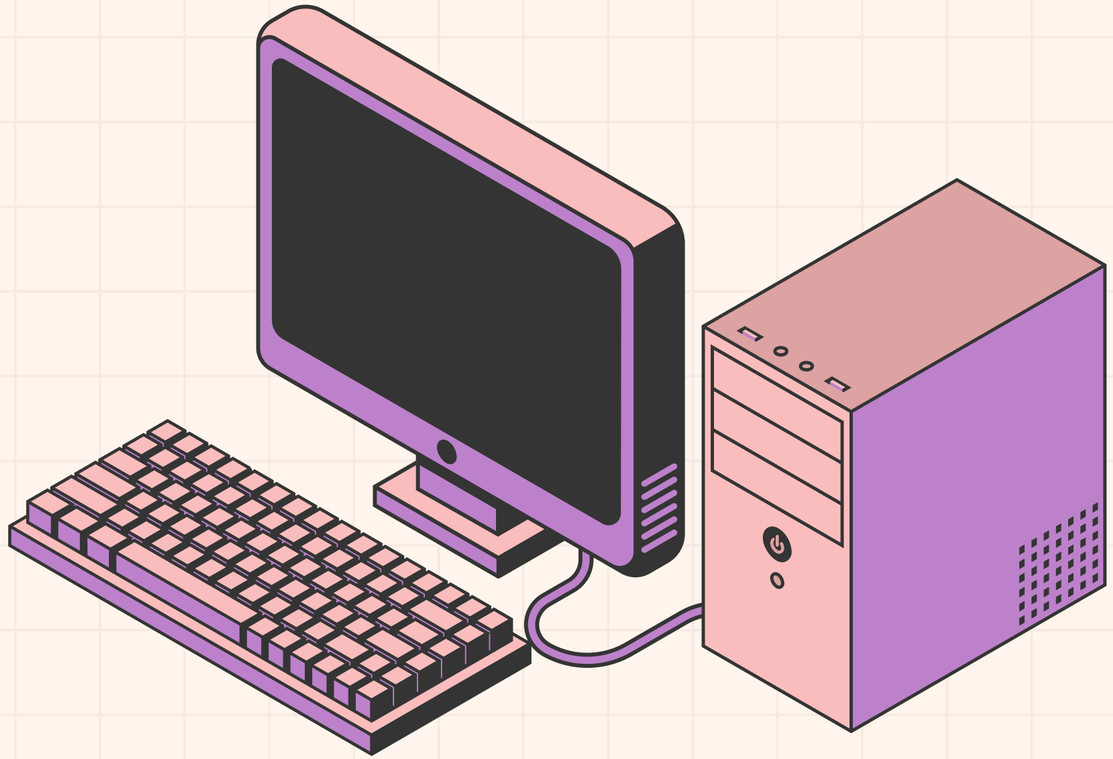
VRP

HOME SERVICE ABOUT US CONTACT US

VEHICLE
ROUTING SYSTEM

Group 8



MEMBERS

Nguyen Van Quoc

(Leader, Dev)

- Project and Progress Management
- MRA Development
- DAs Development
- Base Development
- All Optimizations
- Review and Testing system

Tran Hung Quoc Tuan

(Developer)

- GA Optimization (Calculate Fitness)
- GA + OR (no modified fitness) Optimization
- GA + OR (modified fitness) Optimization
- Testing System
- Maintain the system

Nguyen Ha Minh Chau

(QA, Testing, Documentation)

- Write test cases
- Documentations
- Weekly Report
- Tracking works for team

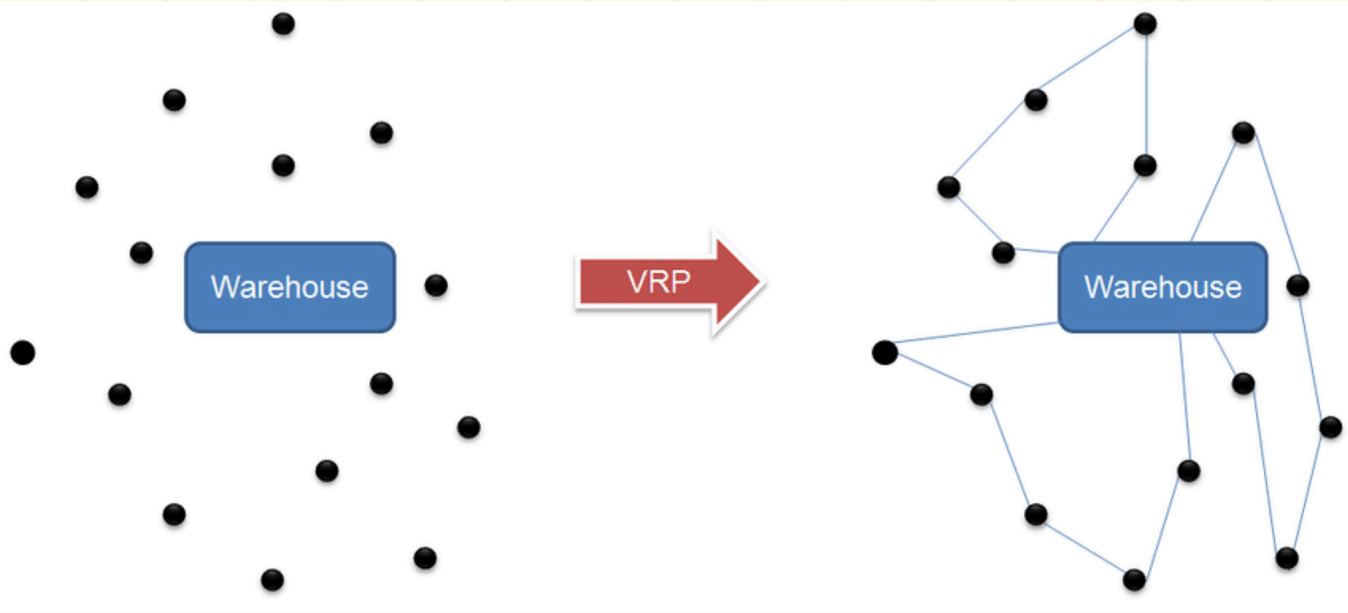
Mai Hoang Dai Vy

(Developer, Documentation)

