Similarity Measures for Small Molecules

July 1, 2024

Abstract

Computational drug discovery leverages clustering of small molecules on a Cartesian map to develop a large language model (LLM) specifically designed for generating novel molecules based on a vector input of properties. This approach employs molecular fingerprinting techniques to assign similarity metrics, facilitating the clustering of molecules with specific attributes. This paper explores two promising fingerprinting techniques: SELFIES (Self-Referencing Embedded Strings) and network graphs, utilizing similarity metrics such as string edit distance and maximal subgraph isomorphism. We discuss the development of these techniques and examine the benefits and challenges associated with each method.

1 Introduction

1.1 Molecular Fingerprinting

The field of drug discovery is constantly evolving, with researchers seeking new and innovative methods to accelerate the development of life-saving medications. Computational drug discovery offers a promising approach by harnessing the power of artificial intelligence to streamline the process. This paper explores one such avenue: the use of large language models (LLMs) specifically trained for generating novel drug candidates.

Our approach within this domain leverages the power of clustering small molecules within a Cartesian space. By grouping molecules with similar properties, we can train an LLM to recognize these patterns and predict the creation of novel molecules based on a vector input of desired characteristics.

This methodology relies heavily on molecular fingerprinting techniques, which are computerized representations of molecules. These usually contain a unique sequence of bits or tokens that encodes certain properties, structural and chemical, of a molecule. By leveraging molecular fingerprinting techniques, researchers can assign similarity metrics to cluster molecules with specific attributes, streamlining the drug discovery process. This paper delves into two particularly promising fingerprinting methods: SELFIES (Self-Referencing Embedded Strings) and network graphs. We will explore the development of these techniques, analyze the string edit distance and maximal subgraph isomorphism similarity metrics used in conjunction with them, and discuss the distinct benefits and challenges associated with each approach.

1.2 Data Acquisition

String and graph representations of the molecules were obtained through a two-step process. First, bulk downloads of molecule data in .mol file format were retrieved from the ChEMBL database. These .mol files represent the molecular structure in a standardized format.

Next, using Python's RDKit library, the .mol files were converted into SMILES strings. SMILES strings offer a compact, text-based representation of the molecule's structure. However, for this study, we aimed for a string representation with guaranteed bijectivity (one-to-one correspondence between string and molecule). Therefore, we further converted the SMILES strings into SELFIES strings using a custom conversion model. This ensures that every SELFIES string uniquely corresponds to a valid molecule.

For the graph representation, the same .mol files were used to generate adjacency matrices. These matrices capture the connectivity information between atoms in the molecule. Finally, the NetworkX library was employed to convert the adjacency matrices into network planar graphs. In these graphs, nodes represent individual atoms, and edges represent the bonds connecting them. This network representation allows us to leverage graph-based algorithms for further analysis and exploration of the molecular properties.

2 SMILES

2.1 Introduction

The most widely used molecular fingerprinting technique in computational chemistry is SMILES (Simplified Molecular Input Line Entry System), which stores chemical structures as linear text strings. This system encodes the structure of a molecule by mapping its atomic and bond configurations into a sequence of characters. Each atom is denoted by its chemical symbol, and bonds are represented by specific characters. Branches and ring structures are depicted using additional symbols, allowing for the description of complex molecular geometries. Various libraries, such as Python's RDKit, offer SMILES functionality, along with the ability to convert to different molecular representations.

The SMILES format is an extension of the molecular graph representation. The program traverses through the graph to encode branches, rings, and aromaticity using numerical indices, parantheses, and brackets. Additionally, the format highlights non-organic elements through special character codes.

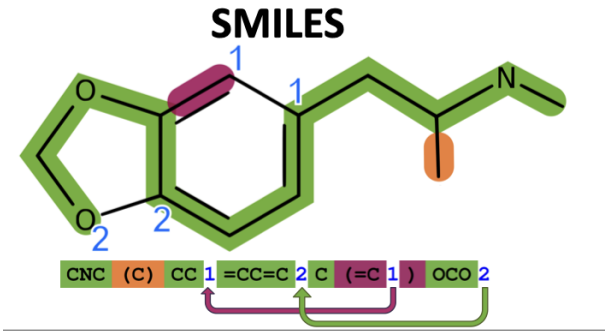


Figure 1: SMILES representation of 3,4-Methylenedioxymethamphetamine, a small organic molecule. The original path is highlighted in green, the ring is highlighted in purple and represented by the arrows, and the additional branch is highlighted in orange. [3]

2.2 Advantages

SMILES offers several benefits that have contributed to its broad adoption in cheminformatics. One of the significant advantages of SMILES is its relative ease of interpretation. Chemists familiar with the notation can quickly read and write SMILES strings, facilitating manual data entry and examination. Unlike other representations such as InChl, SMILES is very easy to transfer into its graphical representation.

Secondly, SMILES provides a concise way to represent molecules, making it efficient

for storage, transmission, and computational processing. This compactness is particularly useful in databases and large-scale cheminformatics applications. Due to this simplicity and efficiency, SMILES has become a standard in many cheminformatics tools and databases, allowing the ability to acquire SMILES values in large databases.

