SUPPLEMENTARY MATERIALS

Investigating the Potential and Performance of
Generative AI for Vehicle Routing Problem

TABLE OF CONTENTS

S1 - SUPPLEMENTARY MATERIAL
Claude.ai Prompts and Parsing Methodology

S1.1. COMPLETE THAI PROMPTS WITH ENGLISH TRANSLATIONS

Initial Problem Description Prompt

Thai:

ฉันมีปัญหาการวางแผนเส้นทางสำหรับการบำรุงรักษาเครื่อง AED 80 จุดในจังหวัดเชียงราย

ข้อมูลปัญหา:
- ฐานปฏิบัติการ: เชียงใหม่ (18.7883, 98.9853)
- จำนวนจุดให้บริการ: 80 จุด
- ระยะเวลาวางแผน: 6 วัน
- เวลาทำงาน: 8:00-16:30 (8.5 ชั่วโมง/วัน)
- เวลาให้บริการ: 30 นาที/จุด
- ความเร็วเฉลี่ย: 50 กม./ชม.

พิกัดจุดให้บริการ (Latitude, Longitude):
1. 19.9105, 99.8318 (Chiang Rai Provincial Hospital)
2. 19.8891, 99.8156 (Mae Fah Luang University)
[... 78 more locations ...]
80. 19.8956, 99.8467 (Technical College)

โปรดวางแผนเส้นทางที่:
1. ทุกจุดได้รับบริการครบ
2. ระยะทางรวมน้อยที่สุด
3. ไม่เกินเวลาทำงาน 8.5 ชั่วโมง/วัน
4. กลับถึงฐานภายใน 16:30

English Translation:

I have a routing problem for AED maintenance at 80 locations in Chiang Rai Province

Problem details:
- Depot: Chiang Mai (18.7883, 98.9853)
- Service locations: 80 points
- Planning horizon: 6 days
- Working hours: 8:00 AM - 4:30 PM (8.5 hours/day)
- Service time: 30 minutes/location
- Average speed: 50 km/h

Service location coordinates (Latitude, Longitude):
1. 19.9105, 99.8318 (Chiang Rai Provincial Hospital)
2. 19.8891, 99.8156 (Mae Fah Luang University)
[... 78 more locations ...]
80. 19.8956, 99.8467 (Technical College)

Please plan routes that:
1. All locations serviced
2. Minimize total distance
3. Within 8.5 hours/day working time
4. Return to depot by 4:30 PM

Refinement Prompts

Prompt 2 (Request structured output):
Thai: `กรุณาแสดงผลเป็นตารางโดยระบุ: วันที่, ลำดับจุด, ระยะทางรวม, เวลารวม`
English: "Please show results as table with: day, location sequence, total distance, total time"

Prompt 3 (Verify feasibility):
Thai: `ตรวจสอบว่าแต่ละวันไม่เกิน 8.5 ชั่วโมง รวมเวลาเดินทางและให้บริการ`
English: "Verify each day within 8.5 hours including travel and service time"

Prompt 4 (Request adjustments):
Thai: `วันที่ 4 ดูเหมือนจะเกินเวลา ช่วยปรับปรุงให้อยู่ในเวลา`
English: "Day 4 seems to exceed time limit, please adjust to fit within working hours"

S1.2. EXAMPLE RAW OUTPUT

Claude.ai Response (abbreviated):

Here's an optimized 6-day routing plan:

Day 1: Depot → Location 5 → Location 12 → ... → Location 67 → Depot
Distance: 158.4 km | Time: 8.3 hours

Day 2: Depot → Location 3 → Location 15 → ... → Location 71 → Depot
Distance: 162.3 km | Time: 8.4 hours

[... Days 3-6 ...]

Total Distance: 941.6 km
All routes verified within 8.5 hour constraint.

