | $\mathrm{color}(R[j-1]) = 0$ |
| 20. | **for** $k$ = min_k **to** maxno |
| 21. | **for** $i = 0$ **to** $\lvert C[k]\rvert$ - 1 |
| 22. | $R[j] = C[k][i]$ |
| 23. | $\mathrm{color}(R[j]) = k$ |
| 24. | $j = j + 1$ |

**Table 5**
Expand procedure for the maximum clique algorithm developed in [15] and for the new N highest weight k-cliques algorithm developed in this work. The set Q that contains the currently growing clique, the set Qmax which contains the largest clique currently found, and the set P which contains up to N highest weight k-cliques currently found, as well as parameters K and N, are global variables. For definitions of all variables see Table 2.

| Line | Pseudocode | Algorithm* |
|---|---|---|
| 1. | **procedure** Expand (R) | All |
| 2. | **while** $R \neq \varnothing$ | |
| 3. | $p$ = last vertex in R # choose vertex with highest color and/or cumulative weight | |
| 4. | **if** $\lvert Q\rvert + \mathrm{color}(p) > \lvert Qmax\rvert$ | MCQ(D) |
| 5. | **if** $\lvert Q\rvert + \mathrm{color}(p) \geq K$ **and** weight(Q) + wcolor(p) > lowest_weight(P) | KCQ(D)W |
| 6. | $Q = Q \cup \{p\}$ | All |
| 7. | $Rp = R \cap \Gamma(p)$ | |
| 8. | **if** $Rp \neq \varnothing$ | MCQ(D) |
| 9. | **if** $\lvert Q\rvert < K$ **and** $Rp \neq \varnothing$ | KCQ(D)W |
| 10. | **if** T[level] < Tlimit | MCQD, KCQDW |
| 11. | SortDecDeg (Rp) | MCQD |
| 12. | SortDecDegIncWeigh (Rp, W) | KCQDW |
| 13. | ColorSort (Rp) | MCQ(D) |
| 14. | ColorSortKWeight (Rp) | KCQ(D)W |
| 15. | Expand (Rp) | All |
| 16. | **else if** $\lvert Q\rvert > \lvert Qmax\rvert$ | MCQ(D) |
| 17. | **else if** $\lvert Q\rvert = K$ **and** weight(Q) > lowest_weight(P) | KCQ(D)W |
| 18. | **if** $\lvert P\rvert = N$ | |
| 19. | $P = P \setminus \{Q_{\mathrm{lowest\_weight\_in\_P}}\}$ | |
| 20. | $P = P \cup \{Q\}$ | |
| 21. | $Q = Q \setminus \{p\}$ | All |
| 22. | **else return** | |
| 23. | $R = R \setminus \{p\}$ | |

* »All« denotes that pseudocode in this section (between horizontal lines) is used by all the developed algorithms (weighted and unweighted); MCQ(D): only by unweighted MaxClique and MaxCliqueDyn algorithms; KCQ(D)W: only by K-CliqueWeight and K-CliqueDynWeight.

steps required to find the maximum clique and improve the run time of the algorithm by as much as an order of magnitude on dense graphs, while preserving its performance on sparse graphs.

Until now, in the MaxClique algorithm the calculation of the degrees and sorting of vertices was performed only once with the initial set of vertices R (see line 1 in Table 3). In [15], we developed a new algorithm MaxCliqueDyn that recalculates at certain steps of the Expand procedure, determined heuristically as explained below, the degrees of vertices in Rp in the graph induced by these vertices, i.e., G(Rp); these vertices are then sorted in a decreasing order with respect to their degrees in G (Rp). The ColorSort algorithm thus considers vertices in Rp sorted by their degrees in the induced graph G(Rp) rather than in G. The upper bounds given by this coloring algorithm are tighter with this approach for the steps of the Expand procedure where the degrees are recalculated, than for the steps where the degrees are not recalculated. However, the calculation of degrees is computationally expensive ($O(|Rp|^2)$, therefore we need to determine at which steps this should be performed to decrease the overall running time of the maximum clique search.

The heuristic by which we determine the steps where the recalculation of degrees in G(Rp) and resorting of vertices is performed assumes that the calculation time is improved only when the candidate set Rp is large. Obviously, set Rp is larger on initial (close to root) levels of the recursion of the Expand procedure than on the higher levels (close to leafs). With the recursion level we denote the number of recursive calls of the Expand procedure from the first call to the current branch. For large candidate sets the computational expense related to the computation of tighter bound is much smaller than the cost of investigating false solutions, which arise when applying less tight bounds.

