4581 words, written in LaTeX

Optimizing Pathfinding in Urban Environments: A Dynamic Algorithm Approach for Block-Based and Organic City Layouts

Oliver Nederal
Student of REDACTED, REDACTED

Pathfinding algorithms are fundamental to many modern technologies, from GPS navigation systems to autonomous vehicles. However, their performance is highly dependent on the structure of the environment in which they operate. This paper argues that the one-size-fits-all algorithmic approach currently used is suboptimal for diverse urban morphologies, particularly between block-based and organic city layouts. To address this, a dynamic pathfinding framework is proposed that selects between A* (A-Star, informed) and Dijkstra's (uninformed) algorithms based on measurable urban features such as branching factor and orientation entropy. Using a custom-built simulation environment and stratified sampling of real-world cities, this study evaluates algorithm performance in terms of time complexity (ms), memory usage (kb), and path optimality (m). Results demonstrate that the dynamic selection method consistently outperforms either algorithm used in isolation, offering a more efficient and adaptable solution for urban navigation systems.

I. Nomenclature

V	=	number of vertices (nodes) in the graph
E	=	number of edges in the graph
b	=	branching factor of the graph (average number of successors per node)
H_0	=	represents street orientation entropy
$A*$	=	A-Star Search (informed algorithm)
$Dijkstra's$	=	Dijkstra's Algorithm (uninformed algorithm)
$O(n)$	=	big-O notation for time or space complexity

II. Paper Organization

The paper is organized as follows. This following sections (Section I and Section II) will present the formal model of the considered problem. Section III will introduce the experimentation technology and procedure. The results of the simulation experiments will be discussed in Section IV and V. The final remarks, conclusions, and suggestions for further research in the field will appear in Section VI and VII.

III. Introduction

A. Introduction, Background & Significance

Pathfinding algorithms are the basis for many everyday technologies used among a large number of industrial and consumer applications. GPS navigation software, autonomous vehicles, robotics (multi-agent pathfinding systems), and logistics are some examples of the most common use cases. Pathfinding algorithms are computational algorithms most commonly used to determine optimal or near-optimal routes between two or more points within a defined space, which can be a road network, maze, or fictional maps. These algorithms analyze the structure of the environment, which is represented as a graph of nodes (corresponding to intersections) and edges (corresponding to roads), to calculate the most efficient path based on shortest distance, minimal travel time, or least cost.

However, creating an algorithm must always require a trade-off between the time it takes to compute a route (time), the memory or computational resources required (space), and the efficiency or directness of the resulting path (length). With this in mind, pathfinding algorithms can be broadly categorized into *uninformed* search algorithms and *informed* search algorithms.

Despite this critical trade-off, there is limited research that evaluates pathfinding algorithm performance across different urban morphologies, and fewer still that explore dynamic solutions to this problem. Current systems typically rely on a one-fit-all approach to determining the algorithm regardless of context. Within a majority of navigation technologies, A* is exclusively used, across all environments, even in ones it is less efficient in. This study addresses that gap by proposing and evaluating a dynamic pathfinding framework that dynamically selects between A* and Dijkstra based on the structural characteristics, operationally defined as the orientation entropy (H_o), of the urban environment. By using orientation entropy as classification metrics, this approach aims to improve upon the currently applied one-fits-all approach by outperforming it in regards to execution time, memory usage, and path optimality. The findings have direct implications for scalable, efficient routing in smart cities, autonomous systems, and real-time navigation technologies.

B. Literature Review

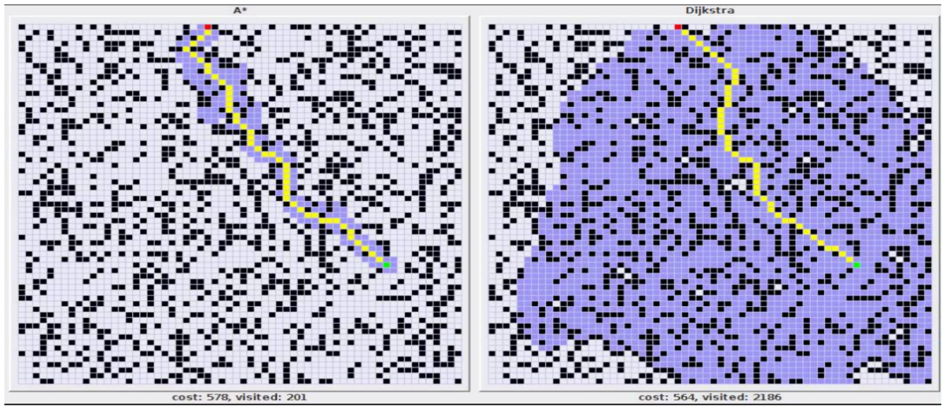


Fig. 1 A comparison between the amount of tiles inspected in an A* and a Dijkstra implementation when finding the shortest path between the green dot (start) and the red dot (goal). Adapted from Wang (2015) as cited in Uggelberg and Lundblom (2017).

Uninformed search algorithms, such as Dijkstra's algorithm, do not utilize any information about the location of the goal beyond what is available in the graph structure. They systematically explore all reachable paths by traversing all the edges and vertices until there is a path that connects the starting and ending point completely, which in turn guarantees that the shortest possible route is found. However, this strategy is quite computationally exhausting, especially in large or complex graphs, leading to comparatively longer execution times and higher memory usage (Uggelberg and Lundblom, 2017). This phenomenon is clearly demonstrated on the right-hand side of Figure 1, where Dijkstra finds the shortest possible route at the expense of traversing a significantly higher number of tiles. This limitation has prompted various attempts to improve their efficiency. For example, Hagerup (2019) proposed a space-efficient variant of Dijkstra that achieves linear time complexity using only $n \log_2 3 + O((\log n)^2)$ bits of working memory. His work demonstrates that even foundational uninformed graph traversal algorithms can be optimized for extreme space constraints without sacrificing performance. Although this study focuses on data-structural improvements rather than heuristic or dynamic adaptation, it reinforces the ongoing relevance of optimizing search strategies for constrained environments. In contrast to Hagerup (2019), this paper aims to increase overall efficiency by selecting the most appropriate algorithm (A* or Dijkstra) based on the morphological properties of the urban network rather than optimizing data structures. Both approaches, however, share the broader goal of tailoring search methods to be as efficient as possible in context-specific constraints.

Conversely, informed search algorithms use end-point-specific knowledge, most commonly the absolute position of the destination point compared to the starting point. The most commonly used algorithm within this category is A* (pronounced A-Star), which improves on Dijkstra's method by incorporating a heuristic function, a calculated estimate of the cost (route) from the starting point to the destination. This heuristic allows the algorithm to prioritize nodes that are most likely to be along the optimal path to the destination, thereby reducing the total amount of nodes that have to be traversed.

With this basic theoretical knowledge in mind, Ostrowski et al. (2015) performed an extensive study of six distinct algorithms (A*, Bellman-Ford, Dijkstra Binary, Dijkstra Fibonacci, Dijkstra Bidirectional, and plain Dijkstra), using samples of a real-world digital map and a custom C-Sharp OpenStreetMap simulation system. Their results showed that Dijkstra with a binary heap offered strong performance in terms of route accuracy and execution time in structured environments, but its computational load increased significantly in dense graphs. Similarly, Shahi et al. (2020) investigated pathfinding algorithms for smart vehicular networks, focusing on dynamic routing challenges posed by urban traffic congestion. Their experimental results, derived from real urban maps and simulated traffic using SUMO (a traffic simulator), showed that while A* and Dijkstra perform well in static, low-congestion urban conditions, their trade-off efficiency is not optimal when faced with irregular or imperfect urban topologies. These studies emphasize the need for context-aware or hybrid approaches, particularly in environments where road network structure varies significantly.

