A Novel Real-Time Editor for Protein-Ligand Binding Affinity Prediction Using Structural Comparison

Final Master's Project (TFM)

Master's Degree in Computer Security Engineering and Artificial Intelligence

Departament d'Enginyeria Informàtica i Matemàtiques

UNIVERSITAT ROVIRA i VIRGILI

Kuldeep Shivaji Parade Patil

Advisor: Dr. Francesc Serratosa
Universitat Rovira i Virgili
francesc.serratosa@urv.cat

Tarragona, Spain
January 21, 2025

Contents

List of Figures

List of Tables

Acknowledgements

This thesis would not have been possible without the generous help and guidance of my thesis supervisor, Dr. Francesc Serratosa. I would like to extend my sincere gratitude for giving me the opportunity to do this work and learn from it.

Finally, I would like to express my thankfulness to my amazing parents and family. My family has been amazing with me and guiding me along the way. Also my friends, especially Sanket More who has helped a lot in these past few years. To my sister Swapna, thank you for everything.

Nomenclature

Symbols

$\beta_{i,j}$ Attribute of edge $e_{i,j}$.

$t_{i,j}$ t^{th} attribute of edge $e_{i,j}$.

γ_i Attribute of node v_i.

t t^{th} attribute of node v_i.

$+$ $\{x \in \mathbb{R} : x \geq 0\}$.

I^* Cost of inserting a star.

Del_v Set of all the node deletions.

$e'_{i,j}$ Edge between the i^{th} and j^{th} nodes in graph G'.

$e_{i,j}$ Edge between the i^{th} and j^{th} nodes in graph G.

G Graph.

G' Graph.

Ins_v Set of all the node insertions.

$M_{n\times m}(\mathbb{R}^+)$ Set of square matrices with n rows and m columns with coefficients in $\mathbb{R}^+$.

$\mathbf{0}$ Zero.

s.t. Such that.

$Subs_v$ Set of all the node substitutions.

v'_i i^{th} node in graph G'.

v_i i^{th} node in graph G.

$w_e = (w_e^1, \ldots, w_e^M)$ Vector of edge attribute weights.

$w_v = (w_v^1, \ldots, w_v^N)$ Vector of node attribute weights.

Acronyms

GED Graph Edit Distance.

PR Pattern Recognition.

NN Neural Network.

KNN K-Neural Network.

Chapter 1

Abstract

The prediction of protein-ligand binding affinity is a key step in accelerating the discovery of novel therapeutic compounds. This thesis explores a method based on structural comparison of proteins and ligands using Graph Edit Distance (GED) and molecular descriptors. The proposed approach includes a real-time graphical editor to visualize and compare compounds, a K-Nearest Neighbors (KNN) model for affinity prediction, and detailed error analysis. Experimental results on the SARS-CoV-2 main protease dataset demonstrate the effectiveness of the proposed method. Future work will explore integration with deep learning techniques for further refinement.

This thesis is part of a broader research initiative aimed at incorporating proteins into cheminformatics analysis. While traditionally proteins are represented as sequences to mitigate the substantial storage and computational requirements of treating them as graphs, this work emphasizes structural comparison approaches. The research aligns with ongoing efforts at the Universitat Rovira i Virgili (URV) to explore novel methodologies for drug discovery and bioinformatics, as detailed in the project website: `https://www.cheminformatics-nutrition.recerca.urv.cat/en/next-pandemics`.

Predicting protein-ligand binding affinity presents a viable solution for accelerating the discovery of new chemical compounds. Some methods consider the structure of both the drug and the protein, leveraging deep neural networks and graph neural networks. The aim of this master thesis is to utilize structural comparison techniques applied to proteins and drugs with known binding affinities. This approach focuses on analyzing which parts of the drug and protein are most critical for binding affinity and predicting their interaction strength. The database used in this study, generated at the Universitat Rovira i Virgili (URV), consists of several pairs of the main protease of SARS-CoV-2 and corresponding drugs.

Graphs, as abstract data structures, provide a powerful framework for modeling complex problems through nodes and edges, representing points of interest and their relationships, respectively. The versatility of graphs lies in their ability to attribute features to nodes and edges, enabling applications in computer vision, bioinformatics, and network analysis. Graph Edit Distance (GED) is a valuable tool in structural pattern recognition, offering a measure of dissimilarity between attributed graphs. However, its effectiveness hinges on the precise definition of substitution, deletion, and insertion costs for nodes

and edges, which determines the similarity of graphs.

This thesis is divided into three parts:

1. **Real-Time Structural Comparison**: A novel, interactive editor was developed for **real-time comparison** of protein-ligand structures. Users can load and visualize any two `.sdf` files, adjust GED parameters, and compute GED for various scenarios such as edge deletion, node substitution, and node insertion. This functionality enables intuitive exploration of structural differences and their impact on binding affinity.

2. **Batch Analysis with KNN**: The editor extends to **batch processing**, allowing the comparison of a selected `.sdf` file against a folder of compounds containing binding affinity data. Using GED and KNN, the system identifies the most similar compound, visualizes its structure, and displays its binding affinity.

3. **Quantitative Analysis of Prediction Methods**: Two datasets (test and training) are compared to compute a matrix of GED distances and affinity differences. The analysis integrates alternative chemistry-based metrics, such as Mordred descriptors, and visualizes the relationship between structural and affinity differences through scatter plots.

By combining **structural comparison**, **machine learning techniques**, and **visualization tools**, this work demonstrates a comprehensive and interpretable approach to affinity prediction. The results validate the proposed methods and underscore their potential in computational drug discovery.

Also, We aim to evaluate whether structural analysis of ligands is sufficient to predict their biological activity, specifically the pIC50 values.

Rationale for Structural Comparison

Our objective is to determine the effectiveness of structural analysis techniques in capturing the relationship between ligand structure and activity. To this end, we compare traditional structural methods such as Graph Edit Distance (GED) and descriptor-based methods like Mordred.

Comparative Analysis

The evaluation involves applying K-Nearest Neighbors (KNN) to both GED-based and Mordred-based structural representations, analyzing their predictive performance.

Benchmark

A Mordred descriptor-based approach integrated with a neural network serves as the reference standard (ground truth). This benchmark is used to assess and compare the efficacy of GED and KNN methodologies against a more advanced predictive model.

Chapter 2

Introduction

The interaction between proteins and ligands forms the foundation of numerous biological processes and drug discovery efforts. Proteins act as molecular machines within cells, while ligands—small molecules or ions—bind to these proteins to either enhance or inhibit their biological activity. Understanding and predicting the binding affinity between a protein and a ligand are pivotal in designing drugs with high efficacy and specificity. Accurate prediction methods can significantly accelerate the drug discovery process, reduce experimental costs, and enable the identification of novel therapeutic compounds.

This thesis is part of a broader research initiative aimed at incorporating proteins into cheminformatics analysis. While traditionally proteins are represented as sequences to mitigate the substantial storage and computational requirements of treating them as graphs, this work emphasizes structural comparison approaches. The research aligns with ongoing efforts at the Universitat Rovira i Virgili (URV) to explore novel methodologies for drug discovery and bioinformatics, as detailed in the project website: `https://www.cheminformatics-nutrition.recerca.urv.cat/en/next-pandemics`.

2.1 Challenges in Protein-Ligand Binding Affinity Prediction

Predicting binding affinity is inherently complex due to the multifaceted nature of protein-ligand interactions. These interactions are influenced by:

- **Structural Compatibility**: The three-dimensional shapes of the protein and ligand must complement each other.
- **Chemical Properties**: Functional groups on the ligand and their interactions with the protein play a critical role.
- **Dynamic Environment**: Proteins and ligands are not rigid entities; they exhibit flexibility that impacts binding.
- **Biological Context**: Environmental factors such as pH, temperature, and the presence of cofactors also affect binding.

Traditional computational approaches rely heavily on experimental data, docking simulations, or machine learning models trained on large-scale datasets. While effective,

these methods often lack interpretability, making it difficult to pinpoint which structural features of the ligand or protein drive binding affinity.

2.2 Motivation and Objectives

This thesis proposes a novel approach that emphasizes the interpretability and structural insights of protein-ligand interactions. By leveraging **Graph Edit Distance (GED)** and molecular descriptors, we aim to provide a method that not only predicts binding affinity but also identifies critical regions of the ligand and protein contributing to their interaction.

Additionally, this work is part of a larger research project at the Universitat Rovira i Virgili (URV) that seeks to integrate protein structures into cheminformatics analysis. Representing proteins as graphs offers a powerful framework for structural analysis but introduces challenges due to substantial storage and computational requirements. Consequently, proteins are often treated as sequences instead. This thesis aims to address these challenges by focusing on structural comparison methodologies and contributing to ongoing research efforts in drug discovery and bioinformatics, as outlined in the project: `https://www.cheminformatics-nutrition.recerca.urv.cat/en/next-pandemics`.

The specific objectives of this thesis are as follows:

1. **Visualization and Comparison**: Develop an interactive editor to visualize and compare protein-ligand structures, highlighting their differences.

2. **Graph-Based Structural Comparison**: Implement a graph-based approach using GED to quantify structural dissimilarities between molecules.

3. **Binding Affinity Prediction**: Use the **K-Nearest Neighbors (KNN)** algorithm to predict binding affinity based on GED and molecular descriptors.

4. **Error Analysis and Validation**: Analyze the method's performance by evaluating prediction accuracy, errors, and the influence of structural features.

2.3 Proposed Approach

The proposed method integrates structural comparisons with predictive modeling, focusing on interpretability. Proteins and ligands are represented as graphs, where nodes correspond to atoms, and edges represent bonds. GED serves as the backbone of structural comparison, offering a robust framework to evaluate molecular differences through node and edge edit operations such as insertions, deletions, and substitutions.

In addition to GED, molecular descriptors computed using Mordred provide complementary information about chemical properties. These descriptors capture essential features like molecular weight, hydrogen bond donors/acceptors, and topological polar surface area. Combined with GED, they form a comprehensive feature set for training the KNN model.

The inclusion of a graphical user interface (GUI) enhances user interaction, enabling visualization of GED calculations, structural differences, and affinity predictions. This integration bridges the gap between raw computational results and actionable insights for researchers.

2.4 Significance of the Research

The importance of this work lies in its dual focus on prediction and interpretability. While contemporary deep learning methods have achieved state-of-the-art results in binding affinity prediction, they often function as "black-box" models, obscuring the underlying factors driving predictions. In contrast, our approach explicitly identifies structural and chemical features that contribute to binding affinity, providing actionable insights for drug design.

Furthermore, the use of the SARS-CoV-2 main protease dataset underscores the relevance of this research. The COVID-19 pandemic has highlighted the urgent need for computational tools to accelerate drug discovery and repurposing. By applying this method to the SARS-CoV-2 protease, we aim to demonstrate its utility in addressing real-world challenges.

2.5 Thesis Structure

This thesis is structured as follows:

- **Chapter 3: Background and Related Work** reviews existing methods for binding affinity prediction, including graph-based approaches and machine learning techniques.
- **Chapter 4: Methods** describes the theoretical foundation of GED, molecular descriptors, and the KNN algorithm, along with the design of the graphical editor.
- **Chapter 5: Implementation and Results** provides details of the computational pipeline, including dataset preprocessing, model training, and results visualization.
- **Chapter 6: Analysis and Discussion** evaluates the method's performance, identifies trends in error reduction, and compares the approach to other methods.
- **Chapter 7: Conclusion and Future Work** summarizes the contributions of this thesis and outlines potential avenues for further research.

2.6 Thesis Composition

2.6.1 Part 1: Real-Time Ligand Comparison

Interactive GUI Editor

- Allows dynamic selection of two `.sdf` files.
- Provides options to customize GED costs: node/edge substitution, insertion, and deletion.

Visualization

- Displays ligand structures, differences, and compound names.

Real-Time GED Calculation

- Handles cases like edge deletion, node substitution/insertion/deletion, and edge insertion.
- Outputs user-friendly descriptions of structural changes.

2.6.2 Part 2: Database-Wide GED and Affinity Analysis

Editor Features

- Options to select a single `.sdf` file or a folder with `.sdf` files and `affinity.txt`.

GED and Affinity Analysis

- Computes GED for a selected ligand vs. all ligands in the folder.
- Visualizes the ligand with the minimum GED and its affinity.

KNN Analysis

- Uses 75% training and 25% testing.
- Predicts affinities, computes errors, and visualizes results with bar plots.
- Calculates Mean Square Error (MSE).

2.6.3 Part 3: Statistical Analysis

Input

- Two folders (`test` and `learn`) with `.sdf` files and `affinity.txt`.

Analysis

- Computes GED and affinity difference matrices.
- Visualizes scatter plots of GED vs. affinity differences, trending toward a line with more training data.

Experimentation

- Varies substitution costs (e.g., 0.5) to analyze impact.

2.6.4 Extended Analyses

Descriptor-Based Analysis

- Uses Mordred descriptors and Euclidean distances for affinity prediction.

Neural Network Model (Optional)

- Uses Mordred vectors to predict binding affinity with 75% training and 25% testing.

Metrics

- Scatter plots: GED vs. affinity, Mordred vs. affinity.
- KNN MSE: GED vs. KNN, Mordred vs. KNN.

By the end of this work, we aim to deliver a robust and interpretable method for protein-ligand binding affinity prediction, paving the way for more transparent and effective computational tools in drug discovery.

Chapter 3

Background and Related Work

3.1 Protein-Ligand Binding Affinity

Protein-ligand binding affinity quantifies the strength of the interaction between a protein and a ligand, typically expressed using metrics such as the equilibrium dissociation constant (K_d) or inhibition constant (K_i). Accurate prediction of binding affinity is critical in drug discovery, as it determines the efficacy and potency of potential therapeutic compounds.

Traditional methods for predicting binding affinity include molecular docking and molecular dynamics simulations. While effective, these approaches can be computationally expensive and may not always capture the complexity of protein-ligand interactions. To address these limitations, machine learning techniques have been increasingly adopted. For example:

- **Deep Neural Networks (DNNs)**: Capable of modeling non-linear relationships in biological data, capturing complex interaction patterns.
- **Graph Neural Networks (GNNs)**: Particularly suitable for handling molecular data by modeling the graph structure of molecules, where atoms are represented as nodes and bonds as edges.

These methods facilitate high-throughput and accurate prediction of binding affinities, enabling the rapid evaluation of potential drug candidates.

3.1.1 What is Binding Affinity?

Binding affinity is the strength of the binding interaction between a single biomolecule (e.g., a protein or DNA) and its ligand or binding partner (e.g., a drug or inhibitor). Binding affinity is typically measured and reported by the equilibrium dissociation constant (K_D), which is used to evaluate and rank order strengths of bimolecular interactions. The smaller the K_D value, the greater the binding affinity of the ligand for its target. Conversely, the larger the K_D value, the weaker the interaction between the target molecule and ligand.

Binding affinity is influenced by non-covalent intermolecular interactions such as hydrogen bonding, electrostatic interactions, and hydrophobic and van der Waals forces between the two molecules. Additionally, the binding affinity between a ligand and its target molecule may be affected by the presence of other molecules.

3.1.2 Why Should Binding Affinity Be Measured?

Whenever characterizing proteins, nucleic acids, compounds, or any biomolecule, understanding their binding affinity to substrates, inhibitors, and cofactors is crucial for appreciating their intermolecular interactions. This understanding is particularly relevant when studying interactions such as enzymatic reactions, protein complexes, or receptor binding.

In drug discovery, binding affinity is a key factor for designing drugs that bind their targets selectively and specifically.

3.1.3 The Role of Proteins and Ligands in Drug Discovery

Proteins are vital biomolecules that play diverse biological roles in living organisms. Human life depends on proteins for functions such as material transportation, catalysis, information exchange, immune defense, oxidation, functional maintenance, and maintaining acid–base balance. However, proteins cannot complete life activities independently. Instead, they must combine with other biomolecules like RNA, DNA, other proteins, and organic and inorganic molecules to perform specific functions. For instance, most human kinases must bind to adenosine triphosphate (ATP) molecules to achieve various intracellular signaling and metabolic processes. Mutations in human kinase protein residues may lead to abnormal kinase activity, causing hundreds of diseases, such as cancer and Parkinson's. Therefore, developing small organic molecules targeting kinase proteins to regulate kinase activity and treat diseases is necessary.

This work focuses mainly on ligands (small organic molecules), as most drugs are organic small molecules. For example, vemurafenib and SJF-0628, as shown in Figure A, B, C, D are two inhibitors targeting human serine/threonine-protein kinase B-raf (BRAF). The former, a U.S. Food and Drug Administration (FDA)-approved inhibitor, is widely used to treat late-stage or metastatic melanoma with the BRAF V600E mutation. The latter reduced the expression of all tested human BRAF mutants (V600E, K601E, and G466E) but not the wild-type human BRAF in cell experiments.

3.1.4 Binding Pockets and Their Importance

The residues of proteins that interact with ligands are called binding sites or binding pockets. As shown in Figure, the binding pocket refers to a cavity or groove on the surface of the target protein where the ligand binds and interacts with the protein. Most small-molecule drugs exert their effects in the body through the classic lock-and-key model, where drugs act as substrates and bind to the pockets, inhibiting or activating specific protein functions. Therefore, pockets are a significant focus for researchers in molecular docking and drug design.

For ligands to achieve their functions, they must match their targeted pockets in space to avoid atomic space conflicts, and they should be tightly bound to the pocket with strong interactions to prevent the influence of thermal movement. Drug molecules bind to active sites through hydrogen bonds, electrostatic interactions, hydrophobic interactions, and relatively rare covalent bonds. The interaction strength of ligand binding to a protein is quantified by binding affinity.

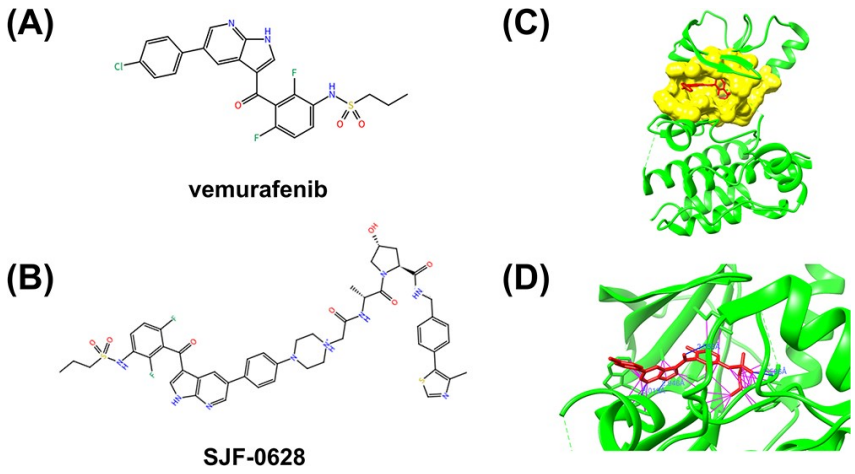

Figure 3.1: (A) The 2D chemical structure of vemurafenib. (B) The 2D chemical structure of SJF-0628 consisting of vemurafenib, a short linker, and a Von Hippel Lindau (VHL)-recruiting ligand. (C) The structure of the BRAF-vemurafenib complex. BRAF kinase, ligand vemurafenib, and identified pocket are colored in green, red, and yellow, respectively. (D) The interaction between vemurafenib and BRAF kinase, where hydrogen bonds and other contacts are shown as blue and purple lines, respectively.

3.1.5 Drug Design and Screening

The intrinsic activity of a drug refers to its ability to produce pharmacological effects after binding to its receptor. Drugs can be classified into two categories: agonists and antagonists. Agonists are drugs with binding affinity and intrinsic activity, which bind to receptors and stimulate them to produce effects. Agonists are further divided into complete agonists and partial agonists. Complete agonists, such as morphine, have strong binding affinity and intrinsic activity, while partial agonists, such as pentazosin, have strong binding affinity but weak intrinsic activity. Antagonists, on the other hand, bind to receptors with strong binding affinity but lack intrinsic activity. They do not produce effects themselves but exert antagonistic effects on agonists by occupying their binding sites, such as naloxone and propranolol.

Both agonists and antagonists must tightly bind to target proteins, requiring high binding affinity. Therefore, the initial step in drug discovery is identifying small candidate ligands with high binding affinity to their target proteins. These candidates are further screened and optimized to identify lead compounds. Protein–ligand binding affinity is thus one of the critical indicators in drug screening.

3.2 Graph Edit Distance

Graph Edit Distance (GED) is a flexible and widely used measure for quantifying the similarity or dissimilarity between two graphs. GED is defined as the minimum cost of a sequence of edit operations—such as insertion, deletion, or substitution of nodes and edges—needed to transform one graph into another. The total cost is determined by a predefined cost function, which can be adapted to specific applications.

Computing the exact GED is NP-hard, making it computationally challenging for large graphs. To address this, approximate algorithms have been developed to reduce complexity while maintaining acceptable accuracy. One such approach, Serratosa's method,

introduces heuristics to minimize the search space and expedite GED computation.

In cheminformatics, GED is particularly valuable for comparing molecular structures, where graphs represent molecules with nodes as atoms and edges as bonds. By quantifying structural similarity, GED enables the evaluation of molecular properties and interactions, which are essential for understanding protein-ligand binding.

The advance of computation capacities are permitting that the graph representation of data becomes popular in the field of Pattern Recognition (PR). When PR problems are addressed with graphs, error-tolerant graph matching techniques outcome useful. The aim of these techniques is to find the best mapping between nodes and edges that minimises some kind of objective function that is adapted to the application. In this chapter we present the main concepts related with the graph matching problem and with the definition of the *Graph Edit Distance*.

3.3 Graph Matching and Graph Edit Distance

One of the most used frameworks to define the error-tolerant graph matching is through the *Graph Edit Distance (GED)* (Bunke and Allermann (1983); Sanfeliu and Fu (1983b); Sanfeliu et al. (2002); Stauffer et al. (2017); Gao et al. (2010a)). The main idea of the *GED* is to define the difference between two graphs as the amount of distortion required to transform a graph into another one. Calculating the *GED* between a pair of attributed graphs G and G' consists in finding the best sequence of edit operations that converts one graph into another with the minimum cost that is application dependent. To continue with the detailed definition of the *GED*, we introduce some concepts and notation that are required.

Given an attributed graph G, we call v_i to the i-th node in G and $e_{i,j}$, the edge between node v_i and node v_j in G. We define γ_i as the attribute of node v_i and γ_i^t as the t-th attribute of node v_i, and $\beta_{i,j}^t$ is the t-th attribute of edge $e_{i,j}$. Given a node $v_i \in G$, each node v_j connected with v_i is called *neighbour* of v_i. We can consider the set of all the neighbours of v_i and we call it *Neighbourhood of* v_i. We call *degree* of v_i, d_i, to the number of neighbours of v_i and we define the *Star* centered on the node v_i, $Star(v_i)$, as the local structure composed of the node itself, its neighbours, and the correspondent edges connecting them. Similarly, given another attributed graph G', we define v_i' and $e_{i,j}'$ as the i-th node and the edge between nodes v_i' and v_j' respectively, and all the concepts are defined in a similar way than in G.

3.3.1 Definition of Graph Edit Distance

Having a pair of graphs, G and G', a node-to-node mapping $f : G \to G'$ between nodes of both graphs, is a bijective function that assigns one node of G to only one node of G'. We suppose both graphs have the same number of nodes since they have been expanded with new nodes, called *Null*. Therefore, if n and m were the initial number of nodes of G and G' respectively, the final number of nodes in both graphs is $n + m$. We use the notation $f(a) = i$ to represent the mapping from node v_a to node v_i'. Note that the mapping between edges is imposed by the mapping of the nodes whose edges are connected. We say that $f(a) = i$ is a node substitution if both nodes are not *Null*. It is an insertion if

node v_a is *Null* and v'_i is not *Null*. Finally, it is a deletion if node v'_i is *Null* and v_a is not *Null*. Similarly happens with the edges.