Therefore, we count the number of steps up to and including each level of the recursion in the Expand procedure and also the number of all steps completed so far. Using these two values, we calculate T[*level*], which is the fraction of steps up to the current level among all the steps completed so far (see line 10 in Table 5). With a new heuristic parameter, which we call Tlimit, we can then limit the use of tighter bounds (recalculation of degrees) to levels close to the root. While T[level] is less than Tlimit, we perform the calculations of the degrees and sorting, so that in the ColorSort algorithm (see line 13 in Table 5) we consider vertices in Rp sorted by their degrees in the induced graph G(Rp). The Tlimit parameter is set to 0.025 by default, which limits the calculation of degrees to the lower levels of the recursion where Rp is the largest. We found this value of the Tlimit parameter optimal for random as well as for DIMACS graphs [15,18,19].

In the following subsection, a new algorithm is presented for finding up to N highest weight k-cliques in weighted undirected graphs. This algorithm is used for molecular docking in our developed ProBiS-Dock algorithm [7], but it can, due to its generality, also be applied in other bioinformatics and drug discovery settings, as well as in other research fields and industry.

### *2.5. A new algorithm to find N highest weight k-cliques in a weighted undirected graph*

We introduce a new algorithm K-CliqueWeight, which is an extension of the MaxClique maximum clique algorithm (the core algorithm) developed in [15], and which we have described in the previous section. The new algorithm finds up to N highest weight k-cliques in an undirected vertex-weighted graph. A k-clique is defined as a clique with exactly k vertices, where k can take values between one and the size of a maximum clique in the current graph. The new algorithm is shown in Tables 5 and 6.

The algorithm works by recursively exploring the search tree of the possible weighted k-cliques using a branch-and-bound technique to efficiently prune parts of the search space that cannot contain a weighted k-clique that is among the N highest weight k-cliques. The algorithm consists of the Expand procedure (see Table 5), the recursive procedure that performs the branch-and-bound search, and the new ColorSortKWeight function (see Table 6) that provides the upper bounds to the weight and size of the k-clique that can be found at each step of the search tree.

**Table 6**
A new approximate coloring algorithm ColorSortKWeight for vertex-weighted undirected graphs that is used in the new N highest weight k-cliques algorithm. The set Q that contains the currently growing clique, the set Qmax that contains the largest clique currently found, and the set P that contains up to N highest weight k-clique currently found, as well as the parameter K, are global variables. For definitions of all variables see Table 2.

```
Line  Pseudocode
1.    procedure ColorSortKWeight (R)
2.    maxno = 1
3.    for $i = 0$ to $|R|$ - 1
4.    $p = R[i]$
5.    $k = 1$
6.    $C[1] = \varnothing$, $C[2] = \varnothing$
7.    weight(C[1]) = -inf, weight(C[2]) = -inf
8.    while $C[k] \cap \Gamma(p) \neq \varnothing$
9.    $k = k + 1$
10.   if $k >$ maxno
11.   maxno = $k$
12.   C[maxno + 1] = $\varnothing$
13.   weight(C[$maxno$ + 1]) = -inf
14.   weight(C[$k$]) = max W[$p$], weight(C[$k$])
15.   $C[k] = C[k] \cup \{p\}$
16.   color(p) = $k$
17.   min_k_$w$ = 1
18.   while min_k_$w$ < maxno and $\Sigma_{k=1..min\_k\_w}$(weight(C[$k$])) $\leq$ lowest_weight(P)
      – weight(Q)
19.   min_k_$w$ = min_k_$w$ + 1
20.   min_k_$s$ = max 1, K - |Q|
21.   min_$k$ = max min_k_w, min_k_s
22.   $j = 0$
23.   for $i = 0$ to $|R|$ - 1
24.   if color(R[$i$]) < min_k
25.   R[j] = $R$[i]
26.   $j = j + 1$
27.   if $j > 0$
28.   color(R[$j$]) = 0
29.   for $k$ = min_k to maxno
30.   Ck = {weight(C[1]), weight(C[2]), …, weight(C[$k$])}
31.   Ch = {weight(C[$i$]) | weight(C[$i$]) is among the min_k_s highest weights in Ck}
32.   for $i = 0$ to |C[$k$]| - 1
33.   wcolor(R[$j$]) = $\Sigma_{w \in Ch}$(w)
34.   color(R[$j$]) = $k$
35.   R[j] = C[$k$][i]
36.   $j = j + 1$
```

**Input.** The algorithm takes as input a weighted graph G represented by a set of vertices R, a set of adjacent vertices $\Gamma(v)$ for each vertex $v \in R$, and a set of weights W, where each vertex $v \in R$ has assigned a weight w, which is a positive number ($w > 0$). In addition, it takes as input parameter K, which is the number of vertices in a k-clique that we wish to be found, and parameter N, which is the maximum number of highest weight k-cliques that we wish to be found.

