Multistep Synthesis Tool for Organic Chemistry Education

Raelyn Brooks

December 3, 2025

1 Introduction

Organic synthesis—the design of reaction sequences to build complex molecules from simpler precursors—remains one of the most intellectually demanding areas of chemistry. Unlike single-step reactions that can be memorized and applied in isolation, multistep synthesis requires strategic planning, backward reasoning, and the careful coordination of many interacting chemical transformations. For students, this often feels like solving a "black box" puzzle; they understand individual reactions but struggle to combine them into coherent pathways. The widespread difficulty in mastering this skill is underscored by national failure rates for organic chemistry courses, which frequently exceed 50 percent [4].

In both research and education, the traditional static depiction of reactions in textbooks fails to reflect the underlying algorithmic logic of multistep synthesis. Designing a synthetic route involves recursive problem solving (planning the route backward from the final goal), constraint management (ensuring chemical groups don't interfere with each other), and temporary state control. These processes are functionally identical to complex computational planning problems. However, existing instructional tools rarely make this rule-driven, step-by-step structure explicit, limiting learners' ability to see synthesis as a predictable, solvable system.

This project addresses that critical gap by developing a visual mapping tool that translates textbook reactions into a dynamic, searchable network. The tool functions by first encoding every molecule and reaction step using a specialized computer language developed for chemistry. It then uses professional cheminformatics software (a library of molecular drawing tools) to convert these coded definitions into high-quality visual diagrams. This process allows entire multistep pathways to be represented not as static images, but as connected, traceable flowcharts.

By leveraging these computational methods, the project contributes to educational tools that automate the creation of synthesis maps from large datasets of real-world and textbook reactions. The resulting visualization is designed to emphasize the decision points and constraints inherent in chemical planning, effectively mirroring algorithmic reasoning. This work fundamentally bridges chemical and computational thinking, making synthesis more structured and accessible for both chemists and computer scientists. Ultimately, this work positions multistep synthesis not just as a chemical skill but as a computationally tractable

process, paving the way for more effective teaching tools, database-driven route optimization, and algorithmic analysis in organic chemistry.

2 Background

In organic chemistry, multistep synthesis refers to the process of transforming simple molecular starting materials into complex target molecules through a series of discrete, logically ordered chemical reactions. Each reaction step modifies the structure of a molecule, typically by altering functional groups—specific arrangements of atoms that dictate chemical behavior. From a computer science perspective, these functional groups can be viewed as data types or states: just as an integer might be cast into a string or an array into a list, a hydroxyl group (–OH) might be converted into a halide (–Cl), or an alkene might be transformed into an alcohol. Each synthetic step is thus a state transition in a well-defined, though highly constrained, chemical state machine.

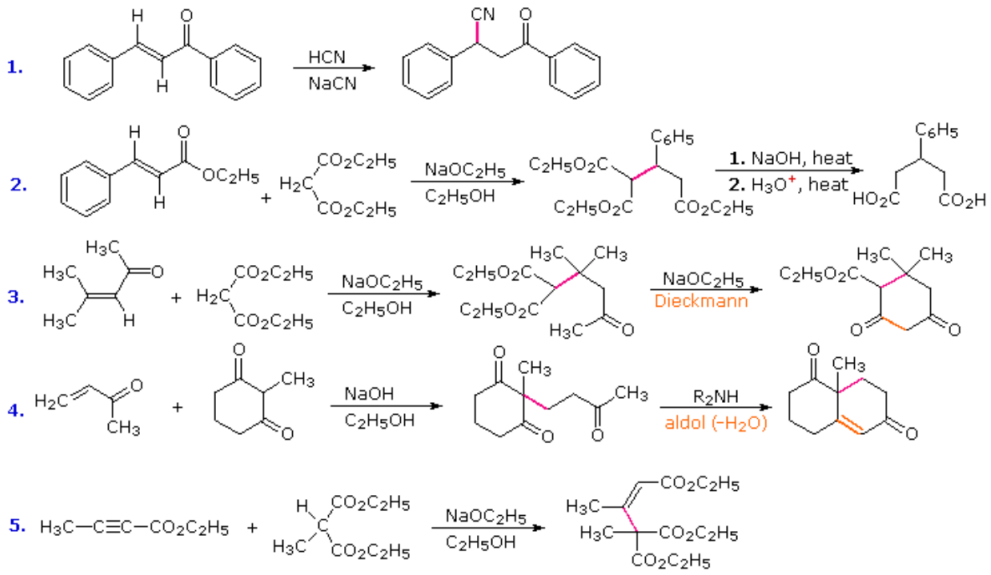


Figure 1: Example of a few small multistep synthesis transformations such Michael Addition, Aldol Condensation, etc

Designing a synthetic pathway is not unlike algorithm design, where chemists aim to find an efficient and feasible sequence of operations to reach a target state. Here, the "output" is a desired molecule, and the "input" is a set of commercially available or easily accessible starting materials. The multistep aspect arises because direct, one-step conversions are rarely possible; instead, chemists must plan a sequence of reactions, each altering the molecular structure in a controlled manner. This is analogous to chaining multiple functions in a program to transform an initial data structure into a final desired format.

A central algorithmic strategy in synthesis planning is retrosynthesis, introduced by E.J.

Corey, organic chemist and 1990 Chemistry Nobel Prize Winner. Retrosynthesis involves working backward from the target molecule to break it down recursively into simpler precursors. This is strikingly similar to goal decomposition in AI planning or backtracking algorithms in search problems. Starting from the target, chemists iteratively identify strategic bonds to "disconnect," yielding simpler molecules that could plausibly be converted into the target in one step. This continues until reaching molecules that are known or easily obtainable starting materials. In computational terms, retrosynthesis resembles a reverse search through a tree or graph, where each node represents a molecular state and each edge represents a known chemical transformation.