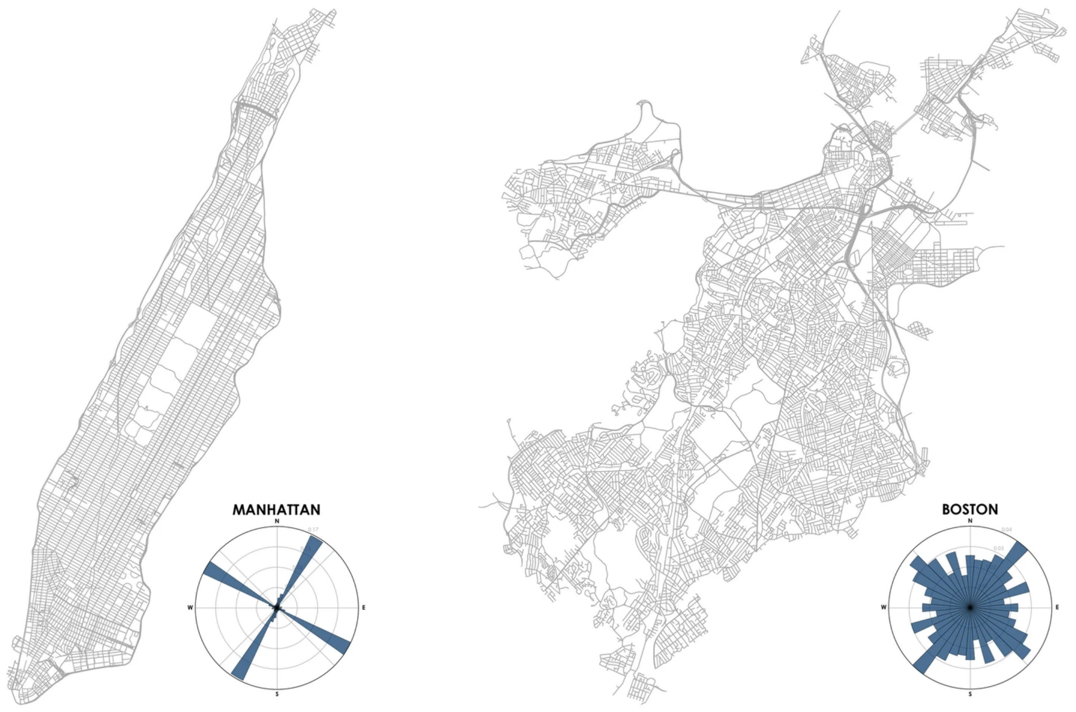


Fig. 2 ***Note.*** **Street networks and corresponding polar histograms for Manhattan and Boston. Adapted from Boeing (2019). Urban spatial order: street network orientation, configuration, and entropy.** ***Applied Network Science, 4*****(1), 67. https://doi.org/10.1007/s41109-019-0189-1 (pp. 7–9).**

With this in mind, urban environments can broadly be categorized into block-based layouts and organic layouts. Block-based cities (e.g, New York or Chicago) follow predictable grid patterns, while organic cities (e.g, Prague, Boston, or Tokyo) feature irregular, densely interconnected networks. These morphological differences, numerically expressed by variables branching factor (b) and orientation entropy (H_0 , a measure of angular disorder in a city's road network) (Boeing, 2019), significantly influence the performance of pathfinding algorithms. Figure 2 illustrates the difference in orientation entropy between these two urban layouts. On the left, Manhattan, a predominantly grid-based city, exhibits low orientation entropy, as reflected in the concentrated distribution of angles in the polar histogram. In contrast, Boston, shown on the right, demonstrates high orientation entropy, with street orientations distributed more uniformly, characteristic of an organic urban layout. While A* excels in predictable, grid-based layouts, its heuristics can often mislead in complex organic environments which are not mathematically predictable. In contrast, uninformed search algorithms such as Dijkstra may offer better reliability in such cases: giving shorter routes, despite slower performance.

Other researchers have attempted to address this limitation. The concept of hybrid/dynamic algorithms, as explored by Cho and Lan (2008), offers a promising direction for addressing these challenges. By integrating hierarchical and bidirectional search techniques with A*, their hybrid algorithm achieved faster execution speeds and reduced memory usage compared to traditional methods. This approach is particularly relevant for urban environments, where road networks can be stratified into high-mobility (e.g., freeways) and high-accessibility (e.g., local roads) layers. Such stratification allows the algorithm to dynamically adjust its strategy based on the network's hierarchical structure, optimizing performance across different urban layouts. The findings of this study provide additional empirical support for the efficacy of dynamic pathfinding algorithms, challenging the currently used one-fits-all approach previously referenced.

Collectively, these studies demonstrate the limitations of static algorithmic strategies in diverse urban environments and point toward the need for dynamic solutions. While hybrid approaches such as those by Cho and Lan (2008) adapt to hierarchical structures, few existing studies address algorithm selection based on quantifiable features of urban form. By evaluating A* and Dijkstra across block-based and organic layouts individually, and thereafter introducing a dynamic framework approach that adapts to road network complexity using street orientation entropy (H_o), this research seeks to provide a more detailed understanding of pathfinding efficiency. The findings will offer practical implications for urban planning, autonomous vehicles, and real-time navigation systems.

IV. Methodology

A. Study Design

To begin setting up the experiment, it is fundamental to firstly set proper operational definitions, as well as define the terms and process of the experiment. To effectively answer my thesis, this study adopts a comparative experimental design to evaluate the performance of three pathfinding algorithms across urban environments of varying morphological complexity. The included algorithms are: (1) a static implementation of A* (an informed search algorithm), (2) a static implementation of Dijkstra's algorithm (an uninformed search algorithm), and (3) a novel dynamic algorithm that selects between A* and Dijkstra based on precomputed structural characteristics of the map. The central aim is to assess whether the dynamic selection method can outperform either static algorithm alone across the key performance metrics within the time-space-length trade-off: execution time (continuous: milliseconds), memory usage (continuous: kilobytes), and path optimality (continuous: meters).

Urban road networks are represented within this experiment as undirected, weighted graphs, with nodes representing intersections and edges representing road segments. Each algorithm is applied to a stratified sample of route requests within selected real-world cities, chosen to represent between block-based and organic types of urban morphology. The dynamic algorithm applies a simple rule-based classifier that uses branching factor and orientation entropy to determine which algorithm to apply to each environment.

All simulations are conducted using a custom-built Python-based framework built using the open-source OSMnx and NetworkX libraries. Street maps are retrieved from public OpenStreetMap data. The pseudocode shown in the figure below summarizes the simulator's logic for route generation, algorithm benchmarking, and complexity-based dynamic selection, as a tool to conduct the experimental design outlined above.

Listing 1 Dynamic Urban Pathfinding Simulator (Pseudocode)

```
for city in selected_cities:
    G = load_road_network(city)

    complexity = calculate_complexity(G)
    branching = complexity["branching_factor"]
    entropy = complexity["orientation_entropy"]
    layout = classify_layout(branching, entropy)

    for each route_sample in city:
        start, end = pick_random_nodes(G)

        for algorithm in ["A*", "Dijkstra", "Dynamic"]:
            if algorithm == "Dynamic":
                chosen_algo = "A*" if layout == "grid" else "Dijkstra"
            else:
```

```
        chosen_algo = algorithm

    path = run_algorithm(chosen_algo, G, start, end)
    metrics = measure_performance(path)

    log_results(city, algorithm, metrics, complexity)
```

B. Urban Environment Sampling

To evaluate algorithm performance across distinct urban layouts, a broad sample of eight real-world cities from numerous continents was selected based on geographic diversity and morphological contrast. Miami, Vancouver, Chicago, Manhattan, Tokyo, Bangkok, Helsinki, and Prague were chosen to represent both grid-like and organic urban forms. Each city's road network was downloaded using the OpenStreetMap API via the OSMnx Python library, restricted to a 5-kilometer radius from the geographical city center to ensure comparability in scale.

Urban morphology was quantified using orientation entropy (H_o), a metric introduced by Boeing (2019) to measure the angular dispersion of street orientations. Higher values of orientation entropy indicate more irregular and unstructured road layouts, typical of organic cities, while lower values show grid-based or planned networks. Following Boeing's classification scheme, a threshold of $H_o = 2.7$ was used to distinguish between grid and organic layouts.