**Output.** The set P containing up to N highest weight k-cliques found in the input vertex-weighted graph sorted by their decreasing weights. If N is not specified or if there are less than N k-cliques in a graph, the algorithm returns all k-cliques in a graph.

**Initialization.** Vertices in the set R (see Table 3) are sorted by their decreasing degrees, then vertices with equal degrees are sorted by their increasing weights. This order of vertices produces the tightest upper bounds to the size of a maximum weight clique in our experiments. This is different from the maximum clique algorithm (see Section 4), where vertices are sorted by their decreasing degrees. The ColorSortKWeight procedure is called once on the input vertices in the set R (line 4 in Table 3). This procedure efficiently determines the initial upper bounds to the size (color) of a clique for each vertex $v \in R$, as well as for the weight (wcolor) of a clique if vertex v is selected to be part of the growing clique. Both these upper bounds are used in the prunning conditions in the Expand procedure.

**ColorSortKWeight procedure: determination of clique weight and size upper bounds.** The ColorSortKWeight procedure takes a set of vertices R as input and partitions these vertices into color classes C, where vertices in the same color class C[k] are not connected by an edge as shown in Table 6. Here, k represents the color of all vertices in color class C[$k$]. For each vertex p∈R, the procedure determines the lowest color k such that no vertex in the k-th color class C[k] is adjacent to p (see line 8 in Table 6). If k is greater than the maximum number of colors seen so far represented by the variable maxno (line 10 in Table 6), a new color class is created. Vertex p is then inserted into this color class k and its color k is assigned to it, i.e., color(p) $= k$. In contrast to unweighted ColorSort (see Table 4), at each step, the weight of color class C[$k$], initially set to negative infinity, is updated to the weight of the vertex p that was inserted into C[k] if the weight of p is larger than the weight of C[k] (line 14 in Table 6). This results in each color class C[k] being assigned the maximum weight of any of its vertices. Thus, any current clique Q consisting of vertices in the set R that is found will have at most k vertices and its weight will be less or equal the sum of the maximum weights of the color classes 1 through k, i.e., weight(Q) $\leq \Sigma_{n=1..k}$(weight (C[$n$])). This condition holds after line 16 in Table 6.

In the next step, the ColorSortKWeight procedure determines the minimum color class (min_k). Vertices with colors less than min_k do not satisfy either of the size or weight conditions and cannot be used to extend the currently growing clique Q. Such vertices are kept in their initial ordering in the resulting set R (see line 25 in Table 6), which was found an efficient strategy [15]. The color class min_k is determined as the maximum of two auxiliary minimum color classes, min_k_s and min_k_w (see line 21 in Table 6), of which min_k_s is the minimum color class below which vertices cannot be used to extend the growing clique because such a clique would not satisfy the size condition (it would always have less than K vertices), and min_k_w is the minimum color class below which vertices cannot be used since such a clique containing such vertices would not satisfy the weight condition (such cliques would have lower weight than the weight of the currently lowest weight clique in set P).

The value of min_k_w is determined iteratively by starting with min_k_$w = 1$ (see line 17 in Table 6) and increasing it to maxno; the search is stopped when the sum of the weights of the k smallest color classes is greater than the difference between the weight of the lowest weight k-clique in set P found so far and the weight of the currently growing clique Q. If vertices from color classes below min_k_w were selected to extend the currently growing clique Q, each such clique would be of lower weight than the lowest weight clique out of the highest weight k-cliques currently in the set P as returned by lowest_weight(P). The size of set P is at most N, so that it can contain at most N k-cliques. However, if there are less than N k-cliques in P (e.g., this occurs at the beginning of the search), then lowest_weight(P) returns negative infinity. This ensures that each k-clique is accepted into set P even if it has a lower weight than those k-cliques already in the set P, until the set P is full.

The second minimum color class min_k_s is calculated as the difference between parameter K and the size of the current clique Q (line 20 in Table 6). It is used to identify vertices that do not satisfy the clique size condition. If a vertex is selected from a color class below min_k_s, each weighted clique containing such a vertex would have less than K vertices. Therefore, such vertices are discarded and not used further in construction of k-cliques.