2.3 Limitations

Despite its advantages, SMILES has several notable limitations that can impact its utility in certain applications. SMILES strings are not inherently unique; the same molecule can be represented by multiple valid SMILES strings, thus they lack bijectivity. This lack of a unique representation can complicate data comparison and retrieval processes, as different databases can use different programs to build their SMILES annotations.

However, a more significant issue is its lack of semantic robustness. SMILES strings are extremely sensitive to small errors or changes, meaning they can be problematic in a machine learning setting which focuses on adding small mutations to build new options. Additionally, not all possible SMILES strings correspond to valid chemical structures. This characteristic can result in the generation of invalid molecules during computational tasks, such as molecule generation or random mutations, necessitating additional validation steps. Thus, a large language model cannot be built solely from SMILES representations.

3 SELFIES

3.1 Introduction

SELFIES (Self-Referencing Embedded Strings) is a more recent development in the field of molecular fingerprinting that addresses many of the limitations inherent in SMILES. SELFIES provides a robust, error-tolerant string representation of molecular structures, ensuring that every syntactically correct string corresponds to a valid molecule. This feature makes SELFIES particularly advantageous for various applications in computational chemistry and machine learning. [3]

3.2 Synatic Robustness

SELFIES offers several key benefits that enhance its utility as a molecular fingerprint in a large language model environment. Firstly, SELFIES is a robust syntatic representation, meaning each valid molecular structure has a unique SELFIES representation, thus preserving bijectivity. This greatly simplifies data collection and allows the use of similarity metrics such as string edit distance as each representation is canonical.

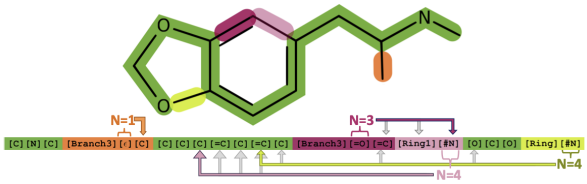


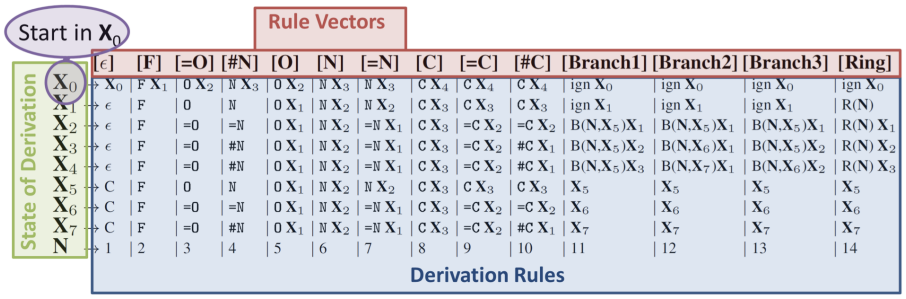
Figure 2: SELFIES representation of 3,4-Methylenedioxymethamphetamine, a small organic molecule. Note the robustness in this representation, contrasting with the previous SMILES equivalent. [3]

Additionally, each substructure in the string is stored in an independent derivation. This allows for more efficient substructure searching as pattern information is stored locally within the string and is not subject to other derivation rules. Similarly, non-local branches and rings are defined throughout the subsections of the string in which they are present. As such, both local and broader operations are stored within subsections of the SELFIES string.

3.3 Chemical Validity

Secondly, unlike SMILES, every SELFIES string is guaranteed to represent a valid chemical structure. While SMILES merely translates a molecular graph into a string, SELFIES enhances this process by applying stringent chemical rules. The primary string is constructed using a rule set that ensures the number of valence bonds per atom does not exceed their physical limits. Additionally, the derivation of each symbol in the SELFIES string is contingent on the state of the derivation X_n, with these rules meticulously designed to enforce the validity of chemical valence bonds. These derivation rules are formally described in Figure 3.

The process of generating a SELFIES string can be understood as a sequence of vector additions. Each SELFIES symbol is interpreted as a rule vector. A SELFIES symbol is replaced by a string determined by the intersection of the rule vector and the derivation state. This string can represent either an atom or another derivation state. The derivation process begins in the state X_0 and proceeds through subsequent states