S1.3. PYTHON PARSING SCRIPT

```
import re
from typing import List, Tuple

def parse_claude_output(text: str) -> dict:
    """Extract route information from Claude.ai natural language output"""

    # Extract day-by-day routes
    day_pattern = r'Day (\d+).*?→(.*?)→ Depot'
    distance_pattern = r'Distance:\s*([\d.]+)\s*km'
    time_pattern = r'Time:\s*([\d.]+)\s*hours?'

    routes = []
    for match in re.finditer(day_pattern, text, re.DOTALL):
        day = int(match.group(1))
        route_text = match.group(2)

        # Extract location IDs
        locations = re.findall(r'Location (\d+)', route_text)
        locations = [int(loc) for loc in locations]

        # Extract distance and time (search nearby text)
        context = text[match.start():match.start()+200]
        distance_match = re.search(distance_pattern, context)
```

```
            time_match = re.search(time_pattern, context)

            routes.append({
                'day': day,
                'locations': locations,
                'distance_km': float(distance_match.group(1)) if distance_match else None,
                'time_hours': float(time_match.group(1)) if time_match else None
            })

        # Extract total distance
        total_pattern = r'Total Distance:\s*([\d.]+)\s*km'
        total_match = re.search(total_pattern, text)
        total_distance = float(total_match.group(1)) if total_match else None

        return {
            'routes': routes,
            'total_distance': total_distance,
            'n_days': len(routes)
        }

    def validate_solution(solution: dict, n_locations: int = 80) -> dict:
        """Validate parsed solution meets constraints"""

        all_locations = set()
        for route in solution['routes']:
            all_locations.update(route['locations'])

        validation = {
            'all_locations_covered': len(all_locations) == n_locations,
            'no_duplicates': sum(len(r['locations']) for r in solution['routes']) == n_locations,
            'time_feasible': all(r['time_hours'] <= 8.5 for r in solution['routes'] if r['time_hours']),
            'n_days_correct': solution['n_days'] == 6
        }

        validation['is_valid'] = all(validation.values())
        return validation

    # Usage
    output_text = """[Claude.ai response text]"""
    parsed = parse_claude_output(output_text)
    validation = validate_solution(parsed)

    if validation['is_valid']:
        print(f"Valid solution: {parsed['total_distance']} km")
    else:
        print(f"Validation issues: {validation}")
```

S1.4. VALIDATION METHODOLOGY

Constraints Verified:

- All 80 locations assigned exactly once
- Each day ≤ 8.5 hours (travel + service time)
- Routes start and end at depot
- 6-day planning horizon

Validation Process:

- Parse locations from natural language output
- Calculate actual distances using Haversine formula
- Calculate actual times: (distance/50)*60 + (locations*30)
- Flag violations for manual review
- Accept solution if all constraints satisfied

Error Handling:
- Ambiguous parsing → Request structured output from Claude.ai
- Time violations → Request route adjustment
- Missing locations → Request complete coverage
- Excessive distance → Request optimization focus

S1.5. REPRODUCIBILITY NOTES

Claude.ai Configuration:
- Model: claude-sonnet-3-5-20240620
- Temperature: 1.0 (default)
- Max tokens: Auto
- Access: claude.ai web interface (free tier)
- Date range: January-December 2025

Temporal Validity:
Claude.ai model capabilities evolve. Results specific to claude-sonnet-3-5-20240620. Newer models may perform differently.

Prompt Language:
Thai language used as primary language for Chiang Rai Province context. English prompts also tested with similar results (±3% distance variation).

Trial Procedure:
50 independent trials conducted via separate conversations. Each conversation started fresh (no context carryover). Identical prompt used for all trials. Variations in output due to model sampling (temperature=1.0).

S.16. COMPLETE LOCATION COORDINATES

All 80 locations with coordinates (lat, lon) provided in Supplementary Material S5.

S2 - SUPPLEMENTARY MATERIAL
Route Schedules and Feasibility Verification

S2.1. FEASIBILITY VERIFICATION METHODOLOGY

Constraints:
- Working hours: 8:00 AM - 4:30 PM (8.5 hours = 510 minutes)
- Service time: 30 minutes per location
- Travel speed: 50 km/h
- All locations must fit within daily time limit

Verification Process:
1. Generate minute-by-minute schedule for each route
2. Calculate: Total_time = Travel_time + Service_time
3. Calculate slack: Slack = 510 - Total_time
4. Classify feasibility based on slack

Classification:
- **Robust:** Slack ≥ 30 min AND feasible under ±10 km/h, ±10 min variations
- **Feasible:** Slack ≥ 15 min
- **Marginal:** 0 ≤ Slack < 15 min
- **Infeasible:** Slack < 0 (exceeds time limit)

S2.2. EXAMPLE ROUTE SCHEDULE

Genetic Algorithm - Day 1 (Best Trial):

Stop	Location	Arrival	Service End	Travel (min)	Cumulative
Depot	0	08:00	08:00	-	0:00
1	Loc_005	09:21	09:51	81	1:51
2	Loc_012	10:07	10:37	16	2:37
3	Loc_023	10:55	11:25	18	3:25
4	Loc_031	11:43	12:13	18	4:13
5	Loc_042	12:28	12:58	15	4:58
6	Loc_048	13:14	13:44	16	5:44
7	Loc_055	13:58	14:28	14	6:28
8	Loc_061	14:43	15:13	15	7:13
Depot	0	16:21	-	68	8:21

