ASCEC-v04 & Similarity-v01

User Manual

Manuel Gómez – Sara Gómez – Albeiro Restrepo

Química Física Teórica, Instituto de Química
Universidad de Antioquia, AA 1226 Medellín, Colombia

UNIVERSIDAD®
DE ANTIOQUIA

Script Version: 4.0 | **Last Updated:** February 2026

Contents

1 Distribution

The ASCEC-v04 & Similarity-v01 package is distributed via a public GitHub repository:

- **Repository:** https://github.com/manuel2gl/qft-ascec-similarity
- **License:** Open source (see LICENSE file for details)
- **Supported OS:** Scripts are primarily developed and tested for Linux (Ubuntu, CentOS, Fedora). They should also work on macOS and Windows (with minor adjustments, e.g., for shell scripts and environment variables).
- **Contents:**
  - Python scripts for conformational sampling, clustering, and analysis
  - Documentation and brief theoretical description
  - Example input files and workflows
  - Web-based input generator
- **Requirements:**
  - Python 3.9 or higher (3.11 suggested)
  - ORCA (recommended) or Gaussian for quantum chemistry calculations
  - See Installation section for full dependencies

2 Introduction

2.1 What is ASCEC?

ASCEC (Annealing Simulado Con Energía Cuántica / Simulated Annealing with Quantum Energy) is a computational tool for automated configurational sampling and analysis of molecular clusters. It integrates:

- **Simulated Annealing**: Efficient exploration of potential energy surfaces
- **Quantum Mechanical Calculations**: High-accuracy energy evaluation (ORCA/Gaussian)
- **Hierarchical Clustering**: Identification of representative structures using a physicochemical feature vector.
- **Boltzmann Analysis**: Population-weighted structure selection

2.2 Key Features

✓ **Multi-Replica Simulations** - Independent runs for a robust sampling

✓ **Fully Automated Workflows** - From sampling to final analysis in one command

✓ **Intelligent Retry System** - Automatic handling of calculation failures

✓ **Quality Control** - Validation of optimized structures (imaginary frequencies)

✓ **Visual Analytics** - Energy evolution diagrams and clustering dendrograms

✓ **Web Interface** - Graphical input file generator with molecular visualization

3 Theoretical Background

3.1 Simulated Annealing Algorithm

The ASCEC module drives the exploration of the configurational space through a Markov Chain Monte Carlo (MCMC) protocol implemented within a simulated annealing framework. To ensure statistical significance, the algorithm generates thousands of trial configurations. The quantum energy of every generated structure is calculated using efficient methods (supporting any theoretical level available in the external package, e.g., PM3, AM1, B3LYP, or MP2) via interfaces with Gaussian or ORCA.

The acceptance of a structural perturbation depends on the change in potential energy, $\Delta E = E_{new} - E_{old}$. If the structural change lowers the energy ($\Delta E < 0$), the move is accepted immediately. In the case of an energy increase ($\Delta E > 0$), the acceptance behavior depends on the chosen criterion:

Standard Metropolis Criterion (Optional) When explicitly requested (via the `--standard` flag), the algorithm follows the classical Boltzmann probability. An uphill move is accepted if a random number $\xi \in [0, 1]$ satisfies:

$$\xi < e^{-\frac{\Delta E}{k_B T}} \tag{1}$$

Modified Metropolis Criterion (Default) To mitigate kinetic trapping in local basins and prevent jumps to physically unrealistic high-energy configurations ASCEC implements a modified acceptance condition. Unlike the standard probabilistic approach, which compares the Boltzmann factor to a random number, this method evaluates the relative energy cost of the perturbation. The new structure is accepted if:

$$\Phi(\Delta E) < P(\Delta E) \tag{2}$$

where $P(\Delta E) = e^{-\Delta E/k_B T}$ is the temperature dependent Boltzmann probability, and $\Phi(\Delta E)$ represents the relative energy change defined as:

$$\Phi(\Delta E) = \frac{\Delta E}{|E_{new}|} \tag{3}$$

This criterion was found to be more adequate for this class of problems. By removing the stochastic element found in the standard Metropolis method, the algorithm avoids the accidental rejection of chemically viable structures due to random number generation, while still filtering out moves where the relative energy increase is drastic.

Temperature Schedules: The simulation temperature decreases over n_T steps to gradually reduce the acceptance of high-energy states. A choice between a linear (decreasing T by a constant amount) and a geometric (decreasing T by a constant percentage) route is available:

- *Linear Quenching:* The temperature is reduced by a fixed amount dT at each step:

$$T_{n+1} = T_n - dT$$

- *Geometric Quenching:* The temperature is scaled down by a cooling factor f (in %):

$$T_{n+1} = T_n \times \left(1 - \frac{f}{100}\right)$$

3.2 Monte Carlo Moves

The configuration space is explored via three independent perturbations:

1. **Translation:** Random displacement of the center-of-mass (Limit: `max_disp`).
2. **Rotation:** Random rotation around principal axes (Limit: `max_rot`).
3. **Conformational (Optional):** Modification of internal dihedral angles.

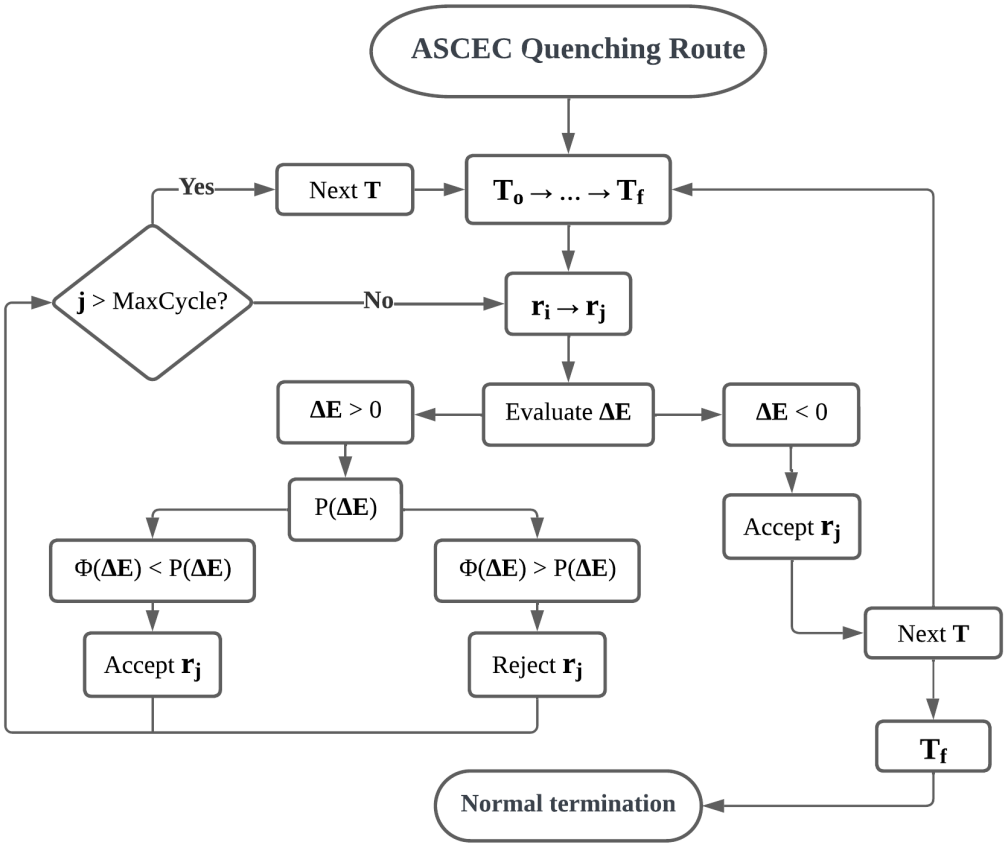

Figure 1: SA procedure with a modified Metropolis test.

3.3 Topological Clustering (Similarity)

While stochastic exploration (ASCEC) is effective at traversing energy barriers, it inherently suffers from a data management limitation: the generation of highly redundant datasets. A typical annealing run yields hundreds of candidate structures, the vast majority of which converge to identical basins of attraction. To address this, the **Similarity** module functions as a post-processing discriminator designed to filter, validate, and cluster the raw output into unique chemical motifs.

The scope of this algorithm is to bridge the gap between stochastic chaos and ordered chemical insight. By reducing massive datasets into non-redundant families, it facilitates the manual revision of the Potential Energy Surface (PES) and ensures that only topologically distinct minima are subjected to high-level quantum mechanical refinement.

3.3.1 The Physicochemical Feature Vector

Standard clustering methods often rely solely on geometric superposition (RMSD), which scales poorly (N^2) and depends on atomic indexing. To overcome this, we define the conformational identity of a structure i using a multi-dimensional scalar feature vector, $\mathbf{x}_i$. This vector aggregates continuous physicochemical descriptors derived from the pre-optimized quantum mechanical data:

$$\mathbf{x}_i = [E_{elec}, G, S, \Delta E_{HL}, \mu, R_g, A, B, C, \nu_1, \nu_N, d_{HB}] \tag{4}$$

where the components represent energetics (E, G, S), electronic properties (HOMO-LUMO gap ΔE_{HL}, dipole moment μ), geometry (Radius of Gyration R_g, Rotational Constants A, B, C), vibrational modes (ν_1, ν_N), and hydrogen bonding metrics (d_{HB}).

3.3.2 Weighted Similarity Metric

Since the vector components possess disparate physical units, a feature-wise Z-score standardization is applied ($z_{ik} = (x_{ik} - \mu_k)/\sigma_k$). The similarity between two structures is then quantified using a Weighted Euclidean Distance (d_{ij}):

$$d_{ij} = \sqrt{\sum_{k=1}^{N} (w_k \cdot z_{ik} - w_k \cdot z_{jk})^2} \tag{5}$$

where w_k represents user-definable weights used to bias the clustering focus (e.g., prioritizing H-bonding patterns over soft vibrational modes).

3.3.3 Hierarchical Two-Step Clustering

To ensure both computational efficiency and high structural fidelity, the clustering process operates via a hierarchical "funnel" protocol involving two distinct steps:

1. **Step I: Physicochemical Clustering (Coarse-Grained).**
   The algorithm first performs hierarchical clustering (Ward's linkage) based on the feature vectors $\mathbf{x}$. This rapidly groups configurations that share similar energetics, electronic states, and general shapes into broad families. This step drastically reduces the dataset size by merging redundant snapshots without performing expensive geometric alignments.

2. **Step II: RMSD Refinement (Fine-Grained).**
   Optionally, a secondary geometric check is applied within each identified cluster. A Root Mean Square Deviation (RMSD) analysis is performed to distinguish stereoisomers or isoenergetic conformers that map to identical scalar vectors (e.g., enantiomers or distinct proton ordering networks with identical energies). This ensures that subtle geometric variations are not lost during the vector-based reduction.

3.3.4 Quality Control

Beyond clustering, the module serves as a topological filter. It screens for the presence of **one or more imaginary frequencies**, indiscriminately identifying first-order saddle points (transition states) and higher-order instabilities. If such a structure clusters tightly with a true minimum, it is discarded as a redundant, non-converged distortion. However, if it forms an isolated cluster, it is flagged as a unique topological feature and output for re-optimization, ensuring that potential basins are not overlooked merely due to incomplete convergence.

4 Installation

4.1 System Recommendations

While the Python scripts are lightweight, the Quantum Chemistry backend (ORCA/Gaussian) requires resources:

- **RAM:** Minimum 8GB (16GB+ recommended for DFT calculations).
- **CPU:** Multi-core processor recommended (ASCEC can parallelize ORCA jobs).
- **Storage:** SSD recommended. Optimization trajectories can generate large text files.
- **OS:** Linux (Ubuntu/CentOS) is the primary supported environment. macOS/Windows (via WSL) are supported but may require path adjustments.

4.2 Required Software

4.2.1 Core Python Dependencies

The core functionality relies on standard scientific Python libraries.

Package	Purpose
NumPy	High-performance array operations and linear algebra.
SciPy	Optimization routines and statistical functions.
scikit-learn	Algorithms for hierarchical clustering and data preprocessing.
Matplotlib	Generation of energy plots and dendrograms.
cclib	Parsing of Gaussian and legacy ORCA ($\leq$ 5.0.x) logs.
OPI	Parsing of modern ORCA ($\geq$ 6.1) outputs.

Table 1: Python libraries required for ASCEC and Similarity modules.

***Compatibility Note:** The **Similarity** module automatically selects the appropriate parser based on the output file version. `cclib` is used for Gaussian and older ORCA versions ($\leq$ 5.0.x), while `OPI` is required for ORCA 6.1+. *Note: ORCA 6.0 is `not` supported by either parser; please use v5.0.x or upgrade to v6.1+.* At least one parser must be installed.

4.2.2 Quantum Chemistry Backends

The workflow requires an external electronic structure package to perform QM calculations.

1. ORCA (Recommended) ORCA is the preferred backend for this workflow due to its computational efficiency and versatile licensing. It is fully supported by both the generation (ASCEC) and analysis (Similarity) modules (v5.0.x and v6.1+). Please refer to Section 12 for a detailed guide on setting up ORCA.

- **Advantages:** Free for academic use, excellent performance, rich feature set, and robust native frequency analysis.

2. Gaussian Gaussian (v09 or v16) is fully supported as a legacy backend. While it requires a commercial license, it remains a robust option for users already integrated into the Gaussian ecosystem.

- **Advantages:** Industry standard, comprehensive method support, extensive documentation, and long-term stability.

4.3 Step-by-Step Installation

4.3.1 Option A: Quick Install (pip)

Note: OpenBabel is difficult to install via pip alone due to C++ compilation requirements. You may need to install OpenBabel via your system package manager (e.g., `sudo apt-get install python3-openbabel`).

```
git clone https://github.com/manuel2gl/qft-ascec-similarity.git

cd qft-ascec-similarity

# Install Python dependencies
pip install numpy scipy matplotlib scikit-learn cclib orca-pi

chmod +x ascec-v04.py similarity-v01.py
```

4.3.2 Option B: Conda Environment (Recommended)

This method ensures both OpenBabel, cclib and OPI are installed correctly with all dependencies.

I. Prepare the directory and clone the repository
Establish a dedicated directory for the software and clone the source repository. To synchronize an existing installation with the latest updates, execute `git pull`.

```
mkdir -p ~/software/ascec04
```

```
git clone https://github.com/manuel2gl/qft-ascec-similarity.git ~/software/ascec04/
```

II. Install Miniconda (Skip if Conda is already installed)
Download and run the installer. During installation, type `yes` when asked to initialize conda.

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
```

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Restart your shell after installation to apply changes:

```
source ~/.bashrc
```

III. Create and configure the environment

Create a clean Python 3.11 environment named `py11` (or your preferred name).

