

Dynamic Parking Space Allocation for Real-Time Urban Management

Dariush Ebrahimi[1(✉)], Darpan Rathwa[1], Ishan Shah[1], Rushil Shah[1], Krish Gohil[1], and Fadi Alzhouri[2]

[1] Department of Physics and Computer Science, Wilfrid Laurier University, 75 University Ave W, Waterloo, ON N2L 3C5, Canada
debrahimi@wlu.ca, {rath9450,shah8443,shah3670,gohi9630}@mylaurier.ca
[2] Department of Electrical and Computer Engineering, Gulf University for Science and Technology (GUST), Mishref, Kuwait
alzhouri.f@gust.edu.kw

Abstract. Urbanization and rising vehicle ownership have intensified the challenge of finding parking in metropolitan areas. This research presents the Dynamic Parking Space Allocation System (DPSAS), an advanced framework for real-time parking management optimization. The system models the urban environment as a weighted graph, where intersections and parking zones are nodes, and roads, affected by dynamic traffic conditions, form the edges. A key innovation is the use of hexagonal grid partitioning, which efficiently segments the urban space, reducing the search area and accelerating computational processes for real-time applications. Additionally, DPSAS features dynamic adaptation mechanisms that continuously recalibrate routes and parking availability in response to evolving traffic conditions. The system adapts to dynamic factors such as travel time, cost, and parking availability, ensuring user-centric optimization and adaptability in fluctuating urban settings. Extensive simulations demonstrate that DPSAS significantly reduces parking search times, optimizes traffic flow, and improves resource utilization. The results highlight the system's robustness and scalability, positioning DPSAS as a transformative solution to urban parking challenges, bridging gaps in existing systems, and enhancing urban mobility.

Keywords: Time-Dependent Shortest Path · Vehicle Routing with Time Windows · Dynamic Parking Allocation · Urban Mobility Optimization

1 Introduction

In recent years, urbanization and the increasing number of vehicles per household have led to a growing demand for parking in metropolitan areas [1]. As urban spaces expand, the challenge of finding reasonable parking increases due to limited parking supply and heightened traffic congestion. Traditional parking systems, which are typically static, fail to offer flexible, dynamic solutions

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2025
O. Gervasi et al. (Eds.): ICCSA 2025, LNCS 15650, pp. 204–220, 2025.
https://doi.org/10.1007/978-3-031-96962-1_14

that can adapt to changing traffic conditions and user requirements [2]. This not only results in long search times, high costs for private drivers, and worsening congestion, but also contributes to pollution and inefficiencies in urban mobility. Dynamic pricing has emerged as a key strategy for managing parking demand and availability [3,4].

The primary goal of this research is to develop smarter and more interactive parking solutions capable of handling new challenges such as real-time traffic fluctuations, current parking infrastructure status, and growing user demands. Recent studies highlight the difficulties faced by existing systems in real-time parking allocation, particularly due to the lack of seamless integration of traffic, location, and cost data in a scalable, user-friendly manner. Works such as [5] on parking optimization and [6] on traffic-aware navigation expose the limitations of current methods, which often fail to consider both traffic-induced delays and user preferences in real-time. While IoT-based parking systems and GPS navigation technology have seen improvements, a gap remains in their integration to provide more optimized and cost-effective solutions [6,7].

This paper proposes a dynamic parking allocation system that utilizes a greedy algorithm to assign parking spots based on real-time traffic conditions, proximity to destinations, travel time, and parking costs. The system models each parking zone as a node in a weighted graph, where intersections are also nodes representing potential delays due to congestion. It calculates the best parking spot within a 1–2 km radius of the user's destination, using a weighted formula that incorporates travel time, distance, and parking charges. The inclusion of traffic data at intersection nodes provides more accurate estimates of travel times to parking zones, enabling better decision-making.

The proposed system introduces several innovations that set it apart from existing solutions. Unlike traditional systems, it divides urban areas into hexagonal grids, allowing for more efficient spatial queries and reducing computational complexity [8–10]. The system combines traffic analytics with dynamic parking management in a unique way, enabling seamless updates and recalculations based on changing conditions. This offers superior adaptability compared to previous approaches. Additionally, the integration of hexagonal grids and dynamic graph updates reduces computational overhead, making the system scalable and capable of handling larger metropolitan networks—something that static or heuristic-based methods often fail to do [11].

This work contributes to the advancement of real-time parking systems by addressing both travel time and cost while incorporating traffic-induced delays. The remainder of this paper is organized as follows: Sect. 2 reviews related work in dynamic parking and traffic-aware navigation systems. Section 3 describes the setup of the proposed system, including the design of the data structure, algorithm, and infrastructure. Section 4 delves into the algorithm design and analysis, detailing the optimization techniques used. Section 5 presents the experimental setup and performance analysis, demonstrating the system's effectiveness under simulated conditions. Finally, Sect. 6 concludes the paper.

2 Literature Review

As urbanization continues and vehicle ownership rises globally, parking allocation systems have become a critical concern [1]. Modern parking systems often suffer from long search times for available parking spaces, excessive traffic, environmental harm, and inefficient use of resources [7]. Researchers have explored various approaches, including graph-based optimization, constraint modeling, dynamic pricing, and machine learning, to address these issues, but a unified, comprehensive framework remains elusive [2,8].