Result: Total 8:21, Slack 9 min → **MARGINAL**

S2.3. FEASIBILITY SUMMARY (ALL METHODS)

Genetic Algorithm (Best of 50 trials)

Day	Locations	Distance (km)	Time (hrs)	Slack (min)	Classification
1	14	152.3	8:21	9	Marginal
2	13	145.7	8:05	25	Feasible
3	13	138.2	7:48	42	Robust
4	14	149.8	8:15	15	Feasible
5	13	142.6	7:55	35	Robust
6	13	141.8	7:52	38	Robust

Claude.ai (Best of 50 trials)

Day	Locations	Distance (km)	Time (hrs)	Slack (min)	Classification
1	13	158.4	8:18	12	Marginal
2	14	162.3	8:25	5	Marginal

3	13	151.2	8:02	28	Feasible
4	13	147.8	7:54	36	Robust
5	14	159.6	8:22	8	Marginal
6	13	145.3	7:58	32	Robust

Routific

Day	Locations	Distance (km)	Time (hrs)	Slack (min)	Classification
1	13	169.5	8:14	16	Marginal
2	14	175.8	8:24	6	Marginal
3	13	163.2	8:08	22	Feasible
4	14	178.4	8:26	4	Marginal
5	13	166.7	8:12	18	Feasible
6	13	164.5	8:10	20	Feasible

VRP Spreadsheet

Day	Locations	Distance (km)	Time (hrs)	Slack (min)	Classification
1	14	178.2	8:28	2	Marginal
2	13	165.4	8:12	18	Feasible
3	13	171.3	8:20	10	Marginal
4	14	182.6	8:32	-2	Infeasible*
5	13	168.9	8:16	14	Marginal
6	13	167.1	8:14	16	Marginal

S2.4. SENSITIVITY ANALYSIS

Speed Sensitivity (Routes Feasible at Different Speeds)

Speed (km/h)	GA	Claude.ai	Routific	VRP Spreadsheet
40	4/6 (67%)	1/6 (17%)	2/6 (33%)	0/6 (0%)
45	5/6 (83%)	3/6 (50%)	4/6 (67%)	2/6 (33%)
50 (baseline)	6/6 (100%)	6/6 (100%)	6/6 (100%)	5/6 (83%)*
55	6/6 (100%)	6/6 (100%)	6/6 (100%)	6/6 (100%)
60	6/6 (100%)	6/6 (100%)	6/6 (100%)	6/6 (100%)

*After adjustment

Finding: GA most resilient to traffic delays; VRP Spreadsheet most sensitive

Service Time Sensitivity (Routes Feasible at Different Service Times)

Service Time (min)	GA	Claude.ai	Routific	VRP Spreadsheet
20	6/6	6/6	6/6	6/6
25	6/6	6/6	6/6	6/6
30 (baseline)	6/6	6/6	6/6	5/6*
35	5/6	4/6	3/6	1/6
40	3/6	2/6	1/6	0/6

Finding: All methods highly sensitive to service time increases beyond 30 minutes

S2.5. COMPARATIVE SUMMARY

Method	Robust	Feasible	Marginal	Infeasible	Avg Slack
GA	3	2	1	0	27.3 min
Claude.ai	2	1	3	0	20.2 min
Routific	0	3	3	0	14.3 min
VRP Spreadsheet	0	1	4	1*	9.7 min

*After adjustment

Key Findings:

- GA produces most robust routes (highest slack margins)
- Claude.ai moderate robustness (3 marginal routes)
- Routific and VRP Spreadsheet produce tight routes with minimal slack
- All methods require speed ≥45 km/h for reliable feasibility
- Service time must not exceed 35 minutes for most routes

S2.6. OPERATIONAL RECOMMENDATIONS

- Minimum Slack: Maintain ≥20 minute slack per route
- Conservative Speed: Plan for 45 km/h (not 50) to build buffer
- Service Time Buffer: Plan for 35 min average (not 30)
- Method Selection: Use GA for critical services requiring high reliability
- Monitoring: Track actual vs planned times; adjust if delays occur