```
conda create -n py11 python=3.11 -y
```

IV. Set up Aliases

To make running the programs easier, add shortcuts to your shell configuration. Open your `.bashrc` file:

```
micro ~/.bashrc
```

Scroll to the bottom of the file and paste the following lines:

```
# conda_py11 shortcut
alias py11='conda activate py11'

# ASCEC & Similarity aliases
alias ascec='python $HOME/software/ascec04/ascec-v04.py'
alias simil='python $HOME/software/ascec04/similarity-v01.py'
```

Save the file (Ctrl+S) and exit (Ctrl+Q). Then, reload the configuration:

```
source ~/.bashrc
```

V. Install Dependencies

Activate the environment, should see (py11) instead of (base), and install the required libraries.

```
py11
```

```
conda install numpy scipy matplotlib scikit-learn -y
```

```
conda install -c conda-forge cclib openbabel -y
```

```
pip install orca-pi
```

Note: You can install both parsers for maximum compatibility with different ORCA versions. OPI is installed via pip even in conda environments.

VI. Verify Installation

Ensure that the environment is active, you should see `(py11)` in your terminal, and check that the commands are working.

```
python -c "import numpy, scipy, matplotlib, sklearn, cclib, opi; print('All packages OK')"
```

```
ascec --version
```

```
ascec --help
```

```
simil --help
```

```
orca -v
```

```
g09 < /dev/null
```

```
obabel -V
```

4.3.3 Automatic Installation

We provide a simplified shell script that automates the entire process. It installs Miniconda (if not present), sets up the dependencies (OpenBabel, cclib, OPI), and configures the necessary aliases. First download the install.sh script.

```
wget https://raw.githubusercontent.com/manuel2gl/qft-ascec-similarity/main/install.sh
```

Installation Steps

The installation can be performed by executing the `install.sh` script, which automatically uses a Python 3.11 environment (py11). Alternatively, if the `base` environment is preferred, install the latest Miniconda and downgrade the Python version within `base`, then run the script again.

```
micro install.sh
```

```
#==========================================
# CONFIGURATION
#==========================================
# Set to TRUE to create a separate 'py11' environment with Python 3.11
# Set to FALSE to install into the base conda environment (default)

INSTALL_PY11=TRUE

echo "> Starting ASCEC 'One-Click' Installation..."
```

Save the file (Ctrl+S) and exit (Ctrl+Q).

```
bash install.sh
```

```
source ~/.bashrc
```

Once completed, the `ascec` and `simil` commands will be ready to use immediately.

5 Input File Format

The structure of the ASCEC input file is positional. Below is a concise example.

5.1 Example Input (w6.in)

```
# Input for water hexamer Simulation

1 10            # Line 1: Simulation Mode & Number of Config
                # 1: Annealing; 0: Random configurations
5               # Line 2: Simulation Cube Length (Angstroms)

2               # Line 3: Annealing Quenching route
                # (1: Linear, 2: Geometrical).

100.0  10.0  50   # Line 4: Linear Quenching Parameters

600.0   5.0  100  # Line 5: Geometric Quenching Parameters

3000  50        # Line 6: Maximum MC Cycles per T and floor value

1.0 1.0         # Line 7: Maximum Displacement (A)
                # & Maximum Rotation (radians)

0 60            # Line 8: Conformational sampling (%)
                # & Maximum dihedral rotation (degrees)

2 orca          # Line 9: QM Program Index & alias
                # (1: Gaussian, 2: ORCA, 3: for other, etc.)

pm3             # Line 10: Hamiltonian & Basis Set
                # (e.g., pm3, hf)(e.g., 6-31G*, STO-3G)

1 8             # Line 11: nprocs (QM and ASCEC evaluations)

0 1             # Line 12: Charge & Spin Multiplicity

6               # Line 13: Number of Molecules (nmo)

# Lines below: Molecule Definition
# (Num Atoms, Label, followed by xyz coordinates, separated by *)

*
3
water1
O   0.000000  0.000000  0.000000
H   0.757000  0.586000  0.000000
H  -0.757000  0.586000  0.000000
*
3
water2
O   0.000000  0.000000  0.000000
H   0.757000  0.586000  0.000000
H  -0.757000  0.586000  0.000000
*   ... (W3-6)
```

5.2 Parameter Explanation

Table 2: Detailed description of ASCEC input parameters.

Line	Parameters	Description
1	[mode] [nconfigs]	**1**: Annealing, **0**: Random Generation. `nconfigs` is only used if mode is 0.
2	`box_length`	Length of the cubic simulation box edge in Å.
3	`route`	Temperature schedule: **1** (Linear) or **2** (Geometric).
4	[Ti] [dT] [steps]	Linear: Start Temp (K), Step size (K), Number of steps.
5	[Ti] [fac] [steps]	Geometric: Start Temp (K), Reduction factor (%), Steps.
6	[max] [floor]	Maximum Monte Carlo cycles per temperature step and floor value as it decreases (0.1) with each step.
7	[disp] [rot]	Max translation distance (Å) and max rotation (radians). Maximum displacement for each mass center, and maximum rotation of the principal axes of each molecule. The second field must be 0 for atomic clusters.
8	[prob] [dih]	Probability of conformational move (%) and max dihedral change (°).
9	[code] [alias]	QM Program ID (**1**=Gaussian, **2**=ORCA) and the command alias to run it.
10	[method] [basis]	Hamiltonian/Method (e.g., `pm3`, `b3lyp`) and Basis set (if applicable) for the energy calculations during the annealing.
11	[cpu_Q] [cpu_A]	Processors passed to the QM program, and (optional) processors for ASCEC logic and parsing.
12	[chg] [mult]	Total system Charge and Spin Multiplicity.
13	`nmol`	Number of molecules defined. For atomic clusters each atom must be considered a molecule. If atoms interacting with molecules are considered, then each atom not being part of a molecule must be considered a molecule.

6 Web Input Generator

ASCEC-v04 includes a browser-based tool to generate input files automatically, fetch structures from PubChem, and visualize the simulation box in 3D.

6.1 Features

- **PubChem Integration:** Automatically converts SMILES, IUPAC names (e.g., "aspirin"), or brand names into 3D XYZ coordinates using the PUG REST API.

  Compounds search: https://pubchem.ncbi.nlm.nih.gov/compound/

- **3D Visualization:** Uses 3Dmol.js to preview the simulation box, showing the boundaries and initial molecular placement.

- **Protocol Builder:** easy creation of multi-stage workflows (e.g., `calc`, `similarity`, `opt`).

6.2 Accessing the Tool

You can access the generator in two ways:

1. **URL:** Visit https://manuel2gl.github.io/qft-ascec-similarity/

2. **Direct:** Run the following command in your terminal:

```
ascec input
```

6.3 Interface Overview

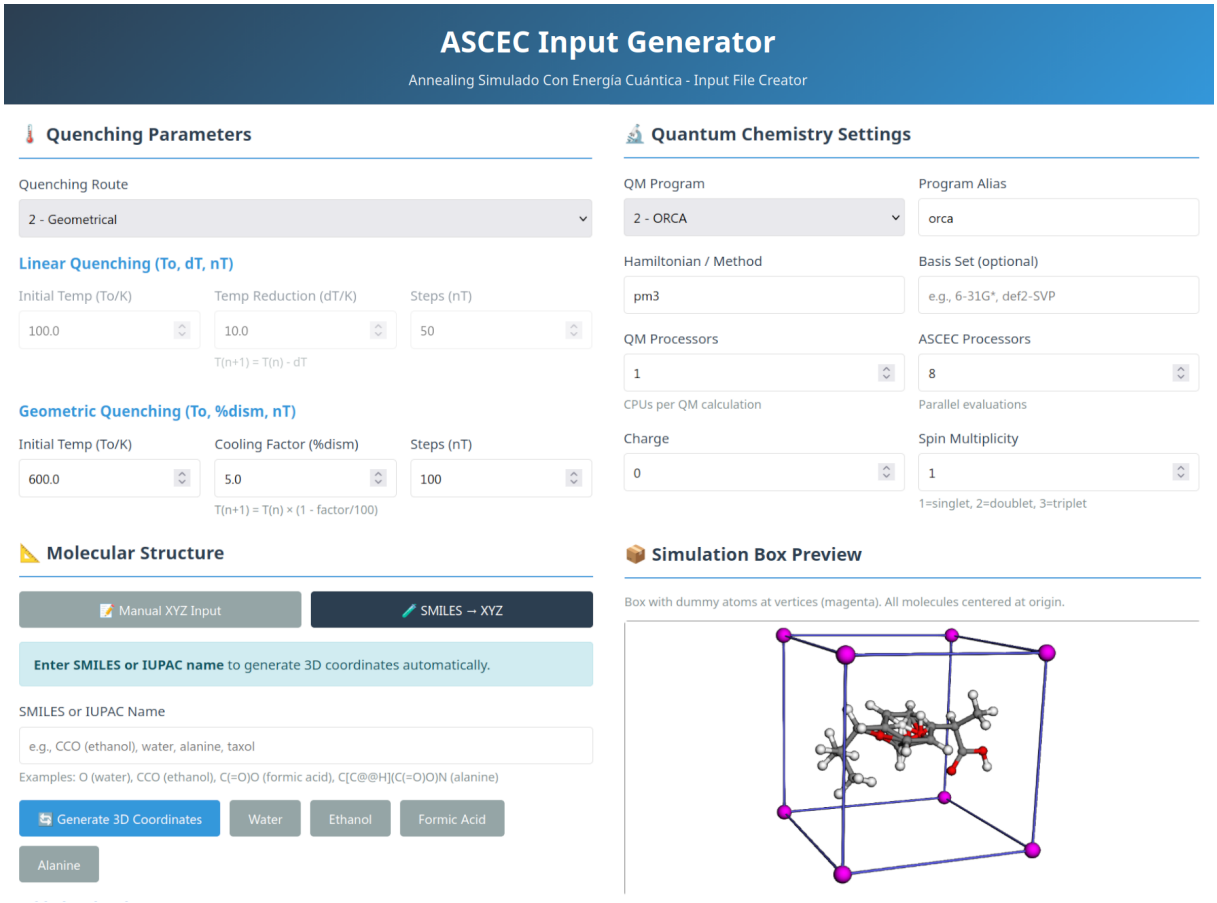

Figure 2: ASCEC Web Input Generator interface with 3D visualization and parameter forms.

As shown in Figure 2, the interface allows users to search for compounds on the left and preview the simulation box on the right. The generated input file can be copied directly or downloaded.

7 Standalone Functions

Commands for individual steps using ORCA 6.1.1 & Gaussian09.

We need the input file (.in) and the input QM file for pre-optimization with its respective launcher (see the examples folder for reference).

7.1 Annealing

7.1.1 Box Size

Triplicated run for water hexamer
Once the input file is generated, we can see the box size suggestions

```
/
└── w6.in
```

```
ascec w6.in box
```

```
(py11) :~/.../examples/water_hexamer/orca6$ ascec w6.in box

================================================================================
Box length analysis
================================================================================

1. Molecular volume and hydrogen bonding analysis:
--------------------------------------------------
Number of molecules to place: 6
Total molecular volume: 6.46 Å³
Total H-bond network volume: 67.86 Å³
Total effective volume: 74.31 Å³

3. Current box analysis:
-------------------------
Cube's length = 5.00 Å
Current effective packing: 59.5%
+ Molecular: 5.2%, H-bond network: 54.3%
Current free volume: 51 Å³ (40.5%)
Largest molecular extent: 1.13 Å
Extremely dense - may prevent proper H-bond formation

4. Recommendations for H-bonded systems:
----------------------------------------
• For isolated clusters: 11.4 Å (5% effective packing)
• For cluster formation: 9.1 Å (10% effective packing)
• For network studies: 7.9 Å (15% effective packing)
• Includes space for H-bond network (avg. bond length: 2.5 Å)
```

We can see that a length of 5.0 Å is not ideal for this system, therefore we can either change it directly in the input file or instruct the ascec script to use a suggested length and rewrite the parameter itself.

```
ascec w6.in r3 --box10
```

The r3 command tells the script to run the annealing in triplicate, and the box10 flag uses a box length for an effective packing density of 10%.

```
Using recommended box size: 9.1 Å (10.0% effective packing)
Creating 3 replicated runs in 'annealing/'...
  Created: w6_1/w6_1.in
  Created: w6_2/w6_2.in
  Created: w6_3/w6_3.in
Created launcher script: launcher_ascec.sh

To run all simulations sequentially, use:
  ./launcher_ascec.sh
```

```
./launcher_ascec.sh
```

A launcher is generated to run all replicas in series. Simply run it.

```
(py11) :~/.../examples/water_hexamer/orca6$ ./launcher_ascec.sh
ASCEC v04 environment is now active via direct script setup.

* ASCEC-v04: Feb-2026 *

Using random seed: 115439

Attempting initial QM energy calculation...

Attempt 1/100 for initial QM calculation.
Calculation successful. Energy: -71.61760217 a.u.

Using rigid-body moves only (translation + rotation)

Starting annealing simulation...


Annealing Step 1/100 at Temperature: 600.00 K

Annealing Step 2/100 at Temperature: 570.00 K

Annealing Step 3/100 at Temperature: 541.50 K

(...)

Annealing Step 99/100 at Temperature: 3.94 K

Annealing Step 100/100 at Temperature: 3.74 K

Annealing simulation finished.

Final lowest energy found: -71.65131472 a.u.
Total QM calculations performed: 2087
======================================================================
```

The process starts from a "Big Bang" position, where the centers of mass of all molecules are placed overlapping at the center of the simulation box. This high-energy state represents the worst possible starting condition, which helps in avoiding biases in the trajectory. To resolve this situation, random movements are made until a stable configuration is found. A configuration is considered successful when its potential energy can be calculated without errors, such as those arising from atomic overlaps.

The system will make up to 100 attempts to find such a state. The first successful configuration becomes the starting point from which the annealing simulation begins. Should all 100 attempts fail, the simulation will terminate, providing data that can be used to diagnose the problem or refine the initial parameters.

7.2 Pre-Optimization Strategy

Preprocessing Annealing Candidates
The stochastic annealing protocol generates a vast ensemble of redundant configurations, making immediate high-level quantum mechanical refinement computationally prohibitive. To address this bottleneck, a pre-optimization stage is implemented using the efficient semi-empirical **GFN2-xTB** method. Crucially, this intermediate level of theory must maintain the topological fidelity of the Potential Energy Surface (PES), ensuring that geometries relax into their nearest local minima without erroneously merging distinct basins or eliminating valid low-energy isomers. To validate the efficacy of this strategy and strictly balance computational efficiency with accuracy, this work evaluates two distinct protocols: a hierarchical two-step refinement, and a benchmark approach involving the direct processing of the raw annealing data.

ORCA Structure	**Gaussian Structure**
/	/
├── annealing/	├── annealing/
├── b97_orca.inp	├── b97_gau.gjf
├── gfn2_orca.inp	├── launcher_gau.sh
├── launcher_orca.sh	└── w6.in
└── w6.in	

7.2.1 Input Templates

I. ORCA Input Template (`gfn2_orca.inp`)
This is a standard ORCA input file with specific placeholders that ASCEC utilizes to inject configuration data. In ORCA, lines starting with `#` are comments.

- `#name`: (Optional) ASCEC replaces this with the filename for traceability.
- `#`: A single hash mark tells the script where to insert the Cartesian coordinates block.