	[ϵ]	[F]	[=O]	[#N]	[O]	[N]	[=N]	[C]	[=C]	[#C]	[Branch1]	[Branch2]	[Branch3]	[Ring]
X_0	X_0	F X_1	O X_2	N X_3	O X_2	N X_3	N X_3	C X_4	C X_4	C X_4	ign X_0	ign X_0	ign X_0	ign X_0
X_1	ϵ	F	O	N	O X_1	N X_2	N X_2	C X_3	C X_3	C X_3	ign X_1	ign X_1	ign X_1	R(N)
X_2	ϵ	F	=O	=N	O X_1	N X_2	=N X_1	C X_3	=C X_2	=C X_2	B(N,X_5)X_1	B(N,X_5)X_1	B(N,X_5)X_1	R(N) X_1
X_3	ϵ	F	=O	#N	O X_1	N X_2	=N X_1	C X_3	=C X_2	#C X_1	B(N,X_5)X_2	B(N,X_6)X_1	B(N,X_5)X_2	R(N) X_2
X_4	ϵ	F	=O	#N	O X_1	N X_2	=N X_1	C X_3	=C X_2	#C X_1	B(N,X_5)X_3	B(N,X_7)X_1	B(N,X_6)X_2	R(N) X_3
X_5	C	F	O	N	O X_1	N X_2	N X_2	C X_3	C X_3	C X_3	X_5	X_5	X_5	X_5
X_6	C	F	=O	=N	O X_1	N X_2	=N X_1	C X_3	=C X_2	=C X_2	X_6	X_6	X_6	X_6
X_7	C	F	=O	#N	O X_1	N X_2	=N X_1	C X_3	=C X_2	#C X_1	X_7	X_7	X_7	X_7
N	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 3: Derivation rules of SELFIES for small organic molecules [3]

derived from previous steps. These derivation states ensure compliance with syntactical and chemical constraints, such as the maximum number of valence bonds. The rules in state X_n for $n = 1$ to $n = 4$ are designed so that the next atom can use up to n valence bonds. The function $B(N, X_n)$ creates a branch in the graph using the next N symbols and starts in state X_n. Similarly, the function $R(N)$ creates rings, connecting the current atom to the $(N + 1)$-st previously derived atom. In both cases, the letter following R or B is interpreted as a number N, which is defined in the final line of the table. Through this, all valence-bond rules and ring rules will be satisfied. The table can be extended to represent ions, stereochemistry, and larger molecules, thereby broadening the scope of SELFIES. [3]

We can derive an example string, [O][=C][=O] using the rules above. Starting in the state X_0, we can add up the rule vectors in the following manner:

$$X_0 \xrightarrow{[O]} OX_2 \xrightarrow{[=C]} O=CX_2 \xrightarrow{[=O]} O=C=O \tag{1}$$

The final molecule O=C=O, carbon dioxide, satisfies all valence-bond rules. From here, we can insert rings only if the number of valence-bond at the target has not yet reached the maximum. Thereby, using the rules in Figure 3, validity can be guaranteed for small biomolecules.

This property eliminates the risk of generating invalid molecules during computational processes, such as molecule generation or random mutations, which is crucial for reliable automated workflows. Thus, random walks can be conducted on SELFIES strings to generate an LLM model. The generation of ML-built molecules will always be valid, allowing the use of AI models such as genetic algorithms.

3.4 Random Mutations

Another notable property of SELFIES's syntactic robustness is the ability that the addition of random mutations still results in valid chemical structures. This robustness stands in stark contrast to SMILES, which does not support small linear changes effectively. In tools like RDKit, even slight modifications to SMILES strings often lead to invalid representations. However, SELFIES maintains validity even after linear mutations, consistently generating chemically meaningful strings. [3]

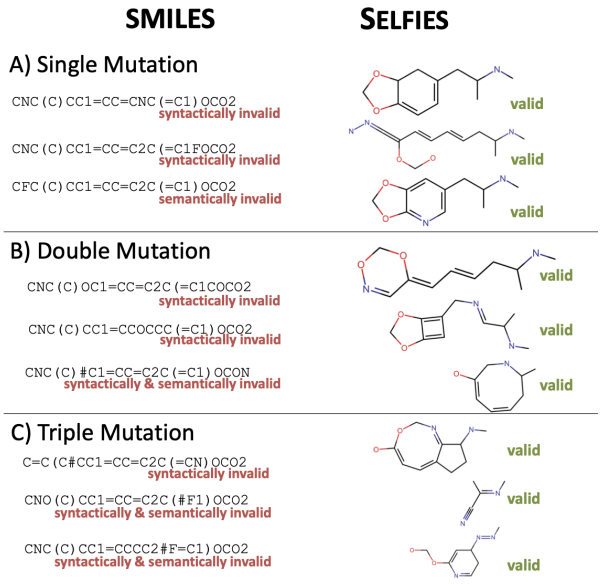


Figure 4: Random Mutations of SELFIES vs SMILES [3]

The ability to incorporate random mutations is significant in the context of large language models (LLMs) and machine learning (ML). In generative models, random mutations allow for the exploration of chemical space, enabling the discovery of novel molecules with potentially valuable properties. Additionally, the syntactic robustness of SELFIES ensures that the generated molecules are always valid, reducing the need for extensive post-generation validation and filtering. This efficiency can accelerate the iterative cycles of training and inference in ML models.

3.5 Feature vs Cost Metrics

There are two main types of similarity measures, *feature-based metrics* and *cost-based metrics*. In feature-based metrics, a set of invariant features are established from a graph which are then vectorized. From here, similarity coefficiencts are obtained by running calculations on these vectorized molecular fingerprints, the most common being the Tanimoto Coefficient.

However, for string representations, we will be running cost-based metrics. In cost-based metrics, the similarity between two molecules is related to the number of edit operations that are required to transform one SELFIES string into the other.