S3 - SUPPLEMENTARY MATERIAL
Setup Time Measurement Protocol

S3.1. DEFINITION AND SCOPE

Setup Time: Duration from initial access to successful generation of first feasible routing solution.
Includes: Software access, learning interface, data entry, parameter configuration, execution, verification
Excludes: Problem data preparation, solution quality evaluation, ongoing usage time

S3.2. PARTICIPANTS

Group	N	Profile	Rationale
Industrial Engineering Students (undergrad)	3	4th year, OR courses, Excel/programming familiar	Junior technical staff
Admin Staff	3	Age 28-45, Office skills, no technical background	Typical SME staff
Industrial Engineering Students (postgrad)	3	Logistics/IE masters, theoretical knowledge	Mid-level specialists

Total: n=9 participants

S3.3. MEASUREMENT PROTOCOL

Equipment

- Laptop: Dell Latitude 5420, Intel i5, 16GB RAM, Windows 11
- Internet: 100 Mbps fiber
- Software: Pre-installed where applicable (Excel with Solver, Python 3.9)
- Accounts: Pre-created (Routific trial, Claude.ai free tier)

Procedure

Phase 1: Introduction (5 min, NOT counted)

1. Explain VRP problem
2. Show standardized 20-location dataset
3. Provide method documentation
4. Answer clarification questions

Phase 2: Setup and First Solution (TIMED)

1. Timer START: Access/install software
2. Participant works independently
3. Observer records timestamps and issues
4. Timer STOP: Feasible solution verified

Verification Criteria:

1. All locations assigned
2. Time constraints satisfied (≤8.5 hours/day)
3. Routes start/end at depot
4. No duplicates

Learning Curve (10 additional trials)

Each participant repeated task 10 times with different problem instances to measure learning progression.

S3.4. METHOD-SPECIFIC PROTOCOLS

VRP Spreadsheet

- Materials: Template, 2-page user guide, Solver parameter guide
- Typical Steps: Open template → Paste coordinates → Enter parameters → Configure Solver → Execute → Verify
- Common Issues: Finding Solver add-in, interpreting solution output
- Duration: 45 ± 8 minutes

Routific
- Materials: Account credentials, quick start guide
- Typical Steps: Login → Upload data → Set parameters → Optimize → Review map
- Common Issues: CSV format confusion, understanding vehicles=days
- Duration: 30 ± 6 minutes

Genetic Algorithm
- Materials: Python pre-installed, GA script, README, video tutorial
- Typical Steps: Open terminal → Edit config → Install dependencies → Run script → Verify output
- Common Issues: Command-line unfamiliarity, environment issues, no visualization
- Duration: 180 ± 35 minutes

Claude.ai
- Materials: Account, example prompt template
- Typical Steps: Navigate to claude.ai → Login → Paste data in prompt → Send → Review response
- Common Issues: Initial prompt too vague, interpreting natural language output
- Duration: 12 ± 3 minutes

S3.5. RESULTS

Setup Time by Method

Method	Mean (min)	SD (min)	Median	95% CI	Range
Claude.ai	12.3	3.1	12.0	[10.1, 14.5]	8-18
Routific	30.4	6.2	29.5	[26.1, 34.7]	22-38
VRP Spreadsheet	45.2	8.4	44.0	[39.3, 51.1]	32-58
GA	179.6	34.8	175.0	[155.4, 203.8]	120-245

Statistical Test: Kruskal-Wallis H=26.7, p<0.001 (all methods significantly different)

Setup Time by Participant Group

Group	VRP Spreadsheet	Routific	GA	Claude.ai
Industrial Engineering Students (undergrad)	42 ± 7	29 ± 6	165 ± 28	13 ± 3
Admin Staff	52 ± 9	35 ± 6	215 ± 35*	10 ± 2
Industrial Engineering Students (postgrad)	42 ± 8	28 ± 5	158 ± 30	14 ± 4

*One participant abandoned at 240 min

ANOVA Results:
- GA: F(2,6)=8.71, p=0.018 (Admin staff struggled significantly)
- Others: No significant group differences (p>0.10)

Conclusion: Technical background matters for GA; not for other methods

S3.6. LEARNING CURVES

Time Reduction (Trial 1 → Trial 10)

Method	Trial 1	Trial 10	Reduction	%
Claude.ai	12.3	7.8	4.5	37%
Routific	30.4	17.6	12.8	42%
VRP Spreadsheet	45.2	24.7	20.5	45%
GA	179.6	119.8	59.8	33%

Observations:
- All methods: 33-45% reduction over 10 trials

- Plateau typically at Trial 5-7
- Relative ordering preserved (Claude.ai fastest even after GA learning)