```
#name

! Opt GFN2-xTB TightSCF Numfreq PAL8

%geom
  maxiter 5000
end

* xyz 0 1
#
*
```

II. Gaussian Input Template (`b97_gau.gjf | .com`)
This is a standard Gaussian input file with corresponding placeholders. In Gaussian, lines starting with `!` are comments.

- `!name`: (Optional) ASCEC replaces this with the filename.
- `!`: A single exclamation mark indicates the insertion point for Cartesian coordinates.
- *Note: Ensure the template ends with some blank lines as required by Gaussian format, example has two of them.*

```
!name

%nprocshared=8
%mem=16GB
# wB97XD/6-31+G(d) Opt=Cartesian Freq DensityFit iop(1/6=500)

Water Hexamer

0 1
!
```

III. Launcher Templates (`launcher_orca.sh | launcher_gau.sh`)
This shell script defines the environment variables and execution paths for the quantum chemistry software. It includes a specific placeholder (`###`) where ASCEC will append the sequence of execution commands for all generated input files.

```
#!/bin/bash

# Define paths to ORCA 6.1.1 installation
export ORCA_BASE=$HOME/software
export ORCA611_ROOT=$ORCA_BASE/orca_6_1_1
export OPENMPI418_ROOT=$ORCA_BASE/openmpi-4.1.8

# Save system paths to prevent nesting
_SYSTEM_PATH="$PATH"
_SYSTEM_LD_LIBRARY_PATH="$LD_LIBRARY_PATH"

# Export ORCA and OpenMPI paths
export PATH="$ORCA611_ROOT:$OPENMPI418_ROOT/bin:$_SYSTEM_PATH"
export LD_LIBRARY_PATH="$ORCA611_ROOT:$OPENMPI418_ROOT/lib:
$_SYSTEM_LD_LIBRARY_PATH"

echo "ORCA 6.1.1 environment is now active."
mpirun --version

###
```

Listing 1: ORCA 6.1.1 Launcher Template

```
#!/bin/bash

# For Gaussian 09:
export g09root=$HOME/software/gaussian
export GAUSS_SCRDIR=$HOME/scratches/g09
(...)

echo "Gaussian 09 environment is now active."

###
```

Listing 2: Gaussian09 Launcher Template

7.2.2 Calculation Setup

Generating Calculation Files
The ASCEC `calc` command scans the `annealing` directory for result files (`.xyz`) and interactively guides the user to generate the necessary input files for pre-optimization.

```
ascec calc gfn2_orca.inp launcher_orca.sh
```

```
(py11) :~/.../water_hexamer/orca6$ ascec calc gfn2_orca.inp launcher_orca.sh
Found 3 XYZ file(s) to process:
- ./annealing/w6_1/result_115439.xyz
- ./annealing/w6_2/result_609695.xyz
- ./annealing/w6_3/result_154701.xyz

============================================================
XYZ file selection
============================================================

Special options:
a. Process all result_*.xyz files independently
c. Combine all result_*.xyz, then process the combined file

Select files (numbers separated by spaces, 'a' for all, or 'c' to combine): c

Merged 3 files into calculation/combined_r3.xyz with 416 configurations
  Created: opt_conf_1.inp
  Created: opt_conf_2.inp
  (...)
  Created: opt_conf_416.inp

Completed processing. Created 416 input files total.
Created calculation system in 'calculation' directory.
```

Resulting Input Files
The script automatically injects the coordinate data and comments into the templates. Note how the placeholders have been replaced:

```
# Configuration: 1 | E = -71.61760217 a.u. | result_115439

! Opt GFN2-xTB TightSCF Numfreq PAL8

%geom
  maxiter 5000
end

* xyz 0 1
O       7.521687      8.123473      2.987348
H       7.729738      7.604909      3.779022
(...)
*
```

Listing 3: Generated ORCA Input

```
#!/bin/bash
# ... (Environment variables as defined in template) ...

echo "ORCA 6.1.1 environment is now active."

###

# Run QM using the full path
$ORCA611_ROOT/orca opt_conf_1.inp > opt_conf_1.out ; \
$ORCA611_ROOT/orca opt_conf_2.inp > opt_conf_2.out ; \
(...)
$ORCA611_ROOT/orca opt_conf_416.inp > opt_conf_416.out
```

Listing 4: Generated Launcher Script

Executing the Calculations
Navigate to the newly created `calculation` directory and execute the launcher script.

```
cd calculation
./launcher_orca.sh
```

7.2.3 Result Organization

Sorting and Summarizing
Upon completion of the calculations, the directory will be populated with thousands of output files. The `sort` command organizes these files into subfolders (`inputs`, `outputs`, `xyz`) and generates a summary of the computational wall time.

```
ascec sort
```

```
(py11) :~/.../water_hexamer/orca6/calculation$ ascec sort
================================================
ASCEC Sort Process Started
================================================

1. Sorting files by base names...
(...)
Processing: ./opt_conf_415/opt_conf_415.xyz
Processing: ./opt_conf_416/opt_conf_416.xyz

4. Creating calculation summary...
Processing 416 ORCA files in parallel...
Completed processing 416 ORCA files successfully.
Summary written to orca_summary.txt

5. Collecting .out files...
Copied 416 .out files to similarity_2/orca_out_416
```

Note: If the summary is not required (e.g., to save time on I/O operations), use the `--nosum` *flag to perform a quick sort only.*

```
head -n 20 summary_walltime.txt
```

```
Summary of all calculations:
Number of jobs: 416
Total execution time: 4:6:33.611
Mean execution time: 0:0:35.562
Shortest execution time: 0:0:19.172
Longest execution time: 0:1:39.882
Total wall time: 4 hours, 6 minutes

=> 1. opt_conf_400.out
energy = -30.49184195291
time = 0:0:19.172

=> 2. opt_conf_272.out
energy = -30.49391845293
time = 0:0:19.831
```

Listing 5: Example of Pre-optimization Summary

The pre-optimization of 416 structures was completed in approximately 4 hours, which is a reasonable time with room for improvement.

7.3 Similarity Clustering

7.3.1 Configuration Screening

The `sort` command (executed previously) extracts all valid output files (e.g., `.out` or `.log`) into a dedicated subdirectory. This prepares the dataset for the topological analysis.

```
/
├── annealing/
├── calculation/
├── similarity/
│   └── orca_out_416/
│       ├── opt_conf_1.out
│       ├── ...
│       └── opt_conf_416.out
└── w6.in
```

To initiate the topological analysis, navigate to the `similarity` directory. The script is executed using the `simil` command, which accepts two primary control arguments:

- `--th` **(Threshold):** Defines the clustering granularity. This value represents the maximum Euclidean distance in the physicochemical feature space for two structures to be considered part of the same motif.
- `-j` **(Jobs):** Specifies the number of CPU cores to use for parallel parsing of the output files. If not specified, the script will attempt to use all available cores.

Understanding the Threshold:
A lower threshold creates tighter, more homogeneous clusters (distinguishing subtle geometric variations), while a higher threshold yields broader, more inclusive families. The choice of threshold depends on the system size and the desired resolution:

Granularity	Value (τ)	Typical Use Case
Strict / Tight	`0.5 - 1.5`	**Small/Rigid Systems.** Distinguishes between specific rotamers or iso-energetic conformers with slight dihedral changes.
Standard	`2.0 - 2.5`	**General Purpose.** Default for molecular clusters (e.g., water hexamers). Balances distinct topology against vibrational noise.
Loose / Broad	`> 3.0`	**Large/Flexible Systems.** Groups broad structural families (e.g., folding patterns) while ignoring local side-chain fluctuations.

Table 3: Recommended clustering threshold (τ) values.

7.3.2 Execution and Directory Selection

To initiate the analysis, navigate to the `similarity` directory. The script offers two primary modes of execution:

A. Standard Physicochemical Clustering
Uses the feature vector approach to group structures based on electronic and energetic properties.

```
cd similarity/
simil --th=2 -j4
```

In this example, a standard threshold of $\tau = 2.0$ is applied using 4 CPU cores.

B. Hierarchical Two-Step Refinement
Activates the secondary geometric check.

```
simil --th=2 --rmsd=1 -j4
```

This command executes the hierarchical protocol. First, the algorithm performs the standard physicochemical screening ($\tau = 2.0$). Subsequently, it applies a geometric refinement within each identified family, subdividing structures that exceed a Root Mean Square Deviation (RMSD) of 1.0 Å. Used to distinguish stereoisomers or iso-energetic conformers

Interactive Selection
Upon execution, the script detects available directories containing output files. Select the target folder by entering its corresponding index number.

```
Found the following folder(s) containing quantum chemistry log/out files:

[1] orca_out_416 (Contains: .out)

Enter the number of the folder to process, or type 'a' to process all: 1

Extracting data from 416 files...
Using 4 CPU cores for parallel processing
Data extraction complete. Proceeding to clustering.
```

```
H-bond group 7: Clustering 81 configurations...

Cluster 1 (31 configurations)
Cluster 2 (11 configurations)
Cluster 3 (22 configurations)
```

7.3.3 Performance Optimization: Data Caching

The parsing of hundreds of quantum chemical log files is an I/O-intensive process that constitutes the primary wall-time bottleneck. To mitigate this, the script implements a **serialization mechanism**. Upon the first successful run, the parsed physicochemical descriptors are saved into a binary cache file (e.g., `data_cache_xxxxxx.pkl`).

```
/
├── orca_out_416/
└── data_cache_203971.pkl
```

Using the Cache
For subsequent analyses (e.g., testing different clustering thresholds), the script automatically detects and loads the binary cache, bypassing the heavy parsing stage entirely.

```
Attempting to load data from cache: 'data_cache_203971.pkl'
Using cached data
Data extraction complete. Proceeding to clustering.
(...)
```

Forcing Reprocessing
If the underlying output files have been modified or added to, the cache must be updated. Use the `--reprocess-files` flag to do this, or simply delete the cache.

```
simil --th=2 -j4 --reprocess-files
```

7.3.4 Cluster Representatives and Motifs

The algorithm groups structurally similar configurations based on physicochemical feature vectors. Within each identified cluster, the structure with the lowest potential energy is selected as the representative configuration for that cluster and marked as a unique motif.

```
Creating 50 motifs from cluster representatives...
Created motifs dendrogram: motifs_dendrogram.png
Motifs created: 50 representatives saved to motifs_50

Boltzmann distribution saved to 'boltzmann_distribution.txt'
Clustering summary saved to 'clustering_summary.txt'

Finished processing folder: orca_out_416

All selected molecular analyses complete!
```

7.4 Final Optimization Setup

Generating High-Level Input Files
Once the unique motifs have been identified, they typically require refinement at a higher level of theory (e.g., DFT with larger basis sets). The ASCEC `opt` command streamlines this process.

Similar to the `calc` step, this command scans the directory for coordinate files. However, instead of processing raw annealing results, it targets the **pre-processed configurations**.

```
ascec opt b97_orca.inp launcher_orca.sh
```

```
(py11) :~/.../water_hexamer/orca6$ ascec opt b97_orca.inp launcher_orca.sh

============================================================
Optimization XYZ file selection
============================================================

Combined files:
0) ./calculation/combined_results.xyz
1) ./calculation/combined_r3/combined_r3.xyz
2) ./similarity/motifs_53/all_motifs_combined.xyz

Motif files:
3) ./similarity/motifs_53/motif_01_opt_conf_228.xyz
4) ./similarity/motifs_53/motif_02_opt_conf_119.xyz
5) ./similarity/motifs_50/motif_03_opt_conf_91.xyz
```

*Select **Option 2** (`all_motifs_combined.xyz`) to create input files for the 53 detected motifs. Submit these files for higher-level optimization (e.g., B97-3c), then perform a second `sort` (creating the `similarity_2` directory) and similarity analysis. This workflow—advancing from raw configurations to preliminary motifs, and finally to optimized unique motifs—is required to obtain an accurate Boltzmann distribution.*

8 Output Files

A successful run of ASCEC with the annealing option ("1" in Line 1) with candidate structure clustering using the Similarity script for this example protocol will generate several files.

8.1 Annealing output

8.1.1 Main Log File (w6.out)

This file serves as the primary log, recording the complete history of the annealing protocol. To verify that the simulation completed successfully without runtime errors, users should inspect the final lines of the file (e.g., via the command `tail w6.out`). A successful run is confirmed by the presence of the termination flag:

```
** Normal annealing termination **
```

```
tail -n 15 w6_1.out
```

```
============================================================

** Normal annealing termination **
Total Wall time: 0 days 0 h 30 min 52 s 880 ms
Energy was evaluated 2919 times

Energy evolution in tvse_115889.dat
Configurations accepted by Max.-Boltz. statistics = 72
Accepted lower energy configurations = 55
Accepted configurations in result_115889.xyz = 127
Lowest energy configuration in rless_115889.out
Lowest energy = -0.51829172 u.a. (Config. 127)

============================================================
```

8.1.2 Reproducibility and Metadata

To ensure scientific rigor, ASCEC generates essential metadata for data validation and traceability. Each annealing run is initialized with a unique 6-digit random seed, which is used to seed all internal pseudorandom number generators (PRNGs). This mechanism ensures the stochastic independence of parallel runs, preventing statistical bias in the exploration of the potential energy surface. The seed is logged in the output file and appended to the names of key output files (e.g., result_957775.xyz, tvse_957775.dat), enabling clear identification and organization of results. Rerunning ASCEC with the same seed and input parameters will reproduce the same sequence of proposed molecular moves. However, exact reproduction of the full annealing trajectory additionally requires an identical software environment — the same Python and NumPy versions, as well as the same version, compilation, and parallelization settings of the external quantum-chemistry program (e.g., ORCA or Gaussian) — since numerical differences in energy evaluations, however small, can alter accept/reject decisions and cause trajectories to diverge.

8.1.3 Simulation Generated Data

I. Global Minimum Candidate (`rless_xxxxxx.out`)
This file contains the Cartesian coordinates corresponding to the lowest energy configuration visited during the entire annealing run. The suffix `xxxxxx` corresponds to the unique random seed of the run.