3.6 String Edit Distances

There are numerous edit distance calculations which exist for strings. However, the more common calculations showing potential are the *Hamming distance*, *Levenshtein distance*, and *Jaro similarity*, and *Jaro-Winkler similarity*. Each of these calculations have similar constraints and are built upon each other. For each of these calculations, let $|s_1$ and s_2 be the length of the strings being compared.

Hamming Distance: The most basic edit distance metric is the Hamming distance, which measures the dissimilarity between two strings of the same length by overlaying one string over another and count how many positions have different characters. From here, we can calculate the normalized similarity to get a value between 0 and 1.

Levenshtein Distance: The second edit distance, and the most common, is the Levenshtein distance. This is a string similarity metric which calculates the minimum number of string operations (insertions, substitutions, deletions) required to transform one string into another, providing a measure of how different the two strings are. A variation of this metric, the *Damerau-Levenshtein distance* also includes the transposition operation. From these two distances, we can calculate the normalized similarity to get a value between 0 and 1. Let $lev(s_1, s_2)$ be the Levenshtein distance for two strings s_1 and s_2. Then,

$$lev(a,b) = \begin{cases} a & \text{if } b = 0, \\ |b| & \text{if } a = 0, \\ lev(\text{tail}(a), \text{tail}(b)) & \text{if head}(a) = \text{head}(b), \\ 1 + \min \begin{pmatrix} lev(\text{tail}(a), b) \\ lev(a, \text{tail}(b)) \end{pmatrix} & \text{otherwise.} \end{cases} \tag{2}$$

Jaro-Winkler Similarity: The third edit distance is the Jaro-Winkler similarity metric, based on the Jaro similarity. The Jaro similarity is an algorithm based on the Bamerau-Levenstein distance except that it outputs a similarity score rather than a distance measure. It also removes the constrain that transpositions have to occur within adjacent characters. Additionally, it states that two characters are matching if they are

the same and within $\max(|s_1|, |s_2|)/2 - 1$ characters apart. Let d_j be the Jaro similarity between two strings s_1 and s_2, m be the number of matching characters between them, and t be the number of transpositions that are not in the right order divided by two. Then,

$$d_j = \begin{cases} 0 & \text{if } m = 0, \\ \frac{1}{3}\left(\frac{m}{|s_1} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise,} \end{cases} \tag{3}$$

The Winkler variation slightly changes the similarity by providing additional weight to a common prefix of length ℓ between the two strings. Let d_w be the Jaro-Winkler similarity of two strings s_1 and s_2, and p be a constant scaling factor for how much the score is adjusted upwards for having common prefixes (the traditional value for p is 0.1). Then,

$$d_w = d_j + \ell p(1 - d_j) \tag{4}$$

3.7 Token-based Similarity

While edit distance metrics like Levenshtein distance or Jaro-Winkler similarity offer efficient methods for string comparison, they struggle to capture nuanced chemical differences arising from substituent group variations within a molecule. These metrics primarily focus on individual character-level edits, which may not accurately reflect the semantic and functional significance of substitutions. In this context, token-based similarity techniques offer a more suitable approach for large language models (LLMs) in cheminformatics. Unlike edit distance, token-based methods analyze strings by considering their constituent substructures or functional groups, termed tokens. This approach aligns better with the way LLMs process and understand chemical information, leading to a more accurate representation of chemical similarity that considers the functional impact of substituent groups.The two most common calculations are the *Jaccard similarity* and *n-gram analysis*.

Jaccard Similarity: The simplest token-based algorithm is the Jaccard metric. The Jaccard metric measures similarity between the sets of tokens within the strings, and is defined as the size of the intersection divided by the size of the union of the token sets. Let $J(t_1, t_2)$ be the Jaccard similarity between the set of tokens of two strings. Then,

$$J(s_1, s_2) = \frac{|t_1 \cap t_2|}{|t_1| + |t_2| - |t_1 \cup t_2|} \tag{5}$$

n-gram Analysis: The most common token-based algorithm for LLMs and natural

language processing is n-gram analysis, which measures the similarity between two strings by analyzing their subsequence of n-adjacent characters, called n-grams. The process of n-gram analysis involves the following steps:

n-gram extraction: Divide each string into overlapping sequences of n characters (where n is pre-defined)

n-gram count: Count the occurence of each unique n-gram in each string

Similarity calculation: Calculate the similarity score using the Jaccard similarity coefficient

3.8 Evaluation of Metrics

To compare the various string edit distance metrics, we calculate each metric for the SELFIES representations of terephthalic acid and aminobenzoic acid.