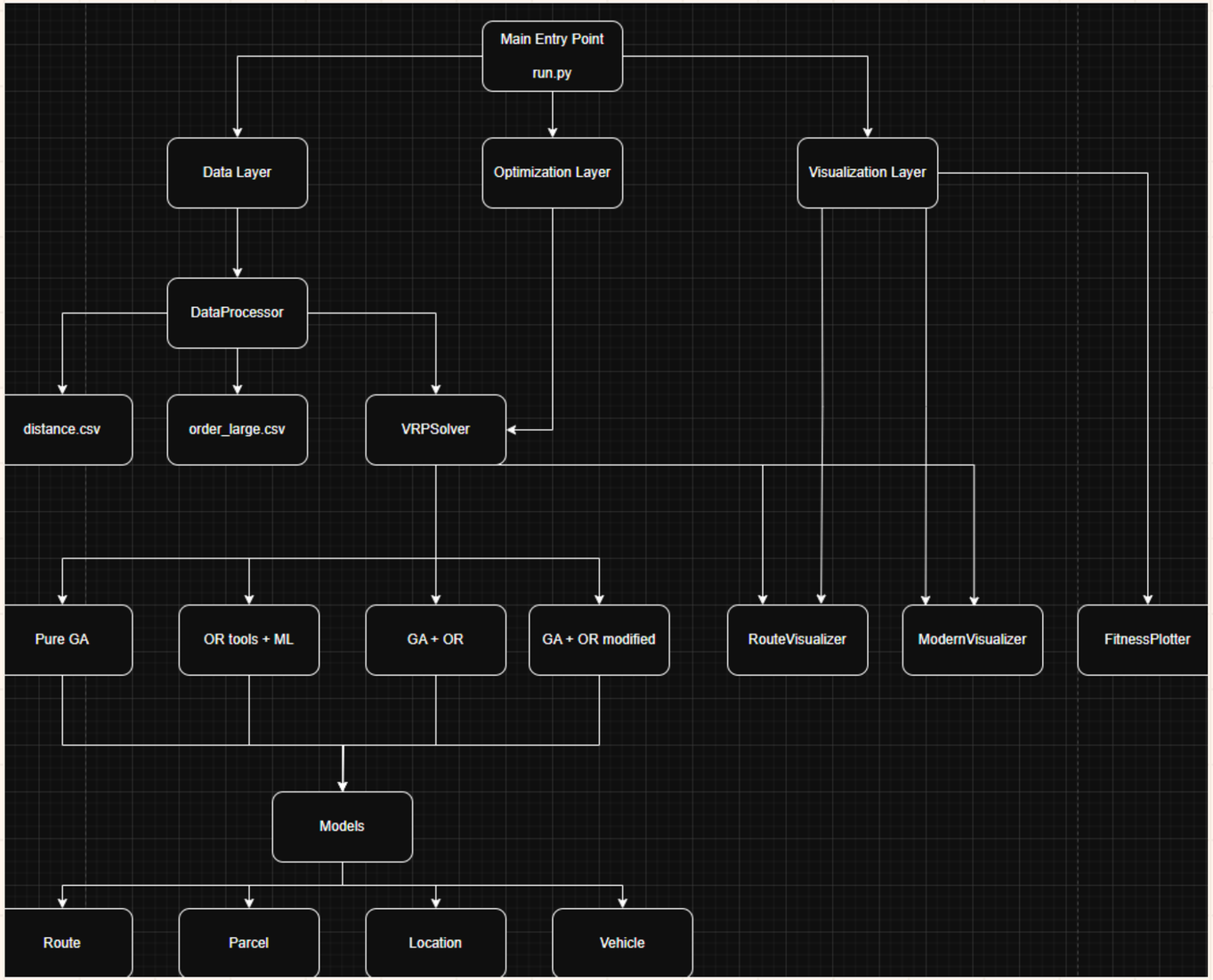
- Data Research
- GUI Development
- Documentations

WHAT IS VRP ?

- Finding **optimal** routes for a fleet of vehicles to deliver goods
- **Minimizing** total cost while satisfying all constraints
- Constraints include:
  - Vehicle capacity
  - Time windows
  - Driver availability

