Adaptive Heuristic Learning for Stress-Optimized Pedestrian Navigation

An AI-Driven Approach Using Path Sampling and Regularization

Gil Raitses

2025-10-27

Abstract

Urban pedestrians navigate complex environments where traditional shortest-path algorithms fail to account for psychological stress factors including sidewalk cycling violations, pedestrian density, and infrastructure quality that significantly impact route satisfaction and safety. While conventional navigation systems optimize geometric distance using static **heuristic functions** like **Euclidean distance** or **Manhattan distance**, these approaches cannot adapt to changing conditions or learn from user preferences across large-scale networks such as New York City's 907 camera zones spanning 3626 square kilometers. This research addresses the question: how can we design **adaptive heuristic functions** that learn to optimize multiple conflicting objectives including **path length**, **stress reduction**, and **computational efficiency** while maintaining real-time performance for practical urban routing applications? We develop an **adaptive heuristic learning framework** integrating **Monte Carlo path sampling** to explore diverse **Pareto-optimal routes**, **machine learning weight optimization** combining geometric, stress-based, and safety-based **base heuristics**, and **dynamic regularization** inspired by **weighted A*** that adjusts search intensity based on query complexity. Using 5 real-world camera zone analyses with 17-dimensional feature vectors from **Google Cloud Vision API** augmented to 455 training samples through **bootstrap resampling**, **SMOTE**, and **parametric generation**, we train **Ridge regression** models achieving less than 15 percent **mean absolute error** on validation data. Comparative evaluation against 6 baseline algorithms including **Dijkstra**, **standard A***, and **weighted A*** demonstrates 30 to 60 percent reduction in **nodes expanded** while preserving 90 percent solution quality measured against optimal paths, with **response times** under 3 seconds for 95 percent of queries on the full 907-node network. This work establishes that learned heuristics combining domain knowledge initialization with data-driven weight adaptation can substantially improve routing quality for stress-sensitive applications, providing a replicable methodology for incorporating human factors into classical search algorithms with implications for accessible urban planning, emergency response optimization, and multi-objective navigation across transportation networks.

Table of contents

Introduction

Subject Area within Artificial Intelligence

This project operates within the **informed search** and **heuristic learning** domain of artificial intelligence, specifically addressing the intersection of **heuristic search algorithms** including **A***, **weighted A***, and **adaptive variants**, alongside **machine learning for heuristic optimization** where we learn both **admissible** and **inadmissible heuristic functions** from empirical data. The work encompasses **multi-objective optimization** by balancing competing objectives such as **path length**, **computational efficiency**, and **route quality**, while incorporating principles from **reinforcement learning** through **adaptive weight adjustment** based on search performance feedback. This research directly builds upon fundamental concepts taught in CIS 667, particularly Chapter 3 covering Solving Problems by Searching and Chapter 4 addressing Search in Complex Environments from Russell and Norvig's foundational text Artificial Intelligence: A Modern Approach, extending these classical algorithms with modern machine learning techniques to create practical systems for real-world urban navigation.

Problem Statement and Research Motivation

Traditional pedestrian navigation systems optimize exclusively for the shortest path or fastest route under the implicit but flawed assumption that all streets are functionally equivalent when normalized for distance and travel time. This oversimplification ignores the substantial variation in psychological stress, safety conditions, and comfort levels that urban pedestrians experience across different routes due to factors including safety violations such as sidewalk cycling and aggressive traffic behavior, pedestrian density creating overcrowding and bottlenecks that impede natural flow, infrastructure quality encompassing missing barriers and inadequate lighting, and environmental conditions including weather impact and visibility that vary significantly throughout the day. These stress factors create compelling need for stress-optimized routing that balances traditional efficiency metrics against human well-being factors such as mental stress, physical safety, and overall comfort during the journey. The core challenge addressed by this research asks: how can we design adaptive heuristic functions that learn to optimize for multiple conflicting objectives while maintaining computational tractability necessary for real-time urban navigation across large-scale networks containing hundreds or thousands of nodes?

Existing approaches to urban pedestrian routing face three critical limitations that motivate this research. Static heuristics employed in traditional A* implementations use fixed, domain-independent functions such as Manhattan distance or Euclidean distance that cannot adapt to changing urban conditions including construction, special events, or weather. These approaches ignore user-specific preferences for route quality versus speed, treat all graph edges as equivalent except for geometric distance, and provide no mechanism for learning from accumulated experience or user feedback. Current navigation systems demonstrate a complete lack of stress modeling, providing no quantitative models for pedestrian stress factors, no integration of real-time safety violation data from camera networks or crowdsourced reports, and no multi-objective optimization frameworks that consider psychological well-being alongside traditional efficiency metrics. Multi-objective optimization in large urban graphs presents severe computational challenges where cities like New York contain 907 camera zones forming large-scale graphs. Real-time route queries require response times under 3 seconds to remain practical for mobile applications, yet exhaustive search becomes infeasible due to exponential state space growth, and the trade-offs between solution quality and search efficiency remain poorly characterized in existing literature.