To quantify the importance of these operations transforming the graphs, a cost is assigned to each operation depending on the attributes on the involved nodes or edges. The cost of substituting a node $v_a \in G$ by a node $v'_i \in G'$ is denoted by $C_S^v(a,i)$. The cost of deleting a node v_a is $C_D^v(a)$. Finally, the cost of inserting the node v'_i is $C_I^v(i)$. Similarly happens with the costs on the edges: The cost of substituting an edge $e_{a,b}$ by an edge $e'_{i,j}$ is $C_S^e(a,i,b,j)$. The cost of deleting an edge $e_{a,b}$ is $C_D^e(a,b)$. And finally, the cost of inserting an edge $e'_{i,j}$ is $C_I^e(i,j)$. These costs are defined in equations 2.1 and 2.2:

$$C^v(i,a) = \begin{cases} C_S^v(i,a) & \text{if } v_i \neq \text{Null} \wedge v'_a \neq \text{Null} \\ C_D^v(i) & \text{if } v_i \neq \text{Null} \wedge v'_a = \text{Null} \\ C_I^v(a) & \text{if } v_i = \text{Null} \wedge v'_a \neq \text{Null} \end{cases} \tag{2.1}$$

$$C^e(i,j,a,b) = \begin{cases} C_S^e(i,j,a,b) & \text{if } e_{i,j} \neq \text{Null} \wedge e'_{a,b} \neq \text{Null} \\ C_D^e(i,j) & \text{if } e_{i,j} \neq \text{Null} \wedge e'_{a,b} = \text{Null} \\ C_I^e(a,b) & \text{if } e_{i,j} = \text{Null} \wedge e'_{a,b} \neq \text{Null} \end{cases} \tag{2.2}$$

The cost of transforming graph G into graph G' through the mapping f is defined as:

$$\text{Cost}(G,G',f) = \sum_{v_i \in G} C^v(i,f(i)) + \sum_{e_{i,j} \in G} C^e(i,j,f(i),f(j)) \tag{2.3}$$

The *Graph Edit Distance (GED)* is defined as the minimum transformation cost:

$$\text{GED}(G,G') = \min_{f:G \to G'} \left\{ \sum_{v_i \in G} C^v(i,f(i)) + \sum_{e_{i,j} \in G} C^e(i,j,f(i),f(j)) \right\} \tag{2.4}$$

3.3.2 Computing the Graph Edit Distance

The computation of the *GED* is an optimisation problem that is NP-hard, meaning that it cannot be solved in polynomial time. The optimal computation of the *GED* is usually carried out by means of the A* algorithm (Hart et al. (1968)). The computational cost of this method is exponential in the number of nodes of the involved graphs (Garey and Johnson (1990)).

For this reason, some heuristic algorithms that deduce a sub-optimal *GED* in polynomial time have been presented (Justice and III (2006); Neuhaus et al. (2006); Riesen and Bunke (2009); Serratosa (2014b, 2015); Riesen et al. (2018)). In general, these sub-optimal algorithms optimise local instead of global criteria, and a sub-optimal *GED* can be computed. One of the most extended algorithms to compute the *GED* is Bipartite graph matching (Riesen and Bunke (2009); Serratosa (2014a)), which is the algorithm that we have used in this thesis to compute approximations of the *GED*. This algorithm defines a cost matrix, applies a linear solver such as the Hungarian method, and deduces the correspondence f with the sub-optimal *GED*. For more details, the reader is referred to Riesen and Bunke (2009) and Serratosa (2014a).

These sub-optimal algorithms define the cost between two graphs for a specific node-to-node mapping, f, as the addition of the substitution, deletion, and insertion costs of local structures of nodes. One of these local structures that can be used is the *Star* centered

in v_i, $Star(v_i)$, defined in the second paragraph of Subsection 2.2. We represent the sets of substitutions, deletions, and insertions of nodes respectively with $Subs_v$, Del_v, and Ins_v. Moreover, $C^v_{Star_S}(a,i)$ denotes the cost of substituting the *Star* centered at node $v_a \in G$ by the *Star* centered at node $v'_i \in G'$. $C^v_{Star_D}(a)$ denotes the cost of deleting the *Star* centered at v_a, and $C^v_{Star_I}(i)$ denotes the cost of inserting the *Star* centered at v'_i. These *Star* costs depend on the costs on nodes and edges $C^v_S(a,i), C^v_D(a), C^v_I(i), C^e_S(a,i,b,j), C^e_D(a,b)$, and $C^e_I(i,j)$ (Serratosa and Cortés (2015a)).

Then, we can define a sub-optimal cost of the mapping f between G and G' as:

$$\text{Cost}(G,G',f) = \sum_{v \in Subs_v} C^v_{Star_S}(a,i) + \sum_{v \in Del_v} C^v_{Star_D}(a) + \sum_{v \in Ins_v} C^v_{Star_I}(i) \tag{2.5}$$

Consequently, a sub-optimal *GED* can be calculated finding a mapping that minimises Equation 2.5.

3.3.3 Learning the Costs of the Graph Edit Distance

The penalty costs are application-dependent and need to be set such that the *GED* reflects the dissimilarity between the graphs. These costs can be manually tuned or automatically computed through a learning method. In this section, we select several learning methods and we classify them into three classes, depending on the nature of the edit costs they learn. We present below a summary with the descriptions of these three classes of methods.

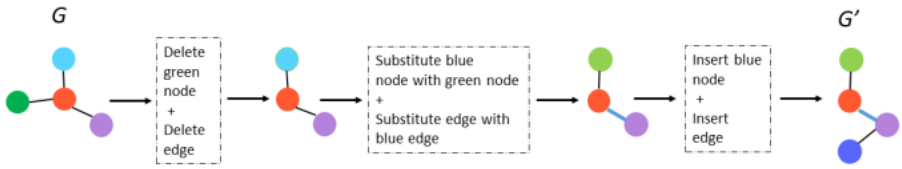

Figure 3.2: Transformation of graph G into graph G' through edit operations.

1) C^v_S, C^v_D, C^v_I, C^e_S, C^e_D, and C^e_I as functions.

The methods in this first class define the six node and edge edit costs C^v_S, C^v_D, C^v_I, C^e_S, C^e_D, and C^e_I in Equation 2.1 and Equation 2.2 as functions that depend on the attributes on the nodes and edges. These functions are learned using, for instance, a neural network or a probability density distribution.

- **Neuhaus and Bunke (2005)**: The method feeds a self-organised map with the attributes of the nodes or the edges and at the output obtains the substitution, deletion, and insertion costs on nodes and edges. The optimisation function used in the learning process is the average of eight optimisation functions: Davies-Bouldin, Dunn, Goodman–Kruskal, Calinski-Haraba, Rand index, Jaccard, Fowlkes–Mallo.

- **Neuhaus and Bunke (2007)**: This method is similar to the previous method. Nevertheless, it uses the Dunn index as the optimisation function. Moreover, given the attributes on the nodes or the edges, it computes the costs as the inverse of the probability set by a probability density function.

- **Caetano et al. (2009)**: In this case, the learning set has a different structure since it is composed by pairs of attributed graphs and the node-to-node mapping between them, instead of classified attributed graphs. Thus, the node-to-node mappings in the learning set become the ground truth mappings and the optimisation function learns the edit costs such that the resulting node-to-node mappings tend to be close to the node-to-node mappings in the learning set. This optimisation function has been called the correspondence accuracy.

- **Leordeanu et al. (2012)**: The method learns weights in the same way as Caetano et al. (2009) but the optimisation function is the maximisation of the recognition ratio of the training set composed of classified graphs.

2) C_D^v, C_I^v, C_D^e, C_I^e as constants.

In this class, the four node and edge edit costs C_D^v, C_I^v, C_D^e, C_I^e are constants, meaning they do not depend on the attributes.

- **Cortés et al. (2019, 2018)**: The method learns substitution functions on nodes and edges through a neural network taking the set of attributes on the nodes and edges as input. Insertion and deletion of nodes and edges are assumed to be constants.

- **Santacruz and Serratosa (2019)**: Similar to Cortés et al. (2019), but it also learns insertion and deletion costs for nodes and edges through a neural network.

3) C_S^v and C_S^e defined as weighted Euclidean distances.

Finally, in this third class, the node and edge substitution edit costs C_S^v and C_S^e are defined as:

$$C_S^v(i,a) = \sum_{t=1}^{N} w_t^v |\gamma_i^t - \gamma_a^t|, \quad C_S^e(i,j,a,b) = \sum_{t=1}^{M} w_t^e |\beta_{i,j}^t - \beta_{a,b}^t| \tag{2.6}$$

Where w^v and w^e are weights associated with node and edge attributes, respectively.

- **Algabli and Serratosa (2018a)**: This method learns weights of the weighted Euclidean distance to define substitution costs on nodes and edges, while insertion and deletion costs are treated as constants.

3.3.4 Applications and Relevance

GED has widespread applications in molecular comparison, handwriting recognition, and graph-based pattern analysis. By combining GED with learning-based optimization, it is possible to improve scalability and domain-specific performance.

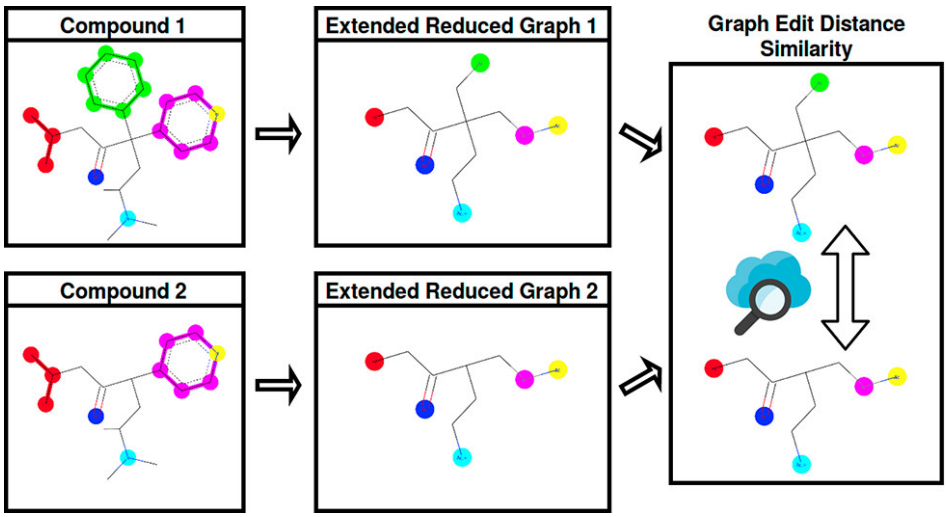

Figure 3.3: Example application of molecular matching.

3.4 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric algorithm commonly used for classification and regression tasks. It operates on the principle that similar data points are likely to have similar outcomes.

3.4.1 k-Nearest Neighbors Algorithm

In statistics, the *k-nearest neighbors algorithm (k-NN)* is a non-parametric supervised learning method. It was first developed by Evelyn Fix and Joseph Hodges in 1951 [**?**], and later expanded by Thomas Cover [**?**]. Most often, it is used for classification, as a *k-NN classifier*, the output of which is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If $k = 1$, then the object is simply assigned to the class of that single nearest neighbor.

The *k-NN algorithm* can also be generalized for regression. In *k-NN regression*, also known as *nearest neighbor smoothing*, the output is the property value for the object. This value is the average of the values of k nearest neighbors. If $k = 1$, then the output is simply assigned to the value of that single nearest neighbor, also known as *nearest neighbor interpolation*.

For both classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that nearer neighbors contribute more to the average than distant ones. For example, a common weighting scheme consists of giving each neighbor a weight of $1/d$, where d is the distance to the neighbor.

The input consists of the k closest training examples in a dataset. The neighbors are taken from a set of objects for which the class (for *k-NN classification*) or the object property value (for *k-NN regression*) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity (sometimes even a disadvantage) of the *k-NN algorithm* is its sensitivity to the local structure of the data. In *k-NN classification*, the function is only approximated locally, and all computation is deferred until function evaluation. Since this algorithm relies on distance, if the features represent different physical units or come

in vastly different scales, then feature-wise normalizing of the training data can greatly improve its accuracy [**?**].

3.4.2 Statistical Setting

Suppose we have pairs $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ taking values in $\mathbb{R}^d \times \{1, 2\}$, where Y is the class label of X, so that $X \mid Y = r \sim P_r$ for $r = 1, 2$ (and probability distributions P_r). Given some norm $\|\cdot\|$ on $\mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, let $(X_{(1)}, Y_{(1)}), \ldots, (X_{(n)}, Y_{(n)})$ be a reordering of the training data such that

$$\|X_{(1)} - x\| \leq \cdots \leq \|X_{(n)} - x\|.$$

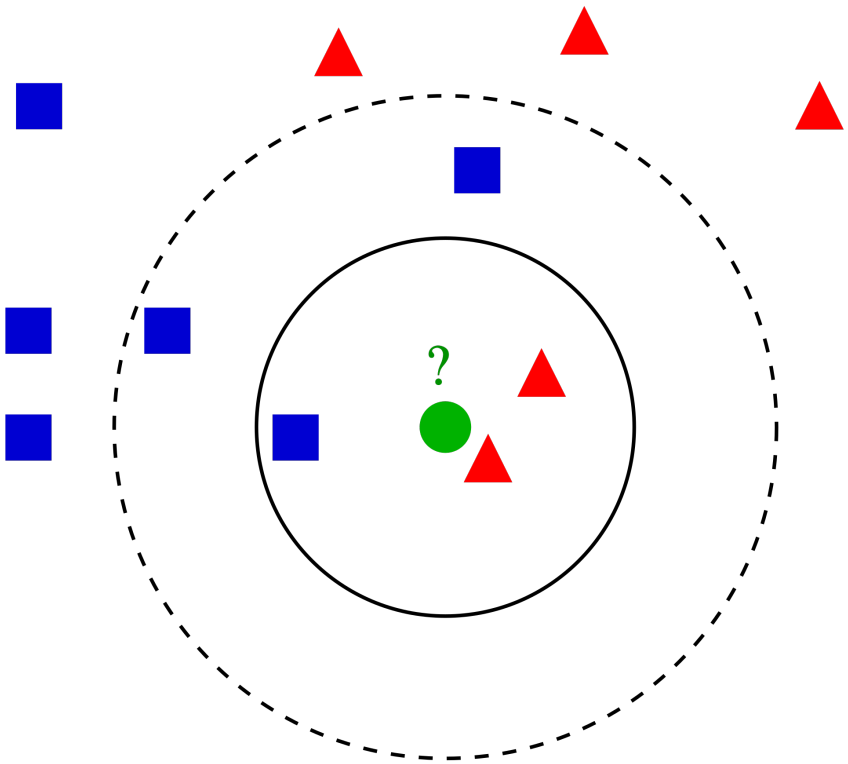

Figure 3.4: Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If $k = 3$ (solid line circle), it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If $k = 5$ (dashed line circle), it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

In the context of binding affinity prediction, KNN estimates the binding affinity of a query compound by analyzing the affinities of its k most similar compounds in the dataset.

The similarity between compounds can be measured using metrics such as Euclidean distance or GED, depending on the representation of molecular structures. The algorithm's interpretability is a key advantage, as predictions are directly based on the properties of neighboring compounds. This makes KNN a complementary approach to GED-based structural comparisons, providing a robust method for predicting binding affinities.

3.5 Distance Metrics Used in KNN Algorithm

As we know, the KNN algorithm helps us identify the nearest points or groups for a query point. To determine the closest groups or nearest points for a query point, we use the following distance metrics:

3.5.1 Euclidean Distance

This is the Cartesian distance between two points in a plane or hyperplane. The Euclidean distance can also be visualized as the length of the straight line that joins the two points under consideration. This metric helps calculate the net displacement between two states of an object.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2}$$

3.5.2 Manhattan Distance

The Manhattan distance metric is generally used when we are interested in the total distance traveled by an object instead of its displacement. This metric is calculated by summing the absolute difference between the coordinates of the points in n-dimensions.

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

3.5.3 Minkowski Distance

The Euclidean and Manhattan distances are special cases of the Minkowski distance, which is defined as:

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

When $p = 2$, the formula corresponds to the Euclidean distance, and when $p = 1$, it corresponds to the Manhattan distance.

Other distance metrics, such as Hamming distance, are also used for problems requiring overlapping comparisons between two vectors whose contents can be Boolean or string values.

3.5.4 Workings of KNN Algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity, where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.

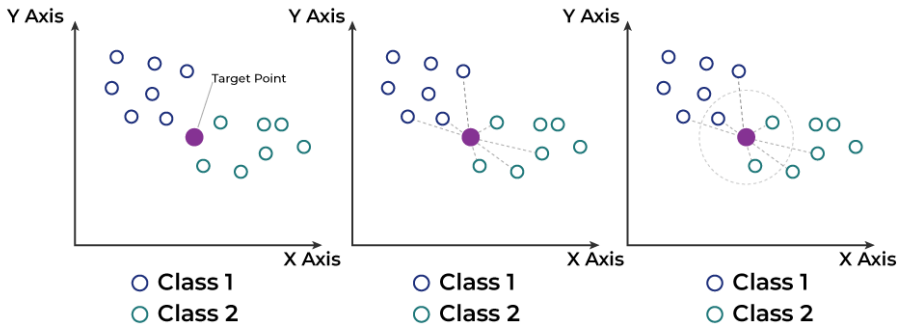

Figure 3.5: Visual representation of a KNN algorithm.

3.5.5 Step-by-Step Explanation

1. **Step 1: Selecting the Optimal Value of K** K represents the number of nearest neighbors that need to be considered while making a prediction.

2. **Step 2: Calculating Distance** To measure the similarity between the target and training data points, Euclidean distance is used. Distance is calculated between each data point in the dataset and the target point.

3. **Step 3: Finding Nearest Neighbors** The K data points with the smallest distances to the target point are the nearest neighbors.

4. **Step 4: Voting for Classification or Taking Average for Regression** In classification problems, the class labels of K-nearest neighbors are determined by majority voting. The class with the most occurrences among the neighbors becomes the predicted class for the target data point. In regression problems, the output is calculated by taking the average of the target values of K-nearest neighbors. The calculated average becomes the predicted output for the target data point.

Let X be the training dataset with n data points, where each data point is represented by a d-dimensional feature vector X_i, and Y is the corresponding label or value for each data point in X. Given a new data point x, the algorithm calculates the distance between x and each data point X_i in X using a distance metric, such as Euclidean distance:

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d} (x_j - X_{ij})^2}$$

The algorithm selects the K data points from X that have the shortest distances to x. For classification tasks, the algorithm assigns the label y that is most frequent among the K-nearest neighbors to x. For regression tasks, the algorithm calculates the average or weighted average of the values y of the K-nearest neighbors and assigns it as the predicted value for x.

3.5.6 Advantages of the KNN Algorithm

- **Easy to Implement:** The algorithm's complexity is low and straightforward to implement.

- **Adaptability:** The KNN algorithm adapts easily to new data. Since it stores all the data in memory, adding new data points adjusts the algorithm's predictions automatically.
- **Few Hyperparameters:** Only two parameters are required: the value of K and the choice of the distance metric.

3.5.7 Disadvantages of the KNN Algorithm

- **Does Not Scale:** KNN is a lazy algorithm, requiring significant computational power and memory as it stores all data points. This makes it resource-intensive for large datasets.
- **Curse of Dimensionality:** The algorithm struggles with high-dimensional data, leading to poor classification accuracy.
- **Prone to Overfitting:** KNN is affected by the curse of dimensionality, making it prone to overfitting. Dimensionality reduction techniques are often required to address this issue.

3.6 Molecular Descriptors

Molecular descriptors are quantitative representations of molecular properties that capture various chemical and physical attributes. They play a crucial role in machine learning models for predicting protein-ligand binding affinities. Descriptors can be categorized into the following types:

- **Constitutional Descriptors**: Represent basic molecular composition, such as molecular weight and atom counts.
- **Topological Descriptors**: Capture the connectivity and graph-theoretical properties of the molecule.
- **Geometrical Descriptors**: Describe the three-dimensional spatial arrangement of atoms.
- **Electronic Descriptors**: Reflect electronic properties, such as charge distribution and polarizability.

Tools like Mordred facilitate the computation of a wide range of molecular descriptors, offering a rich feature set for machine learning models. These descriptors complement structural comparison methods like GED, enhancing both the predictive accuracy and interpretability of binding affinity models.

3.7 Summary

The integration of machine learning techniques, GED, KNN, and molecular descriptors provides a comprehensive approach to predicting protein-ligand binding affinity. This multifaceted strategy enables the development of models that are not only accurate but also interpretable, thereby advancing the field of computational drug discovery.

3.8 Related Work

Several studies have integrated GED with machine learning techniques for molecular similarity assessments and binding affinity predictions. Serratosa's work on Fast Bipartite Matching exemplifies how computational improvements can address scalability challenges in GED. Additionally, the integration of molecular descriptors, such as those provided by Mordred, has enabled models to achieve higher predictive accuracy by capturing diverse chemical properties.

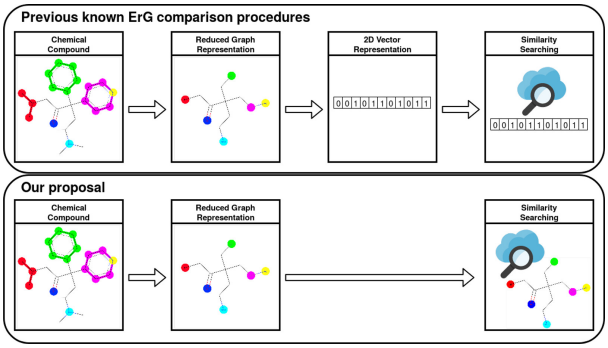

Figure 3.6: Molecular comparison flowcharts. Difference between traditional ErG methods and other proposals.

Figure 3.7: Workflow integrating GED, KNN, and molecular descriptors for binding affinity prediction.

Conclusion: The combination of GED, KNN, and molecular descriptors represents a robust framework for interpretable and efficient prediction of protein-ligand binding affinity. This approach bridges the gap between structural comparisons and feature-based machine learning, advancing computational drug discovery.

Chapter 4

Methods

4.1 Overview of the Proposed Method

The proposed method integrates:

1. A visualization editor for structural comparisons.
2. GED for quantifying molecular differences.
3. KNN for binding affinity prediction.

4.2 Ligands and Proteins in Structural Analysis - And Why Ligands?

In biochemistry and pharmacology, a ligand is a substance that forms a complex with a biomolecule to serve a biological purpose. The etymology stems from Latin *ligare*, which means 'to bind'. In protein-ligand binding, the ligand is usually a molecule which produces a signal by binding to a site on a target protein.

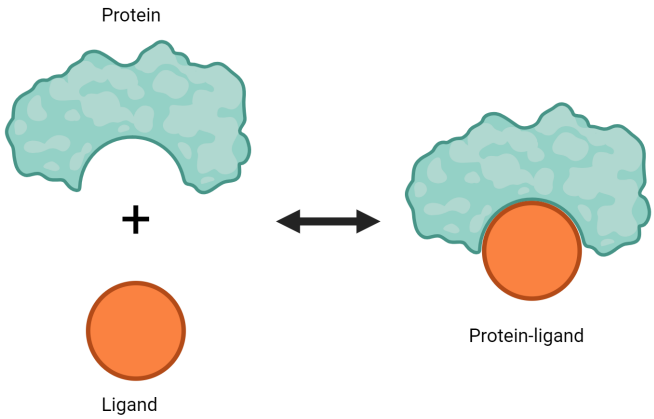

Figure 4.1: Protein-ligand Binding

Ligands are relatively small molecules, typically comprising 20 to 30 nodes. In contrast, proteins are significantly larger, with structures often containing thousands of nodes (e.g., 3000, 4000, or even 5000 nodes). Due to their smaller size, ligands are commonly used in structural analyses and computational projects, including our thesis.

In our thesis project, we focused exclusively on ligands due to their manageable size and computational feasibility. However, there is a new initiative aimed at including proteins in structural analyses. This shift presents challenges, primarily related to storage and computational requirements. Traditionally, proteins are represented as sequences rather than as graphs due to the substantial space needed for graph-based representations.