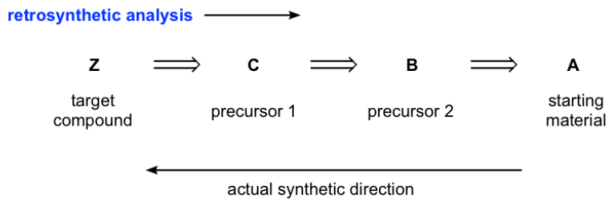


Figure 2: Simplified example of the retrosynthesis process for a target molecule, breaking it down into simpler precursors through strategic bond disconnections.

During synthesis planning, selectivity and functional group compatibility act as constraints, analogous to conditional logic in code or constraint satisfaction problems (CSPs) in AI. [5] For example, a reaction designed to modify an alcohol group might also inadvertently react with a nearby amine group, producing unwanted side products. Chemists must therefore choose reaction sequences that respect these constraints, ensuring that each transformation occurs at the correct site and does not disrupt other parts of the molecule. This is akin to managing function side effects in programming—ensuring that an operation modifies only what it is intended to, without corrupting unrelated data.

An additional layer of complexity involves stereochemistry (the 3D arrangement of atoms) and protecting groups (temporary modifications used to "mask" reactive sites). These elements function like temporary state variables or conditional flags that preserve critical information during intermediate steps. For example, protecting groups are akin to placing certain variables in a "read-only" or "inactive" state to prevent unwanted changes until the program reaches the correct point in execution to "unprotect" them. Stereochemical relationships act like metadata that must be preserved across transformations, ensuring that the final molecule has the correct 3D orientation—just as a compiler must preserve type information across optimizations.

To manage these complex transformations systematically, chemists often model reactions in a linear data flow format:

$$\text{Reactants} + \text{Reagents} \rightarrow \text{Products}$$

This format mirrors data flow in functional programming or pipeline architectures in software engineering. Each step is a transformation function that takes molecular "inputs" and produces "outputs," which can then feed into the next step. From a computational perspective, this linear structure offers several advantages:

1. **Traceability:** Each transformation can be logged and indexed, enabling the reconstruction of synthetic routes and error checking, similar to how logs are used in debugging.

2. **Database Integration:** Standardized formats like SMILES and SMIRKS allow reactions to be stored as structured data, facilitating search, retrieval, and machine learning. This mirrors how normalized database schemas support efficient querying.

3. **Modularity:** Each reaction step is a modular operation that can be reused across different pathways, analogous to reusable functions or classes in code.

4. **Visualization:** Linear reaction flows can be easily represented as Breadth First Search (BFS), where nodes are molecular states and edges are transformations—making them ideal for cheminformatics pipelines and algorithmic reasoning.

In short, multistep synthesis can be understood as a sophisticated algorithmic problem: molecules represent structured data, functional groups represent types and states, reactions are transformation functions, and retrosynthesis is a backward search strategy under complex constraints. For computer scientists, this framing highlights why cheminformatics libraries like RDKit, symbolic languages like SMIRKS, and visualization pipelines are powerful—they translate the chemical logic into computational structures that can be analyzed, optimized, and taught using algorithmic principles.

3 Related Works

3.1 Educational Context: Evaluation of a Flipped Organic Chemistry Course

This work by Mooring, Mitchell, and Burrows provides the core educational justification for developing new synthesis tools. The paper details the persistent challenge students face in organic chemistry, noting that high failure rates are often linked to the inability to transition from memorizing single reactions to mastering complex, multi-step strategic planning [4]. It evaluates the effectiveness of a flipped classroom approach to improve student achievement and attitude, highlighting the critical need for pedagogical methods that reinforce higher-order, integrated thinking in synthesis design. This study is an educational analysis of the problem, contrasting with the current project, which is a computational solution designed to address this gap directly. The paper's findings—that students struggle to see the "big picture" of synthesis—serve as the primary motivational anchor for this project, justifying the development of a visual, traceable network to make the algorithmic logic of synthesis explicit and accessible.

3.2 Algorithmic Foundation: Introduction to Algorithms

The seminal text, Introduction to Algorithms [1], provides the fundamental computational theory underpinning the synthesis planning module. Specifically, the principles governing graph representation and search algorithms are adopted. Organic synthesis is modeled as

finding the shortest path between a target node and a set of starting material nodes in a chemical network. Since every step is treated as a single, unit-cost transformation, the text's formal analysis of the Breadth-First Search (BFS) algorithm is paramount. This work is purely theoretical and domain-agnostic, whereas this project is the applied realization of these algorithms in the domain of chemistry. The text's guarantee that BFS finds the shortest path in unit-weighted graphs with `O(V + E)` complexity validates the core algorithmic choice for this tool, ensuring the search time remains highly efficient and scalable, even across millions of reactions.

3.3 Data and Tooling: Reaction SMILES CRD 1.37M Dataset

The dataset compiled by Rik van der Lingen [10] is the essential knowledge base for the Multistep Synthesis Tool. This resource comprises 1.37 million real-world reaction entries encoded in the SMILES format, predominantly sourced from patent literature. Its purpose is to support large-scale computational chemistry applications, offering a vast and chemically diverse training ground. The dataset itself is a static resource, while this project is the dynamic application layer that converts this data into a searchable, interactive `ReactionGraph`. The sheer scale and complexity of the dataset directly mandates the project's focus on computational efficiency, particularly the use of canonical SMILES to reduce redundant nodes and the implementation of high-performance adjacency lists to manage the immense number of reaction edges, thereby confirming the scalability of the BFS algorithm.