Aim of the Project

This research aims to develop and empirically validate an **adaptive heuristic learning framework** for **stress-optimized pedestrian navigation** that integrates 3 complementary technical innovations. The framework employs **path sampling techniques** to systematically explore the solution space of multi-objective urban routes, enabling discovery of diverse **Pareto-optimal paths** that represent different trade-offs between competing objectives such as minimizing distance, reducing stress exposure, and avoiding safety violations. We implement **heuristic learning algorithms** that adaptively combine multiple **base heuristics** including **geometric heuristics** based on Euclidean distance, **stress-based heuristics** derived from camera zone safety scores, and **safety-based heuristics** incorporating violation density, through data-driven weight learning using **Ridge regression** with **L2 regularization** and potential online adaptation through **gradient descent** updates based on user feedback signals. The system incorporates **adaptive regularization mechanisms** inspired by **weighted A*** search that dynamically adjust search intensity based on query complexity measured by estimated path length, computational constraints including maximum node expansion budgets, and solution quality requirements specified by minimum acceptable route quality thresholds.

The framework will be implemented as a standalone Python system with comprehensive evaluation using real NYC urban navigation data, demonstrating practical applicability to large-scale routing problems while maintaining theoretical rigor in algorithm design and empirical analysis. The implementation focuses entirely on local computation using standard scientific Python libraries including NumPy for numerical operations, pandas for data manipulation, scikit-learn for machine learning models, and NetworkX for graph algorithms, explicitly avoiding cloud-based or serverless architectures to ensure reproducibility and facilitate deployment across diverse computational environments. This approach enables researchers and practitioners to replicate results, extend the methodology to new domains, and deploy the system in resource-constrained settings without dependence on external API services or commercial cloud platforms.

Objectives

This project defines eight comprehensive objectives with precise success criteria that together constitute a complete investigation demonstrating mastery of AI search algorithms, machine learning integration, and rigorous empirical evaluation methodology. The objectives span the full research lifecycle from algorithm design through theoretical validation, organized into three major research themes: algorithm development (Objectives 1-3) establishing the core adaptive heuristic learning framework, empirical evaluation (Objectives 4-6) validating performance on real NYC data, and theoretical foundations (Objectives 7-8) proving algorithm properties and scalability characteristics. Each objective addresses a distinct technical challenge while contributing to the overarching goal of creating practical stress-optimized routing systems that balance path length, computational efficiency, and human well-being factors.

Path Sampling Framework Development

The first objective develops a Monte Carlo-based path sampling system that generates diverse candidate routes exploring different regions of the solution space rather than converging prematurely to locally optimal paths. We implement three distinct sampling strategies: uniform random walk with goal-directed bias using temperature parameter τ ranging from 0.1 to 1.0 to control exploration intensity, importance-weighted sampling where sample probability remains proportional to heuristic promise allowing the system to focus computational effort on promising regions, and Thompson sampling using a Bayesian approach that naturally balances exploration of uncertain routes against exploitation of known good paths. The framework incorporates diversity mechanisms through controlled randomness in action selection, path overlap penalties computed using Jaccard distance to avoid redundant routes sharing excessive common segments, and Pareto frontier identification for multi-objective trade-offs that retains only non-dominated solutions where no other path is strictly better across all objectives simultaneously.

Success for this objective requires achieving sample diversity exceeding 70 percent unique paths measured by Jaccard distance between generated routes, Pareto coverage exceeding 80 percent of the true Pareto

front validated by comparison to exhaustive search on small graphs containing 10 nodes, computation time remaining under 1 second for generating 50 samples to maintain real-time responsiveness, and quality distribution spanning at least 3 distinct quality tiers identified through clustering analysis of path scores. We validate the sampling approach by comparing sampled paths to exhaustive enumeration on 10-node subgraphs extracted from the full NYC network, measuring coverage of Pareto-optimal solutions and analyzing the distribution of path qualities to ensure we capture both high-quality routes and diverse alternatives.

Adaptive Heuristic Learning System

The second objective designs and trains a machine learning system that learns to combine multiple base heuristics adaptively based on historical routing data and simulated user preferences. We implement three complementary base heuristic functions: a geometric heuristic computing Euclidean distance that remains admissible and provides lower bounds on path length guaranteeing optimality when weight parameter W equals 1.0, a stress heuristic that remains inadmissible but estimates aggregate stress along the remaining path based on zone stress scores interpolated from known measurements, and a safety heuristic that remains inadmissible but penalizes routes passing through high-violation zones using learned penalty parameter . The learned combination architecture implements linear weighted models where the learned heuristic h_learned equals w·h_geo + w·h_stress + w·h_safety with weights = (w, w, w) learned through offline training using Ridge regression on over 100 synthetic training samples, online adaptation through stochastic gradient descent from user feedback signals, and L2 regularization with penalty parameter = 0.01 to prevent overfitting to individual preferences.

Addressing the challenge of limited real data consisting of only 5 records from BigQuery, we employ a sophisticated hybrid data augmentation strategy combining four complementary techniques. Parametric synthetic generation fits multivariate Gaussian distributions to real 17-dimensional feature vectors, samples from fitted distributions with controlled noise injection, enforces physical constraints maintaining feature values in valid range from 0 to 3, and validates realism using holdout real samples to detect distribution drift. Bootstrap resampling with noise injection resamples with replacement adding Gaussian noise with standard deviation 0.15, preserving correlations between features while introducing realistic variation, maintaining the original data distribution shape, and generating 100 augmented samples. SMOTE for minority class oversampling creates samples along line segments connecting neighbors using k = 2 neighbors, avoids exact duplication while preserving local structure, and generates 150 intelligently interpolated samples. Domain knowledge initialization leverages urban planning principles to initialize feature weights, assigning high impact weights (0.8–1.0) for violation features (indices 0–4), moderate impact weights (0.4–0.6) for density and traffic features (indices 5–7), moderate-low impact weights (0.3–0.5) for infrastructure features (indices 8–11), high impact weights (0.7–0.9) for behavior features (indices 12–14), and context-dependent weights (0.2–0.8) for environment features (indices 15–16). This comprehensive pipeline produces 5 real samples for validation and testing, 100 bootstrap samples preserving structure, 150 SMOTE samples through intelligent interpolation, and 200 parametric samples covering feature space, yielding 455 total samples with 450 allocated for training and 5 reserved for testing.