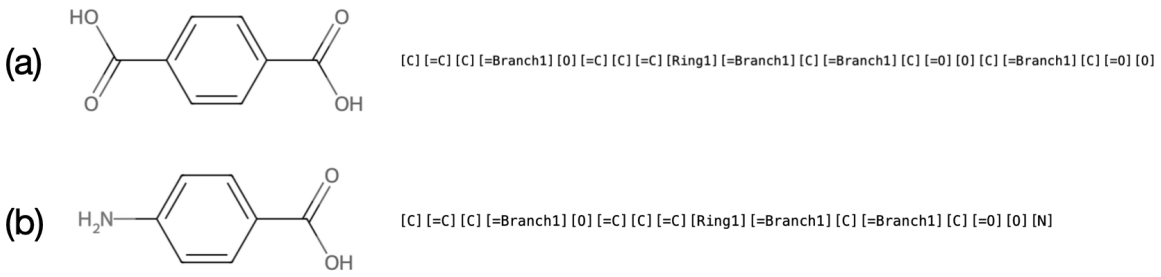


Figure 5: (a) terephthalic acid and its SELFIES string (b) aminobenzoic acid and its SELFIES string

Metric	Value
Hamming Distance	0.783
Levensthein Distance	0.783
Jaro-Winkler Similarity	0.954
Jaccard Similarity	0.875

Table 1: Values of various similarity metrics

4 Graph Fingerprinting

4.1 Molecular Graph Searching

The simplest method of comparing the similarity of two compounds is to compare their molecular graphs. From this, the concept of graph fingerprinting was created. The common approach is to store elements as nodes and bonds as edges. From here, rings and branches can be depicted as well. The immediate inherent advantage is the close resemblance to the actual depiction of the molecule, allowing for more properties and structures to be preserved and compared.

From here, structure searching in cheminformatics encompasses two primary methods: *graph isomorphism search* and *subgraph isomorphism search*. Graph isomorphism identifies exact matches between a query molecule and database entries, facilitating the retrieval of synthetic procedures, spectral properties, and other associated data. However, its computational complexity limits its application to smaller databases. Subgraph isomorphism searching, on the other hand, performs a partial-match search based on a user-defined query fragment. This approach is computationally more efficient and is widely employed in cheminformatics due to its effectiveness in identifying similar molecules relevant for building machine learning models focused on tasks like activity prediction or property estimation. While large language models (LLMs) are not directly involved in substructure searching, they can potentially be trained on the data retrieved through such searches to identify patterns and relationships within chemical space. [2]

4.2 Maximum Common Subgraph

The *maximum common subgraph* is calculated through the isomorphism of two graphs. *Isomorphism* is a property in graph theory related to bijection, indicating a one-to-one correspondence between the vertices of two graphs such that an edge exists between two vertices in one graph if and only if an edge exists between the corresponding vertices in the other graph.

An *induced subgraph* is a subset S of vertices of a graph G along with all the edges of G that have both endpoints in S. For instance, a graph G_{12} is a *common induced subgraph* of graphs G_1 and G_2 if G_{12} is isomorphic to induced subgraphs of both G_1 and G_2. A *maximum common induced subgraph* (MCIS) is defined as a graph G_{12} with the largest number of vertices that meet this isomorphism criterion. Closely related to the MCIS is the *maximum common edge subgraph* (MCES), which is the subgraph consisting of the

largest number of edges common to both G_1 and G_2. It is important to note that the MCIS or MCES between two graphs is not necessarily connected or unique by definition. Figures 5a and 5b provide examples of an MCIS and an MCES, respectively. [5]

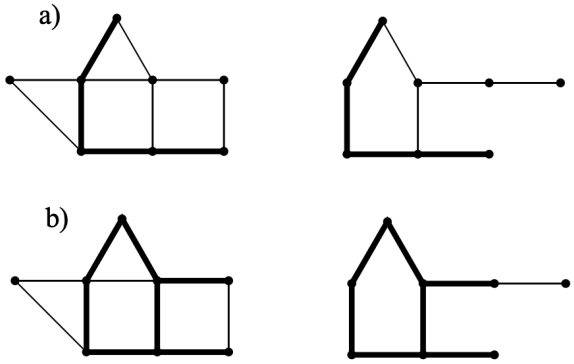


Figure 6: (a) Maximum Common Induced Subgraph (b) Maximum Common Edge Subgraph [4]

The concept of maximum common subgraph (MCS) in cheminformatics can be further distinguished between *connected* and *disconnected* MCS (dMCS). The standard MCS, a connected graph, is one where each vertex is connected to every other vertex by at least one path within the graph, meaning the MCS forms a single subgraph. However, in some cases, a fragmented, yet chemically relevant, common substructure might exist. A disconnected MCS consists of two or more common subgraph components. Figure 6 illustrates the distinction between a connected MCS and a disconnected MCS. [5]

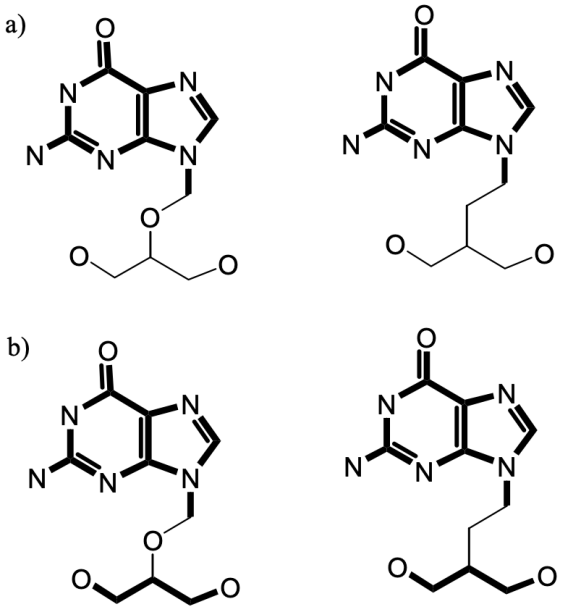


Figure 7: (a) Connected MCS (b) Disconnected MCS [4]