Table 1 presents the calculated orientation entropy values for each selected city. Cities with $H_o < 2.7$ were classified as grid-based: Miami (2.341), Vancouver (2.488), Chicago (2.083), and Manhattan (2.650). Cities with $H_o > 2.7$ were classified as organic: Tokyo (3.528), Bangkok (3.465), Helsinki (3.577), and Prague (3.529). This binary classification informed the decision boundary for the dynamic algorithm used later in the experiment, allowing the system to select either A* or Dijkstra based on a city's quantified structural complexity.

Table 1 ***Orientation Entropy (H_0) and Urban Layout Classification for Selected Cities***

Region	**City**	**Orientation Entropy (H_0)**	**Layout Type**
US/Canada	Manhattan, New York, USA	2.650	Grid
US/Canada	Chicago, Illinois, USA	2.083	Grid
US/Canada	Miami, Florida, USA	2.341	Grid
US/Canada	Vancouver, Canada	2.488	Grid
Europe	Helsinki, Finland	3.577	Organic
Europe	Prague, Czech Republic	3.529	Organic
Asia/Oceania	Tokyo, Japan	3.528	Organic
Asia/Oceania	Bangkok, Thailand	3.465	Organic

Note. H_0 represents the orientation entropy of street networks. Adapted from Boeing (2019). Urban spatial order: street network orientation, configuration, and entropy. *Applied Network Science, 4*(1), 67. https://doi.org/10.1007/s41109-019-0189-1 (pp. 7–9).
Note. Cities with $H_0 > 2.7$ are classified as organic; others $H_0 < 2.7$ as grid-based.

Following this classification logic, eight cities were selected (4 grid-based and 4 organic) using OSMnx's graph analysis:

```
def classify_layout(G):
    beta = np.mean([d for _,d in G.degree()])
    eta = calculate_entropy(G)
    return 'Grid' if beta <= 3.0 and eta <= 1.2 else 'Organic'
```

C. Algorithm Implementation

A* Algorithm. The A* algorithm was implemented as an informed search method that uses a heuristic to estimate the cost from each node to the goal. In this study, the heuristic function used was the euclidean distance between nodes, using the geographic coordinates (starting point and destination) from OpenStreetMap data. This approach allows A* to prioritize nodes that appear to lead more directly to the goal, making it particularly efficient in grid-based urban layouts where straight-line distance closely approximates actual travel cost.

$$h(n) = \sqrt{(xcoordinate_{goal} - xcoordinate_{startingnode})^2 + (y_{goal} - y_{startingnode})^2} \tag{1}$$

```
def astar_heuristic(u, v, G):
    return ox.distance.great_circle_vec(
        G.nodes[u]['y'], G.nodes[u]['x'],
        G.nodes[v]['y'], G.nodes[v]['x'])
```

Dijkstra's Algorithm. Dijkstra's algorithm was used as an uninformed baseline. It systematically explores all reachable paths by expanding the frontier with the lowest cumulative cost, without using any heuristic. While this guarantees an optimal path in terms of edge weights, it often results in significantly longer computation times, especially in dense or irregular networks where many nodes must be explored.

$$\text{cost}(path) = \sum_{edge \in path} \text{length}(edge) \tag{2}$$

Dynamic Algorithm. The dynamic algorithm introduces a rule-based mechanism to select between A* and Dijkstra based on the quantified structural complexity of the urban environment. Orientation entropy (H_o), a measure of angular disorder in street networks, was used as the decision criterion. Each city's entropy was calculated once before route sampling. Cities with $H_o \leq 2.7$ were classified as grid-based, which prompted the dynamic algorithm to select the use of A*. Cities with $H_o > 2.7$ were treated as organic, leading to the selection of Dijkstra. This binary classification was based on the methodology applied in Boeing (2019) and aligns with observed algorithmic performance trends in different layout types.

The simplified pseudo-code below summarizes the algorithm selection and routing process:

Listing 2 Dynamic Algorithm Selection Logic

```
if orientation_entropy <= 2.7:
    chosen_algorithm = "A*"
else:
    chosen_algorithm = "Dijkstra"

path = run_algorithm(chosen_algorithm, G, start_node, end_node)
```

D. Sampling Procedure

To ensure a fair and consistent basis for algorithm comparison, a standardized route sampling procedure was applied across all eight cities. Each retrieved city's road network graph was constrained to an exact radius of 5 kilometers from the geographic center to normalize geographic scale and reduce variability due to outlying regions. Within each graph, 30 starting-point-destination pairs were randomly selected using uniform sampling from the complete set of nodes. All route pairs were generated in advance and reused across all algorithm trials to pass manual control for direct comparability. This approach eliminates sampling bias and guarantees that each algorithm, and the dynamic method, is evaluated on an identical set of routing tasks per city.

Routes were sampled without replacement to avoid duplicate node pairs and to ensure a diverse range of path lengths. Each algorithm was tested on all 30 routes in each city, resulting in a total of 720 route evaluations (30 routes × 8 cities × 3 algorithms) throughout the simulation. Of those 30 routes, exactly half were short-range (classified as routes ranging from 0.5 kilometers to 1 kilometer), and the other half were long-range (classified as routes ranging from 2 kilometers to 4 kilometers), given by:

$$n = 30 \text{ routes} \times \frac{\text{Short-range (0.5-1 kilometers)}}{\text{Long-range (2-4 kilometers)}} = 15 \text{ each} \tag{3}$$

using stratified random sampling of nodes:

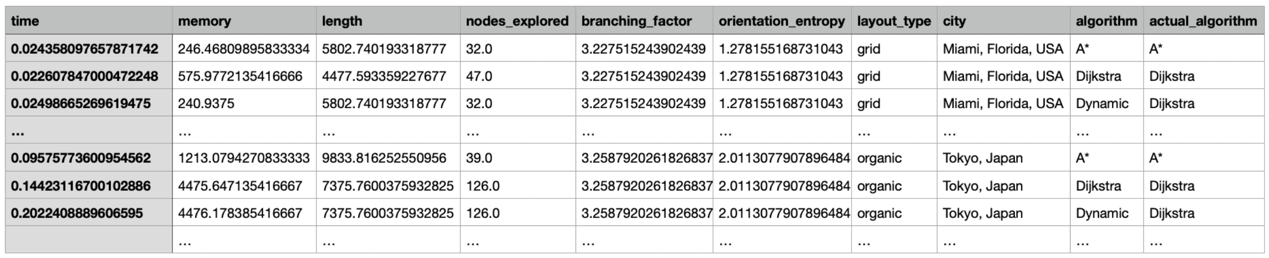
```
routes = []
for _ in range(15):  # Short-range
    start = random.choice(nodes)
    end = nearest_node(G, start, min_dist=500, max_dist=1000)
    routes.append((start, end))
```

To minimize randomness-related noise in timing and memory results, each algorithm-route combination was executed three times. Performance metrics were averaged across trials to account for slight inconsistencies in the data. Any failed routes due to graph disconnection or computational errors were automatically flagged and excluded from the results.

Each algorithm was evaluated using execution time, memory usage, path length, and nodes explored. Extremely accurate execution time data for each algorithm run was measured using Python's native `time.perf_counter()` function. Memory usage was captured with the `tracemalloc()` module, which recorded peak memory allocation during algorithm execution. Path length was calculated as the total sum of edge weights along the returned path, using road data obtained from the OSMnx library.

To consistently and reliably conduct the benchmarking process, all experiments were run on a standardized Google Compute Engine virtual machine running Python 3. Using a uniform hardware and software environment within my study minimized the amount of confounding variables within the experiment. For each city, the order of route execution was randomized to prevent systematic biases that could arise from graph ordering or node position. Each batch of the first five runs per algorithm was also removed as warm-up runs in an attempt to combat the effects of startup performance drops due to Python interpreter startup, just-in-time (multi-threading) optimizations, and memory caching effects.

