```
import pulp
import numpy as np
import time
from collections import defaultdict

class VRPTWOptimizer:
  def __init__(self, customers, depot_start, depot_end, costs,
time_windows, demands,
               vehicle_capacity, K=2, capacity_granularity=3,
time_granularity=3):
  self.customers = customers
self.depot_start = depot_start
self.depot_end = depot_end
self.costs = costs
self.time_windows = time_windows
self.demands = demands
self.vehicle_capacity = vehicle_capacity

# Initialize model
self.model = pulp.LpProblem("VRPTW", pulp.LpMinimize)

# Create valid edges
self.E_star = [(i,j) for i in [self.depot_start] + self.customers
               for j in self.customers + [self.depot_end] if i != j]

# For debugging, start with just basic variables
self._create_basic_variables()
self._add_basic_constraints()

print(f"\nModel initialized with:")
print(f"Number of customers: {len(customers)}")
print(f"Number of edges: {len(self.E_star)}")

def _create_basic_variables(self):
  """Create only basic routing variables"""
# Route variables x_{ij}
self.x = pulp.LpVariable.dicts("x",
                               self.E_star,
                               cat='Binary')

# Time of service variables
self.tau = pulp.LpVariable.dicts("tau",
                                 self.customers + [self.depot_start,
self.depot_end],
                                 lowBound=0)

```

```
# Vehicle load variables
self.load = pulp.LpVariable.dicts("load",
                                  self.customers + [self.depot_start, self.depot_end],
                                  lowBound=0,
                                  upBound=self.vehicle_capacity)

def _add_basic_constraints(self):
  """Add only essential routing constraints"""
# Objective function
self.model += pulp.lpSum(self.costs[i,j] * self.x[i,j] for i,j in self.E_star)

# Visit each customer once
for u in self.customers:
  self.model += pulp.lpSum(self.x[i,u] for i,j in self.E_star if j == u) == 1
self.model += pulp.lpSum(self.x[u,j] for i,j in self.E_star if i == u) == 1

# Time windows
M = max(tw[1] for tw in self.time_windows.values())
for (i,j) in self.E_star:
  if j != self.depot_end:
  self.model += self.tau[j] >= self.tau[i] + self.costs[i,j]/5 - M * (1 - self.x[i,j])

# Time window bounds
for i in self.customers + [self.depot_start, self.depot_end]:
  self.model += self.tau[i] >= self.time_windows[i][0]
self.model += self.tau[i] <= self.time_windows[i][1]

# Vehicle capacity
self.model += self.load[self.depot_start] == self.vehicle_capacity
for (i,j) in self.E_star:
  if j != self.depot_end:
  self.model += self.load[j] <= self.load[i] - self.demands[j] + \
self.vehicle_capacity * (1 - self.x[i,j])

# Minimum vehicles
total_demand = sum(self.demands[u] for u in self.customers)
min_vehicles = int(np.ceil(total_demand / self.vehicle_capacity))
self.model += pulp.lpSum(self.x[self.depot_start,j]
                         for i,j in self.E_star if i == self.depot_start) >= min_vehicles