Vertices in color classes with k greater or equal to the determined min_k can form k-cliques with at least k vertices and with higher weights than the weight of the lowest weight k-clique in the set P. These vertices are copied from their respective color classes C, starting from C[min_k] and ending with C[$maxno$], back to the set R in the order in which they appear in each color class (see line 35 in Table 6). Each such vertex with color of k is assigned a weight upper bound (wcolor), which is the sum of min_k_s highest color class weights (see lines 30–31 in Table 6). On each step of the loop on lines 29–36 in Table 6, a set Ch is constructed (line 31 in Table 6), which is comprised of min_k_s highest color class weights selected from the set Ck which contains weights for color classes 1 to k. A cumulative weight (wcolor) of each vertex in the set R that is going to be used to extend future cliques is then determined as the sum of color class weights in Ch. This results in tight weight upper bounds of these vertices, since min_k_s is always less or equal to K (see line 20 in Table 6), and therefore, the sum of weights in set Ch, which is used as the upper bound, is always less than the sum of weights in Ck.

In addition, clique size upper bounds (colors) are assigned to each vertex with k greater or equal to determined min_k (see line 34 in Table 6). As in the maximum clique algorithm described in Section 4, these upper bounds are used to prune the branches that would give k-cliques with less than K vertices in the Expand procedure, which is described next.

**Expand procedure: finding up to N highest weight k-cliques.** On each step, the recursive Expand procedure selects the last vertex p from set R (see line 3 in Table 5), which is a set of remaining graph vertices yet to be explored. On line 5 in Table 5, the size upper bound (color) and the weight upper bound (wcolor) of vertex p are used to check if adding p to the growing clique Q would increase the size of Q to be greater than or equal to K, and if it would result in a higher weight clique than the lowest weight k-clique in set P. If both of these conditions are true, vertex p is added to Q (line 6 in Table 5).

The subset of vertices Rp⊂R, in which each vertex is adjacent to p, is determined (line 7 in Table 5), and if this set Rp is not empty, the ColorSortKWeight procedure is called with Rp as an argument. This sets the size and weight upper bounds, i.e., colors and wcolors, respectively, for vertices in set Rp. The Expand procedure is then called recursively with Rp as argument. The recursive calls continue until Rp is empty.

If Rp is empty (see line 17 in Table 5), and if both the size of the growing clique Q equals to K and the weight of Q is greater than the weight of the lowest weight k-clique in set P (if P has not yet reached its final size N, then lowest_weight(P) returns negative infinity), the clique Q is inserted into set P by replacing the lowest weight k-clique in P if size of P is N or by adding a new k-clique to set P is not full yet (see lines 17–20 in Table 5). Once there are no more branches to explore, the resulting set P holds up to N highest weight k-cliques that were found in the input vertex-weighted graph. The set P is the output of the algorithm.

## 3. Results and discussion

To evaluate the developed N highest weight k-cliques algorithm K-CliqueWeight and its dynamic variant K-CliqueDynWeight we have tested them on the test set of random weighted graphs as well as on the test set of real-world docking graphs. We have compared our algorithms to the Cliquer algorithm [20,21] for finding a maximum weight clique, which is widely used and well established in the research community.

### 3.1. *Test set of random weighted graphs*

The results for random weighted graphs are in Fig. 3. The total calculation time over all graphs for K-CliqueWeight with the parameter N set to 100 and parameter K set individually to the size of a maximum clique in each of the test graphs, the total calculation time is 8820 s, for K-CliqueDynWeight it is 6612 s, while for Cliquer it is 1362,774 s, resulting in 155x speedup for the K-CliqueWeight and 206x speedup for its dynamic variant K-CliqueDynWeight. These are lower estimates for speedups since we stopped the Cliquer algorithm after 5 days, when it was still calculating three heavy random weighted graph instances, namely the 200/0.95, 200/0.99 and 300/0.99, where each graph name indicates the number of verticies vertices and edge density of a graph. All these graphs are characterized by their high edge densities, where our algorithms performed excellently.