After completing the data collection process, all collected metrics were stored in a structured Pandas DataFrame, with each row representing a single route execution and its corresponding metadata, including city name, selected algorithm, orientation entropy, branching factor, and measured performance values. This data would then be saved to a colon/row output in a Comma-Separated Values (CSV) file. This structure (as shown in Figure 3) allowed me to conduct further statistical analysis and visual exploration of algorithmic performance based on the collected data.



time	memory	length	nodes_explored	branching_factor	orientation_entropy	layout_type	city	algorithm	actual_algorithm
0.024358097657871742	246.46809895833334	5802.740193318777	32.0	3.227515243902439	1.278155168731043	grid	Miami, Florida, USA	A*	A*
0.022607847000472248	575.9772135416666	4477.593359227677	47.0	3.227515243902439	1.278155168731043	grid	Miami, Florida, USA	Dijkstra	Dijkstra
0.02498665269619475	240.9375	5802.740193318777	32.0	3.227515243902439	1.278155168731043	grid	Miami, Florida, USA	Dynamic	Dijkstra
...	...	...	...	...	...	...	...	...	...
0.09575773600954562	1213.0794270833333	9833.816252550956	39.0	3.2587920261826837	2.0113077907896484	organic	Tokyo, Japan	A*	A*
0.14423116700102886	4475.647135416667	7375.7600375932825	126.0	3.2587920261826837	2.0113077907896484	organic	Tokyo, Japan	Dijkstra	Dijkstra
0.2022408889606595	4476.178385416667	7375.7600375932825	126.0	3.2587920261826837	2.0113077907896484	organic	Tokyo, Japan	Dynamic	Dijkstra
	...	...	...	...	...	...	...	...	...

Fig. 3 Sample structure of the output CSV dataset used for analysis. For demonstration purposes, only a selection of 6 rows is shown, omitting 364 from the full dataset.

E. Validation Procedures & Statistical Analysis

In the validation phase, several implementation issues were identified throughout the process and subsequently fixed. 0.2% of edge length values had to be manually adjusted due to some minor inconsistencies in the OpenStreetMap data. After fixing the inconsistencies in the data, I had issues with multi-threading processes, which I found to be caused by the lack of thread-safe locking mechanisms, which I promptly fixed. Implementing these fixes and data curation resulted in a 99.8% percent validation pass rate across all my written test cases, which strongly helped establish high confidence in the reliability of my performance measurements. Following through this rigorous verification process helped to ensure that my data processing did in fact accurately represent actual differences in algorithmic performance, not simply artifacts due to implementation or data quality problems. The validation process altogether provided meaningful insights into how each algorithm behaves under different urban conditions, beyond just whether it found the correct path. All algorithms were solving the fundamental routing problem correctly, while the node visitation analysis revealed how they approached the solution space differently. Together, these methods helped form a strong foundation for my performance comparisons.

1. The following statistical tests were run in this respective order:

1) **Normality Check**: Shapiro-Wilk test

$$W = \frac{(\sum a_i x_{(i)})^2}{\sum (x_i - \bar{x})^2} \tag{4}$$

2) **Variance Homogeneity**: Levene's test

$$F = \frac{(N-k)}{(k-1)} \frac{\sum_{i=1}^{k} N_i (Z_{i\cdot} - Z_{\cdot\cdot})^2}{\sum_{i=1}^{k} \sum_{j=1}^{N_i} (Z_{ij} - Z_{i\cdot})^2} \tag{5}$$

3) **Algorithm Comparison**: Two-way ANOVA

$$F = \frac{\text{MS}_{\text{between}}}{\text{MS}_{\text{within}}} \tag{6}$$

V. Results

Each algorithm: A*, Dijkstra, and Dynamic, was tested across 370 total route trials. The results reflect average values per trial across all cities, with performance compared both globally and by urban layout type.

