

Google开源框架or-tools解决路径规划问题

问题列表:

- 旅行商TSP问题
- 路径规划VRP问题
- 容量限制CVRP问题
- 取货送货pickups and deliveries问题
- VRP时间窗约束问题 (time window)

TSP问题

TSP问题是旅行商问题的简写，问题非常简单：从原点出发经过所有需求点并回到原点，使得途经的距离最短。

```
1  """simple travelling salesman problem between cities."""
2
3  from __future__ import print_function
4  from ortools.constraint_solver import routing_enums_pb2
5  from ortools.constraint_solver import pywrapcp
6
7
8
9  def create_data_model():
10     """Stores the data for the problem."""
11     data = {}
12     #节点之间的距离矩阵，对称矩阵
13     data['distance_matrix'] = [
14         [0, 2451, 713, 1018, 1631, 1374, 2408, 213, 2571, 875, 1420, 2145,
15         1972],
16         [2451, 0, 1745, 1524, 831, 1240, 959, 2596, 403, 1589, 1374, 357,
17         579],
18         [713, 1745, 0, 355, 920, 803, 1737, 851, 1858, 262, 940, 1453,
19         1260],
20         [1018, 1524, 355, 0, 700, 862, 1395, 1123, 1584, 466, 1056, 1280,
21         987],
22         [1631, 831, 920, 700, 0, 663, 1021, 1769, 949, 796, 879, 586, 371],
23         [1374, 1240, 803, 862, 663, 0, 1681, 1551, 1765, 547, 225, 887,
24         999],
25         [2408, 959, 1737, 1395, 1021, 1681, 0, 2493, 678, 1724, 1891, 1114,
26         701],
27         [213, 2596, 851, 1123, 1769, 1551, 2493, 0, 2699, 1038, 1605, 2300,
28         2099],
29         [2571, 403, 1858, 1584, 949, 1765, 678, 2699, 0, 1744, 1645, 653,
30         600],
31         [875, 1589, 262, 466, 796, 547, 1724, 1038, 1744, 0, 679, 1272,
32         1162],
33         [1420, 1374, 940, 1056, 879, 225, 1891, 1605, 1645, 679, 0, 1017,
34         1200],
35         [2145, 357, 1453, 1280, 586, 887, 1114, 2300, 653, 1272, 1017, 0,
36         504],
37         [1972, 579, 1260, 987, 371, 999, 701, 2099, 600, 1162, 1200, 504,
38         0],
```

```

27     ] # yapf: disable
28     #货车数量
29     data['num_vehicles'] = 1
30     data['depot'] = 0
31     return data
32
33 #打印结果
34 def print_solution(manager, routing, solution):
35     """Prints solution on console."""
36     print('Objective: {} miles'.format(solution.ObjectiveValue()))
37     index = routing.Start(0)
38     plan_output = 'Route for vehicle 0:\n'
39     route_distance = 0
40     while not routing.IsEnd(index):
41         plan_output += ' {} ->'.format(manager.IndexToNode(index))
42         previous_index = index
43         index = solution.Value(routing.NextVar(index))
44         route_distance += routing.GetArcCostForVehicle(previous_index,
index, 0)
45     plan_output += ' {}\n'.format(manager.IndexToNode(index))
46     print(plan_output)
47     plan_output += 'Route distance: {}miles\n'.format(route_distance)
48
49
50 def main():
51     """Entry point of the program."""
52     # Instantiate the data problem.
53     data = create_data_model()
54
55     # Create the routing index manager.
56     manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
data['num_vehicles'],
data['depot'])
57
58
59     # Create Routing Model.
60     routing = pywrapcp.RoutingModel(manager)
61
62     #距离回调函数，将两节点序号转化为两节点距离
63     def distance_callback(from_index, to_index):
64         """Returns the distance between the two nodes."""
65         # Convert from routing variable Index to distance matrix NodeIndex.
66         from_node = manager.IndexToNode(from_index)
67         to_node = manager.IndexToNode(to_index)
68         return data['distance_matrix'][from_node][to_node]
69
70     transit_callback_index =
routing.RegisterTransitCallback(distance_callback)
71
72     #定义每条弧的权重
73     # Define cost of each arc.
74     routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
75
76     # Setting first solution heuristic.
77     search_parameters = pywrapcp.DefaultRoutingSearchParameters()
78     search_parameters.first_solution_strategy = (
routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
79
80
81     # Solve the problem.

```

```

82     solution = routing.SolveWithParameters(search_parameters)
83
84     # Print solution on console.
85     if solution:
86         print_solution(manager, routing, solution)
87
88
89 if __name__ == '__main__':
90     main()

```

VRP问题

VRP问题是车辆路径问题的缩写。问题是：有N辆车，都从原点出发，每辆车访问一些点后回到原点，要求所有的点都要被访问到，求最短的车辆行驶距离或最少需要的车辆数或最小化最长行驶距离。常见的限制要求包括：车辆容量限制、时间窗限制、点访问顺序要求等。

```

1  """Vehicles Routing Problem (VRP)."""
2
3  from __future__ import print_function
4  from ortools.constraint_solver import routing_enums_pb2
5  from ortools.constraint_solver import pywrapcp
6
7
8  def create_data_model():
9      """Stores the data for the problem."""
10     data = {}
11     #对称矩阵
12     data['distance_matrix'] = [
13         [
14             0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388,
15             354,
16             468, 776, 662
17         ],
18         [
19             548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594,
20             480, 674,
21             1016, 868, 1210
22         ],
23         [
24             776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278,
25             1164,
26             1130, 788, 1552, 754
27         ],
28         [
29             696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514,
30             628, 822,
31             1164, 560, 1358
32         ],
33         [
34             582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400,
35             514, 708,
36             1050, 674, 1244
37         ],
38         [
39             274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662,
40             628,

