A heuristic approach for a special pick-up-and-delivery problem

Harald Mumm

April 30, 2007

Abstract

The task to find an optimal solution for our pick-up-and-delivery problem is NP-complete. Our heuristic approach is based on optimal solutions for vehicle routing problems with split delivery. With these solutions we construct pick-up's for those stages, where the trucks are not full. In this kind of computation the costs of transport depends on the sequence of computation for all factories. We are looking for the best sequence.
In further papers we want also to solve the problem of bring-back-tours. Every transport of goods in little plastic containers implies a transposed demand matrix to bring back the empties. From monday to thursday only those trucks, which are not full are used for this bring-back-tours. On friday all empties of the week are to bring back to the production factories.
If it is not necessary to use the own trucks of the company, we only use values of demands less then the truck capacity.

1. Introduction

In [2] we used an approach from [1] for a decision support tool in an application domain with delivery problems. In our paper we set out to take this approach for a more general task: Delivery-and-Pick-Up (in literature often used as pick-up-and-delivery) in the special case, that every location has to send and receive some goods.

Delivery-and-Pick-Up is always required, when the production program of a company is carried out in more than one factory and when in each of these factories (which have a certain and exclusive production program) all goods must be permanently available, although not all of these goods were produced there.

2. The Problem

A certain company operates with six factories in six different locations. All distances between two locations are described in a distances matrix like this:

Figure 1: An example for a distance-matrix

$$distances(km) = \begin{pmatrix} 0 & 88 & 303 & 170 & 357 & 353 \\ 88 & 0 & 339 & 123 & 335 & 305 \\ 303 & 339 & 0 & 302 & 521 & 178 \\ 170 & 123 & 302 & 0 & 249 & 234 \\ 357 & 335 & 521 & 249 & 0 & 453 \\ 353 & 305 & 178 & 234 & 453 & 0 \end{pmatrix}$$

We assume that the triangle inequality is valid.

Due to the fact that not all factories have the identical production program, every day an exchange of goods has to take place to make sure that all products are available in each factory. From each factory certain goods, which are not produced at the other five locations have to be brought there. As an example we present you the following matrix, that depicts the demands of each factory in relation to the others:

From the first location a set of 112 units has to be delivered to the second location. From the second location the set of 86 units shall be transported to the factory at the first location and so on. Note, that one truck has a 36 unit capacity.

Figure 2: An example for demands

$$demands = \begin{pmatrix} 0 & 112 & 43 & 21 & 70 & 20 \\ 86 & 0 & 40 & 34 & 50 & 45 \\ 10 & 33 & 0 & 118 & 28 & 74 \\ 44 & 0 & 39 & 0 & 39 & 54 \\ 95 & 0 & 38 & 31 & 0 & 43 \\ 20 & 41 & 31 & 51 & 23 & 0 \end{pmatrix}$$

3. A Heuristic Algorithm for a special Pick-up-and-delivery Problem

Our algorithm is based on the optimal solution described in [2] regading one factory as a depot and the other factories as customers.

Since the truck capacity is not fully used anymoreafter its first stage, it can again pick up goods to its next station, which functions again as depot. But we do not use this method transitively.

We use the idea, that the costs in whole depends on the order of simple calculations for all six factories with the algorithm described in [2]. We are looking for the best possible order.

(This does not mean, that the trucks have to depart at different times. If the calculation is ready all trucks at all six locations can start at the same time for example in the morning at 6 o'clock.)

The data type for a solution described in [2] is named `SolutionOfOneVrpsd`. It consists of many schedules for all used trucks. The data type for one schedules is named `SchedulerOfOneRoute`. It contains all stages (Etappen,Staionen) for one truck. One stage is described with the properties:

depotnumber , the number of the location, from where the truck delivers some goods to other locations,

fromnumber , the number of the location of the last stop

tonumber , the number of the following location of one stage

freecap , the free capacity of the truck at this stage.

onBoard , the real set of goods (in number of pallets) on the truck

deliveryset , the set of goods on the truck, which have to stay at the location with `tonumber` ,

truckcap , all trucks have a fixed capacity

When the truck reaches it first destination, it delivers some goods. Consequently, it is not fully loaded anymore, that means: `freecap >0`.