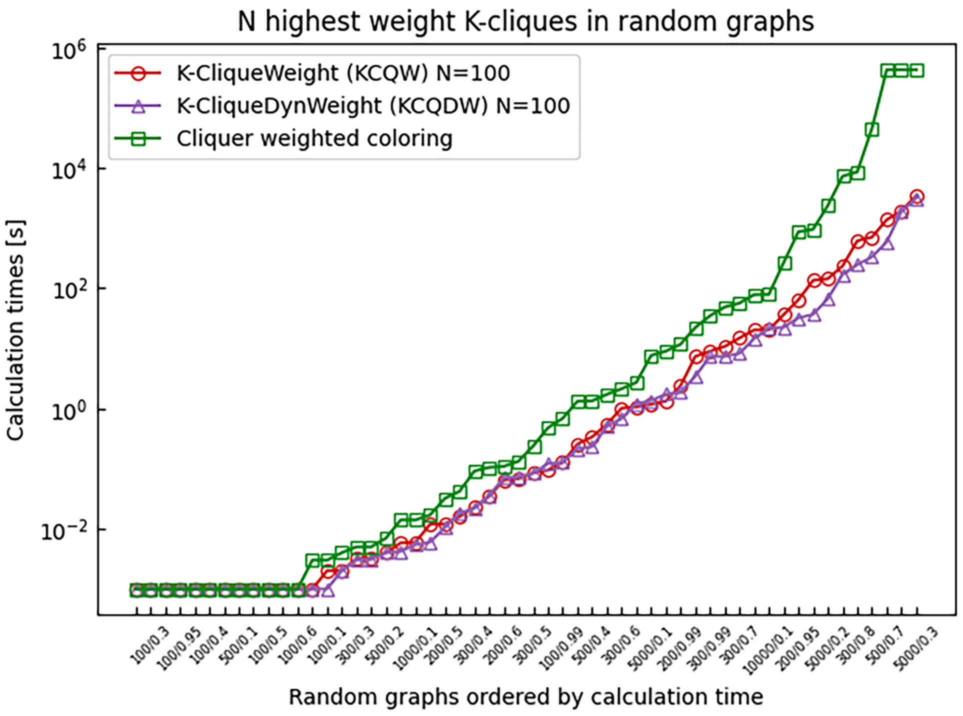

**Fig. 3.** Calculation times for the K-CliqueWeight algorithm and K-CliqueDynWeight variant compared to the state-of-the-art Cliquer algorithm on random weighted graphs. The $N = 100$ indicates that each calculation returned 100 or less highest weight k-cliques; K was set for each graph separately to the size of the maximum clique in that graph. For the sake of clarity, alternating graph labels, valid for the KCQW algorithm, and indicating number of vertices and edge densites are shown beneath the x-axis. Each curve on the graph illustrates calculation times, arranged from smallest to largest, for each algorithm individually. To ensure accuracy, every calculation was iterated 10 times, with graph vertices randomly shuffled before each iteration. The reported calculation time for each data point represents an average across these 10 calculations. This method effectively mitigates the impact of the initial vertex order within the input graph, which could potentially cause variations in clique-finding speed. Calculations completing in under 1 ms were recorded as 1 ms, while those exceeding 5 days were stopped and assigned a fixed calculation time of 5 days.

