Navigating Complexity: Multi-Objective Pathfinding with EV Charging Optimization

Colby Ziyu Wang

Toronto Metropolitan University
Depart of Computer Science
c12wang@torontomu.ca

Abstract

As electric vehicles (EVs) become increasingly prevalent, efficient pathfinding algorithms are critical to address range limitations and optimize travel routes. In this paper, we investigate the application of the state-of-the-art multi-objective search algorithm, A*pex, to the EV pathfinding problem. We formulate EV pathfinding as a multi-objective search where each edge is associated with a vector of costs, including distance, a binary indicator of whether the edge leads to a charging station, and an additional cost component that penalizes excessively long distances or both excessively long and short distances. Our experiments explore the impact of varying charging station densities and different levels of penalization on algorithm performance, highlighting the flexibility and effectiveness of A*pex in addressing multi-objective EV routing challenges.

Introduction

The adoption of electric vehicles (EVs) has surged globally, driven by the lower cost of electricity compared to gasoline and their reduced environmental impact. Despite these advantages, a significant challenge persists: the limited range of EVs often falls short for medium- to long-distance travel, such as intercity journeys. As a result, efficient route planning becomes essential, aiming to minimize the total distance traveled while ensuring sufficient access to charging stations along the path. This problem can be framed as a multi-objective path planning task, where each edge in the graph is annotated with multiple cost components.

In this paper, we investigate the application of *A*pex*, a state-of-the-art multi-objective search algorithm proposed by (Zhang et al. 2022), to the EV pathfinding problem. Each edge is associated with two primary cost components: the first represents the physical distance, and the second is a binary value indicating charging station accessibility (0 if the edge leads to a charging station, and 1 otherwise).

The main contributions of this work are as follows:

1. We analyzed the impact of the number of charging stations on algorithm runtime, exploring its effect on search efficiency.
2. We extended the problem by introducing a third cost component to penalize edges based on their deviation from an ideal distance. Two scenarios were examined: penalizing both long and short edges, and penalizing only long edges. Aggregated runtime results were analyzed for various ideal distance values in both cases.

This study provides insights into the computational behavior of multi-objective pathfinding algorithms under varying conditions and offers practical implications for optimized EV route planning.

Background

In this section, we briefly review the necessary background for multi-objective search algorithms, including Pareto-optimal and ϵ-Pareto-optimal solution sets, and the best-first multi-objective search framework.

Pareto-optimal and ϵ-Pareto-optimal Solution Sets

The Pareto-optimal solution set consists of all undominated paths. A path π with cost vector $\mathbf{c}$ dominates another path π' with cost vector $\mathbf{c}'$ if and only if:

$$\forall i \in \{0, \ldots, N-1\},\ c[i] \leq c'[i], \text{ and}$$
$$\exists j \in \{0, \ldots, N-1\},\ c[j] < c'[j],$$

where N is the number of cost components.

The size of the Pareto-optimal solution set can grow exponentially with the size of the graph. To address this, it is often more efficient to compute an approximate Pareto-optimal solution set for a user-provided approximation factor ϵ. An ϵ-Pareto-optimal solution set is a set of solutions such that, for any Pareto-optimal solution π' with cost vector $\mathbf{c}'$, there exists a solution π with cost vector $\mathbf{c}$ in the ϵ-Pareto-optimal solution set, where π ϵ-dominates π'. This means:

$$\forall i \in \{0, \ldots, N-1\},\ c'[i] \leq (1+\epsilon) \cdot c[i].$$

Best-First Multi-Objective Search Framework

A best-first multi-objective search algorithm computes a solution set that is usually Pareto-optimal or ϵ-approximate Pareto-optimal. The algorithm maintains a priority queue, *Open*, which contains the generated but not yet expanded nodes. Each node n consists of a state $s(n)$ and a g-value $g(n)$. The f-value of a node is defined as:

$$f(n) = g(n) + h(s(n)),$$

where $h(s(n))$ is the heuristic estimate for the state $s(n)$.