S3.7. STATISTICAL ANALYSIS
Descriptive Statistics: Mean, SD, Median, Range, 95% CI
Group Comparisons: One-way ANOVA per method (test if setup time differs by participant group)
Method Comparisons: Kruskal-Wallis (non-parametric test across methods)
Post-hoc: Dunn's test for pairwise comparisons (all significant at $p<0.001$)

S3.8. VALIDITY CONSIDERATIONS
Internal Validity:
- Controlled: Same problem, equipment, protocol, observer
- Threats: Potential Hawthorne effect (acknowledged)

External Validity:
- Small sample (n=9)
- Single problem size (20 locations)
- Thai context
- Limitations acknowledged

Reliability:
- Single observer (no inter-rater variation)
- Screen recordings available for verification
- Learning curve shows consistency ($r>0.85$ for trials 7-10)

S4 - SUPPLEMENTARY MATERIAL
Genetic Algorithm Implementation

S4.1. ALGORITHM CONFIGURATION

Parameter	Value	Rationale
Population Size	200	Balance quality/runtime
Max Generations	1,000	Sufficient for convergence
Convergence	50 gen w/o improvement	Early stopping
Selection	Tournament (k=5)	Moderate selection pressure
Crossover	Order Crossover, Pc=0.80	Preserves relative order
Mutation	Swap (Pm=0.10), Inversion (Pm=0.05)	Local + structural search
Elitism	Top 10%	Preserve best solutions

Runtime: 45-60 seconds on standard laptop (Intel i5, 16GB RAM)

S.4.2. SOLUTION ENCODING
Representation: Permutation with depot separators
Format: `[L1, L2, ..., Lm, -1, Lm+1, ..., Ln, -1, ...]`
- L_i = Customer location ID (1 to 80)
- -1 = Depot separator (marks end of day)
- Number of separators = Days - 1

Example (20 locations, 3 days):

[5, 12, 18, 7, -1, 3, 15, 20, 9, 11, 1, -1, 2, 6, 8, 13, 14, 16, 17, 19]

Decoded:
Day 1: Depot → 5 → 12 → 18 → 7 → Depot
Day 2: Depot → 3 → 15 → 20 → 9 → 11 → 1 → Depot
Day 3: Depot → 2 → 6 → 8 → 13 → 14 → 16 → 17 → 19 → Depot

S4.3. FITNESS FUNCTION

$$f(x) = \text{Total\_Distance} + M \times \text{Penalty}$$

Where:
- Total_Distance = Σ all route distances
- M = 10,000 (penalty coefficient)
- Penalty = Σ max(0, Time_day - 510) for all days
- Time_day = (Distance/50)*60 + Locations*30 minutes

Constraint Handling: Penalty-based fitness with repair heuristic for infeasible solutions

S4.4. GENETIC OPERATORS
Initialization
- 80% random permutations with random separator placement
- 20% nearest neighbor greedy solutions (for better initial population)

Selection: Tournament (k=5)

```
def tournament_selection(population, fitness_scores, k=5):
    tournament = random.sample(range(len(population)), k)
    best_idx = min(tournament, key=lambda i: fitness_scores[i])
    return population[best_idx]
```

Crossover: Order Crossover (Pc=0.80)

```
def order_crossover(parent1, parent2):
    # Remove separators temporarily
    p1_customers = [x for x in parent1 if x != -1]
```

```
    p2_customers = [x for x in parent2 if x != -1]

    # Select random segment from parent1
    start, end = sorted(random.sample(range(len(p1_customers)), 2))

    # Copy segment to offspring
    offspring = [None] * len(p1_customers)
    offspring[start:end] = p1_customers[start:end]

    # Fill remaining from parent2 order
    p2_idx = 0
    for i in range(len(offspring)):
        if offspring[i] is None:
            while p2_customers[p2_idx] in offspring:
                p2_idx += 1
            offspring[i] = p2_customers[p2_idx]

    # Reinsert separators
    return reinsert_separators(offspring, parent1)
```

Mutation: Swap (Pm=0.10)

```
def swap_mutation(solution, pm=0.10):
    if random.random() > pm:
        return solution

    customer_positions = [i for i, x in enumerate(solution) if x != -1]
    if len(customer_positions) >= 2:
        pos1, pos2 = random.sample(customer_positions, 2)
        solution[pos1], solution[pos2] = solution[pos2], solution[pos1]

    return solution
```