Success requires achieving training convergence with loss reduction exceeding 80 percent monitored through training curves, generalization achieving test MAE below 15 percent of mean target validated through leave-one-out cross-validation on real samples, improvement versus baseline demonstrating 15 to 25 percent better user preference match measured through simulated user studies with preference rankings, and weight interpretability where top 5 features align with domain knowledge confirmed through coefficient analysis. We validate using 5-fold cross-validation on synthetic plus real combined dataset, leave-one-out cross-validation on 5 real samples exclusively, comparison to random weight baseline and uniform weight baseline, and statistical significance testing using paired t-test with significance threshold $p < 0.05$.

Adaptive Regularization Implementation

The third objective designs and implements dynamic weight parameter adaptation that adjusts search intensity based on query characteristics, inspired by weighted A* from Programming Problem 3 where we empirically validated that weight W = 1.0 produces optimal solutions but expands more nodes, weight W = 1.2 provides a sweet spot often achieving optimal solutions with 30 percent fewer nodes, and weight W = 1.5 executes faster with 60 percent fewer nodes but loses optimality by approximately 2 steps average. The adaptive weight selection algorithm (see Eq. 1 in Appendix) computes weight dynamically based on query context including path length estimate derived from Euclidean distance serving as proxy for complexity, urgency level (low, medium, or high) affecting user tolerance for computation time, quality threshold specifying minimum acceptable solution quality on scale from 0 to 1, and time budget limiting maximum computation time in seconds for real-time responsiveness.

The algorithm begins with base weight 1.2 validated as optimal from Programming Problem 3, then adjusts for urgency, complexity, and quality requirements before clipping the final weight between 1.0 and 2.0. The system implements computational budget management through anytime algorithm properties including maximum node expansion limit of 10000 nodes preventing pathological cases from consuming excessive resources, time cutoff of 3.0 seconds meeting real-time constraints for mobile applications, progressive relaxation that increases W if no solution is found within budget allowing graceful degradation, and best-so-far tracking that returns the best solution when budget is exhausted ensuring users always receive actionable results even under strict constraints. Success requires speed improvement achieving 30 to 60 percent fewer nodes versus W = 1.0 measured by node expansion count, quality preservation maintaining over 90 percent of optimal quality compared to Dijkstra's algorithm, response time under 3 seconds for 95 percent of queries measured by timing distribution, and weight correlation exceeding $r > 0.7$ between W and query complexity validated through Pearson correlation coefficient.

NYC Vibe Check Data Integration

The fourth objective integrates with existing NYC camera network data providing real-world context and validation for the routing system without relying on cloud-based infrastructure or external API services. We implement a local data integration architecture where all data resides in exported JSON files enabling offline operation and reproducible experiments. The route graph construction process loads 907 camera zones from complete Voronoi zones JSON representing Voronoi cells tessellating NYC geography, computes adjacency based on shared Voronoi polygon boundaries using computational geometry algorithms, assigns edge weights using Euclidean distance between zone centers measured in meters, and attaches node attributes from BigQuery exports including position as latitude-longitude pairs, stress score derived from temperature score field in zone analyses, violation history from recent violations in realtime violations table, and predicted traffic from ARIMA predictions in traffic predictions table.

Since we work with static data snapshots rather than real-time streams, we employ data augmentation for missing coverage through k-nearest neighbors spatial interpolation that trains on known zones using 5 neighbors with distance weighting, predicts for missing zones lacking direct measurements, and validates through spatial consistency requiring neighboring zones to have similar stress with correlation coefficient $r > 0.6$. Success requires graph construction producing 907 nodes with approximately 2500 edges validated through NetworkX graph analysis, data coverage achieving 100 percent nodes with stress estimates through combination of imputation and real data, feature extraction successfully parsing all 17 dimensions for each node with parse success rate exceeding 95 percent, and spatial consistency where neighboring zones exhibit similar stress validated through spatial autocorrelation analysis using Moran's I statistic.

Comparative Algorithm Analysis

The fifth objective conducts rigorous empirical comparison of the proposed adaptive heuristic system against established baseline algorithms across four evaluation dimensions. We implement and compare six algorithms: Dijkstra's algorithm serving as optimal baseline, A* with Euclidean distance as admissible baseline, A* with Manhattan distance as alternative admissible baseline, weighted A* with W = 1.2 as suboptimal fast baseline

from Programming Problem 3, greedy best-first search as inadmissible fast baseline, and adaptive heuristic A* as the proposed system combining learned weighted combination with dynamic W adjustment and path sampling for diversity.