```
# Configuration: 141 | Energy = -71.65131472 u.a.
3
water1
O       3.114949    -2.572173     0.846802
H       3.740289    -3.084485     1.381054
H       2.849619    -3.185249     0.144893
3
water2
O      -1.728986     1.116733    -0.116378
H      -2.554571     0.817409    -0.525959
(...)
```

II. Energy Evolution Log (`tvse_xxxxxx.dat`)
This dataset records the thermodynamic history of the simulation. It is formatted in three columns: **Step Number**, **Temperature (K)**, and **Potential Energy (Hartree)**. Data is recorded exclusively for accepted Monte Carlo steps. This file is the source for generating the energy profile plots (see Figure 3) and the aggregate replica comparison (see Figure 4). These diagrams are automatically produced by the ASCEC script. To regenerate them manually, use the `diagram` command with the `--scaled` flag to auto-scale the Y-axis.

```
ascec diagram --scaled
```

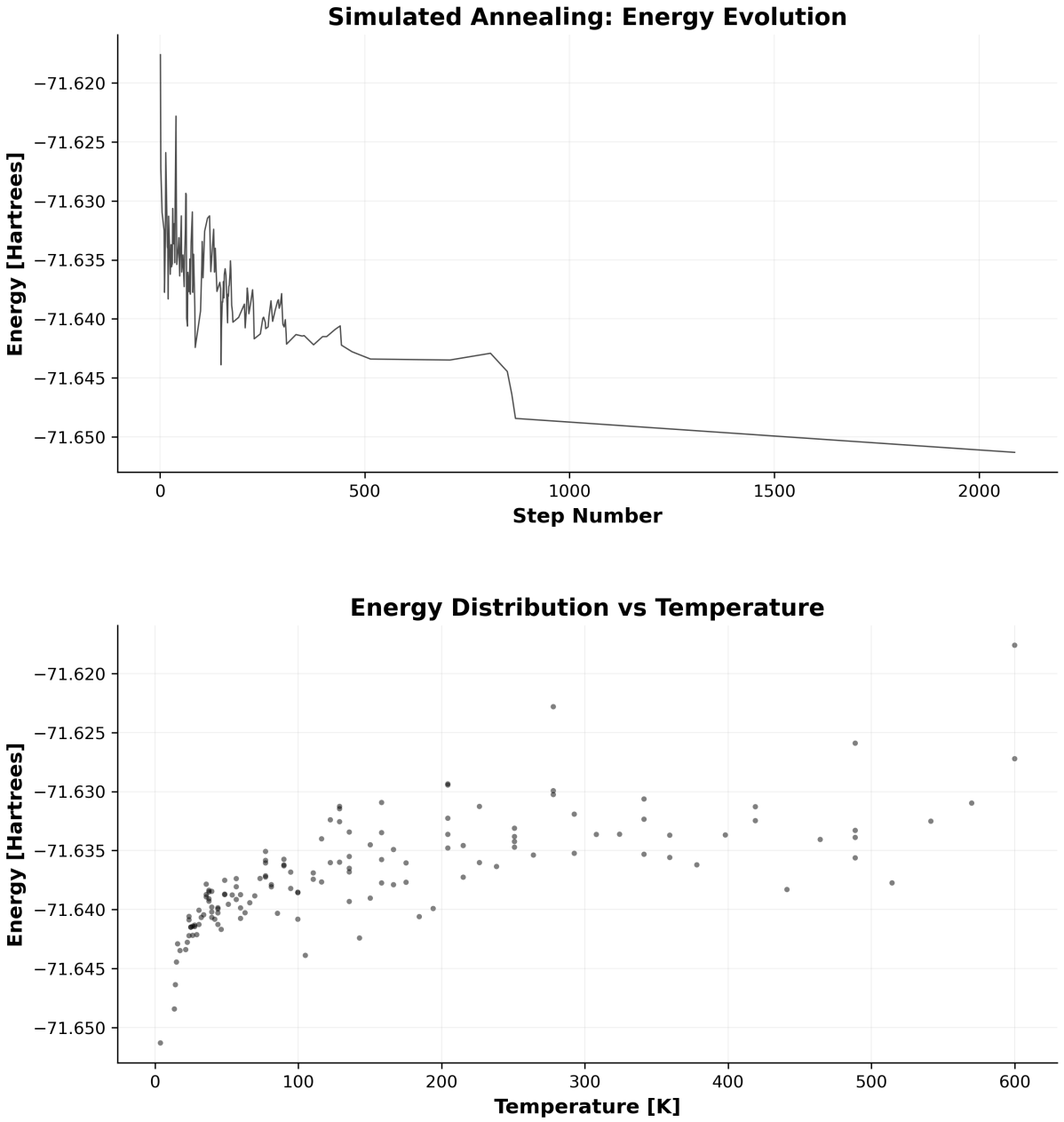

Figure 3: Single-run energy evolution profile (Temperature vs. Energy) generated by ASCEC.

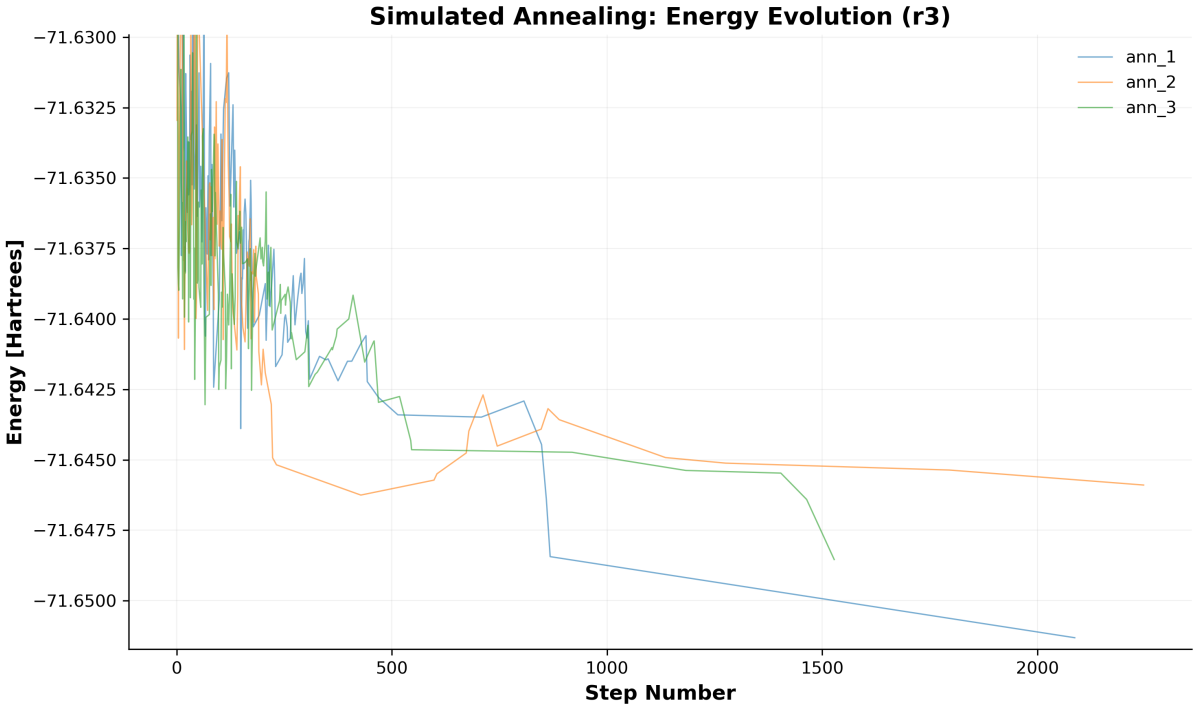

Figure 4: Combined energy profiles comparing multiple replica executions.

III. Structural Trajectory (`result_xxxxxx.xyz` | `.mol`)

These files contain the Cartesian coordinates of the entire ensemble of accepted structures. They serve as the full trajectory of the simulation, formatted for immediate visualization and animation using standard molecular editors (e.g., Avogadro2, Molekel, IQmol, or GaussView).

```
18
Configuration: 1 | E = -71.61760217 a.u. | T = 600.0 K
O       7.521687     8.123473     2.987348
H       7.729738     7.604909     3.779022
(...)
18
Configuration: 2 | E = -71.62722332 a.u. | T = 600.0 K
O       7.649736     8.853069     2.485145
H       8.312217     8.157098     2.610376
18
Configuration: 3 | E = -71.63098312 a.u. | T = 570.0 K
O       8.358924     7.910310     1.934682
H       9.138323     7.334893     1.954008
```

IV. Simulation Box Visualization (`resultbox_xxxxxx.xyz` | `.mol`)

Unless the `-nobox` flag is used, the program generates auxiliary files containing the simulation box vertices as dummy atoms. This allows for the visual verification of the confinement boundaries relative to the molecular system.

Compatibility Note:

- **Avogadro2:** Supports both `.xyz` and `.mol` formats for box visualization.
- **IQmol & GaussView:** Require the `.mol` file to correctly render the dummy atoms.

8.2 Calculation & Optimization Output

Once the process of calculating the generated input configurations and their sorting has been completed, the working directory (e.g., `calculation/` or `optimization/`) will typically contain the following key components:

- **Master Trajectories (`combined_results.xyz | .mol`):** Single master files containing the aggregated Cartesian coordinates for all processed configurations. These are generated to allow for the immediate visualization of the entire ensemble.

```
18
Coordinates from ORCA-job opt_conf_1 E -30.494666183120
O           4.89755049315488      6.57402652599668      3.11667076076455
H           4.49984221242737      6.65106539506547      4.01165830268415
(...)
18
Coordinates from ORCA-job opt_conf_2 E -30.489323286040
O           5.03567346983380      6.71360166724240      4.08623792011938
H           4.16925086054156      7.14879619512198      4.15686651181911
H           4.91370051289682      6.02838117620930      3.40385389606381
```

- **Job Subdirectories:** Dedicated folders containing the specific input/output files for each.
  - *In Calculation:* Naming depends on the input generation strategy:
    - **Option 'c' (Combine):** `opt_conf_#` (e.g., `opt_conf_1`). Numbering is sequential across the merged dataset.
    - **Option 'a' (Separate):** `opt#_conf_#` (e.g., `opt1_conf_1`). Explicitly identifies the source replica (e.g., Configuration 1 from Replica 1).
  - *In Optimization:* Named `motif_#` (e.g., `motif_01`).
- **Execution Summary (`*_summary.txt`):** A text file detailing the computational duration (wall time) and final energies for each calculation.
- **Launcher Script:** The shell script used to execute the batch jobs (e.g., `launcher_orca.sh`).

Exclusively in the **Calculation** directory, if the **Combine** strategy (Option `c`) was selected during setup, a file named `combined_r#` (e.g., `combined_r3`) is generated. This holds the raw merged data from the annealing replicas that were used to generate the initial input files.

8.3 Similarity Output

Once the clustering process concludes, the results are stored in a dedicated directory. Note that the directory creation is driven by the `sort` command, which automatically increments the suffix (e.g., `similarity_2`) to prevent overwriting. A typical output structure contains:

```
/
├── dendrogram_images/
├── extracted_clusters/
├── extracted_data/
├── motifs_53/
└── orca_out_416/
```