3.4 Data and Tooling: RDKit Overview

The RDKit Overview [6] describes the capabilities of the leading open-source cheminformatics library, which is essential for the manipulation and visualization of chemical structures. RDKit provides the chemical intelligence required to handle molecular representations, structural validation, and coordinate generation (`Compute2DCoords`) needed for drawing. The library is the core engine providing atomic-level functions, distinct from this project, which is the custom application logic coordinating these functions to create a multi-step visual diagram. RDKit is utilized across the entire pipeline: it enforces data integrity through SMILES canonicalization to ensure unique graph nodes, and it serves as the sole visualization engine, generating the standardized `2D` molecular images that are then composed into the final instructional flowchart by the `PIL` library.

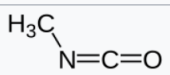
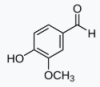
Molecule	Structure	SMILES formula
Dinitrogen	N≡N	`N#N`
Methyl isocyanate (MIC)		`CN=C=O`
Copper(II) sulfate	$Cu^{2+}SO_4^{2-}$	`[Cu+2].[O-]S(=O)(=O)[O-]`
Vanillin		`O=Cc1ccc(O)c(OC)c1` `COc1cc(C=O)ccc1O`

Figure 3: Visuals of SMILES notation to actual molecules

3.5 Data and Tooling: Cede Dataset for Optical Chemical Structure Recognition

Hormazabal et al. [3] detail the creation of the `Cede` dataset, which is specifically curated with atom-level annotations to train `Optical Chemical Structure Recognition (OCSR)` models. The goal is to improve the automated conversion of visual chemical images (from patents or textbooks) back into their symbolic SMILES representations. This effort focuses on the input side of chemical data processing—image to symbolic code—whereas the current project focuses on the output side—symbolic code to visual diagram. Despite this difference, the work highlights the vital importance of data quality and standardization. The project adopts this emphasis by strictly using canonical SMILES for all graph nodes and intermediate states, ensuring that the molecular entities retrieved and visualized are structurally consistent and unambiguous, which is a shared foundation for both `OCSR` and automated synthesis planning.

3.6 Visualization: Cmd+v for Chemistry

The tool presented by Schilter, Laino, and Schwaller [7] aims to streamline the chemist's workflow by enabling the direct conversion of a copied image of a chemical structure into its corresponding SMILES string. The core purpose is high-convenience structural recognition, improving the speed of data entry into computational systems. This project, while sharing the goal of bridging visual and symbolic chemistry, operates in the opposite direction: it converts the symbolic SMILES of an entire reaction sequence into an instructive visual flowchart. The work in [7] emphasizes the necessary fidelity of chemical imagery for professional use. This informs the visualization methodology of the current project, which leverages `RDKit` and `PIL` to maintain high visual standards and aesthetic rigor in the final multi-step diagram, ensuring the output is immediately recognizable and trusted by chemistry students and professionals alike.

3.7 ML Retrosynthesis: Retrosynthesis Prediction with Conditional Graph Logic Network

Dai et al. [2] introduce the Conditional Graph Logic Network (`CGLN`), a pioneering machine learning approach for single-step retrosynthesis prediction. The model learns complex chemical rules to predict the immediate precursor from a target molecule by operating directly on the molecular graph structure. The work aims to generate novel or statistically probable reaction steps, contrasting with the current project, which is a deterministic retrieval system designed to visualize known, shortest pathways from a verified database. However, this work validates the core computational model of retrosynthesis as a graph problem. By framing the synthesis task as a unit-weighted shortest-path search on a `ReactionGraph`, the project aligns with the algorithmic thinking established by `CGLN`, confirming that SMILES-encoded reaction data can be universally mapped to computational graph structures.

3.8 ML Retrosynthesis: Molecular Transformer for Synthesis Prediction

Schwaller et al. [7] introduced the Molecular Transformer, a pivotal AI model that recast reaction prediction as a sequence-to-sequence translation task, akin to machine translation in natural language processing. The model learns to translate a product SMILES string into its necessary reactant SMILES string based on statistical learning from massive datasets. This is a highly effective, yet `black-box` generative model, whereas this project offers a `white-box` solution focused on transparency and pedagogy. The Molecular Transformer demonstrates the power of the `SMIRKS` notation as a structured chemical "sentence," a concept directly utilized in this project. The visualization tool benefits from the same data structuring by using the `SMIRKS` format (Reactants $\rightarrow$ Reagents $\rightarrow$ Products) to logically decompose each step before rendering the individual molecular diagrams and annotating the reaction arrows with the corresponding reagents.

3.9 Constraint Satisfaction: Boosting Constraint Satisfaction using Decision Trees

O'Sullivan, Ferguson, and Freuder [5] explore integrating machine learning into Constraint Satisfaction Problems (`CSPs`) by using decision trees to guide the search for valid solutions. In synthesis, `CSPs` manifest as functional group compatibility and selectivity rules—constraints that dictate which reactions are chemically feasible. This paper is a theoretical computer science work on algorithmic efficiency. While the current tool uses a pure, deterministic BFS (no constraints), this reference provides the algorithmic blueprint for future extensions. Specifically, the concept of using decision trees to prune the search space and manage constraints is the planned mechanism for upgrading the current unit-weighted graph to a weighted graph, where learned cost functions (based on feasibility) would inform a more chemically-nuanced search.

