Faculty of Computing

Semester I 2025/2026

SECJ3553 Artificial Intelligence

Section 07

Progress 2: State Space Search

Project Theme:

Smart City - Traffic & Parking Advisor Application (Parkora.ai)

Team: TriSpark Tech

TEAM MEMBERS	MATRIC NO.
AUSTIN SEE YONG HUI	A23CS5015
WONG JIA XUAN	A23CS0197
YAP EN THONG	A23CS0284

Lecturer: Dr. Ruhaidah binti Samsudin

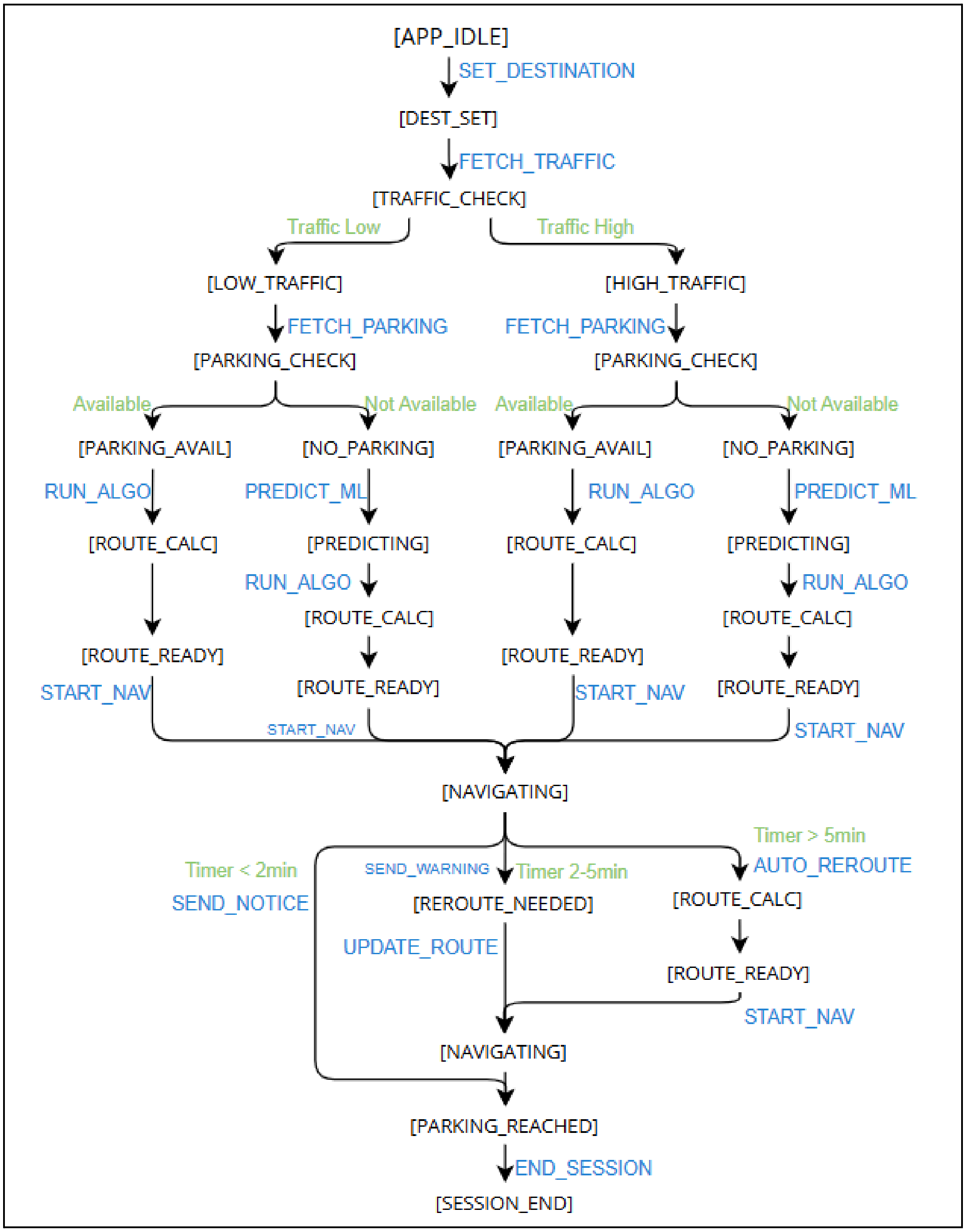
Submission Date: 6^{TH} DECEMBER 2025

Table of Contents

1.0 Introduction

Parkora.ai tackles urban parking inefficiencies by implementing a state space search model that transforms real-time traffic and parking data into an actionable AI decision-making framework, guiding drivers optimally from start to parked destination.

2.0 State Space Graph



3.0 State Space Definition

3.1 States

The state space consists of 15 distinct states representing the system's condition at any moment:

State ID	State Name	Description
S0	APP_IDLE	Application open, no destination set. Initial state.
S1	DEST_SET	Destination entered by the user
S2	TRAFFIC_CHECK	System is fetching live traffic data. System analyses: - Real-time congestion - Road conditions - Travel time estimates
S3	LOW_TRAFFIC	Traffic congestion below threshold. Clear paths. Normal travel.
S4	HIGH_TRAFFIC	Traffic congestion above threshold. Congested. Delays expected.
S5	PARKING_CHECK	System checking parking availability. Query sensors. Check the database.
S6	PARKING_AVAIL	Available parking spots detected.
S7	NO_PARKING	No parking spot is available around the destination.
S8	PREDICTING	ML model predicting future availability.
S9	ROUTE_CALC	Running Dijkstra/A* algorithm.
S10	ROUTE_READY	Optimal route computed.
S11	NAVIGATING	User following guided route. Timer starts.

S12	REROUTE_NEEDED	Traffic/parking change detected.
S13	PARKING_REACHED	User successfully parked. Goal State.
S14	SESSION_END	Journey completed

3.2 Actions