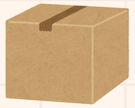


SYSTEM ARCHITECHTURE



SYSTEM INFRASTRUCTURE

CORECOMPONENT



MODELS

- Location
- Parcel
- Route
- Vehicle

PROTOCOLS

- Message Protocol
- Agent Protocol
- Message Queue

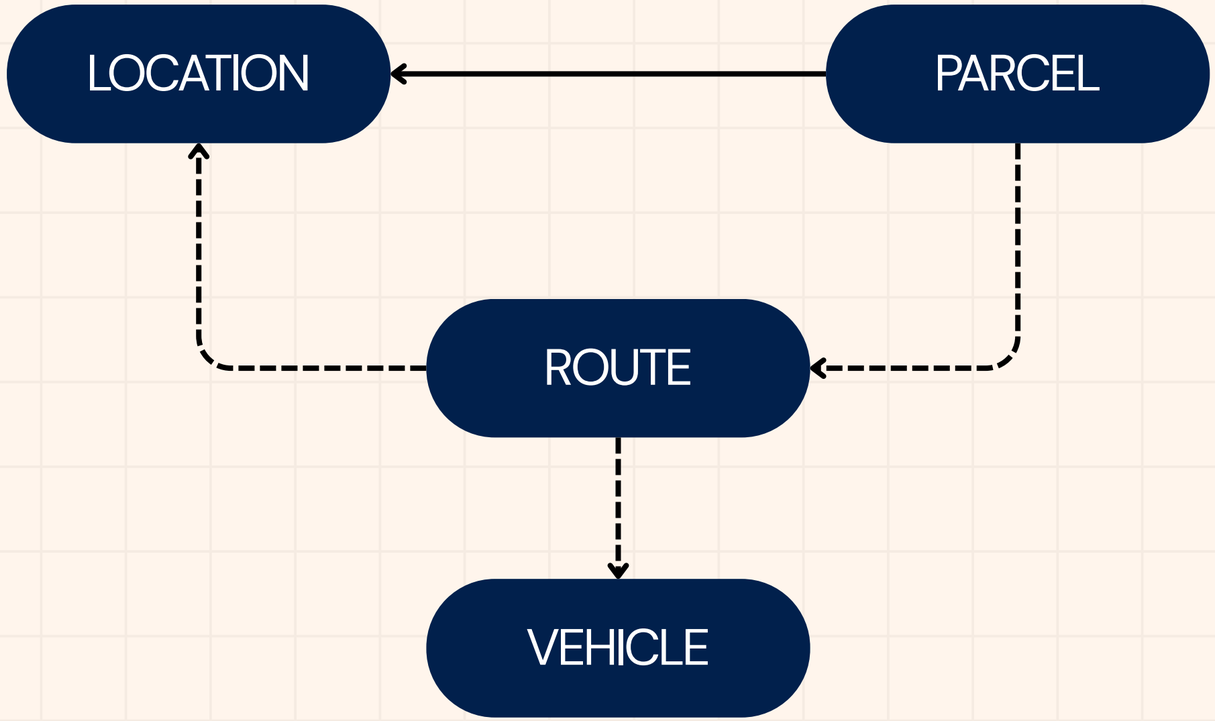
UTILS

- Performance
- Memory Metrics
- Distance Calc

VISUALIZATION

- RouteVisualize
- ModernVisualize
- Fitnness Plotter

MODEL PACKAGE



AGENT-BASED ARCHITECTURE

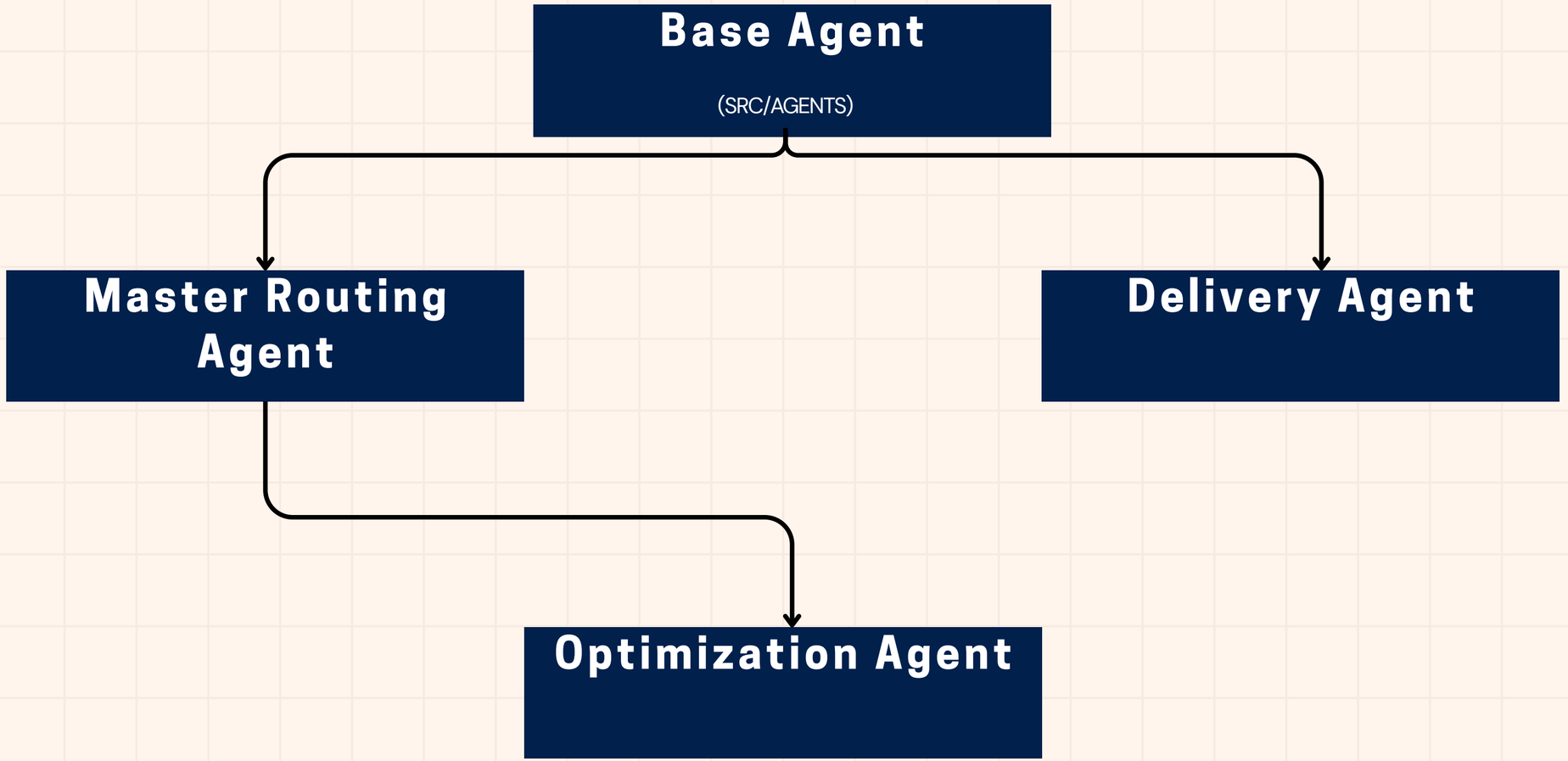
MULTI-AGENT SYSTEM FOR VRP

- Decentralized decision-making
- Enhanced modularity & flexibility
- Parallel processing capabilities
- Improved fault tolerance
- Easier adaptation to dynamic scenarios

Key Components:

- Agent Protocol for communication
- Message-based interaction system
- Specialized agent roles

AGENT TYPES AND HIERARCHY



BaseAgent Implementation

```python
from abc import ABC, abstractmethod
from typing import Dict, Any, Optional
from src.protocols.message_protocol import Message

class BaseAgent(ABC):
    def __init__(self, agent_id: str):
        self.agent_id = agent_id
        self.state: Dict[str, Any] = {}

    @abstractmethod
    def process_message(self, message: Message) -> Optional[Message]:
        pass
```

- Abstract base class for all agent types
- Defines core agent functionality
- Message processing interface
- Internal state management

DeliveryAgent Implementation

```python
class DeliveryAgent(BaseAgent):
    def __init__(self, agent_id: str, capacity: float, max_distance: float):
        super().__init__(agent_id)
        self.capacity = capacity
        self.max_distance = max_distance
        self.current_route = None
        self.message_handler = self._setup_handlers()
```

```python
def process_message(self, message: Message) -> Optional[Message]:
    if message.msg_type in self.message_handler:
        return self.message_handler[message.msg_type](message)
    return None
```

- Represents individual delivery vehicles
- Handles capacity and distance constraints
- Processes route assignments
- Validates routes before accepting

MasterRoutingAgent Implementation

```python
class MasterRoutingAgent(BaseAgent):
    def __init__(self, agent_id: str):
        super().__init__(agent_id)
        self.delivery_agents = {}
        self.message_handler = {
            MessageType.CAPACITY_RESPONSE: self._handle_capacity_response,
            MessageType.ROUTE_CONFIRMATION: self._handle_route_confirmation,
            MessageType.OPTIMIZATION_RESPONSE: self._handle_optimization_response,
            MessageType.STATUS_UPDATE: self._handle_status_update
        }

    def set_optimization_agent(self, agent_id: str):
        """Register the optimization agent"""
        self.optimization_agent_id = agent_id
```

- Coordinates routing across the system
- Manages delivery agent fleet
- Requests optimization from specialists
- Assigns routes to delivery agents

Message Protocol Integration

```python
class MessageType(Enum):
    CAPACITY_REQUEST = "CAPACITY_REQUEST"
    CAPACITY_RESPONSE = "CAPACITY_RESPONSE"
    ROUTE_ASSIGNMENT = "ROUTE_ASSIGNMENT"
    ROUTE_CONFIRMATION = "ROUTE_CONFIRMATION"
    STATUS_UPDATE = "STATUS_UPDATE"
    ERROR = "ERROR"


    OPTIMIZATION_REQUEST = "OPTIMIZATION_REQUEST"
    OPTIMIZATION_RESPONSE = "OPTIMIZATION_RESPONSE"
    OPTIMIZATION_STATUS = "OPTIMIZATION_STATUS"
    PARAMETERS_UPDATE = "PARAMETERS_UPDATE"

@dataclass
class Message:
    msg_type: MessageType
    sender_id: str
    receiver_id: str
    content: Dict[str, Any]
    conversation_id: Optional[str] = None
    timestamp: float = time.time()
```

- Standardized message format
- Type-based message routing
- Content dictionary for flexible payloads
- Conversation tracking for multi-step processes

Message Protocol Integration

```python
class CommunicationManager:
    def __init__(self, data_processor: DataProcessor):
        self.message_tracker = MessageTimeTracker()
        self.queue_tracker = QueueRateTracker()
        self.memory_tracker = MemoryTracker()

        # Initialize agents dictionary first
        self.agents = {}
        self.message_queue = MessageQueue()
        self._running = False

        # Only create optimizer if data_processor is not None
        self.optimizer = None
        self.master_agent = None

        if data_processor is not None:
            self.optimizer = ORToolsMLOptimizer(data_processor)
            self.master_agent = MasterRoutingAgent("MASTER")
            self.register_agent(self.master_agent)
```

```python
def process_message(self, message):
    """Process a single message"""
    start_time = self.message_tracker.start_tracking()

    #Take memory snapshot
    self.memory_tracker.take_snapshot()

    #Process message
    receiver = self.agents[message.receiver_id]
    response = receiver.process_message(message)

    #Record metrics
    self.message_tracker.stop_tracking(start_time, message.msg_type.value)
    self.queue_tracker.record_message()

    return response
```

- Central messaging backbone
- Tracks message performance
- Monitors memory and queue metrics
- Registers and manages agent instances
- Routes messages between agents

DATA MANAGEMENT

ORDER_LARGE

	A	B	C	D	E	F	G	H	I	J	K
1	Order_ID	Material_ID	Item_ID	Source	Destination	Available_Tim	Deadline	Danger_Type	Area	Weight	
2	A140109	B-6128	P01-79c46a0	City_61	City_54	##########	##########	type_1	38880	30920000	
3	A140109	B-6128	P01-43f08b0f	City_61	City_54	##########	##########	type_1	38880	30920000	
4	A140109	B-6128	P01-899d738	City_61	City_54	##########	##########	type_1	38880	30920000	
5	A140109	B-6128	P01-acc23cd	City_61	City_54	##########	##########	type_1	38880	30920000	
6	A140109	B-6128	P01-cd0377d	City_61	City_54	##########	##########	type_1	38880	30920000	
7	A140109	B-6128	P01-ba00d24	City_61	City_54	##########	##########	type_1	38880	30920000	
8	A140109	B-6128	P01-6994ea5	City_61	City_54	##########	##########	type_1	38880	30920000	
9	A140109	B-6128	P01-d06ba80	City_61	City_54	##########	##########	type_1	38880	30920000	
10	A140109	B-6128	P01-06c3fc32	City_61	City_54	##########	##########	type_1	38880	30920000	
11	A140110	B-6128	P01-69f2a2b9	City_61	City_54	##########	##########	type_1	38880	30920000	
12	A140110	B-6128	P01-1fc56a04	City_61	City_54	##########	##########	type_1	38880	30920000	
13	A140110	B-6128	P01-ce8e41e	City_61	City_54	##########	##########	type_1	38880	30920000	
14	A140110	B-6128	P01-e0ff5073	City_61	City_54	##########	##########	type_1	38880	30920000	
15	A140110	B-6128	P01-0cd66c2	City_61	City_54	##########	##########	type_1	38880	30920000	
16	A140110	B-6128	P01-afb96755	City_61	City_54	##########	##########	type_1	38880	30920000	
17	A140110	B-6128	P01-9901aba	City_61	City_54	##########	##########	type_1	38880	30920000	
18	A140110	B-6128	P01-32ea426	City_61	City_54	##########	##########	type_1	38880	30920000	
19	A140110	B-6128	P01-8819dc2	City_61	City_54	##########	##########	type_1	38880	30920000	
20	A140112	B-6128	P01-9046e25	City_61	City_54	##########	##########	type_1	38880	30920000	
21	A140112	B-6128	P01-da3626c	City_61	City_54	##########	##########	type_1	38880	30920000	
22	A140112	B-6128	P01-6b71ea7	City_61	City_54	##########	##########	type_1	38880	30920000	
23	A140112	B-6128	P01-84ac394	City_61	City_54	##########	##########	type_1	38880	30920000	
24	A140112	B-6128	P01-17b9602	City_61	City_54	##########	##########	type_1	38880	30920000	