The *Open* queue is initialized with a node containing the start state s_{start} and a g-value of 0. At each iteration, the algorithm performs the following steps:

1. Extracts a node from *Open* with the smallest f-value among all nodes in *Open*.
2. Performs a dominance check to determine whether the extracted node or any of its descendants have the potential to belong to the solution set:
   - If not, the node is discarded.
   - Otherwise:
   – If the node contains the goal state, it is added to the solution set.
   – Otherwise, the node is expanded by generating a new node for each successor of the state contained in the node.
3. For each generated node, the algorithm performs a dominance check to determine whether it or any of its descendants have the potential to be part of the solution set:
   - If not, the generated node is discarded.
   - Otherwise, it is added to *Open*.

The algorithm terminates when *Open* becomes empty and returns the solution set.

Related Works

In this section we review common multi-objective search algorithms, including NAMOA* (Mandow and De La Cruz 2010), NAMOA*dr (Pulido, Mandow, and Pérez-de-la Cruz 2015), PPA (Goldin and Salzman 2021)*, and A*pex (Zhang et al. 2022).

NAMOA* and NAMOA*dr

NAMOA* (Mandow and De La Cruz 2010) is a best-first multi-objective search algorithm designed to find the Pareto-optimal solution set. Each node in the Open list corresponds to a path π, which includes:

- The state $s(\pi)$, representing the last state in path π.
- The g-value, $g(\pi) = c(\pi)$, representing the cost of path π.

NAMOA* maintains two sets of f-values for each state s:

1. The closed set $G_{\text{cl}}(s)$, containing the f-values of all expanded paths passing through s.
2. The open set $G_{\text{op}}(s)$, containing the f-values of all generated but not yet expanded paths passing through s.

At each iteration, NAMOA* performs the following:

- Extracts a path π from Open whose f-value is undominated by any other path in Open.
- Adds $f(\pi)$ to $G_{\text{cl}}(s(\pi))$.

If $s(\pi)$ is the goal state, NAMOA*:

- Adds the path π to the solution set.
- Removes all paths from Open whose f-values are dominated by $f(\pi)$.

Otherwise, for each edge $\langle s(\pi), s' \rangle \in E$, NAMOA*:

- Extends π to an extended path π'.
- Discards π' if its f-value is dominated by any f-value in $G_{\text{cl}}(s') \cup G_{\text{op}}(s') \cup G_{\text{cl}}(s_{\text{goal}})$.
- Removes all paths containing s' whose f-values are dominated by $f(\pi')$ from Open (and their f-values from $G_{\text{op}}(s')$).
- Adds π' to Open and updates $G_{\text{op}}(s')$ with its f-value.

NAMOA*dr (Pulido, Mandow, and Pérez-de-la Cruz 2015) improves on NAMOA* by optimizing the dominance checking process, which is often the computational bottleneck in multi-objective search. NAMOA*dr always extracts a path from Open with the lexicographically smallest f-value among all paths in Open.

When NAMOA*dr generates a path π, it ensures:

$$f_1(\pi) \geq \max\{f_1(\pi') : f_1(\pi') \in G_{\text{cl}}(s(\pi))\}$$
$$f_1(\pi) \geq \max\{f_1(\pi') : f_1(\pi') \in G_{\text{cl}}(s_{\text{goal}})\},$$

assuming the heuristic function is consistent. Consequently, NAMOA*dr skips checking the first cost component during dominance checks against $G_{\text{cl}}(s(\pi)) \cup G_{\text{cl}}(s_{\text{goal}})$.

Instead of maintaining the set $G_{\text{cl}}(s)$ for each state s, NAMOA*dr maintains the smaller set $G^T_{\text{cl}}(s)$ of undominated truncated f-values, significantly improving efficiency.