Dynamic parking allocation frameworks have emerged as one of the most promising solutions to address the inefficiencies of static systems. For example, [1] introduced a scalable framework that adjusts parking allocations dynamically using 0–1 programming models. This system relies on real-time GPS data to assign parking spaces to vehicles, helping reduce congestion and maximize resource utilization. However, it lacks dynamic pricing and fails to incorporate user-specific preferences, such as proximity to destinations or priority access.

In contrast, [2] proposed a constraint optimization model that considers user schedules, organizational constraints, and shifting priorities. The system reallocates unused parking spaces dynamically to ensure flexible utilization. Despite its benefits, this approach does not integrate real-time traffic data or include adaptive mechanisms to handle fluctuating demand, limiting its scalability and effectiveness in complex urban environments.

Dynamic pricing has become a vital mechanism for balancing parking demand and availability. In [7], the authors developed a game-theoretic model that uses machine learning to predict parking occupancy and adjust pricing dynamically. Their approach optimizes revenue for parking authorities while enhancing user satisfaction by considering the interaction between restricted parking users and parking pricing.

While the above approaches offer valuable insights, they tend to address individual aspects of the parking allocation problem rather than providing a unified solution [4]. Most systems focus either on allocation efficiency or revenue optimization, neglecting user-centric features like dynamic prioritization, real-time traffic adjustments, and equitable resource distribution. Furthermore, reliance on static datasets or predefined parameters often hinders the scalability and adaptability of these systems in real-world conditions.

Our research builds upon the strengths and addresses the limitations of previous work by proposing a unified framework for dynamic parking allocation [8,9]. By leveraging real-time traffic data and occupancy predictions, parking allocations are adjusted dynamically, along with pricing adjustments according to current demand. We employ graph-based models to treat parking zones and intersections as nodes, facilitating efficient route and slot optimization while accounting for congestion and travel costs. Additionally, our framework incorporates user-specific dynamics, such as high-priority access for frequent users or proximity to destinations. Furthermore, adaptive revenue optimization ensures balanced revenue generation while providing value to users, promoting long-term sustainability. Together, these components form a flexible, scalable framework

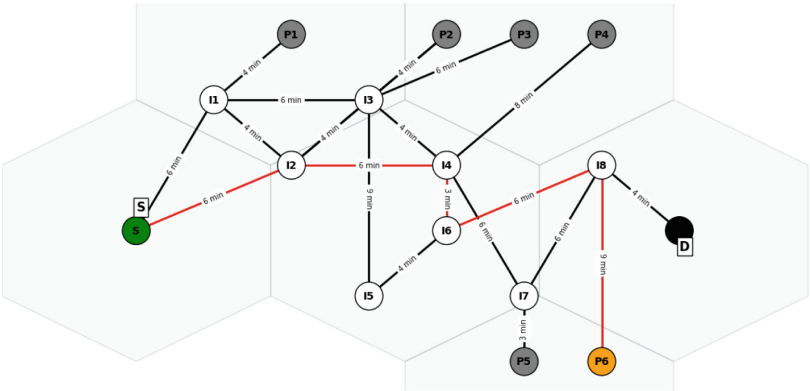


Fig. 1. Graphical representation of the urban road network, where nodes denote intersections, parking lots, the source (S), and the destination (D). Edges represent roads with dynamic travel times. Hexagonal cells enable efficient spatial queries.

capable of solving long-standing problems and responding to the changing needs of modern urban parking systems.

This work's primary innovation lies in its holistic framework, combining real-time traffic analytics, dynamic pricing, and user-centric prioritization [10,11]. By integrating these elements, our system optimizes parking allocation, reduces search times, enhances user satisfaction, and maintains scalability to accommodate diverse urban environments. This unified approach marks a significant advancement in the field of parking allocation systems.

3 System Model and Problem Description

In this section, we present the system model for Dynamic Parking Space Allocation (DPSAS) and Route Planning in an urban environment and formally define the problem addressed. The system aims to guide users from a source location to an available parking slot near their destination while accounting for dynamic traffic conditions and parking availability.

3.1 System Model

The urban environment, as illustrated in Fig. 1, is modeled as an undirected graph $G = (V, E)$, where V is the set of nodes, $V = \{v_1, v_2, \ldots, v_n\}$, representing intersections, parking lots, the source, and the destination, and E is the set of edges, $E = \{e_1, e_2, \ldots, e_m\}$, representing roads connecting the nodes. Each node $v \in V$ has associated attributes. The position of a node is given as $pos(v) = (x_v, y_v)$, representing its coordinates in a 2D plane. Each node is assigned a type, $type(v) \in \{\text{source}, \text{destination}, \text{intersection}, \text{parking_lot}\}$. Additionally, nodes are mapped to hexagonal cells using the function AssignHexCell($pos(v)$),

which assigns the node to a specific hexagonal cell $hex(v) = (q_v, r_v)$. For nodes of type parking lot, additional attributes are defined, including the parking slot ID, $id(v)$, a capacity, $cap(v) \in \mathbb{N}$, representing the maximum number of vehicles the parking lot can accommodate, available spaces, $avail(v) \in \mathbb{N}$, representing the current number of available parking spaces, the parking cost, $cost(v) \in \mathbb{R}^+$, and the availability schedule, $schedule(v) = \{(t_{s_i}, t_{e_i})\}$, which is a set of booked time intervals.