```

35		514, 1050, 708
36],	
37	[
38		502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004,
	890, 856,	
39		514, 1278, 480
40],	
41	[
42		194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354,
	320,	
43		662, 742, 856
44],	
45	[
46		308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696,
	662,	
47		320, 1084, 514
48],	
49	[
50		194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422,
	388,	
51		274, 810, 468
52],	
53	[
54		536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878,
	764,	
55		730, 388, 1152, 354
56],	
57	[
58		502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0,
	114,	
59		308, 650, 274, 844
60],	
61	[
62		388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114,
	0, 194,	
63		536, 388, 730
64],	
65	[
66		354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308,
	194, 0,	
67		342, 422, 536
68],	
69	[
70		468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650,
	536,	
71		342, 0, 764, 194
72],	
73	[
74		776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152,
	274,	
75		388, 422, 764, 0, 798
76],	
77	[
78		662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844,
	730,	
79		536, 194, 798, 0
80],	
81]	

```

82     #货车数量
83     data['num_vehicles'] = 4
84     data['depot'] = 0
85     return data
86
87
88 def print_solution(data, manager, routing, solution):
89     """Prints solution on console."""
90     max_route_distance = 0
91     for vehicle_id in range(data['num_vehicles']):
92         index = routing.Start(vehicle_id)
93         plan_output = 'Route for vehicle {}: \n'.format(vehicle_id)
94         route_distance = 0
95         while not routing.IsEnd(index):
96             plan_output += ' {} -> '.format(manager.IndexToNode(index))
97             previous_index = index
98             index = solution.Value(routing.NextVar(index))
99             route_distance += routing.GetArcCostForVehicle(
100                 previous_index, index, vehicle_id)
101             plan_output += ' {} \n'.format(manager.IndexToNode(index))
102             plan_output += 'Distance of the route:
103             {}m \n'.format(route_distance)
104             print(plan_output)
105             max_route_distance = max(route_distance, max_route_distance)
106         print('Maximum of the route distances:
107         {}m'.format(max_route_distance))
108
109
110 def main():
111     """Solve the CVRP problem."""
112     # Instantiate the data problem.
113     data = create_data_model()
114
115     # Create the routing index manager.
116     manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
117     data['num_vehicles'],
118     data['depot'])
119
120     # Create Routing Model.
121     routing = pywrapcp.RoutingModel(manager)
122
123     # Create and register a transit callback.
124     def distance_callback(from_index, to_index):
125         """Returns the distance between the two nodes."""
126         # Convert from routing variable Index to distance matrix
127         NodeIndex.
128         from_node = manager.IndexToNode(from_index)
129         to_node = manager.IndexToNode(to_index)
130         return data['distance_matrix'][from_node][to_node]
131
132     transit_callback_index =
133     routing.RegisterTransitCallback(distance_callback)
134
135     # Define cost of each arc.
136     routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