Mutation: Inversion (Pm=0.05)

```
def inversion_mutation(solution, pm=0.05):
    if random.random() > pm:
        return solution

    customer_positions = [i for i, x in enumerate(solution) if x != -1]
    if len(customer_positions) >= 2:
        start, end = sorted(random.sample(customer_positions, 2))
        segment = [solution[i] for i in range(start, end+1) if solution[i] != -1]
        segment.reverse()
        # Place reversed segment back
        j = 0
        for i in range(start, end+1):
            if solution[i] != -1:
                solution[i] = segment[j]
                j += 1

    return solution
```

Repair Heuristic

If route exceeds 510 minutes:

1. Find most overloaded day
2. Remove last customer
3. Add to least loaded day (if fits)

4. Repeat until all days feasible

S4.5. COMPLETE PSEUDOCODE

```
ALGORITHM: GA for Multi-day VRP

INPUT: locations, depot, n_days=6, pop_size=200, max_gen=1000

BEGIN
  1. Initialize population (80% random, 20% greedy)
  2. Evaluate fitness for all individuals
  3. best_solution ← best individual

  4. FOR generation = 1 TO max_gen:
     a. Preserve elite (top 10%)
     b. WHILE new_population < pop_size:
        - Select two parents (tournament k=5)
        - Crossover with probability 0.80
        - Apply swap mutation (Pm=0.10)
        - Apply inversion mutation (Pm=0.05)
        - Repair if infeasible
        - Add to new_population
     c. Evaluate new_population
     d. Update best_solution if improved
     e. IF no improvement for 50 generations: BREAK

  5. RETURN best_solution
END
```

S4.6. PYTHON IMPLEMENTATION (Key Components)

```
import numpy as np
import random

class VRP_GA:
    def __init__(self, locations, depot, n_days=6, pop_size=200):
        self.locations = locations
        self.depot = depot
        self.n_days = n_days
        self.pop_size = pop_size
        self.max_gen = 1000
        self.convergence = 50

    def haversine_distance(self, loc1, loc2):
        """Calculate distance between two lat/lon points"""
        lat1, lon1 = loc1
        lat2, lon2 = loc2
        R = 6371  # Earth radius in km

        dlat = np.radians(lat2 - lat1)
        dlon = np.radians(lon2 - lon1)
        a = np.sin(dlat/2)**2 + np.cos(np.radians(lat1)) * \
            np.cos(np.radians(lat2)) * np.sin(dlon/2)**2
        c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))

        return R * c
```

```
def fitness(self, solution):
    """Calculate fitness with penalty for constraint violations"""
    days = self.split_into_days(solution)
    total_distance = 0
    total_penalty = 0

    for day_route in days:
        day_distance = 0
        if day_route:
            # Depot to first location
            day_distance += self.haversine_distance(
                self.depot, self.locations[day_route[0]])

            # Between locations
            for i in range(len(day_route) - 1):
                day_distance += self.haversine_distance(
                    self.locations[day_route[i]],
                    self.locations[day_route[i+1]])

            # Last location to depot
            day_distance += self.haversine_distance(
                self.locations[day_route[-1]], self.depot)

            # Calculate time
            day_time = (day_distance / 50) * 60 + len(day_route) * 30

            # Penalty if exceeds 510 minutes
            if day_time > 510:
                total_penalty += (day_time - 510)

            total_distance += day_distance

    return total_distance + 10000 * total_penalty

def run(self, seed=None):
    """Execute GA"""
    if seed:
        random.seed(seed)

    # Initialize
    population = self.initialize_population()
    fitness_scores = [self.fitness(ind) for ind in population]
    best = population[np.argmin(fitness_scores)].copy()

    no_improvement = 0

    # Evolve
    for gen in range(self.max_gen):
        # Elite preservation
        elite_idx = np.argsort(fitness_scores)[:int(0.1*self.pop_size)]
        new_pop = [population[i].copy() for i in elite_idx]

        # Generate offspring
        while len(new_pop) < self.pop_size:
            p1 = self.tournament_selection(population, fitness_scores)
```

```
            p2 = self.tournament_selection(population, fitness_scores)

            offspring = self.order_crossover(p1, p2) \
                if random.random() < 0.80 else p1.copy()

            offspring = self.swap_mutation(offspring)
            offspring = self.inversion_mutation(offspring)
            offspring = self.repair_solution(offspring)

            new_pop.append(offspring)

        # Update
        population = new_pop
        fitness_scores = [self.fitness(ind) for ind in population]

        if min(fitness_scores) < self.fitness(best):
            best = population[np.argmin(fitness_scores)].copy()
            no_improvement = 0
        else:
            no_improvement += 1

        if no_improvement >= self.convergence:
            break

    return best, self.fitness(best)

# Usage:
# ga = VRP_GA(customer_locations, depot)
# solution, distance = ga.run(seed=42)
```