Each edge, $e = (u, v) \in E$, has associated attributes. The distance $dist(e) = d(u, v)$ is calculated using the Euclidean distance between $pos(u)$ and $pos(v)$. The travel time $time(e)$ is dynamically updated based on real-time traffic conditions.

Dynamic Traffic Model: The travel time on each edge is affected by dynamic traffic conditions [12]. At any time t, the travel time $time_t(e)$ is calculated as:

$$time_t(e) = \left(\frac{dist(e)}{v_{\text{adjusted}}(e, t)} \right) \times 60, \tag{1}$$

where $v_{\text{adjusted}}(e, t)$ is the adjusted speed on edge e at time t, computed as:

$$v_{\text{adjusted}}(e, t) = \max(v_{\text{base}} \times tf(e, t), \epsilon), \tag{2}$$

with v_{base} being the base speed (e.g., 60 km/h), $tf(e, t) \in [0.3, 1]$ being a traffic factor simulating congestion, and ϵ being a small constant to avoid division by zero.

Hexagonal Cell Partitioning: The area is partitioned into hexagonal cells to optimize spatial queries [10]. The function AssignHexCell($pos(v)$) maps the position of a node v to a hexagonal cell (q_v, r_v). The set of hexagonal cells within a certain radius R of a node v is denoted as $\mathcal{H}_v(R)$.

3.2 Problem Explanation

The core problem is to find an optimal path from the source node S to an available parking slot p^* near the destination D, considering dynamic traffic conditions and parking slot availability, and to book the parking slot for a desired time duration Δt.

Objectives: The objectives of the problem include minimizing the total travel time, selecting the optimal parking slot, and ensuring dynamic adaptation. The total travel time is minimized by finding a path $P = \{v_0, v_1, \ldots, v_k\}$ from S to p^* that minimizes the total travel time:

$$T_{\text{total}} = \sum_{i=0}^{k-1} time_t(e_i), \tag{3}$$

where $e_i = (v_i, v_{i+1})$. The parking slot p^* should balance proximity to the destination and parking cost. Additionally, the path and parking slot selection must adapt dynamically in response to real-time changes in traffic and parking availability [11].

Constraints: The problem is subject to several constraints. The parking slot p^* must be available for booking during the desired time interval $[t_{\text{start}}, t_{\text{end}}]$, where t_{start} is the estimated arrival time at p^*, and $t_{\text{end}} = t_{\text{start}} + \Delta t$. The path from S to p^* must traverse nodes within the relevant hexagonal cells $\mathcal{H}$ to limit the search space and improve computational efficiency [8,13]. Additionally, the travel times on edges are subject to dynamic updates due to traffic conditions, which may impact the optimality of the path over time.

Problem Definition: Given a graph $G = (V, E)$ with dynamic edge weights $time_t(e)$, a source node S, a destination node D, a desired parking duration Δt, and the current time t_{current}, the objective is to find a parking slot $p^* \in V$ with $type(p^*) = \text{parking_lot}$ that is available for booking during $[t_{\text{start}}, t_{\text{end}}]$. Additionally, a path $P = \{v_0, v_1, \ldots, v_k\}$ must be identified, starting at $v_0 = S$ and ending at $v_k = p^*$, such that T_{total} is minimized. If traffic conditions or parking availability change, the path and p^* must be dynamically updated.

The problem is further constrained by parking slot capacity, which requires:

$$\left|\left\{ (s_i, e_i) \in \text{schedule}(p^*) \mid [t_{\text{start}}, t_{\text{end}}] \cap [s_i, e_i] \neq \emptyset \right\}\right| < \text{cap}(p^*). \tag{4}$$

Nodes in the path P must be within the hexagonal cells $\mathcal{H}$ relevant to S, D, and p^*. Additionally, real-time updates to $time_t(e)$ and $avail(p)$ for $e \in E$ and $p \in V$ must be considered.

Challenges and Complexity: The problem presents several challenges. The dynamic nature of the environment requires the algorithm to adapt in real-time to changes in both traffic conditions and parking availability. The combinatorial nature of the problem, with numerous possible paths and parking slot combinations, adds to its complexity, especially in densely connected urban graphs. Furthermore, the multi-objective optimization aspect, which involves balancing travel time, parking cost, and proximity to the destination, introduces trade-offs that complicate the optimization process. This problem can be classified as a variant of the *Time-Dependent Shortest Path Problem* (TDSPP) with additional constraints, which is known to be NP-hard [13].

3.3 Example Scenario

This example illustrates how the system guides a user from a source location to an available parking lot near their destination.

The urban road network consists of a start point (S), a destination (D), intersections ($I_1, I_2, ..., I_8$), and parking lots ($P_1, P_2, ..., P_6$). Intersections allow

movement in different directions, while parking lots, connected to certain intersections, may change availability dynamically due to new bookings or cancellations. Roads (edges) link these points and have travel times that update in real-time based on traffic conditions, allowing the system to adjust the optimal route as the user progresses. The network is divided into hexagonal cells, enabling efficient spatial queries and faster route calculations [8]. The system continuously recalculates the best route to direct the user through intersections to an available parking lot near D, adapting dynamically to traffic and parking changes.

