

# Example 1

## JSprit – Single Depot

```
1 public class SmallWetteraul {
2
3     public static final int UNITS = 0;
4
5     public static void main(String[] args) {
6
7         VehicleRoutingProblem.Builder vrpBuilder
8             = VehicleRoutingProblem.Builder.newInstance();
9
10        Location n0 = Location.newInstance(3, -4);
11
12        Service n1 = Service.Builder.newInstance("n1")
13            .setLocation(Location.newInstance(-6, 8))
14            .addSizeDimension(UNITS, 15)
15            .setTimeWindow(TimeWindow.newInstance(30, 45))
16            .setServiceTime(3)
17            .build();
18
19        Service n2 = Service.Builder.newInstance("n2")
20            .setLocation(Location.newInstance(-8, -8))
21            .addSizeDimension(UNITS, 10)
22            .setTimeWindow(TimeWindow.newInstance(30, 60))
23            .setServiceTime(7)
24            .build();
25
26        Service n3 = Service.Builder.newInstance("n3")
27            .setLocation(Location.newInstance(11, -3))
28            .addSizeDimension(UNITS, 15)
29            .setTimeWindow(TimeWindow.newInstance(15, 30))
30            .setServiceTime(12)
31            .build();
32
33        Location n4 = Location.newInstance(-12, 8);
34
35        vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3));
36
37        IncompleteCostMatrix.Builder costMatrixBuilder =
38            IncompleteCostMatrix.Builder.newInstance();
39
40        // First variant used to add distances
41        costMatrixBuilder.addTransportTime(n0,n1.getLocation(),
42            slowWayFunction(n0,n1.getLocation()));
43
44        // Second (perhaps more easy variant)
45        Set<RelationKey> normalWays = new HashSet<>();
46        normalWays.add(RelationKey.newKey(n1, n4));
47        normalWays.add(RelationKey.newKey(n1, n2));
48        normalWays.add(RelationKey.newKey(n1, n3));
49
50        for(RelationKey key : normalWays)
51            costMatrixBuilder.addTransportTime(key.from, key.to, normalWayFunction(key.from, key.to));
52
53        Set<RelationKey> fastWays = new HashSet<>();
54        fastWays.add(RelationKey.newKey(n0,n2));
55        fastWays.add(RelationKey.newKey(n0, n3));
56        fastWays.add(RelationKey.newKey(n2, n4));
57
58        for(RelationKey key : fastWays)
59            costMatrixBuilder.addTransportTime(key.from, key.to, fastWayFunction(key.from, key.to));
60
61        costMatrixBuilder.completeTransportTimeMatrix();
62        IncompleteCostMatrix cm = costMatrixBuilder.build();
63        vrpBuilder.setRoutingCost(cm);
64
65        // Vehicle type definition
66        VehicleType vehicleType = VehicleTypeImpl.Builder.newInstance("unitVehicleType")
67            .addCapacityDimension(UNITS, 200).build();
68
69        // Vehicle instance definition
70        VehicleImpl vehicleInstance = VehicleImpl.Builder.newInstance("unitVehicleInstance")
71            .setType(vehicleType)
72            .setLatestArrival(300)
73            .setStartLocation(n0)
74            .build();
75
76        // Adding vehicle instance to the problem
77        vrpBuilder.addVehicle(vehicleInstance);
78
79        VehicleRoutingProblem vrp = vrpBuilder.build();
80    }
81
82    public static double fastWayFunction(Location end1, Location end2) {
83        return 0.5 * EuclideanDistanceCalculator
84            .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
85    }
86
87    public static double normalWayFunction(Location end1, Location end2) {
88        return EuclideanDistanceCalculator
89            .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
90    }
91
92    public static double slowWayFunction(Location end1, Location end2) {
93        return 2 * EuclideanDistanceCalculator
94            .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1;
95    }
96 }
```