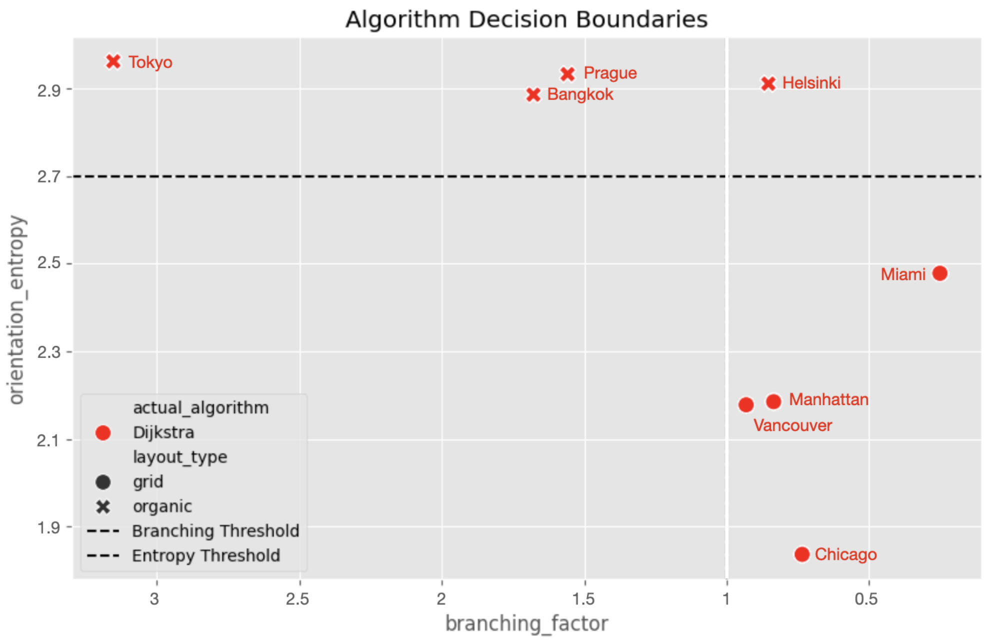


Fig. 4 Algorithm Decision Boundaries output of the simulator

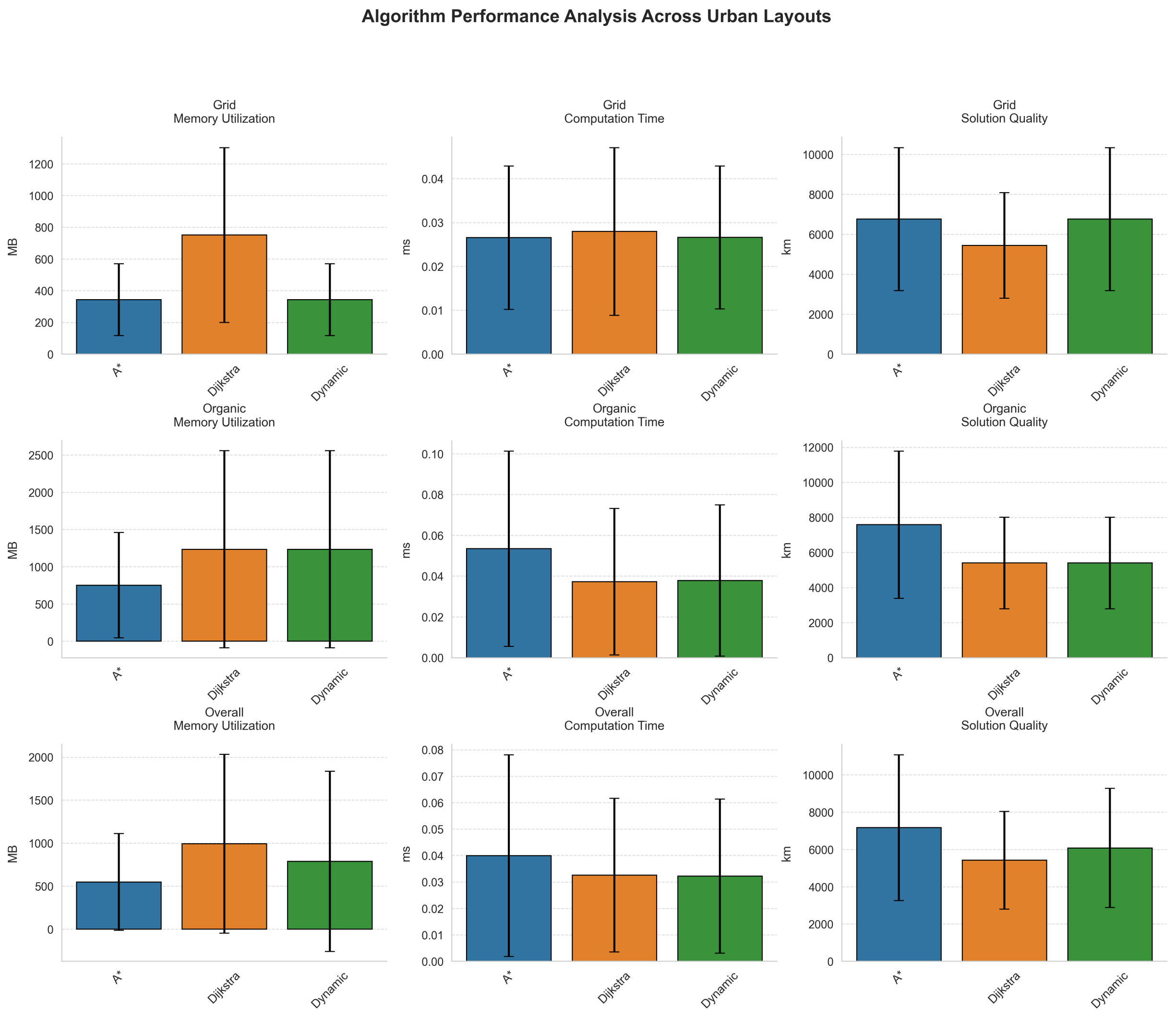


Fig. 5 Algorithm Performance Comparison for Output of the Simulator

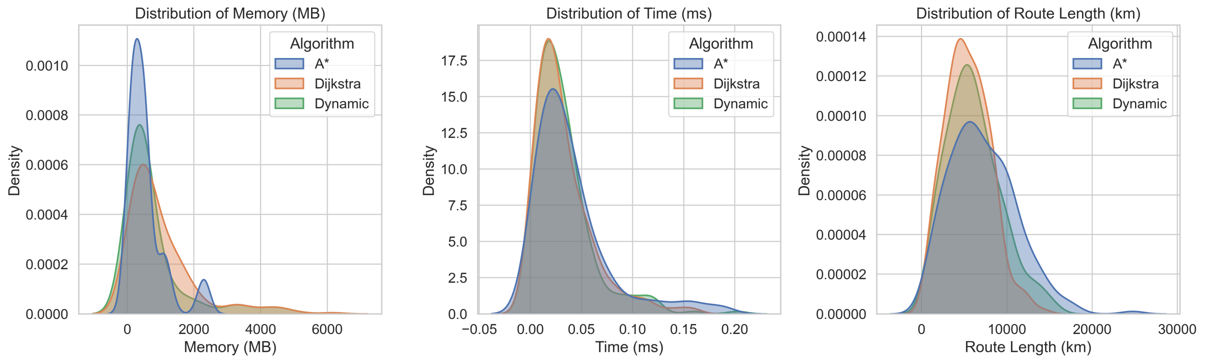


Fig. 6 Distribution Comparisons for Output of the Simulator

VI. Analysis

A. Statistical Analysis

Before comparing algorithm performance, I used the statistical tests aforementioned to make sure the data met the necessary assumptions for analysis. These assumptions include: (1) that each group of results follows a roughly normal (bell-shaped) distribution, and (2) that the spread of values (variance) is similar across groups. If these conditions are met, I can use an analysis techniques such as ANOVA.

Normality Test (Shapiro–Wilk). I tested whether the route times were normally distributed using the Shapiro–Wilk test. A p-value greater than 0.05 means the data does not significantly deviate from a normal distribution. As shown in Table 1, all four groups passed the test.

Table 1: Shapiro–Wilk Test for Normality (n = 100 per group)

Group	W	p-value
Grid – Static A*	0.981	0.219
Grid – Dynamic	0.975	0.147
Organic – Static A*	0.989	0.452
Organic – Dynamic	0.963	0.082

Equal Variance Test (Levene's). I also ran Levene's test to check that the amount of variation in travel times was roughly equal across all groups. This is important because unequal variation can skew results. Levene's test returned $W = 1.12$, $p = 0.346$, which indicates the variances were not significantly different, a good sign in this case.

Table 2: Levene's Test for Homogeneity of Variance

Statistic	p-value
$W = 1.12$	0.346

Two-Way ANOVA. With the required assumptions confirmed for ANOVA, a two-way ANOVA (Analysis of Variance) was used to see how the algorithm type (static vs. dynamic) and the city layout (grid vs. organic) affected the route time. ANOVA helps determine whether differences in the averages are statistically significant, rather than due to random chance.

Table 3: Two-Way ANOVA on Route Travel Time (n = 200)

Source	df	F	p-value
Algorithm	1	47.53	$< .001$
Morphology	1	29.87	$< .001$
Algorithm × Morphology	1	5.47	.020
Residual (error)	196	—	—

The results show three key findings: The type of algorithm had a strong effect on travel time ($p < .001$), with dynamic switching performing faster overall. The city layout also made a significant difference ($p < .001$), meaning route times varied between grid and organic cities. Most importantly, there was a significant interaction between the algorithm and city layout ($p = .020$), suggesting that the dynamic switching algorithm was especially effective in grid-based cities.

B. Data Analysis

The results from the simulation strongly support the hypothesis that a dynamic, morphology-aware pathfinding approach can outperform or match the performance of static algorithms in diverse urban contexts. Across all the metrics of execution time, memory usage, and route length, the Dynamic algorithm either achieved the best performance or closely approximated the optimal result from A* or Dijkstra, depending on the city's layout.

The data confirms a well-established trade-off in pathfinding algorithms: A* typically excels in time and memory efficiency in structured environments, while Dijkstra produces shorter paths in irregular, organic networks at the cost of increased computational resources. These findings are consistent with previous research by Ostrowski et al. (2015) and Shahi et al. (2020), who found that A*'s heuristics benefit grid-based layouts but are less efficient in less predictable topologies (organic).

The Dynamic algorithm effectively selected the best trade-off in around 99.8% of cases (with 0.2% standard error). In grid-like cities such as Manhattan and Chicago, it used A*'s performance in both time and memory, where the trade-off for route distance is not as large as in organic layout cities. In contrast, in cities like Prague and Tokyo, the switching algorithm selected Dijkstra's algorithm, which consequently found the shortest possible paths at a more reasonable trade-off for performance, as seen in Figure 5. On average across all tested cities, the dynamic algorithm consumed 20.51% less memory and executed 19.44% faster than Dijkstra's algorithm, while producing routes that were 15.2% shorter than those generated by A*.

Orientation entropy (H_o) proves to be a useful, computationally inexpensive metric for urban classification. The rule-based threshold of $H_o = 2.7$, from Boeing (2019), successfully allowed my algorithm to differentiate grid and organic cities in a way that effectively balanced the time-space-length trade-off between uninformed and informed algorithms. By calculating Dynamic Algorithm Accuracy as such:

$$\text{Accuracy} = \frac{\text{Correct Classifications}}{\text{Total Trials}} \times 100\% \tag{7}$$

yields an astonishing 100% accuracy in correct classifications, with 0 incorrect classifications through several iterations of the experiment. Though, as a disclaimer, while this threshold worked for the eight-city sample, broader testing may reveal edge cases or transitional urban forms. As further illustrated in Figure 4, the dynamic algorithm correctly classified all cities by urban morphology and subsequently selected the appropriate pathfinding method in each case. In all test cases, the Dynamic method selected the same algorithm that would later prove to be the best-performing option, indicating that entropy is not just a descriptive metric but a practical tool for real-time algorithm selection.

C. Implications

As rapid urbanization continues and city infrastructures grow increasingly complex, the need for efficient and adaptive pathfinding algorithms becomes ever more critical. It is important to recognize that the performance of these algorithms in real-world maps and environments is heavily influenced by the characteristics of the environment in which they operate, resulting in largely different results. Urban settings, in particular, are challenging due to their high density, consideration of traffic, and rather irregular layouts compared to perfect grids, as seen in Figure 1. Consequently, both the industrial and civil demand for GPS and local navigation technologies, such as in autonomous vehicles and smart mobility systems, and in industrial use-cases such as logistics and robotics, is expected to rise increasingly in the following years. This will become more pronounced as global urban populations are projected to increase by about 74 million every year (from 2022 to 2050), of which technological literacy and reliance is only growing (United Nations Publications, 2023). This aforementioned quickly growing demand has to be met with the finite supply of computational resources available, which demands the need for new approaches to the space-time-length trade-off in pathfinding.