The four-dimensional evaluation framework assesses solution quality through normalized weighted combination (see Eq. 2 in Appendix) with user preference weights reflecting stress-prioritized preferences (w_d = 0.3, w_s = 0.5, w_v = 0.2) and comparison to Dijkstra's optimal for distance component. Computational efficiency metrics include nodes expanded as primary efficiency metric, CPU time measured by wall-clock timing averaged over 5 runs, memory usage tracking peak memory consumption during search, and scalability measuring performance versus graph size on networks containing 100, 500, and 907 nodes. Optimality analysis computes optimality gap (see Eq. 3 in Appendix) measuring deviation from Dijkstra's optimal baseline, with acceptable gap defined as less than 10 percent for practical applications. User preference match simulates user preferences where 80 percent of users prefer lower stress over shorter distance, ranks algorithm solutions by simulated preference, and measures rank correlation using Spearman's .

The experimental design constructs test sets containing 25 real routing scenarios from NYC manually designed by domain experts and 75 synthetic scenarios using random start-goal pairs sampled uniformly across the network, stratified by difficulty (33 percent easy routes under 2 km, 33 percent medium routes 2–5 km, 33 percent hard routes exceeding 5 km). Statistical analysis employs paired t-tests comparing each algorithm to proposed system, ANOVA across all six algorithms, effect size using Cohen's d for practical significance, significance level = 0.05, and multiple comparison correction through Bonferroni adjustment. Success requires achieving less than 10 percent longer distance versus Dijkstra with 40 percent fewer nodes, similar distance versus A* with Euclidean with 20 percent fewer nodes, better user preference match versus weighted A* by +15 percent, and better solution quality versus greedy best-first by +25 percent.

Real-World Validation

The sixth objective validates stress predictions and route quality assessments against real-world pedestrian experience data through four complementary experiments. Experiment 1 evaluates stress prediction accuracy by predicting stress scores for two real zones using learned heuristic, comparing predicted versus actual temperature score from BigQuery data, targeting mean absolute error below 20 percent of score range spanning 5 to 24. Experiment 2 validates feature importance by extracting learned feature weights from Ridge regression, comparing to domain knowledge rankings derived from urban planning principles, targeting Spearman rank correlation > 0.6. Experiment 3 assesses time-of-day consistency by grouping synthetic samples by time-of-day categories (morning, midday, evening, night), validating that predictions vary appropriately with night showing lower traffic stress, targeting ANOVA F-statistic significance with $p < 0.05$. Experiment 4 conducts simulated user study with N = 100 virtual users each assigned random user profile with varied preferences, presents each user with 25 scenarios comparing routes from different algorithms, collects pairwise comparisons using utility functions to simulate choice behavior, and analyzes preference patterns across algorithm pairs.

Success requires stress prediction MAE below 3.5 on scale from 5 to 24 comparing to real temperature score, user preference agreement exceeding 75 percent with simulated profiles measuring how often system recommendations match profile preferences, feature importance correlation > 0.6 validating that learned weights align with domain knowledge, and cross-validation consistency showing less than 10 percent variance across folds measured by 5-fold CV standard deviation indicating stable model performance.

Theoretical Analysis

The seventh objective provides rigorous theoretical analysis of algorithm properties including admissibility, consistency, optimality guarantees, and computational complexity. For admissibility analysis we prove that a learned heuristic h_learned = w_i·h_i remains admissible if and only if all base heuristics h_i are admissible and all weights w_i are non-negative (see Theorem 1 in Appendix), with corollary that our learned heuristic with inadmissible components including stress and safety remains inadmissible, thus weighted A* with W > 1 may not find optimal paths. For consistency analysis we prove that if all base heuristics are consistent (h(n) ≤

c(n,a,n') + h(n') for all edges), then their positive weighted combination remains consistent (see Theorem 2 in Appendix), with implication that consistency ensures each node is expanded at most once improving efficiency.

Optimality analysis applies Pohl's theorem from 1970 stating that if h(n) is admissible, weighted A* with weight W finds solutions with cost at most W times optimal, conducts empirical investigation measuring actual optimality gaps on test scenarios comparing to theoretical upper bound, and identifies conditions where bound is tight versus loose. Computational complexity analysis characterizes time complexity for Dijkstra as O(E + V log V), for A* with admissible h as O(b^d) in worst case where b is effective branching factor and d is solution depth, for weighted A* as O(b^(d/W)) in expected case, and for adaptive heuristic as O(b^d + T_learning) where T_learning ≈ 0.001 seconds with trained model (see Appendix for details). Success requires delivering admissibility proof as formal theorem validated through peer review and empirical counter-examples, optimality bounds through theoretical gap derivation compared to measured gaps, complexity analysis providing Big-O characterization validated by empirical scaling experiments, and scalability model providing predictive formula with $R^2 > 0.85$ on test graphs.

Scalability and Performance Analysis

The eighth objective systematically measures and analyzes algorithm performance as function of problem size, characterizing scalability properties and identifying computational bottlenecks. The experimental design varies graph size across small networks (100 nodes representing Manhattan subset), medium networks (500 nodes spanning multiple boroughs), and large networks (907 nodes covering full NYC camera network). Difficulty varies across short queries (under 1 km within neighborhood), medium queries (1–5 km crossing neighborhoods), and long queries (exceeding 5 km crossing boroughs). Replication uses 10 random scenarios per size-difficulty combination with 5 runs per scenario reporting median and interquartile range, totaling 450 runs per algorithm.