Fundamentally, the Maximum Common Edge Subgraph (MCES) more accurately

represents chemical similarity than the Maximum Common Induced Subgraph (MCIS), as it is the interactions between bonds that primarily determine a molecule's reactivity. Additionally, the significance of a disconnected MCES lies in its ability to capture interactions between different crucial substructures, providing a more comprehensive view of molecular similarities. [4]

4.3 MCS Problem

The similarity metrics studied later in this section are inherently built from calculating the MCS between two graphical representations. However, the fundamental issue is that MCS calculation is NP-complete (i.e. MCS calculation algorithms are exponential-time or worse, a polynomial-time algorithm does not exist yet). For example, a simple comparison between a pair of graphs with x and y nodes, respectively, takes a maximum of

$$\frac{x! \cdot y!}{(x-k)!(y-k!)k!} \tag{6}$$

node-by-node comparisons to determine all common subgraphs of k nodes, an unwieldy number of non-trivial values of x, y, and k. [4]

As such, numerous attempts have been made to devise algorithms for specific subsets of MCS-type graphs. Two important distinctions are between MCIS and MCES, as seen in the next two sections.

4.4 MCIS Calculation

Through numerous efforts, it has been shown that the MCIS calculation can be reduced to the problem of finding the maximum clique in the modular product graph $G_1 \diamond G_2$ for the two compared graphs G_1 and G_2. The modular product $G_1 \diamond G_2$ of two graphs G_1 and G_2 is defined on the vector set $V_(G_1 \diamond G_2) = V(G_1) \cdot V(G_2)$, where $\cdot$ denotes the Cartesian product of two sets. Furthermore, two vertices u_i, v_i and u_j, v_j are adjacent whenever

$$(u_i, u_j) \in E(G_1) \text{and} (v_i, v_j) \in E(G_2) \text{and} w(u_i, u_j) = w(v_i, v_j) \tag{7}$$

or

$$(u_i, u_j) \notin E(G_1) \text{and} (v_i, v_j) \notin E(G_2) \tag{8}$$

where $w(u_i, u_j) = w(v_i, v_j)$ indicates that the vertex and edge labels for each respective pair of vertices are compatible. This process is denoted in Figure 9. [6]

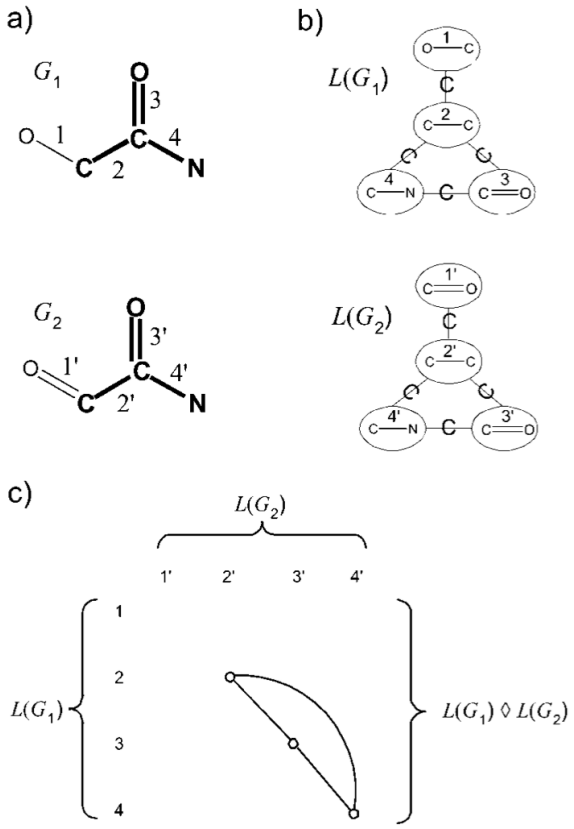


Figure 8: The modular product of two graphs G_1 and G_2 [6]

A *clique* is a subset of vertices in a graph G such that each pair of vertices in the subset is connected by an edge in the graph G. The *maximum clique* is the largest subset present in the graph G. The MCIS calculation can be converted into the maximum clique problem by constructing a modular product of the two graphs G_1 and G_2. [4] While the maximum clique problem is also NP-complete, it has the advantage of being compatible with advanced clique-based algorithms, such as the MCP algorithm introduced by Carraghan and Pardalos. Further methods being researched include backtracking algorithms and quantum-based algorithms.

4.5 MCES Calculation

The other advantage of converting the maximum common subgraph problem to the maximum clique problem in the modular product graph is the ease of conversion to the maximum common edge subgraph problem through the ΔY *exchange test*.