order_large

DISTANCE

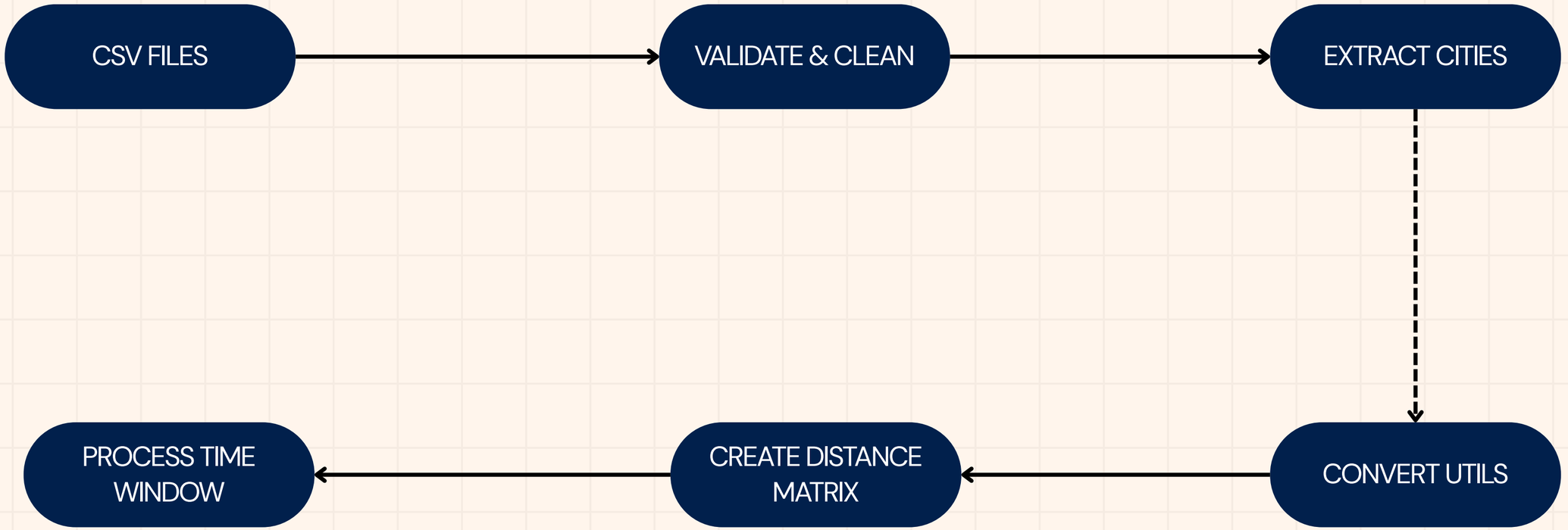
	A	B	C	D
1	Source	Destination	Distance(M)	
2	City_24	City_47	1114251	
3	City_24	City_31	97187	
4	City_24	City_54	1716028	
5	City_24	City_53	1729925	
6	City_24	City_19	1594107	
7	City_24	City_12	774894	
8	City_24	City_46	1146028	
9	City_24	City_45	1147045	
10	City_24	City_51	1107377	
11	City_24	City_6	1688216	
12	City_24	City_20	1615472	
13	City_24	City_56	1636630	
14	City_24	City_35	1053903	
15	City_24	City_48	1300876	

distance

TRUCK SPECIFICATION

Truck Type (length in m)	Inner Size (m^2)	Weight Capacity (kg)	Cost Per KM	Speed (km/h)
16.5	16.1×2.5	10000	3	40
12.5	12.1×2.5	5000	2	40
9.6	9.1×2.3	2000	1	40

DATA PROCESSOR



OPTIMIZATION

OPTIMIZATION METHOD

Genetic Algorithm

- Evolution-based
- Population of route solution

OR-Tools + ML

- Google’s solver
- Enhance with Machine Learning

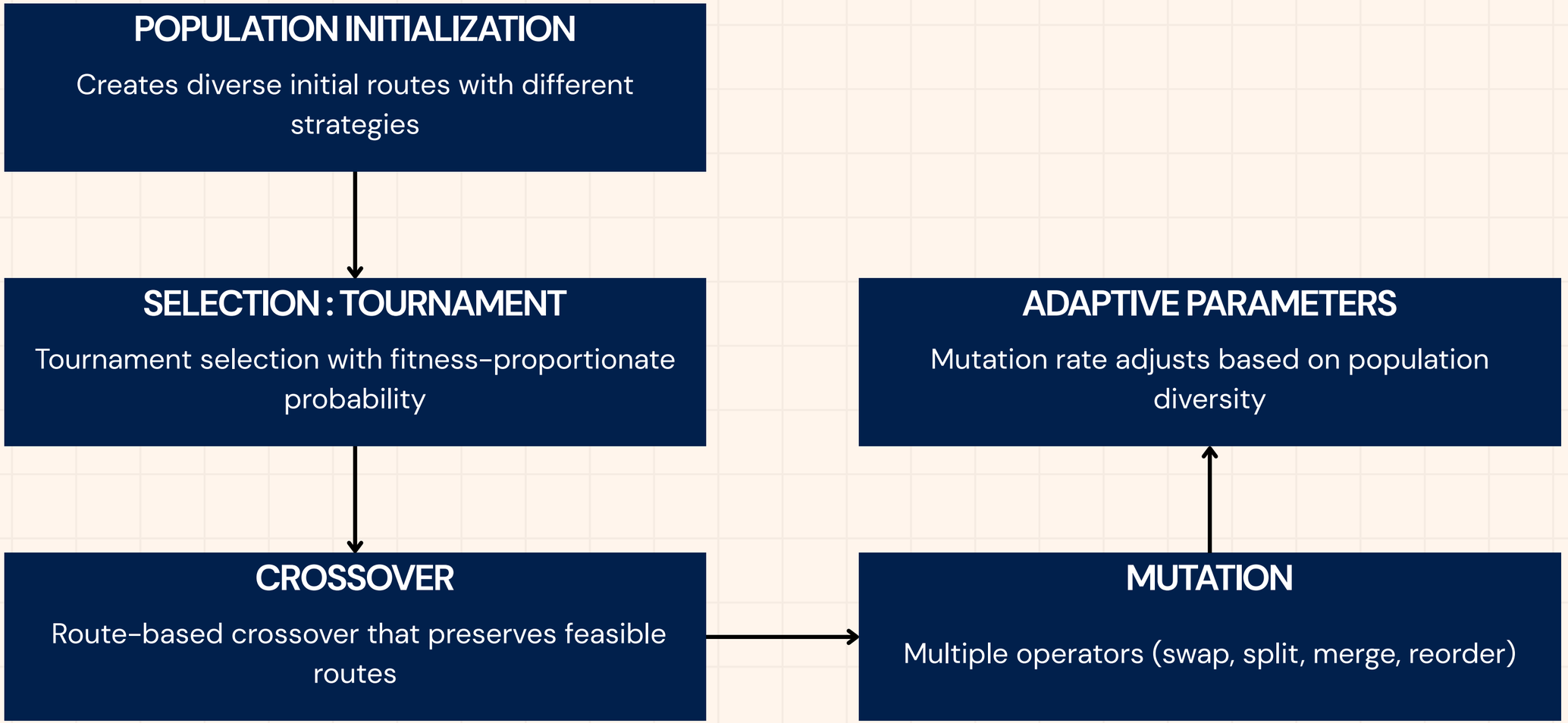
GA + OR-Tools

- Hybrid approach
- OR-Tools init
- GA evolution

GA + OR-Tools (modified fitness)