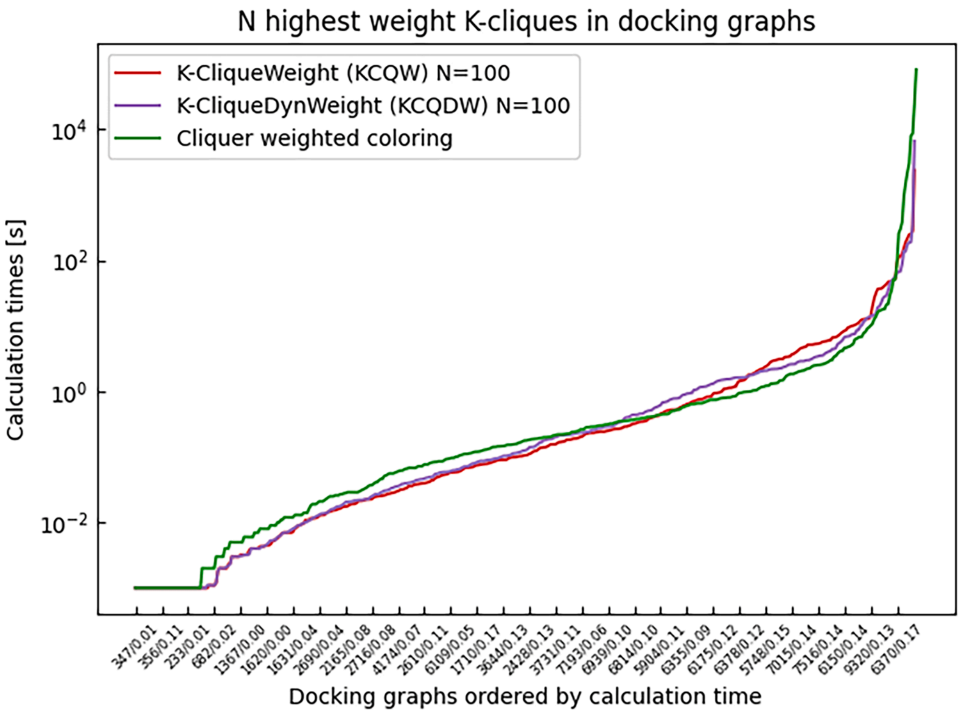

**Fig. 4.** Calculation times for the K-CliqueWeight algorithm and K-CliqueDynWeight variant compared to the state-of-the-art Cliquer algorithm on docking weighted graphs. The $N = 100$ indicates that each calculation was requested to return $\leq$100 highest weight k-cliques; K was set to the size of the maximum clique (unweighted) in each graph separately. For brevity, every 15th graph label (num.vertices/density) is shown below the x-axis. Graph labels are valid for the KCQW algorithm, and each curve represents calculation times sorted from smallest to largest separately for each algorithm. Each calculation was repeated 10-times with randomly shuffled vertices, so that each reported value of the calculation time is an average over 10 calculations. Calculation times that were <1 ms were set to 1 ms.

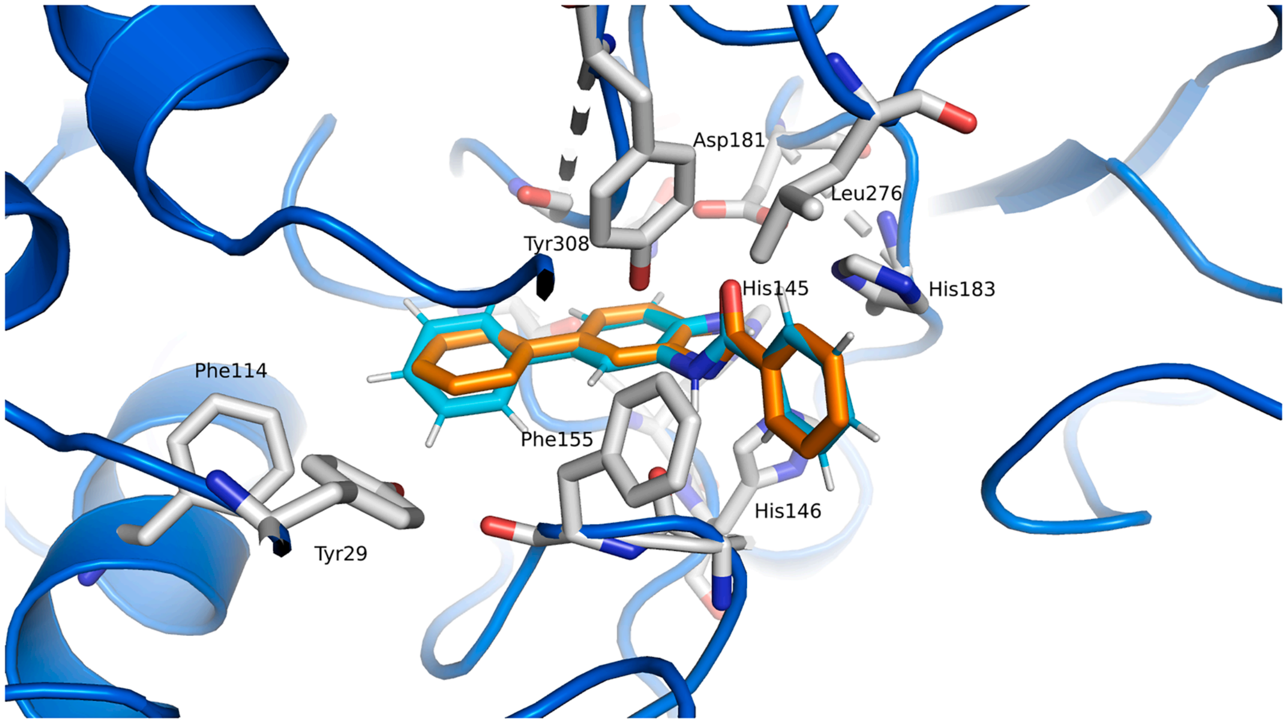

**Fig. 5.** Docked pose of small molecule inhibitor N-(4-aminobiphenyl-3-yl) benzamide (orange sticks) in the protein binding site (pink sticks) of histone deacetylase 2 (blue cartoon) calculated using the ProBiS-Dock algorithm in comparison with the crystal pose of the same inhibitor (cyan sticks).