PP-A*

PP-A* (Goldin and Salzman 2021) is a bi-objective search algorithm that finds an ϵ-approximate Pareto-optimal solution set. Each node in Open corresponds to an ϵ-bounded path pair $PP = \langle \pi_{\text{tl}}, \pi_{\text{br}} \rangle$, where π_{tl} and π_{br} represent the top-left and bottom-right paths, respectively, with $s(\pi_{\text{tl}}) = s(\pi_{\text{br}})$, $g_1(\pi_{\text{tl}}) \leq g_1(\pi_{\text{br}})$, and $g_2(\pi_{\text{tl}}) \geq g_2(\pi_{\text{br}})$. The g-value of PP is $g(PP) = [g_1(\pi_{\text{tl}}), g_2(\pi_{\text{br}})]$, and PP is ϵ-bounded if:

$$g_1(\pi_{\text{br}}) \leq (1+\epsilon_1) g_1(\pi_{\text{tl}}) \quad \text{and} \quad g_2(\pi_{\text{tl}}) \leq (1+\epsilon_2) g_2(\pi_{\text{br}}).$$

PP-A* generalizes single-path dominance by representing sets of paths using ϵ-bounded path pairs, reducing the number of expansions. At each iteration, the algorithm extracts the path pair in Open with the lexicographically smallest f-value. Paths with dominated f-values are discarded based on the following dominance checks:

1. If an expanded path pair ϵ-dominates the current path pair PP.
2. If an expanded path pair with the same state weakly dominates PP.

When expanding a path pair PP, the algorithm generates child path pairs by extending both π_{tl} and π_{br} with the successor states. Before adding a child path pair to Open, PP-A* merges it with an existing path pair if the merged result is still ϵ-bounded.

Upon termination, PP-A* returns the bottom-right paths of all path pairs in the solution set as an ϵ-approximate Pareto-optimal solution set. This merging mechanism and ϵ-bounded representation allow PP-A* to handle multi-objective problems efficiently while maintaining a manageable number of expansions.

A*pex

A*pex (Zhang et al. 2022) is a multi-objective search algorithm that generalizes PP-A* for any number of cost components, improving efficiency and flexibility in representing sets of paths. Each node in Open corresponds to an ϵ-bounded apex-path pair $AP = \langle A, \pi \rangle$, where A is a vector of N cost components and π is a representative path with $A \preceq g(\pi)$. The g-value $g(AP) = A$ is the component-wise minimum of the g-values of all paths in the set represented by AP. This representation allows A*pex to merge larger sets of paths early in the search, reducing the number of expansions and solution set size.

A*pex maintains the ϵ-bounded condition for apex-path pairs:

$$f(\pi) \preceq_\epsilon f(AP),$$

where f and g are defined similarly to NAMOA*.

Merging Apex-Path Pairs

Two apex-path pairs with the same state can be merged into a single pair. The apex of the merged pair is the component-wise minimum of the two original apexes. The representative path is chosen from the candidates that result in an ϵ-bounded merged pair. If no candidates exist, the pairs are not merged. A*pex offers three policies for selecting the representative path:

- **Random Method (RANDOM):** Selects a representative path randomly from the candidates.
- **Lexicographically Smallest Reverse g-Value Method (SMALL_G):** Selects the representative path with the lexicographically smallest reverse g-value. If no such path exists among the candidates, merging is skipped. For two cost components, this approach resembles PP-A*'s bottom-right path selection.
- **Greedy Method (MORE_SLACK):** Maximizes the slack, defined as:

$$\min_{i=1,2,\ldots,N} \frac{1 + \epsilon_i - \frac{f_i(\pi)}{f_i(AP)}}{\epsilon_i},$$

  where $f_i(\pi)$ is the i-th component of the f-value of the representative path π, and $f_i(AP)$ is the i-th component of the f-value of the apex-path pair. This approach prioritizes paths that leave more room for future merges.

By using these policies, A*pex adapts its merging strategy to balance efficiency and solution quality, enabling it to handle large multi-objective problems effectively.

Method Tested

Currently, the optimal approach for solving this problem is the A*-pex algorithm, which was used for subsequent experiments. I utilized the code base available at https://github.com/HanZhang39/A-pex, which implements the A*-pex algorithm.

Preliminary Experiments

A Small Graph

For our preliminary experiments, we first obtained the EV charging station data for Connecticut from the publicly available dataset at https://data.ct.gov/Transportation/Electric-Vehicle-Charging-Stations/jfhb-ebu6/about_data. This dataset includes longitude and latitude information for each charging station. To map these coordinates to a 2D grid suitable for our analysis, we used the following formula to convert the geographical coordinates into Cartesian coordinates:

$$x = R \cdot \lambda \cdot \cos(\phi)$$
$$y = R \cdot \phi$$

where:

- R is the Earth's radius (6371 km),
- λ is the longitude in radians,
- ϕ is the latitude in radians.