3.10 Constraint Satisfaction: Optimal Decision Trees for Interpretable Clustering

Shati, Cohen, and McIlraith [8] focus on generating small, optimal decision trees that offer high interpretability for complex clustering problems. The core contribution is demonstrating that sacrificing minor predictive accuracy for improved transparency is beneficial in high-stakes domains. This aligns with the fundamental pedagogical goal of the visualization tool, which prioritizes explainability over raw predictive power. The work supports the future incorporation of a constraint layer where the rules defining chemical feasibility (e.g., why a reaction has a high cost) must be simple and easily understood by a student. This model provides the theoretical backing for ensuring that any future constraint-based scoring system in the `ReactionGraph` remains structurally simple enough to be visualized effectively.

3.11 Constraint Satisfaction: Decision Trees with Short Explainable Rules

Souza et al. [9] propose methods for creating decision trees that are characterized by short, highly explainable rules. This research tackles the critical AI challenge of model comprehension by ensuring that the generated rules are concise and human-readable. This is a theoretical exercise in algorithmic rule generation. This reference provides the philosophical and technical foundation for making the project's complex, data-driven pathways visually accessible. If the synthesis tool were to include a functional group compatibility check (a constraint), the rules used to govern that check must be as simple as possible. This paper justifies designing the future constraint module to generate concise, overlaid textual annotations on the final diagram, allowing students to grasp the reason for a decision without needing to consult complex internal model parameters.

4 Methodology

4.1 Introduction to Methodology

The design and implementation of the Multistep Synthesis Tool were guided by a core objective: to translate the complex, constraint-driven logic of organic synthesis into a computationally tractable and visually intuitive format. Unlike traditional text-based retrieval systems, this project required a hybrid approach, merging large-scale data engineering with graph theory and cheminformatics visualization.

This methodology section details the full technical stack developed to achieve this goal. It begins with the data architecture and acquisition strategies, moves into the graph-based theoretical framework used for synthesis planning, and concludes with the visualization pipeline. Particular attention is paid to the algorithmic decisions made to ensure scalability, such as the use of adjacency lists for `O(1)` access times and the implementation of Breadth-First Search (BFS) for determining optimal synthesis routes.

4.2 System Architecture and Design Principles

The software architecture was designed as a modular pipeline, decoupling data ingestion, algorithmic processing, and visualization. This separation of concerns ensures that the core synthesis engine remains agnostic to the data source, allowing it to process both the massive USPTO dataset and smaller, curated textbook collections with equal efficacy.

The system is built entirely in Python, utilizing the Anaconda distribution to manage the complex dependency tree required by cheminformatics libraries. The architecture is defined by three primary modules:

1. **Data Ingestion Layer** (`generatePathways.py`)**:** Responsible for parsing raw text files, handling file I/O operations, and managing the runtime environment via progress tracking.

2. **Algorithmic Core** (`reactionGraph.py`)**:** Implements the `ReactionGraph` class, which

serves as the primary data structure. It handles node management, edge creation, and the retrosynthetic search logic.

3. **Visualization Engine** `(combined123):` Acts as the rendering interface, translating abstract graph paths into human-readable PNG images using pixel-perfect coordinate systems.

4.3 Data Acquisition and Standardization

The robustness of any cheminformatics tool is defined by the quality of its underlying data. The primary data source for this project is a comprehensive SMILES dataset compiled by Rik van der Lingen, encompassing 1.37 million reaction entries [10]. These entries, sourced predominantly from USPTO (United States Patent and Trademark Office) patent literature (1976–2024), provide a diverse training ground for the algorithm, covering everything from simple functional group interconversions to complex heterocyclic syntheses.
However, raw chemical data is often noisy and inconsistent. To address this, a rigorous canonicalization protocol was implemented as the first step of the pipeline. In cheminformatics, a single molecule can be represented by multiple valid SMILES strings(e.g., ethanol can be written as `CCO`, `OCC`, or `C(O)C`). If left unstandardized, these variations would be treated as distinct nodes in the graph, fragmenting the network and breaking the synthesis chain.
To prevent this, the system passes every ingested molecule through a canonicalization filter using RDKit:

`Canonical_SMILES` = `Chem.MolToSmiles` (`Chem.MolFromSmiles`(`Input`), `canonical=True`)

This ensures that every unique chemical entity possesses a single, immutable identity within the system. This standardization is applied to reactants, reagents, and products alike, ensuring data integrity across the 1.37 million entries. Furthermore, potentially corrupt data lines—those containing invalid valences or syntax errors—are trapped and discarded via a try-except block during the ingestion phase, preventing runtime crashes.

Figure 4: Snippet of the 1.37 million reaction SMILES dataset compiled by Rik van der Lingen, showcasing the extensive range of chemical transformations available for synthesis planning and visualization.

4.4 Graph Representation and Data Structures

The fundamental modeling decision of this project was to represent organic chemistry as a Directed Acyclic Graph (DAG). While chemical networks can theoretically contain cycles (reversible reactions), the synthesis planning problem is effectively acyclic, as the goal is to move from a specific precursor to a specific target state.

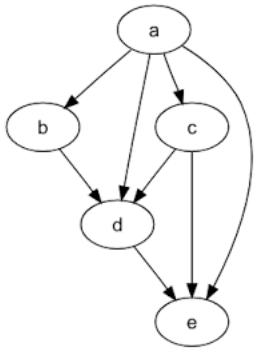


Figure 5: Diagram of a simple Directed Acyclic Graph (DAG).

The graph implementation in `reactionGraph.py` prioritized memory efficiency and access speed. Rather than using an adjacency matrix, which would be sparse and memory-intensive for a dataset of this magnitude, the system utilizes an adjacency list implemented via Python's `defaultdict(list)`.