Now it can pick up a certain amount of goods depending on the next location's needs. In this case the location with the certain number, stored in the variable `fromnumber` plays the role of the depot for the location with the number stored in the variable `tonumber`.

Now the whole pseudocode for the used data types is presented:

```
Type PickUpAndDeliverySolution = Array[1..maxFactory] of
                                        SolutionOfOneVrpsd

Type SolutionOfOneVrpsd= Array[1..numberOfTrucks] of
                                    ScheduleOfOneRoute

Type ScheduleOfOneRoute = Array[1..numberOfStages] of
                                                Stage

Type Stage= Record (

depotnumber: 1..maxFactory
fromnumber : 1..maxFactory
tonumber   : 1..maxFactory
freecap    : unsignedInt
onBoard    : unsignedInt
delivery   : unsignedInt
truckcap   : unsignedInt)


Type AllDemands= Array[1..maxFactory] of Demands
Type Demands   = Array[1..maxFactory] of OneDemand
Type OneDemand = unsignedInt

Type Distances = Array[1..maxFactory] of Array[1..maxFactory]
                                                of unsignedInt
```

Now we show the code for the heuristic. The whole calculation runs in the following function. `scan ..` means a loop over the six values 1,2,3,4,5 and 6. Because in the function `solveSixVrpsp` there are pick-ups the matrix of demands has to be recover for the following calculation.

```
Function CostsPickUpAndDelivery6Factorys(
                        -> ad: Demands
                        -> di: Distances
                        -> tc: TruckCap): Costs

Variable adt: Demands
Variable minimumCosts: float

Variable result: SolutionOfOneVrpsd

Beginn
 adt = ad
```

```
 mimimumCosts:= bigNumber

 scan i1;scan i2;scan i3;scan i4;scan i5;scan i6
 Begin

  if (allUnequal(i1,i2,i3,i4,i5,i6)) then
    Begin
     costs= solveSixVrpsp(ad,di,tc,i1,i2,i3,i4,i5,i6,result );

     recover(ad);

     if (costs<minimumCosts) then minimumCosts:= costs
    End
 CostsPickUpAndDelivery := minimumCosts
End
```

This function needs another function `solveSixVrpsp`, which calculates six solutions of six vehicle routing problems with split delivery (vrpsp) . In such solutions we insert certain pick-ups, when the truck is not full. These pick-ups reduces the demands for the calculation of the next vrpsp.

```
Funktion solveSixVrpsp(-> ad: AllDemands
                       -> di: Distances
                       -> tc: TruckCap
                       -> i1,i2,i3,i4,i5,i6: unsignedInt
                       ): unsignedInt
Variable sov: SolutionOfOneVrpsd
Begin

   costs1:=solveOneVrpsp(ad,di,tc,i1,sov)
   //PickUps reduces the demands in ad
   r1= PickUp(ad,di,tc,sov);


   costs2:=solveOneVrpsp(ad,di,tc,i2,sov)
   r2= PickUp(ad,di,tc,sov);

   costs3:=solveOneVrpsp(ad,di,tc,i3,sov)
   r3= PickUp(ad,di,tc,sov);

   costs4:=solveOneVrpsp(ad,di,tc,i4,sov)
   r4= PickUp(ad,di,tc,sov);

   costs5:=solveOneVrpsp(ad,di,tc,i5,sov)
   r3= PickUp(ad,di,tc,sov);

   costs6:=solveOneVrpsp(ad,di,tc,i6,sov)
   r4= PickUp(ad,di,tc,sov);
```

```
   return costs1+costs2+costs3+costs4+costs5+costs6

End
```