```

```

135
136     # Add Distance constraint.
137     dimension_name = 'Distance'
138     routing.AddDimension(
139         transit_callback_index,
140         0, # no slack
141         #货车最大送货距离
142         3000, # vehicle maximum travel distance
143         True, # start cumul to zero
144         dimension_name)
145     distance_dimension = routing.GetDimensionOrDie(dimension_name)
146     distance_dimension.SetGlobalSpanCostCoefficient(100)
147
148     # Setting first solution heuristic.
149     search_parameters = pywrapcp.DefaultRoutingSearchParameters()
150     search_parameters.first_solution_strategy = (
151         routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
152
153     # Solve the problem.
154     solution = routing.SolveWithParameters(search_parameters)
155
156     # Print solution on console.
157     if solution:
158         print_solution(data, manager, routing, solution)
159
160
161 if __name__ == '__main__':
162     main()

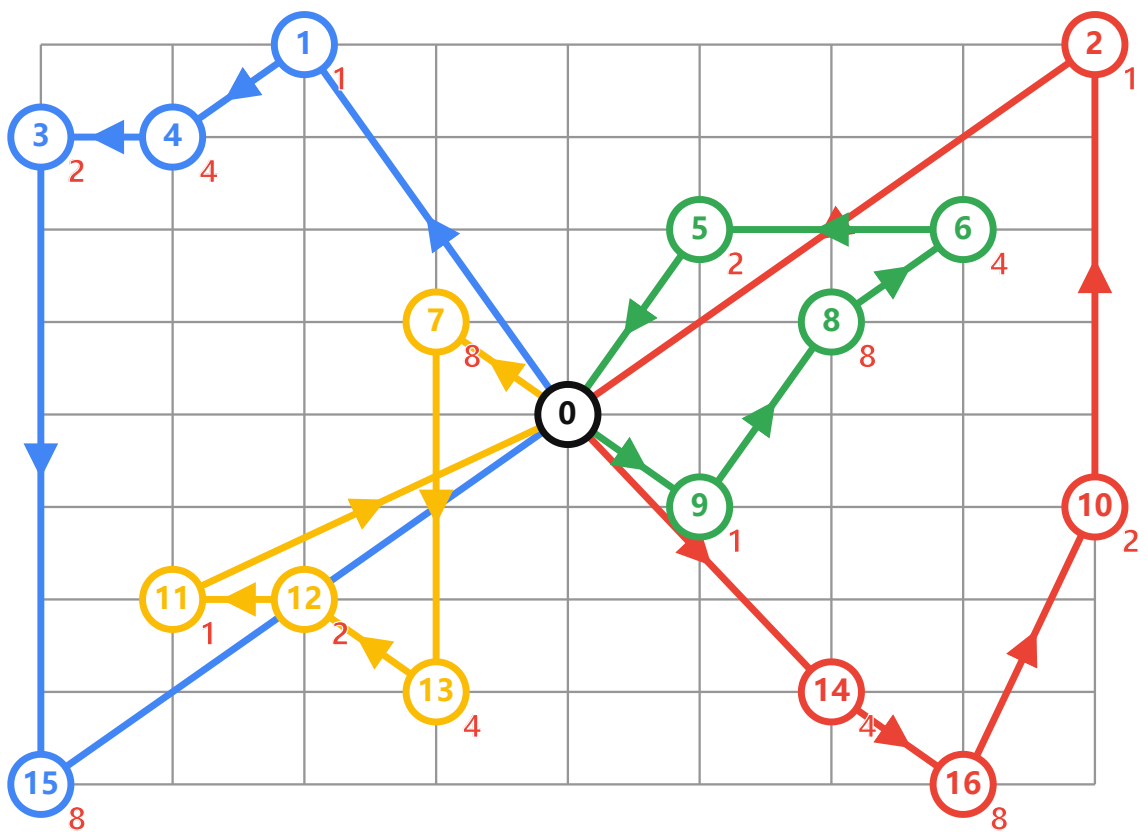
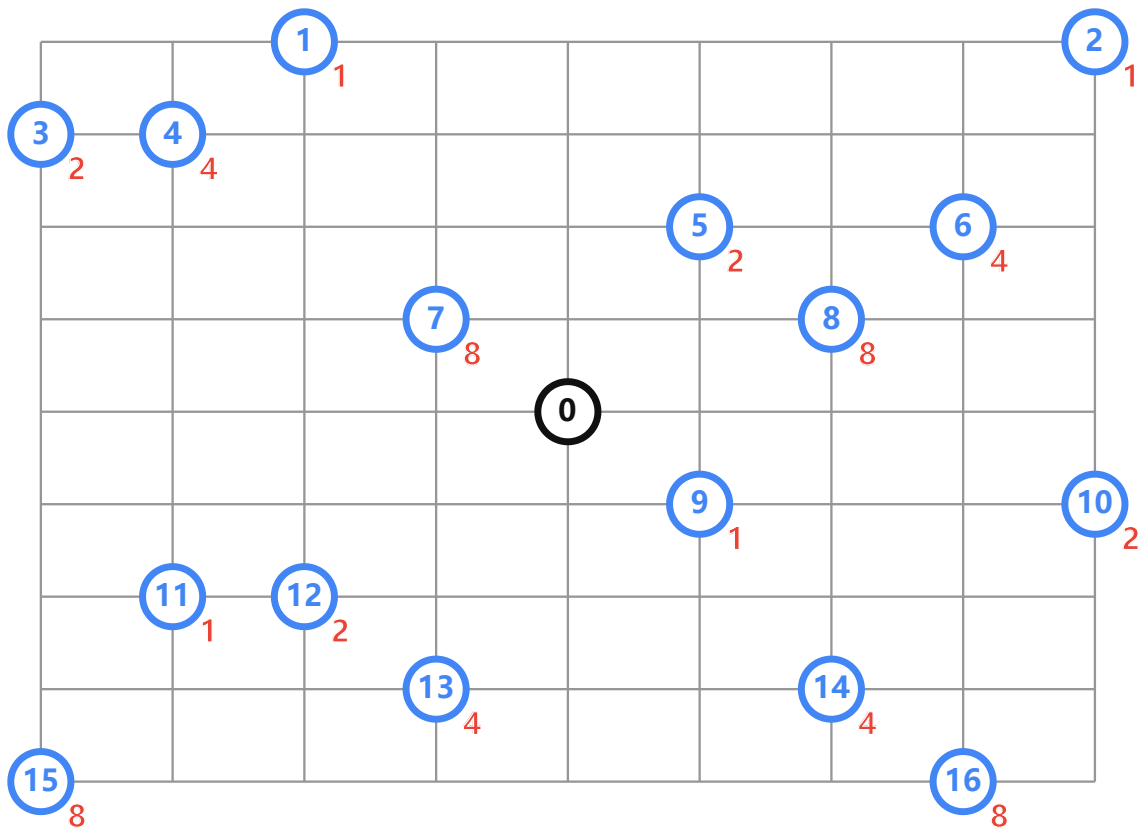
```

CVRP问题

CVRP指的是有容量（capacity）限制的VRP模型，是最常见的VRP模型。

每个点多了一个送货需求。每辆车的最大容量是15，最小化总运输距离。

dimension可以使用AddDimensionWithVehicleCapacity方法，和AddDimension唯一的区别就是，第三个参数从一个固定值变成了一个列表，表示每一辆车有自己单独的最大容量限制。



Route for vehicle 0:
 0 Load(0) -> 1 Load(1) -> 4 Load(5) -> 3 Load(7) -> 15 Load(15) -> 0 Load(15)
 Distance of the route: 2192m
 Load of the route: 15

Route for vehicle 1:
 0 Load(0) -> 14 Load(4) -> 16 Load(12) -> 10 Load(14) -> 2 Load(15) -> 0 Load(15)
 Distance of the route: 2192m
 Load of the route: 15

Route for vehicle 2:
 0 Load(0) -> 7 Load(8) -> 13 Load(12) -> 12 Load(14) -> 11 Load(15) -> 0 Load(15)
 Distance of the route: 1324m
 Load of the route: 15

Route for vehicle 3:
 0 Load(0) -> 9 Load(1) -> 8 Load(9) -> 6 Load(13) -> 5 Load(15) -> 0 Load(15)
 Distance of the route: 1164m
 Load of the route: 15

Total Distance of all routes: 6872m

```

1  """Capacited Vehicles Routing Problem (CVRP)."""
2
3  from __future__ import print_function
4  from ortools.constraint_solver import routing_enums_pb2
5  from ortools.constraint_solver import pywrapcp
6
7
8  def create_data_model():
9      """Stores the data for the problem."""
10     data = {}
11     #对称矩阵
12     data['distance_matrix'] = [
13         [
14             0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388,
15             354,
16             468, 776, 662
17         ],
18         [
19             548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594,
20             480, 674,
21             1016, 868, 1210
22         ],
23         [
24             776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278,
25             1164,
26             1130, 788, 1552, 754
27         ],
28         [
29             696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514,
30             628, 822,
31             1164, 560, 1358
32         ],
33         [
34             582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400,
35             514, 708,
36             1050, 674, 1244
37         ],
38         [
39             274, 502, 274, 810, 468, 742, 400, 1278,
40             1130, 788, 1552, 754
41         ],
42         [
43             502, 730, 354, 696, 742, 1084, 594,
44             480, 674,
45             1016, 868, 1210
46         ],
47         [
48             194, 308, 194, 536, 502, 388,
49             354,
50             468, 776, 662
51         ],
52         [
53             308, 194, 536, 502, 388,
54             468, 776, 662
55         ],
56         [
57             536, 502, 388,
58             468, 776, 662
59         ],
60         [
61             502, 388,
62             468, 776, 662
63         ],
64         [
65             388,
66             468, 776, 662
67         ]
68     ]