- Pattern Analysis
- Adapted fitness

GENETIC ALGORITHM



VRP

GENETIC ALGORITHM CALCULATION

```python
def calculate_fitness(self, solution: List[Route]) -> float:

    Calculate fitness score for a solution with improved differentiation.
    """
    if not solution:
        return 0.0

    # Get solution metrics
    evaluation = self.evaluate_solution(solution)
    total_parcels = evaluation['parcels_delivered']
    total_cost = evaluation['total_cost']
    total_distance = sum(route.total_distance for route in solution)

    # Calculate normalized metrics with non-linear scaling
    if total_parcels > 0:
        # Exponential reward for more parcels delivered
        parcels_score = (total_parcels / len(self.data_processor.time_windows)) ** 1.2

        # Progressive penalty for cost with more granular scaling
        cost_per_parcel = total_cost / total_parcels
        max_acceptable_cost = 5000  # Example threshold
        cost_score = 1.0 - min(1.0, (cost_per_parcel / max_acceptable_cost) ** 0.6)

        # Route efficiency with smoother scaling
        optimal_routes = max(1, len(self.data_processor.time_windows) // 5)  # Assume average 5 parcels per route
        route_ratio = len(solution) / optimal_routes
        route_score = 1.0 - abs(1.0 - route_ratio) ** 0.7

        # Load balancing score with improved scaling
        weights = [route.get_total_weight() for route in solution]
        if weights:
            avg_weight = sum(weights) / len(weights)
            weight_variations = [abs(w - avg_weight) / avg_weight if avg_weight > 0 else 1.0 for w in weights]
            balance_score = 1.0 - min(1.0, sum(weight_variations) / len(weights)) ** 0.8
        else:
            balance_score = 0.0

        # Distance efficiency with more granular scaling
        avg_distance_per_parcel = total_distance / total_parcels
        max_acceptable_distance = 100  # Example threshold per parcel
        distance_score = 1.0 - min(1.0, (avg_distance_per_parcel / max_acceptable_distance) ** 0.5)

        # Combine scores with adjusted weights
        weights = {
            'parcels': 0.40,  # Increased importance of delivering parcels
            'cost': 0.25,     # Balanced cost consideration
            'routes': 0.15,   # Route count importance
            'balance': 0.10,  # Load balancing importance
            'distance': 0.10  # Distance efficiency
```

```python
        fitness = (weights['parcels'] * parcels_score +
                   weights['cost'] * cost_score +
                   weights['routes'] * route_score +
                   weights['balance'] * balance_score +
                   weights['distance'] * distance_score)

        # Apply penalty for infeasible solutions with smoother scaling
        infeasible_routes = sum(1 for route in solution if not route.is_feasible
        if infeasible_routes > 0:
            penalty = (infeasible_routes / len(solution)) ** 1.5
            fitness *= (1.0 - penalty)

        return max(0.0, min(1.0, fitness))
    return 0.0
```

OR-TOOLS WITH ML ENHANCED

OR-TOOLS

- VRP model
- Distance & Capacity constraints
- Multiple solver strategies

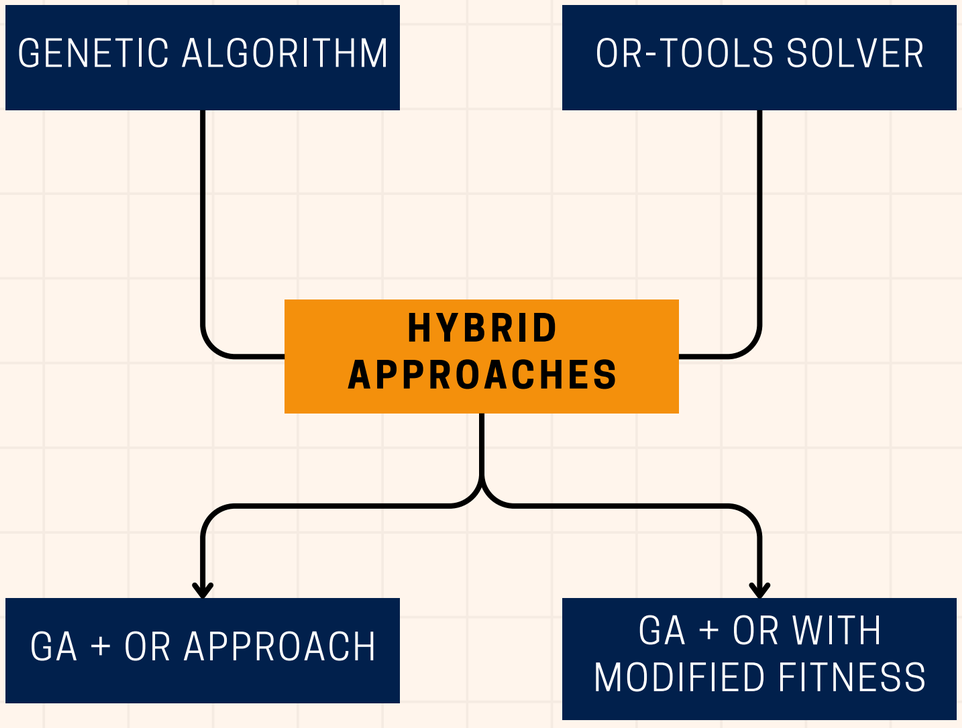
ML MODEL

- Random Forest Regression
- Predicts route costs
- Feature:

1. Parcels
2. Weights
3. Distannce

ROUTE PREDICTION AND OPTIMIZATION

HYBRID APPROACHES



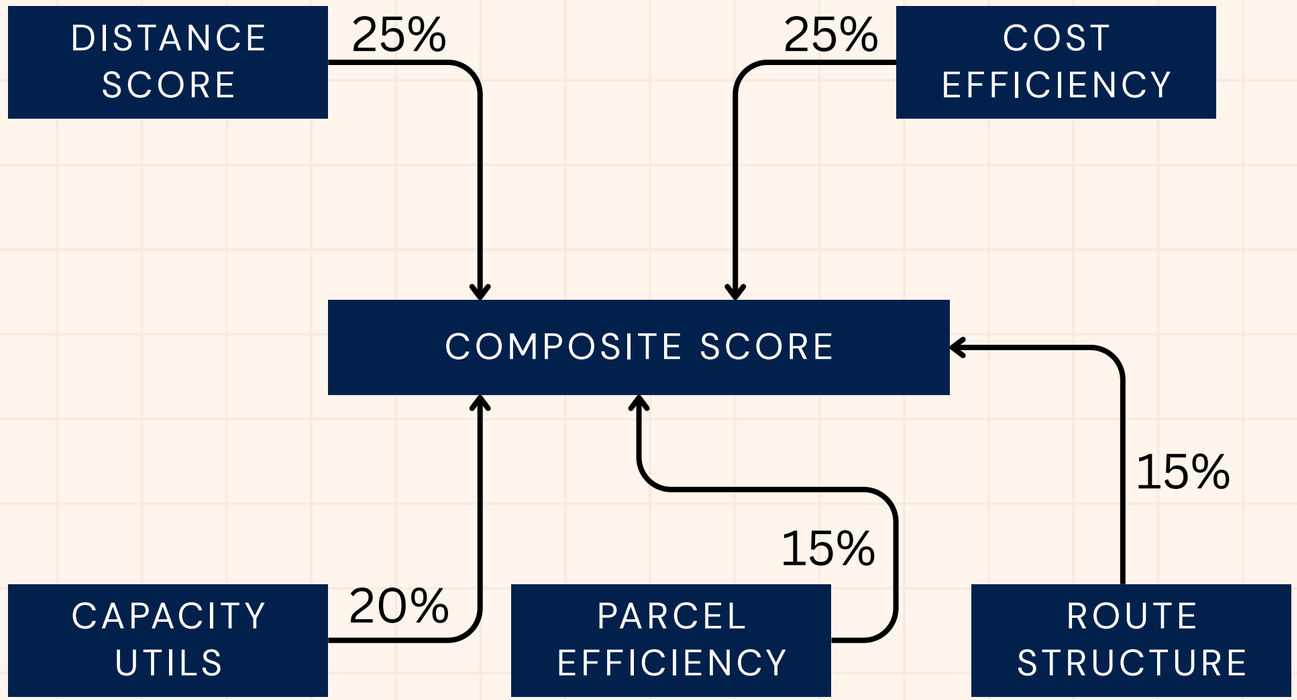
GA + OR:

- OR-Tools solutions
- Initial GA population
- Standard Fitness

GA + OR Modified:

- Pattern Extraction
- Weighted Fitness
- Adapted Influence

COMPOSITE SCORE CALCULATION



SCORE CALCULATION DETAILS

1. Distance Score

max(0, 1 – (avg_distance_per_parcel / (total_parcels * 100)))

2. Cost Efficiency Score

max(0, 1 – (cost_per_parcel / (total_distance * 2.5)))

3. Capacity Utilization Score

Average of 1 - abs(0.85 - utilization) across all routes

4. Parcel Efficiency Score

min(1.0, avg_parcels_per_route / 15)

5. Route Structure Score

Average of max(0, 1 - (avg_consecutive_distance / 1000)) across routes

Example value

- Distance : 0.83
- Cost: 0.85
- Capacity: 0.79
- Parcel: 0.88
- Structure: 0.84

Composite

Composite Score = 0.83 * 0.25 + 0.85 * 0.25 + 0.79 * 0.2 + 0.88 * 0.15 + 0.84 * 0.15 = 0.84

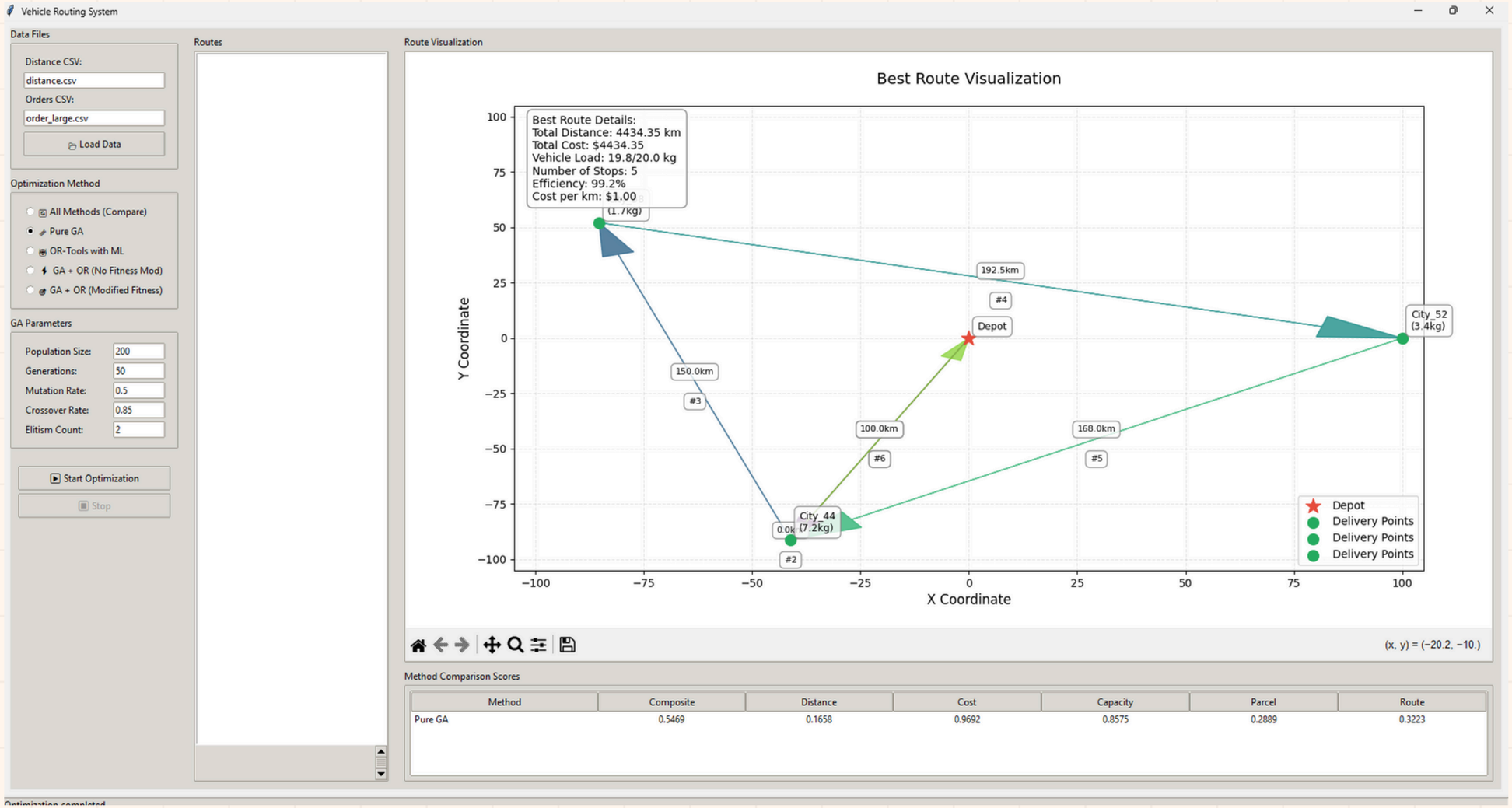
SCORE CALCULATION

```python
# 1. Distance Score (Lower is better)
total_distance = sum(route.total_distance for route in routes)
total_parcels = sum(len(route.parcels) for route in routes)
avg_distance_per_parcel = total_distance / total_parcels if total_parcels > 0 else float('inf')
# Use logarithmic scaling for large distances
distance_score = max(0, 1 - (avg_distance_per_parcel / (total_parcels * 100)))

# 2. Cost Efficiency Score
total_cost = sum(route.total_cost for route in routes)
cost_per_parcel = total_cost / total_parcels if total_parcels > 0 else float('inf')
# Use logarithmic scaling for large costs
cost_efficiency_score = max(0, 1 - (cost_per_parcel / (total_distance * 2.5)))

# 3. Capacity Utilization Score
utilization_scores = []
for route in routes:
    capacity = route.vehicle_capacity
    if capacity > 0:
        utilization = route.get_total_weight() / capacity
        # Penalize both under and over-utilization
        utilization_score = 1 - abs(0.85 - min(utilization, 1.0))  # 85% utilization is ideal, cap at 100%
        utilization_scores.append(utilization_score)
capacity_utilization_score = sum(utilization_scores) / len(utilization_scores) if utilization_scores else 0

# 4. Parcel Efficiency Score
avg_parcels_per_route = total_parcels / len(routes)
# Assume 5 parcels per route is minimum efficient, 15 is optimal
parcel_efficiency_score = min(1.0, avg_parcels_per_route / 15)

# 5. Route Structure Score
structure_scores = []
for route in routes:
    if len(route.locations) > 2:  # More than just depot-return
        # Calculate average distance between consecutive stops
        consecutive_distances = []
        for i in range(len(route.locations) - 1):
            if self.data_processor:
                source_idx = self.data_processor.city_to_idx.get(route.locations[i].city_name, 0)
                dest_idx = self.data_processor.city_to_idx.get(route.locations[i + 1].city_name, 0)
                consecutive_distances.append(self.data_processor.distance_matrix[source_idx][dest_idx])
```

```python
# Calculate composite score with weighted components
weights = {
    'distance': 0.25,
    'cost_efficiency': 0.25,
    'capacity_utilization': 0.2,
    'parcel_efficiency': 0.15,
    'route_structure': 0.15
}

composite_score = (
    weights['distance'] * distance_score +
    weights['cost_efficiency'] * cost_efficiency_score +
    weights['capacity_utilization'] * capacity_utilization_score +
    weights['parcel_efficiency'] * parcel_efficiency_score +
    weights['route_structure'] * route_structure_score
)
```