```
├── skipped_structures/
├── boltzmann_distribution.txt
├── clustering_summary.txt
└── data_cache_105638.pkl
```

8.3.1 Dendrograms

The dendrogram is the main visualization tool for understanding the topology of the sampled set. It represents a hierarchical tree in which the vertical axis (Y) indicates the **Euclidean distance** (dissimilarity) between configurations.

Understanding the Clustering Logic (Figure 5) The structure of the dendrogram directly informs the choice of the clustering threshold (τ). As illustrated in the specific case of the H-Bond=10 family in the final step, where three clusters were identified:

1. **Redundancy Identification:** Configurations **30** and **31** merge at a low height ($d < 1.0$). This implies they are geometrically and electronically equivalent.
2. **Cluster Differentiation (Intermediate):** Configuration **51** joins the (30+31) pair at a distance of $d \approx 3.5$. With a standard threshold of $\tau = 2.0$, this structure falls **above** the cutoff. Consequently, it is **not** grouped with the pair and is identified as a distinct motif.
3. **Global Separation:** Configuration **48** joins the ensemble at a much higher distance ($d \approx 5.8$). Since $5.8 \gg \tau$, it is recognized as a completely distinct basin of attraction, indicating significant structural and electronic differences.

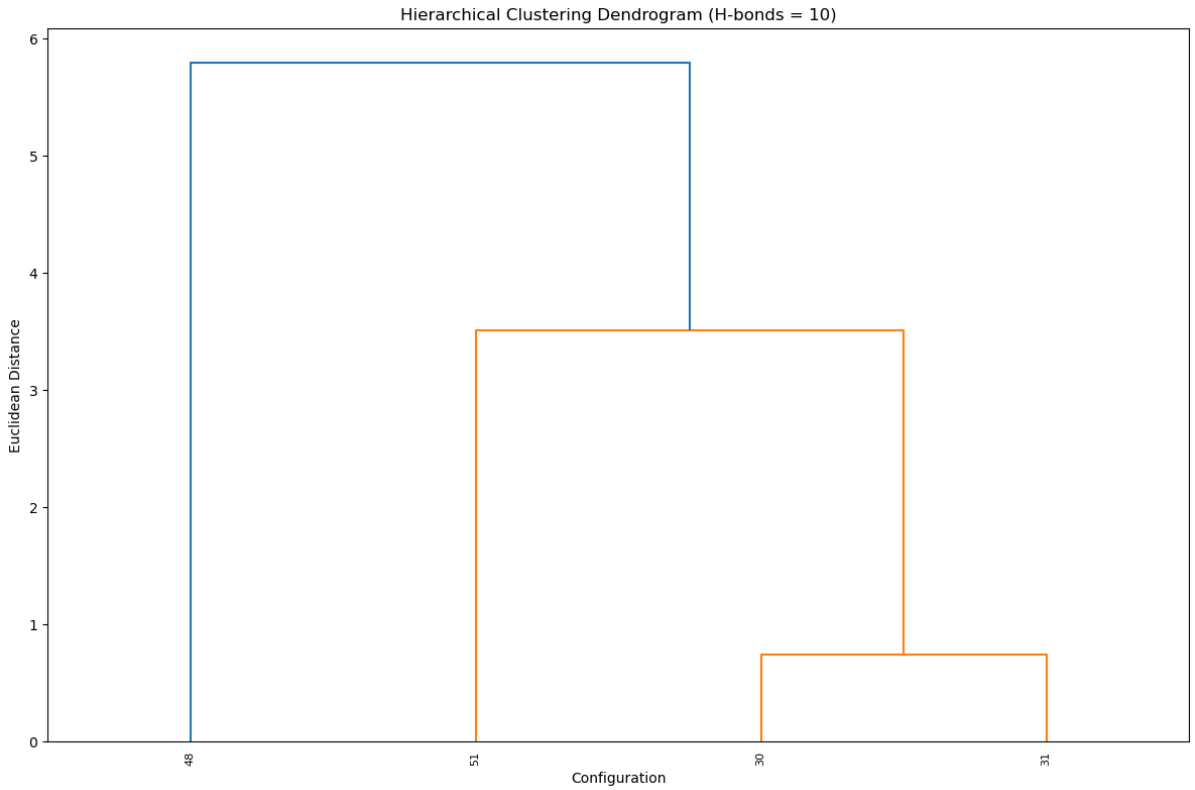

Figure 5: Detailed view of a dendrogram branch (H-bonds = 10). The vertical height of horizontal nodes indicates the dissimilarity between branches.

Family-Specific Trees: To simplify the analysis of complex systems, the Similarity module automatically splits the dataset based on the number of detected Hydrogen Bonds (HB), using geometric factors. A separate dendrogram is generated for each subset allowing for granular inspection of specific interaction networks.

Pre-Optimization Analysis (GFN2-xTB) Figure 6 displays the clustering results for the H-Bond=9 family following the initial semi-empirical pre-optimization.

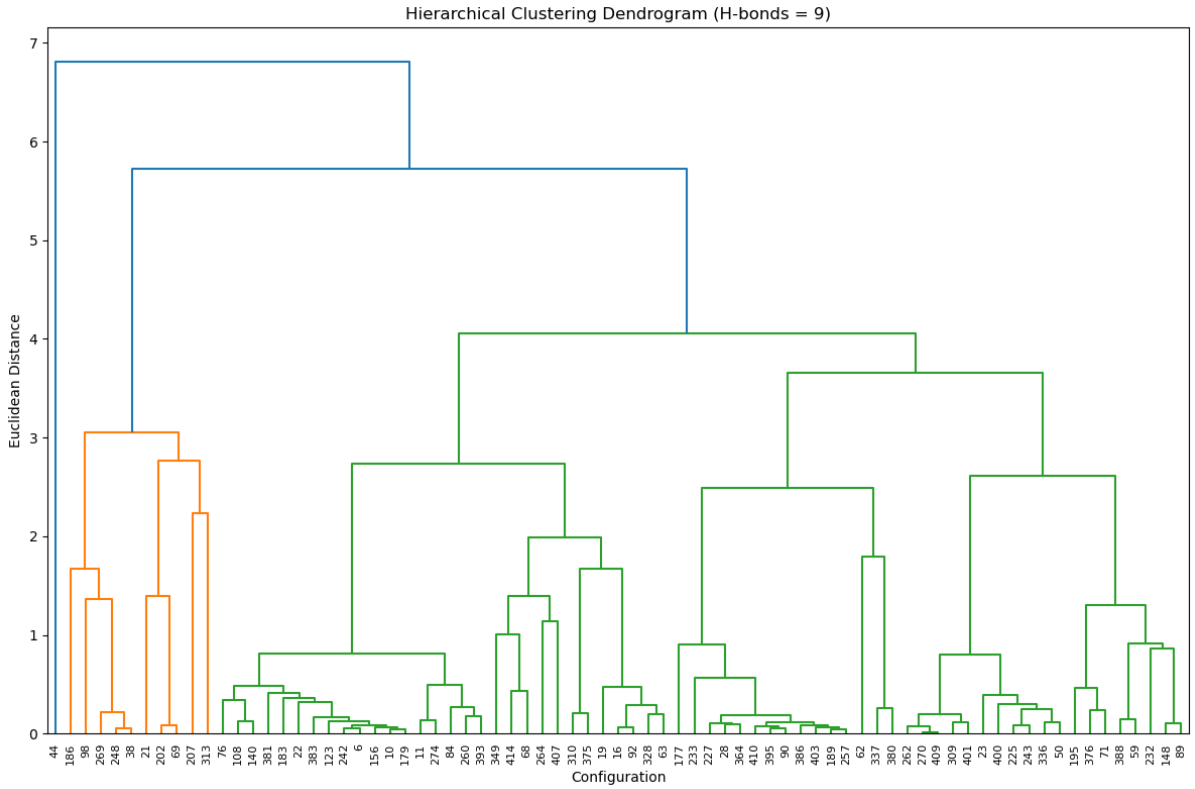

Figure 6: Dendrogram for the H-Bond=9 family following GFN2-xTB pre-optimization.

As shown in Figure 6, the tree exhibits a high density of leaves with deep, complex branching. The presence of distinct major families at large Euclidean distances indicates that the annealing process successfully located fundamentally different structural basins. However, the dense grouping at the bottom of the clusters suggests a high degree of redundancy.

Hierarchical Reduction Strategy:
In this specific example, the preprocessing step successfully reduced the ensemble from **416 raw candidates** to **53 motifs**. These representatives were subsequently re-optimized using a higher level of theory (B97-3c), followed by a secondary clustering analysis which yielded **20 unique motifs**. It is crucial to note that the final number of motifs diverges from the intermediate count, as the topology of the Potential Energy Surface (PES) is sensitive to the electronic structure method employed; distinct semi-empirical minima may merge into a single basin upon DFT refinement. Furthermore, for this example, we did not correct the structures that the script flagged as critical and requiring manual inspection, so this final figure could be slightly higher.

- Impact of critical structures on total dataset: 8/416 configurations (1.9%)
- Impact of critical structures on total dataset: 4/53 configurations (7.5%)

Final Optimization Analysis (DFT/B97-3c) Figure 7 presents the topological classification of the H-Bond=7 family after high-level refinement.

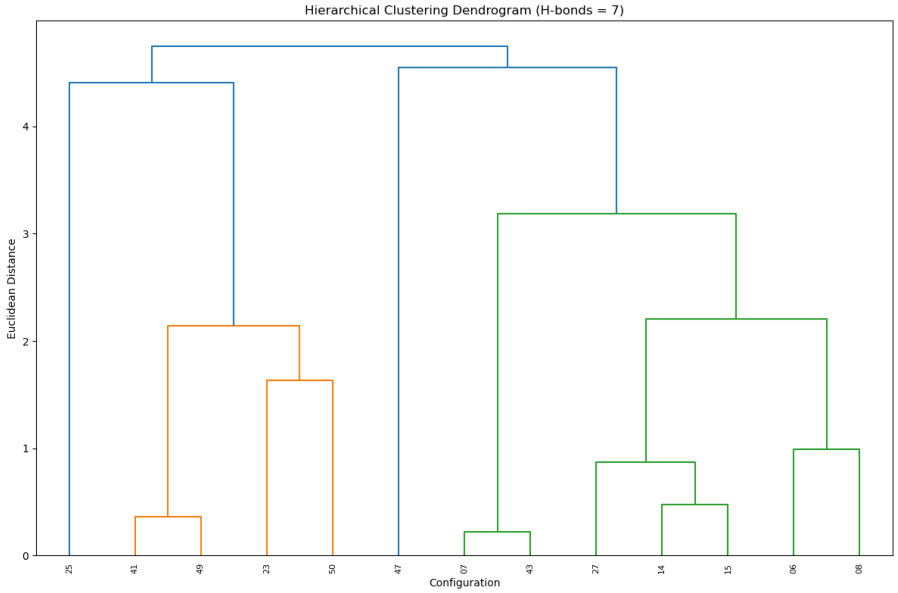

Figure 7: Dendrogram for the H-Bond=7 family following B97-3c final optimization.

As shown in Figure 7, the topology is significantly more structured than the pre-optimization stage. The clear vertical separation between nodes confirms that the high-level optimization has relaxed the structures into well-defined, distinct minima, facilitating the selection of a robust threshold for the final Boltzmann population analysis.

Motifs Dendrogram Figure 8 presents the topological classification of the motifs found.

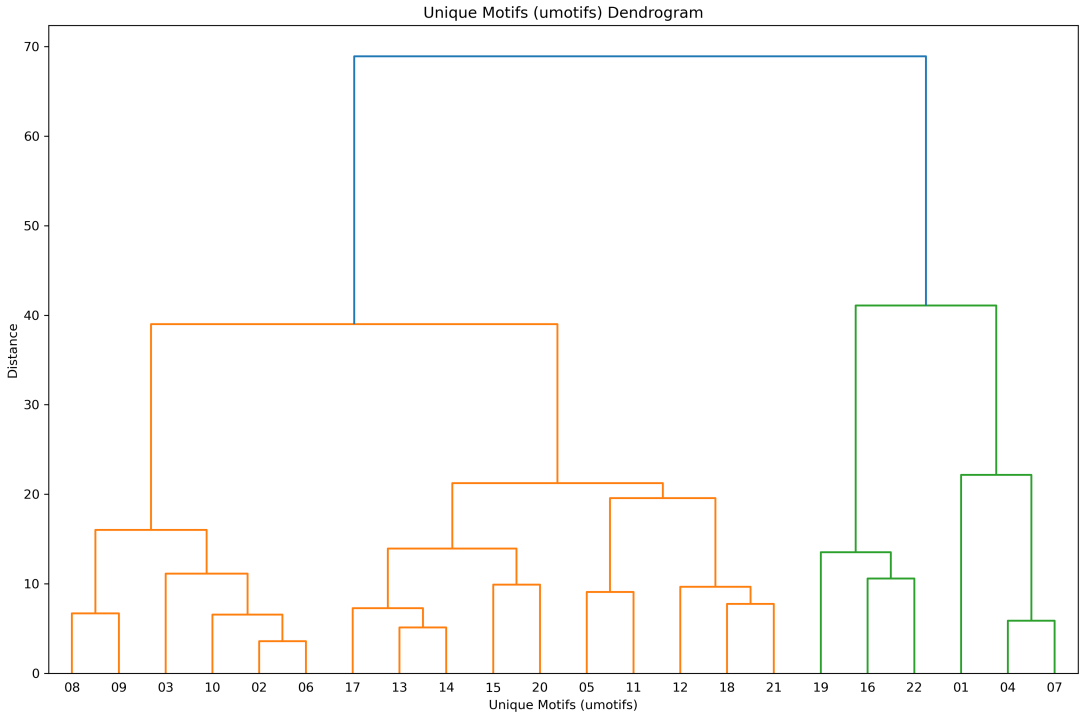

Figure 8: Dendrogram for the 22/23 unique motifs found after final optimization using B97-3c.

8.3.2 Clustering Summary Log

The script generates a comprehensive summary file (`clustering_summary.txt`) containing global statistics and detailed cluster breakdowns.

```
Clustering Results for: orca_out_416
Similarity threshold (distance): 2.0
Total configurations processed: 416
Total files skipped: 23 (5.5%)
Critical skipped files: 8 (1.9%)
Total number of final clusters: 53
```

Example of Non-Converged Output For a specific example similar to the H-Bond=10 family (shown above in Figure 5), with two clusters found, the summary report indicates "N/A" for Gibbs energy. This indicates that the associated ORCA/Gaussian jobs reached the maximum number of optimization cycles without converging, so the thermochemical calculation step was not performed. Even so, the script was able to analyze the last Cartesian coordinates and attempted to cluster them.

```
Hydrogen bonds: 10

Configurations: 4
Number of clusters: 2

Cluster 22 (3 configurations):
Files:
- motif_25_opt.out (Gibbs Energy: N/A)
- motif_26_opt.out (Gibbs Energy: N/A)
- motif_31_opt.out (Gibbs Energy: N/A)

Cluster 23 (1 configurations):
Files:
- motif_42_opt.out (Gibbs Energy: N/A)
```

Valid Cluster Analysis For successfully converged systems, the summary provides precise thermodynamic data. Below is an example from the H-Bond=7 family, showing a clear separation between a singleton cluster (Cluster 6) and a grouping of two configurations (Cluster 5).

```
Hydrogen bonds: 7

Configurations: 13
Number of clusters: 7

Cluster 5 (2 configurations):
Files:
- motif_23_opt.out (Gibbs Energy: -458.357020 Hartree (-287623.37 kcal/mol))
- motif_50_opt.out (Gibbs Energy: -458.356211 Hartree (-287622.87 kcal/mol))

Cluster 6 (1 configurations):
Files:
- motif_25_opt.out (Gibbs Energy: -458.356863 Hartree (-287623.27 kcal/mol))
```

8.3.3 Error Handling and Diagnostics

The Similarity module implements a comprehensive diagnostic protocol to ensure ensemble integrity. It classifies files based on termination status, convergence, and vibrational topology:

1. Anomaly Classification The algorithm detects three specific types of problematic structures, which can be found in the output files:

- **Execution Failures:** Files lacking the "Normal Termination" flag (e.g., due to crash).
- **Non-Converged Geometries:** Files that did not reach a stationary point within the maximum optimization cycles despite normal termination.
- **Topological Instabilities:** Structures containing **one or more imaginary frequencies**. This includes first-order saddle points (Transition States), higher-order instabilities (Hilltops), or soft modes (e.g., unrelaxed methyl rotations).

2. Filtering Logic To balance data retention with redundancy reduction, anomalies are processed via clustering:

- **Skipped Files:** Files that cannot be parsed or failed execution are automatically skipped. Additionally, structures with imaginary frequencies are skipped if they cluster tightly ($d <$ threshold) with a known true minimum, as these are considered "noisy" versions of a successfully identified basin.
  - **Flagged as Critical:** Any structure with imaginary frequencies that forms an **isolated cluster** (distinct from valid minima) is preserved and flagged as **Critical**. These are stored separately as they potentially represent:
    - Unique Transition States (TS).
    - Distinct local minimum that failed to converge purely due to numerical issues.
    - Valid basin with a flat potential surface (soft modes).

    Action Required: These files are exported for review and generally require manual inspection or re-optimization.

```
/
└── skipped_structures/
    ├── clustered_with_minima/
    │   ├── motif_26_opt.out
    │   └── motif_26_opt.xyz
    ├── need_recalculation/
    │   ├── combined_need_recalc.xyz
    │   ├── motif_11_opt.out
    │   └── motif_11_opt.xyz
    └── skipped_summary.txt
```

8.3.4 Generated Cluster Data

Beyond the summary, the script populates two key directories for detailed inspection:

1. `extracted_clusters/`: Contains subfolders for every identified cluster (e.g., `cluster_3_11` denotes Cluster #3 containing 11 configurations).

   Inside, the individual or combined (`.xyz | .mol`) files allow for visual superposition and verification of the cluster.

2. `extracted_data/`: Contains statistical data files (e.g., `cluster_15_4.dat`). These text files list the raw physicochemical descriptors (dipole moments, rotational constants, energies) for every member of the group, enabling quantitative analysis of the cluster's homogeneity, along with weighting and tolerances.