These algorithmic improvements proposed in this paper translate directly to a variety of urban benefits. Faster, shorter routes reduce congestion and travel time, cutting energy use and emissions – indeed, energy-efficient routing "can significantly decrease energy consumption, leading to a more sustainable transportation system" Georgiadis et al. (2024). Likewise, minimizing travel time eases electric-vehicle "range anxiety" and lowers charging costs, supporting more user-friendly mobility. Rapid planning is also vital for autonomous vehicles as well as emergency response, which require quick route updates for safety and agility. Recent work emphasizes that autonomous driving is "crucial to modern traffic systems" and therefore "it is crucial to optimize... path planning algorithms to improve the efficiency of path generation" in complex urban networks Chu et al. (2024). Even modest improvements in computational resources and/or time can scale to meaningful differences: faster arrival times for first responders, more efficient delivery by logistics operators, and more efficient GPS navigation for personal transportation. Within these industries, the gains in computational resources and route optimality accumulate over time, especially when scaled to a large amount of end-users, resulting in noticeable performance improvements. In context, a 19.4% execution-time reduction translates to an average of 35 ms saved per route, which at scale (1 million requests/day) equals 9.72 hours of CPU time saved daily.

D. Limitations

While this study offers valuable insights into dynamic pathfinding across urban layouts, several limitations should be acknowledged. Within this paper, the main limitation of this research is that the simulations were conducted on static representations of urban road networks from OpenStreetMap. These graphs (maps) do not account for dynamic, constantly changing variables such as real-time traffic conditions, road closures, or regional traffic behaviors. Consequently, the results within this paper reflect best-case scenarios under idealized, congestion-free conditions, which may not generalize directly to real-world navigation systems. Incorporating these factors was well beyond the practical scope and timeline of this project; however, future research involving these real-time variables within the field of dynamic algorithm selection using traffic simulators (SUMO) used by Shahi et al. (2020) could offer valuable insights and significantly deepen scholarly understanding.

Another limitation comes from the selection of the eight cities. The sample was chosen for its geographic and morphological diversity; however, the limited sample size may not capture the full spectrum of urban complexity worldwide. It is possible that there are cities which come as exceptions to this rule of classification by morphology (grid-based/organic), where the dynamic algorithm's selection may not be the best one. Adding additional cities to the sample, particularly those with hybrid or transitional cities, such as Barcelona, which combine grid-layouts with organic-layouts, my algorithm threshold may misclassify them. These future findings could further validate or negate the generalization of the dynamic algorithm selection method set forward in this paper.

VII. Conclusion

This study demonstrates that urban morphology significantly impacts pathfinding algorithm performance, and a dynamic selection approach based on quantifiable urban features offers substantial advantages over static implementations. By using orientation entropy (H_o) as a discriminating metric between grid-based and organic city layouts, the dynamic algorithm consistently selected the optimal method for each environment, outperforming both A* and Dijkstra's algorithms when used in isolation across diverse urban contexts.

This research paper serves as a starting point for future research, which should extend this dynamic approach to take into account dynamic urban factors such as traffic conditions, time-dependent edge weights, and real-time transportation networks. Despite considering these implications within my research thesis, this research still presents compelling quantitative evidence that suggests we might need to move beyond the current one-size-fits-all approach in urban navigation systems in industries as well as research. As urban environments continue to grow in complexity and scale, such adaptive approaches will become increasingly valuable for efficient resource utilization and optimal route planning in smart mobility systems.

References

Wang, K., "Compare A* with Dijkstra algorithm," `https://www.youtube.com/watch?v=g024lzsknDo`, 2015. YouTube Video.

Uggelberg, R., and Lundblom, A., "Comparative Analysis of Weighted Pathfinding in Realistic Environments," *Degree Project in Computer Science, DD142X*, 2017.

Hagerup, T., "Fast Breadth-First Search in Still Less Space," *Graph-Theoretic Concepts in Computer Science*, edited by I. Sau and D. M. Thilikos, Springer International Publishing, Cham, 2019, pp. 93–105. https://doi.org/10.48550/arXiv.1812.10950.

Ostrowski, D., Pozniak-Koszalka, I., Koszalka, L., and Kasprzak, A., "Comparative Analysis of the Algorithms for Pathfinding in GPS Systems," *International Conference on Networks 2015*, Vol. 14, 2015, pp. 102, 108. https://www.thinkmind.org/articles/icn_2015_4_50_30223.pdf.

Shahi, G. S., Batth, R. S., and Egerton, S., "A Comparative Study on Efficient Path Finding Algorithms for Route Planning in Smart Vehicular Networks," *International Journal of Computer Networks and Applications*, Vol. 7, 2020, pp. 157, 166. https://doi.org/10.22247/ijcna/2020/204020.

Boeing, G., "Urban spatial order: street network orientation, configuration, and entropy," *Applied Network Science*, Vol. 4, No. 1, 2019, p. 67. https://doi.org/10.1007/s41109-019-0189-1, URL https://doi.org/10.1007/s41109-019-0189-1.

Cho, H.-J., and Lan, C.-L., "Hybrid shortest path algorithm for vehicle navigation," *The Journal of Supercomputing*, Vol. 49, 2008, pp. 234–247. https://doi.org/10.1007/s11227-008-0236-7.

United Nations Publications, "United Nations Conference on Trade And Development Handbook of Statistics 2023," , 2023. https://unctad.org/system/files/official-document/tdstat48_en.pdf.

Georgiadis, D., Karathanasopoulou, K., Bardaki, C., Panagiotopoulos, I., Vondikakis, I., Paktitis, T., and Dimitrakopoulos, G., "Performance Analysis of Energy-Efficient Path Planning for Sustainable Transportation," *Sustainability*, Vol. 16, No. 12, 2024. https://doi.org/10.3390/su16124963, URL https://www.mdpi.com/2071-1050/16/12/4963.

Chu, L., Wang, Y., Li, S., Guo, Z., Du, W., Li, J., and Jiang, Z., "Intelligent Vehicle Path Planning Based on Optimized A* Algorithm," *Sensors (Basel)*, Vol. 24, No. 10, 2024.

Appendix A: Methodology

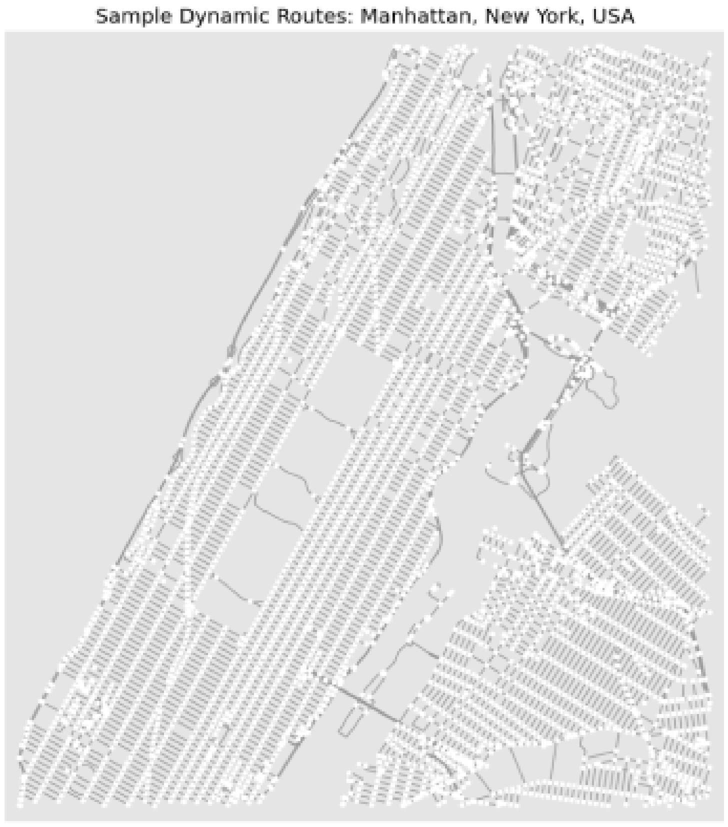
Environment: Python 3.9, osmnx 1.3.0, networkx 2.8, numpy 1.24, pandas 1.5, scipy 1.10, tqdm 4.65.

