CoSMoS User Manual

Global Structure Search Program

Version 1.0

lipai@mail.sim.ac.cn

February 9, 2026

Contents

1 Introduction

1.1 Overview

CoSMoS (Global Structure Search Program) is a sophisticated computational tool designed to find stable atomic structures through advanced global optimization algorithms. It combines Monte Carlo sampling with a climbing algorithm to efficiently explore the potential energy surface and identify low-energy structural configurations.

1.2 Key Features

- **Multi-calculator support**: Compatible with various energy calculators including EAM, CHGNet, DeepMD, FAIRChem, VASP, LAMMPS, and custom Python calculators
- **Flexible mobility control**: Supports index-based and region-based constraints on atomic movement
- **Permutation-invariant structure comparison**: Uses sorted SOAP descriptors combined with energy gating for robust duplicate detection
- **Automatic geometry classification**: Detects cluster, wire, slab, or bulk structures with appropriate handling of vacuum axes
- **Energy-weighted sampling**: Prioritizes movement of unstable atoms through per-atom energy analysis

2 Theoretical Background

2.1 Stochastic Surface Walking (SSW) Algorithm

CoSMoS is based on the Stochastic Surface Walking method [1], which combines:

1. **Random displacement**: Structure perturbation along a weighted random direction
2. **Climbing phase**: Addition of Gaussian bias potentials to escape local minima
3. **Local optimization**: Minimization to nearest stationary point
4. **Metropolis acceptance**: Temperature-dependent Monte Carlo acceptance

2.2 Algorithm Workflow

2.3 Energy-Weighted Direction Sampling

Unlike traditional Maxwell-Boltzmann sampling, CoSMoS uses per-atom potential energies to guide sampling:

$$\mathbf{N}_i \sim \mathcal{N}(0, \sigma_i^2), \quad \sigma_i = \sqrt{k_B T} \cdot e^{E_i^{\text{norm}}} \tag{1}$$

where E_i^{norm} is the normalized energy of atom i relative to the lowest-energy atom of the same element. This prioritizes movement of high-energy, unstable atoms.

Algorithm 1 CoSMoS Search Algorithm

Initialize structure $\mathbf{R}_0$ and energy pool $\mathcal{E} = \emptyset$
Optimize $\mathbf{R}_0 \to \mathbf{R}_{\min}$, add to pool
for step = 1 to N_{MC} **do**
    Generate random direction $\mathbf{N}$ (energy-weighted)
    Displace: $\mathbf{R}' = \mathbf{R}_{\text{current}} + d_s\mathbf{N}$
    for $h = 1$ to H (Gaussian potentials) **do**
        Add Gaussian bias: $V_{\text{bias}} = \sum_{i=1}^{h} we^{-|\mathbf{R}-\mathbf{R}_i|^2/(2\sigma^2)}$
        Optimize on biased PES: $\mathbf{R}_h \to \mathbf{R}_{\text{climb}}$
        if converged or max steps reached **then**
            break
        end if
    end for
    Optimize on real PES: $\mathbf{R}_{\text{climb}} \to \mathbf{R}_{\text{new}}$
    Compute $E_{\text{new}} = E(\mathbf{R}_{\text{new}})$
    if $\mathbf{R}_{\text{new}}$ is not duplicate **then**
        Add $(\mathbf{R}_{\text{new}}, E_{\text{new}})$ to pool
    end if
    Accept/reject via Metropolis: $P = \min(1, e^{-\Delta E/k_B T})$
    Update $\mathbf{R}_{\text{current}}$
end for
return lowest energy structure from pool

2.4 Structure Comparison via SOAP Descriptors

To handle atomic permutations, CoSMoS uses permutation-invariant descriptors:

1. Compute per-atom SOAP vectors $\{\mathbf{s}_i\}_{i=1}^{N}$
2. Sort by L2 norm: $\mathbf{s}_{\sigma(1)}, \ldots, \mathbf{s}_{\sigma(N)}$ where $\|\mathbf{s}_{\sigma(i)}\| \leq \|\mathbf{s}_{\sigma(i+1)}\|$
3. Flatten to structure descriptor: $\mathbf{D} = [\mathbf{s}_{\sigma(1)}; \ldots; \mathbf{s}_{\sigma(N)}]$
4. Compare: structures are duplicates if $\|\mathbf{D}_1 - \mathbf{D}_2\| < \tau_{\text{desc}}$ and $|E_1 - E_2| < \tau_E$

3 Installation

3.1 System Requirements

- Python 3.8 or higher
- pip (Python package manager)
- ASE $\geq$ 3.26.0 (Atomic Simulation Environment)
- dscribe $\geq$ 2.1.0 (for SOAP descriptors)

3.2 Installation Steps

Listing 1: Installation from source

```
# Clone the repository
git clone https://github.com/lipai-ustc/CosMos.git
cd cosmos
```

```
# (Optional) Create Conda environment
conda create -n cosmos python=3.10 -y
conda activate cosmos

# Install CoSMoS
pip install .

# (Optional) Install additional calculators
pip install deepmd-kit fairchem-core
```

Development Mode

For active development, use editable installation:

```
pip install -e .
```

This allows code modifications without reinstallation.