```


33	[
34		274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662,
	628,	
35		514, 1050, 708
36],	
37	[
38		502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004,
	890, 856,	
39		514, 1278, 480
40],	
41	[
42		194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354,
	320,	
43		662, 742, 856
44],	
45	[
46		308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696,
	662,	
47		320, 1084, 514
48],	
49	[
50		194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422,
	388,	
51		274, 810, 468
52],	
53	[
54		536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878,
	764,	
55		730, 388, 1152, 354
56],	
57	[
58		502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0,
	114,	
59		308, 650, 274, 844
60],	
61	[
62		388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114,
	0, 194,	
63		536, 388, 730
64],	
65	[
66		354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308,
	194, 0,	
67		342, 422, 536
68],	
69	[
70		468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650,
	536,	
71		342, 0, 764, 194
72],	
73	[
74		776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152,
	274,	
75		388, 422, 764, 0, 798
76],	
77	[
78		662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844,
	730,	

```

79         536, 194, 798, 0
80     ],
81 ]
82 #每个客户的需求
83 data['demands'] = [0, 1, 1, 2, 4, 2, 4, 8, 8, 1, 2, 1, 2, 4, 4, 8, 8]
84 #每个货车的货物装载容量
85 data['vehicle_capacities'] = [15, 15, 15, 15]
86 data['num_vehicles'] = 4
87 data['depot'] = 0
88 return data
89
90
91 def print_solution(data, manager, routing, solution):
92     """Prints solution on console."""
93     total_distance = 0
94     total_load = 0
95     for vehicle_id in range(data['num_vehicles']):
96         index = routing.Start(vehicle_id)
97         plan_output = 'Route for vehicle {}: \n'.format(vehicle_id)
98         route_distance = 0
99         route_load = 0
100         while not routing.IsEnd(index):
101             node_index = manager.IndexToNode(index)
102             route_load += data['demands'][node_index]
103             plan_output += ' {0} Load({1}) -> '.format(node_index,
route_load)
104             previous_index = index
105             index = solution.Value(routing.NextVar(index))
106             route_distance += routing.GetArcCostForVehicle(
previous_index, index, vehicle_id)
107             plan_output += ' {0}
Load({1}) \n'.format(manager.IndexToNode(index),
route_load)
108         plan_output += 'Distance of the route:
{0}m \n'.format(route_distance)
109         plan_output += 'Load of the route: {0} \n'.format(route_load)
110         print(plan_output)
111         total_distance += route_distance
112         total_load += route_load
113     print('Total distance of all routes: {0}m'.format(total_distance))
114     print('Total load of all routes: {0}'.format(total_load))
115
116
117
118
119 def main():
120     """Solve the CVRP problem."""
121     # Instantiate the data problem.
122     data = create_data_model()
123
124     # Create the routing index manager.
125     manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
data['num_vehicles'],
data['depot'])
126
127
128     # Create Routing Model.
129     routing = pywrapcp.RoutingModel(manager)
130
131
132     # Create and register a transit callback.