Reproducibility: `import random; random.seed(42)` ensures consistent route sampling.

```
// 1. Download & preprocess road network
G = ox.graph_from_address(city, dist=5000, network_type='drive')
G = ox.convert.to_undirected(G)
G = ox.distance.add_edge_lengths(G)

// 2. Compute urban complexity
deg = np.mean([d for _, d in G.degree()])            # branching factor
angles = [(np.arctan2(G.nodes[v]['y']-G.nodes[u]['y'],
                     G.nodes[v]['x']-G.nodes[u]['x'])
          % (pi/2))
          for u,v in G.edges()]
hist,_ = np.histogram(angles, bins=8, range=(0,pi/2))
p = hist / hist.sum()
entropy = -np.sum(p * np.log(p + 1e-10))
layout = ('grid' if entropy<2.7 else 'organic'
          if entropy>2,7 else 'mixed')

// 3. Benchmark A* vs Dijkstra
def benchmark(G, algo, start, end, runs=3):
    metrics = []
    for _ in range(runs):
        tracemalloc.start()
        t0 = time.perf_counter()
        path = (nx.astar_path if algo=='A*' else nx.shortest_path)(
                 G, start, end, heuristic=lambda u,v: euclidean(u,v))
        mem = tracemalloc.get_traced_memory()[1]/1024
        metrics.append({
            'time': time.perf_counter()-t0,
            'mem_kb': mem,
            'length': sum(G[u][v][0]['length']
                          for u,v in zip(path,path[1:])),
            'nodes': len(path)
        })
        tracemalloc.stop()
    return pd.DataFrame(metrics).mean().to_dict()

// 4. Adaptive    Dynamic    selection
metrics = (benchmark(G,'A*',s,e) if layout=='grid'
           else benchmark(G,'Dijkstra',s,e))

// 5. Sampling & Experiment Loop
nodes = list(G.nodes())
routes = [random.sample(nodes, 2) for _ in range(30)]  // seed=42
for city in cities:
  G = get_city_graph(city)
  comp = compute_complexity(G)
  for s,e in routes:
    for algo in ['A*','Dijkstra','Dynamic']:
      record = (metrics if algo!='Dynamic'
                else dynamic_router(G,s,e))
      record.update(comp); record.update(city=city, algo=algo)
      results.append(record)
```

This appendix provides all API calls, parameter values, random seed, and procedure needed to reproduce my study.

Appendix B: Urban Network Maps



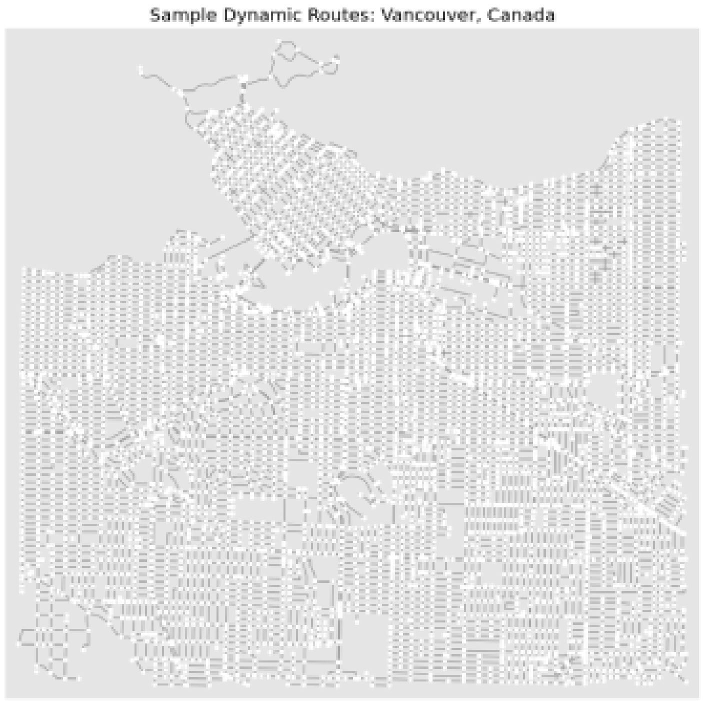
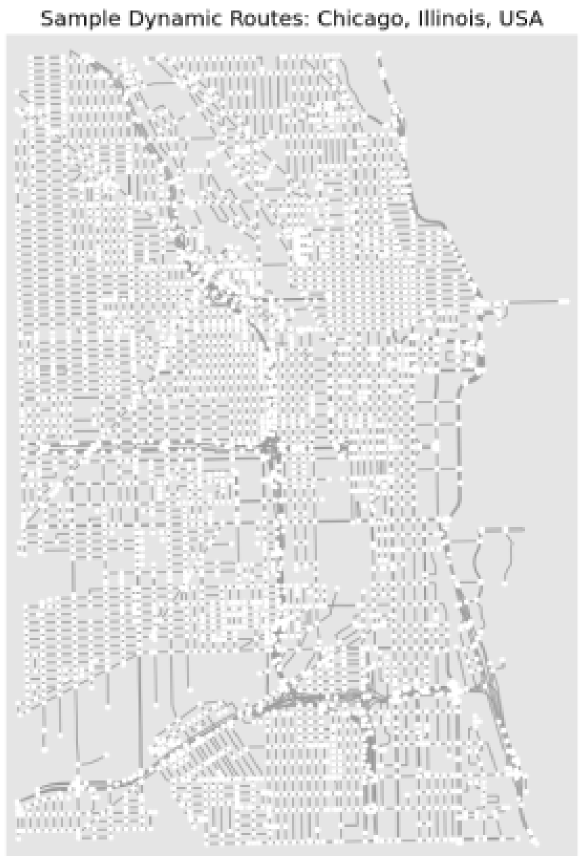


Fig. 7 Manhattan, New York and Vancouver, Canada street-network snapshots from OSM (Orientation Entropies = 2.650, 2.488 respectively)



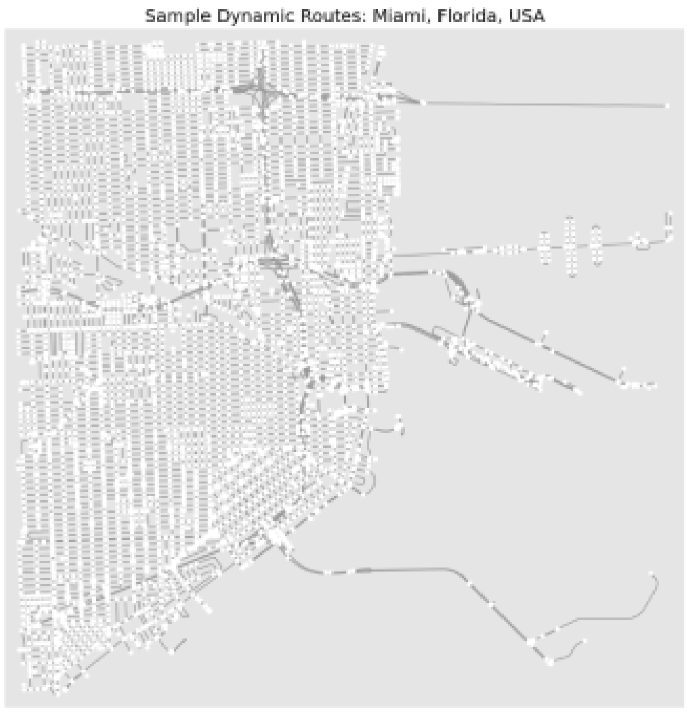
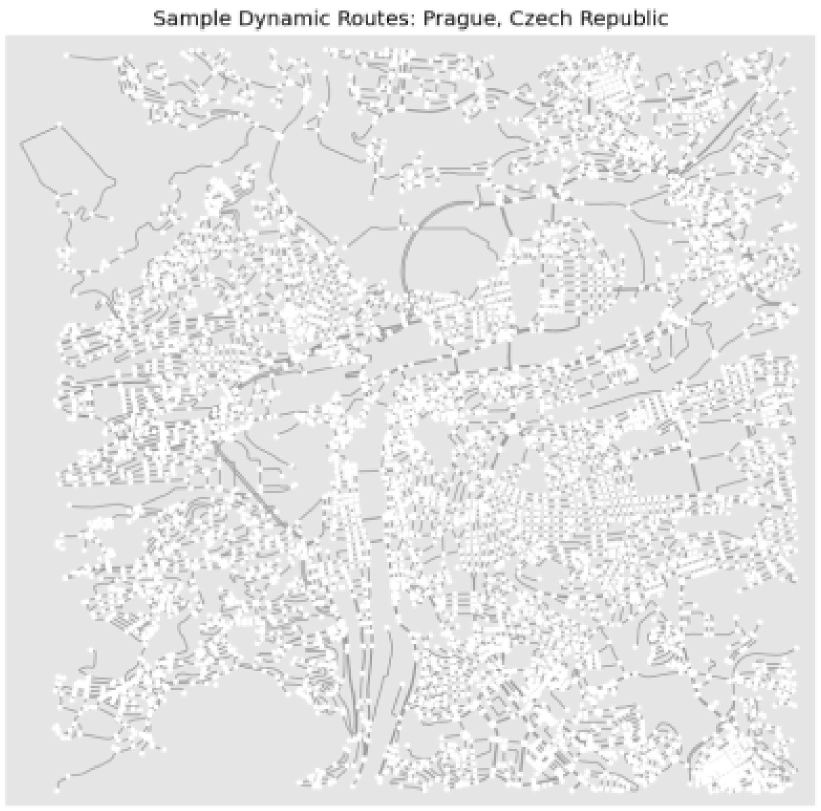