As shown in Fig. 1, the system computes an initial optimal route from S to a selected parking lot (P_6) near D, highlighted in red. Nodes are organized within hexagonal partitions, ensuring focused spatial processing. Intersections serve as key junctions, while parking lots are color-coded based on availability. Edge travel times are dynamically updated based on current traffic conditions.

Upon reaching intersection I_2, the system reevaluates travel times due to traffic changes, potentially altering the planned path. Similarly, at intersection I_4, fresh traffic data is incorporated, refining the route to P_6. The figure illustrates this adaptive process, demonstrating how real-time updates influence navigation decisions. The use of hexagonal spatial partitioning localizes computations, ensuring an efficient and responsive navigation approach.

3.4 Key Considerations in Solving the Problem

To effectively address the problem, the solution must:

- Dynamically manage parking slot availability and bookings.
- Identify the shortest path from S to an available parking slot p^* within relevant hexagonal cells, considering real-time traffic conditions.
- Adapt to changing conditions by rebooking parking slots and recalculating routes when necessary.
- Reduce computational complexity by constraining the search space using hexagonal cell partitioning.

These considerations guide the development of our approach. In the next section, we propose an algorithm that incorporates these principles to optimize parking and routing decisions in dynamic urban environments.

4 Algorithm Design

To propose the *MainTraversalAndBooking* solution for the Dynamic Parking Space Allocation (DPSAS) problem, we first define three key functions: *ManageParkingSlots*, *GetHexCellsBetweenNodes*, and *FindShortestPathWithHexGrid*. Afterward, we present the main algorithm.

4.1 ManageParkingSlots

The *ManageParkingSlots* algorithm dynamically updates and queries the availability of parking lots to secure a suitable booking time window. Initially, it attempts to identify any available parking slots within a set of hex cells $\mathcal{H}$ near the destination. If no such slots are found, the algorithm broadens its search, considering all available parking lots in the network as a fallback.

Each slot is then prioritized according to its proximity to the destination and its associated cost. These metrics are inserted into a priority queue, ensuring that the most desirable options (i.e., closest and least expensive) are evaluated first. The algorithm then checks whether the slot can accommodate the desired booking interval $[t_{\text{start}}, t_{\text{end}}]$ given its capacity and existing reservations. If it can, the interval is added to the slot's schedule, and this slot is immediately returned as the chosen option. If no candidate slot can fulfill the requirements, the algorithm returns None, indicating that a suitable parking opportunity was not found. The time complexity of this process is $O(|P| + |P_{\text{available}}| \log |P_{\text{available}}|)$, where $|P|$ is the total number of parking slots and $|P_{\text{available}}|$ is the subset currently available for consideration. This reflects the complexity of first identifying suitable candidates and then efficiently extracting and validating them via the priority queue.

4.2 GetHexCellsBetweenNodes

The *GetHexCellsBetweenNodes* function identifies all hexagonal cells that lie between two nodes in the graph. It retrieves the hex cell assignments for the start node S and the end node D, calculates the minimum and maximum indices along the hexagonal grid axes (q and r), and slightly expands this range to account for potential deviations.

4.3 FindShortestPathWithHexGrid

The *FindShortestPathWithHexGrid* algorithm, illustrated in Algorithm 1, computes the shortest path between two nodes u and v, considering only nodes within specified hex cells $\mathcal{H}$. This modification of Dijkstra's algorithm enhances efficiency by reducing the search space to relevant areas [10]. The algorithm initializes a priority queue Q and maintains distance estimates $d[n]$ and predecessor links $\pi[n]$. Select the node with the smallest tentative distance and explore its neighbors, considering only those within $\mathcal{H}$. Once the destination node v is reached, the algorithm reconstructs the shortest path using the predecessor links. The total travel time T is obtained from the final distance estimate $d[v]$.

The time complexity is $O(|E'| + |V'| \log |V'|)$, where $|V'|$ and $|E'|$ are the number of nodes and edges within the hex cells $\mathcal{H}$. Since $|V'| \ll |V|$ and $|E'| \ll |E|$ as the search is restricted to nodes within a specific subset of hexagonal cells $\mathcal{H}$, rather than considering the entire graph. These hex cells are chosen based on proximity to the source and destination, significantly reducing the number of nodes evaluated. This algorithm is more efficient than standard Dijkstra's algorithm on the full graph.

Algorithm 1. FindShortestPathWithHexGrid

Require: Graph G, Start node u, End node v, Hex cells $\mathcal{H}$
Ensure: Shortest path P, Total travel time T

Initialize priority queue $Q \leftarrow [(0, u)]$
Set distances $d[n] \leftarrow \infty$ for all $n \in G$
Set predecessors $\pi[n] \leftarrow$ None for all $n \in G$
Set $d[u] \leftarrow 0$
while Q is not empty **do**
  Extract node with smallest distance: $(d_{\text{current}}, n_{\text{current}}) \leftarrow Q$.pop
  if $n_{\text{current}} = v$ **then**
    break
  end if
  for each neighbor m of n_{current} **do**
    if m.hex_cell $\notin \mathcal{H}$ **then**
      continue
    end if
    Calculate tentative distance: $d_{\text{tentative}} \leftarrow d_{\text{current}} + G[n_{\text{current}}][m]$.time
    if $d_{\text{tentative}} < d[m]$ **then**
      Update distance: $d[m] \leftarrow d_{\text{tentative}}$
      Update predecessor: $\pi[m] \leftarrow n_{\text{current}}$
      Add $(d[m], m)$ to Q
    end if
  end for
end while
Reconstruct path P using predecessors π
return $P, d[v]$