### *3.2. Real docking graphs*

Next, we tested the developed N highest weight k-cliques algorithm and its variant on docking graphs. These graphs are used in drug discovery for molecular docking and are implemented in the ProBiS-Dock algorithm that enables screening of libraries of millions of small molecules against thousands of proteins [7]. The developed set of docking graphs represent a diverse set of possible drug discovery scenarios (see Methods section for details on how the graphs were generated). Therefore, the results presented are meaningful estimates of real-world performance of our algorithm.

The results for finding N highest weight k-cliques, where the parameter N is set to 100, in docking graphs are in Fig. 4. The K-CliqueWeight algorithm's total calculation time for docking graphs was 5171 s, while K-CliqueDynWeight it was 8838 s, which represents a 25x and 14.8x speedup, respectively, compared to the Cliquer algorithm that achieved a total calculation time of 131,219 s. These results clearly show that our developed algorithm and its variant outperform the reference algorithm by a large margin on docking graphs, confirming their role in drug discovery.

A result of docking a small molecule inhibitor to a protein histone deacetylase 2 (PDB ID: 3max) using the ProBiS-Dock algorithm [7] is shown in Fig. 5. It can be seen that the predicted docked pose of the inhibitor corresponds well to the co-crystallized pose of the same ligand in this case. This result confirms that using K-CliqueWeight algorithm to detect N highest weight k-cliques in docking graphs can be a successful strategy to predict accurate protein-ligand complexes.

A limitation of our test approach is that our algorithm produces solutions that differ slightly from those of the compared maximum weight clique algorithm. While our algorithm identifies N highest weight k-cliques, the compared algorithm returns a single maximum weight clique per input graph. To address this issue, for each test graph, we set the parameter k to match the size of the maximum clique in that particular graph. This ensures the comparability of the results.

## 4. Conclusions

We present a novel algorithm, K-CliqueWeight, designed for efficient identification of the N highest weight k-cliques in vertex-weighted graphs, together with its variant, K-CliqueDynWeight. Our algorithm is general and can find use in various research areas such as molecular docking and beyond. To demonstrate its effectiveness, we have performed tests on both randomly weighted graphs and real docking graphs used in the ProBiS-Dock molecular docking algorithm. Our results show a 155-fold speedup of the K-CliqueWeight algorithm on random weighted graphs and a 25-fold speedup on docking graphs, while for K-CliqueDynWeight variant the relevant speedups are 206-fold and 15-fold compared to the widely used maximum weight clique detection algorithm. The developed algorithm has the potential to significantly accelerate the drug discovery process.

## CRediT authorship contribution statement

**Kati Rozman:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **An Ghysels:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Formal analysis, Data curation, Conceptualization. **Bogdan Zavalnij:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Formal analysis, Data curation, Conceptualization. **Tanja Kunej:** Writing – review & editing, Writing – original draft, Visualization, Validation, Formal analysis, Data curation, Conceptualization. **Urban Bren:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Conceptualization. **Dušanka Janežič:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Janez Konc:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data will be made available on request.

## Acknowledgments

This work was supported by the Slovenian Research and Innovation Agency project grants N1-0142, N1-0209, J1-4414, J1-1715, and L7-8269.

## References

[1] V. Vassilevska, Efficient algorithms for clique problems, Inf. Process Lett. 109 (2009) 254–257, https://doi.org/10.1016/j.ipl.2008.10.014.

[2] S. Szabó, B. Zaválnij, Clique search in graphs of special class and job shop scheduling, Mathematics 10 (2022) 697, https://doi.org/10.3390/math10050697.

[3] G. Palla, D. Ábel, I.J. Farkas, P. Pollner, I. Derényi, T. Vicsek, k-Clique percolation and clustering, B. Bollobás, R. Kozma, D. Miklós (Eds.). Handbook of Large-Scale Random Networks, Springer, Berlin, Heidelberg, 2008, pp. 369–408, https://doi.org/10.1007/978-3-540-69395-6_9.

[4] G. Bass, C. Tomlin, V. Kumar, P. Rihaczek, J. Dulny, Heterogeneous quantum computing for satellite constellation optimization: solving the weighted k-clique problem, Quantum Sci. Technol. 3 (2018) 024010, https://doi.org/10.1088/2058-9565/aaadc2.

[5] B. Balasundaram, S. Butenko, Graph domination, coloring and cliques in telecommunications, M.G.C. Resende, P.M. Pardalos (Eds.). Handbook of Optimization in Telecommunications, Springer US, Boston, MA, 2006, pp. 865–890, https://doi.org/10.1007/978-0-387-30165-5_30.