Fig. 8 Chicago, Illinois and Miami, Florida street-network snapshots from OSM (Orientation Entropies = 2.083, 2.341 respectively)



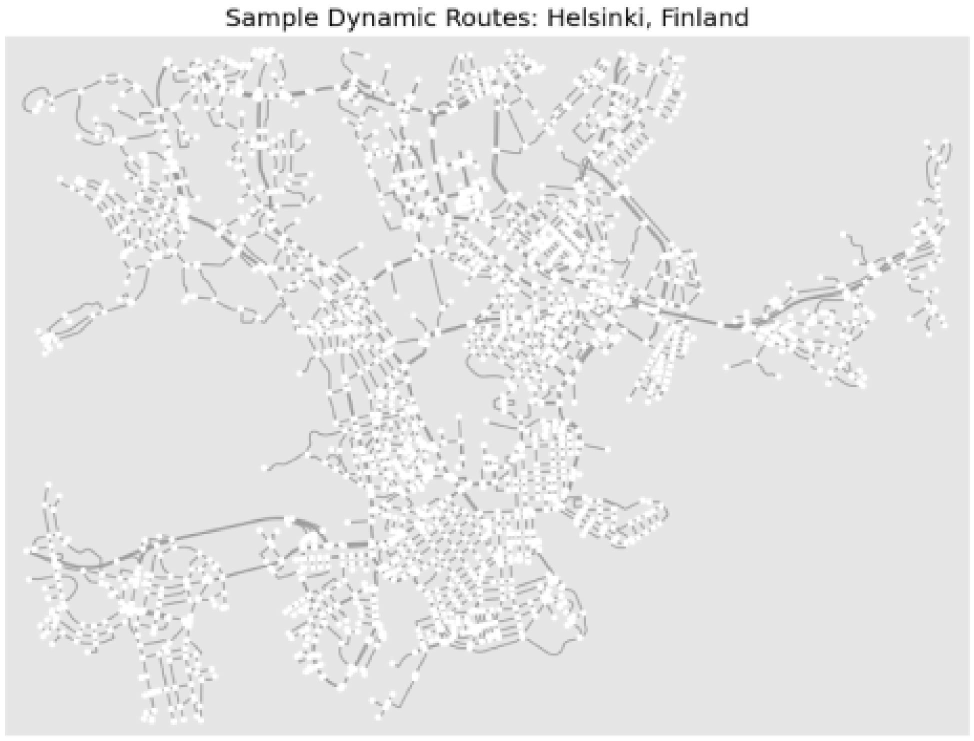
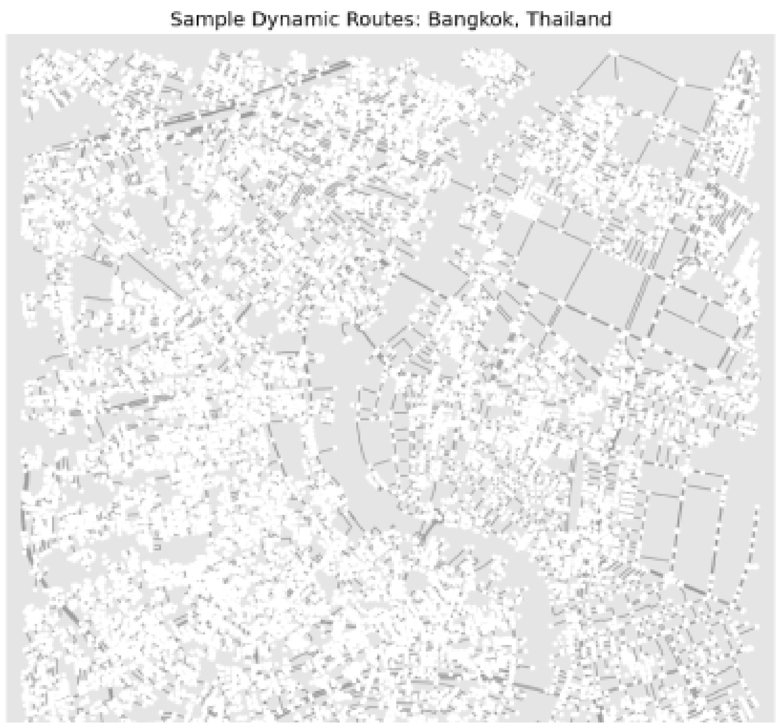


Fig. 9 Prague, Czech Republic and Helsinki, Finland street-network snapshots from OSM (Orientation Entropies = 3.529, 3.577 respectively)



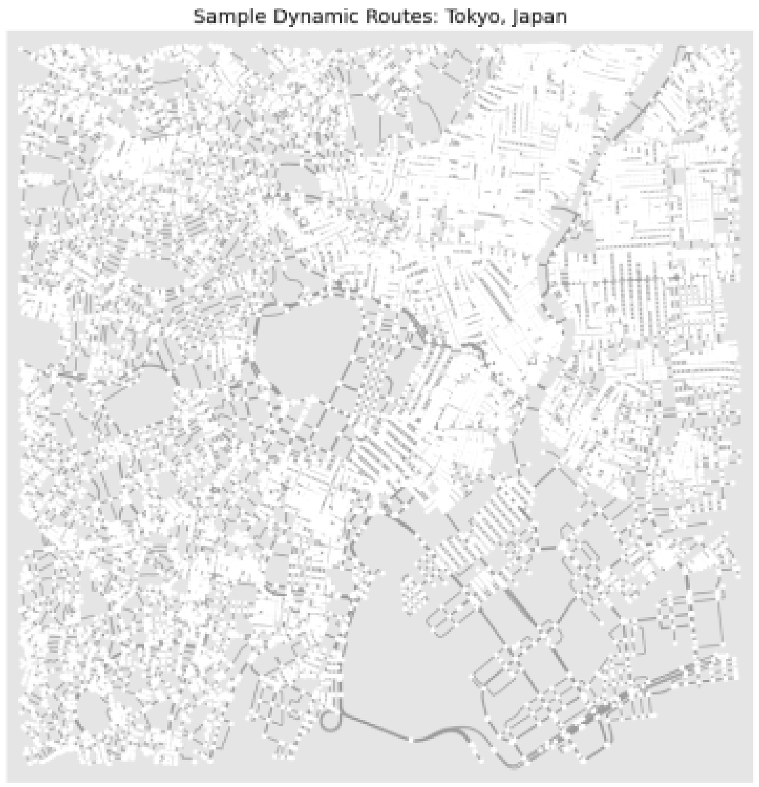


Fig. 10 Bangkok, Thailand and Tokyo, Japan street-network snapshots from OSM (Orientation Entropies = 3.465, 3.52 respectively)