We collect comprehensive performance measurements per run including efficiency metrics (nodes expanded, CPU time, memory), quality metrics (path length, path stress, optimality gap), and convergence metrics (first solution time, final solution time, iterations to convergence). The scaling analysis fits empirical power law hypothesizing that nodes expanded scales as $N \propto V^{\alpha}$ where α depends on algorithm, using logarithmic regression $\log N = \alpha \log V + \beta$ to estimate exponent. Expected results show Dijkstra with $\alpha \approx 1.0$ indicating linear scaling, A* with $\alpha \approx 0.7$–0.9 showing sublinear due to heuristic guidance, weighted A* with W = 1.2 showing $\alpha \approx 0.6$–0.8 through more aggressive pruning, and adaptive heuristic showing $\alpha \approx 0.5$–0.7 achieving best pruning with learned weights.

Bottleneck identification profiles using cProfile and line_profiler to identify computational hotspots, expecting bottlenecks in priority queue operations (heap push-pop at O(log n) per operation), heuristic function evaluation (O(1) for geometric, O(k) for learned), and path reconstruction (O(d) where d is solution depth). Optimization opportunities include lazy evaluation of heuristics computing only when needed, heuristic value caching for revisited nodes storing computed values, and Fibonacci heap for improved priority queue offering O(1) amortized decrease-key. Success requires scaling exponent $\alpha < 0.8$ for adaptive heuristic validated by power law fit $R^2 > 0.85$, real-time performance solving 95 percent queries in under 3 seconds on 907-node graph measured by empirical timing, memory efficiency maintaining peak usage below 100 MB tracked by memory profiling, and speedup versus Dijkstra achieving 5–10× faster on average measured by comparative timing.

Success Criteria Summary

Table 1 summarizes the success criteria across all eight objectives, providing a comprehensive view of the measurable outcomes defining project success.

Table 1: Success criteria for project objectives

Objective	Primary Metric	Target	Validation Method
Path Sampling	Sample diversity	>70% unique	Jaccard distance
Heuristic Learning	Test MAE	<15% of mean	Leave-one-out CV
Adaptive Regularization	Speed improvement	30-60% fewer nodes	Node count
Data Integration	Graph construction	907 nodes connected	NetworkX
Algorithm Comparison	Statistical significance	$p < 0.05$	Paired t-test
Real-World Validation	User preference match	>75%	Simulated study
Theoretical Analysis	Proof completeness	All theorems proved	Peer review
Scalability	Scaling exponent	$\square < 0.8$	Power law fit

Overall project success requires six out of eight objectives meeting targets, establishing comprehensive validation through multiple independent criteria ensuring robust evidence for methodology effectiveness.

Work Plan and Timeline

The project follows a structured 10-week timeline divided into five major phases, with clear deliverables and milestones enabling progress tracking and risk mitigation. Weeks 1-2 (November 1-14) focus on foundation and design, conducting comprehensive literature review surveying papers on heuristic learning, weighted A*, and path sampling to establish theoretical foundation, finalizing system design with complete algorithm architecture and data structures documented in pseudocode, and setting up Python development environment with all dependencies (NumPy ≥1.21, pandas ≥1.3, scikit-learn ≥1.0, NetworkX ≥2.6) with version control through Git. Deliverable: design document with algorithm pseudocode ready for instructor review at Checkpoint 1 (November 14).

Weeks 3-4 (November 15-28) address core implementation, implementing path sampling with three sampling strategies (uniform random walk, importance-weighted sampling, Thompson sampling), implementing base heuristics (geometric, stress-based, safety-based), and implementing baseline algorithms (Dijkstra, A* with Euclidean and Manhattan heuristics, weighted A* with W = 1.2, greedy best-first search). Deliverable: working prototype with all baseline algorithms demonstrated at Checkpoint 2 (November 28).

Weeks 5-6 (November 29-December 12) develop learning and adaptation, executing data augmentation generating 450 synthetic training samples through bootstrap resampling, SMOTE, and parametric generation, training models using Ridge regression validated on real data with hyperparameter tuning, implementing adaptive regularization with dynamic W selection, and integrating learned weights into search algorithms. Deliverable: complete adaptive heuristic system validated at Checkpoint 3 (December 12).

Weeks 7-8 (December 13-26) execute comprehensive experimentation, running experimental suite with 450 test runs across all scenarios (6 algorithms × 100 scenarios × 5 replications), collecting performance metrics (nodes expanded, CPU time, memory usage, path quality, optimality gaps), conducting statistical analysis (paired t-tests, ANOVA, effect size calculations with Bonferroni correction), and performing bottleneck profiling using cProfile. Deliverable: complete experimental results dataset with statistical validation.

Weeks 9-10 (December 27-January 9) complete analysis and reporting, performing theoretical analysis proving admissibility conditions and deriving complexity bounds, creating result visualizations using matplotlib and seaborn, writing final report (15-20 pages following scientific writing principles), documenting code with comprehensive comments and README, and preparing presentation with slides and demonstration materials. Deliverable: final report, presentation, and code repository for submission (January 9).

All work will be completed individually by Gil Raitses with instructor feedback at four key checkpoints: Checkpoint 1 (November 14) for design document review, Checkpoint 2 (November 28) for prototype demonstration, Checkpoint 3 (December 12) for learning system validation, and final submission (January 9) for complete project deliverables.

Reference List

Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson. This foundational textbook provides comprehensive coverage of search algorithms in Chapter 3 covering uninformed and informed search including A* and its variants, and Chapter 4 addressing search in complex environments including local search and optimization, establishing theoretical framework for this research.