```
Cluster (represented by: cluster_15_4) (4 configurations)

- opt_conf_239.out
- opt_conf_287.out
- opt_conf_322.out
- opt_conf_53.out

Deviation Analysis (Max-Min / |Mean|):
Gibbs Free Energy (Hartree) %Dev: 0.00%
HOMO-LUMO Gap (eV) %Dev: 0.04%
Dipole Moment (Debye) %Dev: 2.80%
Radius of Gyration (Å) %Dev: 0.21%
Rotational Constant A (GHz) %Dev: 0.49%
Rotational Constant B (GHz) %Dev: 0.49%
Rotational Constant C (GHz) %Dev: 0.62%
Last Vibrational Frequency (cm^-1) %Dev: 0.02%
Number of Hydrogen Bonds %Dev: 0.00%

Clustering Weights Applied:
Gibbs Free Energy: 1.00

Clustering Absolute Tolerances Applied:
Gibbs Free Energy: 0.000005
```

8.3.5 Boltzmann Distribution

After optimization and clustering, the `boltzmann_distribution.txt` file is generated, ranking unique motifs by their Gibbs Free Energy at 298.15 K. While this reference temperature can be modified via the `-T=#` flag, the thermochemical properties must be recomputed accordingly to maintain accuracy. As illustrated below, the global minimum (UMotif 01) and the first local minimum (UMotif 02) are nearly iso-energetic, comprising more than 50% of the population.

```
Population by Energy Minimum
(assuming non-degeneracy, gi = 1)

Unique Motif (umotif) Assignment Summary
(sorted by Boltzmann population)

cluster_15 (umotif_01)
From structure: motif_03_opt
Gibbs Energy: -458.360815 Hartree (-287625.75 kcal/mol, -12472.63 eV)
Population: 26.77 %

cluster_1 (umotif_02)
From structure: motif_06_opt
Gibbs Energy: -458.360808 Hartree (-287625.75 kcal/mol, -12472.63 eV)
Population: 26.56 %
```

9 Workflow Mode

The automated protocol enables the execution of complex, multi-stage pipelines through a single directive. Specifically configured for the **ORCA** package. Unlike standalone execution—which requires user intervention to filter and resubmit files—the Workflow Mode autonomously manages the transition between annealing, optimization, and clustering. It includes built-in logic to correct problematic structures (e.g., imaginary frequencies), significantly reducing the manual overhead required to locate global minima.

9.1 Pipeline Logic

Figure 9 delineates the execution logic of the automated pipeline for a standard two-step refinement protocol. It illustrates the iterative progression from initial annealing to final refinement, including the automated feedback loops for correcting critical failures.

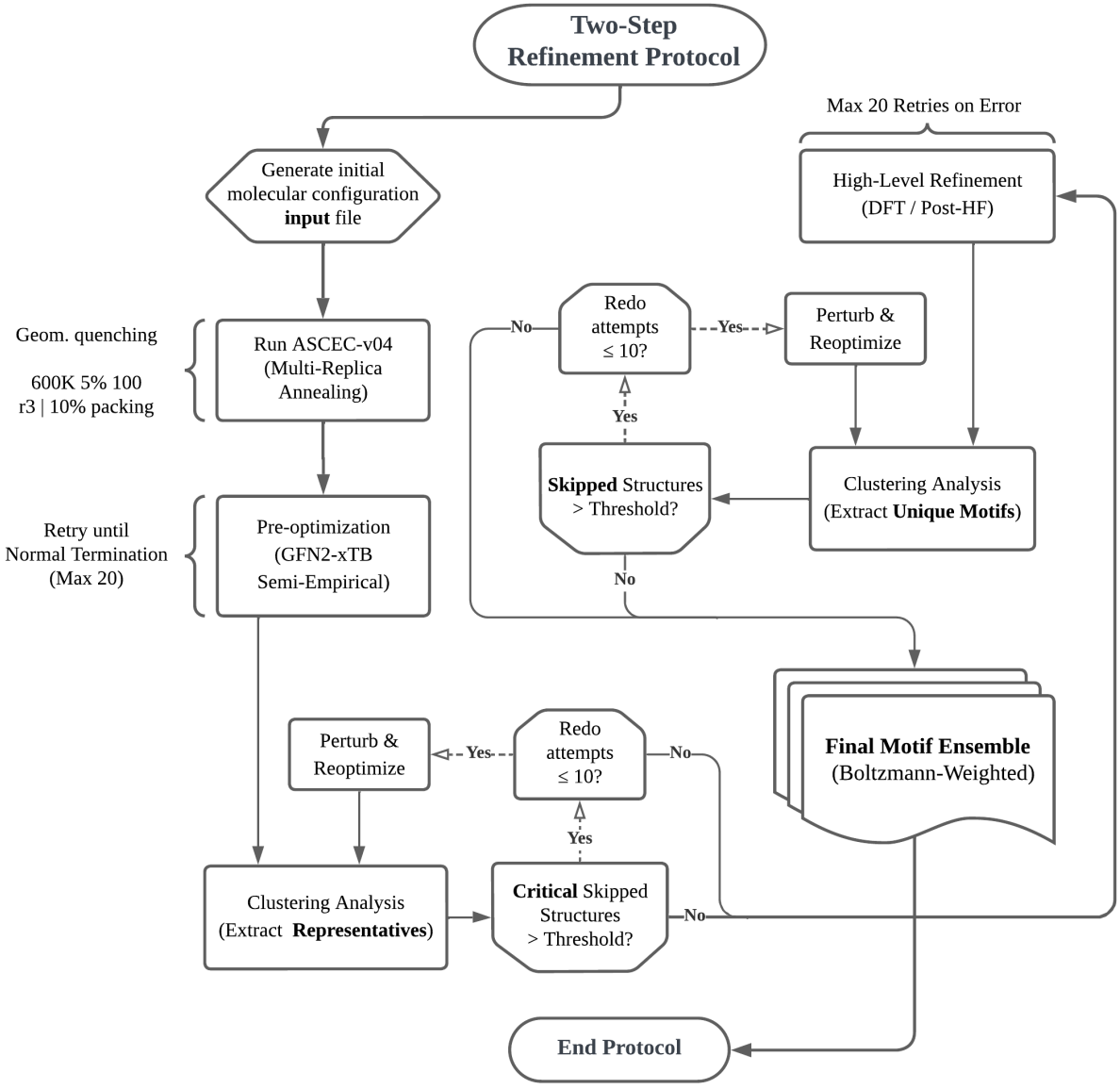

Figure 9: Schematic representation of the Two-Step Refinement Protocol.

9.2 Workflow Input (formic.in)

To illustrate this functionality, we consider a conformational search for the formic acid dimer using **ORCA 6.1.1**. Starting from the high-energy *cis* dimer configuration, the workflow is tasked with automatically locating the experimentally observed *trans* global minimum. The following file represents the complete configuration, defining molecular parameters, simulation settings, and the multi-stage protocol.

```
# Input for formic acid dimer Simulation

1 10            # Line 1: Simulation Mode & Number of Config
8               # Line 2: Simulation Cube Length (Angstroms)
2               # Line 3: Annealing Quenching route
100.0 10.0 50   # Line 4: Linear Quenching Parameters
600.0 5.0 100   # Line 5: Geometric Quenching Parameters
3000  50        # Line 6: Maximum Monte Carlo Cycles per T and floor value
1.0 1.0         # Line 7: Maximum Displacement (A) & Rotation (radians)

30 60           # Line 8: Conformational sampling (%)
                # & Maximum dihedral rotation (degrees)
2 orca          # Line 9: QM Program Index & alias
pm3             # Line 10: Hamiltonian & Basis Set
1 8             # Line 11: nprocs (QM calculations and ASCEC evaluation)
0 1             # Line 12: Charge & Spin Multiplicity

2               # Line 13: Number of Molecules (nmo)

# Lines below: Molecule Definition

*
5
Formic_acid1
C    0.14624609581422     -0.35536580959883     -0.00000002457506
O    1.17429417535772      0.24095314517570      0.00000001235630
H    0.07493003024184     -1.46799267964298      0.00000000612430
O   -1.05240386842550      0.23715532959413     -0.00000000061204
H   -1.79206643298827     -0.38334998552802      0.00000000670651
*
5
Formic_acid2
C    0.14624609581422     -0.35536580959883     -0.00000002457506
O    1.17429417535772      0.24095314517570      0.00000001235630
H    0.07493003024184     -1.46799267964298      0.00000000612430
O   -1.05240386842550      0.23715532959413     -0.00000000061204
H   -1.79206643298827     -0.38334998552802      0.00000000670651
*

# Protocol
# (annealing, (pre)optimization, clustering, separated by "," or ".")

.in,
r3 --box10,
calc --critical=0 --retry=20 --redo=10 ./preopt_gfn2.inp ./launcher_orca.sh,
similarity --th=2 -j8,
opt --skipped=0 --retry=20 --redo=10 ./opt_wb97x.inp ./launcher_orca.sh,
similarity --th=2 -j4
```

9.3 Protocol Command Syntax

The specific commands used in the `Protocol` section of the input file are detailed below.

Command / Flag	Description
`.in`	**Workflow Initiator.** Designates the file as a multi-stage workflow script.
`r3 -box10`	**Annealing.** Executes a triplicate ($r = 3$) annealing process. The `-box10` flag sets the effective packing box length to 10%.
`calc ... ./preopt_gfn2.inp`	**Pre-optimization.** Runs the preprocessing stage.
–critical=0	**Critical Tolerance.** Sets the allowance for "Critical" structures (isolated imaginary frequencies) to 0%, forcing the workflow to resolve all such cases.
–retry=20	**Execution Retries.** Automatically retries failed jobs (e.g., crashes) up to 20 times.
–redo=10	**Convergence Fixes.** Attempts to correct imaginary frequencies (e.g., saddle points) up to 10 times.
`similarity -th=2 -j8`	**Clustering.** Groups structures using a similarity threshold of 2.0, utilizing 8 CPU cores (`-j8`) for parallel processing.
`opt ... ./opt_wb97x.inp`	**Optimization.** Performs high-level DFT optimization on the representative motifs identified in the previous stage.
–skipped=0	**Strict Convergence.** Sets the tolerance for "Skipped" (noisy/unconverged) files to 0%. Requires that all inputs successfully converge to valid minima.
`similarity -th=2 -j4`	**Final Analysis.** Clusters the optimized structures to identify unique motifs and generate the final Boltzmann-weighted ensemble.

Table 4: Description of commands and flags used in the automated workflow.

9.4 Execution and Outputs

The automated protocol is initiated via the following command:

```
ascec formic.in protocol
```

9.4.1 Prerequisites and File Structure

Prior to execution, specific configuration and template files must be present in the working directory. These templates must adhere to the syntax defined in Section 7 (see 7.2.1), utilizing specific placeholders for dynamic data injection:

- `#name`: (Optional) Replaced by the specific filename to ensure traceability in output logs.
- `#`: Indicates the insertion point for the Cartesian coordinate block.
- `###`: Designates the location within the launcher script where serial execution commands will be injected.

Upon initialization, the system parses these files and generates a structured directory tree for each stage of the workflow.

```
Annealing → Calculation → Similarity → Optimization → Similarity(2)
```

Required Input Files	Generated Output Structure
/ ├── formic.in ├── launcher_orca.sh ├── opt_wb97x.inp └── preopt_gfn2.inp	/ ├── annealing/ ├── calculation/ ├── optimization/ ├── similarity/ ├── similarity_2/ ├── protocol_425319.pkl └── protocol_summary.txt

9.4.2 Terminal Output

When the workflow is initiated, the system parses the protocol and begins the first stage:

```
Protocol mode activated
Creating new protocol cache: protocol_425319.pkl (for formic.in)
Resuming workflow

Workflow: Annealing → Calculation → Similarity → Optimization → Similarity

------------------------------------------------------------
[1/5] Annealing
------------------------------------------------------------
Using recommended box size: 8.0 Å (10.0% effective packing)
Creating 3 replicated runs in 'annealing/'...
Created: formic_1/formic_1.in
Created: formic_2/formic_2.in
Created: formic_3/formic_3.in
Running 3 annealing simulation(s)

formic_1...
```

9.5 Execution Control and Resumption

The workflow allows for granular control over the execution pipeline via delimiters and caching.

9.5.1 Stage Delimiters

- **Continuous Execution (,):** A comma at the end of a protocol line indicates that the script should proceed immediately to the next stage upon completion.
- **Breakpoints (.):** A period (or dot) acts as a breakpoint. The workflow will complete the current stage and then pause, allowing the user to manually inspect intermediate files (e.g., to review **Critical** structures).

9.5.2 Resuming the Workflow

The system maintains a cache file (e.g., `protocol_425319.pkl`) to track progress. If a workflow is paused via a breakpoint or interrupted, it can be resumed from the last successful stage using the same command:

```
ascec formic.in protocol
```

This ensures that expensive calculations are not repeated unnecessarily.

Resuming the Pipeline
Users can direct the workflow to start from a specific stage index:

`1.Annealing` → `2.Calculation` → `3.Similarity` → `4.Optimization` → `5.Similarity2`

Force Restart: To start over from the beginning of the Calculation stage (2):

```
ascec formic.in protocol 2
```

Resume Incomplete: To resume the Calculation stage (2) preserving valid progress:

```
ascec formic.in protocol 2 -i
```

9.6 Failure Recovery Mechanisms

The workflow incorporates robust fault-tolerance mechanisms to handle the complex potential energy surfaces often encountered in difficult systems.

9.6.1 Retry Logic (Execution Reliability)

This protocol addresses execution failures where a calculation does not achieve "Normal Termination" (e.g., non-zero exit codes or crashes). The script attempts to recover using one of two strategies: restarting the calculation from scratch or resuming from the last recorded coordinates. This feature is designed to mitigate transient execution errors, including filesystem disruptions and operational noise. It cannot correct systematic errors arising from incorrect input syntax; therefore, the ORCA input templates must be pre-validated for the system under study.

```
------------------------------------------------------------
[2/5] Calculation
------------------------------------------------------------
Executing calculations...