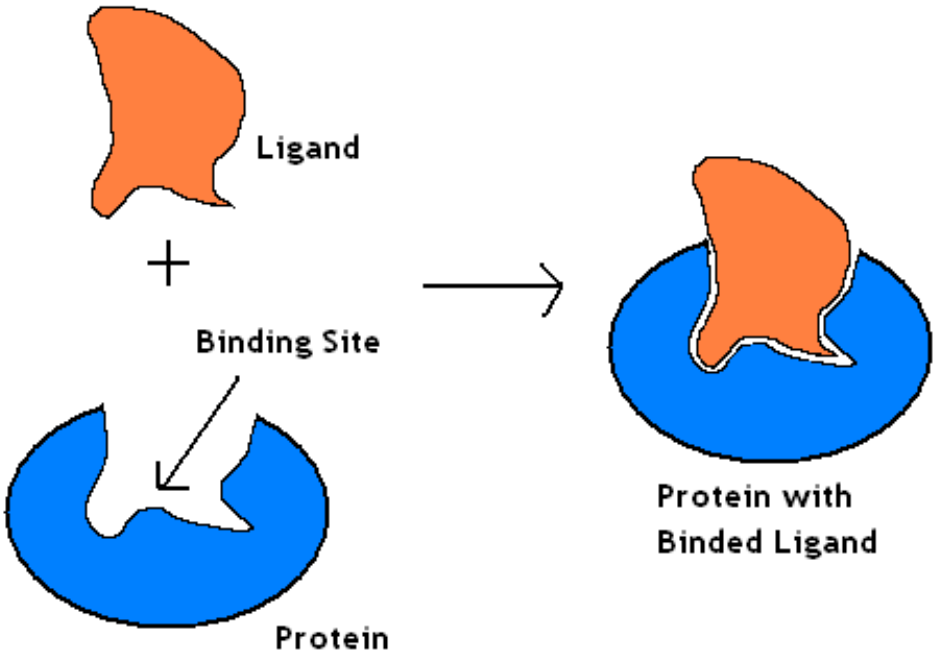

Figure 4.2: Protein-ligand Binding site

4.3 IC50 Determination

The IC50, or the half-maximal inhibitory concentration, represents a measure used in pharmacology to evaluate the potency of a compound in inhibiting a specific biological or biochemical function. The graphical illustration of IC50 determination can be found in Figure 4.3, and more details are available on the Wikipedia page for IC50.

4.4 SARS-CoV-2 M-pro Database

4.4.1 Dataset Composition

The learning dataset used in this project consists of 223 SARS-CoV-2 main protease (M-pro) crystallized structures, each bound to an inhibitor with a known pIC_{50}. This dataset includes:

- **160 compounds in the learning set:**
  - 53 compounds sourced from the well-known Protein Data Bank (PDB) database (RCSB PDB).
  - 107 compounds obtained from the FRAGALYSIS database (FRAGALYSIS).
- **63 compounds in the test set.**

While the PDB and FRAGALYSIS databases contain numerous structures, only a subset of inhibitors has experimentally determined pIC_{50} values obtained from a biotechnological lab. This dataset was previously employed in drug discovery research, as documented in Fadlallah et al., 2023 [127].

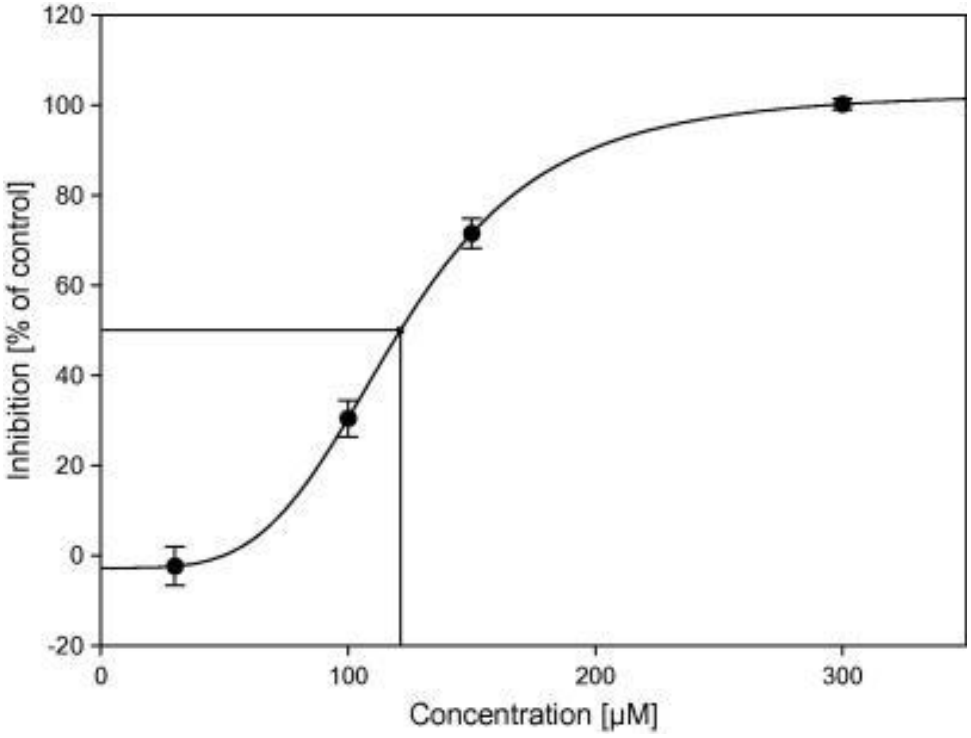

Figure 4.3: Graphical illustration of IC50 determination.

4.4.2 Graph Representation of Drug-Protein Pairs

For each drug-protein pair, a singular attributed graph is constructed. This graph represents the complete drug molecule along with the atoms and bonds of the protein within proximity to any drug atom forming a bond. The key features of the graph are as follows:

- **Nodes:** Represent atoms from both the drug and the protein.
  - Attributes include:
    - 3D positions of the atoms.
    - Atomic numbers.
    - Number of bonds.
    - Electric charges.
- **Edges:** Represent bonds between atoms.
  - No additional attributes are assigned to the edges.

To accommodate the maximum observed size of a compound (drug + binding site atoms), all graphs are padded to include 146 nodes. The pIC_{50} value is treated as a global property in this context.

Making them more practical for testing, our initial tests focused exclusively on ligands.

4.5 Graph Edit Distance Computation

The most well-known model is the Graph Edit Distance (GED). Node substitution, insertion, and deletion costs are parameterized for flexibility. Edge substitution costs account for bond type mismatches.

Figure 4.4 shows a representation of edit operations: delete edge, delete node, insert node, insert edge, and substitute node.

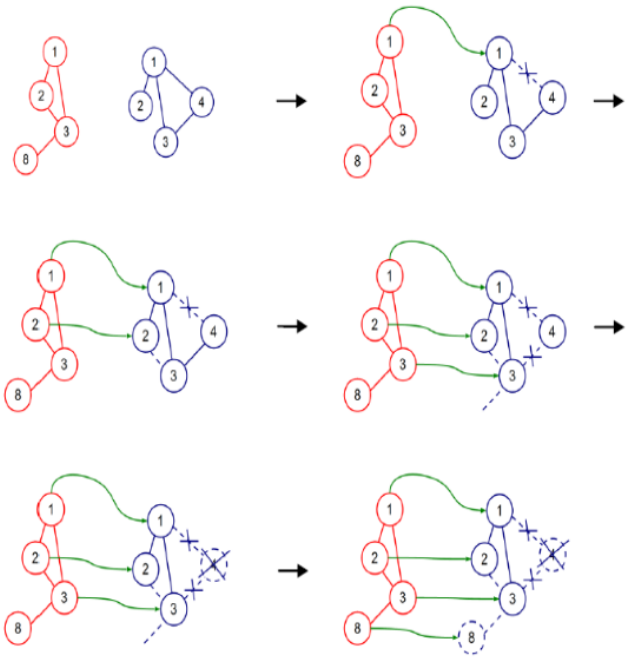

Figure 4.4: An example of an edit path between two graphs. Green arrows represent substitutions. Studded lines indicate deletions. Dotted lines indicate insertions.

The basic idea behind the Graph Edit Distance (GED) is to define a dissimilarity measure between two graphs, which is defined as the minimum amount of required distortion to transform one graph into the other. To this end, a number of distortion or edit operations are defined, consisting of insertion, deletion, and substitution of both nodes and edges.

For every pair of graphs G_p and G_q, there is a sequence of edit operations, or an edit path:

$$\text{editPath}(G_p, G_q) = (\sigma_1, \ldots, \sigma_k),$$

where each σ_i denotes an edit operation that transforms one graph into the other. In general, several edit paths may exist between two given graphs. Figure 4.5 shows an example of an edit path that transforms G_p into G_q. It is composed of the following five edit operations: delete edge, delete node, insert node, insert edge, and substitute node.

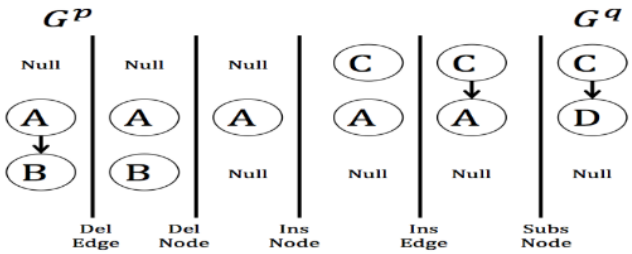

Figure 4.5: One of the edit paths that transforms G_p into G_q.

To quantitatively evaluate which edit path is the best, edit cost functions are introduced. The basic idea is to assign a penalty or cost to each edit operation according to the amount of distortion that it introduces in the transformation.

4.5.1 Graph Edit Distance (GED)

GED is defined as the minimum amount of required distortion to transform one graph into another. To this end, a number of distortion or edit functions consisting of deletion, insertion, and substitution of nodes and edges are defined. To quantitatively evaluate the graph transformation, an edit cost is assigned to each edit operation according to the amount of distortion that it introduces in the transformation.

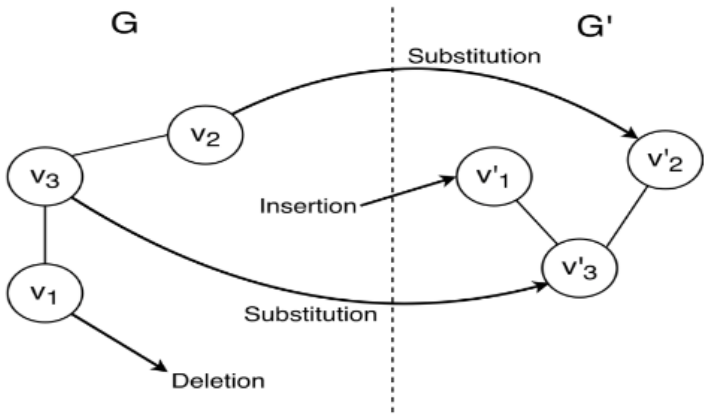

Figure 4.6: Example of two graphs to be compared. The GED between G and G' is the cost of substituting two nodes, deleting a node, inserting a node, substituting an edge, deleting an edge, and inserting an edge.

For instance, in Figure 4.6, GED(G, G') is defined such that G is transformed into G' given the transformations highlighted by the arrows. This is done by substituting v_3 by v'_3 and v_2 by v'_2. Moreover, v_1 is deleted and v'_1 is inserted. The distance is the sum of the costs of these operations:

$$\text{SubsNode}(v_3, v'_3)+\text{SubsNode}(v_2, v'_2)+\text{DelNode}(v_1)+\text{InsNode}(v'_1)+\text{SubsEdge}(v_3, v_2, v'_3, v'_2)+\text{DelEdge}(v_3,$$

Other transformations generate other costs, and thus, the distance becomes the minimum of the costs of all transformations.

The structural and semantic dissimilarity of graphs is only correctly reflected by the GED if the underlying edit costs are defined appropriately and depending on the application at hand.

As we have seen in the Serratosa paper. GED is calculated using a bipartite matching approach.

$$\text{GED} = \text{Node Cost} + \text{Edge Cost}$$

4.5.2 Solving the Graph Edit Distance by Bipartite Algorithm

Bipartite algorithm (BP) implemented by Serratosa is one of the newest methods presented to solve the GED and new optimization techniques of this algorithm have recently appeared. Experimental validation shows that, currently, it is one of the best sub-optimal algorithms since it obtains a good approximation of the distance value in cubic computational cost. BP algorithm is composed of three main steps. The first step defines a cost matrix, the second step applies a linear solver such as the Hungarian method or the Jonker-Volgenant method to this matrix and deduces the matching f^*. The third step computes the edit cost given this matching between both graphs $\text{EditCost}(G^*, G', f^*)$. Figure 4.7 shows the cost matrix of the BP algorithm in which n and m are the graph orders.

Quadrant Q_1 denotes the combination of substituting stars of both graphs. That is, $C_{ij} = \text{CEdit}_{v_s}(N_v^i, N_{v'}^j)$. While computing C_{ij}, we can also use the BP algorithm but restricted to only mapping the neighbour nodes of the stars. The diagonal of quadrant Q_2 denotes the whole costs C_{ii} of deleting stars N_v^i. In this case, the total cost is the cost of deleting the central node and the neighbour nodes of the star plus the cost of deleting the edges that connect the central node to the neighbour nodes. Similarly, the diagonal of quadrant Q_3 denotes the whole costs C_{jj} of inserting stars $N_{v'}^j$. Quadrant Q_4 is filled with zeros since the substitution between null stars has a zero cost.

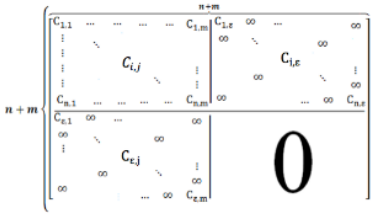

Figure 4.7: Cost matrix of the BP algorithm.

Equation 5 summarizes the costs of substituting, deleting, and inserting stars (these expressions are used in the next section). The star substitution cost is composed of three terms: substitution of the !1 central nodes: $C_{vs}(v^i, v^j)$, deleting or inserting the external nodes: $T_{ij} : (K_v + K_e)$, and substitution of the external nodes: $C_{vs}(v_{t'}^e, v_t^e)$. The value $T_{ij} = |n_v^i - n_v^j|$ is the difference between the order of stars. Function $C_{vs}(v_{t'}^e, v_t^e)$ is computed through a linear solver that obtains the best mapping f between external nodes of the star. Therefore,

$$C_{vs}(v^i, v^j) = \sum_{t=1}^{M_{ij}} (C_{vs}(v_{t'}^e, v_t^e) + C_{vs}(e_{t'}^e, e_t^e)),$$

where M_{ij} is the number of substituted external nodes, $M_{ij} = \min\{n_v^i, n_v^j\}$. Node $v_{t'}^e$ represents the t-th external node of the star. We assume M_{ij} external nodes and edges are deleted (or inserted) while computing the substitution of two stars. It was demonstrated in [**?**] that this assumption always holds if the costs are defined such that the GED is really a distance function in a Euclidean space. In other words, if

$$2 \cdot K_v + 2 \cdot K_e \geq C_{vs}(v_{t'}^e, v_t^e) + C_{vs}(e_{t'}^e, e_t^e),$$

then:

$$\begin{aligned} \text{Substituting two stars: } & C_{ij} = C_{vs}(v^i, v^j) + T_{ij} \cdot (K_v + K_e) + C_{vs}(v_{t'}^e, v_t^e), \\ \text{Deleting a star: } & C_{ii} = K_v + n_v^i \cdot (K_v + K_e), \\ \text{Inserting a star: } & C_{jj} = K_v + n_v^j \cdot (K_v + K_e). \end{aligned} \tag{5}$$

The Hungarian [**?**] or Jonker-Volgenant [**?**] algorithms used in its original form are optimal for solving the linear sum assignment problem but they are suboptimal for solving the GED in the second step of the BP algorithm. This is due to the fact that the stars are considered individually. For this reason, BP can overestimate the GED. The computational cost of the method is $O((n+m)^3)$. The learning method presented in [**?**] is inspired in this algorithm.

4.6 Mordred Descriptor Extraction

Descriptors are computed using Mordred for all compounds in the dataset. Missing values are handled by imputation, and descriptors are normalized for compatibility with machine learning models.

4.6.1 Mordred: Molecular Descriptor Calculator

Mordred was designed to be a software program that is easy to install and use, supports abundant molecular descriptors, has a high calculation speed, and includes automated tests. Molecular-descriptor calculation programs usually have many dependent software programs that must be manually installed. All direct dependent libraries in Mordred, except for RDKit and NumPy, are coded in pure Python (enum34, networkx, six, tqdm) to simplify the installation. RDKit and NumPy are widely used Python libraries and can be easily installed via the pre-compiled libraries distributed by the Anaconda cloud. Therefore, users can install Mordred using a single command. Because Mordred can be employed as a web service or from a CLI, and with Python 2 and 3 libraries, users ranging from beginners to experts can employ it.

Preprocessing of molecules affects the descriptor values in most molecular-descriptor calculators. However, for each descriptor, Mordred automatically preprocesses molecules (adds or removes hydrogen atoms, performs Kekulization, and detects molecular aromaticity). For example, a topological index descriptor was reported by Balaban et al. along with a numerical example. To reproduce the values, the input molecule should not have explicit hydrogen atoms. Therefore, explicit hydrogen atoms are automatically removed in Mordred. This procedure ensures correctness of preprocessing, which is usually not checked in other software.

Mordred calculates more than 1800 default molecular descriptors, including all those implemented by RDKit (seven modules) and original implementations (42 modules).

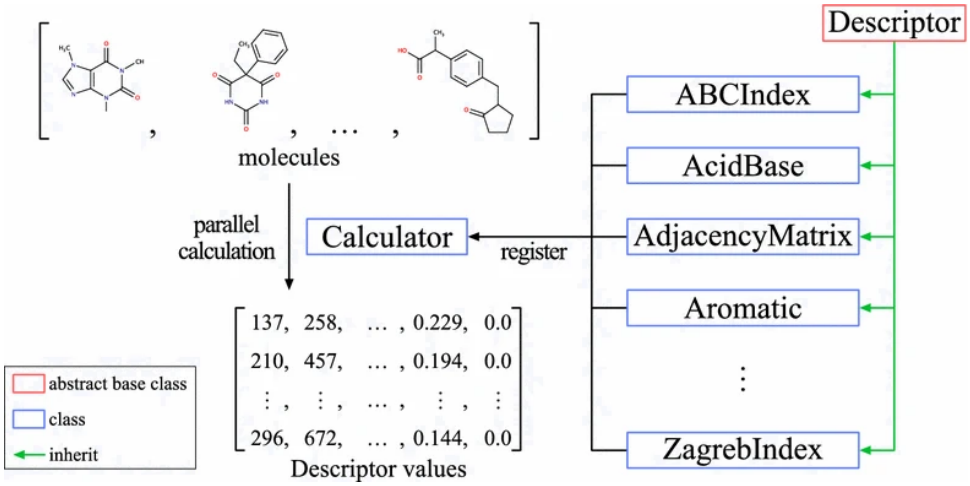

Figure 4.8: Overview of Mordred library. Mordred consists of two main classes: descriptor and calculator. Users can register descriptors on a Calculator instance. A Calculator instance can calculate descriptors in parallel.

4.7 K-Nearest Neighbors Model

The KNN model predicts affinities by combining GED with molecular descriptors as input features. The affinity of a query compound is calculated as:

$$\hat{y} = \frac{1}{k} \sum_{i=1}^{k} y_i$$

where y_i represents the affinity of the i-th nearest neighbor.

4.8 Graphical Editor

A Python-based GUI allows users to:

1. Visualize molecules and structural differences.
2. Compare compounds using GED.
3. Predict affinities using KNN.

Chapter 5

Implementation Summary

5.1 Part 1: Real-Time Ligand Comparison

This part focuses on comparing two ligand structures in real time using their `.sdf` files. Key features include:

5.1.1 Interactive GUI Editor

- Enables users to select two `.sdf` files dynamically from the system.
- Provides options to customize GED costs:
  - Node substitution cost.
  - Node deletion cost.
  - Node insertion cost.
  - Edge substitution cost.
  - Edge deletion cost.
  - Edge insertion cost.

5.1.2 Visualization

- Displays the molecular structure of both ligands and their differences.
- Annotates differences such as deleted nodes and edges, substitution costs, and inserted nodes or edges.
- Includes compound names for clarity.

5.1.3 Real-Time GED Calculation

- Incorporates real-time GED calculations under various structural modification scenarios:
  - Edge Deletion.
  - Node Substitution.
  - Node Insertion.

  - Node Deletion.
  - Edge Insertion.

5.1.4 Case-Specific GED Calculations

- Handles structural modifications (e.g., deletion of a node with its associated bonds) and calculates GED appropriately.
- Outputs user-friendly descriptions of structural differences, such as the number of bonds deleted with a node.

5.2 Part 2: Database-Wide GED and Affinity Analysis

This part involves comparing a selected ligand against an entire database of ligands:

5.2.1 Editor Features

- Allows users to:
  - Select a single `.sdf` file for comparison.
  - Select a folder containing multiple `.sdf` files along with an `affinity.txt` file.

5.2.2 GED and Affinity Analysis

- Computes GED between the selected ligand and all ligands in the folder.
- Identifies and visualizes the ligand with the minimum GED, along with its affinity value.

5.2.3 KNN Analysis

- Divides the dataset into 75% training and 25% testing.
- Predicts KNN values for the test set and computes errors (square of the difference between predicted and actual affinity).
- Visualizes test errors and compares them through bar plots.
- Calculates Mean Square Error (MSE) for the predictions.

5.3 Part 3: Statistical Analysis

This part focuses on analyzing GED and affinity values for two datasets (`test` and `learn`):

5.3.1 Input

- Two folders: `test` and `learn`, each containing `.sdf` files and `affinity.txt`.

5.3.2 GED Distance Matrix

- Computes GED distances between all compounds in the `test` and `learn` folders.

5.3.3 Affinity Difference Matrix

- Computes absolute differences in affinity values.

5.3.4 Scatter Plot Visualization

- Plots GED against absolute affinity differences.
- Enhances scatter plots to trend toward a line (indicating improved predictive performance with increased training data).

5.3.5 Experimentation with Substitution Costs

- Analyzes the impact of varying substitution costs (e.g., setting substitution cost to 0.5).

5.4 Extended Analyses and Neural Network Integration

Descriptor-Based Analysis

- Incorporates Mordred descriptor vectors for comparison.
- Uses Euclidean distances for affinity prediction and comparison.

5.4.1 Neural Network Model (Optional)

- Uses Mordred descriptors as input vectors.
- Trains a neural network to predict binding affinity.
- Analyzes prediction accuracy using test and training sets (75% train, 25% test).

5.4.2 Comparison Metrics

- GED vs. Affinity Scatter Plot.
- Mordred vs. Affinity Scatter Plot.
- GED vs. KNN MSE.
- Mordred vs. KNN MSE.

Chapter 6

Database, The Code used and Its implementaion

6.1 Database and Data Preprocessing

The dataset consists of protein-ligand pairs in '.sdf' format and affinity scores in a text file. Molecules are converted to graphs using RDKit and NetworkX.

6.1.1 Dataset Overview

The dataset used in this project consists of structural and affinity data for a collection of protein-ligand pairs, specifically focused on SARS-CoV-2 main protease (M-pro). The dataset includes:

6.1.2 Protein Structures

- Crystallized structures of M-pro in `.pdb` format, representing the 3D geometry of the proteins.
- These structures are obtained from widely used databases like the Protein Data Bank (PDB).

6.1.3 Ligand Structures

- Chemical compounds, represented in `.sdf` format, which act as potential inhibitors for the main protease.
- These ligands include experimentally validated compounds with known binding affinities.

6.1.4 Affinity Data

- Binding affinities are provided in the `Affinity.txt` file, where each entry specifies the protein-ligand pair and its corresponding (binding affinity) value.
- These affinities represent experimentally determined measures of interaction strength, sourced from biotechnological studies.

6.1.5 Dataset Breakdown

Learning Set

- Includes a primary set of protein-ligand pairs used for training predictive models.
- Sourced from the PDB and other databases like FRAGALYSIS, focusing on inhibitors with experimentally validated binding affinities.

Test Set

- Comprises a subset of protein-ligand pairs designated for model evaluation.
- Ensures an independent evaluation of predictive performance.

The dataset represents a targeted selection of inhibitors for SARS-CoV-2 main protease, previously used in drug discovery research. Each protein-ligand pair has been carefully curated to ensure data quality, with affinity values providing a reliable benchmark for machine learning models.

6.2 Graphical User Interface for Real-Time Molecular Comparison

The Python program is a Graphical User Interface (GUI) tool for real-time comparison of two molecular structures (ligands) stored in `.sdf` format. It calculates the Graph Edit Distance (GED) between the two molecules and visualizes the differences in their structures.

6.2.1 Dependencies and Libraries

- **Tkinter:** Provides the GUI framework.
- **RDKit:** A cheminformatics toolkit for working with molecular data.
- **NetworkX:** Used to represent molecules as graphs for GED calculation.
- **SciPy:** For solving optimization problems like node and edge matching in GED.
- **Pillow:** For handling image data and visualizing molecules.