[6] D.S. Manoharan, A. Sathesh, Patient diet recommendation system using k clique and deep learning classifiers, J. Artif. Intell. Capsule Netw. 2 (2020) 121–130.

[7] J. Konc, S. Lešnik, B. Škrlj, M. Sova, M. Proj, D. Knez, S. Gobec, D. Janežič, ProBiS-Dock: a hybrid multitemplate homology flexible docking algorithm enabled by protein binding site comparison, J. Chem. Inf. Model. 62 (2022) 1573–1584, https://doi.org/10.1021/acs.jcim.1c01176.

[8] J. Konc, S. Lešnik, B. Škrlj, D. Janežič, ProBiS-Dock database: a web server and interactive web repository of small ligand–protein binding sites for drug design, J. Chem. Inf. Model. 61 (2021) 4097–4107, https://doi.org/10.1021/acs.jcim.1c00454.

[9] J. Konc, D. Janežič, ProBiS-Fold approach for annotation of human structures from the alphafold database with no corresponding structure in the PDB to discover new druggable binding sites, J. Chem. Inf. Model. 62 (2022) 5821–5829, https://doi.org/10.1021/acs.jcim.2c00947.

[10] V. Furlan, J. Konc, U. Bren, Inverse molecular docking as a novel approach to study anticarcinogenic and anti-neuroinflammatory effects of curcumin, Molecules 23 (2018) 3351, https://doi.org/10.3390/molecules23123351.

[11] J. Konc, J.T. Konc, M. Penca, D. Janežič, Binding-sites prediction assisting protein-protein docking, Acta Chim. Slov. 58 (2011) 396–401.

[12] L. Banchi, M. Fingerhuth, T. Babej, C. Ing, J.M. Arrazola, Molecular docking with Gaussian boson sampling, Sci. Adv. 6 (2020) eaax1950, https://doi.org/10.1126/sciadv.aax1950.

[13] M.M. Mysinger, M. Carchia, J.J. Irwin, B.K. Shoichet, Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking, J. Med. Chem. 55 (2012) 6582–6594, https://doi.org/10.1021/jm300687e.

[14] A. Gaulton, A. Hersey, M. Nowotka, A.P. Bento, J. Chambers, D. Mendez, P. Mutowo, F. Atkinson, L.J. Bellis, E. Cibrián-Uhalte, M. Davies, N. Dedman, A. Karlsson, M.P. Magariños, J.P. Overington, G. Papadatos, I. Smit, A.R. Leach, The ChEMBL database in 2017, Nucl. Acids Res. 45 (2017) D945–D954, https://doi.org/10.1093/nar/gkw1074.

[15] J. Konc, D. Janezic, An improved branch and bound algorithm for the maximum clique problem, MATCH Commun. Math. Comput. Chem. 58 (2007) 569–590.

[16] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, C.S. Calude, M.J. Dinneen, V. Vajnovszki (Eds.). Discrete Mathematics and Theoretical Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 278–289, https://doi.org/10.1007/3-540-45066-1_22.

[17] N. Biggs, Some heuristics for graph coloring, in: R. Nelson, R.J. Wilson (Eds.), Graph Colourings, Longman, New York, 1990, pp. 87–96.

[18] D.S. Johnson, M.A. Trick, Cliques, coloring, and satisfiability: second DIMACS implementation challenge, Am. Math. Soc. (1993), 1996, https://books.google.com/books?hl=en&lr=&id=-ysHoxUwdMUC&oi=fnd&pg=PP13&dq=David+S+Johnson+and+Michael+A+Trick.+Cliques,+coloring,+and+satisfiability:+second+DIMACS+implementation+challenge,+October+11-13,+1993.+Vol.+26.+American+Math-+ematical+Soc.,+1996.&ots=o4fL8GPnYY&sig=Xropz7WuIemH_0OLihAnZAunMdI. accessed October 19, 2023.

[19] K. Reba, M. Guid, K. Rozman, D. Janežič, J. Konc, Exact maximum clique algorithm for different graph types using machine learning, Mathematics 10 (2022) 97, https://doi.org/10.3390/math10010097.

[20] S. Niskanen, P. Östergård, Cliquer user's guide, version 1.0, Technical Report No. T48 (2003). https://research.aalto.fi/en/publications/cliquer-users-guide-version-10. accessed October 19, 2023.

[21] P.R.J. Östergård, A new algorithm for the maximum-weight clique problem, Nordic J. Comput. 8 (2001) 424–436.