4 Basic Usage

4.1 Input Files

CoSMoS requires two input files in the working directory:

1. `input.json`: Calculation parameters configuration
2. `init.xyz`: Initial atomic structure in XYZ format (default path, can be changed in configuration)

4.2 Running CoSMoS

Listing 2: Basic execution

```
# Navigate to directory containing input.json and init.xyz
cd /path/to/working/directory

# Run search
cosmos
```

4.3 Output Configuration

The `output` section controls output settings:

Parameter	Description	Default
`directory`	Output directory name	"cosmos_output"
`rd_xyz`	Enable output of random direction information	false
`debug`	Enable debug mode for detailed logging	false

Table 1: Output configuration parameters

4.4 Output Files

Results are saved in the specified output directory:

- `all_minima.xyz`: All discovered minimum energy structures (trajectory format)
- `best_str.xyz`: Lowest energy structure
- `cosmos_log.txt`: Detailed search log with energies and algorithm status
- `rd_info.xyz`: Random direction information (only if `rd_xyz` is true)

4.5 Debug Mode Output

In debug mode, the log file contains detailed information for each optimization step:

```
Step 1: E_total = -125.234567 eV, E_base = -125.500000 eV, E_bias = 0.250000 eV,
Step 2: E_total = -125.456789 eV, E_base = -125.650000 eV, E_bias = 0.180000 eV,
...
```

5 Configuration Reference

5.1 input.json Structure

The configuration file is divided into several logical sections:

Listing 3: Minimal input.json example

```
{
  "system": {
    "name": "MySystem",
    "task": "global_search",
    "structure": "init.xyz"
  },
  "potential": {
    "type": "eam",
    "model": "potential.eam.alloy"
  },
  "monte_carlo": {
    "steps": 100,
    "temperature": 300
  },
  "climbing": {
    "gaussian": {
      "height": 0.2,
      "width": 0.2,
      "Nmax": 20
    },
    "random_direction": {
      "mode": ["thermo", "atomic"],
      "ratio": [[[0.5, 0.5], 1]]
    }
  },
  "optimizer": {
    "max_steps": 500,
    "fmax": 0.05
  },
  "output": {
```

```
    "directory": "cosmos_output",
    "debug": false
  }
}
```

5.2 Potential Configuration

5.2.1 EAM Potential

```
"potential": {
  "type": "eam",
  "model": "AlCu.eam.alloy"
}
```

5.2.2 CHGNet

```
"potential": {
  "type": "chgnet",
  "model": "pretrained"
}
```

5.2.3 DeepMD

```
"potential": {
  "type": "deepmd",
  "model": "dp_model.pb"
}
```

5.2.4 FAIRChem (Open Catalyst Project)

```
"potential": {
  "type": "fairchem",
  "model": "EquiformerV2-31M-S2EF-OC20-All+MD",
  "device": "cuda",
  "task_name": "oc20"
}
```

5.2.5 VASP

```
"potential": {
  "type": "vasp",
  "model": "INCAR"
}
```

Reads VASP parameters from the specified INCAR file.

5.2.6 Custom Python Calculator

```
"potential": {
  "type": "python"
}
```

Requires a `calculator.py` file in the working directory defining a `calculator` variable with an ASE Calculator object.

5.3 Monte Carlo Parameters

Parameter	Description	Default
`steps`	Total MC steps	Required
`temperature`	Temperature (K) for Metropolis	Required

Table 2: Monte Carlo configuration parameters

5.4 Climbing Phase Parameters

5.4.1 Gaussian Bias Potentials

Parameter	Description	Default
`gaussian.height`	Height of Gaussian bias potentials (w parameter) in eV	0.2
`gaussian.width`	Width of Gaussian potentials (ds parameter) in Å	0.2
`gaussian.Nmax`	Maximum number of Gaussians per climbing (H parameter)	20

Table 3: Gaussian bias potential parameters

5.4.2 Random Direction Generation

Parameter	Description	Default
`mode`	List of methods for generating random search directions	["thermo", "atomic"]
`ratio`	Weight configurations for combining multiple modes	[[[0.5, 0.5], 1]]
`rotation_param`	Dimer rotation parameter	10
`element_weights`	Element-specific weight factors	

Table 4: Random direction generation parameters

5.4.3 Random Direction Modes

5.5 Mobile Control

Mobile control allows constraining which atoms can move during optimization. Four modes are supported:

5.5.1 Mode 1: All Atoms Mobile (Default)

```
"mobile_control": {
  "mode": "all"
}
```

or omit the `mobile_control` section entirely.

