ELTE Eötvös Loránd University

Faculty of Science

Subgraph isomorphism problems

Tamás Takács

Applied mathematician MSc

Supervisor:

Péter Madarasi

Budapest

2025

In this work, we focus on solving subgraph isomorphism problems and as a special case provide an improved algorithm for protein docking. Protein docking is an essential part of the early stages of drug discovery. The goal of protein docking is to predict the position of a small molecule when it is bound to a protein, which helps to identify likely drug candidates. The goal of this project is to improve the ProBis-Dock algorithm to find the best dockings between two protein molecules by implementing kernelization methods.

1 Protein graphs

Proteins can be represented with protein graphs [2]. The vertices of the graph correspond to the functional groups of the amino acids of the protein surface and there is an edge between two distinct vertices if the corresponding groups are sufficiently close to each other.

We can see that embedding a smaller protein into another is an embedding of the smaller protein graph $G_1 = (V_1, E_1)$ into the other protein graph $G_2 = (V_2, E_2)$. Therefore we wish to find an induced subgraph of G_2 that is isomorphic to G_1. There is also a weight for every pair $(v_1, v_2) \in V_1 \times V_2$ and we want to find the embeddings with the highest weights.

With the following construction, we can also handle this problem as a weighted maximum clique problem. The product of two protein graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E, 2)$ is the graph $G = (V_1 \times V_2, E')$, where there is an edge between (u_1, u_2) and (v_1, v_2) if and only if $u_1v_1 \in E_1$ and $u_2v_2 \in E_2$ or $u_1v_1 \notin E_1$ and $u_2v_2 \notin E_2$. A good embedding of the molecule represented by G_1 into the molecule represented by G_2 is a clique of size $k = |V_1|$ in the product graph. Our goal is to find the 100 greatest weight k-cliques in this product graph.

2 Kernelization

When solving a clique problem, it is common practice to use kernelization techniques and reduce the graph in which we are looking for cliques. In this particular problem, we can notice that if $|V_1| = k$, then the graph is k-partite and the k color classes are $\{(u, v) : v \in V_2\}$ for $u \in V_1$ and we are looking for a k-clique, i.e., a clique of size k. This allows us to use not only standard kernelization methods, but also additional reductions only applicable to k-colored graphs.

We show a few reductions [4] we plan to implement to improve the performance of the clique search. Let $G = (V, E)$ be a simple undirected graph and $C_1, \ldots, C_k$ be the color classes. Let the color index of a vertex v of G (with respect to a proper coloring of G) be the number of color classes C_i that contain at least one vertex adjacent with v.

Property 1. *If the color index of v is at most $k-2$, then v can be deleted from G without losing any k-clique.*

Property 2. *If there are color classes C_1 and C_2, and a vertex $v \in V$ such that there is no edge between $N(v) \cap C_1$ and $N(v) \cap C_2$, then v can be deleted without losing any k-clique.*

We can also delete edges using similar ideas. Let the color index of an edge uv be the number of color classes C_i such that $C_i \cap N(u) \cap N(v)$ is non-empty. Let $C_i' = C_i \cap N(u) \cap N(v)$ be the color classes of the neighbours of uv.

Property 3. *If there are two color classes C_i' and C_j', such that there is no edge between them, then the edge uv can be deleted from G without losing any k-clique.*

Note that Properties 2 and 3 can only be used in k-colored graphs when looking for a k-clique.

Let C_1, C_2 be two color classes and let $e_1, \ldots, e_s$ be the edges between them. We construct a new graph G' by deleting the vertices of $C_1 \cup C_2$ and adding extra vertices $u_1, \ldots, u_s$ for the edges $e_1, \ldots, e_s$. The edge set of the graph contains the edges of the original graph between the vertices of $V \setminus (C_1 \cup C_2)$ and for a new vertex u_i: if $e_i = x_i y_i$, then we connect u_i to every vertex of $N(x_i) \cap N(y_i)$. It is easy to see that by coloring the new vertices with a new color, we get a proper coloring of G' with $k - 1$ colors.