The longitude and latitude were first converted to radians using:

$$\lambda = \text{longitude} \cdot \frac{\pi}{180},$$
$$\phi = \text{latitude} \cdot \frac{\pi}{180}$$

To maintain a discrete grid, the resulting x and y coordinates were rounded to the nearest integer.

After converting all charging station locations to 2D grid coordinates, we constructed a grid graph using the minimum and maximum x and y coordinates as boundaries. Each integer point on this grid represented a node, and nodes corresponding to charging station coordinates were marked accordingly. The graph was constructed as a 4-connected grid, where each edge (u, v) had:

- A distance cost of 1.
- A charging station cost of 0 if the edge led to a charging station (i.e. v has a charging station, and 1 otherwise).

In total, our graph contained 25,990 nodes, 103,274 edges, and 288 charging stations. To visualize this, we created a plot of the grid representation with marked charging stations as shown in Figure 1.

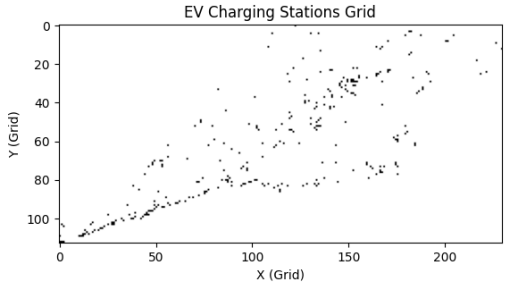

Figure 1: 2D Grid Representation of EV Charging Stations in Connecticut

Below is a table summarizing the average run-time of A*pex (in ms) for 100 randomly generated start-end node pairs:

ϵ	MORE_SLACK	SMALL_G	RANDOM
0.00001	21.04	20.35	20.86
0.0001	21.00	20.47	20.57
0.001	19.36	19.85	19.76
0.005	19.33	18.82	19.55
0.01	19.32	18.84	18.97

Table 1: Table shows the average runtime (in ms) for A*pex variants using different ϵ values across A*pex-G, A*pex-L, and A*pex-R settings.

In this toy example, we observe that the average run time shows minimal variation across different values of ϵ and merging policies. This can be attributed to the relatively small size of the graph, which limits the complexity of the search space.

Number of Charging Stations and Problem Difficulty

In this experiment, we investigate the impact of the number of charging stations on the runtime of our algorithm on a larger graph. We utilize a directed road graph from the DIMACS Implementation Challenge: Shortest Path dataset, which is available at https://aetperf.github.io/2022/09/22/Download-some-benchmark-road-networks-for-Shortest-Paths-algorithms.html. Specifically, we use the Bay Area network. Charging stations are randomly assigned to nodes based on a Bernoulli process with probability p representing the likelihood that a node contains a charging station. In total there are 321270 nodes and 794830 edges.

We analyze the average runtime for 100 random pairs of (start, goal) using $\epsilon = 0.01$ and a merging policy of Random. The x-axis in Figure 2 represents the probability p (ranging from 0.1 to 0.9) of a node having a charging station, while the y-axis shows the corresponding average runtime in ms.

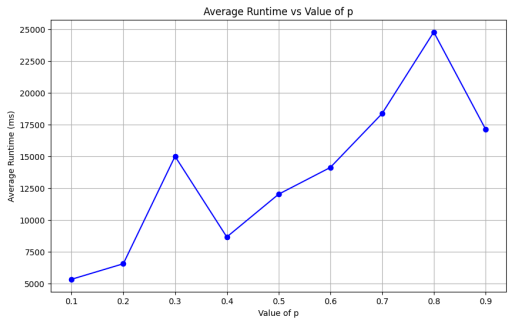

Figure 2: Average Runtime vs. Charging Station Density (p)

We observe that, in general, an increase in the number of charging stations (as p increases from 0.4 to 0.8) leads to longer runtimes. This behavior could be attributed to the following reasons:

- **Increased Search Space Complexity:** With more charging stations available, the algorithm has to evaluate a larger number of potential paths and refueling options, which increases the computational overhead and runtime.
- **Exploration of ϵ-Dominated Solutions:** A higher density of charging stations may lead to multiple paths that are ϵ-dominated by each other, causing the algorithm to explore these similar-cost paths before converging to an optimal solution.