Mode	Description
`thermo`	Use temperature-based random vectors (Boltzmann distribution)
`atomic`	Use energy-weighted random vectors based on per-atom energies
`nl`	Combine base random with local rigid movement (Nl)
`element`	Use element-weighted random vectors
`python`	Use custom user-defined function

Table 5: Random direction generation modes

5.5.2 Mode 2a: Specify Mobile Atoms

```
"mobile_control": {
  "mode": "indices_free",
  "indices_free": [10, 20, 25],
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Only listed atoms (0-based indexing) are mobile; all others are fixed.

5.5.3 Mode 2b: Specify Fixed Atoms

```
"mobile_control": {
  "mode": "indices_fix",
  "indices_fix": [0, 1, 2],
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Listed atoms are fixed; all others are mobile.

5.5.4 Mode 3: Region-Based Control

Spherical Region

```
"mobile_control": {
  "mode": "region",
  "region_type": "sphere",
  "center": [5.0, 5.0, 5.0],
  "radius": 10.0,
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Atoms within the sphere are mobile.

Slab Region

```
"mobile_control": {
  "mode": "region",
  "region_type": "slab",
  "origin": [0.0, 0.0, 0.0],
  "normal": [0, 0, 1],
  "min_dist": -5.0,
  "max_dist": 5.0,
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Atoms between two parallel planes are mobile.

Lower Region

```
"mobile_control": {
  "mode": "region",
  "region_type": "lower",
  "axis": "z",
  "threshold": 10.0,
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Atoms with coordinate threshold along the specified axis are mobile.

Upper Region

```
"mobile_control": {
  "mode": "region",
  "region_type": "upper",
  "axis": "z",
  "threshold": 10.0,
  "wall_strength": 10.0,
  "wall_offset": 2.0
}
```

Atoms with coordinate threshold along the specified axis are mobile.

5.5.5 Wall Potential

When `wall_strength` > 0, a quadratic repulsive potential prevents mobile atoms from penetrating into immobile regions:

$$V_{\text{wall}} = \begin{cases} 0 & \text{if } d \leq d_{\text{offset}} \\ \frac{1}{2} k_{\text{wall}} (d - d_{\text{offset}})^2 & \text{if } d > d_{\text{offset}} \end{cases} \tag{2}$$

where d is the penetration distance, d_{offset} is `wall_offset`, and k_{wall} is `wall_strength`.

6 Low-Dimensional Systems

6.1 Geometry Classification

CoSMoS automatically classifies structures based on vacuum analysis:

- **Cluster (0D)**: Vacuum along all three axes
- **Wire (1D)**: Vacuum along two axes
- **Slab (2D)**: Vacuum along one axis
- **Bulk (3D)**: No significant vacuum

6.2 Structure Preparation

IMPORTANT: Structure Centering

For low-dimensional systems, you **must manually center** the structure in the simulation box before running CoSMoS. The geometry detection algorithm measures distances from atoms to cell boundaries; misplaced structures will be misclassified.
Recommended placement:

- **0D cluster**: Center at $(L_x/2, L_y/2, L_z/2)$
- **1D wire**: Wire extends along one axis, centered in the other two
- **2D slab**: Centered along the vacuum direction

6.3 Vacuum Axis Handling

For structures with vacuum axes, CoSMoS automatically:

1. Detects vacuum axes using absolute coordinate analysis
2. Centers the structure along vacuum axes to the box center
3. Applies recentering after each accepted Monte Carlo move
4. Prevents translational drift during search

7 Examples

7.1 Example 1: AlCu Cluster with EAM

```
cd examples/AlCu-EAM
python generate_structure.py  # Generate init.xyz
cosmos                         # Run search
```

Configuration highlights:

- EAM potential for Al-Cu system
- 100 Monte Carlo steps
- Temperature: 300 K

7.2 Example 2: C_{60} with DeepMD

```
cd examples/C60-deepmd
# Ensure dp_model.pb is present
cosmos
```

Configuration highlights:

- DeepMD neural network potential
- Requires `dp_model.pb` trained model
- Install: `pip install deepmd-kit`

7.3 Example 3: Au Surface with FAIRChem (Custom Calculator)

```
cd examples/Au100-python-fairchem
# Check calculator.py for calculator definition
cosmos
```

Configuration highlights:

- Custom Python calculator using FAIRChem
- Slab region mobility control for surface atoms
- Requires: `pip install fairchem-core`

8 Advanced Topics

8.1 Task Types

CoSMoS supports two task types:

- **Global Search** (`global_search`): Finds stable structures across the potential energy surface
- **Structure Sampling** (`structure_sampling`): Samples atomic structures around existing minima

8.2 Custom Calculator Development

To use a custom ASE-compatible calculator:

1. Create `calculator.py` in your working directory
2. Define a `calculator` variable
3. Set `"type": "python"` in `input.json`

Example `calculator.py`:

```
from ase.calculators.emt import EMT

# Define your calculator
calculator = EMT()
```

8.3 Custom Random Direction Generation

When `random_direction.mode` is set to `"python"`, create a `generate_random_direction.py` file in your working directory:

```
import numpy as np
from ase import Atoms

def generate_random_direction(atoms: Atoms) -> np.ndarray:
    """
␣␣␣␣Generate␣custom␣random␣direction␣vector␣for␣CoSMoS␣algorithm.

␣␣␣␣Parameters:
␣␣␣␣␣␣␣␣atoms:␣ASE␣Atoms␣object␣representing␣current␣structure
```

```
    
    Returns:
        N: 1D numpy array of size (3*n_atoms,) representing the
           direction vector
    """
        n_atoms = len(atoms)
        N = np.random.randn(3 * n_atoms)  # Your custom logic here
        return N
```

Then configure in `input.json`:

```
"climbing": {
  "random_direction": {
    "mode": ["python"]
  }
}
```

8.4 Parallel Execution

CoSMoS is designed for serial execution per search. For parallel exploration:

- Run multiple independent searches with different initial structures
- Use different random seeds (implicitly different due to system time)
- Combine results post-processing

8.5 Troubleshooting

8.5.1 Structure Misclassification

Symptom: Cluster treated as bulk or vice versa.

Solution: Ensure structure is centered in the simulation box. Check with:

```
from ase.io import read
atoms = read('init.xyz')
pos = atoms.get_positions()
center = pos.mean(axis=0)
cell = atoms.get_cell()
box_center = (cell[0] + cell[1] + cell[2]) / 2
print(f"Structure center: {center}")
print(f"Box center: {box_center}")
```

8.5.2 Slow Convergence

Possible causes:

- `gaussian_height` too small: increase to 0.2–0.5 eV
- `temperature` too low: try 500–1000 K
- `max_gaussians` too small: increase to 20–30

8.5.3 Too Many Duplicates

Possible causes:

- `duplicate_tol` too loose: default is 0.01, try 0.005
- Energy tolerance too large: adjust in source (default 1e-3 eV)

9 Parameter Tuning Guidelines

Parameter	Effect	Tuning Strategy
`gaussian_height`	Barrier crossing ability	Increase for rugged PES
`gaussian_width`	Step size	Smaller for precise, larger for exploration
`max_gaussians`	Climbing persistence	Increase if stuck in shallow minima
`temperature`	Acceptance probability	Higher for aggressive search
`monte_carlo.steps`	Total exploration	More steps = better coverage

Table 6: Parameter tuning guidelines

10 References

References

[1] Shang, C., & Liu, Z.-P. (2013). *Stochastic Surface Walking Method for Structure Prediction and Pathway Searching*. Journal of Chemical Theory and Computation, 9(5), 1838–1845.

[2] Shang, C., & Liu, Z.-P. (2013). *Stochastic surface walking method for global optimization of atomic clusters and biomolecules*. The Journal of Chemical Physics, 139(24), 244104.

[3] Zhang, X.-J., & Shang, C., & Liu, Z.-P. (2012). *Double-Ended Surface Walking Method for Pathway Building and Transition State Location of Complex Reactions*. Journal of Chemical Theory and Computation, 9(12), 5745–5753.

11 License

A Complete Configuration Example

Listing 4: Full input.json with all options

```
{
  "system": {
    "name": "CompleteExample",
    "task": "global_search",
    "structure": "init.xyz"
  },
  "potential": {
    "type": "deepmd",
    "model": "dp_model.pb",
    "device": "cpu"
  },
  "monte_carlo": {
    "steps": 200,
    "temperature": 500
  },
  "climbing": {
```

```
    "gaussian": {
      "height": 0.2,
      "width": 0.2,
      "Nmax": 20
    },
    "random_direction": {
      "mode": ["thermo", "atomic"],
      "ratio": [[[0.5, 0.5], 1]],
      "element_weights": {"Cu": 1.5, "Al": 1.0},
      "rotation_param": 10
    }
  },
  "optimizer": {
    "max_steps": 500,
    "fmax": 0.03
  },
  "mobile_control": {
    "mode": "region",
    "region_type": "sphere",
    "center": "center",
    "radius": 8.0,
    "wall_strength": 15.0,
    "wall_offset": 1.5
  },
  "output": {
    "directory": "my_output",
    "rd_xyz": false,
    "debug": true
  }
}
```