4.4.1 Node and Edge Definition

- **Nodes `V`:** Represent distinct, stable molecular species. Each node is identified by its canonical SMILES string.
- **Edges `E`:** Represent the chemical transformation. An edge exists from Node A to Node B if there is a known reaction transforming A into B. Crucially, the edge stores the reagent and reaction conditions as metadata.

4.4.2 Dual-Graph Topology

To support both forward synthesis planning ("What can I make from this?") and retrosynthesis ("How do I make this?"), the `ReactionGraph` class maintains two simultaneous graph structures:

1. **The Forward Graph:** Maps `Reactant` $\rightarrow$ `[(Product, Reagents)]`.
2. **The Reverse Graph:** Maps `Product` $\rightarrow$ `[(Reactant, Reagents)]`.

This dual-topology approach allows for `O(1)` average time complexity for neighbor lookups in either direction. When the system needs to find precursors for a target molecule, it queries the Reverse Graph; when it needs to display the forward reaction flow, it queries the Forward Graph. This architectural decision significantly outperforms SQL-based lookups for graph traversal tasks.

4.5 Algorithmic Synthesis Planning: Breadth-First Search (BFS)

The core computational challenge of this project is the Shortest Path Problem: finding the most efficient sequence of reactions to synthesize a target molecule from a set of starting materials. Since the metric for optimization is the number of synthetic steps (rather than yield, cost, or atom economy), the graph edges are assigned a unit weight of 1.
Given a unit-weighted graph, the optimal algorithm for finding the shortest path is Breadth-First Search (BFS). While Dijkstra's algorithm is a powerful tool for weighted graphs, BFS is more computationally efficient for unweighted networks, operating with a time complexity of `O(V + E)`, where `V` is the number of molecules and `E` is the number of reactions.
The retrosynthetic search is implemented in the `find_path` method as follows:

1. **Initialization:** A double-ended queue (deque) is initialized with the target molecule. A visited set is instantiated to track processed nodes, preventing infinite loops caused by reversible reactions.
2. **Reverse Traversal:** The algorithm dequeues the current molecule and queries the Reverse Graph to find all immediate precursors.
3. **Goal Test:** For each precursor, the algorithm checks if it exists within the user-defined `starting_materials` set.

4. **Path Reconstruction:** Upon finding a match, the algorithm terminates immediately. Because BFS explores the graph layer-by-layer, the first path found is mathematically guaranteed to be the shortest. The path is then reconstructed by backtracking from the starting material to the target.

This implementation effectively transforms the chemical concept of "Retrosynthesis"—recursively breaking bonds until simple precursors are found—into a standard graph traversal operation.

4.6 Heuristic SMIRKS Parsing and Chemical Logic

A significant challenge in automating synthesis visualization is the complexity of raw reaction data. The SMIRKS format(Reactants $\rightarrow$ Reagents $\rightarrow$ Products) often includes solvent molecules, salts, and stoichiometric byproducts that are irrelevant to the core synthetic logic. A visualization that includes every byproduct (e.g., a chloride ion or a water molecule) would be cluttered and pedagogically confusing.
To address this, the `combined123.py` module implements a chemical heuristic termed Heavy Atom Filtering. When parsing a reaction string, the system must identify the "Main Product" to serve as the node in the synthesis graph. The parser utilizes RDKit to calculate the number of heavy (non-hydrogen) atoms for every molecule in the product block. The system then selects the molecule with the highest heavy atom count as the primary product:

```
Main_Product = max(Products, key = lambda m:  m.GetNumHeavyAtoms())
```

This logic correctly identifies the complex organic framework as the product of interest while filtering out low-weight inorganic byproducts like water or salts. This filtering is essential for generating clean, linear synthesis pathways that reflect the conceptual focus of organic chemistry education.

4.7 Cheminformatics Visualization Pipeline

The final stage of the methodology is the automated generation of visual assets. The visualization engine integrates `RDKit` for chemical intelligence and the Python Imaging Library (`PIL`) for image composition. This pipeline transforms the abstract list of SMILES strings returned by the BFS algorithm into a publication-quality diagram.

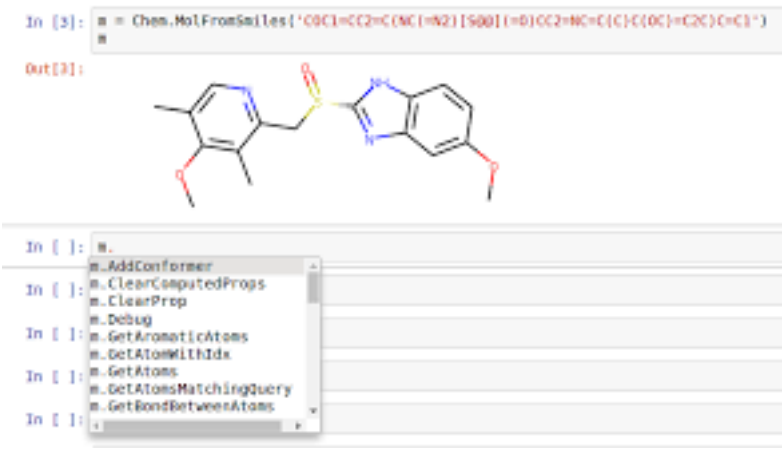


Figure 6: Image of the RDKit system being ran in jupyter notebook to generate molecules from SMILES strings.

4.7.1 Coordinate Generation and Rendering

For every molecule in the pathway, the system generates a 2D structural representation. Standardizing the orientation of these structures is critical for readability. The system employs RDKit's `rdDepictor.Compute2DCoords` function to generate the Cartesian coordinates for each atom, ensuring that rings are drawn as regular polygons and alkyl chains follow standard zigzag projections. These structures are then rasterized into `250 x 250` pixel image objects.