Samadi, M., Felner, A., & Schaeffer, J. (2008). Learning from multiple heuristics. *AAAI*, 8, 357-362. This work demonstrates techniques for combining multiple heuristic functions to improve search performance, directly informing our approach to learning weighted combinations of geometric, stress, and safety heuristics.

Thayer, J. T., & Ruml, W. (2011). Bounded suboptimal search: A direct approach using inadmissible estimates. *IJCAI*, 11, 674-679. This paper analyzes weighted A* and bounded suboptimal search providing theoretical foundation for our adaptive regularization mechanism controlling trade-offs between solution quality and computational efficiency.

Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. *European Conference on Machine Learning*, 282-293. This seminal work on Monte Carlo tree search informs our path sampling approach, particularly the Thompson sampling strategy balancing exploration and exploitation.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3-4), 193-204. This foundational paper establishes theoretical properties of weighted A* including optimality bounds that inform our adaptive weight selection and provide basis for theoretical analysis in Objective 7.

Hansen, E. A., & Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research*, 28, 267-297. This work on anytime algorithms informs our computational budget management enabling the system to return best available solution when time or node expansion limits are reached.

Quercia, D., Schifanella, R., & Aiello, L. M. (2014). The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. *ACM HyperText*, 116-125. This research on quality-optimized urban routing provides motivation and validation for incorporating non-distance factors including aesthetics and psychological comfort into navigation systems.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16, 321-357. This foundational work on synthetic data generation provides methodology for our data augmentation strategy addressing limited real-world training samples through intelligent interpolation.

Conclusion

This proposal presents a comprehensive plan for developing and validating an adaptive heuristic learning framework for stress-optimized pedestrian navigation that makes three key contributions to artificial intelligence research and practice. The methodological innovation integrates path sampling, machine learning, and adaptive search in a unified framework that extends classical search algorithms with modern data-driven optimization techniques. The practical application demonstrates real-world system performance using NYC camera network data spanning 907 zones across 3626 square kilometers with demonstrable impact on routing quality measured through multiple objective functions balancing distance, stress, and safety. The theoretical rigor provides formal analysis of algorithm properties including admissibility, optimality bounds, and computational complexity, connecting empirical results to established search theory.

The project directly applies and extends concepts from CIS 667, particularly informed search from Chapter 3 and learning agents from Chapter 21 of Russell and Norvig's textbook, while addressing a practical urban

planning problem with implications for accessible cities, emergency response optimization, and human-centered transportation systems. The comprehensive evaluation framework with eight well-defined objectives ensures rigorous validation through quantitative metrics (30-60 percent reduction in nodes expanded), statistical significance testing (paired t-tests and ANOVA with $p < 0.05$), and real-world validation through simulated user studies (over 75 percent preference match), producing meaningful results that advance both algorithmic understanding and practical deployment capabilities.

The data robustness strategy addresses limited real-world data (5 records) through sophisticated data augmentation pipeline combining bootstrap resampling (100 samples), SMOTE (150 samples), and parametric generation (200 samples), validated through statistical cross-validation and confirmed through domain knowledge alignment ensuring learned weights match urban planning principles. The proposed work remains feasible within a 10-week timeline for an individual student project, with clear milestones at four key checkpoints and well-defined deliverables at each stage including documented algorithms, working code, experimental results, and comprehensive technical report.

Student Signature: *Gil Raitses*

Date: October 27, 2025

Appendix A: Mathematical Formulations

This appendix presents the mathematical formulations underlying the adaptive heuristic learning framework. We provide formal definitions for key equations referenced throughout the proposal, including the adaptive weight selection algorithm (Eq. 1), solution quality scoring function (Eq. 2), optimality gap computation (Eq. 3), and theoretical results on admissibility and consistency (Theorems 1-2).

Equation 1: Adaptive Weight Selection

The adaptive weight W is computed dynamically based on query context to balance solution quality against computational efficiency:

$$W = \text{clip}(W_{\text{base}} + \Delta_{\text{urgency}} + \Delta_{\text{complexity}} + \Delta_{\text{quality}}, 1.0, 2.0)$$

where $W_{\text{base}} = 1.2$ (validated empirically), $\Delta_{\text{urgency}} \in \{0.0, 0.1, 0.3\}$ for low, medium, high urgency, $\Delta_{\text{complexity}} = +0.1$ if distance > 5000m or -0.1 if distance < 1000m, and Δ_{quality} forces W = 1.0 if threshold > 0.95.

Equation 2: Solution Quality Score

The multi-objective quality score combines normalized distance, stress, and violations:

$$Q(\text{path}) = w_d \cdot \frac{d(\text{path})}{d_{\max}} + w_s \cdot \frac{s(\text{path})}{s_{\max}} + w_v \cdot \frac{v(\text{path})}{v_{\max}}$$

where $w_d = 0.3$, $w_s = 0.5$, $w_v = 0.2$ represent user preference weights for distance, stress, and violations respectively, and the denominators normalize to [0,1] scale.

Equation 3: Optimality Gap

The optimality gap measures deviation from optimal solution as percentage:

$$\text{gap} = \frac{\text{cost}_{\text{algorithm}} - \text{cost}_{\text{optimal}}}{\text{cost}_{\text{optimal}}} \times 100\%$$

where $\text{cost}_{\text{optimal}}$ is obtained from Dijkstra's algorithm for the distance objective.