Action ID	Action Name	Description	Trigger Condition
A1	SET_DESTINATION	User inputs destination	From S0
A2	FETCH_TRAFFIC	Call Google Maps API	From S1
A3	FETCH_PARKING	Query IoT parking sensors	From S3/S4
A4	RUN_ALGO	Execute Dijkstra/A*	From S6/S8
A5	PREDICT_ML	Run ML prediction model	From S7
A6	START_NAV	Begin turn-by-turn guidance	From S10
A7	SEND_NOTICE	Send mild notification. “On track”. “Good time”.	Timer < 2 mins
A8	SEND_WARNING	Send urgent alert. “Consider alternatives”. “Parking may be limited”.	Timer 2-5 mins
A9	AUTO_REROUTE	System finds new route automatically	Timer > 5 mins
A10	UPDATE_ROUTE	Manual route update	From warning state
A11	END_SESSION	Complete journey	From S13

4.0 Detailed State Space Search Write-Up

4.1 State Space Structure

The state space operates as a directed graph where each state represents a specific configuration of the system, and transitions between states occur through well-defined actions.

4.2 Search Methodology

The system performs a real-time search through this state space, aiming to find the optimal path from the initial state to the goal state. The search considers:

1. Real-time constraints (traffic conditions, parking availability)
2. Temporal factors (timer-based service activation)
3. Cost optimisation (minimising time, distance, fuel consumption)

4.3 Decision-Making Process

The search process makes decisions at key branching points:

1. Traffic assessment (S2 → S3/S4)
2. Parking availability (S5 → S6/S7)
3. Timer-based interventions (S11 → various alert states)
4. Rerouting decisions (back to S9 for recalculation)

5.0 Problem Formulation Components

5.1 Initial State

S0: APP_IDLE - The application is open but inactive, awaiting user input.

5.2 Actions

The complete action set: A = {A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11}

5.3 Goal State

S11: PARKING_REACHED - User has successfully found and reached a parking spot.