If a truck is not full it can pick-up some goods for its next destination. Perhaps it can pick-up the whole demand of its next destination.

```
Function PickUp(<->ad : AllDemands
                 ->sov: SolutionOfOneVrpsd): unsignedInt
Variable result: insignedInt;
Variable soor  : SchedulerOfOneRoute
Variable result: unsignedInt

Begin
result := 0
for (int i1:=0; i1< max1;i1++)
Begin
soor = sov[i1]

for (int i2:=0; i2< max2 ;i2++)
Begin
  st = soor[i2]

if (st.fromnumber != st.depotnumber

    and st.freecap >0

    and ad[st.fromnumber][st.tonumber] > 0)  then

    if (ad[st.fromnumber][st.tonumber]<= st.freecap)
       then Beginn
              ad[st.fromnumber][st.tonumber]:=0
        result := result + ad[st.fromnumber][st.tonumber]
             End
       else Begin
             ad[st.fromnumber][st.tonumber]:=
             ad[st.fromnumber][st.tonumber] - st.freecap
             result := result + st.freecap
            End
 End
 End

 PickUp := result
End
```

4. An example for the optimal solution

We assume that the truck capacity is 100 and the matrix of demands looks symmetrically like this:

$$demands = \begin{pmatrix} - & 100 & 100 & 100 & 100 & 100 \\ 100 & - & 200 & 200 & 200 & 200 \\ 100 & 200 & - & 300 & 300 & 300 \\ 100 & 200 & 300 & - & 400 & 400 \\ 100 & 200 & 300 & 400 & - & 500 \\ 100 & 200 & 300 & 400 & 500 & - \end{pmatrix}$$

In this case our algorithm produces the optimal solution with the lowest costs for the company:

$$optimalcosts = 2 * \sum_{i=2}^{6} distances[1][i] + 4 * \sum_{i=3}^{6} distances[2][i] +$$

$$6 * \sum_{i=4}^{6} distances[3][i] +$$

$$+8 * \sum_{i=5}^{6} distances[4][i] + 10 * distances[5][6] = 21350km$$

5. Numerical Results

For the data above the best of the 720 possible orders of calculation is: 6, 5, 4, 1, 2, 3 with costs of 14369 km. That means, we have to compute at first the delivery-problem for the sixth factory, followed by the fifth, the fourth and so on. This result reduces the costs to 84% of the worst order costs.

In the following table you can also see the improvement of our solution respectively the number of trucks assuming that one truck has a 36 unit capacity.

Table 1: Improvement of number of trucks for demands in figure 2

factory no.	sum of units	# trucks theor.	# trucks heur.
6	166	5	5
5	207	6	5
4	176	5	3
1	266	8	4
2	255	8	4
3	263	8	2
	1333	38	19

For 1333 units you theoretical need 38 trucks with a 36 unit capacity. To solve the mentioned pick up and delivery problem, we merely have to

use 19 trucks.

6. Conclusions

Pick-Up-And-Delivery Problems with 6 locations are too large to be solved optimally. By our way it is possible to find a good solution in less than 10 minutes computing time. But we don't know how good our solution is. Therefore, the costs of the solution, produced by our algorithm needs to be compared with the cost of an optimal solution. This is the task for the future (see [3]).

Additionally, we have to solve the problem that our goods are transported in little plastic containers. One euro-pallet has for example the capacity of 64 plastic containers. At the end of the week all plastic containers must be at their origin.

If it is not necessary to use the own trucks of the company, we can calculate a new matrix of demands with the help of the modulo operation. We obtain the values of the new matrix as the rest by division the origin values through the truck capacity.

References

[1] Epelman, Marina and other: A Shortest Path Approach to the Multiple-Vehicle Routing Problem with Split Pick-Ups: Transportation Research, Part B 40(4):265-284, 2006.

[2] Mumm, Harald, Roeck, Hans: Developing operation and decision support tools for a split-delivery vehicle routing application domain , Conference Procedeedings BIR2006, October the 6th 2006, Kaunas, Lithuania

[3] Vazirani, ViJay: Approximation Algorithms, Springer-Verlag, 2001 .

About the author
Prof. Dr. rer. nat. Harald Mumm
Hochschule Wismar
University of Technology, Business and Design
Wismar Business School
Phillipp-Mueller-Strasse
Postbox 1210
23952 Wismar
Germany
Telefon: ++49 / (0)3841 / 753 450
Fax: ++49 / (0)3841 / 753 131
E-mail: harald.mumm@wi.hs-wismar.de