Running: opt_conf_1.inp... ✓ (2nd Attempt)
Running: opt_conf_2.inp... ✓
...
Running: opt_conf_495.inp... ✓
Running: opt_conf_496.inp... ✓

Status: 496/496 calculations completed
All calculations completed successfully
```

9.6.2 Redo Logic (Convergence and Topology)

The final ensemble must consist exclusively of **true local minima**. To ensure no optimized structure retains imaginary frequencies, the Redo logic automatically intervenes when topological instabilities are detected:

1. **Multiple Imaginary Frequencies:** Treated as numerical convergence artifacts. The system attempts to resolve these by re-submitting the optimization using the final coordinates of the previous run.
2. **Single Imaginary Frequency:** Treated as first-order saddle points (Transition States) or soft vibrational modes. The algorithm **perturbs the geometry along the imaginary vibrational mode** to push the structure toward a stable basin before re-optimizing.

Computational Cost Warning: Since this process involves high-level refinement, it is computationally expensive. It is intended to resolve soft modes and saddle points, not to compensate for an inappropriate method.

```
------------------------------------------------------------
[4/5] Optimization
------------------------------------------------------------
Stage redo enabled: max 10 attempts, target skipped ⩽ 0.0%

Executing optimization calculations...
Running: motif_01_opt.inp... ✓
...
Running: motif_35_opt.inp... ✓

Status: 35/35 optimizations completed
Optimization calculations completed

------------------------------------------------------------
[5/5] Similarity
------------------------------------------------------------
Using similarity script: similarity-v01.py

Working directory: similarity_2

Processed 2 structures with imaginary frequencies:
- 0 clustered with true minima (can be ignored)
- 2 may represent missing motifs (need recalculation)
  motif_25_opt.out  -  1 imaginary freq(s)
  motif_12_opt.out  -  2 imaginary freq(s)

Finished processing folder: orca_out_35
All selected molecular analyses complete!

Results: 5.7% critical, 5.7% skipped
→ Threshold exceeded (skipped 5.7% > 0.0%)
```

Clarification of Metrics: The **Critical** set is a specific subset of the **Skipped** set. While all structures containing imaginary frequencies are automatically classified as **Skipped**, the **Critical** designation is reserved exclusively for those that form **isolated motifs**.

```
------------------------------------------------------------
Redo attempt 2/10

Processing redo structures from: similarity_2/skipped_structures
Found 2 structure(s) to retry

motif_12_opt: 2 imaginary freqs, using final geometry ✓
motif_25_opt: 1 imaginary freq, displacing along mode ✓

Regenerated 2 input file(s) with new geometries
Resuming: Using existing optimization directory (Redo Mode)

Executing optimization calculations...
Resuming: 33/35 optimizations already completed
Running: motif_12_opt.inp... ✓
Running: motif_25_opt.inp... ✓

Status: 35/35 optimizations completed

Using similarity script: similarity-v01.py
python /.../similarity-v01.py --th=2 -j4 --update-cache /tmp/tmpktk17rmd.txt
```

10 Results and Validation

This section presents validation case studies demonstrating the efficacy and robustness of the ASCEC & Similarity protocols. We evaluate the software across three distinct dimensions:

1. **Standalone Execution:** A comparative benchmark of a "Two-Step" hierarchical approach (ORCA) versus a "Direct" optimization strategy (Gaussian 09) using the Water Hexamer $(H_2O)_6$.
2. **Automated Workflow:** A demonstration of the fully automated pipeline utilizing the *cis*-Formic Acid Dimer.
3. **Clustering Algorithms:** An assessment of standard topological clustering versus RMSD-refined hierarchical clustering.

10.1 Standalone Execution: Water Hexamer

To validate the modular utility of the software, the Water Hexamer system (`w6.in`) was analyzed using a manual execution strategy. This case study benchmarks the trade-off between computational cost and conformational space coverage.

10.1.1 Hierarchical Two-Step Protocol (ORCA)

This protocol employed a cost-effective pre-optimization filter using the semiempirical GFN2-xTB method, followed by high-level DFT refinement. The workflow processed an initial stochastic pool of **416 raw annealing candidates**.

1. Geometric Reduction (GFN2-xTB): The Similarity module reduced the raw input of 416 structures to **53 representative configurations** based on topological distinctness. Figure 10

illustrates the ensemble members of a specific cluster (`cluster_1_24`), while Figure 11 displays its representative structure.

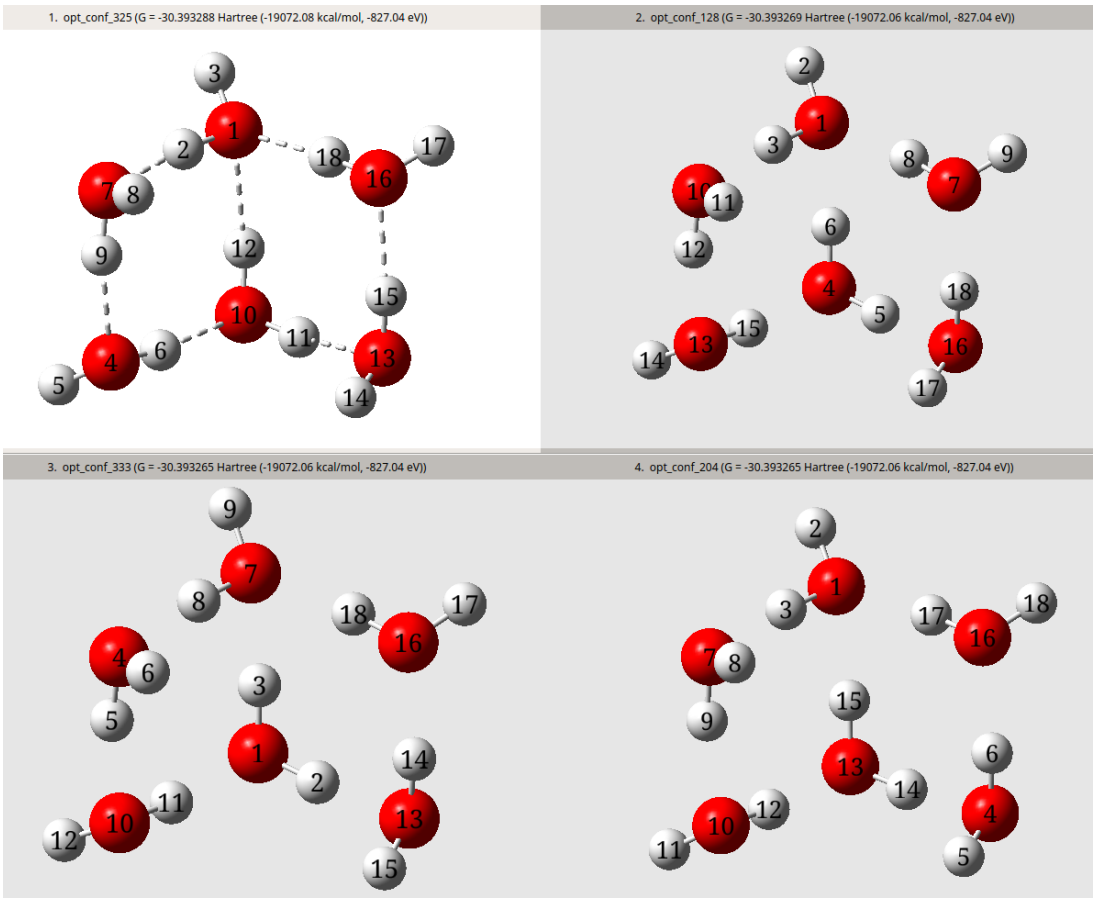

Figure 10: Visual inspection of the `cluster_1_24` ensemble members derived from the initial annealing pool.

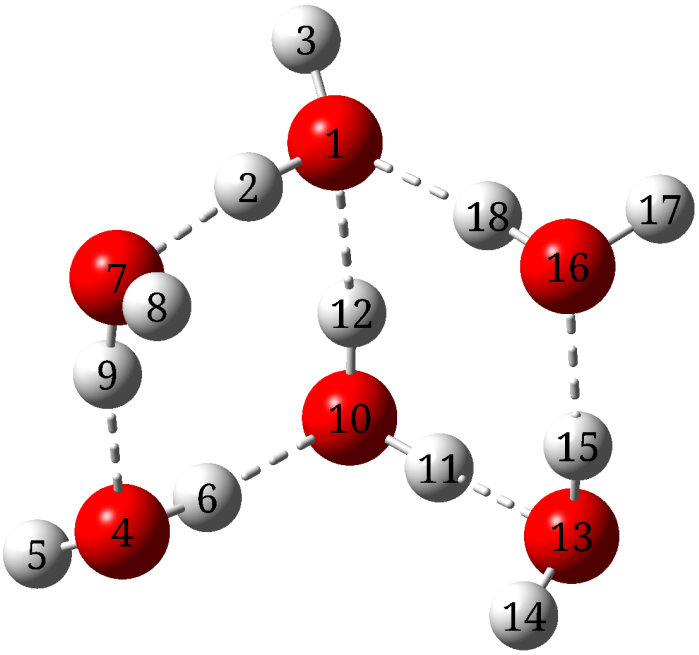

Figure 11: Representative configuration for `cluster_1_24`.

2. High-Level Refinement (B97-3c): The 53 representatives underwent full DFT optimization (B97-3c). A final similarity check (Threshold $\tau = 2.0$) consolidated these into **20 Unique Motifs**. Some of them can be seen in Figure 12. In addition, Figure 13 highlights the three most statistically significant structures.

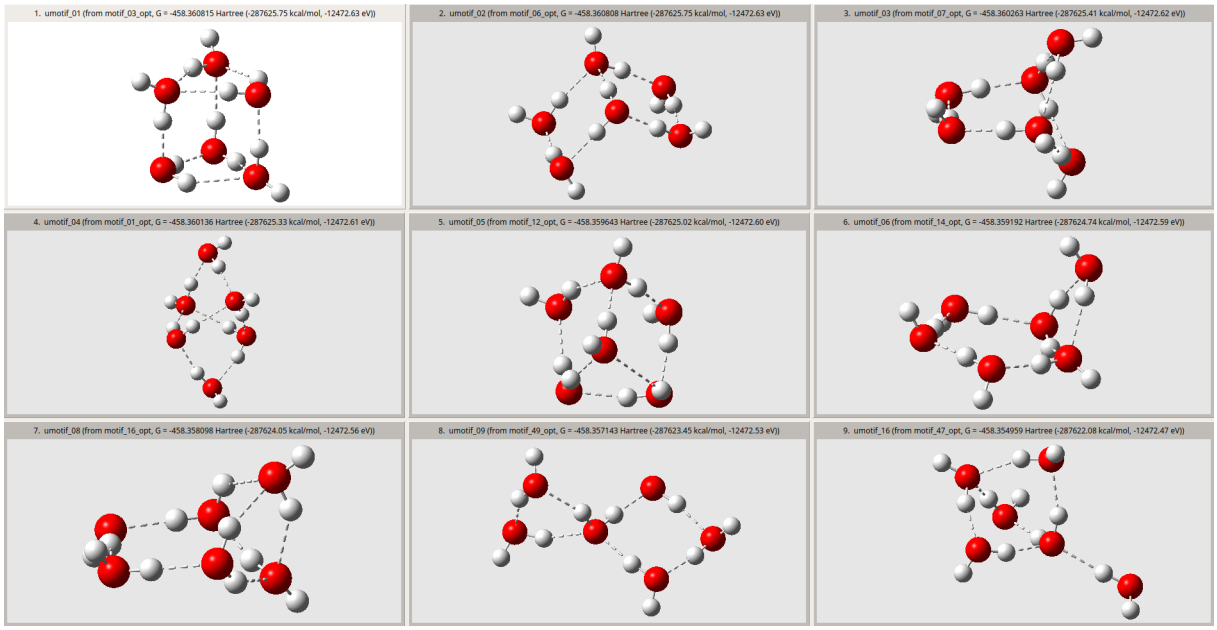

Figure 12: Subset of unique motifs identified for the water hexamer system via the Two-Step protocol.

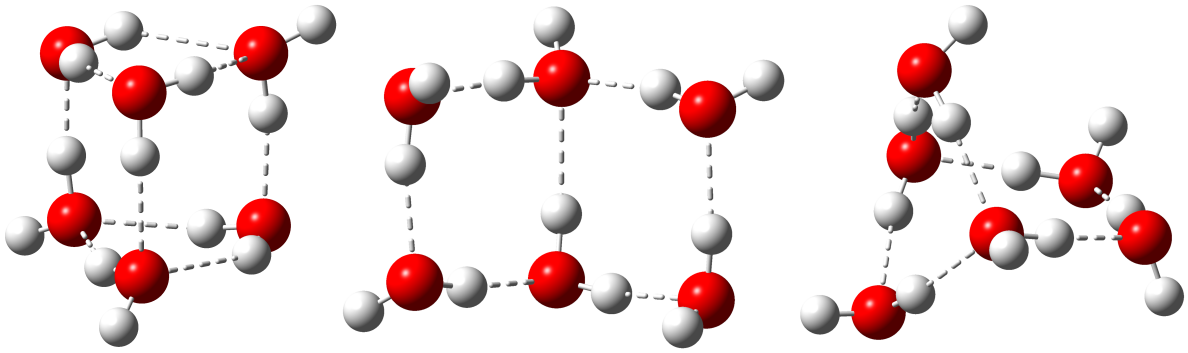

Figure 13: The three dominant water hexamer motifs identified by ASCEC.

Boltzmann Distribution: Thermodynamic analysis at 298.15 K indicates that the top three motifs account for over **68%** of the total ensemble population. The global minimum (Prism-like, UMotif 01) and the first local minimum (Book-like, UMotif 02) are effectively iso-energetic.