4.4 MainTraversalAndBooking

The *MainTraversalAndBooking* algorithm, detailed in Algorithm 2, orchestrates the process of navigating from the source node S to a parking slot near the destination D, while dynamically managing traffic and parking conditions. It begins by attempting to book a parking slot p^* close to the destination within the desired time frame. The algorithm divides the road network into hexagonal cells and identifies relevant hex cells near the destination for the search and management of parking.

The algorithm enters a loop where it continually updates traffic conditions and parking availability to reflect real-time changes. If the previously booked parking slot p^* becomes unavailable, it attempts to rebook another slot. The parking slot selection is stabilized once the user enters a hex cell adjacent to the destination D's hex cell [9] because it provides a seamless experience by locking in the parking choice when the user is close to their destination, avoiding unnecessary detours or last-minute changes that could cause inconvenience.

At this point, the algorithm finalizes the selected parking slot, stops further availability updates, and prevents rebooking to ensure a seamless experience as the user approaches their final destination.

Algorithm 2. MainTraversalAndBooking

Require: Graph G, Source node S, Destination node D, Desired parking duration Δt
Ensure: Final path P_{final}, Total travel time T_{total}

Initialize $u \leftarrow S$, $T_{\text{total}} \leftarrow 0$, $P_{\text{final}} \leftarrow [u]$
$\mathcal{H} \leftarrow \text{GetHexCellsNearDestination}(D)$
$p^* \leftarrow \text{ManageParkingSlots}(G, t_{\text{current}}, [t_{\text{current}}, t_{\text{current}} + \Delta t], \mathcal{H})$
if $p^* =$ **None then**
  return Failure
end if
$\mathcal{H} \leftarrow \mathcal{H} \cup \text{GetHexCellsBetweenNodes}(S, p^*)$
$\mathcal{H}_{\text{adjacent}} \leftarrow \text{GetAdjacentHexCells}(D)$, $adjacentZoneReached \leftarrow$ **False**
while $u \neq p^*$ **do**
  Update: Traffic, t_{current}, and parking availability
  if p^*.availability = False **and not** $adjacentZoneReached$ **then**
    $p^* \leftarrow \text{ManageParkingSlots}(G, t_{\text{current}}, [t_{\text{current}}, t_{\text{current}} + \Delta t], \text{None})$
    if $p^* =$ **None then**
      return Failure
    end if
    $\mathcal{H} \leftarrow \mathcal{H} \cup \text{GetHexCellsBetweenNodes}(u, p^*)$
  end if
  $(P_{\text{temp}}, T_{\text{temp}}) \leftarrow$ FindShortestPathWithHexGrid$(G, u, p^*, \mathcal{H})$
  Move to next node: $u \leftarrow$ next node in P_{temp}, $P_{\text{final}} \leftarrow P_{\text{final}} + [u]$, $T_{\text{total}} \leftarrow T_{\text{total}} + T_{\text{temp}}$
  if $u \in \mathcal{H}_{\text{adjacent}}$ **and not** $adjacentZoneReached$ **then**
    $adjacentZoneReached \leftarrow$ **True**; Stop updates and finalize p^*
  end if
end while
return $P_{\text{final}}, T_{\text{total}}$

At each iteration, the algorithm computes the shortest path from the current node u to the parking slot p^* using the *FindShortestPathWithHexGrid* algorithm. It then moves to the next node along this path, updating the final path P_{final} and the total travel time T_{total}.

The algorithm ensures correctness by dynamically adapting to changing traffic and parking conditions while locking the parking selection when the user approaches the destination. Assuming the number of iterations is k, the overall time complexity is approximately $O(k \cdot |V'| \log |V'|)$, since the dominant operations involve the computation of the shortest path within the hex cells. The use of hex grids and the adjacent hex cell stabilization mechanism helps to limit the size of $|V'|$ and $|E'|$ considered in each iteration, keeping the algorithm efficient for real-time applications.

Although this complexity may seem high, in practice, the number of iterations k is bounded by the distance to the parking slot and the frequency of traffic updates. By dynamically updating only a subset of the graph and focusing computations within relevant hex cells, the algorithm maintains scalability even in large urban networks.

4.5 Correctness Proof

We will prove that the *MainTraversalAndBooking* algorithm correctly finds a feasible path from the source to an available parking slot near the destination, while dynamically adapting to traffic and parking availability updates.

Initialization: The algorithm begins by initializing the graph and trying to book a parking space near the destination. If a parking slot is successfully booked, it proceeds to find a path to that slot.

Dynamic Updates: During traversal, the algorithm updates traffic conditions and parking availability at each time step. If the parking slot becomes unavailable, it attempts to rebook another slot and recalculates the path.

Termination: The algorithm terminates when the user reaches the parking slot. Since the algorithm always attempts to find an available parking slot and computes the shortest path to it, it ensures that the user can reach a parking slot if one is available.

