Liftover_2D: Fast Chromatin Interaction Coordinate Conversion (Open2C)

Applicant Profile

1. Name: Hailey Cheng (Cheng Hei Lam)
2. Email: heilcheng2-c@my.cityu.edu.hk
3. School/University: City University of Hong Kong
4. Graduation Date: 07/2027
5. Major/Focus: Mathematics & Computer Science, with focus in Computational Biology
6. Country: Hong Kong
7. Timezone: UTC +8:00
8. GitHub profile: https://github.com/heilcheng
9. Discord profile: @haileychl

GSoC Information

1. Previous GSoC participation: No
2. Other organizations applied to: DeepChem, for a small project.
3. Hours per week: Taking 2 summer courses, but will be able to dedicate ~40 hours/week to the project.
4. Communicated with the mentor Aleksandra Galitsyna on Discord via DM & communicated with the contributors in the Open2C discord server (#gsoc2025 channel) to receive feedback.

Project information

1. Project Abstract

Liftover_2D will develop a fast and effective tool for converting chromatin interaction coordinates between different genome assemblies. As part of the Open2C ecosystem, it will enable researchers to "lift over" Hi-C and other 3C+ contact data without re-mapping raw reads, using existing chain files for coordinate translation. The tool will support both `.pairs` and `.cool` formats, using optimized algorithms that preserve pairwise context while achieving high performance through vectorization and parallelization.

2. Detailed description

Liftover_2D addresses the challenge of converting chromatin interaction data (pairs of genomic loci) between genome assemblies. Unlike existing tools that primarily handle 1D genomic features, Liftover_2D is specifically designed to preserve the crucial pairwise nature of 3C+ interactions. The tool will leverage standard chain files to map coordinates between assemblies. It employs efficient data structures and algorithms optimized to process millions of interaction pairs quickly.

Core features:

- Fast conversion of coordinate pairs between assemblies.
- Support for both `.pairs` (coordinate pairs) and `.cool` (binned matrix) file formats.
- Optimized interval-based lookups for coordinate mapping (e.g., O(log n) complexity).
- Robust handling of unmappable regions and coordinate conversion edge cases.
- Clean integration with the Open2C ecosystem and its existing toolchain.

Implementation details

A comprehensive demo in Jupyter Notebook:

https://colab.research.google.com/drive/1pD7u4ZDpz-RM5dw-CRhX12KAfh8Je3cf?usp=sharing

Liftover_2D will be implemented in Python, leveraging high-performance libraries like NumPy, Pandas, and potentially Polars to achieve efficiency. The core strategy revolves around using chain files to map coordinates between assemblies while crucially preserving the pairwise nature of chromatin interactions.

1. Chain File Processing and Coordinate Mapping:

- Chain files will be parsed and loaded into an efficient data structure, likely an interval tree (using libraries like `pyranges` or a custom implementation), allowing for rapid lookups (e.g., O(log n) complexity) of corresponding target regions for a given source chromosome and position.
- The fundamental logic for mapping a single coordinate is illustrated below. This function takes the chain-derived interval tree (`chain_tree`), source chromosome (`chrom`), and `position`, and returns the corresponding target chromosome and position:

(Python)

```
# Core mapping function
def lift_coordinate(chain_tree, chrom, position):
    """Map a single coordinate from source to target assembly."""
    if chrom not in chain_tree:
        return None

    # Find chain blocks overlapping the source position
    overlaps = chain_tree[chrom].find(position, position + 1)
    if not overlaps:
        return None

    # Select the best chain based on alignment score (higher is better)
    best_chain = max(overlaps, key=lambda x: x.data['score'])

    # Calculate the position offset based on the chosen chain block
    chain_offset = best_chain.data['target_start'] - best_chain.begin
    target_pos = position + chain_offset

    return (best_chain.data['target_chrom'], target_pos)
```

- This core function forms the basis for lifting over interaction pairs. Both coordinates of a pair will be lifted using this logic. Robust handling for coordinates falling in unmappable regions or regions with complex mapping scenarios (e.g., mapping to multiple locations) will be implemented, returning appropriate null values or flags based on user-defined policies.

2. Handling File Formats:

The tool will support both `.pairs` (coordinate pairs) and `.cool` (binned matrices) formats.

- For `.pairs` files, the `lift_coordinate` logic will be applied efficiently (see Performance Optimization) to both `pos1` and `pos2` for each interaction record.
- For `.cool` files, the bin coordinates defining the matrix axes will be lifted first. This requires careful reconstruction of the target bin table and mapping pixel values (interactions) between the source and target matrices, addressing potential complexities like bins partially mapping, merging, or splitting across assemblies. Correct metadata transfer according to `cooler` specifications will be ensured.

3. Performance Optimization:

Addressing the performance challenge with potentially millions to billions of interaction pairs is critical.