```

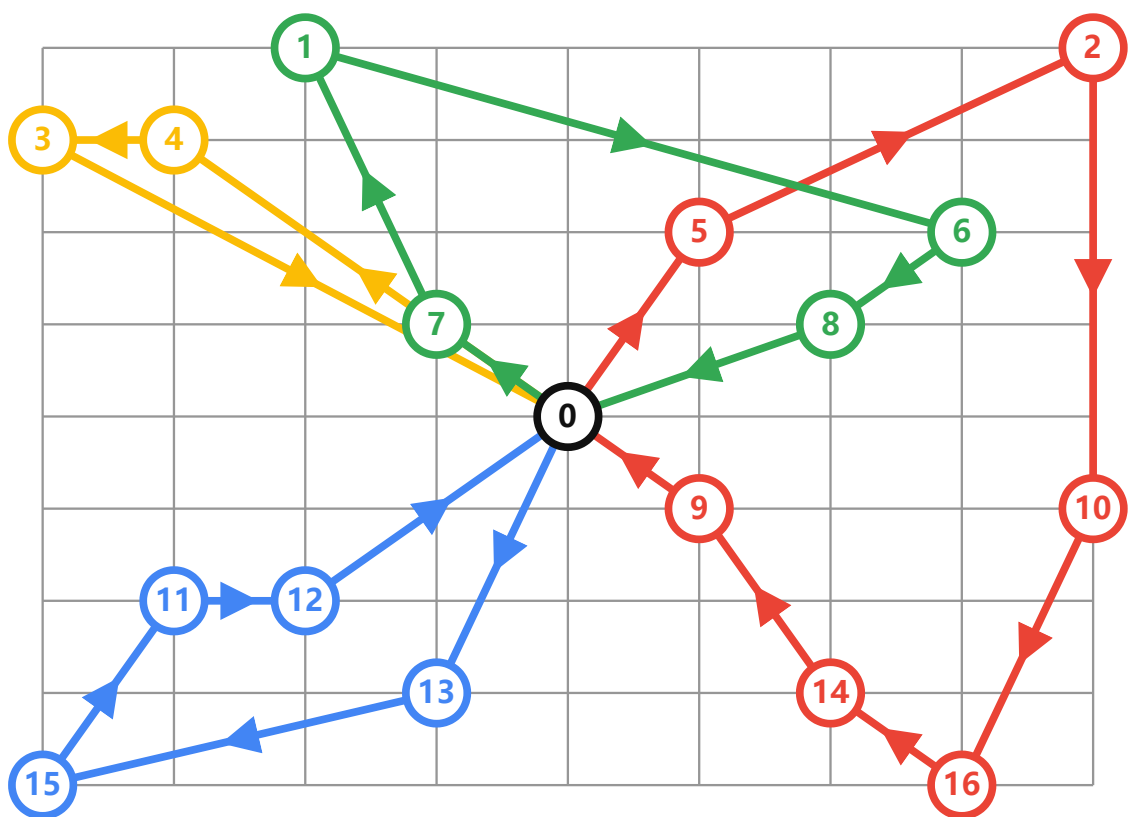
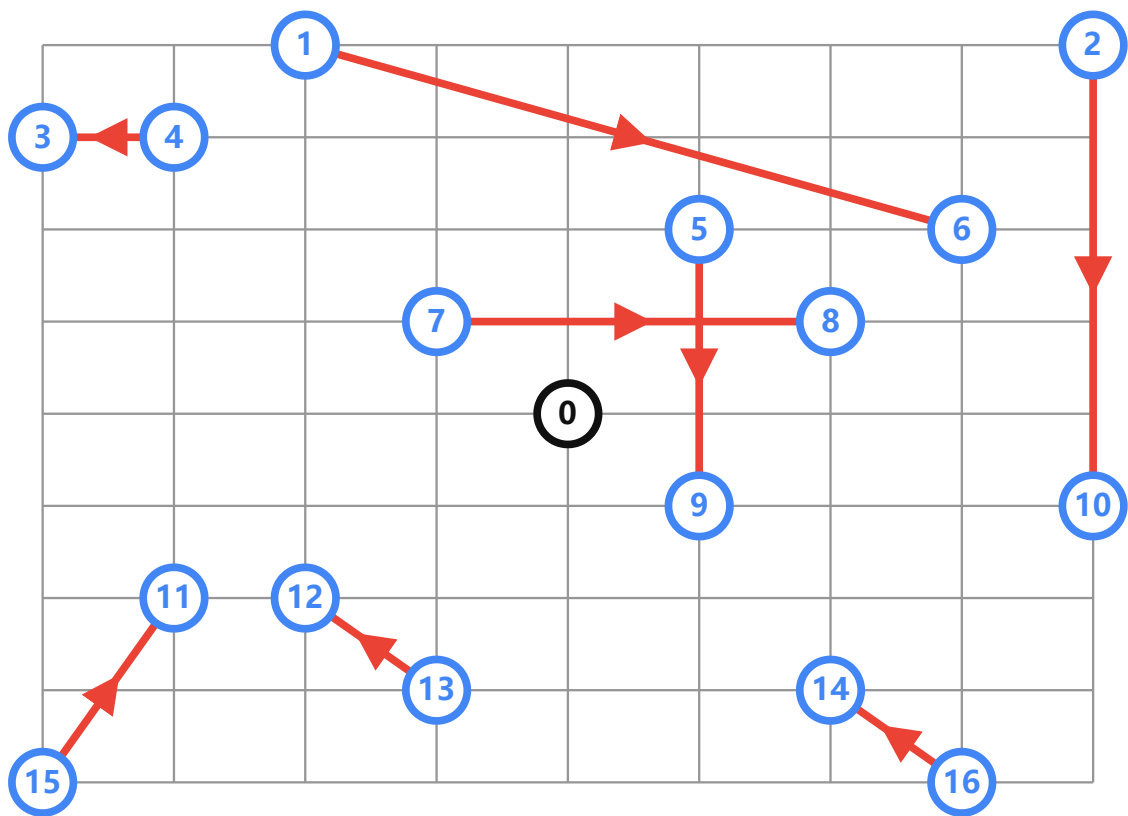
```

133     def distance_callback(from_index, to_index):
134         """Returns the distance between the two nodes."""
135         # Convert from routing variable Index to distance matrix
NodeIndex.
136         from_node = manager.IndexToNode(from_index)
137         to_node = manager.IndexToNode(to_index)
138         return data['distance_matrix'][from_node][to_node]
139
140     transit_callback_index =
routing.RegisterTransitCallback(distance_callback)
141
142     # Define cost of each arc.
143     routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
144
145
146     # Add Capacity constraint.
147     def demand_callback(from_index):
148         """Returns the demand of the node."""
149         # Convert from routing variable Index to demands NodeIndex.
150         from_node = manager.IndexToNode(from_index)
151         return data['demands'][from_node]
152
153     demand_callback_index = routing.RegisterUnaryTransitCallback(
154         demand_callback)
155     routing.AddDimensionWithVehicleCapacity(
156         demand_callback_index,
157         0, # null capacity slack
158         data['vehicle_capacities'], # vehicle maximum capacities
159         True, # start cumul to zero
160         'Capacity')
161
162     # Setting first solution heuristic.
163     search_parameters = pywrapcp.DefaultRoutingSearchParameters()
164     search_parameters.first_solution_strategy = (
165         routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
166
167     # Solve the problem.
168     solution = routing.SolveWithParameters(search_parameters)
169
170     # Print solution on console.
171     if solution:
172         print_solution(data, manager, routing, solution)
173
174
175 if __name__ == '__main__':
176     main()

```

pickups and deliveries问题

每一个需要运送的货物都有自己对应的pickup起点和delivery终点。我们需要从depot出发，依次将货物从起点送到终点。



```

Objective: 226116
Route for vehicle 0:
 0 -> 13 -> 15 -> 11 -> 12 -> 0
Distance of the route: 1552m

Route for vehicle 1:
 0 -> 5 -> 2 -> 10 -> 16 -> 14 -> 9 -> 0
Distance of the route: 2192m

Route for vehicle 2:
 0 -> 4 -> 3 -> 0
Distance of the route: 1392m

Route for vehicle 3:
 0 -> 7 -> 1 -> 6 -> 8 -> 0
Distance of the route: 1780m

Total Distance of all routes: 6916m

