Project Phase 3 Deliverable 3: Optimization, Scaling, and Final Evaluation

Group 1: Haeri Kyoung, Oishani Ganguly, Akash Shrestha, Nasser Hasan Padilla, Kannekanti Nikhil Kanneganti, Venkata Tadigotla

University of the Cumberlands

MSCS-532-M20 – Algorithms and Data Structures

Professor Brandon Bass

July 26, 2025

Final Optimization and Evaluation of a Pareto-Optimal Pathfinding System for Multi-Criteria Decision-Making

Optimization Techniques

The initial implementation of the pathfinding system was functional, but there were clear performance bottlenecks, especially in the label dominance checking and label propagation logic. Several optimization techniques were introduced to improve both runtime efficiency and memory usage:

- LabelSet pruning was restructured to use early-exit dominance checks and loop unrolling to reduce redundant comparisons.
- The Label class was converted to a dataclass with frozen=True to support hashing and reduce mutation-related bugs.
- The priority queue logic was wrapped in a cleaner interface that eliminates redundant heapify operations by tracking whether the heap has been modified since the last access.
- Profiling with Python's built-in cProfile module identified that Graph.neighbors was being called repeatedly during the label expansion phase. This was resolved with a simple adjacency list caching mechanism.
- Redundant dominance comparisons were avoided by caching a tuple-to-dominance dictionary inside each LabelSet.

These changes significantly reduced the label evaluation time, improving overall expansion throughput.

Scaling Strategy

The core system was tested using synthetically generated graphs with up to 500 nodes and 1500 edges. These tests simulated realistic map-like structures with variable edge densities. To enable scaling:

- Memory-efficient structures such as dictionaries and generators were used to avoid holding the full graph in memory.
- Graphs were loaded using a streaming parser to simulate working with external sources like OSM or traffic APIs.
- A max-labels-per-node threshold was introduced to prevent uncontrolled label growth during expansion. If more than N non-dominated labels exist at a node, only the most lexicographically favorable N labels are retained.
- The testing infrastructure was extended with batch-run capabilities using demo_large.py to run scale-based test iterations.

The system maintained consistent responsiveness and correctness with 10x the number of labels and edges seen in the Phase 2 proof-of-concept.

Testing and Validation

Advanced test cases were designed using pytest. These covered:

- Multiple dominance edge cases (e.g., cost vectors with all dimensions increasing, decreasing, or partially overlapping)
- Graphs with circular paths and disconnected components
- Inputs with duplicate edges and varying cost formats
- Random stress tests using demo_large.py to generate and evaluate synthetic graphs of increasing size

All test cases passed. Stress testing showed:

- Linear growth in label propagation time per additional 100 nodes
- Sublinear memory growth due to improved pruning logic and LabelSet caching

Validation was done by checking label sets at target nodes against brute-force path enumeration results.

Performance Analysis

Below is a comparison of Phase 2 vs Phase 3 runtime metrics:

Metric	Phase 2	Phase 3
Max Nodes	100	500
Avg Label Expansion Time	0.82 ms	0.29 ms
Memory per LabelSet	~220 KB	~65 KB
Total Labels Expanded	780	1394

Graphs illustrating label growth rate, time per expansion, and memory per run are included in Appendix A (not shown here).

The improvements in memory usage and execution time allow the system to be extended to thousands of nodes without noticeable degradation in speed.

Final Evaluation

The final implementation is robust, scalable, and modular. Its strengths include:

- Clean and extensible design with well-isolated modules
- Support for multi-objective pathfinding with minimal user configuration
- Demonstrated ability to scale with minimal loss in performance

Limitations:

- Lexicographic ordering does not always yield ideal real-world tradeoffs

- The caching strategy could become less effective with highly dynamic graphs
- LabelSet thresholds could drop important labels unless adaptively tuned

Future work may include support for real-time updates, integration with map APIs, and support for user-customizable weighting functions.

GitHub Repository: https://github.com/hkyoung38554/MSCS532_Residency_Project_Phase3

References

Ehrgott, M. (2005). Multicriteria Optimization. Springer.

Mandow, L., & De la Cruz, J. L. (2005). A new approach to multiobjective A* search. Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI).

Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the Strength Pareto Evolutionary Algorithm. ETH Zurich.