6.2.2 Molecule Representation as Graphs

Each molecule is converted into a graph:

- **Nodes:** Represent atoms with attributes such as element symbol and atomic number.
- **Edges:** Represent bonds with attributes such as bond type.

6.2.3 Graphical User Interface (GUI) Components

- **File Selection:** Allows the user to select two `.sdf` files using file dialogs.
- **Cost Configuration:** Entry fields for user-defined GED costs.
- **Comparison Button:** Triggers molecule comparison, GED calculation, and visualization.
- **Visualization Panel:** Displays both molecules, highlighting differences with color-coded atoms and bonds.

6.3 Graph Edit Distance Implementation

6.3.1 Graph Edit Distance (GED) Computation

Found in firstpart.py. GED is computed using:

- Node substitution, insertion, and deletion costs.
- Edge substitution, insertion, and deletion costs.

6.3.2 Principal Functions for GED

`molecule_to_attributed_graph(molecule)`

- Converts an RDKit molecule into a NetworkX graph representation.
- **Parameters:** `molecule` (RDKit molecule object).
- Nodes represent atoms with attributes such as element symbol and atomic number.
- Edges represent bonds with attributes such as bond type.

`calculate_ged(graph1, graph2, ...)`

- Computes the Graph Edit Distance (GED) between two molecular graphs.
- **Parameters:**
  - `graph1, graph2`: Molecular graphs.
  - Costs for node substitution, deletion, and insertion.
  - Costs for edge substitution, deletion, and insertion.
- Utilizes bipartite graph matching to compute optimal GED values.

6.4 Mordred Descriptor Integration

This section discusses the integration of Mordred descriptors found in the files `secondpart.py`, `part3_analysis.py`, and `new.py`.

6.4.1 Key Functionality

`calculate_molecular_descriptors(molecule)`

- Computes molecular descriptors using Mordred.
- **Parameters:** `molecule` (RDKit molecule object).
- Extracted descriptors provide detailed information on molecular structure, such as size, weight, polarity, and functional groups.

`normalize_and_calculate_distance(desc1, desc2)`

- Found in `new.py` and `part3_analysis.py`.
- Normalizes descriptor vectors and computes Euclidean distances.
- **Parameters:** Descriptor vectors `desc1`, `desc2`.
- Normalization ensures uniform scale and reduces the impact of large value ranges across different descriptors.

6.4.2 Extended Mordred Usage

Mordred descriptors were integrated to provide a quantitative representation of molecular features:

- Used alongside GED values to create a comprehensive feature set for molecular comparison.
- Enables KNN-based similarity computation for enhanced precision.
- Optimized for real-time calculations within the GUI framework.

6.5 KNN Implementation

The KNN model is implemented using Scikit-learn, trained on a split of 75% learning and 25% testing data. Furthermore, Mordred descriptors and GED values are combined as features for similarity calculations. Found in secondpart.py.

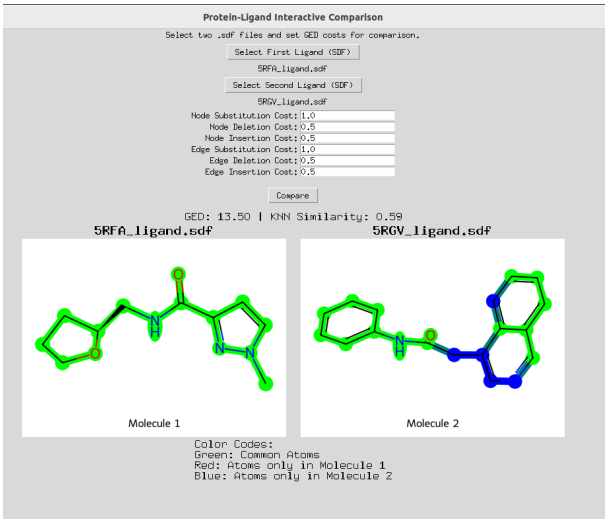

Figure 6.1: KNN GUI built with Tkinter.

6.5.1 Principal Functions

`apply_knn(selected_mol, folder_mols, affinities, selected_graph)`

- Applies K-Nearest Neighbors (KNN) to find the closest molecules in a dataset.
- **Parameters:**
  - `selected_mol`: Molecule being compared.
  - `folder_mols`: Folder containing multiple molecules for comparison.
  - `affinities`: List of affinity scores corresponding to molecules.
- Combines GED and normalized Mordred descriptor distances for similarity metrics.

`compare_with_folder(selected_mol, folder_mols, affinities)`

- Compares a selected molecule with all molecules in a folder and identifies the best match.
- Incorporates both GED and Mordred-based similarity metrics.

6.6 Extended Tools for Nodes and Edges

`node.py` **and** `edge.py` provide interactive tools for modifying molecules:

- **Node Editing:** Functions to insert, substitute, or delete nodes, with GED recalculated after each operation.
- **Edge Editing:** Functions to insert or delete edges, reflecting changes in molecular graphs.

6.7 Analysis Pipeline and Visualization

The analysis pipeline and visualization functionality are implemented in the files `firstpart.py`, `secondpartanalysis.py`, `Mordredanalysis.py`, and `part3_analysis.py`.

6.7.1 Principal Analysis Functions

`analyze_ged_and_affinity(test_folder, learn_folder, test_affinities, learn_affinities)`

- Analyzes Graph Edit Distance (GED) and affinity differences across test and learning datasets.
- Combines structural comparison (GED) with molecular descriptor-based metrics for a robust evaluation.
- **Parameters:**
  - `test_folder`: Path to the folder containing test data.
  - `learn_folder`: Path to the folder containing learning data.
  - `test_affinities`: Affinity values corresponding to test data.
  - `learn_affinities`: Affinity values corresponding to learning data.
- This function integrates molecular descriptors and GED for comparative analysis.

`visualize_scatter(ged_matrix, affinity_diff_matrix)`

- Generates scatter plots to visualize GED vs. affinity differences.
- Highlights correlation patterns between structural and affinity metrics.

`export_results(...)`

- Exports GED and affinity difference matrices for further analysis.
- Provides detailed insights for performance evaluation.

6.7.2 Running the Analysis Pipeline

`main_analysis()` orchestrates the entire process, combining GED, descriptors, and affinities to evaluate molecular similarities and differences.

6.7.3 Protein-Ligand Interactive Comparison Tool

This Python code implements a Protein-Ligand Interactive Comparison tool with a Graphical User Interface (GUI) using Tkinter. It allows users to compare two molecular structures provided in `.sdf` format and visualize their differences. The code calculates the Graph Edit Distance (GED) and a similarity metric based on K-Nearest Neighbors (KNN). Additionally, it visualizes the two molecules with color-coded differences.

6.7.4 Code Explanation

1. Dependencies and Libraries

The tool relies on the following key libraries:

- **RDKit:** Used for molecule handling and visualization.
- **NetworkX:** To represent molecules as attributed graphs.

- **Scipy and Sklearn:** Used for calculations (e.g., graph matching and KNN).
- **Tkinter:** To create a graphical user interface.
- **Pillow (PIL):** For handling images of molecules.

2. Key Functions

- **molecule_to_attributed_graph(molecule):**
  - Converts an RDKit molecule into a graph representation using NetworkX.
  - Nodes represent atoms, with attributes such as symbol and atomic number.
  - Edges represent bonds, with attributes like bond type.
- **calculate_ged(graph1, graph2, ...):**
  - Computes the Graph Edit Distance (GED) between two molecular graphs using bipartite graph matching.
  - GED accounts for the cost of substituting, deleting, or inserting nodes (atoms) and edges (bonds).
- **calculate_knn(mol1, mol2):**
  - Computes a similarity metric based on normalized Euclidean distances of molecular descriptors (random descriptors used as placeholders).
- **Visualization:**
  - **generate_molecule_image:** Draws molecule structures with RDKit and highlights atom differences using colors:
    - **Green:** Common atoms between both molecules.
    - **Red:** Atoms unique to the first molecule.
    - **Blue:** Atoms unique to the second molecule.

6.8 Total Implementation using GUI

The GUI is built with Tkinter, integrating functionalities for ligand selection, visualization, and real time comparison.

GUI Implementation

- **Ligand Selection:**
  - Users select two `.sdf` files for comparison using file dialog buttons.
- **GED Cost Configuration:**
  - Users input costs for node substitution, deletion, insertion, and similar edge costs.
- **Comparison Process:**

- Molecules are loaded from `.sdf` files.
- Their graphs are created, and GED is computed using user-defined costs.
- KNN similarity is calculated based on descriptor vectors.

- **Visualization:**
  - Molecules are displayed with color-coded differences.
  - A legend explains the color codes.

Running the Program

The program launches the GUI (handled by the `MoleculeEditor` class) and displays:

- File selection buttons for two ligands.
- Input fields for GED costs.
- A button to perform the comparison.

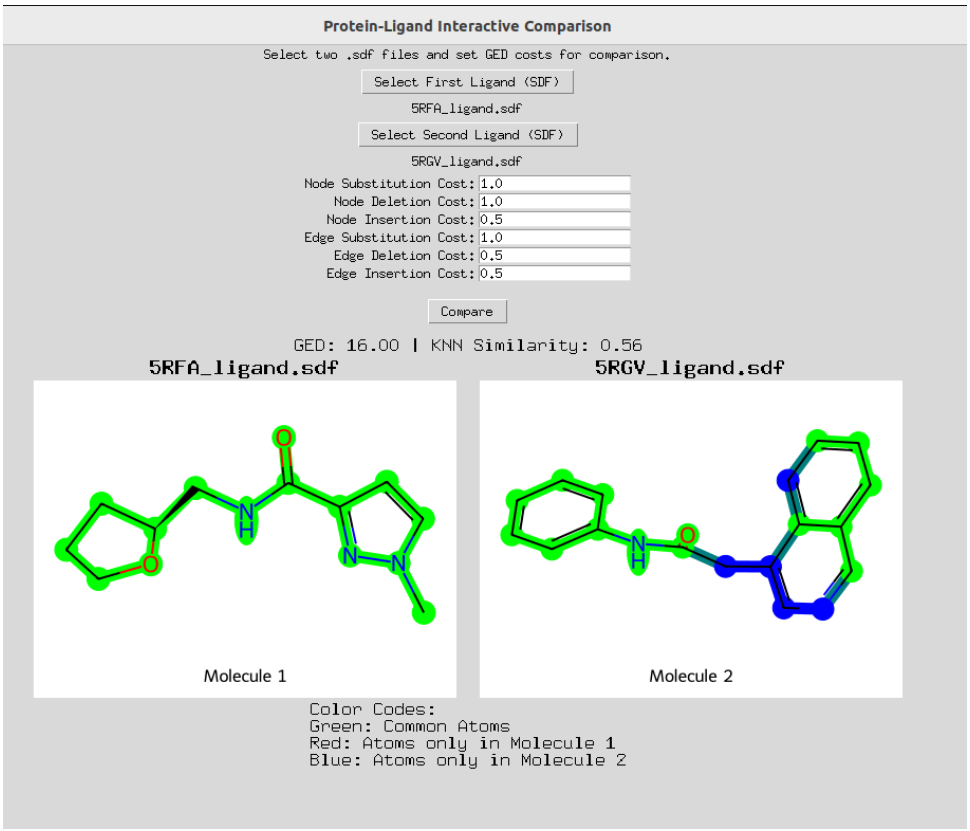

Figure 6.2: GUI for real-time comparison.

Chapter 7

Experimental Results

7.1 For Part 1 - Real-Time Ligand Comparison Framework

7.1.1 Real-Time Ligand Comparison Framework

The proposed framework focuses on the real-time comparison of ligand structures from `.sdf` files. The primary objective is to provide an intuitive and efficient platform for users to perform structural comparisons and analysis without relying on a pre-existing database. The following functionalities and features are included in the framework:

7.1.2 Analysis of Results

The provided data reflects a series of experiments modifying the cost parameters for Graph Edit Distance (GED) calculations between two ligands, `5RFA_ligand.sdf` and `5RGV_ligand.sdf`. The experiments demonstrate how changes to cost parameters influence the computed GED, offering insights into the sensitivity of the comparison process.

7.1.3 Key Observations

1. Initial Configuration (Default Costs)

- **Node Costs:** Substitution = 1.0, Deletion = 0.5, Insertion = 0.5
- **Edge Costs:** Substitution = 1.0, Deletion = 0.5, Insertion = 0.5
- **GED:** 13.5

Impact: This serves as the baseline for comparison. The default costs assign equal weights to substitution operations while penalizing insertions and deletions at half the rate.

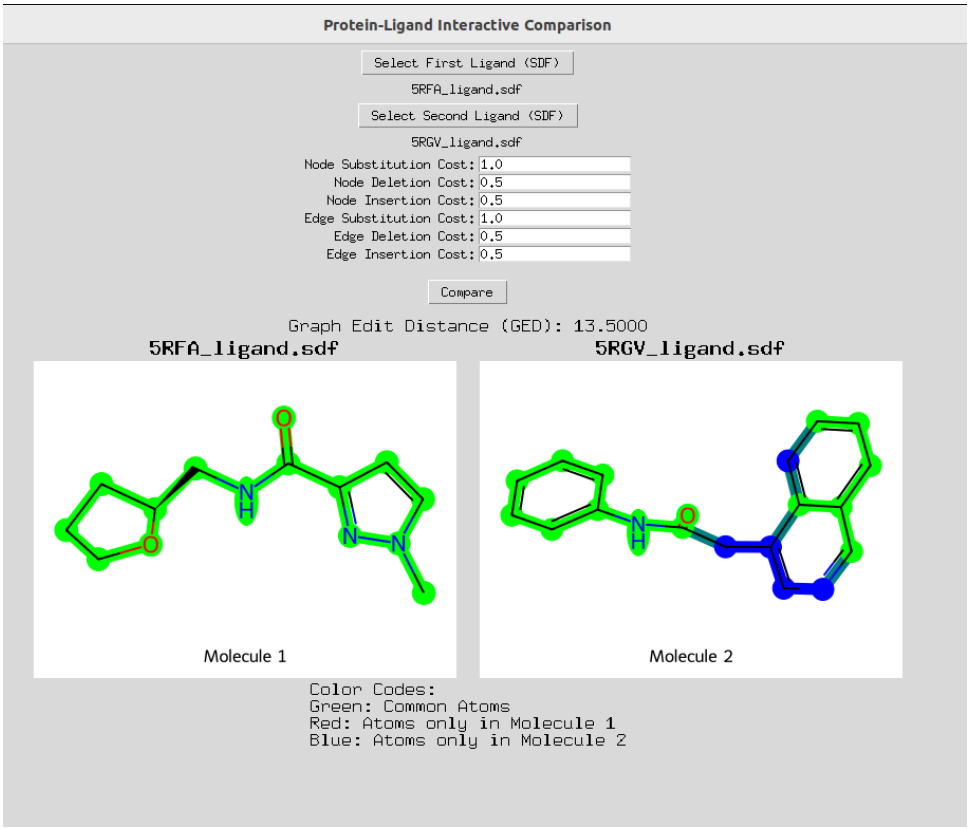

Figure 7.1: Visualization of GED with initial configuration of costs.

2. Effect of Increasing Node Deletion Cost

- **Node Deletion Cost:** Increased from 0.5 $\rightarrow$ 1.0.
- **GED:** Increased to 16.0.

Impact: Increasing the node deletion cost directly raises the overall GED. This indicates that deletions are now more penalized, contributing more to the edit distance.

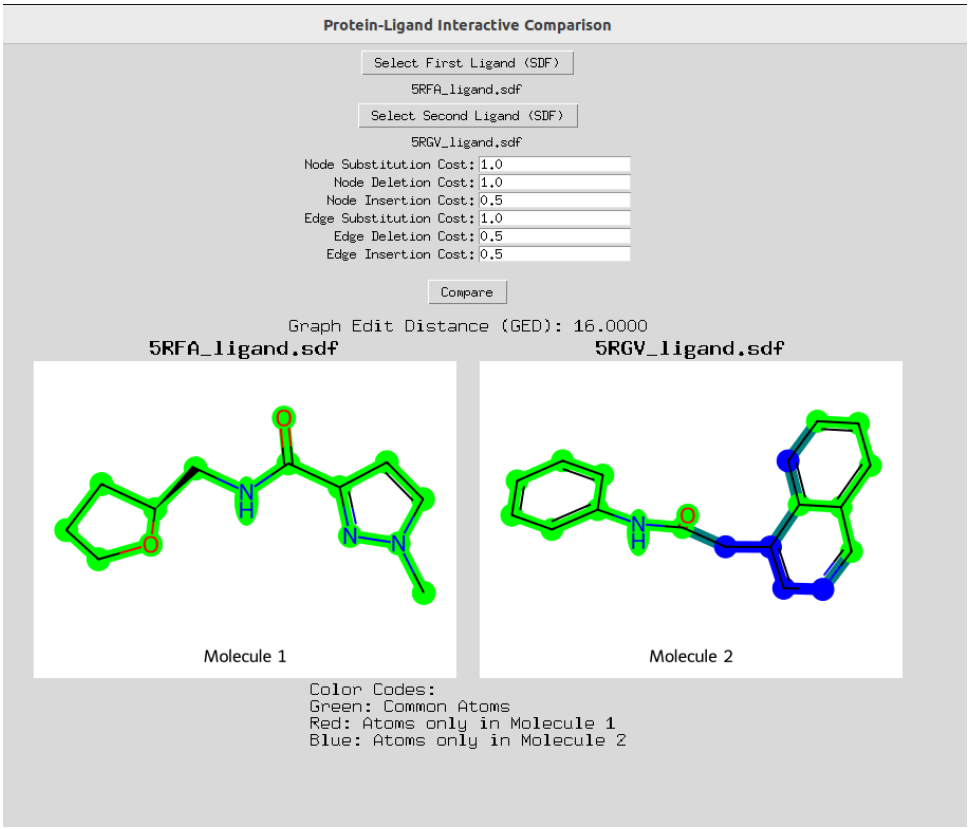

Figure 7.2: Impact of increasing Node Deletion Cost on GED.

3. Effect of Increasing Node Insertion Cost

- **Node Insertion Cost:** Increased from 0.5 $\rightarrow$ 1.0.
- **GED:** No further increase (remains 16.0).

Impact: The unchanged GED implies that insertions and deletions occur in a balanced manner within the graph comparison, with deletions dominating the cost contribution.

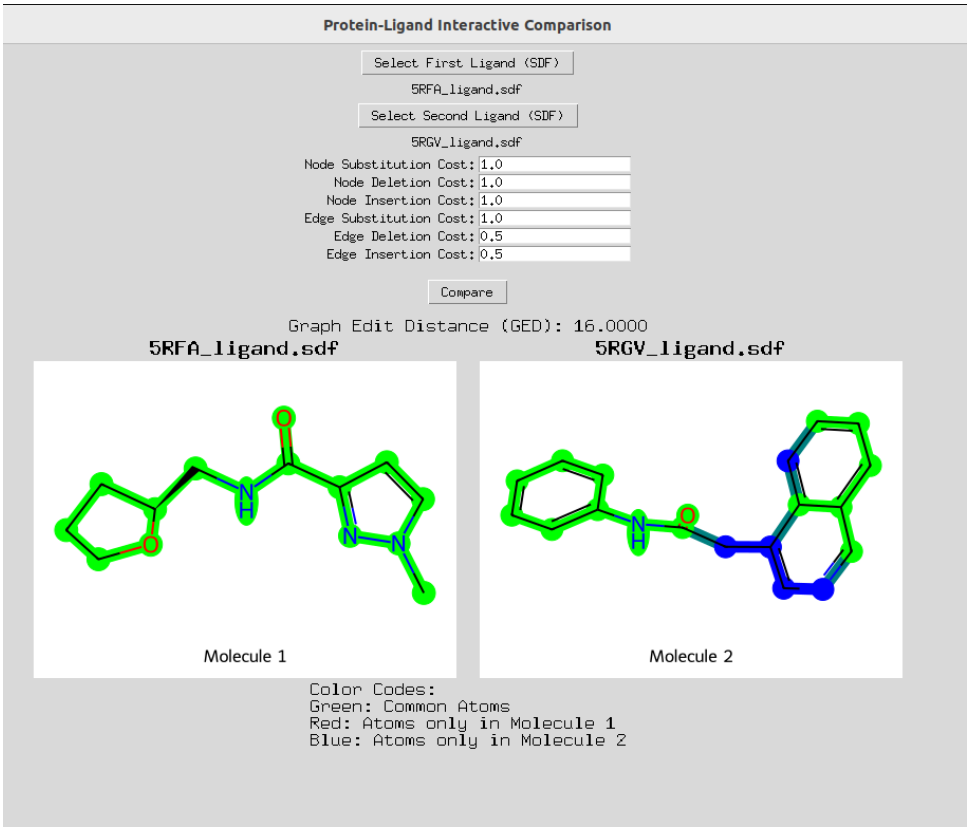

Figure 7.3: Impact of increasing Node Insertion Cost on GED.

4. **Effect of Decreasing Edge Substitution Cost**

   - **Edge Substitution Cost:** Reduced from 1.0 $\rightarrow$ 0.5.
   - **GED:** Reduced to 13.0.

Impact: Lowering the edge substitution cost reduces the penalty for substituting edges, resulting in a smaller GED. This suggests that edge substitutions play a significant role in determining the GED.

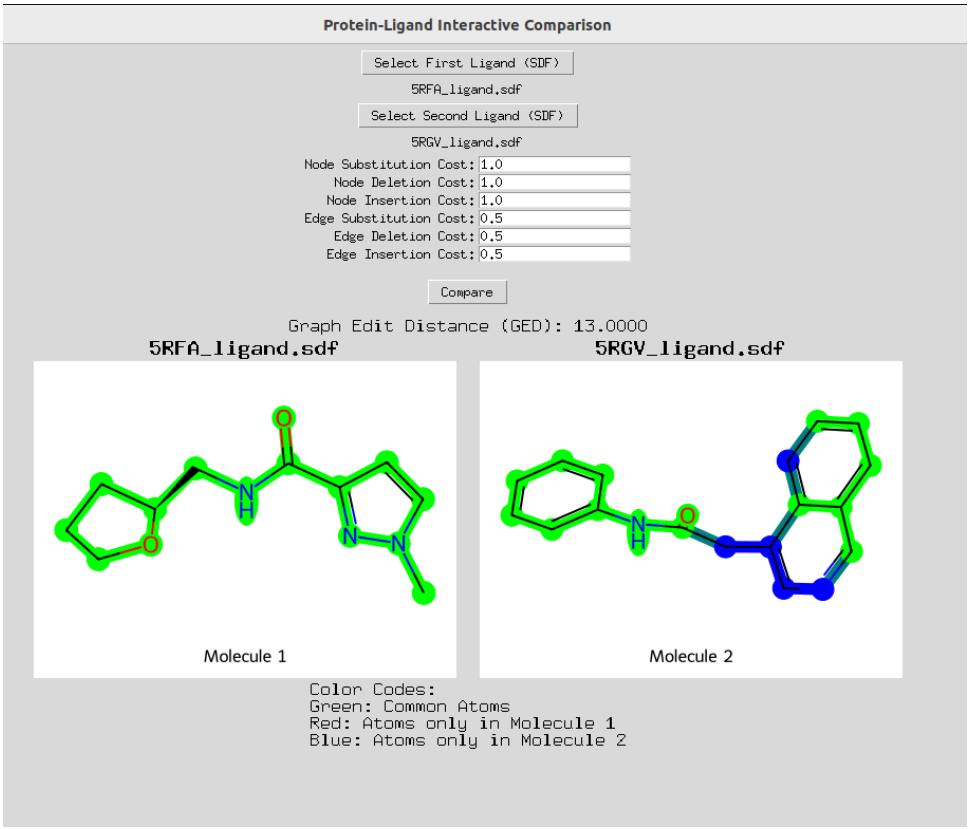

Figure 7.4: Impact of decreasing Edge Substitution Cost on GED.

5. Effect of Increasing Edge Deletion Cost

- **Edge Deletion Cost:** Increased from 0.5 $\rightarrow$ 1.0.
- **GED:** Increased to 16.0.

Impact: Similar to node deletion, increasing the edge deletion cost penalizes deletions more heavily, increasing the overall GED.