```

```

1  """Simple Pickup Delivery Problem (PDP)."""
2
3  from __future__ import print_function
4  from ortools.constraint_solver import routing_enums_pb2
5  from ortools.constraint_solver import pywrapcp
6
7
8  def create_data_model():
9      """Stores the data for the problem."""
10     data = {}
11     data['distance_matrix'] = [
12         [
13             0, 548, 776, 696, 582, 274, 502, 194, 308, 194, 536, 502, 388,
14             354,
15             468, 776, 662
16         ],
17         [
18             548, 0, 684, 308, 194, 502, 730, 354, 696, 742, 1084, 594,
19             480, 674,
20             1016, 868, 1210
21         ],
22         [
23             776, 684, 0, 992, 878, 502, 274, 810, 468, 742, 400, 1278,
24             1164,
25             1130, 788, 1552, 754
26         ],
27         [
28             696, 308, 992, 0, 114, 650, 878, 502, 844, 890, 1232, 514,
29             628, 822,
30             1164, 560, 1358
31         ],
32         [

```

29		582, 194, 878, 114, 0, 536, 764, 388, 730, 776, 1118, 400,
	514, 708,	
30		1050, 674, 1244
31],	
32	[
33		274, 502, 502, 650, 536, 0, 228, 308, 194, 240, 582, 776, 662,
	628,	
34		514, 1050, 708
35],	
36	[
37		502, 730, 274, 878, 764, 228, 0, 536, 194, 468, 354, 1004,
	890, 856,	
38		514, 1278, 480
39],	
40	[
41		194, 354, 810, 502, 388, 308, 536, 0, 342, 388, 730, 468, 354,
	320,	
42		662, 742, 856
43],	
44	[
45		308, 696, 468, 844, 730, 194, 194, 342, 0, 274, 388, 810, 696,
	662,	
46		320, 1084, 514
47],	
48	[
49		194, 742, 742, 890, 776, 240, 468, 388, 274, 0, 342, 536, 422,
	388,	
50		274, 810, 468
51],	
52	[
53		536, 1084, 400, 1232, 1118, 582, 354, 730, 388, 342, 0, 878,
	764,	
54		730, 388, 1152, 354
55],	
56	[
57		502, 594, 1278, 514, 400, 776, 1004, 468, 810, 536, 878, 0,
	114,	
58		308, 650, 274, 844
59],	
60	[
61		388, 480, 1164, 628, 514, 662, 890, 354, 696, 422, 764, 114,
	0, 194,	
62		536, 388, 730
63],	
64	[
65		354, 674, 1130, 822, 708, 628, 856, 320, 662, 388, 730, 308,
	194, 0,	
66		342, 422, 536
67],	
68	[
69		468, 1016, 788, 1164, 1050, 514, 514, 662, 320, 274, 388, 650,
	536,	
70		342, 0, 764, 194
71],	
72	[
73		776, 868, 1552, 560, 674, 1050, 1278, 742, 1084, 810, 1152,
	274,	
74		388, 422, 764, 0, 798

```

75         ],
76         [
77             662, 1210, 754, 1358, 1244, 708, 480, 856, 514, 468, 354, 844,
78             730,
79             536, 194, 798, 0
80         ],
81         #需要取货送货的节点对
82         data['pickups_deliveries'] = [
83             [1, 6],
84             [2, 10],
85             [4, 3],
86             [5, 9],
87             [7, 8],
88             [15, 11],
89             [13, 12],
90             [16, 14],
91         ]
92         data['num_vehicles'] = 4
93         data['depot'] = 0
94         return data
95
96
97 def print_solution(data, manager, routing, solution):
98     """Prints solution on console."""
99     total_distance = 0
100    for vehicle_id in range(data['num_vehicles']):
101        index = routing.Start(vehicle_id)
102        plan_output = 'Route for vehicle {}: \n'.format(vehicle_id)
103        route_distance = 0
104        while not routing.IsEnd(index):
105            plan_output += ' {} -> '.format(manager.IndexToNode(index))
106            previous_index = index
107            index = solution.Value(routing.NextVar(index))
108            route_distance += routing.GetArcCostForVehicle(
109                previous_index, index, vehicle_id)
110            plan_output += ' {} \n'.format(manager.IndexToNode(index))
111            plan_output += 'Distance of the route:
112            {}m \n'.format(route_distance)
113            print(plan_output)
114            total_distance += route_distance
115        print('Total Distance of all routes: {}m'.format(total_distance))
116
117 def main():
118     """Entry point of the program."""
119     # Instantiate the data problem.
120     data = create_data_model()
121
122     # Create the routing index manager.
123     manager = pywrapcp.RoutingIndexManager(len(data['distance_matrix']),
124                                             data['num_vehicles'],
125                                             data['depot'])
126
127     # Create Routing Model.
128     routing = pywrapcp.RoutingModel(manager)
129

```

```

130     # Define cost of each arc.
131     def distance_callback(from_index, to_index):
132         """Returns the manhattan distance between the two nodes."""
133         # Convert from routing variable Index to distance matrix
NodeIndex.
134         from_node = manager.IndexToNode(from_index)
135         to_node = manager.IndexToNode(to_index)
136         return data['distance_matrix'][from_node][to_node]
137
138     transit_callback_index =
routing.RegisterTransitCallback(distance_callback)
139     routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
140
141     # Add Distance constraint.
142     dimension_name = 'Distance'
143     routing.AddDimension(
144         transit_callback_index,
145         0, # no slack
146         3000, # vehicle maximum travel distance
147         True, # start cumul to zero
148         dimension_name)
149     distance_dimension = routing.GetDimensionOrDie(dimension_name)
150     distance_dimension.SetGlobalSpanCostCoefficient(100)
151
152     # Define Transportation Requests.
153     for request in data['pickups_deliveries']:
154         pickup_index = manager.NodeToIndex(request[0])
155         delivery_index = manager.NodeToIndex(request[1])
156         routing.AddPickupAndDelivery(pickup_index, delivery_index)
157         #单个货车实现取货送货
158         routing.solver().Add(
159             routing.VehicleVar(pickup_index) == routing.VehicleVar(
160                 delivery_index))
161         #取货必须先于送货
162         routing.solver().Add(
163             distance_dimension.CumulVar(pickup_index) <=
164             distance_dimension.CumulVar(delivery_index))
165
166     # Setting first solution heuristic.
167     search_parameters = pywrapcp.DefaultRoutingSearchParameters()
168     search_parameters.first_solution_strategy = (
169 routing_enums_pb2.FirstSolutionStrategy.PARALLEL_CHEAPEST_INSERTION)
170
171     # Solve the problem.
172     solution = routing.SolveWithParameters(search_parameters)
173
174     # Print solution on console.
175     if solution:
176         print_solution(data, manager, routing, solution)
177
178
179 if __name__ == '__main__':
180     main()