Correctness of Sub-algorithms

- *ManageParkingSlots*: Correctly maintains parking slot availability and selects an optimal parking slot based on proximity and cost.
- *FindShortestPathWithHexGrid*: Correctly computes the shortest path within the specified hex grid cells using a modified Dijkstra's algorithm.

By ensuring that each component of the algorithm functions correctly and handles dynamic updates appropriately [4], the *MainTraversalAndBooking* algorithm achieves its intended goal of guiding the user to an available parking slot near the destination while adapting to real-time conditions.

5 Performance Evaluation

For evaluation, we generated a geometric structure similar to [11] and integrated a dynamic, stochastic traffic model. A traffic factor $\text{tf} \in [0.3, 1]$ is sampled to modify the base speed v_{base}, yielding an edge-specific adjusted speed v_{uv}. To maintain realism, v_{uv} never approaches zero, ensuring that traffic may slow but rarely comes to a complete stop. The adjusted speed, combined with the computed distance, determines the travel time t_{uv} for each edge, which is then stored as a graph attribute.

We tracked key performance metrics to evaluate our system's effectiveness, focusing on execution time, travel time to destinations, scalability, and robustness. Execution time measures responsiveness, crucial for real-time applications, while travel time indicates routing efficiency. Scalability assesses the system's ability to handle increasing complexity, and robustness reflects its adaptability to dynamic conditions. Together, these metrics ensure the system meets its objectives of minimizing travel time and maintaining efficiency.

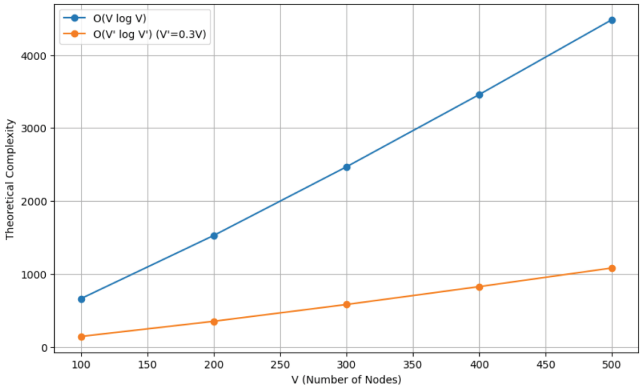


Fig. 2. Theoretical Complexity: Normal vs. Hex-grid Approach

To thoroughly test the system's performance, we vary several parameters. The number of users, or simultaneous queries, is increased from 10 to 50 to test how well the system handles multiple concurrent route requests. This parameter directly influences execution time and tests scalability under high demand. The traffic factor, ranging from 0.3 to 1.0, simulates congestion levels. A low traffic factor, around 0.3, represents heavy congestion, while a factor close to 1.0 indicates near-free-flow conditions. Varying the traffic factor tests the algorithm's ability to adapt to changing road speeds and conditions. The number of nodes, which ranges from 100 to 500, scales the size of the graph to test how well the approach handles increasing problem complexity. By comparing various algorithms at different network sizes, we assess which methods are more scalable. Furthermore, algorithmic approaches are compared by evaluating the proposed Hex grid Dijkstra approach against the Normal Dijkstra and the heuristic-based A* method. Including A* provides a benchmark from the literature, helping us identify where our spatial filtering method stands relative to established shortest path algorithms [9,11,12]. We evaluated the proposed solution against a variety of parameters to showcase its behavior and performance.

Figure 2 illustrates the theoretical complexity $O(|V|\log|V|)$ of Normal Dijkstra and $O(|V'|\log|V'|)$ for the hex-grid approach, where $|V'| \ll |V|$. The complexity curve for the hex-grid approach consistently remains lower, as it effectively reduces the problem size by only considering relevant hexagonal cells. This theoretical insight aligns quite well with the empirical results, confirming that spatial filtering diminishes computational overhead and improves scalability.

Figure 3 shows execution time scaling for Normal Dijkstra, Hex-grid Dijkstra, and A* as the graph size increases from 100 to 500 nodes. Normal Dijkstra experiences rapid growth, reaching approximately 0.0175 s at 500 nodes. Hex-grid Dijkstra, however, remains under 0.003 s at the same scale, significantly narrowing the gap. A* outperforms both, barely exceeding 0.001 s at 500 nodes. Although A* is known for its heuristic efficiency, the hex-grid approach provides a substantial advantage over plain Dijkstra, moving closer to A* performance.