ROUTE VISUALIZATION

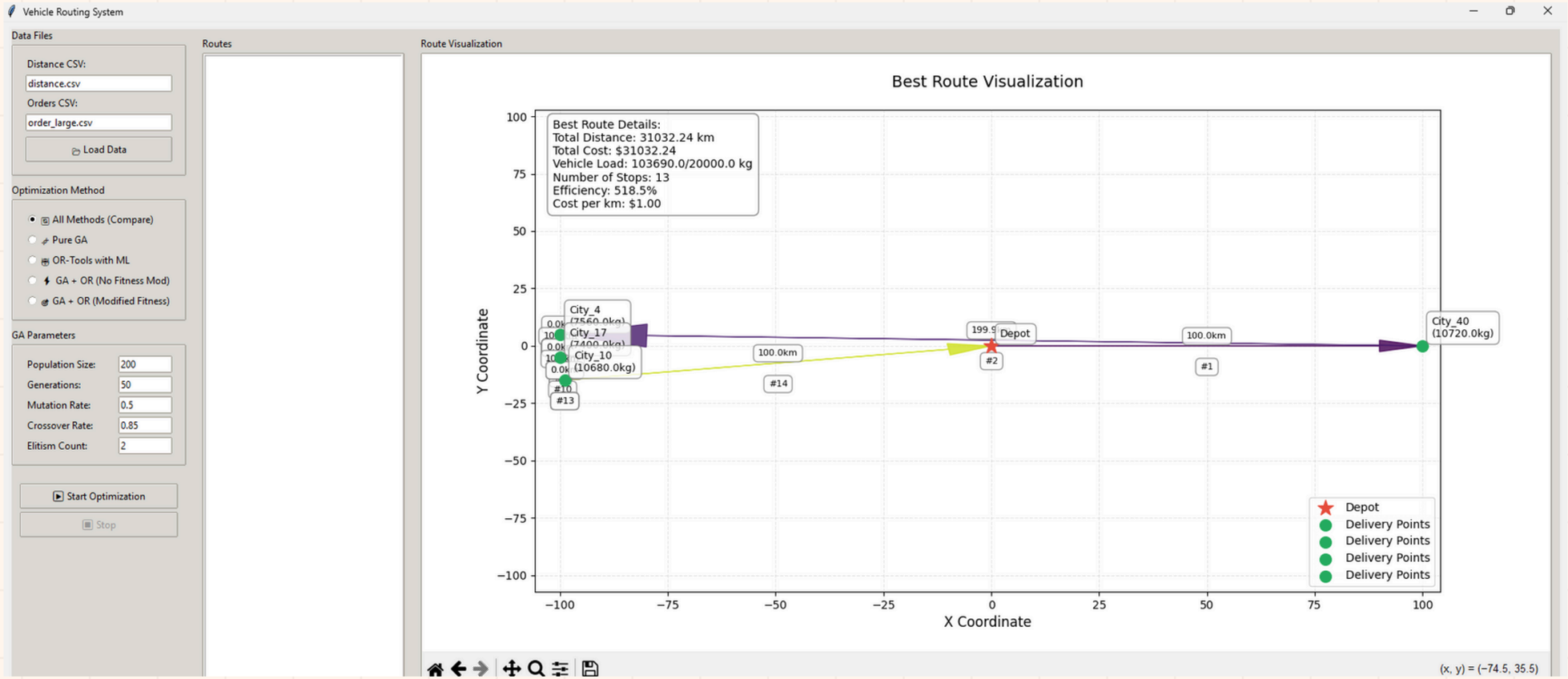
PURE GA


Vehicle Routing System
Data Files
Distance CSV:
distance.csv
Orders CSV:
order_large.csv
Load Data
Optimization Method
All Methods (Compare)
Pure GA
OR-Tools with ML
GA + OR (No Fitness Mod)
GA + OR (Modified Fitness)
GA Parameters
Population Size: 200
Generations: 50
Mutation Rate: 0.5
Crossover Rate: 0.85
Elitism Count: 2
Start Optimization
Stop
Routes
Route Visualization
Best Route Visualization
Best Route Details:
Total Distance: 4434.35 km
Total Cost: $4434.35
Vehicle Load: 19.8/20.0 kg
Number of Stops: 5
Efficiency: 99.2%
Cost per km: $1.00
192.5km
#4
Depot
City_52
(3.4kg)
150.0km
#3
100.0km
#6
168.0km
#5
City_44
(7.2kg)
#2
Depot
Delivery Points
Delivery Points
Delivery Points
Y Coordinate
X Coordinate
(x, y) = (−20.2, −10.)
Method Comparison Scores
Method
Composite
Distance
Cost
Capacity
Parcel
Route
Pure GA
0.5469
0.1658
0.9692
0.8575
0.2889
0.3223