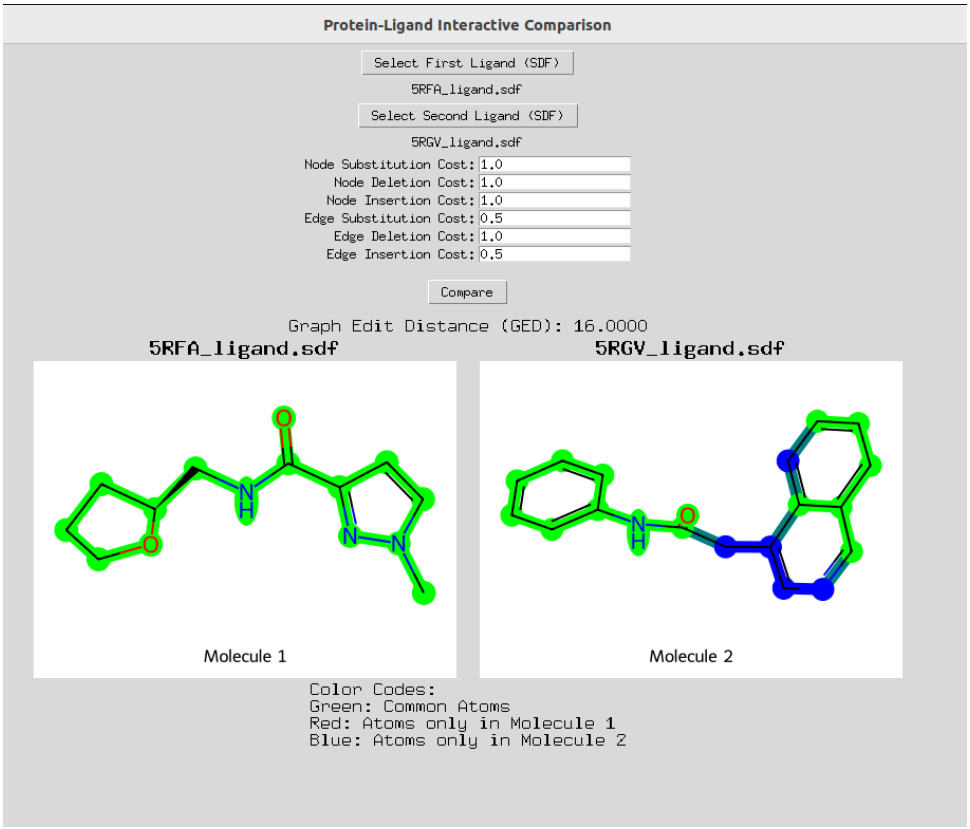

Figure 7.5: Impact of increasing Edge Deletion Cost on GED.

6. Effect of Increasing Edge Insertion Cost

- **Edge Insertion Cost:** Increased from 0.5 $\rightarrow$ 1.0.
- **GED:** Remains 16.0.

Impact: Similar to node insertion, this does not further increase GED, likely because edge insertions are infrequent in this comparison or balanced by deletions.

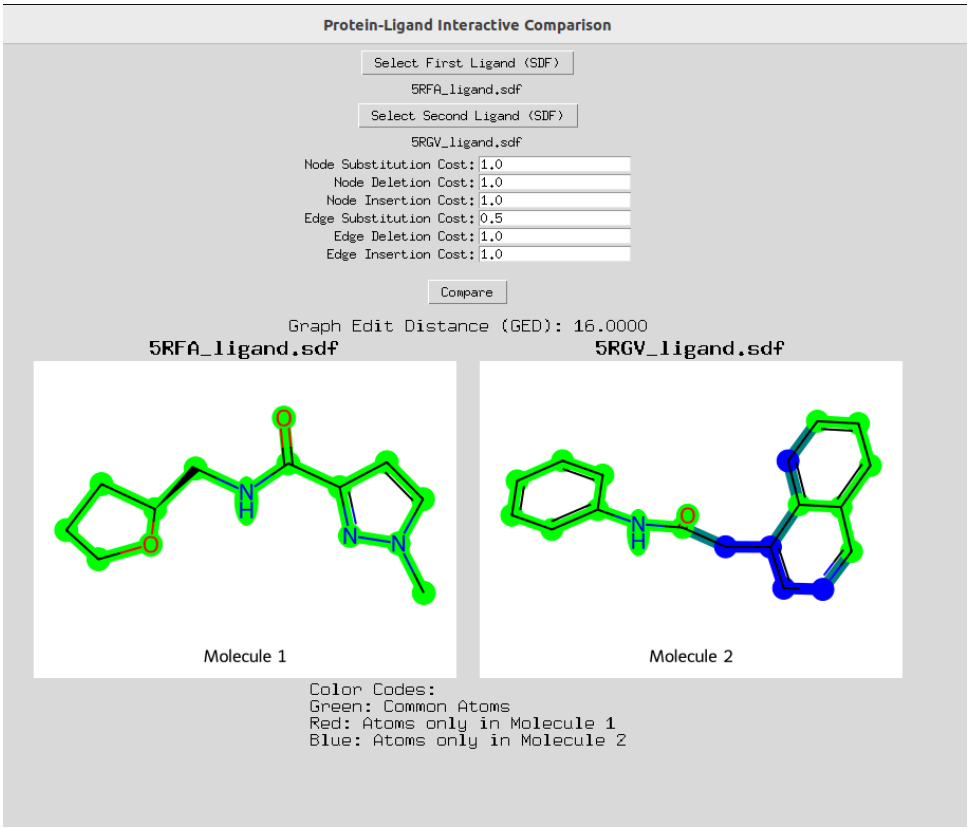

Figure 7.6: Impact of increasing Edge Insertion Cost on GED.

7. Effect of Decreasing Node Substitution Cost

- **Node Substitution Cost:** Reduced from 1.0 $\rightarrow$ 0.5.
- **GED:** Reduced to 15.0.

Impact: Reducing node substitution cost lowers the penalty for these operations, reducing the overall GED. Node substitutions are significant contributors to the edit distance in this case.

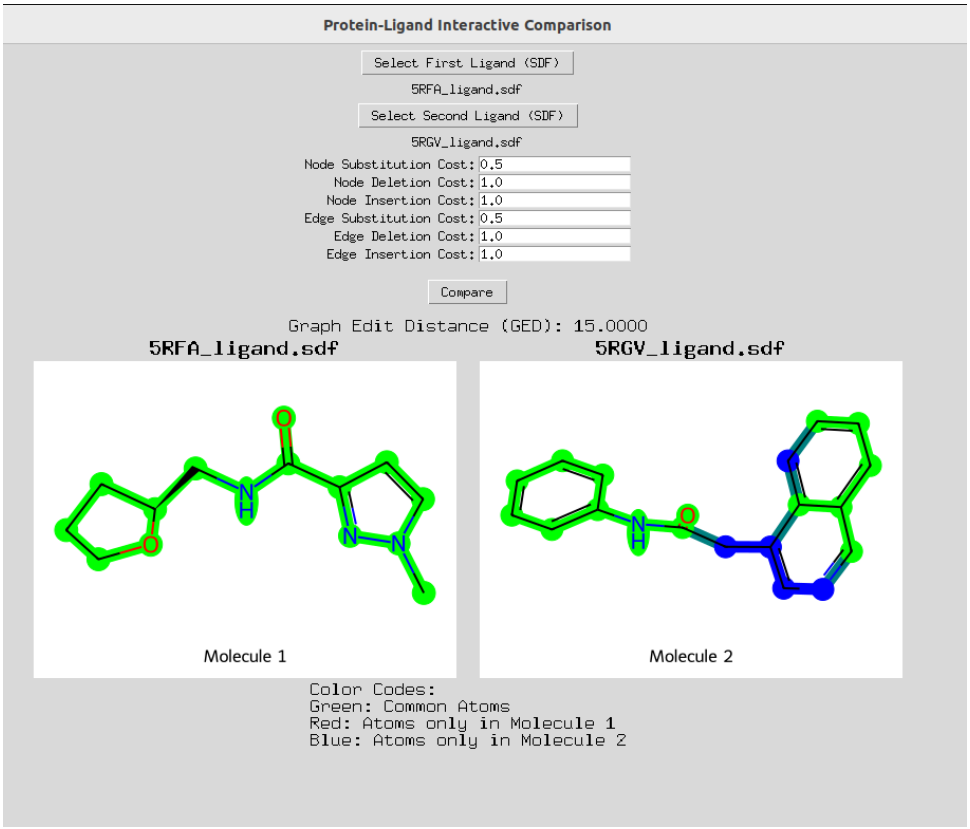

Figure 7.7: Impact of decreasing Node Substitution Cost on GED.

7.1.4 Summary of Sensitivities

- **Most Sensitive Parameters:**
  - Node Deletion and Edge Deletion Costs: Increasing these costs consistently raises the GED significantly, indicating their dominance in the edit distance calculation.
- **Moderately Sensitive Parameters:**
  - Edge Substitution Cost: Reducing this cost decreases GED, showing a clear impact on overall distance.
- **Less Sensitive Parameters:**
  - Node and Edge Insertion Costs: Changing these costs has a limited or no effect on GED, likely due to their lower frequency in the comparison process.
- **Balanced Parameters:**
  - Node Substitution Cost: Reducing this cost decreases the GED, suggesting that substitutions play a role but not as dominant as deletions.

Scenario	Node Substitution	Node Deletion	Node Insertion	Edge Substitution	Edge Deletion	Edge Insertion	GED
Default	1.0	0.5	0.5	1.0	0.5	0.5	13.5
Node Deletion ↑	1.0	1.0	0.5	1.0	0.5	0.5	16.0
Node Insertion ↑	1.0	1.0	1.0	1.0	0.5	0.5	16.0
Edge Substitution ↓	1.0	1.0	1.0	0.5	0.5	0.5	13.0
Edge Deletion ↑	1.0	1.0	1.0	0.5	1.0	0.5	16.0
Edge Insertion ↑	1.0	1.0	1.0	0.5	1.0	1.0	16.0
Node Substitution ↓	0.5	1.0	1.0	0.5	1.0	1.0	15.0

Table 7.1: Comparison of GED with Changing Costs

7.1.5 Scientific Implications

This framework provides an advanced tool for ligand analysis, enabling researchers to:

- Compare molecular structures with precision.
- Tailor comparison metrics to the specific requirements of their study.
- Visualize and analyze structural differences in an interactive, real-time environment.

Such a tool is critical for tasks like drug design, where structural similarity and binding affinity play pivotal roles.

7.2 Graph Edit Distance (GED) Calculation and Case Analysis for Protein-Ligand Compound Comparison

This section describes the development of a compound comparison editor designed to calculate Graph Edit Distance (GED) between two structurally similar molecules represented in `.sdf` files. The editor is tailored to accommodate changes in real-time in the graph structure, such as node insertions, deletions, substitutions, and edge modifications. These operations are evaluated based on a set cost function, ensuring the GED reflects the structural differences precisely.

7.2.1 Editor Overview

The editor enables users to select two `.sdf` files representing chemical compounds for comparison. The key features include:

- **Compound Loading:** Supports loading structurally valid and invalid compounds, ensuring GED calculations proceed even with incomplete or erroneous structures.
- **Case-Based GED Analysis:** Allows detailed GED calculation based on specific structural changes outlined in the cases below.
- **Error Handling:** Provides a user-friendly explanation of changes, detailing node and edge modifications.

7.2.2 Graph Representation

In the context of molecular graphs:

- **Nodes:** Represent atoms, characterized by their attributes (e.g., element type, charge).
- **Edges:** Represent bonds, characterized by bond type (e.g., single, double).

GED is calculated as the sum of costs for the following operations:

1. Node insertion.
2. Node deletion.
3. Node substitution (attribute modification).
4. Edge insertion.
5. Edge deletion.

Each operation's cost is predefined and context-dependent, reflecting the chemical or structural significance of the change.

7.2.3 Few examples for GED Analysis and how the Editor works real time

Case 1: Node Deletion

Scenario: A node is deleted from `bbb`, along with all its edges (bonds).
Calculation:

$$\text{GED} = \text{Cost of node deletion} + N_{\text{bonds deleted}} \times \text{Cost of bond deletion}. \tag{7.1}$$

Output: Specifies the deleted node, the number of bonds removed, and the total cost.

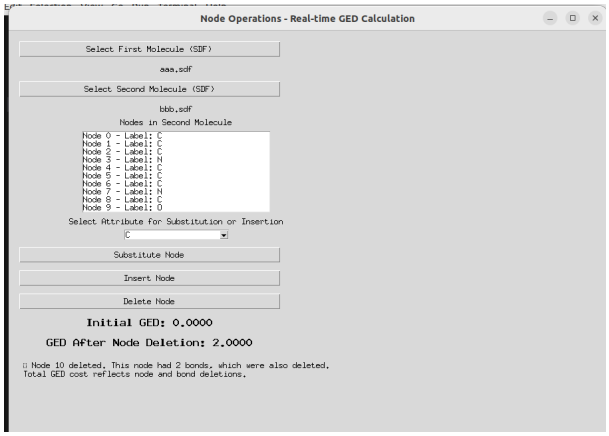

Figure 7.8: Visualization of Case 1: Node Deletion

When a node is deleted from the graph, the Graph Edit Distance (GED) is updated to reflect the deletion of the node and its associated bonds. In this example, a single node deletion incurs a cost of 1.0, and the removal of its two bonds adds 1.0 more, resulting in a total GED increase of 2.0.

```
Debug: Node to delete = 10, Edges to delete = [(10, 8), (10, 0)]
Debug: Node deletion cost = 1.0, Edge deletion cost = 1.0
Debug: Total GED = 2.0
```

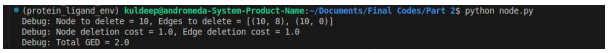

Figure 7.9: A depiction of the node deletion event. The removal of Node 10 and its two connected edges increased the GED by 2.0 units.

Case 2: Edge Deletion

Scenario: The `.sdf` files `Molecule 1` and `Molecule 2` are worked on here and one edge (bond) is removed from `bbb`.
Calculation:

$$\text{GED} = \text{Cost of edge deletion.} \tag{7.2}$$

Output: Specifies the removed bond and the associated cost.

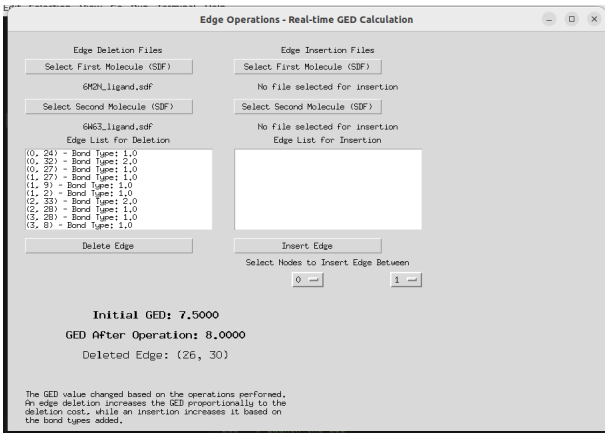

Figure 7.10: Visualization of Case 2: Edge Deletion.

Case 3: Node Substitution

Scenario: In `second molecule`, the attribute of one atom (node) is substituted.
Calculation:

$$\text{GED} = \text{Cost of node substitution.} \tag{7.3}$$

Output: Specifies the node's attribute change and associated cost.

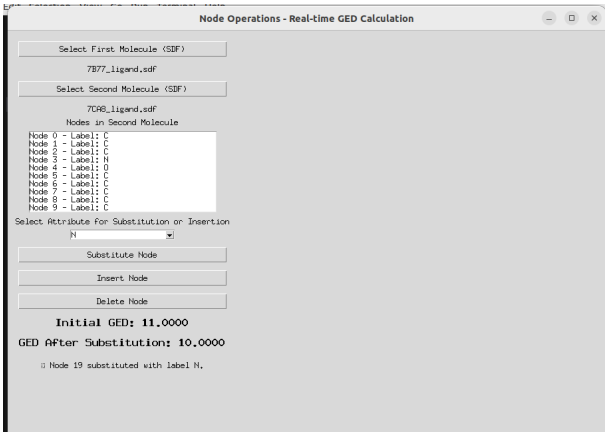

Figure 7.11: Visualization of Case 3: Node Substitution.

Case 5: Node Insertion

Scenario: A new node (atom) is added to `bbb` with distinct attributes.
Calculation:

$$\text{GED} = \text{Cost of node insertion.} \tag{7.4}$$

Output: Specifies the new node and the associated cost.

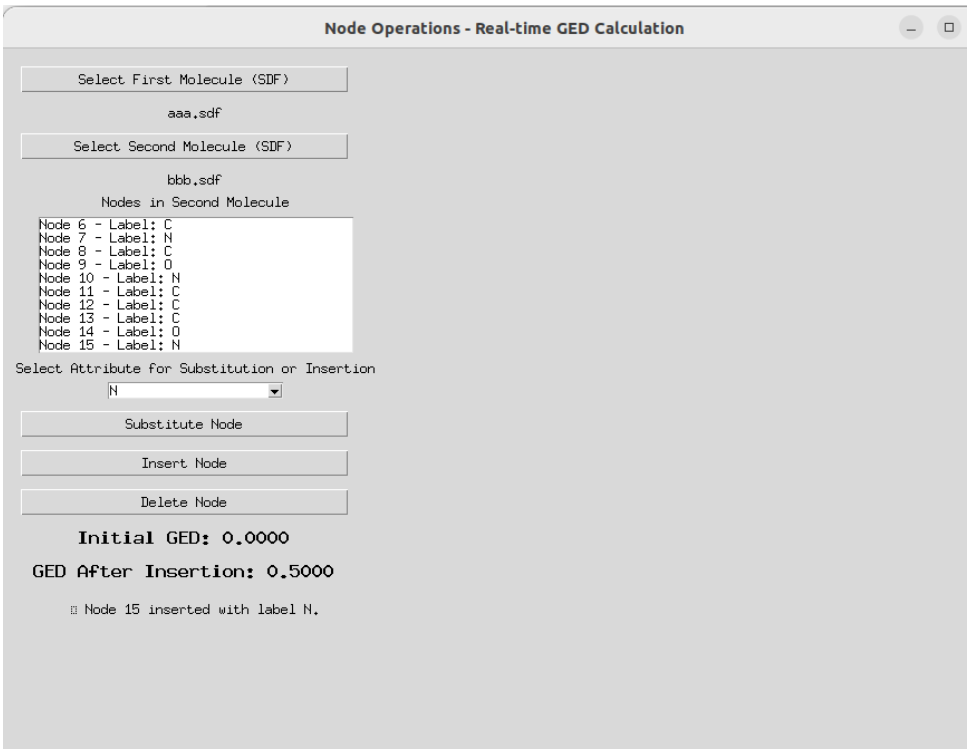

Figure 7.12: Visualization of Case 5: Node Insertion.

Case 6: Edge Insertion

Scenario: A new edge (bond) is added to `bbb`.
Calculation:

$$\text{GED} = \text{Cost of edge insertion.} \tag{7.5}$$

Output: Specifies the inserted bond and the associated cost.

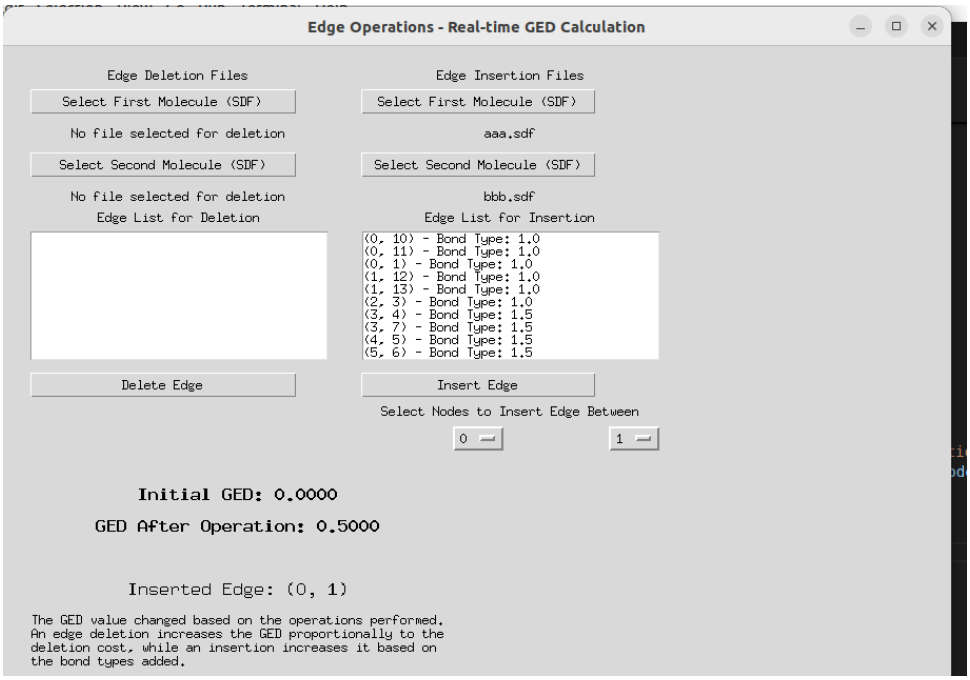

Figure 7.13: Visualization of Case 6: Edge Insertion.

For Part 2: KNN-Based Binding Affinity Prediction

- Use KNN to predict the binding affinity of compounds based on the URV database.
- **Related Code:** `secondpart.py`, `secondpartanalysis.py`, `mordredplot.py`:
  - Implements KNN-based prediction using descriptors (Mordred/PaDEL) and GED as features.
  - `apply_knn` in `secondpart.py` and `knn_analysis_mordred` in `mordredplot.py` are central to this.
- Performs KNN analysis on test/learn datasets split from a database.
- Focuses on descriptor-based affinity prediction and error visualization.

Editor Implementation of KNN and GED-Based Comparison

In the second stage of the project, we implemented a user interface—referred to here as the "editor"—that allows for streamlined input selection and automated comparative analysis. Within this editor, users are first prompted to select a single SDF file representing the reference compound of interest. Next, they are given the option to select a folder containing multiple SDF files and a corresponding `affinity.txt` file, which provides affinity values associated with each molecule in that dataset.

Once these inputs are specified, the system computes the Graph Edit Distance (GED) between the user-selected reference compound and every compound within the chosen folder. The GED serves as a measure of structural similarity: a lower GED indicates a

molecule that more closely resembles the reference structure. Subsequently, a k-Nearest Neighbors (kNN) regression model is applied, integrating both molecular descriptors and computed GED values to predict affinity. Through this approach, the system identifies the molecule that exhibits the minimum GED relative to the reference compound, thus highlighting the most structurally similar candidate within the provided dataset.

Finally, the interface displays the minimum GED value, visualizes the identified molecule, and presents the corresponding affinity associated with this closest structural match. This integrated solution provides both a quantitative measure of molecular similarity and a predictive estimate of affinity, thereby offering a valuable tool for efficient structure-affinity exploration and decision-making in drug discovery workflows.

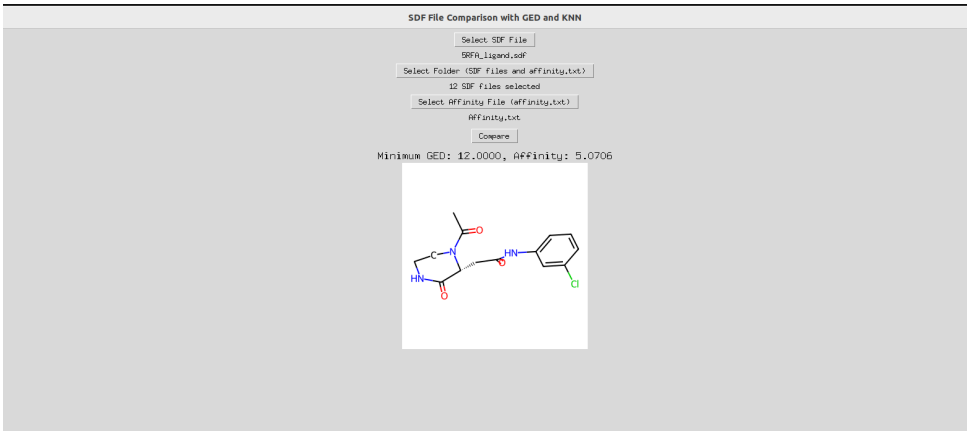

Figure 7.14: KNN-Based Binding Affinity Prediction.png

Code Flow

1. **User Input of Reference Molecule:** The user selects the primary SDF file representing the reference compound of interest.