- **Vectorization:** Coordinate mapping operations will be vectorized using NumPy and/or Polars where possible. *This allows processing large batches of coordinates simultaneously in optimized C code, drastically reducing the overhead compared to iterating through pairs in Python.*
- **Chunked Processing:** To handle large input files within reasonable memory limits, data will be read, processed, and written in manageable chunks (streaming). *This approach keeps memory usage low regardless of input file size.*
- **Parallelization:** Operations that are independent across chromosomes (e.g., lifting pairs assigned to different chromosomes) can potentially be parallelized to leverage multi-core processors, further speeding up execution.

4. API and CLI:

- A user-friendly Command Line Interface (CLI) will be developed following standard Scientific Python and Unix best practices.
- An underlying Application Programming Interface (API) will allow programmatic use of the liftover functionality within other Python tools or workflows. This ensures clean integration with the Open2C ecosystem, potentially utilizing libraries like the `cooler` API for file I/O. Advanced features like progress reporting, filtering options for excluding unmapped pairs, and refined error handling/logging will be included.

Which aspect of the project idea do you see as the most difficult?

The most challenging aspect will be optimizing for performance at scale. Processing millions of interaction pairs efficiently requires careful data structure design, vectorization, and memory management. Specifically:

1. Efficient chain file parsing and interval tree construction
2. Batch processing of coordinates while minimizing Python overhead
3. Handling edge cases (unmappable regions, chromosome naming differences)
4. Ensuring memory efficiency for large datasets

I plan to address these challenges by:

- Using optimized interval tree implementations
- Implementing chunked processing for large files
- Vectorizing coordinate mappings with NumPy
- Parallelizing chromosome-independent operations

Which aspect of the project idea do you see as the easiest?

The conceptual approach to lifting over paired coordinates is straightforward: map each coordinate independently using established chain file techniques, then reconstruct the pairs. Basic CLI implementation and integration with existing Open2C I/O utilities are also relatively standard tasks.

Which portion of the project idea will you start with?

I'll begin with the core coordinate mapping engine and chain file parser, as these form the foundation for all other functionality. The first deliverable will be a function that can accurately map individual genomic coordinates between assemblies using chain files. This core functionality can be independently tested against established tools like UCSC liftOver before building the pairwise logic on top.

3. Weekly timeline

Community Bonding

- Study Open2C codebase, especially cooler and pairtools
- Set up development environment with all dependencies
- Review interval tree implementations and chain file formats
- Establish regular communication with mentors
- Finalize implementation strategy and key design decisions

Week 1

- Implement chain file parser with memory-efficient design
- Create core single-coordinate mapping function with tests
- Validate against UCSC liftOver for basic coordinates
- Develop data structures for efficient coordinate mapping

Week 2

- Implement pairwise coordinate mapping
- Add batch processing capability for multiple coordinates
- Create basic pairs file parser and writer
- Optimize interval lookups for performance

Week 3

- Develop CLI interface for pairs conversion
- Implement streaming/chunked processing for large files
- Add logging and progress reporting
- Create end-to-end test for small pairs file conversion

Week 4

- Handle edge cases in pairs conversion
- Implement error policies for unmappable coordinates
- Optimize memory usage for large files
- Add chromosome name standardization

Week 5

- Begin cooler format support
- Implement bin coordinate lifting
- Design bin table reconstruction algorithm
- Test with small example cool files

Week 6 (Midterm)

- Complete initial cooler format support
- Implement pixel mapping between bin tables
- Create validation tools for accuracy testing
- Prepare midterm evaluation demo and report
- Ensure pairs conversion is fully functional

Week 7

- Finalize cooler conversion implementation
- Handle complex cases (bin merging, overlaps)
- Ensure correct metadata transfer
- Optimize cool file writing

Week 8

- Implement performance optimizations
- Add parallelization for chromosome-independent operations
- Optimize memory usage for large datasets
- Benchmark against baseline implementations

Week 9

- Conduct validation testing
- Compare lifted vs. original matrices
- Test on multiple assemblies and species
- Fix any discrepancies or bugs identified

Week 10

- Polish CLI and API
- Add advanced features (reporting, filtering)
- Refine error handling and logging
- Ensure clean integration with Open2C ecosystem

Week 11

- Write comprehensive documentation
- Create usage examples and tutorials
- Prepare for code review and integration
- Implement final optimizations

Week 12

- Final testing across multiple platforms
- Fix any remaining issues
- Polish documentation
- Prepare final submission

Other commitments

Taking 2 summer courses, but will be able to dedicate ~40 hours/week to the project.

Biography/About Me

- Sophomore in Mathematics & Computer Science in CityUHK, based in Hong Kong, with a background in Biology (former Biology Olympiad student).
- I am applying for Liftover_2D because it perfectly combines my interests in computational biology and high-performance computing, allowing me to contribute meaningfully to the Open2C ecosystem.
- Conduing research project on ODE-based models for drug combination prediction; implemented pathway extensions and multi-task neural networks under dept. of Mathematics.
- Built deep learning pipelines for calcium imaging analysis using PyTorch for signal modeling and classification under dept. of Biomedical Engineering.
- Experienced in Python, NumPy, and handling biological datasets at scale.