```
Unique Motif (UMotif) Assignment Summary
(Sorted by Boltzmann population - 298.15 K)

UMotif_01 (cluster_15)
Gibbs Energy: -458.360815 Hartree (-287625.75 kcal/mol)
Population:   26.35 %

UMotif_02 (cluster_1)
Gibbs Energy: -458.360808 Hartree (-287625.75 kcal/mol)
Population:   26.14 %

UMotif_03 (cluster_2)
Gibbs Energy: -458.360263 Hartree (-287625.41 kcal/mol)
Population:   14.69 %
```

10.1.2 Direct Approach (Gaussian 09) & Comparative Analysis

The Gaussian 09 validation employed a direct optimization strategy at the wB97XD/6-31+G(d) level of theory, skipping the semiempirical pre-screening. Comparing the Hierarchical and Direct method approaches reveals critical insights regarding Potential Energy Surface (PES) sampling and computational efficiency.

1. Topological Accuracy and Semiempirical Bias Both methods successfully identified the dominant low-energy isomers (Prism, Cage, Book). However, a detailed inspection of the full ensembles reveals that the semi-empirical pretreatment (GFN2-xTB) exhibits a specific topological bias that must be taken into account.

GFN2-xTB tends to over-stabilize cooperative hydrogen bonding networks, prioritizing structures where water molecules act as both donors and acceptors. Consequently, configurations containing "pure acceptor" water molecules (which do not donate hydrogen) are occasionally filtered out during the pre-optimization step. Figure 14 illustrates such a structure, which was found by the direct approach but excluded by the two-step protocol.

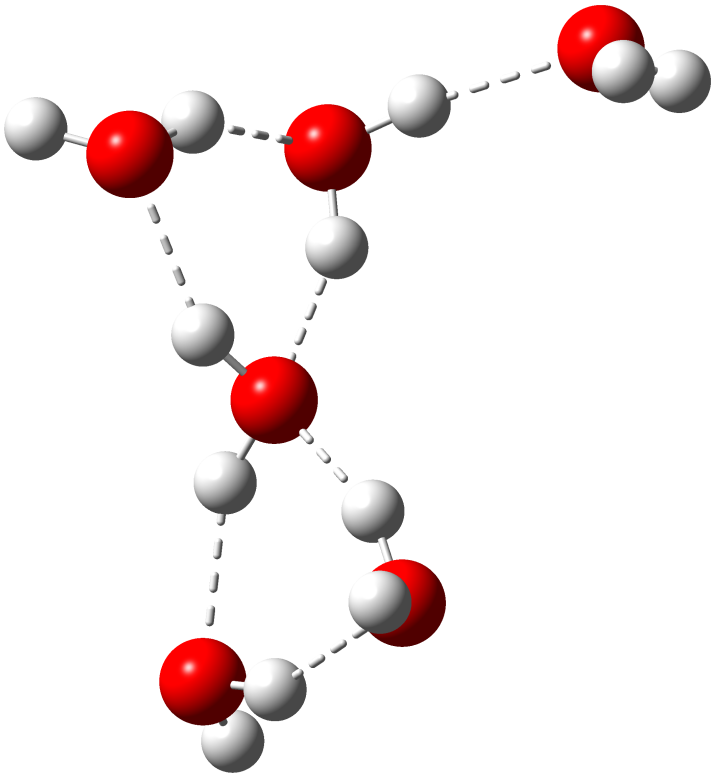

Figure 14: A specific water hexamer configuration identified only by the Direct Approach (Gaussian 09), characterized by a pure-acceptor water molecule.

Despite this topological divergence, the thermodynamic impact is negligible. The "missing" structures possess high relative energies and contribute $\approx 0\%$ to the Boltzmann distribution at 298.15 K. Furthermore, comparisons with high-level benchmarks (Figure 15) confirm that the ASCEC B97-3c structures closely replicate the MP2/6-311++G** geometries reported by Hincapié *et al.* , in cases where the presence of pure-acceptor water molecules is not mandatory.

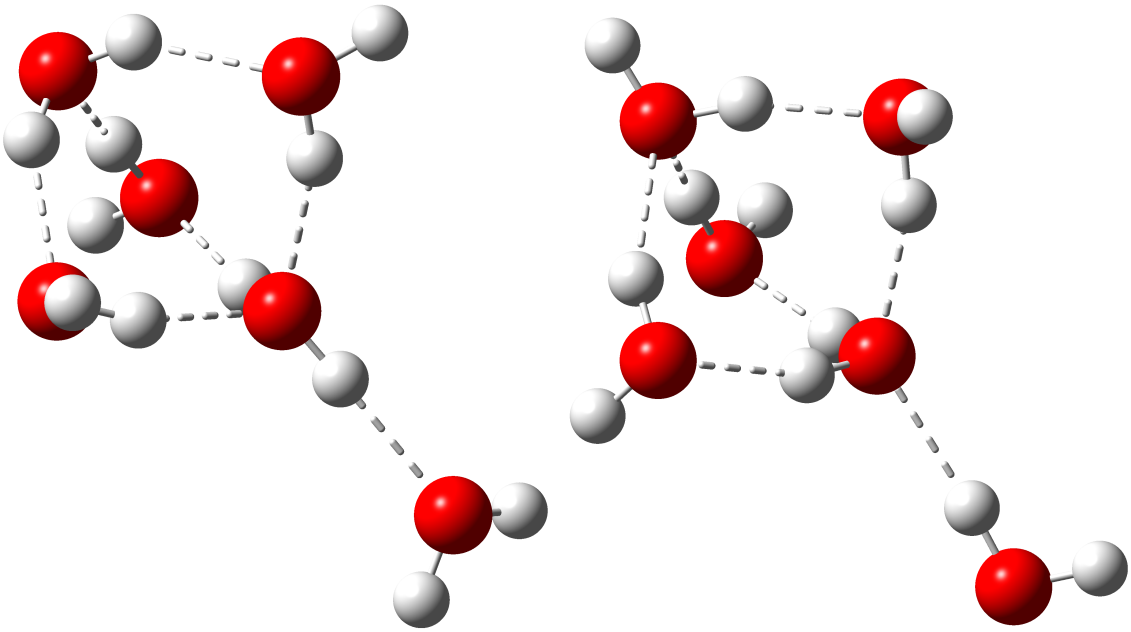

Figure 15: Structural comparison of the MP2/6-311++G** reference geometry (Left) [1] and the ASCEC-derived GFN2//B97-3c structure (Right).

2. Computational Efficiency The decisive advantage of the Two-Step ASCEC protocol is wall-time efficiency. Table 5 details the execution metrics. The Gaussian Direct approach managed only 16 completions out of 304 candidates after nearly two weeks of computation. Based on the mean execution time per job (18.7 hours), the projected completion time for the Direct approach is approximately **237 days** (5,700 hours). In contrast, the ASCEC Two-Step protocol completed the full ensemble analysis in under **16 hours**.

Table 5: Computational cost comparison: ORCA Two-Step Protocol vs. Gaussian Direct Optimization.

Protocol	Stage/Method	Jobs (N)	Mean Time/Job	Total Wall Time	Status
Method A (ORCA)	Step 1: Pre-opt (GFN2-xTB)	416	35 s	4 h 06 min	Complete
	Step 2: Refine (B97-3c)	53	12 min 29 s	11 h 01 min	Complete
	Total	–	–	**≈ 15 h 07 min**	**Complete**
Method B (G09)	Direct (wB97XD/6-31+G*)	16 (of 304)	18 h 46 min	300 h (12.5 days)	Incomplete
	Projected Total	*304*	–	*≈ 5,700 h*	*Estimated*

This comparative analysis demonstrates that while direct high-level optimization offers unbiased sampling of rare high-energy motifs, the ASCEC Two-Step protocol provides a rigorous thermodynamic description of the system at a fraction (approx. 0.2%) of the computational cost.

10.2 Automated Workflow: Formic Acid Dimer

To demonstrate the fully automated capabilities of the software, the Formic Acid Dimer was processed using the **Workflow Mode** with ORCA. Unlike the manual execution above, this protocol autonomously handled error correction, "Critical" structure filtering, and convergence checks to locate the global minimum.

10.3 Clustering Methodologies

This section compares the impact of different similarity algorithms on the final ensemble size.

10.3.1 Standard Topological Clustering

Using feature-vector similarity (threshold = 2.0), the system identifies broad conformational families based on connectivity and energy...

10.3.2 Hierarchical Clustering with RMSD

By incorporating Root Mean Square Deviation (RMSD) validation, the algorithm can distinguish between stereoisomers that share topological features but differ in spatial arrangement...

11 Command Reference

11.1 Global Flags

These options apply to the main `ascec` & `similarity` execution and control the verbosity and behavior of the underlying Monte Carlo engine. Use -h or –help to display all available commands.

Flag	Description
`-v`	**Verbose.** Prints status updates every 10 Monte Carlo cycles.
`--va`	**Very Verbose.** Prints updates every single cycle (for debugging).
`--standard`	Uses the standard Metropolis acceptance criterion.
`--nobox`	Disables the generation of box-visualization XYZ files.
`--nosum`	Skips summary file generation during the `sort` command.
`--justsum`	Generates the summary file only, without sorting files.
`-V, --version`	Displays version information and exits.

11.2 Sampling and Replication

Commands used to initiate Simulated Annealing runs.

```
# Basic Execution
ascec input.in              # Single run

# Replication (rN)
ascec input.in r3           # 3 Replicated runs (Default packing)
ascec input.in r5 --box10   # 5 Replicated runs with 10% packing density

# Box Validation
ascec input.in box          # Analyzes simulation box requirements
```

11.3 Automated Workflow (Protocol)

Controls the execution of the multi-stage pipeline.

Command Format	Description
`... protocol`	Starts (or resumes) the automated workflow defined.
`... protocol N`	**Force Restart.** Restarts the pipeline from stage N (wiping progress for that stage).
`... protocol N -i`	**Resume Interactive.** Resumes stage N from where it left off (incomplete state).

```
Workflow: Annealing → Calculation → Similarity → Optimization → Similarity
```

11.4 Protocol thresholds

These commands generate input files for Quantum Chemistry packages (ORCA/Gaussian) and define the error handling logic.

Flag	Description
`--retry=N`	**Execution Retries.** Maximum attempts to restart a calculation that crashed or failed (e.g., walltime limit).
`--redo=N`	**Topology Correction.** Maximum attempts to fix imaginary frequencies by perturbing coordinates along the unstable mode.
`--critical=N`	**Critical Threshold (Pre-opt).** Max % of allowed "Critical" structures (isolated imaginary freqs). Set to `0` for strict filtering.
`--skipped=N`	**Skipped Threshold (Final Opt).** Max % of allowed "Skipped" structures. Set to `0` to ensure all final motifs are true minima.

11.5 Similarity Analysis Module

Invoked via `ascec simil` or `similarity`, this module handles clustering and Boltzmann analysis.

Flag	Description
`--th, --threshold`	**Similarity Threshold.** Defines the clustering granularity (Default: 1.0). Lower values = stricter clustering.
`--rmsd`	**Geometric Validation.** Enables RMSD checking in Ångströms (Default: 1.0 Å). Useful for distinguishing conformers with similar energies but different shapes.
`-j, --cores`	**Parallelization.** Number of CPU cores to use for parsing and matrix calculation (Default: Auto-detect).
`-T, --temperature`	**Boltzmann Temp.** Temperature in Kelvin for Gibbs Free Energy ranking (Default: 298.15 K).
`--weights`	**Custom Weights.** Define specific feature importance (e.g., `'(energy=0.8)(dipole=0.15)'`).
`--reprocess-files`	Ignores the cached `.pkl` data and forces a re-read of all output.

11.5.1 Similarity Usage Examples

```
# Standard clustering
similarity --th=2

# High-precision clustering with geometric validation
similarity --th=1 --rmsd=0.5

# Performance mode (8 cores) with custom temperature
similarity --th=2 -j8 -T=350.0
```

12 Quick ORCA 6.1.1 Installation for Linux

1. Check for AVX2 Support

To determine if your Linux system's CPU supports the AVX2 instruction set for optimal performance, execute the following command in your terminal:

```
grep -i avx2 /proc/cpuinfo
```

If this command produces any output, your CPU supports AVX2. If the output is empty, you should use the non-AVX2 version of ORCA.

2. Download ORCA and OpenMPI

Register and log in to the official ORCA forum to download the appropriate files for ORCA 6.1.1. You will need to download both the ORCA package and the corresponding OpenMPI version.

Compounds search: https://orcaforum.kofo.mpg.de/app.php/portal

Compounds search: https://www.open-mpi.org/

```
/
├── openmpi-4.1.8.tar.gz
├── orca_6_1_1_linux_x86-64_shared_openmpi418.tar.xz
└── orca_6_1_1_linux_x86-64_shared_openmpi418_avx2.tar.xz
```

3. Install OpenMPI

It is recommended to install the provided OpenMPI version before installing ORCA. Extract and compile the OpenMPI source code according to the instructions included with the download.

```
mkdir openmpi-4.1.8
```

```
tar -xvf openmpi-4.1.8.tar.gz -C openmpi-4.1.8/ --strip-components=1
```

```
cd openmpi-4.1.8
```

```
./configure --prefix=/home/user/software/openmpi-4.1.8
```

```
make -j4
```

It tells make to run 4 jobs simultaneously, you can use make -j$(nproc) instead.

```
make install
```

```
mpirun --version
mpirun (Open MPI) 4.1.8
```

4. Install ORCA

Extract the downloaded ORCA tarball. For example:

```
mkdir orca_6_1_1
```

```
tar -xvf orca_6_1_1_linux_x86-64_shared_openmpi418.tar.xz -C orca_6_1_1 --strip-components=1
```

This will create a directory containing the ORCA executables.

5. Set Environment Variables

To make the ORCA program executable from any location in your terminal, you need to add its installation directory to your system's PATH. Follow these steps:

Open your shell's configuration file.

Open the appropriate configuration file in a terminal-based text editor like micro. If you are using the default Bash shell, use this command:

```
micro ~/.bashrc
```

Add the ORCA path to the file.

Note: The example directories for orca and mpi are located in $HOME/software.

```
# ORCA 6.1.1 Environment Configuration

# Define the paths to your ORCA 6.1.1 installation
export ORCA_BASE=$HOME/software
export ORCA611_ROOT=$ORCA_BASE/orca_6_1_1
export OPENMPI418_ROOT=$ORCA_BASE/openmpi-4.1.8

# ORCA and OpenMPI directories to the system's PATH
export PATH="$ORCA611_ROOT:$OPENMPI418_ROOT/bin:$PATH"

# ORCA and OpenMPI libraries to the library path
export LD_LIBRARY_PATH="$ORCA611_ROOT:$OPENMPI418_ROOT/lib:$LD_LIBRARY_PATH"
```

Save the file (Ctrl+S) and exit (Ctrl+Q). Then, reload the configuration:

Apply the changes.

For the changes to take effect in your current terminal session, you must "source" the configuration file.

```
source ~/.bashrc
```

Verify installation.

Be careful when having the Orca reader package installed. Orca is a screen reader pre-installed on most GNOME-based Linux distributions. To remove the Orca screen reader on Linux, use the terminal command.

```
sudo apt remove --purge orca
```

```
sudo apt autoremove
```

Then verify the ORCA 6.1.1 installation

```
orca --version
```

There is no standalone version command; however, the version (e.g., Program Version 6.1.1) is displayed at the beginning of the standard output for any run.

13 References

1. **Simulated Annealing:** Kirkpatrick, S. et al. *Science* **1983**, 220, 671.
2. **Clustering:** Ward, J. H. *J. Am. Stat. Assoc.* **1963**, 58, 236.
3. **ORCA:** Neese, F. *WIREs Comput. Mol. Sci.* **2022**, 12, e1606.

References

[1] G. Hincapié et al. "Structural Studies of the Water Hexamer". In: *The Journal of Physical Chemistry A* 114.29 (2010), pp. 7809–7814. DOI: 10.1021/jp103683m. (Visited on 02/19/2026).