```


VRPTW (VRP with Time Windows) 问题

当VRP问题有到达时间的约束条件时，问题变为VRPTW (VRP with Time Windows)

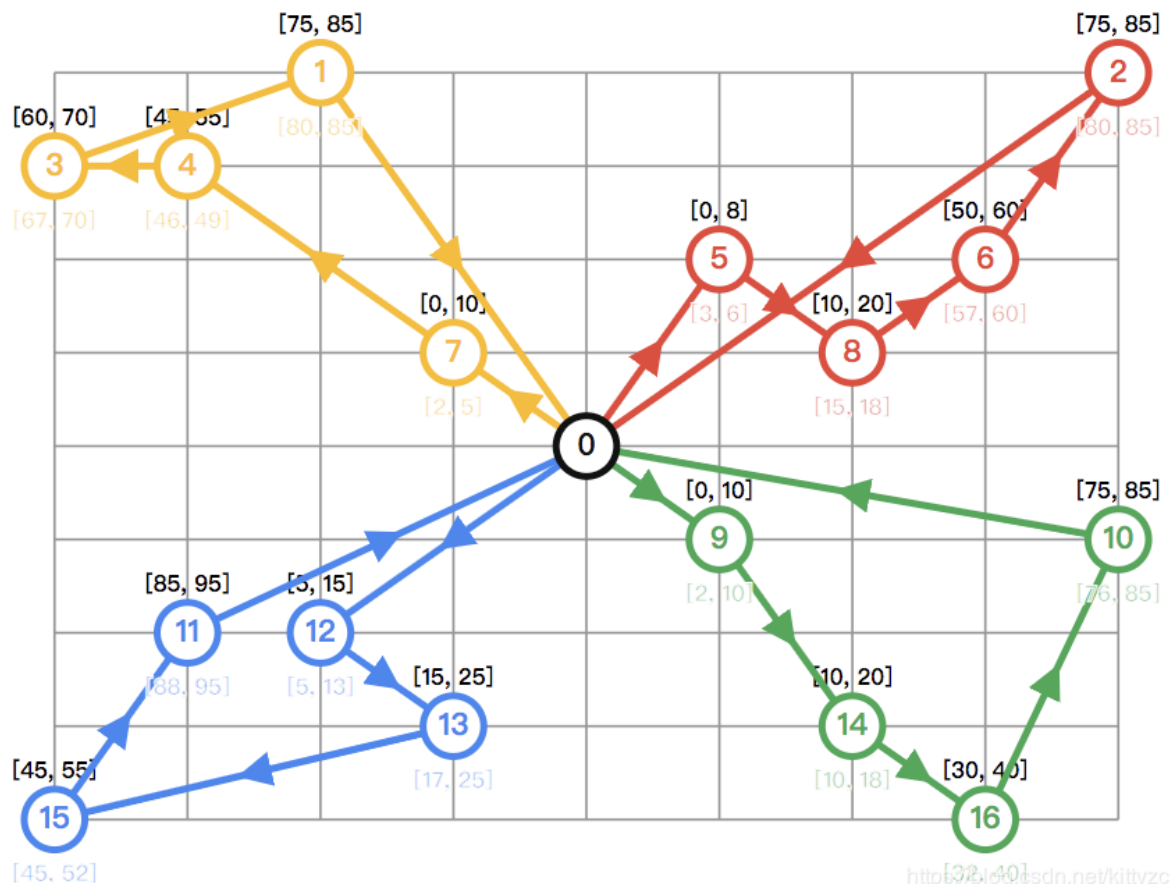
Route for vehicle 0:
0 Time(0,0) -> 9 Time(2,3) -> 14 Time(7,8) -> 16 Time(11,11) -> 0 Time(18,18)
Time of the route: 18min

Route for vehicle 1:
0 Time(0,0) -> 7 Time(2,4) -> 1 Time(7,11) -> 4 Time(10,13) -> 3 Time(16,16) -> 0 Time(24,24)
Time of the route: 24min

Route for vehicle 2:
0 Time(0,0) -> 12 Time(4,4) -> 13 Time(6,6) -> 15 Time(11,11) -> 11 Time(14,14) -> 0 Time(20,20)
Time of the route: 20min

Route for vehicle 3:
0 Time(0,0) -> 5 Time(3,3) -> 8 Time(5,5) -> 6 Time(7,7) -> 2 Time(10,10) -> 10 Time(14,14) -> 0 Time(20,20)
Time of the route: 20min