5.4 Path Cost

The cost function is multi-dimensional:

$C(path) = w_1 \times Travel\_Time + w_2 \times Distance + w_3 \times Fuel\_Cost + w_4 \times Congestion\_Score$

Where:

- $w_1 = 0.4$ (Time weight - most important)
- $w_2 = 0.3$ (Distance weight)
- $w_3 = 0.2$ (Fuel cost weight)
- $w_4 = 0.1$ (Congestion/Stress weight)
- All weights sum to 1.0

6.0 Solution: Sequence Of Actions

Scenario 1: Optimal Path (Low traffic + Available parking)

The best-case scenario is when there is low traffic and available parking.

Action sequence: A1(SET_DESTINATION) → A2(FETCH_TRAFFIC) → A3(FETCH_PARKING) → A4(RUN_ALGO) → A6(START_NAV) → A7(SEND_NOTICE) → A11(END_SESSION)

State Path: S0 → S1 → S2 → S3 → S5 → S6 → S9 → S10 → S11 → S13 → S14

Scenario 2: High traffic + No parking (Predict & Reroute)

Use when FETCH_PARKING returns NO_PARKING at high traffic and no immediate parking situation.

Action sequence: A1(SET_DESTINATION) → A2(FETCH_TRAFFIC) → A3(FETCH_PARKING) → A5(PREDICT_ML) → A4(RUN_ALGO) → A6(START_NAV) → A8(SEND_WARNING) → A10(UPDATE_ROUTE) → A11(END_SESSION)

State Path : S0 → S1 → S2 → S4 → S5 → S7 → S8 → S9 → S10 → S11 [Timer 3min] → S12 → S13 → S14

Scenario 3: Mid-route Reroute (Parking Becomes Unavailable)

Use when navigating to a chosen route, the chosen parking becomes unavailable.

Action sequence: A1(SET_DESTINATION) → A2(FETCH_TRAFFIC) → A3(FETCH_PARKING) → A4(RUN_ALGO) → A6(START_NAV) → A8(SEND_WARNING) → A8(UPDATE_ROUTE) → A11(END_SESSION)

State Path: S0 → S1 → S2 → S3 → S5 → S6 → S9 → S10 → S11 [Parking lost] → S12 → S11 → S13 → S14

Scenario 4: Severe Delay with Auto-reroute

When a parking slot is lost and the timer exceeds the critical threshold, the system will automatically reroute.

Action sequence: A1(SET_DESTINATION) → A2(FETCH_TRAFFIC) → A3(FETCH_PARKING) → A5(PREDICT_ML) → A4(RUN_ALGO) → A6(START_NAV) → A9(AUTO_REROUTE) → A6(START_NAV) → A11(END_SESSION)

State Path: S0 → S1 → S2 → S4 → S5 → S7 → S8 → S9 → S10 → S11 [Timer >5min] → S9 → S10 → S11 → S13 → S14

7.0 How the Problem Formulation Supports the Proposed Knowledge Representation

The problem formulation directly supports the proposed KR by mapping real-world conditions to formal states and actions. States explicitly represent traffic levels and parking availability, while actions model system operations and user interactions. The graph structure clarifies decision flow, and timer-based transitions implement urgency logic.

Search algorithms integrate seamlessly: Action A4 directly implements Dijkstra/A*, State S9 represents algorithm completion, the path cost function guides optimisation, and graph search finds optimal sequences from start to goal.

Predictive ML is embedded through Action A5, which triggers model execution, and State S8, which represents ML processing. ML outcomes intelligently determine transitions between waiting and rerouting for NO_PARKING scenarios.

Smart notifications are formalised as Actions A7–A9, creating a tiered alert system triggered by timer thresholds (2 min, 5 min). Notifications are treated as first-class actions, enabling context-aware, timely communication with the user throughout the navigation process.

8.0 Conclusion

In summary, this state space search formulation successfully transforms Parkora.ai's real-world parking problem into a structured AI search framework, enabling systematic scenario exploration, optimal pathfinding via search algorithms, intelligent ML predictions, timely risk-based notifications and measurable performance through comprehensive cost functions—creating an implementable, robust system for smart urban parking guidance.