However, at $p = 0.9$, we see a decrease in runtime, possibly because the abundance of charging stations simplifies pathfinding by making refueling options readily available, thereby reducing the need for complex routing decisions.

Adding a Third Component

In this experiment, we introduce a third cost component for each edge to penalize paths that are either too long or too short. This approach is based on the rationale that dividing a trip into ideal-length segments can be beneficial. The third cost component is designed to penalize paths based on how much their edge distances deviate from an ideal distance, reflecting a preference for paths composed of segments of a desirable length. This ideal edge distance, denoted as m, is set to 500 for this experiment. To achieve this, we use a logarithmic function to quantify the penalty for deviations, ensuring smooth and diminishing penalties for increasing deviations.

The third cost component for an edge is calculated as:

$$C_{\text{third}}(d, l) = \begin{cases} \log(1 + |d - m|), & \text{if } l = 1, \\ \log(1 + |d - m|) + P, & \text{if } l = 0, \end{cases}$$

where:

- d: distance of the edge,
- m: ideal edge distance (set to 500),
- l: binary indicator for whether the edge leads to a charging station ($l = 1$ if it does, $l = 0$ otherwise),
- P: fixed penalty applied when the edge does not lead to a charging station.

This formulation balances the trade-off between edge distance and the presence of charging stations, facilitating flexible and efficient pathfinding in electric vehicle routing scenarios.

The table below summarizes the average runtime (in milliseconds) across 100 randomly generated start-end node pairs for two tie-breaking strategies: **RANDOM** and **MORE_SLACK**. Note that for this and subsequent experiments, the **SMALL_G** policy was not tested as it is not implemented in the repository.

ϵ	**MORE_SLACK**	**RANDOM**
0.00001	151009.61	152449.46
0.0001	100667.38	103890.71
0.001	36630.07	44184.39
0.005	13286.21	18042.92
0.01	10465.02	14856.48

Table 2: Average runtime (in ms) for the EV third component experiment comparing **MORE_SLACK** and **RANDOM** tie-breaking strategies across different ϵ values.

We observe that as ϵ increases, the runtime generally decreases due to the reduction in the size of the solution set. The **RANDOM** tie-breaking policy exhibited longer run times compared to the **MORE_SLACK** policy. This is because the **MORE_SLACK** policy maximizes the selection of candidates that leave more flexibility for future merges, leading to more efficient exploration and solution convergence.

Adding a Third Component Version 2

In this version of the experiment, the third cost component is designed to penalize only long distances, while short distances remain unpenalized. This adjustment reflects scenarios where shorter segments of travel are considered acceptable or even desirable, and only deviations above a certain threshold are discouraged.

The third cost component for an edge is calculated as:

$$C_{\text{third}}(d, l) = \begin{cases} 0, & \text{if } d \leq m \text{ and } l = 1, \\ P, & \text{if } d \leq m \text{ and } l = 0, \\ \log(1 + (d - m)), & \text{if } d > m \text{ and } l = 1, \\ \log(1 + (d - m)) + P, & \text{if } d > m \text{ and } l = 0, \end{cases}$$

Here:

- d is the distance of the edge,
- m is the ideal distance,
- $l = 1$ indicates the edge leads to a charging station, and $l = 0$ otherwise,
- P is the penalty for edges that do not lead to a charging station.

This approach provides a focused penalty system that prioritizes minimizing excessively long segments while maintaining flexibility for shorter paths. By penalizing only long distances, this version emphasizes the importance of travel efficiency without unnecessarily restricting routing options.