A ΔY exchange, denoted in Figure 10, is where the line graphs $L(G_1)$ and $L(G_2$ of the two graphs being compared are isomorphic, even though the original graphs G_1 and

G_2 are not. The modular product $G_1 \diamond G_2$ of two graphs G_1 and G_2 is defined on the vector set $V_(G_1 \diamond G_2) = V(G_1) \cdot V(G_2)$, where $\cdot$ denotes the Cartesian product of two sets. Furthermore, two vertices u_i, v_i and u_j, v_j are adjacent whenever

$$(u_i, u_j) \in E(G_1) \text{and} (v_i, v_j) \in E(G_2) \text{and} w(u_i, u_j) = w(v_i, v_j) \tag{9}$$

or

$$(u_i, u_j) \notin E(G_1) \text{and} (v_i, v_j) \notin E(G_2) \tag{10}$$

where $w(u_i, u_j) = w(v_i, v_j)$ indicates that the vertex and edge labels for each respective pair of vertices are compatible. [6]

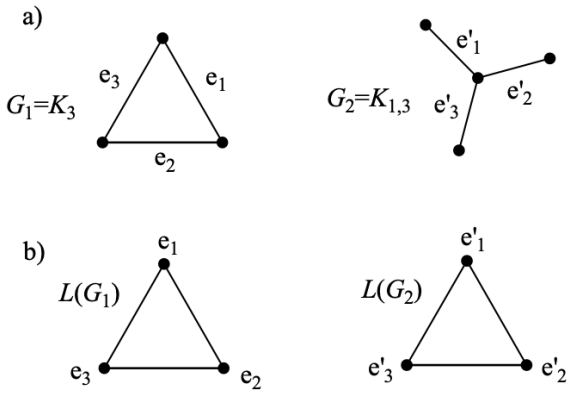


Figure 9: A ΔY exchange

Whitney proved that, provided a ΔY exchange does not occur, an isomorphism between two line graphs $L(G_1)$ and $L(G_2$ induces an edge isomorphism between the original graphs G_1 and G_2 of the two line graphs. [7] Thus, by preventing a ΔY exchange from occuring, it is possible to use the clique-based modular product approach to calculate the MCES. [4]

4.6 Feature vs Cost Metrics

Once the MCIS and MCES have been determined for the two graphs being compared, we calculate the various similarity metrics. For molecular graphical representations, we will be running cost-based metrics. As explained earlier, cost-based metrics, the similarity between two molecules is related to the number of edit operations that are required to transform one graph into the other. This mirrors the effect of string edit distance calculations run on SELFIES representations. Our cost-based metrics will be built upon the MCES between two graphs. [5]

4.7 Similarity Metrics

The three cost-based similarity metrics showing the most potential for graphical representations are the *Tanimoto coefficient*, the *QSBR coefficient*, and the *RASCAL coefficient*. Each method is briefly described below and a reference is provided to the original paper. Additionally, a table of values are provided for each coefficient for an example similarity calculation between terephthalic acid and aminobenzoic acid.

Tanimoto Coefficient [5]: The most popular fingerprinting similarity metric is the Tanimoto coefficient, which was originally built as a feature-based metric. However, this value can also be extended for graphs by comparing the sizes between them. Let G_1 and G_2 be the graphical representations of the molecules being compared, and let G_{12} be the MCES between G_1 and G_2. Note that $G_i = V_i + E_i$. Then,

$$T = \frac{|G_{12}|}{|G_1| + |G_2| - |G_{12}|} \tag{11}$$

QSBR Coefficient [1]: The second coefficient is the QSBR coefficient, which compares the number of nodes within the MCIS. Let G_3 and G_4 be the graphical representations of the molecules being compared, and let G_{34} be the MCIS between G_1 and G_2. Let G_i be the number of non-hydrogen nodes in G_i. Then,

$$QSBR = \frac{|G_{34}|}{|G_3|} \cdot \frac{|G_{34}|}{|G_4|} \tag{12}$$

Note the Tanimoto coefficient uses the MCES while this metric uses the MCIS.

RASCAL Coefficient [5]: The third similarity metric is the RASCAL coefficient, which is built upon the MCES and Tanimoto coefficient. The RASCAL coefficient fixes Tanimoto's issue of graph sizes which treat bonds and atom pairs with equal value, thus better approximating the chemical concept of similarity. Let G_1 and G_2 be the graphical representations of the molecules being compared, and let G_{12} be the MCES between G_1 and G_2. Let $V(G_{12})$ be the number of nodes in G_{12} and $E(G_{12})$ be the number of edges. Let the function $n(p, G)$ represent the number of unconnected subgraph structures within the MCES G containing p or more edges; if all subgraphs have fewer than p edges, then the function will be the total number of subgraph components. The constant β represents the weight assigned to matched bond pairs in compatible atoms, and the constant α is the penalty assigned for each unconnected component in G_{12}. RASCAL then improves

the definitions of graph sizes through the following equations.

$$\begin{aligned} |G_{12}| &= V(G_{12}) + \beta \cdot (1 - \alpha \cdot (n(p, G_{12}) - 1)) \cdot E_{12}) \\ |G_1| &= V(G_1) + \beta \cdot E(G_1) \\ |G_2| &= V(G_2) + \beta \cdot E(G_2) \end{aligned} \tag{13}$$

From here, the Tanimoto coefficient can be calculated to compare the similarity between two molecules. Through analysis, it has been found that the values of $p = 3$, $\alpha = 0.05$, and $\beta = 2.0$ are most effective in measuring chemical similarity. [5]

The three coefficients are compared in the following figures for the similarity between terephthalic acid and aminobenzoic acid (note this was also calculated for their SELFIES representations earlier).