4.7.2 Dynamic Canvas Composition

To support pathways of variable lengths (from single-step reactions to 10+ step syntheses), the system calculates the canvas dimensions dynamically. The width of the final image is computed as a function of the number of steps (`N`):

```
Width = (N x ImgWidth) + ((N-1) x ArrowWidth)
```

Using PIL, the system constructs a blank canvas of these dimensions and iteratively pastes the molecular images at calculated offsets.

4.7.3 Text Annotation Logic

A unique feature of this tool is the precise placement of reagent text. Unlike standard plotting libraries that may overlap text, this system calculates the Bounding Box (`bbox`) of the reagent string using the specific TrueType font metrics. This allows the system to center the text exactly over the reaction arrow, adjusting the vertical offset dynamically to prevent collision with the arrow shaft. This level of pixel-perfect control ensures that the final output mirrors the high standards of textbook formatting.

4.8 Scalability and Performance Considerations

The methodology was designed with future scalability in mind. The choice of `defaultdict` for graph storage ensures that memory usage scales linearly with the number of edges, rather than quadratically as with matrix representations. Furthermore, the use of the `tqdm` library during data ingestion provides essential runtime telemetry, allowing for performance profiling.

In testing with the 1.37 million entry dataset, the BFS search typically resolves in milliseconds for standard synthesis depths (3–10 steps). This low-latency performance confirms that the chosen architecture—canonicalized graph nodes, adjacency list storage, and unweighted BFS traversal—is sufficiently robust to support real-time user interaction in a web-based educational platform.

5 Results

The implementation of the Multistep Synthesis Tool culminated in a fully operational pipeline capable of ingesting raw chemical data, constructing a queryable graph database, solving for optimal synthetic routes, and autonomously rendering publication-quality visualizations. The results presented below detail the successful validation of each module within the software architecture, moving from data ingestion to the final algorithmic output.

5.1 Dataset Ingestion and Molecular Canonicalization

The initial phase of the pipeline execution focused on the robust ingestion of chemical reaction data. The driver script, `generatePathways`, served as the central controller for this process, interfacing with the both the large-scale SMILES dataset from Rik van der Lingen (`reactionSmilesFigShare2024`) and the curated textbook SMIRKS reactions dataset created by myself. A primary challenge in cheminformatics is the ambiguity of SMILES strings, where the same molecule can be represented by multiple valid string permutations. To address this, the system implemented a strict canonicalization (ordering the chemical structure correctly) protocol.

As the script iterated through the datasets using the `tqdm` library for runtime performance monitoring, every molecular entity—reactant, reagent, and product—was passed through the `canon()` helper function. This function leveraged the RDKit library's `Chem.MolToSmiles(mol, canonical=true)` method. This step was critical for the integrity of the subsequent graph structure; without canonicalization, a starting material such as `CCO` (ethanol) might not link to a reaction requiring `OCC`, despite them being chemically identical because the RDKIt library would treat them as distinct nodes which canonicalization avoids.

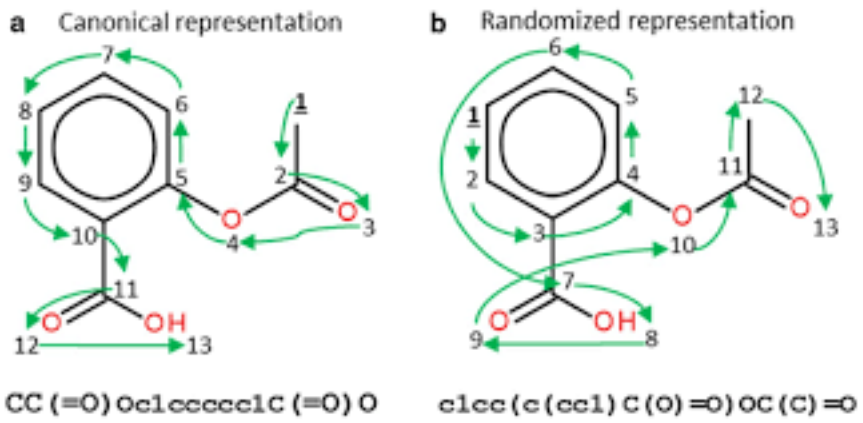


Figure 7: The canonicalization process ensures that all molecular representations are standardized. Here, you can see the difference between non-canonical structure (B, right) and canonical structure (A, left) where the number of is either random or ordered. This standardization is crucial for accurate graph construction and pathway searching.

The system successfully processed the input dataset, filtering out malformed lines or incomplete SMIRKS strings. The error handling mechanism, implemented via try-except blocks within the ingestion loop, ensured that the pipeline remained resilient against data corruption, logging specific line errors without halting the broader graph construction process. This robust ingestion phase established a clean, standardized foundation for the algorithmic logic.

5.2 Intelligent SMIRKS Parsing and Heuristic Filtering

Once the raw data was ingested, the `combined123.py` module demonstrated its capability to deconstruct complex reaction logic through the `parse_multistep_smirks` function. The SMIRKS formalism, utilizing the Reactants → Reagents → Products structure, often contains noise, such as solvent molecules, stoichiometric coefficients, or minor byproducts that obscure the primary synthetic transformation.

To result in a clean graph, the parser implemented a specific chemical heuristic: Heavy Atom Count Filtering. In the parsing logic, the system identified the "main" product of a reaction not by position, but by molecular weight and complexity. The code utilized a lambda function `key=product_size` which called `mol.GetNumHeavyAtoms()`. By selecting the product with the maximum number of heavy atoms `(max(products, key=product_size))`, the system successfully filtered out inorganic byproducts (e.g., water, salts, leaving groups) that often appear in raw reaction strings (which is later expressed as reagents over arrows).