```

```
def solve(self, time_limit=None):
  """Solve the VRPTW instance"""
print("\nSolving model...")
start_time = time.time()

if time_limit:
  status = self.model.solve(pulp.PULP_CBC_CMD(timeLimit=time_limit))
else:
  status = self.model.solve(pulp.PULP_CBC_CMD())

solve_time = time.time() - start_time
print(f"Status: {pulp.LpStatus[status]}")

solution = {
  'status': pulp.LpStatus[status],
  'computation_time': solve_time,
  'objective': pulp.value(self.model.objective) if status == pulp.LpStatusOptimal else None
}

if status == pulp.LpStatusOptimal:
  solution['routes'] = self._extract_routes()

return solution

def _extract_routes(self):
  """Extract routes from solution"""
active_edges = [(i,j) for (i,j) in self.E_star
                if pulp.value(self.x[i,j]) is not None
                and pulp.value(self.x[i,j]) > 0.5]

routes = []
depot_starts = [(i,j) for (i,j) in active_edges if i == self.depot_start]

for start_edge in depot_starts:
  route = []
current = start_edge[1]
route.append(current)

while current != self.depot_end:
  next_edges = [(i,j) for (i,j) in active_edges if i == current]
if not next_edges:
  break
current = next_edges[0][1]
if current != self.depot_end:
  route.append(current)
```

```

routes.append(route)

return routes

def create_small_test_instance():
  """Create a very simple test instance"""
locations = {
  0: (0, 0),    # Depot start
  1: (1, 1),    # Customer 1
  2: (-1, 1),   # Customer 2
  3: (1, -1),   # Customer 3
  4: (-1, -1),  # Customer 4
  5: (0, 0)     # Depot end
}

# Calculate costs - ensure all needed edges exist
costs = {}
for i in range(6):  # Include depot end (5)
  for j in range(6):
  if i != j:  # Don't need cost from node to itself
  x1, y1 = locations[i]
x2, y2 = locations[j]
costs[i,j] = int(np.sqrt((x2-x1)**2 + (y2-y1)**2) * 5)

time_windows = {
  0: (0, 1000),    # Very wide depot window
  1: (0, 100),     # Very wide customer windows
  2: (0, 100),
  3: (0, 100),
  4: (0, 100),
  5: (0, 1000)     # Very wide depot window
}

demands = {
  0: 0,    # Depot
  1: 1,    # Very small demands
  2: 1,
  3: 1,
  4: 1,
  5: 0     # Depot
}

return {
  'customers': [1, 2, 3, 4],
  'depot_start': 0,
```

```python
  'depot_end': 5,
  'costs': costs,
  'time_windows': time_windows,
  'demands': demands,
  'vehicle_capacity': 10
}

def main():
  print("Creating small test instance...")
instance = create_small_test_instance()

print("\nProblem characteristics:")
print(f"Number of customers: {len(instance['customers'])}")
print(f"Vehicle capacity: {instance['vehicle_capacity']}")
print(f"Total demand: {sum(instance['demands'][i] for i in instance['customers'])}")

print("\nCustomer Details:")
for i in sorted(instance['customers']):
  print(f"Customer {i}: Window {instance['time_windows'][i]}, Demand: {instance['demands'][i]}")

optimizer = VRPTWOptimizer(
  customers=instance['customers'],
  depot_start=instance['depot_start'],
  depot_end=instance['depot_end'],
  costs=instance['costs'],
  time_windows=instance['time_windows'],
  demands=instance['demands'],
  vehicle_capacity=instance['vehicle_capacity']
)

solution = optimizer.solve(time_limit=300)

if solution['status'] == 'Optimal':
  print(f"\nOptimal Solution Cost: {solution['objective']:.2f}")
print("\nRoutes:")
total_cost = 0
for idx, route in enumerate(solution['routes'], 1):
  route_demand = sum(instance['demands'][c] for c in route)

# Calculate route cost properly - only between consecutive nodes
route_with_depots = [instance['depot_start']] + route + [instance['depot_end']]
route_cost = sum(instance['costs'][route_with_depots[i], route_with_depots[i+1]]
                 for i in range(len(route_with_depots)-1))
total_cost += route_cost

print(f"\nRoute {idx}: {' -> '.join([str(instance['depot_start'])] + [str(c) for c in route] + [str(instance['depot_end'])])}")
print(f"  Total demand: {route_demand}")
print(f"  Schedule:")
current_time = 0
current_loc = instance['depot_start']
for stop in route:
  travel_time = instance['costs'][current_loc, stop] / 5
arrival_time = max(current_time + travel_time, instance['time_windows'][stop][0])
print(f"    Customer {stop}: Arrive at {arrival_time:.1f} "
      f"(Window: {instance['time_windows'][stop]}, "
      f"Demand: {instance['demands'][stop]})")
current_time = arrival_time
current_loc = stop

print(f"\nTotal Cost: {total_cost}")
else:
  print(f"\nNo optimal solution found. Status: {solution['status']}")

if __name__ == "__main__":
  main()

```