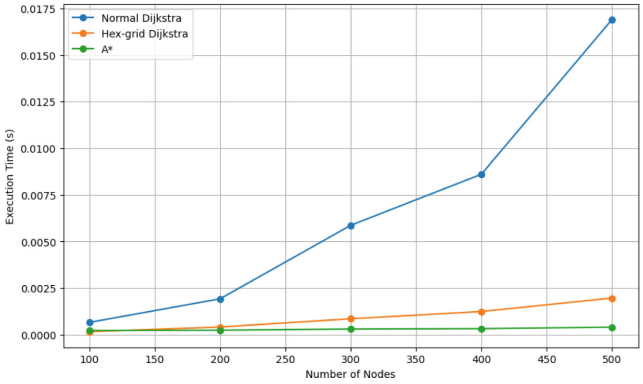


Fig. 3. Execution Time vs Number of Nodes

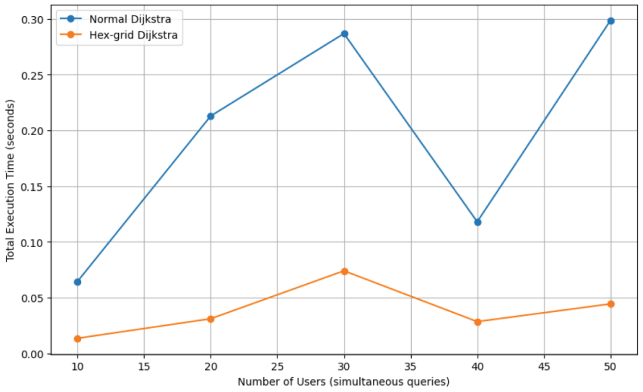


Fig. 4. Number of Users vs. Execution Time

Figure 4 compares the execution times of Normal Dijkstra and Hex-grid Dijkstra as the number of users increases from 10 to 50. Normal Dijkstra's execution time rises substantially, peaking around 0.30 s at heavy loads. In contrast, Hex grid Dijkstra remains under 0.10 s, demonstrating improved scalability and quicker response under higher user loads. The spatial filtering reduces the search space, resulting in more stable and efficient computations.

In Fig. 5, the allocation success rate decreases as the number of users increases from 10 to 50. Hex-grid Dijkstra maintains a success rate above 88% under high user loads, closely following A*'s performance. Normal Dijkstra shows a sharper decline, dropping to 82%. These results demonstrate Hex-grid Dijkstra's robust handling of multiple users, outperforming Normal Dijkstra while staying competitive with A*. The use of spatial filtering in Hex-grid Dijkstra ensures better resource utilization, reducing computational overhead as user demand increases. This highlights its potential to scale effectively in dense urban scenarios.

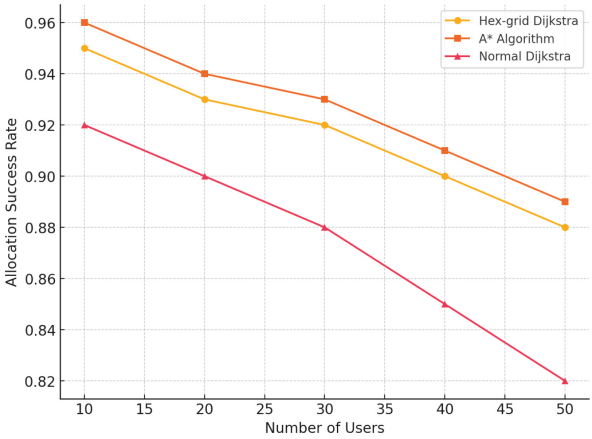


Fig. 5. Allocation Success Rate vs. Number of Users

In Fig. 6, travel time decreases as the traffic factor increases from 0.3 (heavy congestion) to 1.0 (free-flow). Hex-grid Dijkstra demonstrates consistent adaptability, reducing travel times from over 16 min to below 6 min. A* performs slightly better, while Normal Dijkstra exhibits the longest travel times across all traffic conditions. The results highlight Hex-grid Dijkstra's ability to dynamically adjust to traffic variations, approaching A* performance.

5.1 Comparisons with Other Techniques

A* serves as a well-regarded benchmark in shortest path literature [12]. While the hex-grid approach does not surpass A*, it significantly outperforms Normal Dijkstra and edges closer to A*'s execution times by restricting the search space. This result confirms that spatial partitioning can partially emulate the benefits of heuristic-driven search.

A naïve approach that ignores traffic dynamics or simply selects the closest parking lot by Euclidean distance would fail to adapt to real-world complexity [2], resulting in longer and less reliable travel times. By contrast, our dynamic approach, continuously updating edge weights and re-evaluating parking availability, delivers more accurate route and parking recommendations.

While no direct comparisons are shown, genetic algorithms or ML-based approaches often require extensive training or tuning. Our method yields consistent and predictable results with straightforward graph-based techniques.

The implementation handles dynamic changes in parking availability, allowing re-booking when a previously chosen parking slot becomes unavailable. Despite these fluctuations, the hex-grid approach and the A* method maintain low execution times and stable travel times. In practice, this robustness ensures that users receive timely updates and reliable guidance, even in evolving and unpredictable traffic and parking scenarios.

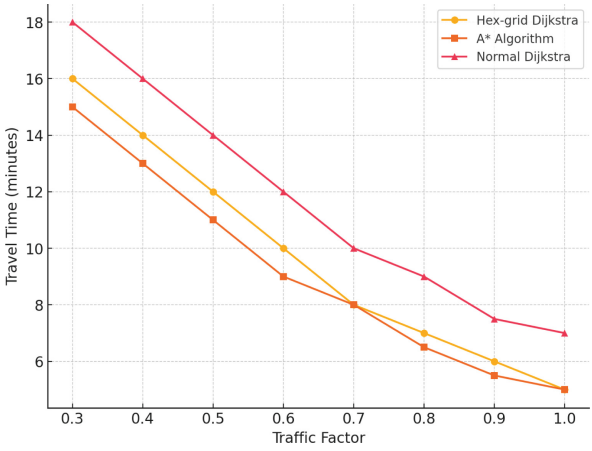


Fig. 6. Travel Time vs Traffic Factor

Overall, the performance evaluation confirms that the proposed hex-grid Dijkstra approach provides a substantial improvement over Normal Dijkstra, especially under heavy user load and large network sizes. It narrows the performance gap with A*, demonstrating that spatial constraints can serve as a powerful means of optimization. The combination of dynamic hex-grid partitioning and continuous adaptation to traffic conditions yields a practical, efficient, and robust solution for dynamic parking space allocation systems.