Below is a table summarzing the runtime:

ϵ	**MORE_SLACK**	**RANDOM**
0.00001	182708.44	151891.88
0.0001	99021.77	102015.20
0.001	34543.42	43221.70
0.005	12959.44	17832.23
0.01	10285.01	14863.84

Table 3: Average runtime (in ms) for different ϵ values across MORE_SLACK and RANDOM tiebreaking policies.

In this setting ($m = 500$), the runtime generally does not differ significantly between the two penalty versions. This is likely because most distances in the graph exceed 500 units, rendering the two-sided penalty effectively equivalent to the one-sided penalty.

Variations to Third Component Experiment

In this experiment, we explore the impact of varying the ideal distance m on runtime and path selection for both versions of the third component calculation. The ideal distance m is tested with the following values: $m = 1000$, and $m = 2000$. This variation allows us to analyze how different thresholds for penalizing edge distances influence the overall search behavior and solution quality.

Below are the tables summarizing the results:

Version 1, $m = 1000$		
ϵ	MORE_SLACK	RANDOM
0.00001	188070.80	188522.40
0.0001	137149.20	138538.36
0.001	48019.72	61252.52
0.005	16930.00	23525.76
0.01	13767.08	19858.64

Table 4: Average runtime (ms) for Version 1 with $m = 1000$ using MORE_SLACK and RANDOM tiebreaking policies.

Version 1, $m = 2000$		
ϵ	MORE_SLACK	RANDOM
0.00001	189283.52	189695.80
0.0001	137763.60	139788.76
0.001	48159.92	57231.96
0.005	16975.36	23101.52
0.01	13697.92	19515.88

Table 5: Average runtime (ms) for Version 1 with $m = 2000$ using MORE_SLACK and RANDOM tiebreaking policies.

Version 2, $m = 1000$		
ϵ	MORE_SLACK	RANDOM
0.00001	177433.28	177432.64
0.0001	128836.88	132103.00
0.001	42998.04	51869.20
0.005	17009.36	20937.92
0.01	13784.28	18643.92

Table 6: Average runtime (ms) for Version 2 with $m = 1000$ using MORE_SLACK and RANDOM tiebreaking policies.

Version 2, $m = 2000$		
ϵ	MORE_SLACK	RANDOM
0.00001	154289.36	154423.44
0.0001	115529.20	117819.00
0.001	36388.68	46417.48
0.005	16542.24	20404.32
0.01	13613.12	18177.96

Table 7: Average runtime (ms) for Version 2 with $m = 2000$ using MORE_SLACK and RANDOM tiebreaking policies.

Due to space constraints, we will analyze the results from all six of the previously presented tables (three for each penalty version) collectively in the Appendix. **Version 1** corresponds to the two-sided penalty, while **Version 2** corresponds to the one-sided penalty. Please refer to the Appendix for the summarized table and analysis.

Future Work

EV Range Constraints

Incorporate range constraints into the pathfinding problem to better simulate real-world scenarios. Specifically, ensure that the distance between consecutive charging stations along the selected path does not exceed the maximum range of the electric vehicle. This addition would make the problem more realistic by factoring in the limited battery capacity of EVs and the need for strategic placement of charging stations. Future experiments could explore the trade-offs between optimal distance paths and the necessity of adhering to range limitations.

Charging Time Considerations

Introduce a time component into the multi-objective framework to account for the duration spent at charging stations. This would involve modeling the time required for charging based on factors such as battery state, charging speed, and station availability. By including this time cost as an additional objective, the algorithm can be evaluated on its ability to minimize total trip time while ensuring sufficient charging stops. Future work could also investigate the impact of prioritizing faster charging stations and integrating dynamic station availability into the model.

Conclusion

In this study, we evaluated the A*pex algorithm on the electric vehicle (EV) pathfinding problem, conducting a series of diverse experiments on both small-scale and large-scale road networks. Our experiments demonstrated the algorithm's effectiveness in handling multi-objective search tasks, particularly in minimizing travel distance while maximizing the number of charging stations along the path. We introduced and analyzed the impact of a third cost component, which penalizes paths based on deviations from an ideal edge distance, exploring variations that target both long and short distances as well as one-sided penalties. Additionally, we investigated the influence of parameters such as the density of charging stations and the value of ideal edge distance on runtime and solution quality.