## Example 2

### JSprit – Multiple Depot

```
1 public class SmallWetterau2 {
2
3     public static final int UNITS_FOOD = 0;
4     public static final int UNITS_BEVERAGE = 1;
5
6     public static void main(String[] args) {
7
8         VehicleRoutingProblem.Builder vrpBuilder
9             = VehicleRoutingProblem.Builder.newInstance();
10
11         Location n0 = Location.newInstance(3, -4);
12
13         Delivery n1 = Delivery.Builder.newInstance("n1")
14             .setLocation(Location.newInstance(-6, 8)).addSizeDimension(UNITS_BEVERAGE, 15)
15             .setTimeWindow(TimeWindow.newInstance(30, 45)).setServiceTime(3)
16             .build();
17
18         Delivery n2 = Delivery.Builder.newInstance("n2")
19             .setLocation(Location.newInstance(-8, -8)).addSizeDimension(UNITS_FOOD, 10)
20             .setTimeWindow(TimeWindow.newInstance(30, 60)).setServiceTime(7).build();
21
22         Delivery n3 = Delivery.Builder.newInstance("n3")
23             .setLocation(Location.newInstance(11, -3)).addSizeDimension(UNITS_FOOD, 15)
24             .setTimeWindow(TimeWindow.newInstance(15, 30)).setServiceTime(12).build();
25
26         Location n4 = Location.newInstance(-12, 8);
27
28         vrpBuilder.addAllJobs(Arrays.asList(n1, n2, n3));
29
30         IncompleteCostMatrix.Builder costMatrixBuilder =
31             IncompleteCostMatrix.Builder.newInstance();
32
33         // First variant used to add distances
34         costMatrixBuilder.addTransportTime(n0, n1.getLocation(),
35             slowWayFunction(n0, n1.getLocation()));
36
37         // Second (perhaps more easy variant)
38         Set<RelationKey> normalWays = new HashSet<>();
39         normalWays.add(RelationKey.newKey(n1, n4));
40         normalWays.add(RelationKey.newKey(n1, n2));
41         normalWays.add(RelationKey.newKey(n1, n3));
42
43         for(RelationKey key : normalWays)
44             costMatrixBuilder.addTransportTime(key.from, key.to, normalWayFunction(key.from, key.to));
45
46         // Second (perhaps more easy variant)
47         Set<RelationKey> fastWays = new HashSet<>();
48         fastWays.add(RelationKey.newKey(n0, n2));
49         fastWays.add(RelationKey.newKey(n0, n3));
50         fastWays.add(RelationKey.newKey(n2, n4));
51
52         for(RelationKey key : fastWays)
53             costMatrixBuilder.addTransportTime(key.from, key.to, fastWayFunction(key.from, key.to));
54
55         costMatrixBuilder.completeTransportTimeMatrix();
56         IncompleteCostMatrix cm = costMatrixBuilder.build();
57         vrpBuilder.setRoutingCost(cm);
58
59         // Vehicle food type definition
60         VehicleType vehicleTypeF = VehicleTypeImpl.Builder.newInstance("unitVehicleTypeF")
61             .addCapacityDimension(UNITS_FOOD, 200).build();
62         // Vehicle food instance definition
63         VehicleImpl vehicleInstanceF = VehicleImpl.Builder.newInstance("unitVehicleInstanceF")
64             .setType(vehicleTypeF).setLatestArrival(300).setStartLocation(n0).build();
65         // Adding vehicle food instance to the problem
66         vrpBuilder.addVehicle(vehicleInstanceF);
67
68         // Vehicle beverage type definition
69         VehicleType vehicleTypeB = VehicleTypeImpl.Builder.newInstance("unitVehicleTypeB")
70             .addCapacityDimension(UNITS_BEVERAGE, 150).build();
71         // Vehicle food instance definition
72         VehicleImpl vehicleInstanceB = VehicleImpl.Builder.newInstance("unitVehicleInstanceB")
73             .setType(vehicleTypeB).setLatestArrival(300).setStartLocation(n4).build();
74         // Adding vehicle beverage instance to the problem
75         vrpBuilder.addVehicle(vehicleInstanceB);
76
77         VehicleRoutingProblem vrp = vrpBuilder.build();
78     }
79
80     public static double fastWayFunction(Location end1, Location end2) {
81         return 0.5 * EuclideanDistanceCalculator
82             .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
83     }
84
85     public static double normalWayFunction(Location end1, Location end2) {
86         return EuclideanDistanceCalculator
87             .calculateDistance(end1.getCoordinate(), end2.getCoordinate());
88     }
89
90     public static double slowWayFunction(Location end1, Location end2) {
91         return 2 * EuclideanDistanceCalculator
92             .calculateDistance(end1.getCoordinate(), end2.getCoordinate()) + 1;
93     }
94 }
```