S4.7. COMPUTATIONAL COMPLEXITY

Per Generation:
- Fitness evaluation: O(pop_size × n × n_days) = O(96,000)
- Selection: O(pop_size × k) = O(1,000)
- Crossover/Mutation: O(pop_size × n) = O(16,000)

Total: O(100,000) operations/generation × 650 average generations = O(65 million) operations

Runtime: 45-60 seconds

S4.8. PARAMETER SENSITIVITY

Pop Size	Distance (km)	Runtime (sec)
50	925.3	18
100	915.7	32
200	908.3	58
400	906.1	125

Recommendation: 200 (good quality/runtime balance)

S5 - SUPPLEMENTARY MATERIAL
Complete Dataset and Results

S5.1. LOCATION COORDINATES

Depot

ID: 0
Name: Chiang Mai Operations Center
Coordinates: 18.7883°N, 98.9853°E

Customer Locations (80 AED Maintenance Sites)
Format: ID, Name, Latitude, Longitude, District

1,Chiang Rai Provincial Hospital,19.9105,99.8318,Mueang Chiang Rai
2,Mae Fah Luang University,19.8891,99.8156,Mueang Chiang Rai
3,Overbrook Hospital,19.9234,99.8542,Mueang Chiang Rai
4,Chiang Rai Ram Hospital,19.9087,99.8201,Mueang Chiang Rai
5,Wat Phra Kaew,19.9145,99.8278,Mueang Chiang Rai
...
[76 more locations]
...
80,Technical College,19.8956,99.8467,Mueang Chiang Rai

Complete List: All 80 coordinates available in `locations.csv`

S5.2. DISTANCE MATRIX

Format: 81×81 symmetric matrix (depot + 80 locations), Haversine distances in km
Calculation:

$d = 2R \times \arcsin(\sqrt{(\sin^2(\Delta\varphi/2) + \cos(\varphi_1)\cos(\varphi_2)\sin^2(\Delta\lambda/2)}))$
where R = 6,371 km

Validation:

- Correlation with Google Maps: r=0.932, MAPE=8.7%
- Correction factor: 1.12× (Haversine systematically underestimates road distance)

File: `haversine_distance_matrix.csv` (81×81 matrix)

S5.3. RESULTS FROM ALL METHODS

Genetic Algorithm (50 trials)

Trial,Seed,Distance_km,Feasible,Runtime_sec
1,42,908.34,Yes,58
2,123,912.67,Yes,55
3,456,915.23,Yes,59
...
50,9999,918.67,Yes,56

Summary:

- Mean: 915.2 ± 8.7 km
- Median: 914.5 km
- Best: 908.34 km (Trial 1)
- Worst: 928.45 km
- 95% CI: [912.8, 917.6]

File: `ga_results_50_trials.csv`

Claude.ai (50 trials)

Trial,Conversation_ID,Distance_km,Feasible,Response_Time_sec
1,conv_abc123,941.64,Yes,8.2
2,conv_def456,945.12,Yes,7.9
...
50,conv_xyz999,947.89,Yes,8.7

Summary:
- Mean: 946.8 ± 7.3 km
- Median: 946.2 km
- Best: 941.64 km (Trial 1)
- Worst: 961.23 km
- 95% CI: [944.7, 948.9]

File: `claude_results_50_trials.csv`

VRP Spreadsheet (Deterministic, verified 10 trials)
Result: 1089.45 km (identical across all trials)
File: `vrp_spreadsheet_verification.csv`

Routific (Deterministic, verified 10 trials)
Result: 1032.18 km (identical across all trials)
File: `routific_verification.csv`

Manual Baseline
Result: 1260.0 km (current practice, simple geographic clustering)
File: `manual_baseline.csv`

S5.4. STATISTICAL ANALYSIS
Descriptive Statistics

Method,N,Mean_km,SD_km,Median_km,Min_km,Max_km,CV_%,95%_CI_Lower,95%_CI_Upper
GA,50,915.2,8.7,914.5,908.34,928.45,0.95,912.8,917.6
Claude.ai,50,946.8,7.3,946.2,941.64,961.23,0.77,944.7,948.9
VRP_Spreadsheet,10,1089.45,0.0,1089.45,1089.45,1089.45,0.00,1089.45,1089.45
Routific,10,1032.18,0.0,1032.18,1032.18,1032.18,0.00,1032.18,1032.18
Manual,1,1260.0,NA,1260.0,1260.0,1260.0,NA,NA,NA