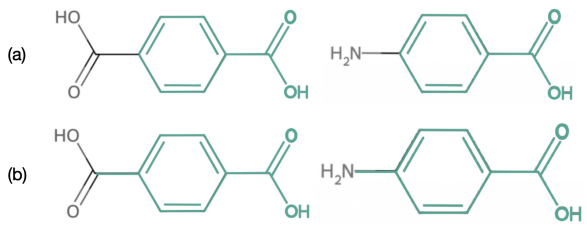


Figure 10: (a) MCIS between terephthalic acid (left) and aminobenzoic acid (right) (b) MCES between (terephthalic acid (left) and aminobenzoic acid (right)

Metric	Value
Tanimoto	0.760
QSBR	0.675
RASCAL	0.784

Table 2: Comparison of graphical similarity metrics

One main issue with these metrics is that the measures use strict atom and bond typing (i.e. MCS and MCES calculations match a specific atom only to the same element, so chlorine and fluorine cannot be matched). It may be possible to improve these metrics by allowing compatibility between similar elements. [5]

5 Conclusion

In this paper, we have described many different methods for developing a clustering model of molecules to build a large language model from. The two fingerprinting methods commonly used to store molecules which were discussed here are SELFIES and network graphs.

SELFIES, an extension of SMILES, offers a robust string representation which is both bijective and chemically valid. As such, it allows the opportunity to introduce random mutations to build a latent space of molecules for machine learning models. The common cost-based similarity metrics for SELFIES are Hamming distance, Levenshtein distance, Jaro-Winker metric, Jaccard similarity, and n-gram analysis.

Network graphs more deeply capture properties of molecules by providing a realistic representation of compounds. The common cost-based similarity metrics for graphs are the Tanimoto metric, QSBR coefficient, and RASCAL value.

Both of these representations have advantages and disadvantages. One issue with network graphs are that the metrics use strict atom and bond typing. While SELFIES addresses that concern through transposition checks, further constraints need to be made to allow transpositions for only similar elements. Unfortunately, SELFIES doesn't capture specific molecular properties as well as graphical representations. Thus, a blend of representations should be used to build an LLM. Additionally, it may be possible to improve upon the similarity metrics by blending different algorithms based on various fingerprinting metrics. The code for all of these metrics can be found on the GitHub page.

References

[1] B. Cuissart et al. "The Maximum Common Substructure as a Molecular Depiction in a Supervised Classification Context: Experiments in Quantitative Structure/Biodegradability Relationships". In: *Journal of Chemical Information and Computer Sciences* 42.5 (2002), pp. 1043–1052. DOI: 10.1021/ci020017w.

[2] V. J. Gillet et al. "Similarity and Dissimilarity Methods for Processing Chemical Structure Databases". In: *The Computer Journal* 41.8 (Jan. 1998), pp. 547–558. ISSN: 0010-4620. DOI: 10.1093/comjnl/41.8.547. eprint: https://academic.oup.com/comjnl/article-pdf/41/8/547/1032857/410547.pdf. URL: https://doi.org/10.1093/comjnl/41.8.547.

[3] Mario Krenn et al. "FIX". In: *Machine Learning: Science and Technology* 1.4 (Oct. 2020), p. 045024. ISSN: 2632-2153. DOI: 10.1088/2632-2153/aba947. URL: http://dx.doi.org/10.1088/2632-2153/aba947.

[4] John W Raymond and Peter Willett. "Maximum common subgraph isomorphism algorithms for the matching of chemical structures". In: *Journal of computer-aided molecular design* 16.7 (2002), pp. 521–33. DOI: 10.1023/a:1021271615909.

[5] John W. Raymond, C.J. Blankley, and Peter Willett. "Comparison of chemical clustering methods using graph- and fingerprint-based similarity measures". In: *Journal of Molecular Graphics and Modelling* 21.5 (2003), pp. 421–433. ISSN: 1093-3263. DOI: 10.1016/S1093-3263(02)00188-2.

[6] John W. Raymond, Eleanor J. Gardiner, and Peter Willett. "RASCAL: Calculation of Graph Similarity using Maximum Common Edge Subgraphs". In: *The Computer Journal* 45.6 (Jan. 2002), pp. 631–644. ISSN: 0010-4620. DOI: 10.1093/comjnl/45.6.631. eprint: https://academic.oup.com/comjnl/article-pdf/45/6/631/1184782/450631.pdf. URL: https://doi.org/10.1093/comjnl/45.6.631.

[7] Hassler Whitney. "Congruent Graphs and the Connectivity of Graphs". In: *American Journal of Mathematics* 54.1 (1932), pp. 150–168. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2371086 (visited on 06/30/2024).