We summarize the key observations. First, it offers improved scalability as hex-grid Dijkstra remains efficient even as the number of users and nodes increases. This scalability enables the system to significantly outperform Normal Dijkstra in handling large-scale urban networks. Additionally, the system exhibits adaptability to traffic conditions, with travel times predictably decreasing as traffic improves. This behavior indicates the correct and responsive updating of route calculations, ensuring the system remains effective under dynamic conditions.

Furthermore, the comparative efficiency of the approach is evident. While hex-grid filtering may not match the speed of A*, it effectively reduces complexity and execution time. This brings its performance closer to heuristic methods, such as A*, without relying on custom heuristics and hence enhancing its generalizability. Finally, the system's robustness is underscored by its ability to handle dynamic parking availability and changing traffic conditions without significant performance loss. This resilience highlights its practicality and applicability in real-world scenarios.

The combination of dynamic hex-grid partitioning and continuous adaptation to traffic conditions yields a practical, efficient, and robust solution for dynamic parking space allocation systems.

6 Conclusion

The Dynamic Parking Space Allocation System (DPSAS) presents a novel, user-centric approach to urban parking management by integrating real-time data with hexagonal grid-based spatial partitioning. This design enhances computational efficiency while ensuring responsive and adaptive decision-making. At its core, DPSAS leverages a modified Dijkstra's algorithm to dynamically recommend optimal parking slots, continuously adjusting to traffic conditions and availability changes. The hexagonal partitioning framework not only improves route computation efficiency but also ensures temporal scalability by distributing the computational load effectively. As a result, users experience more efficient travel with minimal delays caused by congestion and parking uncertainties. Comparative analysis shows that DPSAS outperforms traditional and heuristic-based methods by bridging the gap between computational simplicity and state-of-the-art performance. By seamlessly integrating real-time analytics, adaptive optimization, and scalable processing, DPSAS offers a practical and intelligent solution for smart cities. Its contributions establish a transformative model for addressing the evolving challenges of urban mobility, paving the way for future advancements in intelligent transportation systems.

References

1. Mladenović, M.N., Nourinejad, M., Foreman, K., Kouvelas, C.: A framework for dynamic reallocation of parking spaces in parking facilities. Transport. Res. Part C: Emerg. Technol. **123**, 102990 (2021)
2. Elfaki, A.O., Messoudi, W., Bushnag, A., Abuzneid, S., Alhmiedat, T.: Constraint optimization model for dynamic parking space allocation. Sensors **24**, 3988 (2024)
3. Wang, T., Chen, Y., Yu, J., Xie, S.: Formation mechanisms and clustering differences in risky riding behaviors of electric bike riders. IEEE Access **9**, 119 712–119 721 (2021)
4. Wang, T., Chen, Y., Yan, X., Li, W., Shi, D.: Traffic risk assessment based on warning data. J. Adv. Transp. **2022**, 1 (2022)
5. Liu, Y., Wang, C., Xu, S., Zhou, W., Chen, Y.: Multi-weighted graph 3D convolution network for traffic prediction. Neural Comput. Appl. **35**(20), 15 221–15 237 (2023)
6. Shi, D., Wang, T., Chen, Y., Liu, C.: Exploring the effects of request time, secondary task, and take-over mode on take-over performance. In: International Conference on Intelligent Transportation Engineering. Springer Nature Singapore, pp. 1001–1014 (2021)
7. Saharan, S., Kumar, N., Bawa, S.: An efficient smart parking system based on deep learning and IoT technologies. IEEE Trans. Intell. Transp. Syst. **24**(2), 1–12 (2023)
8. Chen, Y., Wang, T., Yan, X., Wang, C.: An ensemble optimization strategy for dynamic parking-space allocation. IEEE Intell. Transp. Syst. Mag. **15**(1), 347–362 (2023)
9. An, Z., Rui, X., Gao, C.: Improved a* navigation path-planning algorithm based on hexagonal grid. ISPRS Int. J. Geo Inf. **13**(5), 166 (2024)

10. Sahr, K.: Hexagonal discrete global grid systems for geospatial computing, ResearchGate (2011)
11. Li, D., Shi, X., Dai, M.: An improved path planning algorithm based on a* algorithm. In: Proceedings of the 13th International Conference on Computer Engineering and Networks, pp. 187–196 (2024)
12. Ou, Y., Fan, Y., Zhang, X., Lin, Y., Yang, W.: Improved a* path planning method based on the grid map. Sensors **22**(16), 6198 (2022)
13. Wang, Y., Yu, X., Liang, X.: Design and implementation of global path planning system for unmanned surface vehicle among multiple task points, arXiv preprint arXiv:1807.08106 (2018)