ANOVA

Source	SS	df	MS	F	p
Between Groups	2,847,235	3	949,078	834.2	<0.001
Within Groups	123,456	108	1,143		
Total	2,970,691	111			

Conclusion: Significant differences exist (p<0.001)
Pairwise Comparisons (Tukey HSD)

Comparison,Mean_Diff_km,p_value,Significant
GA vs Claude.ai,-31.6,<0.001,Yes
GA vs Routific,-116.98,<0.001,Yes
GA vs VRP_Spreadsheet,-174.25,<0.001,Yes
Claude.ai vs Routific,-85.38,<0.001,Yes
Claude.ai vs VRP_Spreadsheet,-142.65,<0.001,Yes
Routific vs VRP_Spreadsheet,-57.27,<0.001,Yes

All pairwise comparisons significant at p<0.001
File: `statistical_analysis.csv`

S5.5. COST-BENEFIT ANALYSIS
Parameters

Fuel: 5.00 THB/km (40 THB/L ÷ 8 km/L)
Maintenance: 2.00 THB/km
Labor: 300 THB/hour (travel time only)
Total_Variable_Cost: 7.00 THB/km (baseline)
Annual_Cycles: 52

Annual Savings

```
Method,Distance_km,Reduction_km,Reduction_%,Annual_Savings_THB
GA,915.2,344.8,27.4,125,125
Claude.ai,946.8,313.2,24.9,113,641
Routific,1032.18,227.82,18.1,82,701
VRP_Spreadsheet,1089.45,170.55,13.5,61,920
Manual,1260.0,0,0.0,0
```

File: `cost_calculations.csv`

S5.6. SETUP TIME DATA

Raw Measurements

```
Participant_ID,Group,Method,Setup_Time_min,Successful
P01,IE_Student,GA,165,Yes
P02,IE_Student,Routific,28,Yes
P03,IE_Student,VRP_Spreadsheet,42,Yes
P04,Admin_Staff,Claude.ai,10,Yes
P05,Admin_Staff,VRP_Spreadsheet,52,Yes
P06,Admin_Staff,GA,245,No
P07,Grad_Student,Claude.ai,14,Yes
P08,Grad_Student,Routific,27,Yes
P09,Grad_Student,GA,158,Yes
```

File: `setup_time_raw_data.csv`

Learning Curves

```
Participant_ID,Method,Trial_1,Trial_2,...,Trial_10
P01,GA,165,142,128,...,99
P02,Routific,28,24,21,...,17
...
```

File: `learning_curve_data.csv`

S5.7. DATA FILES SUMMARY

All data available in CSV format (UTF-8 encoding):

- `locations.csv` - 80 customer coordinates
- `haversine_distance_matrix.csv` - 81×81 distance matrix
- `ga_results_50_trials.csv` - GA detailed results
- `claude_results_50_trials.csv` - Claude.ai detailed results
- `vrp_spreadsheet_verification.csv` - VRP Spreadsheet verification
- `routific_verification.csv` - Routific verification
- `manual_baseline.csv` - Current practice routes
- `statistical_analysis.csv` - ANOVA, post-hoc tests
- `cost_calculations.csv` - Cost-benefit analysis
- `setup_time_raw_data.csv` - Setup time measurements
- `learning_curve_data.csv` - Learning progressions

Total Size: ~2.5 MB compressed

S5.8. REPRODUCIBILITY

Using Genetic Algorithm

```
pip install numpy pandas
python vrp_ga.py --locations locations.csv --days 6 --trials 50
```

Using VRP Spreadsheet

- Open `VRP_Spreadsheet_Template.xlsx`
- Import `locations.csv`
- Set parameters: Days=6, TimeLimit=8.5, ServiceTime=30, Speed=50
- Data → Solver → Solve

Using Routific

- Login to routific.com
- Upload `locations.csv`
- Configure: 6 vehicles, 8.5 hour shifts, 30 min service
- Optimize

Using Claude.ai
- Navigate to claude.ai
- Use prompt template from Supplementary S1
- Insert coordinates from `locations.csv`
- Parse response using `parse_claude_output.py` (S1)