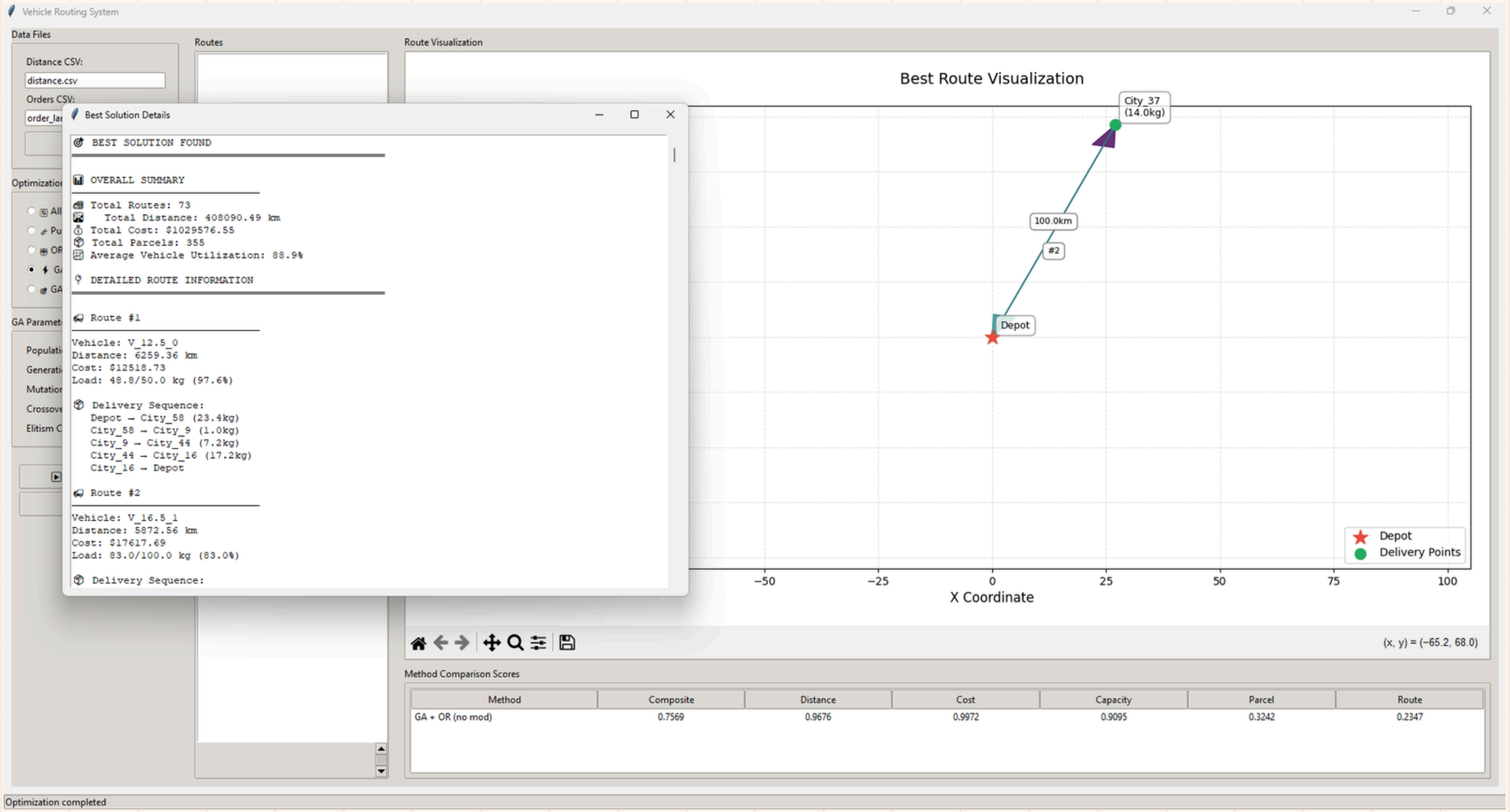
VRP

HOME SERVICE ABOUT US CONTACT US

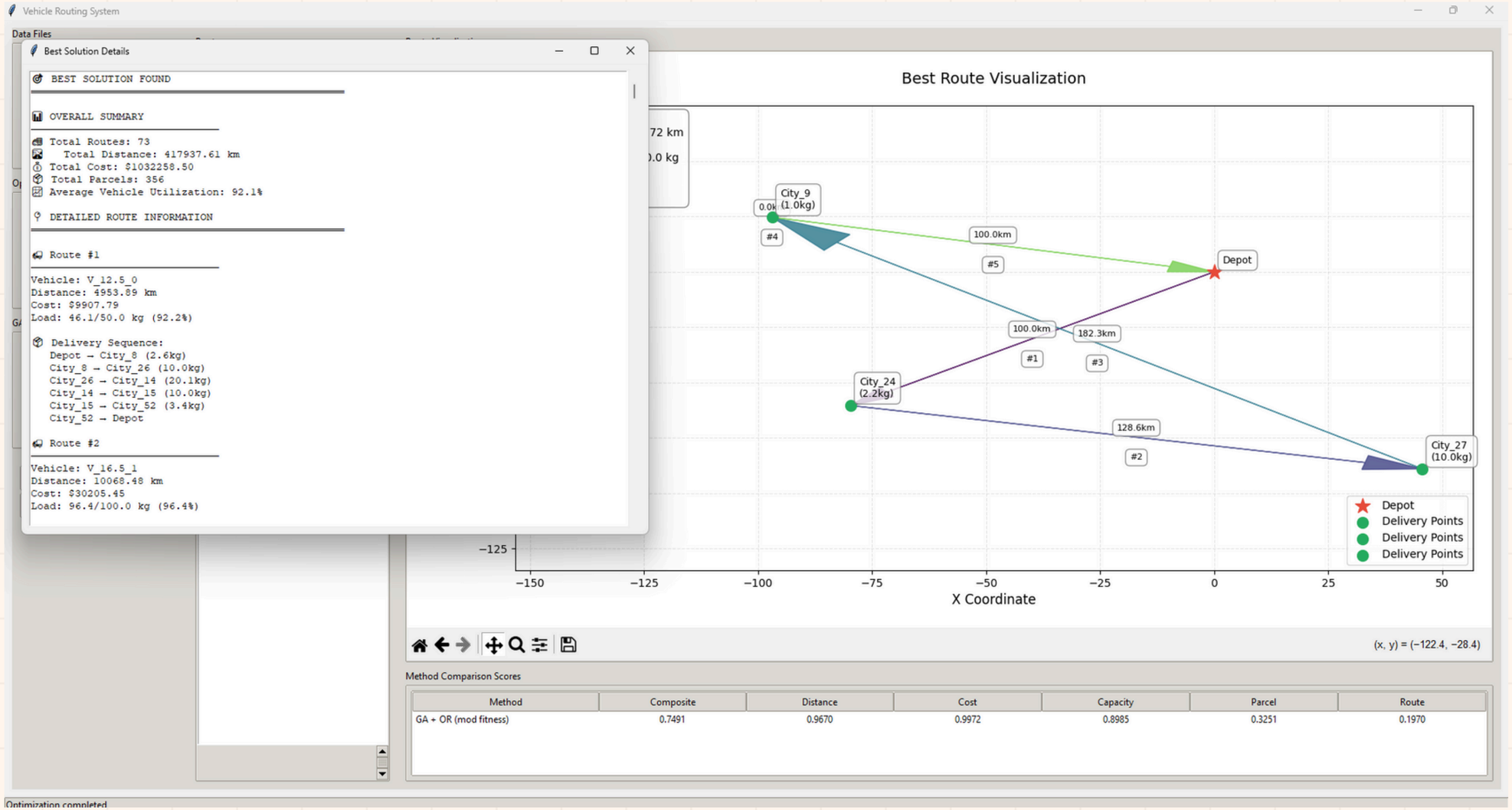
OR + ML



GA + OR


Vehicle Routing System
Data Files
Distance CSV:
distance.csv
Orders CSV:
Routes
Route Visualization
Best Route Visualization
City_37
(14.0kg)
100.0km
#2
Depot
Depot
Delivery Points
X Coordinate
(x, y) = (−65.2, 68.0)
Best Solution Details
BEST SOLUTION FOUND
OVERALL SUMMARY
Total Routes: 73
Total Distance: 408090.49 km
Total Cost: $1029576.55
Total Parcels: 355
Average Vehicle Utilization: 88.9%
DETAILED ROUTE INFORMATION
Route #1
Vehicle: V_12.5_0
Distance: 6259.36 km
Cost: $12518.73
Load: 48.8/50.0 kg (97.6%)
Delivery Sequence:
Depot → City_58 (23.4kg)
City_58 → City_9 (1.0kg)
City_9 → City_44 (7.2kg)
City_44 → City_16 (17.2kg)
City_16 → Depot
Route #2
Vehicle: V_16.5_1
Distance: 5872.56 km
Cost: $17617.69
Load: 83.0/100.0 kg (83.0%)
Delivery Sequence:
Method Comparison Scores
Method
Composite
Distance
Cost
Capacity
Parcel
Route
GA + OR (no mod)
0.7569
0.9676
0.9972
0.9095
0.3242
0.2347
Optimization completed

GA + OR MODIFIED



DETAILS SCORE COMPARISION

Method Comparison Scores

Method	Composite	Distance	Cost	Capacity	Parcel	Route
Pure GA	0.6930	0.5970	0.9765	0.8806	0.3778	0.4454
OR-Tools + ML	0.8531	0.9991	0.9997	0.8500	1.0000	0.2227
GA + OR (no mod)	0.7518	0.9684	0.9971	0.8820	0.3306	0.2293
GA + OR (mod fitness)	0.7522	0.9663	0.9973	0.9051	0.3205	0.2145

KEY FINDINGS

BEST METHOD

OR + ML

- Highest performance on key metrics
- Most efficient execution time
- Excellent parcel efficiency

BEST BALANCE

GA + OR modified fitness

- Good balance across all metrics
- Improvements in capacity utilization
- Better route structure scores

BEST ADPATIVE

Pure GA

- Easily adaptable to new constraints
- No dependency on external solvers
- Highly customizable approach

FUTURE WORKS

Enhanced Optimization

- Deep learning for route prediction
- Real-time traffic integration
- Multi-objective optimization

Additional Constraints

- Driver working hours
- Variable delivery time windows
- Multi-day planning horizon

Real-World Implementation

- Mobile driver application
- Cloud-based optimization service
- Real-time route adjustment

VRP

TIMELINE

W2

- Project Setup and Planning

W3

- Basic Infrastructure

W4

- Agent Communication

W5 – 6

- Data Processing & Initial Optimization Setup.
- Core Optimization Algorithm Implementation.

W7

- Advanced Route Optimization Features

W8

- Optimization Refinement & Testing

W9 – 10

- Testing
- GUI Development

W10 – 12

- Documentation and Presentation

THANK FOR YOUR ATTENTION

VEHICLE ROUTING SYSTEM (VRP)

Q & A