Theorem 1: Admissibility of Weighted Heuristic Combinations

A learned heuristic $h_{\text{learned}} = \sum w_i \cdot h_i$ remains admissible if and only if all base heuristics h_i are admissible and all weights $w_i \geq 0$. Since our learned heuristic includes inadmissible components (stress and safety heuristics), the combined heuristic is inadmissible.

Theorem 2: Consistency of Weighted Heuristic Combinations

If all base heuristics are consistent (i.e., $h(n) \leq c(n, a, n') + h(n')$ for all edges), then their positive weighted combination remains consistent. Consistency ensures each node is expanded at most once, improving search efficiency.

Data Schema Specifications

The 17-dimensional feature vector encoding follows the structure shown in Table 1, where each feature captures distinct aspects of urban stress and safety.

Table 2: Feature vector specification for urban stress modeling {#tbl-features} Features indexed 0 through 4 represent **violation severity** on scale from 0 meaning none to 3 meaning high, covering pedestrian walkway violation where bikes use pedestrian walkways, dangerous bike lane position indicating poor bike lane positioning, bike red light violation where bikes run red lights, blocking pedestrian flow indicating flow obstruction, and car bike lane violation where cars intrude into bike lanes. Features indexed 5 through 7 represent **density and volume counts** on scale from 0 to 3, covering pedestrian density indicating crowd density, vulnerable population indicating presence of elderly, children, or disabled persons, and traffic volume counting vehicle density. Features indexed 8 through 11 represent **infrastructure quality indicators** on scale from 0 to 2, covering visibility conditions assessing lighting and weather visibility, missing barriers identifying infrastructure gaps, poor signage evaluating sign quality, and signal malfunction detecting traffic signal issues. Features indexed 12 through 14 represent **behavior assessments** with cyclist speed estimate on scale 0 to 3 measuring speed, aggressive behavior on scale 0 to 2 assessing aggressive actions, and infrastructure quality on scale 0 to 2 evaluating overall quality. Features indexed 15 through 16 represent **environmental factors** with weather impact as float from 0.0 to 1.0 measuring weather effects, and overall safety risk on scale 0 to 3 providing aggregate risk assessment.

Index	Feature Name	Description	Range	Type
0	pedestrian_walkway_violation	Bikes on pedestrian walkways	0-3	Severity
1	dangerous_bike_lane_position	Poor bike lane positioning	0-3	Severity
2	bike_red_light_violation	Bikes running red lights	0-3	Severity
3	blocking_pedestrian_flow	Flow obstruction	0-3	Severity
4	car_bike_lane_violation	Cars in bike lanes	0-3	Severity
5	pedestrian_density	Crowd density	0-3	Count
6	vulnerable_population	Elderly, children, disabled	0-3	Presence
7	traffic_volume	Vehicle count	0-3	Count
8	visibility_conditions	Lighting, weather visibility	0-2	Quality

Index	Feature Name	Description	Range	Type
9	missing_barriers	Infrastructure gaps	0-2	Boolean
10	poor_signage	Sign quality	0-2	Quality
11	signal_malfunction	Traffic signal issues	0-2	Boolean
12	cyclist_speed_estimate	Speed assessment	0-3	Speed
13	aggressive_behavior	Aggressive actions	0-2	Severity
14	infrastructure_quality	Overall quality	0-2	Quality
15	weather_impact	Weather effects	0.0-1.0	Float
16	overall_safety_risk	Aggregate risk	0-3	Severity

Graph Construction Details

The NYC camera zone network represents urban geography through 907 nodes derived from Voronoi tessellation of camera locations. Each node maintains zone_id as string identifier following format like MN_001 for Manhattan or BK_011 for Brooklyn, position as tuple of float latitude and float longitude enabling geometric calculations, borough as string indicating Manhattan, Brooklyn, Queens, Bronx, or Staten Island for geographic grouping, features as NumPy array with shape 17 storing learned features, stress_score as float representing temperature score ranging 0 to 30, violation_count as integer counting recent violations, and predicted_traffic as integer forecasting vehicle count. Each edge maintains source and target as string node identifiers, distance as float Euclidean distance in meters, estimated_time as float walking time in seconds assuming 1.4 meters per second, stress_cost as float aggregate stress along segment, and safety_cost as float violation density.

Adjacency computation determines that two zones are adjacent if their Voronoi polygons share boundary requiring computational geometry operations using Shapely library, with average degree approximately 3 to 4 edges per node consistent with planar graph properties, and total edges approximately 2500 forming sparse graph structure enabling efficient search. The network exhibits small-world properties common in urban networks where average path length scales logarithmically with network size and high local clustering coefficient indicates neighborhoods with dense internal connections.

Appendix B: Algorithm Pseudocode

This appendix provides detailed pseudocode descriptions for the core algorithms in the adaptive heuristic learning framework. These algorithmic specifications support implementation and provide clear reference for understanding the computational procedures described in the main proposal.

Adaptive Heuristic A* Algorithm

The **adaptive heuristic A* algorithm** integrates learned weights and dynamic regularization into classical A* search. The algorithm receives parameters start_node, goal_node, learned_weights as array of 3 weights, and query_context containing path_length_estimate, urgency_level, quality_threshold, and time_budget. Initialization computes adaptive_weight using compute_adaptive_weight function on query_context, sets nodes_expanded to 0, start_time to current time, initializes priority_queue as min-heap with start_node having f_value equal to 0 plus adaptive_weight times learned_heuristic of start_node using learned_weights, initializes explored_set as empty set, and initializes came_from as empty dictionary for path reconstruction.