Our findings highlight the flexibility of the A*pex algorithm in accommodating different multi-objective criteria and its scalability to larger graphs. The use of ϵ-bounded apex-path pairs allowed for efficient pruning and merging of paths, reducing computational overhead while maintaining solution quality. The results also underscored the importance of incorporating realistic constraints, such as range limitations and charging times, in future work to further refine the applicability of multi-objective search algorithms to real-world EV routing problems.

In conclusion, the A*pex algorithm provides a robust framework for addressing the challenges of multi-objective EV pathfinding, and our experiments pave the way for future studies to explore more complex and realistic scenarios in sustainable transportation systems.

References

Goldin, H.; and Salzman, O. 2021. PP-A*: Efficient Planning with Probabilistic Preferences. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 4297–4303. IJCAI.

Mandow, L.; and De La Cruz, J. L. P. 2010. Multiobjective A* search with consistent heuristics. *Journal of the ACM (JACM)*, 57(5): 1–25.

Pulido, F. J.; Mandow, L.; and Pérez-de-la Cruz, J. L. 2015. Multiobjective search with lexicographic goal sorting and heuristic dominance relations. *European Journal of Operational Research*, 242(3): 826–834.

Zhang, H.; Salzman, O.; Kumar, T. K. S.; Felner, A.; Hernandez, C.; and Koenig, S. 2022. A*pex: Efficient Approximate Multi-Objective Search on Graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 423–433.

Appendix

In this section we analyze the results from all six of the previously presented tables (three for each penalty version) collectively. **Version 1** corresponds to the two-sided penalty, while **Version 2** corresponds to the one-sided penalty.

Version 1, $m = 500$		
ϵ	MORE_SLACK	RANDOM
0.00001	151009.61	152449.46
0.0001	100667.38	103890.71
0.001	36630.07	44184.39
0.005	13286.21	18042.92
0.01	10465.02	14856.48

Version 2, $m = 500$		
ϵ	MORE_SLACK	RANDOM
0.00001	182708.44	151891.88
0.0001	99021.77	102015.20
0.001	34543.42	43221.70
0.005	12959.44	17832.23
0.01	10285.01	14863.84

Version 1, $m = 1000$		
ϵ	MORE_SLACK	RANDOM
0.00001	188070.80	188522.40
0.0001	137149.20	138538.36
0.001	48019.72	61252.52
0.005	16930.00	23525.76
0.01	13767.08	19858.64

Version 2, $m = 1000$		
ϵ	MORE_SLACK	RANDOM
0.00001	177433.28	177432.64
0.0001	128836.88	132103.00
0.001	42998.04	51869.20
0.005	17009.36	20937.92
0.01	13784.28	18643.92

Version 1, $m = 2000$		
ϵ	MORE_SLACK	RANDOM
0.00001	189283.52	189695.80
0.0001	137763.60	139788.76
0.001	48159.92	57231.96
0.005	16975.36	23101.52
0.01	13697.92	19515.88

Version 2, $m = 2000$		
ϵ	MORE_SLACK	RANDOM
0.00001	154289.36	154423.44
0.0001	115529.20	117819.00
0.001	36388.68	46417.48
0.005	16542.24	20404.32
0.01	13613.12	18177.96

Table 8: Average runtime (in ms) for MORE_SLACK and RANDOM tiebreaking policies across different ϵ values for $m = 500$, $m = 1000$, and $m = 2000$. Each column represents one version.

Observe that, in general, for smaller values of ϵ, the average run time of the two-sided penalty is greater than that of the one-sided penalty.

We also observed that the run time increased for the two-sided penalty as m increases, while for the one-sided penalty, the trend is reversed. The reason is that as m increases, more edges have a third component equal to 0 for the one-sided penalty. This simplifies computation and reduces the run time.

For the two-sided penalty, as m increases, the range of third-component values broadens, leading to more complex computations. The larger range introduces greater variability in edge costs, which increases both the time required for dominance checks and the size of the solution set. Additionally, the two-sided penalty requires evaluations for both long and short distances, further contributing to the higher run time as m grows.