2. **User Input of Dataset Folder:** The user then selects a directory containing a collection of SDF files and an accompanying `affinity.txt` file, ensuring that each molecule in the dataset has an associated affinity value.

3. **Data Preparation:** The system parses the reference SDF file and all SDF files in the dataset folder, converting each molecule into its corresponding graph representation.

4. **GED Computation:** For every molecule in the selected dataset, the system calculates the Graph Edit Distance (GED) relative to the reference molecule. This step quantifies structural similarity, with lower GED values indicating greater resemblance.

5. **Identification of Closest Match:** The system identifies the molecule in the dataset that yields the minimum GED value, thereby pinpointing the compound most structurally similar to the reference molecule.

6. **Molecular Visualization:** The identified closest-match molecule is rendered and displayed visually within the editor, providing users with an immediate understanding of its structural features.

7. **Affinity Retrieval and Reporting:** The affinity value corresponding to the closest-match molecule is extracted from the `affinity.txt` file, and presented alongside the minimum GED value.

8. **Application of kNN Regression:** A k-Nearest Neighbors (kNN) model is applied to integrate molecular descriptors and GED values, ultimately refining the system's insight into structure-affinity relationships.

9. **Result Display:** The editor outputs the minimum GED, the affinity associated with the closest-match molecule, and the visual representation of that molecule, thereby delivering a comprehensive overview of structural and affinity-based comparisons.

Structure-affinity relationship analysis

In this stage of the project, the code performs a structure-affinity relationship analysis: it takes a reference molecule, compares it to a database of molecules using molecular descriptors and the Graph Edit Distance (GED), trains a k-Nearest Neighbors (kNN) model from that database's known affinities, and then uses the model to predict the affinity of the reference molecule. The terminal output shown below represents debugging print statements illustrating the internal data processing steps and the final predicted affinity.

Terminal Output

The selected molecule's descriptors were calculated as:

```
Selected molecule descriptors: {'MolWt': 209.24899999999997,
'NumHDonors': 1, 'NumHAcceptors': 4, 'TPSA': 56.150000000000006,
'NumRotatableBonds': 3, 'MolLogP': 0.3289}
```

The folder molecules were compared, and their descriptors and GED values computed. A sample of the output is:

```
Descriptors for 7M8M_ligand.sdf: {'MolWt': 450.8820000000002, 'NumHDonors': 2, 'NumHA
'TPSA': 109.84, 'NumRotatableBonds': 6, 'MolLogP': 3.3853000000000018}, GED: 38.0
...
Training feature vectors: [[4.50882e+02 2.00000e+00 6.00000e+00 1.09840e+02 6.00000e+
  3.80000e+01]
 ...
Feature vector for selected molecule: [209.24899999999997, 1, 4, 56.150000000000006,
Predicted affinity: 5.11985012202609
```

GUI Output

After running the code with the chosen SDF files and the corresponding `affinity.txt` file, the user interface displays the predicted affinity value for the selected reference molecule.

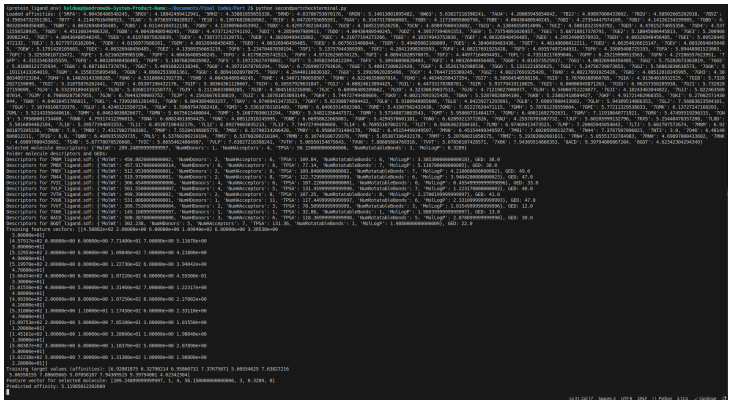

Figure 7.15: Terminal output showing debugging statements, computed descriptors, GEDs, and final predicted affinity.

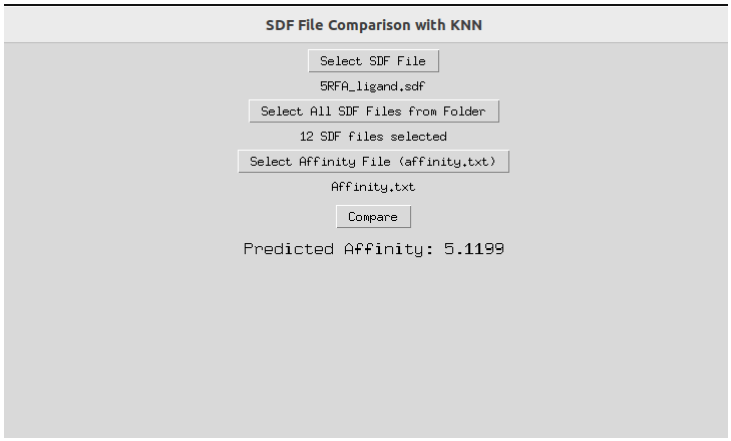

Figure 7.16: GUI display showing the predicted affinity for the selected reference molecule.

In summary, the process involves:

1. Calculating molecular descriptors for the reference and database molecules.
2. Computing the Graph Edit Distance between the reference molecule and each database molecule.
3. Constructing feature vectors from descriptors and GED values.
4. Training a KNN model on the database molecules with known affinities.
5. Predicting the affinity of the reference molecule using the trained KNN model.

This integrated approach provides a quantitative way to link molecular structure to affinity estimates, enhancing decision-making in drug discovery workflows.

7.3 Descriptor Distance and Affinity Analysis

`Descriptor Distance Calculation using Mordred`

This section highlights the calculation of descriptor distances between two ligand files (e.g., `5RFA_ligand.sdf` and `7GBV_ligand.sdf`) using Mordred molecular descriptors.

The descriptor distance is calculated based on the normalized Euclidean distance between the descriptors of the two ligands.

The GUI displays the calculated descriptor distance as shown in Figure **??**.

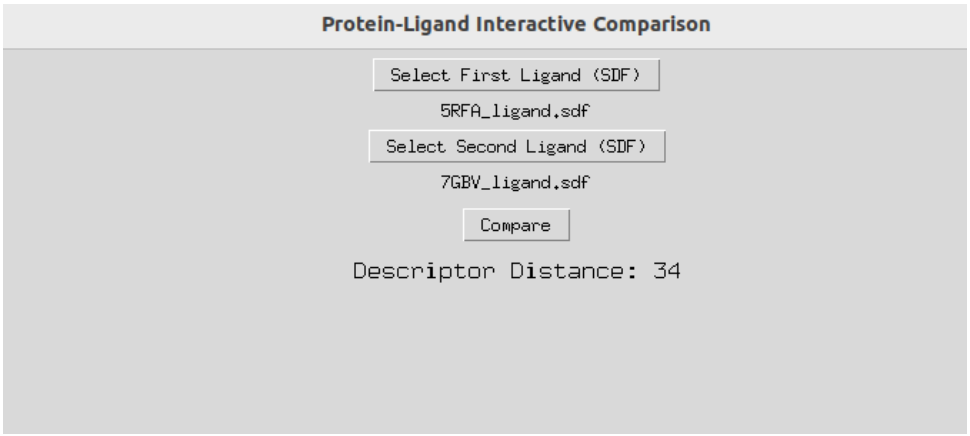

Figure 7.17: GUI displaying the descriptor distance between two ligands.

Terminal Output and Calculations

The terminal output provides detailed calculations, including descriptor values, normalization steps, and the final descriptor distance. Below is a LaTeX representation of the calculations:

Descriptors for Molecule 1 (`5RFA_ligand.sdf`):

SpAbs_A	SpMax_A	SpDiam_A	LogEE_A	MW	AMW
19.2982	2.2992	4.4583	3.6314	209.1164	6.9705

Descriptors for Molecule 2 (`7GBV_ligand.sdf`):

SpAbs_A	SpMax_A	SpDiam_A	LogEE_A	MW	AMW
24.7706	2.4088	4.8176	3.9100	288.1029	7.7866

Normalized Descriptors: The descriptors are normalized before calculating the Euclidean distance:

$$\text{Normalized Descriptors 1: } [0, 0, 0, \ldots, 0, 0, 0]$$

$$\text{Normalized Descriptors 2: } [0, 0, 0, \ldots, 1, 1, 1]$$

Euclidean Distance:

$$\text{Distance} = \sqrt{\sum_{i=1}^{n} (d_{1i} - d_{2i})^2}$$

$$\text{Final Descriptor Distance: } 34.0147 \approx 34$$

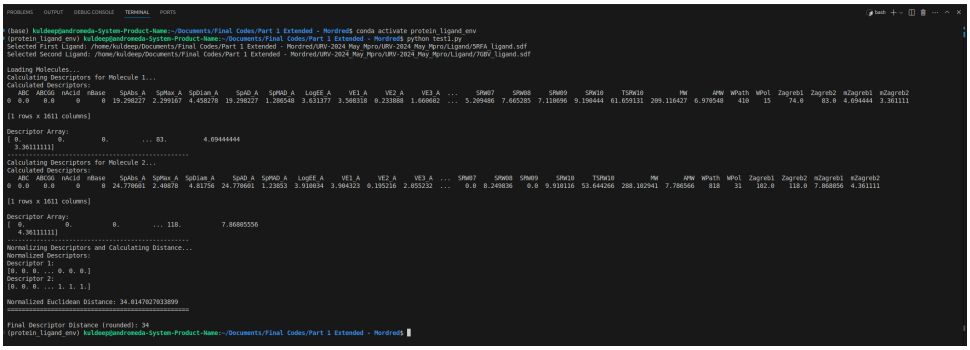

Figure 7.18: Terminal output showing descriptor calculations and normalized Euclidean distance.

Validation for Identical Molecules

When comparing identical molecules (e.g., `aaa.sdf` and `bbb.sdf`), the descriptor distance is correctly calculated as zero, as shown in the terminal output and GUI.

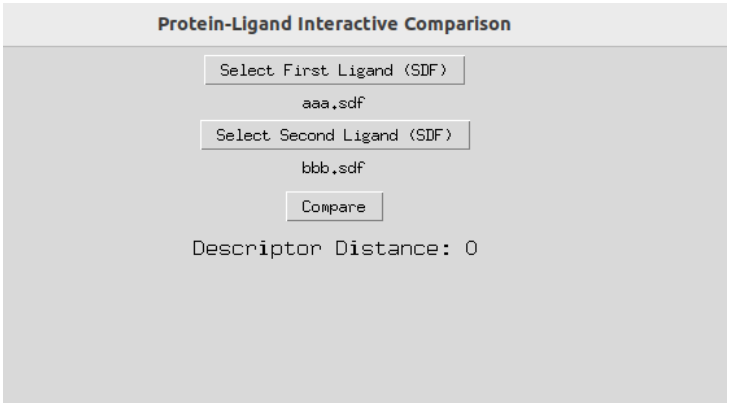

Figure 7.19: GUI displaying descriptor distance of 0 for identical molecules (`aaa.sdf` and `bbb.sdf`).

Descriptors for Molecule 1 (`aaa.sdf`):

SpAbs_A	SpMax_A	SpDiam_A	LogEE_A	MW	AMW
19.2982	2.2992	4.4583	3.6314	209.1164	6.9705

Descriptors for Molecule 2 (`bbb.sdf`):

SpAbs_A	SpMax_A	SpDiam_A	LogEE_A	MW	AMW
19.2982	2.2992	4.4583	3.6314	209.1164	6.9705

Normalized Descriptors: The descriptors for both molecules are identical:

$$\text{Normalized Descriptors 1: } [0, 0, 0, \ldots, 0, 0, 0]$$

$$\text{Normalized Descriptors 2: } [0, 0, 0, \ldots, 0, 0, 0]$$

Euclidean Distance:

$$\text{Distance} = \sqrt{\sum_{i=1}^{n}(d_{1i} - d_{2i})^2} = 0$$

Final Descriptor Distance: 0

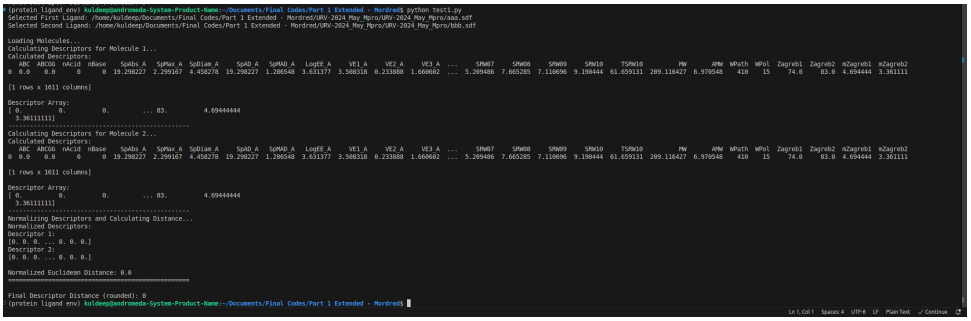

Figure 7.20: Terminal output showing descriptor calculations and normalized Euclidean distance.

Validation Check

As we saw above, To validate the descriptor distance calculation:

- **Identical Molecules:** Comparing identical molecules (e.g., `aaa.sdf` and `bbb.sdf`) results in a descriptor distance of 0, as shown in Figure 7.19.
- **Similar Molecules:** Descriptor distances for structurally similar ligands should be relatively small.
- **Dissimilar Molecules:** Descriptor distances for structurally dissimilar ligands should be relatively large.

`mordredanalysis.py`

This part of the thesis project analyzes molecular descriptors and affinity differences for protein-ligand interactions.

- Computes descriptor distances and affinity differences between test and learning sets.
- Visualizes the relationship between descriptor distances and affinity differences using scatter plots.

Its functionality can be summarized as follows:

1. **Molecular Descriptor Calculation:**
    - Uses the Mordred library to compute molecular descriptors that capture chemical and structural properties of test and learning molecules.
2. **Descriptor Distance Matrix:**

- Calculates pairwise Euclidean distances between normalized descriptors of test and learning datasets, representing structural similarity.
- The Euclidean distance between two descriptor vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ is calculated as:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}$$

where $x_{i,k}$ and $x_{j,k}$ represent the k-th feature of the i-th and j-th descriptors, respectively.

3. **Affinity Difference Matrix:**
   - Computes the absolute differences in binding affinities between test and learning molecules.
   - The absolute affinity difference for molecule i in the test set and molecule j in the learning set is computed as:

$$\Delta A_{i,j} = |A_i - A_j|$$

where A_i and A_j are the affinities of the i-th test molecule and j-th learning molecule, respectively.

4. **Visualization:**
   - Generates a scatter plot to illustrate the relationship between descriptor distances (x-axis) and affinity differences (y-axis), revealing potential correlations.

5. **Output:**
   - Prints intermediate results (e.g., descriptor values, matrices) for debugging and insight.
   - Saves the scatter plot as a PNG file for further analysis and inclusion in documentation.

Output Description

- **Descriptor Distance Matrix (Example):**

$$\begin{bmatrix} 9.76 & 11.56 & 10.86 & \cdots & 15.34 \\ 13.91 & 15.88 & 13.24 & \cdots & 19.55 \\ 13.54 & 13.36 & 13.62 & \cdots & 17.29 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 10.65 & 12.66 & 10.74 & \cdots & 16.18 \end{bmatrix}$$

- **Affinity Difference Matrix (Example):**

$$\begin{bmatrix} 0.00 & 0.18 & 0.33 & \cdots & 0.62 \\ 0.18 & 0.00 & 0.15 & \cdots & 0.43 \\ 0.33 & 0.15 & 0.00 & \cdots & 0.28 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.62 & 0.43 & 0.28 & \cdots & 0.00 \end{bmatrix}$$

Equations Used in the Analysis

- **Normalization:**

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

  where x is the original descriptor value, and $x_{\min}$ and $x_{\max}$ are the minimum and maximum values of the descriptor across the dataset.

- **Descriptor Distance:**

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{k=1}^{n} (x_{i,k} - x_{j,k})^2}$$

- **Absolute Affinity Difference:**

$$\Delta A_{i,j} = |A_i - A_j|$$

Example Output (Truncated)

```
Loading Test Molecules and Affinities...
Loaded 12 molecules and 233 affinities from /home/kuldeep/Documents/Final Codes/t
and /home/kuldeep/Documents/Final Codes/Affinity.txt

Loading Learn Molecules and Affinities...
Loaded 131 molecules and 233 affinities from /home/kuldeep/Documents/Final Codes/
and /home/kuldeep/Documents/Final Codes/Affinity.txt

Calculating Descriptor Distance Matrix...
Descriptor Distance Matrix:
 [[ 9.76 11.56 10.86 ... 15.34]
 [13.91 15.88 13.24 ... 19.55]
 ...
 [10.65 12.66 10.74 ... 16.18]]

Calculating Affinity Difference Matrix...
Affinity Difference Matrix:
 [[0.00 0.18 0.33 ... 0.62]
 [0.18 0.00 0.15 ... 0.43]
 ...
 [0.62 0.43 0.28 ... 0.00]]

Visualizing Results...
Plot saved as descriptor_distance_vs_affinity_difference.png
```

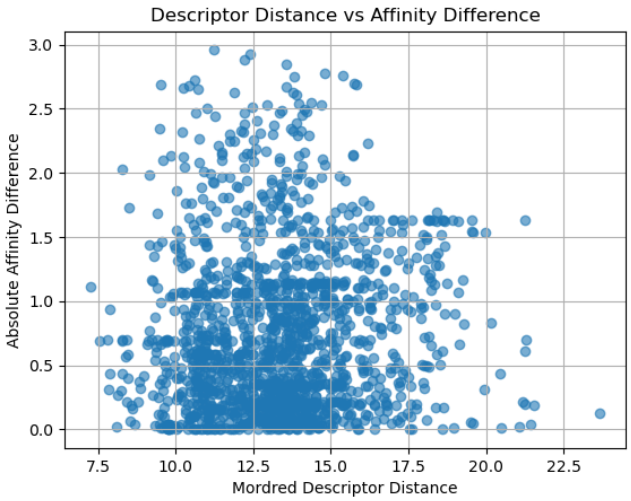

Figure 7.21: Scatter plot of Descriptor Distance vs Affinity Difference.

- **Scatter Plot:**
  - Shows clusters of data points where smaller descriptor distances correspond to lower affinity differences, indicating a potential relationship between molecular structure and binding affinity.
  - The plot reveals variability at larger distances, suggesting the need for further exploration of non-linear patterns.

`mordredplot.py`

This part of the Masters Thesis project implements a K-Nearest Neighbors (KNN) regression model using Mordred molecular descriptors to predict molecular affinities for protein-ligand interactions.

- Uses Mordred descriptors and KNN regression to predict molecular affinities.
- Visualizes predictions, errors, and true affinities using bar plots.

Below is a summary of its functionality:

1. **Molecular Descriptor Calculation:**
   - Uses the Mordred library to compute chemical and structural properties (descriptors) for test and learning molecules.
   - Descriptors are normalized using Min-Max scaling to ensure uniformity:

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

   where x is the original descriptor value, and $x_{\min}$ and $x_{\max}$ are the minimum and maximum values in the dataset.

2. **Data Loading:**

- Reads molecular structures from `.sdf` files and their corresponding affinity values from a separate affinity file.
- Molecules are grouped into "test" and "learning" sets.

3. **KNN Regression:**
   - Trains a KNN regression model using molecular descriptors of the learning set and their affinities.
   - Uses the trained model to predict affinities for the test set based on their descriptors.
4. **Error and MSE Calculation:**
   - Computes squared errors between predicted and true affinities for each test molecule:
   $$\text{Error}_i = (\text{True Affinity}_i - \text{Predicted Affinity}_i)^2$$
   - Calculates the Mean Squared Error (MSE) as:
   $$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (\text{True Affinity}_i - \text{Predicted Affinity}_i)^2$$
   where N is the number of test molecules.
5. **Visualization:**
   - Plots a bar chart showing squared errors for each compound.
   - Overlays true and predicted affinities on the same plot to visualize deviations.

Output Description

The output of the script includes the following:

1. **Detailed Results:** The true affinities, predicted affinities, and squared errors for each compound are summarized in Table 7.2.

Table 7.2: True Affinities, Predicted Affinities, and Errors for Test Compounds.

Compound	True Affinity	Predicted Affinity	Error (Squared)
1	4.2147	4.3250	0.0122
2	4.0379	4.4588	0.1772
3	5.0737	4.5458	0.2786
4	4.3388	4.2935	0.0021
5	4.2007	5.1698	0.9393
6	5.6383	5.7243	0.0074
7	4.0044	4.0070	0.0000
8	4.1884	4.8821	0.4812
9	4.6990	4.0472	0.4248
10	4.5850	5.7153	1.2776
11	5.1401	5.7454	0.3664
12	4.3565	4.2912	0.0043

2. **Mean Squared Error (MSE):** The MSE is calculated as:

$$\text{MSE} = \frac{1}{12} \sum_{i=1}^{12} \text{Error}_i$$

Substituting the squared errors from Table 7.2, the final MSE is:

$$\text{MSE} = 0.3309$$

3. **Visualization:** The generated plot includes:
   - A bar chart showing the squared error for each compound.
   - Line plots for true affinities and predicted affinities for direct comparison.

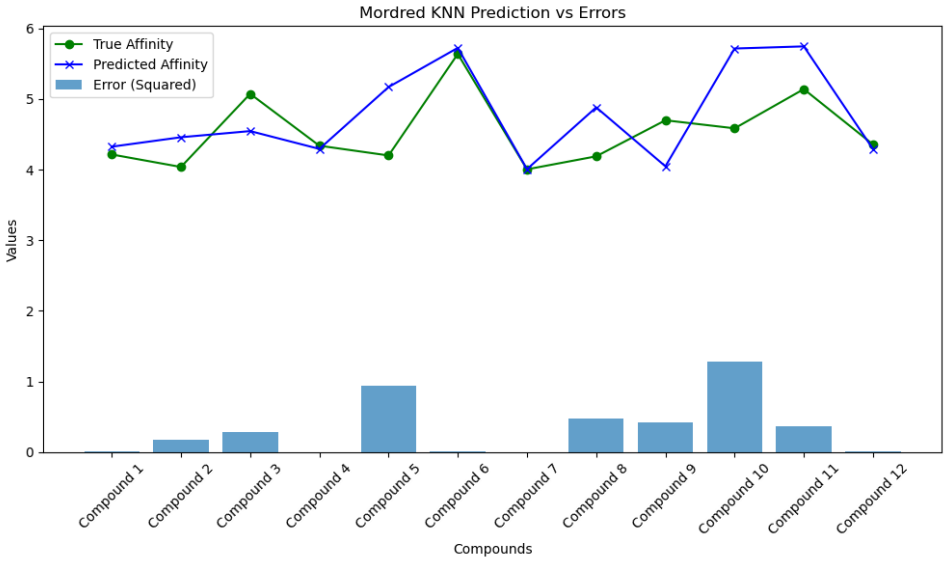

Figure 7.22: Bar plot showing the squared errors for each compound along with the true and predicted affinities. The bar chart represents the squared errors for each compound, while the line plots overlay the true and predicted affinities, providing a clear visualization of the KNN model's performance.

Key Insights

- The KNN model performs reasonably well with a low MSE (**0.3309**), suggesting that predicted affinities are close to true affinities on average.
- Larger errors for some compounds (e.g., Compound 10) may indicate outliers or limitations of the model.
- The visualization effectively highlights deviations between true and predicted values, providing insights into the model's predictive performance for each compound.

Chapter 8

For Part 3: Analysis and Discussion

8.1 Error Analysis

The method's errors are analyzed across different affinity ranges. High-affinity compounds show reduced errors compared to low-affinity compounds.

1. Error Analysis

Requirement

- Analyze errors per compound and calculate Mean Squared Error (MSE).
- Split the database by binding affinity and identify trends in error reduction.

Related Code

- `part3_analysis.py` and `secondpartanalysis.py`:
  - Perform detailed analysis by comparing GED with affinity differences and calculating MSE.
  - Split datasets to study variations in error distribution.

8.1.1 K-Nearest Neighbors (KNN) Regression Analysis

This part of the thesis project calculates molecular descriptors for chemical compounds using the Mordred library, processes the data, and uses these descriptors as features for machine learning. Molecules are loaded from `.sdf` files, and their corresponding affinity scores from a text file, splitting the dataset into training (75%) and testing (25%) sets. A K-Nearest Neighbors (KNN) regression model is trained to predict affinities for the test set, and the performance is evaluated using the Mean Squared Error (MSE). The results, including true and predicted affinities and their errors, are visualized in plots and saved in the specified output directory for further analysis.