The main loop continues while priority_queue not empty and nodes_expanded less than 10000 and elapsed time less than time_budget. Each iteration extracts current_node and current_f from priority_queue, checks if current_node equals goal_node returning reconstruct_path result if true, skips if current_node in explored_set already processed, adds current_node to explored_set, increments nodes_expanded counter, and iterates over neighbors of current_node. For each neighbor, the algorithm skips if neighbor in explored_set, computes tentative_g as g_value of current_node plus edge_cost from current_node to neighbor, computes h_value as

adaptive_weight times learned_heuristic of neighbor using learned_weights, computes f_value as tentative_g plus h_value, and if neighbor not in priority_queue or tentative_g less than existing g_value of neighbor, updates came_from with neighbor pointing to current_node, sets g_value of neighbor to tentative_g, and pushes neighbor with f_value to priority_queue. If loop exits without finding goal, the algorithm returns best partial solution found or None if no valid path.

Learned Heuristic Function

The learned heuristic function combines three base heuristics with learned weights. The function receives node, goal_node, and learned_weights containing w_geo, w_stress, and w_safety. It computes h_geo as Euclidean distance from node.position to goal_node.position, computes h_stress as node.stress_score plus estimated path stress to goal using linear interpolation, computes h_safety as node.violation_count plus estimated violations to goal using zone violation density, and returns w_geo · h_geo + w_stress · h_stress + w_safety · h_safety.

Adaptive Weight Computation Function

The adaptive weight computation function implements the dynamic W selection described in Equation 1. The function receives query_context and initializes base_weight to 1.2. It adjusts for urgency by adding urgency_adjustment where urgency_adjustment equals 0.0 if urgency_level equals low, 0.1 if medium, or 0.3 if high. It adjusts for complexity by adding 0.1 if path_length_estimate exceeds 5000 or subtracting 0.1 if path_length_estimate below 1000. It adjusts for quality by forcing weight to 1.0 if quality_threshold exceeds 0.95 or allowing weight up to 2.0 if quality_threshold below 0.80. Finally it clips weight between 1.0 and 2.0 and returns final weight.

Appendix C: Implementation Code

This appendix provides Python implementation code for key data processing and evaluation components. These code listings demonstrate the practical realization of the algorithms and methods described in the proposal, supporting reproducibility and implementation clarity.

Bootstrap Resampling with Noise Injection

This function implements bootstrap resampling with Gaussian noise to preserve feature correlations while introducing realistic variation.

```
def bootstrap_with_noise(X, y, n_samples=100, noise_std=0.15):
    """Generate bootstrap samples with additive noise"""
    from sklearn.utils import resample
    X_boot, y_boot = resample(X, y, n_samples=n_samples, replace=True)
    noise = numpy.random.normal(0, noise_std, X_boot.shape)
    X_boot = numpy.clip(X_boot + noise, 0, 3)
    return X_boot, y_boot
```

SMOTE Augmentation

This function applies Synthetic Minority Over-sampling Technique (SMOTE) to create samples along line segments connecting neighbors.

```
def smote_augmentation(X, y, n_samples=150):
    """Generate SMOTE samples through intelligent interpolation"""
    from imblearn.over_sampling import SMOTE
    smote = SMOTE(sampling_strategy='auto', k_neighbors=2)
```

```
    X_augmented, y_augmented = smote.fit_resample(X, y)
    return X_augmented, y_augmented
```

Parametric Synthetic Generation

This function fits multivariate Gaussian distributions to real feature vectors and samples from fitted distributions with controlled noise.

```
def parametric_generation(real_samples, n_synthetic=200):
    """Generate parametric samples from fitted Gaussian"""
    mu = numpy.mean(real_samples, axis=0)
    sigma = numpy.cov(real_samples, rowvar=False)
    synthetic = numpy.random.multivariate_normal(mu, sigma, n_synthetic)
    synthetic = numpy.clip(synthetic, 0, 3)
    return synthetic
```

Solution Quality Scoring

This function implements the multi-objective quality score described in Equation 2, combining normalized distance, stress, and violation metrics.

```
def solution_quality_score(path, w_d=0.3, w_s=0.5, w_v=0.2,
                           max_distance=10000, max_stress=30, max_violations=100):
    """Compute weighted quality score for a path"""
    normalized_distance = path.distance / max_distance
    normalized_stress = path.stress / max_stress
    normalized_violations = path.violations / max_violations
    return (w_d * normalized_distance +
            w_s * normalized_stress +
            w_v * normalized_violations)
```

Optimality Gap Computation

This function implements Equation 3 to measure deviation from optimal solution as a percentage.

```
def optimality_gap(algorithm_cost, optimal_cost):
    """Calculate optimality gap as percentage"""
    return ((algorithm_cost - optimal_cost) / optimal_cost) * 100
```

Statistical Significance Testing

This function performs paired t-tests to assess statistical significance of performance differences between algorithms.

```
def compute_statistical_significance(results_proposed, results_baseline):
    """Compute paired t-test and effect size"""
    import scipy.stats
    paired_differences = results_proposed - results_baseline
    t_statistic, p_value = scipy.stats.ttest_rel(results_proposed, results_baseline)
    effect_size = numpy.mean(paired_differences) / numpy.std(paired_differences)
    return {
        'p_value': p_value,
        't_statistic': t_statistic,
        'effect_size': effect_size,
```

```
        'significant': p_value < 0.05
    }
```