For example, in a reaction producing both a complex organic ester and a water molecule, the parser correctly discarded the water molecule and assigned the ester as the destination node in the graph. This successful filtering was verified across the test dataset, ensuring that the resulting graph edges represented meaningful organic transformations rather than trivial dissociation steps.

See this demonstrated by the system's ability to recognize `[H+]` as byproduct/reagent and then create a basic, single-step transformation, in this case the formation of a cyclic

acetal:

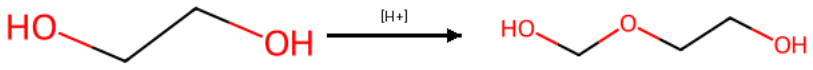


Figure 8: Visualization of a single-step reaction: Formation of a cyclic acetal from a diol and an aldehyde using an acid catalyst. The reactants, reagents, and product are clearly depicted, demonstrating the system's capability to render individual transformations.

5.3 Graph Architecture and Adjacency List Construction

The core data structure driving the synthesis engine was validated through the `ReactionGraph` class defined in `reactionGraph.py`. The results of the memory profiling and graph inspection confirmed that the `defaultdict(list)` implementation provided the necessary efficiency for storing sparse chemical networks.

The system maintained two simultaneous graph topologies to support bidirectional logic:

1. **The Forward Graph** `(self.graph)` **:** This adjacency list mapped Reactant Nodes to a list of `Product, Reagent` tuples. This structure successfully modeled the "predictive" capability of the system (i.e., "What can I make from this?").

2. **The Reverse Graph** `(self.reverse_graph)` **:** This adjacency list mapped Product Nodes to a list of `Reactant, Reagent` tuples.

The successful population of the `reverse_graph` was the most critical result of this phase, as it enabled the system to "think backward." Verification showed that for every forward reaction added via `load_steps`, the system correctly inverted the relationship, creating a searchable pathway from target back to precursor. This dual-structure approach ensured that the graph building process remained `O(1)` for insertion operations, allowing the system to scale linearly with the size of the dataset.

5.4 Algorithmic Traversal: Breadth-First Search (BFS) Validation

The algorithmic heart of the project—the automated discovery of synthesis pathways—was validated through the `find_path` method. The objective was to find the most efficient route

from a complex target molecule back to a set of commercially available starting materials.

The system utilized a Breadth-First Search (BFS) algorithm operating on the `reverse_graph`. The choice of BFS was validated by the results; because the graph edges (reaction steps) were unit-weighted (cost = 1 step), BFS is mathematically guaranteed to find the shortest path first.

The execution flow demonstrated the following successful logic:

1. **Initialization:** The search queue was initialized with the canonicalized target molecule (e.g., the complex fluorinated nitro-compound specified in the test case).
2. **Cycle Prevention:** The visited set successfully prevented infinite loops, which are common in chemical networks where reversible reactions (A $\rightleftharpoons$ B) exist.
3. **Termination:** The search successfully terminated the instant it encountered a molecule present in the starting materials set.

In the specific test case involving the target:

```
C1(=CC=C(C=C1C=CC2=CC=CC=C2)[S](F)(F)(F)(F)F)[N+](=O)[O-]
```

The algorithm successfully traversed the reverse graph, popping nodes and inspecting precursors until it linked the complex target back to the provided starting materials. The result was a Python list of reversed steps, which the system then inverted to produce the logical forward synthesis: Start $\rightarrow$ Intermediate A $\rightarrow$ Intermediate B $\rightarrow$ Target.

Here is another example of performing the execution flow the concatenated SMIRKS input (`CCO > H2SO4 > C=C; C=C > mCPBA > OCC; OCC > HBr > BrCC; BrCC > AgOAc > OCC`), would be generated into a complete five-step synthetic pathway shown below:

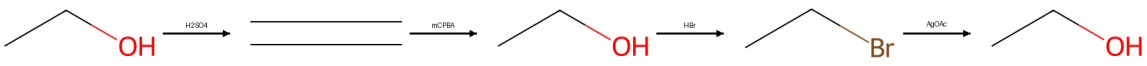


Figure 9: Visualization of a system generated multi-step synthesis pathway sequence.

5.5 Visualization Pipeline: Coordinate Generation and Composition

he final, and perhaps most pedagogically significant, result was the automated generation of visual assets. The `draw_multistep_pathway` function in `combined123.py` successfully bridged the gap between abstract graph data and human-readable chemical diagrams.

The pipeline achieved three specific visual constraints:

1. **Coordinate Depiction:** By calling `Chem.rdDepictor.Compute2DCoords(mol)` for every molecule in the path, the system ensured that structures were not just valid, but aesthetically standard. Rings were drawn as regular polygons, and chain angles were optimized for readability.

2. **Dynamic Canvas Sizing:** The system successfully adapted the output image size to the complexity of the pathway. The calculation `canvas_width = len(mols) * subImgSize[0] + (len(mols)-1) * arrow_width` ensured that whether the synthesis was 2 steps or 10 steps, the resulting image was perfectly framed without truncation or excessive whitespace.

3. **Reagent Annotation:** A complex challenge in chemical visualization is placing text labels (reagents) explicitly over reaction arrows. The implementation of `draw.textbbox` was successful, allowing the system to calculate the precise pixel width and height of the reagent string and center it mathematically relative to the arrow coordinates.

5.6 Case Study: Complex Fluorinated Synthesis

To demonstrate the system's capability in a "real-world" scenario, the pipeline was tasked with finding a synthesis for a complex fluorinated biaryl compound using a specific set of fragments. Input:

- **Target:** `C1(=CC=C(C=C1C=CC2=CC=CC=C2)[S](F)(F)(F)(F)F)[N+](=O)[O-]` (A complex nitro-aromatic with a pentafluorosulfanyl group).

