SMILES-to-SMILES Morphing & Intermediate Extraction

Shreyansh Seth Shashwat Chandra
June 2025

Contents

1 Problem Statement

Given two valid SMILES strings **A** (*source*) and **B** (*target*), the script

1. identifies a common *core* between A and B,
2. prunes A down to that core,
3. grows the molecule atom-by-atom until it matches B—logging *every unique, chemically valid intermediate* encountered,
4. repeats the process in the reverse direction (B → A), and
5. writes all intermediates—indexed by exploration path—to a user-supplied CSV for downstream analytics.

The objective is to obtain a chemically interpretable trajectory while penalising synthetically unattractive detours.

2 Theoretical Foundations

2.1 Core Invariance

Why? Preserving a shared substructure constrains the search space and keeps each step chemically meaningful (e.g., scaffold hops rather than total synthesis). It minimises arbitrary bond rearrangements and encourages transformations that a synthetic chemist would recognise as feasible.

How? Three increasingly liberal strategies are attempted until one succeeds:

1. **SMARTS seeding** – user-supplied aromatic motifs (e.g., `c1ccccc1`).
2. **Murcko scaffold match** – retains ring systems that medicinal chemists typically consider immutable.
3. **Strict MCS** – RDKit maximum common substructure with element/valence and ring-closure constraints.

Precautions

- `molvs.Standardizer` normalises tautomers, charges, and stereochemistry, ensuring both molecules start from comparable baselines.
- Every sanitisation is followed by `Chem.Kekulize` to avoid mixed aromaticity states, which could otherwise mislead SMARTS matching or fingerprint generation.
- `verify_core_invariance` asserts that neither core atoms nor core bonds mutate, providing a hard guarantee that the reaction path stays chemically interpretable.

2.2 Search Strategy

2.2.1 Phase I – Beam Search

A stochastic best-first search scoring each state with

$$\text{Composite} = \alpha \cdot \text{Sim}_{\text{target}} \; - \; \beta \cdot \text{SA} \; + \; \gamma \cdot \text{Sim}_{\text{source}}, \tag{1}$$

where similarities are Tanimoto indices on Morgan fingerprints ($\times 100$) and SA is the synthetic-accessibility score.
We chose beam search here because it efficiently explores multiple high-scoring paths in parallel, guiding the search toward promising partial structures without exhaustively expanding low-probability branches.

Table 1: Default scoring weights

Symbol	Meaning	Default
α	importance of reaching B	1.0
β	penalty for poor SA	0.1
γ	encouragement to stay near A	0.5

Diversity vs. exploitation The algorithm maintains a beam of size `BEAM_SIZE`. We keep the top $\lfloor (1 - \varepsilon)\,\text{BEAM\_SIZE} \rfloor$ states strictly by score while injecting a random sample from the next best `RANK_CUTOFF` candidates. This hybrid approach avoids premature convergence on sub-optimal routes yet focuses compute on promising areas.

Termination criteria

- Similarity $\geq$ `SIM_THRESHOLD` (95 % by default), signalling that the target framework has effectively been reached.
- Frontier exhausted (no further children meet validity checks).
- `MAX_STEPS` reached—an upper bound to prevent unbounded exploration.

2.2.2 Phase II – Breadth-First Sweep

Once a high-similarity molecule is found, a complete breadth-first search over the remaining unmapped atoms ensures *exhaustive* enumeration of intermediates along that path. Because every addition is validated and canonicalised, duplicates are discarded via a global `seen` set.
We use BFS here—rather than other uninformed methods like DFS or uniform-cost search—because in comparative trials BFS consistently delivered the best coverage and shortest-step enumeration, so we stuck with it for maximal reliability.

2.3 Why the Method Works

Fingerprint Guidance Morgan fingerprints capture atom environments; optimising their Tanimoto similarity makes geometric sense because adding the correct atom in the correct context will typically increase similarity to the target while retaining overlap with the source core.

Synthetic Plausibility Bias The synthetic-accessibility (SA) penalty biases the search away from molecules that are chemically exotic or densely functionalised. In practice, this removes a large tail of unreasonable candidates, making the beam more efficient and the resulting trajectories more publication-ready.

Bidirectional Coverage Running the algorithm in both directions (A $\rightarrow$ B and B $\rightarrow$ A) catches asymmetries. Some atoms that are easy to add might be hard to delete and vice-versa. Logging from both directions therefore increases the chance of discovering a symmetric, interpretable midpoint set.

Core-Lock Guarantee By explicitly preventing any change to the mapped core atoms and bonds, we avoid artefacts where the algorithm temporarily destroys part of the scaffold only to rebuild it later—a common failure mode in naive graph searches. The core-lock thus anchors the trajectory in meaningful chemical space.