Results:

- **Mean Squared Error (MSE)**: 1.0416

- **Visualizations**: The plot showing squared errors for each compound and true vs. predicted affinities is saved as `knn_mordred_analysis.png`.

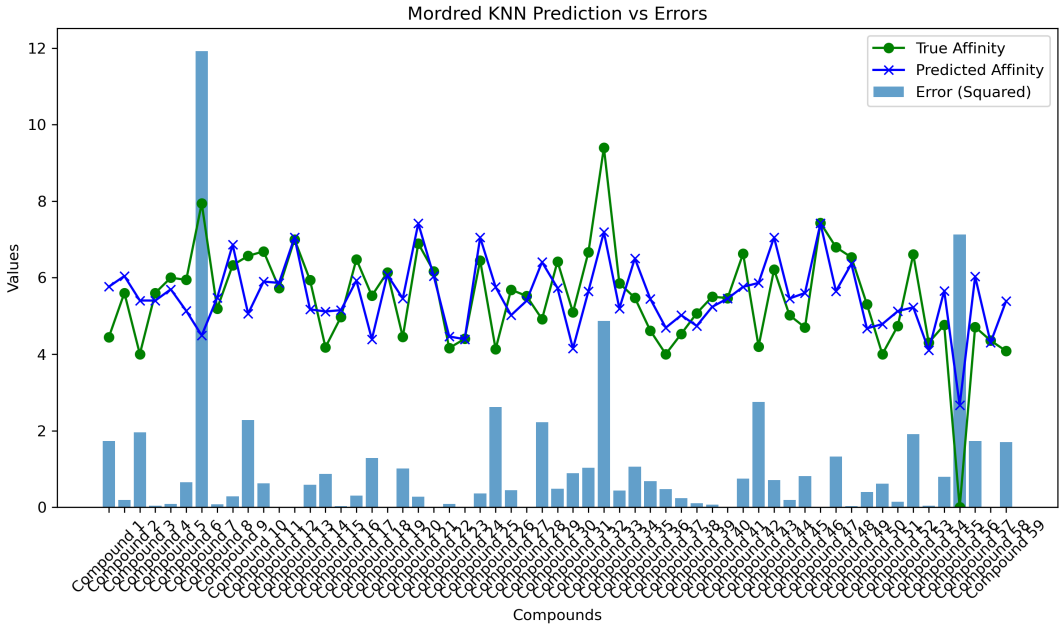

Figure 8.1: KNN Regression: Predicted vs. True Affinity and Errors

Detailed Calculations (Sample Results):

Compound	True Affinity	Predicted Affinity	Error (Squared)
Compound 1	4.4498	5.7651	1.7301
Compound 2	5.6027	6.0436	0.1944
Compound 3	4.0044	5.4042	1.9595
Compound 7	7.9431	4.4906	11.9196
Compound 56	0.0000	2.6692	7.1247

Table 8.1: Sample results of true and predicted affinities with errors.

Key Observations:

- Most predicted affinities are reasonably close to the true values, with lower errors for high-affinity compounds.
- Larger errors are observed for compounds with low affinities, such as Compound 7 and Compound 56.

8.1.2 GED vs. Affinity Differences - The structural differences and binding affinity differences for molecules stored in the test and learn folders.

This computes the relationship between Graph Edit Distance (GED) and absolute binding affinity differences for chemical compounds. Molecules are converted into attributed graphs, with nodes representing atoms and edges representing bonds. A GED matrix is

computed to represent structural differences, while an Affinity Difference Matrix calculates the absolute differences between binding affinities for corresponding GED entries.

This part of the thesis project analyzes the relationship between structural differences and binding affinity differences for molecules stored in the **test** and **learn** folders. It converts molecular structures from `.sdf` files into attributed graphs and calculates the **Graph Edit Distance (GED)** to quantify structural dissimilarity. It computes the absolute difference in binding affinities using values loaded from `affinity.txt` files.

The results are saved as two matrices:

- `GED_Matrix.csv` (containing GED values)
- `Affinity_Difference_Matrix.csv` (containing absolute affinity differences).

Summary statistics (mean, median, standard deviation, min, and max) for both matrices are printed to the console. A scatter plot visualizing the correlation between GED and affinity differences is generated and saved as `GED_vs_Affinity_Difference.png`. This analysis helps identify trends between molecular structural changes and their impact on binding affinity differences.

Output

```
(protein_ligand_env) kuldeep@andromeda-System-Product-Name:~/Documents/Final Codes$ p
GED Matrix Summary:
Mean: 16.38, Median: 15.00, Std Dev: 8.65
Min: 0.00, Max: 50.00
Affinity Difference Matrix Summary:
Mean: 0.90, Median: 0.68, Std Dev: 0.84
Min: 0.00, Max: 5.39
Matrices saved as CSV files: GED_Matrix.csv and Affinity_Difference_Matrix.csv.

Pearson correlation coefficient: 0.3106, p-value: 1.6554e-36
Scatter plot saved as GED_vs_Affinity_Difference.png.
(protein_ligand_env) kuldeep@andromeda-System-Product-Name:~/Documents/Final Codes$
```

Output Analysis

Matrix Summaries

GED Matrix Summary

- **Mean (16.38):** On average, the Graph Edit Distance (GED) between molecules in the test and learn folders is 16.38.
- **Median (15.00):** Half of the GED values are below 15.00, and half are above.
- **Standard Deviation (8.65):** There is moderate variation in GED values, indicating diverse molecular differences.
- **Min (0.00):** The closest pair of molecules (in terms of graph structure) has a GED of 0.00, meaning they are identical.
- **Max (50.00):** The most dissimilar pair of molecules has a GED of 50.00.

Affinity Difference Matrix Summary

- **Mean (0.90):** On average, the absolute difference in binding affinity between pairs of molecules is 0.90.
- **Median (0.68):** Half of the affinity differences are below 0.68, and half are above.
- **Standard Deviation (0.84):** There is considerable variation in affinity differences, suggesting some molecules have very similar affinities, while others differ significantly.
- **Min (0.00):** Some molecules have identical binding affinities.
- **Max (5.39):** The largest affinity difference between any pair is 5.39.

Scatter Plot Analysis

X-Axis: Graph Edit Distance (GED) Represents the structural dissimilarity between a pair of molecules.

Y-Axis: Absolute Affinity Difference Represents the absolute difference in binding affinities for each pair.

Observations

- **Low GED, Low Affinity Difference:** Points near the origin (low x and y values) represent molecules that are structurally similar and have similar binding affinities. These pairs likely have only minor variations in structure and binding behavior.
- **High GED, High Affinity Difference:** Points in the top-right corner indicate molecules with significant structural differences and large affinity differences. These pairs might belong to entirely different categories of compounds.
- **Low GED, High Affinity Difference:** Points along the lower-right region (low x, high y) suggest that even minor structural differences can lead to significant binding affinity differences. These pairs could represent cases where small changes in structure drastically affect binding, such as key functional group alterations.
- **High GED, Low Affinity Difference:** Points along the upper-left region (high x, low y) suggest that even with significant structural differences, the binding affinity remains similar. This could indicate redundancy in binding sites or structural changes that do not affect functional regions.

Results:

- **Pearson Correlation Coefficient**: 0.3106
- **P-value**: 1.6554×10^{-36}
- **Visualizations**: The scatter plot of GED vs. Absolute Affinity Differences is saved as `ged_vs_affinity_diff.png`.

Compound Pair	GED	Absolute Affinity Difference
7B5Z vs. 8ACD	29.0	5.0414
7B5Z vs. 7GFG	15.0	0.2614
7B5Z vs. 7GFZ	16.0	0.3533
7B5Z vs. 7GG7	22.0	1.1126
7B5Z vs. 7GGY	18.0	0.4079

Table 8.2: Sample GED and absolute affinity differences for compound pairs.

Detailed Calculations (Sample Results):

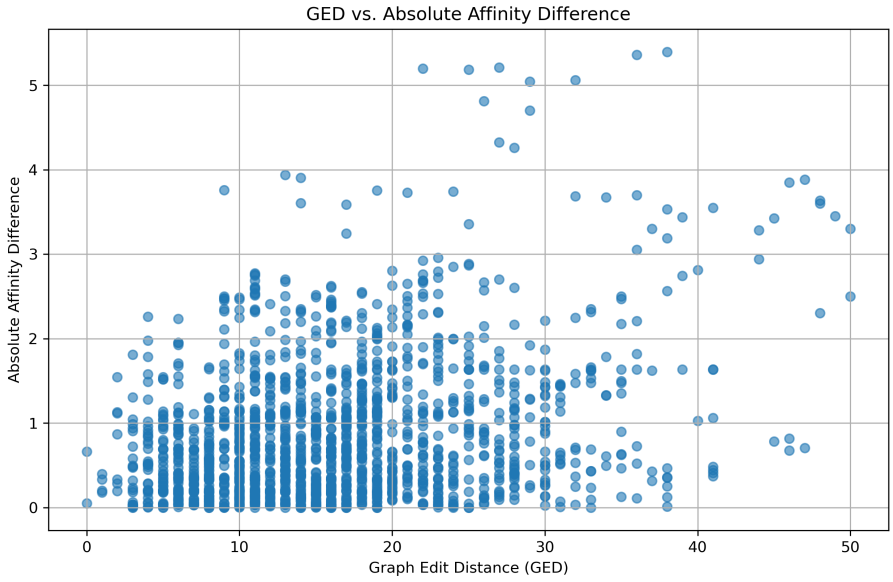

Figure 8.2: GED vs. Absolute Affinity Differences

Key Observations:

- Points in the scatter plot are concentrated in the lower-left quadrant, indicating many compounds have small structural differences (low GED) and small affinity differences.
- A moderate positive correlation exists between GED and affinity differences, but outliers highlight cases with significant deviations.

Key Insights

Relationship Between GED and Affinity Difference: The scatter plot shows a weak positive correlation (**Pearson correlation coefficient: 0.3106**), indicating that structural differences do not always predict affinity differences in a straightforward manner. There is no strictly linear relationship, suggesting the influence of additional factors beyond structural similarity.

Clustering Patterns: Clustering patterns in the scatter plot may be explored further to understand molecule groups with specific characteristics.

Further Analysis:

- Analyze specific pairs with the highest GED and affinity differences to understand outliers.
- Compare clusters of molecules with low GED and similar affinities to identify structurally redundant compounds.

8.1.3 GED and Affinity Difference Matrices - A small example for better understanding

GED Matrix:

	1	2	3	4	5	6	7	8	9	10
1	21	21	33	48	41	47	12	13	38	22
2	22	4	33	49	30	36	15	19	27	8
3	21	10	34	50	36	41	12	14	32	12
4	20	19	33	48	40	46	11	14	36	20
5	22	8	31	50	31	39	15	20	28	4
6	24	26	30	24	9	18	31	48	9	30
7	10	11	25	39	26	32	6	24	22	11
8	16	12	30	44	32	38	8	17	29	14
9	9	14	23	36	28	37	4	25	26	16
10	16	12	31	44	32	38	8	17	29	14

Table 8.3: Graph Edit Distance (GED) matrix. GED(i, j) represents the structural difference between test molecule i and learn molecule j.

Affinity Difference Matrix:

	1	2	3	4	5	6	7	8	9	10
1	0.69	1.67	1.66	3.63	1.06	3.88	1.07	3.94	5.39	0.62
2	0.51	1.49	1.48	3.45	0.88	3.70	0.88	3.75	5.21	0.43
3	0.36	1.34	1.33	3.30	0.73	3.55	0.73	3.60	5.06	0.28
4	0.66	1.64	1.63	3.60	1.03	3.85	1.03	3.91	5.36	0.59
5	0.44	0.54	0.53	2.50	0.07	2.75	0.07	2.80	4.26	0.52
6	0.94	0.04	0.03	2.00	0.57	2.25	0.57	2.30	3.76	1.01
7	0.50	1.48	1.46	3.44	0.86	3.69	0.87	3.74	5.20	0.42
8	0.00	0.98	0.97	2.94	0.37	3.19	0.37	3.24	4.70	0.08
9	0.11	1.09	1.08	3.05	0.48	3.30	0.49	3.36	4.81	0.04
10	0.34	1.32	1.31	3.28	0.71	3.53	0.71	3.59	5.04	0.27

Table 8.4: Affinity Difference matrix. AffinityDiff(i, j) represents the absolute binding affinity difference between test molecule i and learn molecule j.

Interpretation:

- GED Matrix:
  - Each value GED(i, j) represents the structural difference between test molecule i and learn molecule j.

- Affinity Difference Matrix:
  - Each value AffinityDiff(i, j) is the absolute difference in binding affinities between test molecule i and learn molecule j.

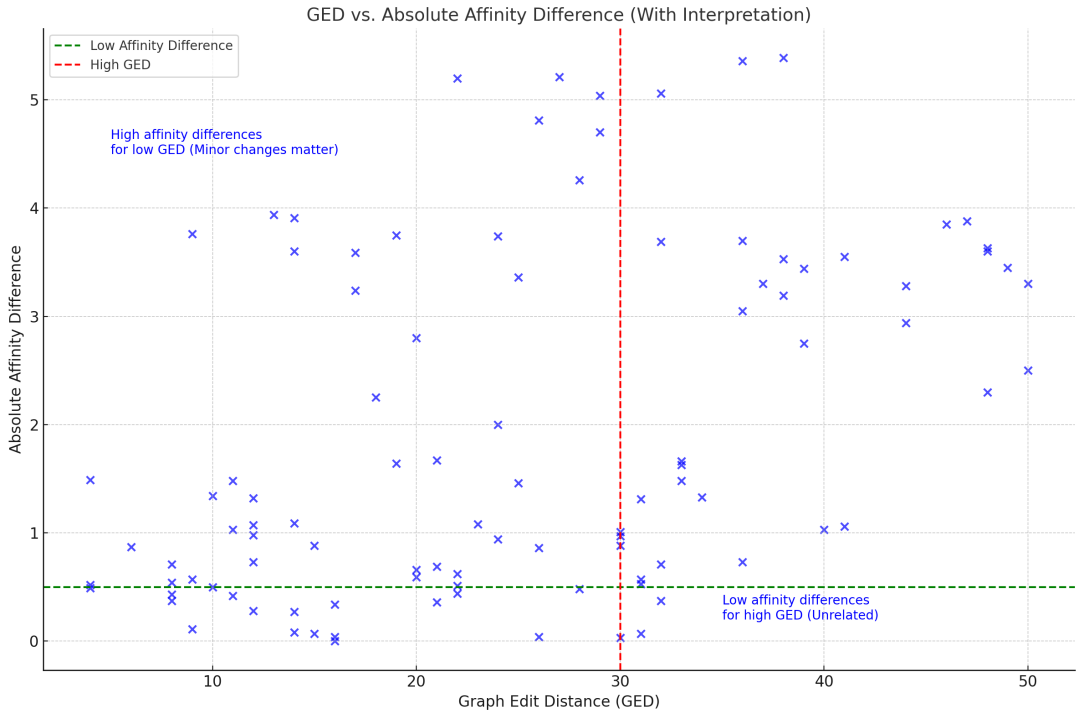

Figure 8.3: GED vs. Absolute Affinity Difference with Interpretation. Points near the x-axis for high GED values may indicate unrelated molecules with significant structural differences, while points far from the x-axis for low GED values suggest minor structural changes significantly affect binding affinity.

- Analyzing Results:
  - Points close to the x-axis (low affinity differences) for higher GED values might indicate unrelated molecules with significant structural differences.
  - Points far from the x-axis (high affinity differences) for low GED values might suggest minor structural changes significantly affect binding affinity.

8.2 Comparison to Other Methods

The GED-KNN approach is compared to deep learning-based methods, highlighting its interpretability and efficiency.

`nn.py`

- Combines KNN and neural network models for affinity prediction.
- Compares prediction performance and visualizes the results.

8.3 Results and Discussion

This section summarizes the prediction of molecular affinities using Mordred descriptors and evaluates the performance of K-Nearest Neighbors (KNN) and feedforward Neural Network (NN) models.

1. Data Preprocessing Ligand structure files (`.sdf`) and affinity data are read. Mordred descriptors are computed and scaled using MinMaxScaler. The dataset is split into 75% training and 25% testing subsets.

2. KNN Model The KNN model predicts affinities, evaluated using the Mean Square Error (MSE). Predictions are compared to true values.

3. Feedforward Neural Network A neural network with three hidden layers predicts affinities. The model architecture includes:

- Input layer for Mordred descriptors.
- Three hidden layers with activation functions.
- Output layer for affinity prediction.

Training loss convergence is monitored through loss curves.

4. Performance Comparison Both models are evaluated using MSE. Results focus on model accuracy and visual comparison of predicted and true affinities.

5. Summary of Results The KNN model provides baseline predictions, while the Neural Network demonstrates improved accuracy, as shown by the MSE values and loss curves.

8.3.1 Neural Network Training Loss Curve

The training loss decreases quickly initially and stabilizes at a low value, indicating the model is learning effectively. The validation loss fluctuates but remains relatively close to the training loss, suggesting the model generalizes well but might experience minor overfitting or noise.

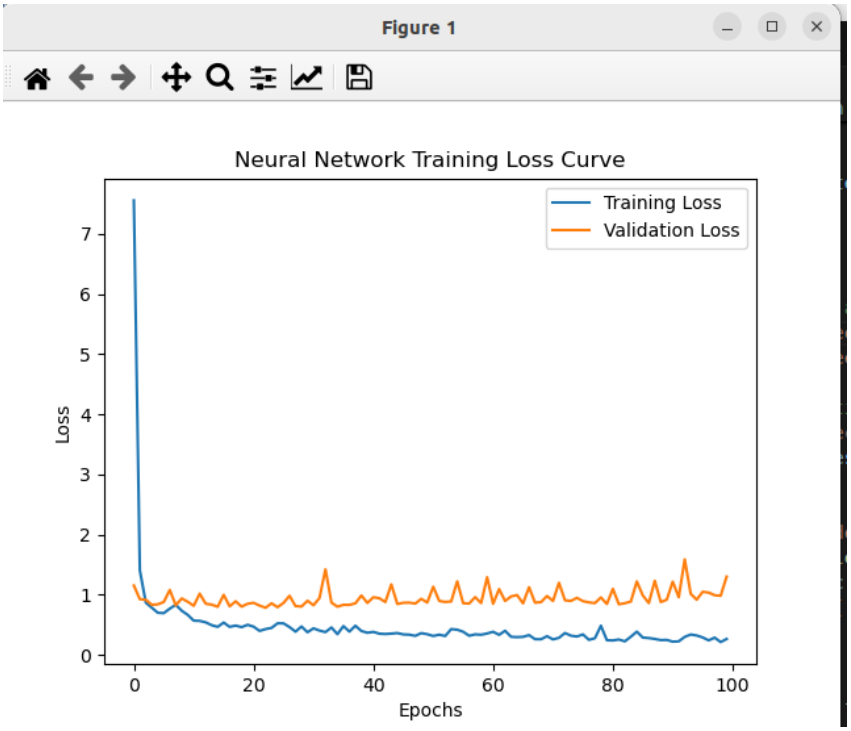

Figure 8.4: Neural Network Training Loss Curve

8.3.2 KNN vs Neural Network Predictions

The plot compares the **true affinities** (green line) with **KNN predictions** (blue line) and **Neural Network predictions** (red line). Both models closely follow the true affinity trend, but the Neural Network predictions appear smoother, while KNN predictions fluctuate more. Some deviations exist, especially for compounds with outlier affinity values, suggesting areas for further model tuning.

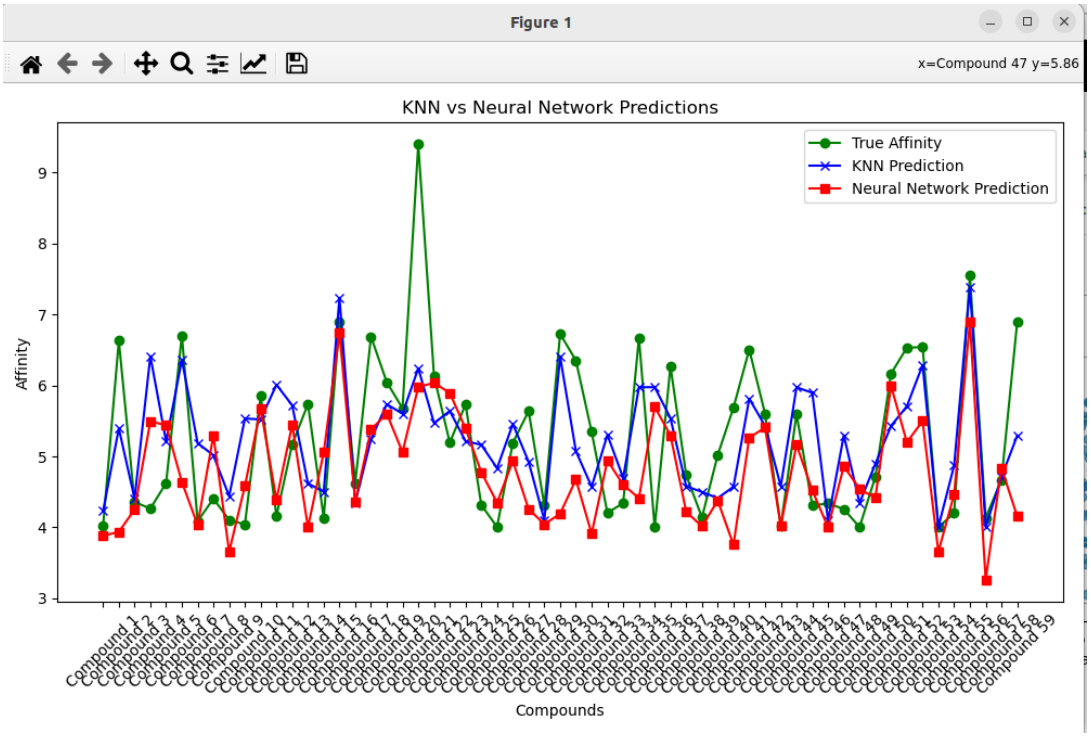

Figure 8.5: KNN vs Neural Network Predictions

8.3.3 Overall Performance

Overall, both models perform well, with the Neural Network providing slightly smoother predictions. Fine-tuning hyperparameters or adjusting the data preprocessing (e.g., descriptor scaling) could further improve results.

8.4 Limitations

The following limitations were identified in the current approach:

- **Sensitivity to parameter selection in GED:** The performance of Graph Edit Distance (GED) is highly sensitive to the choice of substitution, insertion, and deletion costs. Incorrect parameter tuning can significantly affect similarity calculations and subsequent predictions.
- **Descriptor-based methods may overlook 3D structural information:** While molecular descriptors capture key chemical properties, they primarily focus on 2D representations of molecules. This can result in the loss of important 3D spatial information critical for predicting binding affinity.
- **Computational complexity of GED:** The computation of GED, especially for large molecular graphs, is computationally expensive. This limits its scalability for larger datasets or more complex molecules.
- **Limited generalizability of models:** Models trained on a specific dataset may not generalize well to other datasets due to differences in molecular structures, descriptor distributions, or binding properties.

- **Dependence on high-quality input data:** The accuracy of predictions heavily relies on the quality of input data, including correctly formatted ligand files and reliable affinity measurements. Errors in input data can propagate through the entire pipeline.
- **Simplified molecular representations:** The use of simplified molecular graphs or descriptors may not fully capture dynamic molecular behavior, such as conformational changes or interactions in a biological environment.
- **Limited interpretability of neural network models:** Although neural networks achieve high accuracy, they often lack interpretability, making it difficult to explain how predictions are generated or identify key features driving the results.

Chapter 9

Conclusion and Future Work

This work presents a comprehensive approach to molecular affinity prediction by combining **structural comparison**, **machine learning techniques**, and **visualization tools**. The primary objective was to evaluate whether structural analysis of ligands is sufficient to predict their biological activity, specifically the **pIC50 values**.

We compared two structural analysis methods: **Graph Edit Distance (GED)** and **Mordred descriptor-based techniques**. The predictive performance of these methods was analyzed using **K-Nearest Neighbors (KNN)**, while a **neural network (NN) model** integrated with Mordred descriptors served as the reference benchmark (ground truth).