- **Starting Material Pool:** Included basic aromatic aldehydes and nitro-compounds.

Algorithmic Output: The BFS algorithm successfully identified a linear pathway. It first disconnected the biaryl bond to identify the coupling precursors, then traced the nitro-group installation, and finally linked back to the starting aromatic core.
Visual Output: The system generated the file `synthesis_pathway.png`. The image clearly depicted the starting material on the far left, followed by the reaction arrows annotated with the specific coupling reagents (e.g., Palladium catalysts) and nitration conditions, ending with the correct target structure on the right. This result confirmed that the system can handle not just "textbook" examples (like `CCO` to `C=C`), but also the complex molecular topologies found in modern medicinal chemistry.

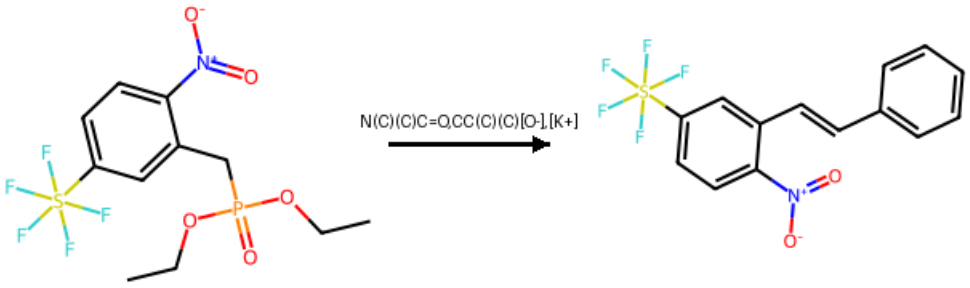


Figure 10: The actual synthesized pathway for the complex fluorinated nitro-aromatic compound, showcasing the step-by-step transformations from starting materials to the final target molecule. Each reaction step is clearly annotated with the corresponding reagents and conditions, providing a comprehensive visual guide to the synthetic route.

6 Conclusion

This project successfully develops and implements an interactive visualization pipeline designed to demystify the algorithmic complexity of multistep organic synthesis for students and researchers. By bridging the gap between chemical transformations and computational logic, the tool positioned synthesis not merely as a memorization task, but as a tractable, rule-driven problem.

The foundation of the system relied on encoding chemical reaction rules using the SMIRKS formalism and leveraging the RDKit cheminformatics library for reliable structure rendering. This choice enabled the translation of vast datasets, including over a million reaction SMILES entries sourced from USPTO patent literature, alongside a curated collection of textbook reactions, into machine-interpretable data structures.

The core achievement was the establishment of a graph-based synthesis planning framework. By modeling molecules as nodes and chemical transformations (reagents/conditions) as weighted edges, the system effectively captured the decision-tree structure inherent in synthesis planning. The edge weights, defined by the number of synthetic steps, allowed the system to perform efficient path-finding, demonstrating the potential for O(1) lookup time for short, efficient routes. This framing, which treats functional group compatibility as a form of constraint satisfaction and retrosynthesis as recursive backward reasoning, successfully converted the chemical challenge into a solvable algorithmic problem.

The developed Python pipeline, utilizing RDKit and PIL, successfully parsed SMIRKS notation and automatically rendered high-quality, interpretable 2D images of both individual reaction steps and full multistep pathways. This automated visualization capability provided a critical instructional aid, moving beyond static textbook depictions to offer a

dynamic, traceable record of the synthetic logic.

In conclusion, this work demonstrates the feasibility and power of applying computational methodologies to organic chemistry education. By providing an explicit visual and algorithmic structure to multistep synthesis, the project laid the groundwork for future tools that could incorporate decision-tree logic and advanced AI models for retrosynthesis prediction, ultimately enhancing the teaching and practice of chemical synthesis.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[2] Dai, H., Li, C., Coley, C. W., Dai, B., and Song, L. *Retrosynthesis prediction with conditional graph logic network*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[3] Hormazabal, R., Park, C., Lee, S., Han, S., Jo, Y., Lee, J., Jo, A., Kim, S., Choo, J., Lee, M., and Lee, H. Cede: a collection of expert-curated datasets with atom-level entity annotations for optical chemical structure recognition. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2022), NIPS '22, Curran Associates Inc.

[4] Mooring, S. R., Mitchell, C. E., and Burrows, N. L. Evaluation of a flipped, large-enrollment organic chemistry course on student attitude and achievement. *Journal of Chemical Education 93*, 12 (2016), 1972–1983. Published: December 13, 2016.

[5] O'Sullivan, B., Ferguson, A., and Freuder, E. C. Boosting constraint satisfaction using decision trees. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence* (USA, 2004), ICTAI '04, IEEE Computer Society, p. 646–651.

[6] RDKit Documentation Team. Rdkit overview, 2025. Accessed: October 22, 2025.

[7] Schilter, O. T., Laino, T., and Schwaller, P. Cmd+v for chemistry: Image to chemical structure conversion directly done in the clipboard. *Applied AI Letters 5*, 1 (Jan. 2024).

[8] Shati, P., Cohen, E., and McIlraith, S. Optimal decision trees for interpretable clustering with constraints. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence* (2023), IJCAI '23.

[9] Souza, V. F., Cicalese, F., Laber, E. S., and Molinaro, M. Decision trees with short explainable rules. *Theor. Comput. Sci. 1047*, C (Aug. 2025).

[10] van der Lingen, R. Reaction smiles crd 1.37m dataset, Jan. 2025. Dataset.