## Example 3

### Athos – Single Depot

```
1 model WetterauOrders1
2 products
3   unit weight 1.0
4 functions
5   durationFunction slowWayFunction 2 * length + 1
6   durationFunction normalWayFunction length
7   durationFunction fastWayFunction 0.5 * length
8 network
9   nodes
10    n0 at (3, -4) isDepot unit sprouts vehicles customers n1, n2, n3 at 0 latestTime 300
11    n1 at (-6, 8) hasDemand unit units 15 earliestTime 30 latestTime 45 serviceTime 3
12    n2 at (-8, -8) hasDemand unit units 10 earliestTime 30 latestTime 60 serviceTime 7
13    n3 at (11, -3) hasDemand unit units 15 earliestTime 15 latestTime 30 serviceTime 12
14    n4 at (-12, 8)
15   edges
16    s1 from n0 to n1 function slowWayFunction
17    group normalWayGroup function normalWayFunction members
18      w1 from n1 to n4
19      w2 from n1 to n2
20      w3 from n1 to n3
21    group fastWayGroup function fastWayFunction members
22      f1 from n0 to n2
23      f2 from n0 to n3
24      f3 from n2 to n4
25 agentTypes
26   agentType vehicles maxWeight 200
27   behaviour awt awaitTour when finished do die
28   behaviour die vanish
```

## Example 4

### Athos – Multiple Depot

```
1 model WetterauOrders2
2 products
3   unitsFood weight 1.0
4   unitsBeverage weight 1.0
5 functions
6   durationFunction slowWayFunction 2 * length + 1
7   durationFunction normalWayFunction length
8   durationFunction fastWayFunction 0.5 * length
9 network
10  nodes
11    n0 at (3, -4) isDepot unitsFood sprouts vehiclesFood customers n2, n3 at 0 latestTime 300
12    n1 at (-6, 8) hasDemand unitsBeverage units 15 earliestTime 30 latestTime 45 serviceTime 3
13    n2 at (-8, -8) hasDemand unitsFood units 10 earliestTime 30 latestTime 60 serviceTime 7
14    n3 at (11, -3) hasDemand unitsFood units 15 earliestTime 15 latestTime 30 serviceTime 12
15    n4 at (-12, 8) isDepot unitsBeverage sprouts vehiclesBeverage customers n1 at 0 latestTime 300
16  edges
17    s1 from n0 to n1 function slowWayFunction
18    group normalWayGroup function normalWayFunction members
19      w1 from n1 to n4
20      w2 from n1 to n2
21      w3 from n1 to n3
22    group fastWayGroup function fastWayFunction members
23      f1 from n0 to n2
24      f2 from n0 to n3
25      f3 from n2 to n4
26 agentTypes
27   agentType vehiclesFood maxWeight 200
28     behaviour awt awaitTour when finished do die
29     behaviour die vanish
30   agentType vehiclesBeverage maxWeight 150
31     behaviour awt awaitTour when finished do die
32     behaviour die vanish
```