2.4 Implementation Highlights

This subsection links each theoretical element above to the concrete Python code in `morph.py`.

- **Module Setup** – Global constants (`BEAM_SIZE`, `EPSILON`, etc.), RDKit log suppression, Morgan fingerprint generator, and SA-Score resource path are declared at the top of the script.
- **Core Detection Functions**

  `seed_core_by_smarts` Implements SMARTS seeding.

  `try_murcko_core` Extracts Murcko scaffolds using RDKit's ring-info utilities.

  `strict_mcs` Wraps `rdFMCS.FindMCS` with strict ring and valence parameters.
- **Sanitisation Pipeline** – `validate_smiles` $\rightarrow$ MolVS standardiser $\rightarrow$ RDKit sanitise $\rightarrow$ Kekulise, guaranteeing consistent input.
- **Beam Search Engine** – `beam_search` embodies Phase I, computing composite scores, maintaining a priority-ordered list, and applying ε-greedy diversification.

- **BFS Enumeration** – `add_atoms_bfs` performs Phase II, traversing the target adjacency map once the similarity threshold is satisfied.
- **Integrity Guards**
  - `verify_core_invariance` halts execution if any core atom symbol or bond order mutates.
  - Fragmentation checks ensure molecules stay single-component.
- **Data Logging** – Novel intermediates are streamed to `explored_molecules.csv` via `csv.DictWriter`; canonical + isomeric SMILES compose the uniqueness key used by the global `seen` set.
- **Bidirectional Driver** – `morph_bidirectional` sequentially calls `morph_direction` for A → B and then B → A, passing along an incrementing path ID counter to preserve lineage across both directions.
- **Post-processing Utility** – The optional statistics block at the end of the script re-opens the CSV to compute min/mean/median/std for SA and similarity metrics and prints the five highest-scoring intermediates.

3 File Outputs

- `explored_molecules.csv` – each intermediate with `SA`, `SimSource`, `SimTarget`, `CompositeScore`, `PathID`.
- Console summary – descriptive statistics and **Top-5** intermediates by composite score.
- Inline molecule grids (Jupyter / VS Code) for visual inspection.

4 Usage Guide

4.1 Prerequisites

Ensure you have Python 3.8 or newer installed. Then run this single Jupyter cell to install all dependencies via pip:

```
# -------- In Cell 1: Install Dependencies --------
%pip install -q rdkit molvs
%pip install -q ipython notebook matplotlib
%pip install -q pillow urllib3
```

Note: Also put sascore.py & `fpscores.pkl.gz` in the same directory as the jupyter notebook.

4.2 Editing the Script

```
# -------- In Cell 2: USER PARAMETERS --------
CSV_PATH = r"PATH_TO_explored_molecules.csv"    # <- CSV Path
SOURCE = "SMILES_A_HERE"    # <- Source (A)
TARGET = "SMILES_B_HERE"    # <- Target (B)
#OR
if __name__ == "__main__":
    morph_bidirectional(
        "SMILES_A_HERE",   # <- Source (A)
        "SMILES_B_HERE",   # <- Target (B)
        r"PATH_TO_explored_molecules.csv"   # <- CSV Path
    )
```

```
# -------- In Cell 4(Optional): Visualization --------
smile_to_show = "ANY_SMILES_TO_PLOT"
```

4.3 Running in Jupyter Notebook

Execute the notebook cells in order:

1. **Cell 1:** Run the first cell once to download all the needed libraries and dependencies.
2. **Cell 2:** Run the second cell to load all imports, set parameters, and define the core functions and search routines.
3. **Cell 3:** Run the third cell to post-process the CSV (summary statistics for SA and similarity values).
4. **Cell 4:** Run the fourth cell to visualize any single SMILES structure using the PlotSmiles function.

5 Parameter Tuning Tips

Knob	Effect	Typical Range
`BEAM_SIZE`	search breadth vs. RAM	200 – 5000
ε (`EPSILON`)	exploration rate	0.05 – 0.25
`SIM_THRESHOLD`	early stop criterion	90 – 99 %
α, β, γ	scoring weights	project-specific

6 Conclusion

This pipeline delivers a robust, chemically grounded sequence of transformations between any two SMILES strings by:

- Anchoring on a verified core to maintain interpretability,
- Employing a hybrid beam/BFS search to balance exploration and exhaustive coverage,
- Steering the search with similarity and synthetic-accessibility metrics to favour plausible intermediates,
- Logging every unique, valid intermediate for downstream analysis or visualization.

Overall, this approach yields a transparent, reproducible set of molecular waypoints that bridge source and target structures, facilitating both theoretical study and practical route planning.