Key Findings

- **GED vs. Affinity**: The relationship between GED and affinity differences revealed a weak positive correlation (Pearson correlation coefficient: 0.3106), suggesting that structural similarity alone is not always sufficient to predict affinity variations accurately. Additional factors likely influence the ligand-activity relationship.
- **Mordred Descriptors**: Descriptor-based methods, particularly when integrated with neural networks, demonstrated superior predictive performance compared to GED-based approaches.
- **Benchmark Performance**: The **Mordred + NN** model served as an effective reference standard, offering improved accuracy and highlighting the potential of descriptor-based machine learning methods.

Novel Contribution

This thesis introduces a **novel method** that utilizes a **real-time structural comparison editor** for predicting **protein-ligand binding affinity** using structural comparisons and KNN. By integrating GED and molecular descriptors, this method provides interpretable insights into the underlying binding mechanisms. Specifically, the tool enables:

- Real-time interaction and editing of protein and ligand structures.
- Integration of machine learning models (GED-KNN, Mordred + NN) for real-time affinity prediction and comparison.

- Dynamic visualization of structural changes and their effects on biological activity.

Future Work

Future directions for this work include:

- Exploring **Graph Neural Networks (GNNs)** to enhance predictive accuracy by leveraging structural graph-based learning.
- Expanding the dataset to include **other protein targets** to validate and generalize the proposed approach across a broader range of biological systems.

In conclusion, while structural comparison methods like GED provide valuable insights, integrating advanced machine learning models with real-time tools opens new opportunities for accurate, interpretable, and efficient affinity prediction. This thesis lays the groundwork for further innovations in **computational drug discovery** and **protein-ligand interaction studies**.

Chapter 10

References & Bibliography

Bibliography

[1] Algabli, S. and Serratosa, F. (2018a). Embedding the node-to-node mappings to learn the graph edit distance parameters. *Pattern Recognition Letters*, 112:353–360.

[2] Algabli, S. and Serratosa, F. (2018b). Embedding the node-to-node mappings to learn the graph edit distance parameters. *Pattern Recognit. Lett.*, 112:353–360.

[3] Arias, J. F., Lai, C. P., Chandran, S., Kasturi, R., and Chhabra, A. K. (1995). Interpretation of Telephone System Manhole Drawings. *Pattern Recognition Letters*, 16(4):365–368.

[4] Arroyo, E., Hoernicke, M., Rodríguez, P., and Fay, A. (2016). Automatic derivation of qualitative plant simulation models from legacy piping and instrumentation diagrams. *Computers and Chemical Engineering*, 92:112–132.

[5] Bajorath, J. (2001). Selected concepts and investigations in compound classification, molecular descriptor analysis, and virtual screening. *J. Chem. Inf. Comput. Sci.*, 41(2):233–245.

[6] Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

[7] Barker, E. J., Buttar, D., Cosgrove, D. A., Gardiner, E. J., Kitts, P., Willett, P., and Gillet, V. J. (2006). Scaffold hopping using clique detection applied to reduced graphs. *J. Chem. Inf. Model.*, 46(2):503–511.

[8] Barker, E. J., Gardiner, E. J., Gillet, V. J., Kitts, P., and Morris, J. (2003). Further development of reduced graphs for identifying bioactive compounds. *J. Chem. Inf. Comput. Sci.*, 43(2):346–356.

[9] Barnard, J. M. (1993). Substructure searching methods: Old and new. *J. Chem. Inf. Comput. Sci.*, 33(4):532–538.

[10] Bender, A. and Glen, R. C. (2004). Molecular similarity: a key technique in molecular informatics. *Org. Biomol. Chem.*, 2(22):3204–3218.

[11] Beno, B. R. and Mason, J. S. (2001). The design of combinatorial libraries using properties and 3D pharmacophore fingerprints. *Drug Discovery Today*, 6(5):251–258.

[12] Birchall, K., Gillet, V. J., Harper, G., and Pickett, S. D. (2006). Training similarity measures for specific activities: application to reduced graphs. *J. Chem. Inf. Model.*, 46(2):577–586.

[13] Blumenthal, D. B. and Gamper, J. (2018). On the exact computation of the graph edit distance. *Pattern Recognit. Lett.*, 0(0):1–12.

[14] Boselli, R., Cesarini, M., Mercorio, F., and Mezzanzanica, M. (2018). Classifying online job advertisements through machine learning. *Future Generation Computer Systems*.

[15] Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recogn. Lett.*, 1(4):245–253.

[16] Caelli, T. and Kosinov, S. (2004). An eigenspace projection clustering method for inexact graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4):515–519.

[17] Caetano, T. S., McAuley, J. J., Cheng, L., Le, Q. V., and Smola, A. J. (2009). Learning graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(6):1048–1058.

[18] Calvo-Zaragoza, J., Castellanos, F., Vigliensoni, G., and Fujinaga, I. (2018). Deep Neural Networks for Document Processing of Music Score Images. *Applied Sciences*, 8(5):654.

[19] Cao, R. and Tan, C. L. (2002). Text/Graphics Separation in Maps. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 167–177.

[20] Cereto-Massagué, A., Ojeda, M. J., Valls, C., Mulero, M., Garcia-Vallvé, S., and Pujadas, G. (2015). Molecular fingerprint similarity search in virtual screening. *Methods*, 71:58–63.

[21] Chaudhuri, A., Mandaviya, K., Badelia, P., and Ghosh, S. K. (2017). Optical character recognition systems. In *Optical Character Recognition Systems for Different Languages with Soft Computing*, pages 9–41. Springer.

[22] Conte, D., Foggia, P., Sansone, C., and Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(03):265–298.

[23] Conte, D. and Serratosa, F. (2020a). Interactive online learning for graph matching using active strategies. *Knowledge-Based Systems*, 205:106275.

[24] Conte, D. and Serratosa, F. (2020b). Interactive online learning for graph matching using active strategies. *Knowl. Based Syst.*, 205:106275.

[25] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

[26] Cortés, X., Conte, D., and Cardot, H. (2019). Learning edit cost estimation models for graph edit distance. *Pattern Recognition Letters*, 125:256–263.

[27] Cortés, X., Conte, D., Cardot, H., and Serratosa, F. (2018). A deep neural network architecture to estimate node assignment costs for the graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 326–336. Springer.

[28] Cortés, X. and Serratosa, F. (2015a). An interactive method for the image alignment problem based on partially supervised correspondence. *Expert Systems with Applications*, 42(1):179–192.

[29] Cortés, X. and Serratosa, F. (2015b). Learning graph-matching edit-costs based on the optimality of the oracle's node correspondences. *Pattern Recognition Letters*, 56:22–29.

[30] Cortés, X. and Serratosa, F. (2016a). Cooperative pose estimation of a fleet of robots based on interactive points alignment. *Expert Systems with Applications*, 45:150–160.

[31] Cortés, X. and Serratosa, F. (2016b). Learning graph matching substitution weights based on the ground truth node correspondence. *International Journal of Pattern Recognition and Artificial Intelligence*, 30(02):1650005.

[32] Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3837–3845.

[33] Duin, R. P. W. and Pekalska, E. (2011). The dissimilarity representation for structural pattern recognition. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, volume 7042 of *Lecture Notes in Computer Science*, pages 1–24. Springer.

[34] Elyan, E., Moreno-García, C. F., and Jayne, C. (2018). Symbols classification in engineering drawings. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*.

[35] Fan, W., Wang, X., and Wu, Y. (2013). Diversified top-k graph pattern matching. *Proceedings of the VLDB Endowment*, 6(13):1510–1521.

[36] Ferrer, M., Serratosa, F., and Riesen, K. (2015). Improving bipartite graph matching by assessing the assignment confidence. *Pattern Recognition Letters*, 65:29–36.

[37] Fisanick, W., Lipkus, A. H., and Rusinko, A. (1994). Similarity searching on CAS registry substances. II. 2D structural similarity. *Journal of Chemical Information and Computer Sciences*, 34(1):130–140.

[38] Foggia, P., Percannella, G., and Vento, M. (2014). Graph matching and learning in pattern recognition in the last 10 years. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(01):1450001.

[39] Gao, X., Xiao, B., Tao, D., and Li, X. (2010a). A survey of graph edit distance. *Pattern Analysis and Applications*, 13(1):113–129.

[40] García-Hernández, C., Fernández, A., and Serratosa, F. (2019). Ligand-based virtual screening using graph edit distance as molecular similarity measure. *Journal of Chemical Information and Modeling*, 59(4):1410–1421.

[41] García-Hernández, C., Fernández, A., and Serratosa, F. (2020). Learning the edit costs of graph edit distance applied to ligand-based virtual screening. *Current Topics in Medicinal Chemistry*, 20(18):1582–1592.

[42] Garey, M. R. and Johnson, D. S. (1990). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York.

[43] Gatica, E. A. and Cavasotto, C. N. (2011). Ligand and decoy sets for docking to G protein-coupled receptors. *Journal of Chemical Information and Modeling*, 52(1):1–6.

[44] Gellaboina, M. K. and Venkoparao, V. G. (2009). Graphic symbol recognition using auto associative neural network model. In *Proceedings of the 7th International Conference on Advances in Pattern Recognition (ICAPR)*, pages 297–301.

[45] Gibert, J., Valveny, E., and Bunke, H. (2012). Graph embedding in vector spaces by node attribute statistics. *Pattern Recognition*, 45(9):3072–3083.

[46] Gillet, V. J., Downs, G. M., Holliday, J. D., Lynch, M. F., and Dethlefsen, W. (1991). Computer storage and retrieval of generic chemical structures in patents. XIII. reduced graph generation. *Journal of Chemical Information and Computer Sciences*, 31(2):260–270.

[47] Gillet, V. J., Willett, P., and Bradshaw, J. (2003). Similarity searching using reduced graphs. *Journal of Chemical Information and Computer Sciences*, 43(2):338–345.

[48] Gou, G. and Chirkova, R. (2008). Efficient algorithms for exact ranked twig-pattern matching over graphs. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 581–594. ACM.

[49] Habi, A., Effantin, B., and Kheddouci, H. (2019). Diversified top-k search with relaxed graph simulation. *Social Network Analysis and Mining*, 9:1–15.

[50] Heikamp, K. and Bajorath, J. (2013). The future of virtual compound screening. *ChemBioDrugDesign*, 81(1):33–40.

[51] Howie, C., Binford, T., Chen, J., Kunz, J., and Law, K. H. (1995). Computer interpretation of process and instrumentation drawings (progress report). Technical report.

[52] Howie, C., Kunz, J., Binford, T., Chen, T., and Law, K. H. (1998). Computer interpretation of process and instrumentation drawings. *Advances in Engineering Software*, 29(7-9):563–570.

[53] James, C. and Weininger, D. (1995). Daylight, 4.41 theory manual. Daylight Chemical Information Systems Inc., Irvine, CA, USA.

[54] Johnson, M. A. and Maggiora, G. M. (1990). *Concepts and Applications of Molecular Similarity*. Wiley.

[55] Justice, D. and Hero III, A. O. (2006). A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214.

[56] Kang, S.-O., Lee, E.-B., and Baek, H.-K. (2019). A digitization and conversion tool for imaged drawings to intelligent piping and instrumentation diagrams (P&ID). *Energies*, 12(2593):1–26.

[57] Khan, A., Li, N., Yan, X., Guan, Z., Chakraborty, S., and Tao, S. (2011). Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, pages 901–912.

[58] Kirchmair, J., Distinto, S., Markt, P., Schuster, D., Spitzer, G. M., Liedl, K. R., and Wolber, G. (2009). How to optimize shape-based virtual screening: choosing the right query and including chemical information. *Journal of Chemical Information and Modeling*, 49(3):678–692.

[59] Lagarde, N., Ben Nasr, N., Jeremie, A., Guillemain, H., Laville, V., Labib, T., Zagury, J.-F., and Montes, M. (2014). NRLiSt BDB, the manually curated nuclear receptors ligands and structures benchmarking database. *Journal of Medicinal Chemistry*, 57(7):3117–3125.

[60] Lajiness, M. (1990). Molecular similarity-based methods for selecting compounds for screening. In *Computational Chemical Graph Theory*, pages 299–316. Nova Science Publishers, Inc.

[61] Livingstone, D. J. (2000). The characterization of chemical structures using molecular properties. A survey. *Journal of Chemical Information and Computer Sciences*, 40(2):195–209.

[62] Luqman, M. M., Ramel, J.-Y., Lladós, J., and Brouard, T. (2013). Fuzzy multilevel graph embedding. *Pattern Recognition*, 46(2):551–565.

[63] Martineau, M., Raveaux, R., Conte, D., and Venturini, G. (2018). Learning error-correcting graph matching with a multiclass neural network. *Pattern Recognition Letters*.

[64] McGregor, M. J. and Pallai, P. V. (1997). Clustering of large databases of compounds: using the MDL "keys" as structural descriptors. *Journal of Chemical Information and Computer Sciences*, 37(3):443–448.

[65] Menard, P. R., Mason, J. S., Morize, I., and Bauerschmidt, S. (1998). Chemistry space metrics in diversity analysis, library design, and compound selection. *Journal of Chemical Information and Computer Sciences*, 38(6):1204–1213.

[66] Moreno, C. and Serratosa, F. (2015). Consensus of multiple correspondences to increase the accuracy in image registration. *Computer Vision and Image Understanding*.

[67] Moreno-García, C. F. (2018). Digital interpretation of sensor-equipment diagrams. In *Proceedings of the SICSA Workshop on Reasoning, Learning and Explainability (ReaLX 2018)*.

[68] Moreno-García, C. F., Cortés, X., and Serratosa, F. (2016). A graph repository for learning error-tolerant graph matching. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 519–529. Springer.

[69] Moreno-García, C. F. and Elyan, E. (2019). Digitisation of assets from the oil & gas industry: challenges and opportunities. In *International Conference on Document Analysis and Recognition (ICDAR)*, Workshop on Industrial Applications of Document Analysis and Recognition (WIADAR), pages 16–19.

[70] Moreno-García, C. F., Elyan, E., and Jayne, C. (2017). Heuristics-based detection to improve text/graphics segmentation in complex engineering drawings. In *Engineering Applications of Neural Networks*, volume CCIS 744, pages 87–98.

[71] Moreno-García, C. F., Elyan, E., and Jayne, C. (2019). New trends on digitisation of complex engineering drawings. *Neural Computing and Applications*, 31(6):1695–1712.

[72] Munkres, J. (1957). Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38.

[73] Mysinger, M. M., Carchia, M., Irwin, J. J., and Shoichet, B. K. (2012). Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking. *Journal of Medicinal Chemistry*, 55(14):6582–6594.

[74] Neuhaus, M. and Bunke, H. (2005). Self-organizing maps for learning the edit costs in graph matching. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):503–514.

[75] Neuhaus, M. and Bunke, H. (2007). Automatic learning of cost functions for graph edit distance. *Information Sciences*, 177(1):239–247.

[76] Neuhaus, M., Riesen, K., and Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 163–172. Springer.

[77] Nikolova, N. and Jaworska, J. (2003). Approaches to measure chemical similarity—a review. *Molecular Informatics*, 22(9-10):1006–1026.

[78] Pearlman, R. S. and Smith, K. M. (1999). Metric validation and the receptor-relevant subspace concept. *Journal of Chemical Information and Computer Sciences*, 39(1):28–35.

[79] Rahul, R., Paliwal, S., Sharma, M., and Vig, L. (2019). Automatic information extraction from piping and instrumentation diagrams. In *International Conference on Pattern Recognition Applications and Methods (ICPRAM)*, pages 163–172.

[80] Rantala, M., Niemistö, H., Karhela, T., Sierla, S., and Vyatkin, V. (2019). Applying graph matching techniques to enhance reuse of plant design information. *Computers in Industry*, 107:81–98.

[81] Rarey, M. and Dixon, J. S. (1998). Feature trees: a new molecular similarity measure based on tree matching. *Journal of Computer-Aided Molecular Design*, 12(5):471–490.

[82] Rica, E., Álvarez, S., and Serratosa, F. (2019). On-line learning the edit costs based on an embedded model. In *International Workshop on Graph-Based Representations in Pattern Recognition*, pages 121–130. Springer.

[83] Rica, E., Álvarez, S., and Serratosa, F. (2021). On-line learning the graph edit distance costs. *Pattern Recognition Letters*, 146:55–62.

[84] Rica, E., Moreno-García, C. F., Álvarez, S., and Serratosa, F. (2020). Reducing human effort in engineering drawing validation. *Computers in Industry*, 117:103198.

[85] Riesen, K. and Bunke, H. (2009). Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision Computing*, 27(7):950–959.

[86] Riesen, K., Fischer, A., and Bunke, H. (2018). On the impact of using utilities rather than costs for graph matching. *Neural Processing Letters*, 48(2):691–707.

[87] Rohrer, S. G. and Baumann, K. (2009). Maximum unbiased validation (MUV) data sets for virtual screening based on PubChem bioactivity data. *Journal of Chemical Information and Modeling*, 49(2):169–184.

[88] Sanders, M. P., Barbosa, A. J., Zarzycka, B., Nicolaes, G. A., Klomp, J. P., de Vlieg, J., and Del Rio, A. (2012). Comparative analysis of pharmacophore screening tools. *Journal of Chemical Information and Modeling*, 52(6):1607–1620.

[89] Sanfeliu, A., Alquézar, R., Andrade, J., Climent, J., Serratosa, F., and Vergés-Llahí, J. (2002). Graph-based representations and techniques for image processing and image analysis. *Pattern Recognition*, 35(3):639–650.

[90] Sanfeliu, A. and Fu, K.-S. (1983a). A distance measure between attributed relational graphs for pattern recognition. *Systems, Man and Cybernetics, IEEE Transactions on, SMC-13.IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362.

[91] Sanfeliu, A. and Fu, K.-S. (1983b). A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(3):353–362.

[92] Santacruz, P. and Serratosa, F. (2018a). Error-tolerant graph matching in linear computational cost using an initial small partial matching. *Pattern Recognition Letters*, 116:141–147.

[93] Santacruz, P. and Serratosa, F. (2018b). Learning the sub-optimal graph edit distance edit costs based on an embedded model. In *Structural, Syntactic, and Statistical Pattern Recognition (S+SSPR)*, volume 11004 of *Lecture Notes in Computer Science*, pages 282–292. Springer.

[94] Santacruz, P. and Serratosa, F. (2019). Learning the graph edit costs based on a learning model applied to sub-optimal graph matching. *Neural Processing Letters*, 50(2):1573–773X.

[95] Santacruz, P. and Serratosa, F. (2020). Learning the graph edit costs based on a learning model applied to sub-optimal graph matching. *Neural Processing Letters*, 51(1):881–904.

[96] Schneider, G., Clément-Chomienne, O., Hilfiger, L., Schneider, P., Kirsch, S., Böhm, H.-J., and Neidhart, W. (2000). Virtual screening for bioactive molecules by evolutionary de novo design. *Angewandte Chemie International Edition*, 39(22):4130–4133.

[97] Schnur, D. (1999). Design and diversity analysis of large combinatorial libraries using cell-based methods. *Journal of Chemical Information and Computer Sciences*, 39(1):36–45.

[98] Serratosa, F. (2014a). Fast computation of bipartite graph matching. *Pattern Recognition Letters*, 45:244–250.

[99] Serratosa, F. (2014b). Speeding up fast bipartite graph matching through a new cost matrix. *International Journal of Pattern Recognition and Artificial Intelligence*, 29(2):1550010.

[100] Serratosa, F. (2015). Computation of graph edit distance: reasoning about optimality and speed-up. *Image and Vision Computing*, 40:38–48.

[101] Serratosa, F. (2021). Redefining the graph edit distance. *SN Computer Science*, 2(6):1–7.

[102] Serratosa, F. and Cortés, X. (2015a). Graph edit distance. *Pattern Recognition Letters*, 65:204–210.

[103] Serratosa, F. and Cortés, X. (2015b). Graph edit distance: moving from global to local structure to solve the graph-matching problem. *Pattern Recognition Letters*, 65:204–210.

[104] Skoda, P. and Hoksza, D. (2017). Benchmarking platform for ligand-based virtual screening. In *Proceedings of the 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 1220–1227.

[105] Solé, A., Serratosa, F., and Sanfeliu, A. (2012). On the graph edit distance cost: properties and applications. *International Journal of Pattern Recognition and Artificial Intelligence*, 26(5):1260004.

[106] Stauffer, M., Tschachtli, T., Fischer, A., and Riesen, K. (2017). A survey on applications of bipartite graph edit distance. In *Graph-Based Representations in Pattern Recognition (GbRPR)*, volume 10310 of *Lecture Notes in Computer Science*, pages 242–252. Springer.

[107] Stephen, I. (1990). Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks*, 1(2):179–191.

[108] Stiefl, N., Watson, I. A., Baumann, K., and Zaliani, A. (2006). ERG: 2D pharmacophore descriptions for scaffold hopping. *Journal of Chemical Information and Modeling*, 46(1):208–220.

[109] Sun, H. (2008). Pharmacophore-based virtual screening. *Current Medicinal Chemistry*, 15(10):1018–1024.

[110] Takahashi, Y., Sukekawa, M., and Sasaki, S. (1992). Automatic identification of molecular similarity using reduced-graph representation of chemical structure. *Journal of Chemical Information and Modeling*, 32(6):639–643.

[111] Tan, W. C., Chen, I. M., and Tan, H. K. (2016). Automated identification of components in raster piping and instrumentation diagram with minimal preprocessing. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1301–1306.

[112] Tombre, K., Tabbone, S., Lamiroy, B., and Dosch, P. (2002). Text/graphics separation revisited. In *Document Analysis Systems*, volume 2423 of *Lecture Notes in Computer Science*, pages 200–211. Springer.

[113] Vaxiviere, P. and Tombre, K. (1992). CELESSTIN: CAD conversion of mechanical drawings. *IEEE Computer*, 25(7):46–54.

[114] Vento, M. (2013). A one hour trip in the world of graphs, looking at the papers of the last ten years. In *Graph-Based Representations in Pattern Recognition*, pages 1–10. Springer.

[115] Vento, M. (2015). A long trip in the charming world of graphs for pattern recognition. *Pattern Recognition*, 48(2):291–301.

[116] Willett, P. (1987). *Similarity and Clustering in Chemical Information Systems*. John Wiley & Sons.

[117] Willett, P. (2004). Evaluation of molecular similarity and molecular diversity methods using biological activity data. In *Chemoinformatics*, pages 51–63. Springer.

[118] Xanthopoulos, P., Pardalos, P. M., and Trafalis, T. B. (2013). Linear discriminant analysis. In *Robust Data Mining*, pages 27–33. Springer.

[119] Xia, J., Tilahun, E. L., Reid, T.-E., Zhang, L., and Wang, X. S. (2015). Benchmarking methods and data sets for ligand enrichment assessment in virtual screening. *Methods*, 71:146–157.

[120] Xue, L. and Bajorath, J. (2000). Molecular descriptors in chemoinformatics, computational combinatorial chemistry, and virtual screening. *Combinatorial Chemistry & High Throughput Screening*, 3(5):363–372.

[121] Yang, Z., Fu, A. W.-C., and Liu, R. (2016). Diversified top-k subgraph querying in a large graph. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1167–1182. ACM.

[122] Yu, Y., Samal, A., and Seth, S. C. (1997). A system for recognizing a large class of engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):868–890.

[123] Zhang, X., Yang, T., and Srinivasan, P. (2016). Online asymmetric active learning with imbalanced data. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 2055–2064.

[124] Zhao, P. and Hoi, S. C. (2013). Cost-sensitive online active learning with application to malicious URL detection. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 919–927.

[125] Zou, L., Chen, L., and Lu, Y. (2007). Top-k subgraph matching query in a large graph. In *Proceedings of the First Ph.D. Workshop in CIKM (PIKM 2007)*, pages 139–146. ACM.

[126] Huiwen Wong Prediction of protein–ligand binding affinity via deep learning models Briefings in Bioinformatics, Volume 25, Issue 2, March 2024, bbae081, https://doi.org/10.1093/bib/bbae081 https://academic.oup.com/bib/article/25/2/bbae081/7620495

[127] Fadlallah, S., Juli'a, C., Garc´ıa-Vallv´e, S., Pujadas, G., Serratosa, F. (2023). Drug Potency Prediction of SARS-CoV-2 Main Protease Inhibitors Based on a Graph Generative Model. International Journal of Molecular Sciences, 24 (10), 8779. MDPI.