Property 4. *G contains a k-clique if and only if G' contains a $(k-1)$-clique.*

3 Our work

In the last semester, we became familiar with the insidrug codebase [3], to which we plan to add an improved clique search algorithm. We made minor adjustments to the already existing algorithm and added a naive weighted click search.

In this semester, we took the first step by adding a heuristic to our naive weighted clique search, based on Property 1. To be able to measure the efficiency of our new clique search and further improvements we plan to implement in the future, we needed to create a test dataset, on which we can easily test the correctness and runtime of our algorithms.

To create these test cases, we generated random graphs G, and iteratively chose induced subgraphs H, by deleting one vertex in each iteration from the previous H, starting from $H = G$. We repeated this process until the number of embedding of H into G reached a desired interval (usually between 50 and 200). Then we created the product graph of H and G as described in Section 1 and listed all maximum size cliques of the product graph (which we got from the embeddings of H into G), and calculated their weights according to random non-negative integer weights generated for the vertices. To find all embeddings of H into G, we used the VF2++ algorithm, implemented in the LEMON graph library [1].

Our naive clique search uses backtracking in such a way, that in each step we color the remaining vertices greedily, and choose our vertex from the color class with the least cardinality. In each step there is an ordering of the vertices such that the color classes form intervals, and they follow each other with non-increasing cardinality. The coloring

also gives an upper bound for the size of a clique we can possibly get if we choose a given vertex in the next step. However, this upper bound can be improved by Property 1, as the number of color classes that contain a neighbor of the given vertex is also an upper bound for the size of a clique, possibly eliminating more choices for the next vertex. Although this heuristic helps to reduce the size of the search tree compared to the naive approach it also comes with an additional computational cost, and as shown in the table below, this actually makes the heuristic approach slower in practice than our naive algorithm.

Vertices	Clique size	Naive time	Heuristic time
300	10	$1,209\ s$	$1,289\ s$
360	12	$1,222\ s$	$1,296\ s$
360	12	$1,385\ s$	$1,499\ s$
390	13	$8,740\ s$	$9,964\ s$
420	14	$10,388\ s$	$11,745\ s$
450	15	$12,575\ s$	$13,841\ s$
546	13	$14,674\ s$	$16,193\ s$
588	14	$45,208\ s$	$50,172\ s$
588	14	$150,040\ s$	$158,158\ s$
588	14	$258,531\ s$	$273,352\ s$
630	15	$100,544\ s$	$109,181\ s$
630	15	$916,733\ s$	$968,657\ s$
756	18	$510,858\ s$	$574,866\ s$

The usage of Property 1 can be extended to the weighted setting the following way: If for a given vertex v, we sum up the weight of their highest weight neighbor from each color class and add the weight of v, then we get an upper bound for the weight of a clique if we choose v in the next step. Implementing this should further decrease the possible choices for choosing the next vertex, therefore decreasing the size of the search tree even more.

4 Future plans

Implementing heuristics based on Properties 2 - 4 could also further reduce the size of the search tree and perhaps make our algorithm run faster. Of course, these properties rely on the fact that the graph is colored with k colors and we are looking for a k-clique. In our algorithms, we used greedy colorings that may use more than k colors, therefore we have find analogous properties for the greedily colored graph.

The k-cliques in the product graph do not always provide a feasible docking, because the graph representation does not contain every necessary information about the molecules. In the future we plan to figure out what extra information we should add to the graph so that we do not get infeasible embeddings.

It is easy to see that the k-clique problem we solve is in fact a subgraph isomorphism problem, therefore it is a logical next step to look at how we can use the kernelization techniques discussed here to solve a subgraph isomorphism problem more efficiently.

References

[1] LEMON: Library for Efficient Modeling and Optimization in Networks. `http://lemon.cs.elte.hu`.

[2] Janez Konc and Dušanka Janežic. Protein- protein binding-sites prediction by protein surface structure conservation. *Journal of Chemical Information and Modeling*, 47(3):940–944, 2007.

[3] Implementation of the ProBis-Dock algorithm. `https://gitlab.com/janezkonc/insidrug`, Accessed: 2025.

[4] Sándor Szabó and Bogdán Zaválnij. Clique search in graphs of special class and job shop scheduling. *Mathematics*, 10(5):697, 2022.