Total time of all routes: 82min



```
1 """Vehicles Routing Problem (VRP) with Time windows."""
2
3 from __future__ import print_function
4 from ortools.constraint_solver import routing_enums_pb2
5 from ortools.constraint_solver import pywrapcp
6
7
8 def create_data_model():
9     """Stores the data for the problem."""
10     data = {}
```

```

11 #时间矩阵
12 data['time_matrix'] = [
13     [0, 6, 9, 8, 7, 3, 6, 2, 3, 2, 6, 6, 4, 4, 5, 9, 7],
14     [6, 0, 8, 3, 2, 6, 8, 4, 8, 8, 13, 7, 5, 8, 12, 10, 14],
15     [9, 8, 0, 11, 10, 6, 3, 9, 5, 8, 4, 15, 14, 13, 9, 18, 9],
16     [8, 3, 11, 0, 1, 7, 10, 6, 10, 10, 14, 6, 7, 9, 14, 6, 16],
17     [7, 2, 10, 1, 0, 6, 9, 4, 8, 9, 13, 4, 6, 8, 12, 8, 14],
18     [3, 6, 6, 7, 6, 0, 2, 3, 2, 2, 7, 9, 7, 7, 6, 12, 8],
19     [6, 8, 3, 10, 9, 2, 0, 6, 2, 5, 4, 12, 10, 10, 6, 15, 5],
20     [2, 4, 9, 6, 4, 3, 6, 0, 4, 4, 8, 5, 4, 3, 7, 8, 10],
21     [3, 8, 5, 10, 8, 2, 2, 4, 0, 3, 4, 9, 8, 7, 3, 13, 6],
22     [2, 8, 8, 10, 9, 2, 5, 4, 3, 0, 4, 6, 5, 4, 3, 9, 5],
23     [6, 13, 4, 14, 13, 7, 4, 8, 4, 4, 0, 10, 9, 8, 4, 13, 4],
24     [6, 7, 15, 6, 4, 9, 12, 5, 9, 6, 10, 0, 1, 3, 7, 3, 10],
25     [4, 5, 14, 7, 6, 7, 10, 4, 8, 5, 9, 1, 0, 2, 6, 4, 8],
26     [4, 8, 13, 9, 8, 7, 10, 3, 7, 4, 8, 3, 2, 0, 4, 5, 6],
27     [5, 12, 9, 14, 12, 6, 6, 7, 3, 3, 4, 7, 6, 4, 0, 9, 2],
28     [9, 10, 18, 6, 8, 12, 15, 8, 13, 9, 13, 3, 4, 5, 9, 0, 9],
29     [7, 14, 9, 16, 14, 8, 5, 10, 6, 5, 4, 10, 8, 6, 2, 9, 0],
30 ]
31 #每个客户要求的时间窗
32 data['time_windows'] = [
33     (0, 5), # depot
34     (7, 12), # 1
35     (10, 15), # 2
36     (16, 18), # 3
37     (10, 13), # 4
38     (0, 5), # 5
39     (5, 10), # 6
40     (0, 4), # 7
41     (5, 10), # 8
42     (0, 3), # 9
43     (10, 16), # 10
44     (10, 15), # 11
45     (0, 5), # 12
46     (5, 10), # 13
47     (7, 8), # 14
48     (10, 15), # 15
49     (11, 15), # 16
50 ]
51 data['num_vehicles'] = 4
52 data['depot'] = 0
53 return data
54
55
56 def print_solution(data, manager, routing, solution):
57     """Prints solution on console."""
58     time_dimension = routing.GetDimensionOrDie('Time')
59     total_time = 0
60     for vehicle_id in range(data['num_vehicles']):
61         index = routing.Start(vehicle_id)
62         plan_output = 'Route for vehicle {}: \n'.format(vehicle_id)
63         while not routing.IsEnd(index):
64             time_var = time_dimension.CumulVar(index)
65             plan_output += '{0} Time({1},{2}) -> '.format(
66                 manager.IndexToNode(index), solution.Min(time_var),
67                 solution.Max(time_var))
68             index = solution.Value(routing.NextVar(index))

```

```

69         time_var = time_dimension.CumulVar(index)
70         plan_output += '{0} Time({1},
{2})\n'.format(manager.IndexToNode(index),
71
72         solution.Min(time_var),
73
74         solution.Max(time_var))
75         plan_output += 'Time of the route: {}min\n'.format(
76             solution.Min(time_var))
77         print(plan_output)
78         total_time += solution.Min(time_var)
79         print('Total time of all routes: {}min'.format(total_time))
80
81 def main():
82     """Solve the VRP with time windows."""
83     # Instantiate the data problem.
84     data = create_data_model()
85
86     # Create the routing index manager.
87     manager = pywrapcp.RoutingIndexManager(len(data['time_matrix']),
88                                             data['num_vehicles'],
89                                             data['depot'])
90
91     # Create Routing Model.
92     routing = pywrapcp.RoutingModel(manager)
93
94     # Create and register a transit callback.
95     def time_callback(from_index, to_index):
96         """Returns the travel time between the two nodes."""
97         # Convert from routing variable Index to time matrix NodeIndex.
98         from_node = manager.IndexToNode(from_index)
99         to_node = manager.IndexToNode(to_index)
100         return data['time_matrix'][from_node][to_node]
101
102     transit_callback_index =
103     routing.RegisterTransitCallback(time_callback)
104
105     # Define cost of each arc.
106     routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
107
108     # Add Time Windows constraint.
109     time = 'Time'
110     routing.AddDimension(
111         transit_callback_index,
112         #单个节点最大等待时间
113         30, # allow waiting time
114         #每个货车最大运送时间
115         30, # maximum time per vehicle
116         False, # Don't force start cumul to zero.
117         time)
118     time_dimension = routing.GetDimensionOrDie(time)
119     # Add time window constraints for each location except depot.
120     for location_idx, time_window in enumerate(data['time_windows']):
121         if location_idx == 0:
122             continue
123         index = manager.NodeToIndex(location_idx)

```

```

122     #硬时间窗口
123     time_dimension.CumulVar(index).SetRange(time_window[0], time_window[1])
124     # Add time window constraints for each vehicle start node.
125     for vehicle_id in range(data['num_vehicles']):
126         index = routing.Start(vehicle_id)
127         time_dimension.CumulVar(index).SetRange(data['time_windows'][0]
128 [0],
129                                             data['time_windows'][0]
130 [1])
131
132     # Instantiate route start and end times to produce feasible times.
133     for i in range(data['num_vehicles']):
134         routing.AddVariableMinimizedByFinalizer(
135             time_dimension.CumulVar(routing.Start(i)))
136         routing.AddVariableMinimizedByFinalizer(
137             time_dimension.CumulVar(routing.End(i)))
138
139     # Setting first solution heuristic.
140     search_parameters = pywrapcp.DefaultRoutingSearchParameters()
141     search_parameters.first_solution_strategy = (
142         routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
143
144     # Solve the problem.
145     solution = routing.SolveWithParameters(search_parameters)
146
147     # Print solution on console.
148     if solution:
149         print_solution(data, manager, routing, solution)
150
151 if __name__ == '__main__':
152     main()

```