Operations Research B
A large neighborhood search approach to the dail-a-ride problem

Lars Burghardt, Alexander Wördekemper, Ahmad Hashemi

22.02.2017

Contents

1 Initial solution

For our initial solution we decided to use a sequential construction algorithm with a hill climbing algorithm introduced in [$HoMu12$]. We have chosen these algorithms because they seemed to produce fast and good initial solutions in the literature. We had to do some little adjustments because we could not open as many routes as we want because we have a fix number of vehicles. In line 1 of Algorithm 1 we first initialize an empty solution s. Then we repeat the following until all customers have been inserted into a route or the number of routes is equal to the number of vehicles we have in our instance. In the next step we initialize an empty route r (line 3). Then we start a loop through all unassigned customers. In line 5 we get a next random customer i and we insert the him at the end of our current route r. In line 7 we call the hill climbing algorithm to improve the route r. If the new route is feasible, we mark customer i as inserted, else we remove customer i from the route again. After the loop through all unassigned customers we add the current route r to the solution s in line 12.

The hill climbing see Algorithm 2 has given a route r which is tried to be improved. We repeat the following procedure until there is no improvement achieved in the previous pass. Start a loop through all possible pair of locations (line 3) and if the latter location is more urgent in its upper time window (line 4), we swap the current two locations in route r to get a new route r'. In line 6 we calculate both cost function of r and r'. If the new route has a lower value for his cost function value we set $r \leftarrow r'$.
The cost function is a sum over the route duration, the number of time window violations and the number of capacity violations. They are all weighted with the weights w1, w2 and w3 equal to 1.

Algorithm 1 Sequential construction

```
1:  Initialize an empty solution s
2:  repeat
3:      Initialize an empty route r
4:      for (All unassigned customers) do
5:          Get a random next customer i
6:          Insert the customer i at the end of the current route r
7:          Call HC (Algorithm 2) to improve route r
8:          if (route r is feasible) then
9:              Mark i as inserted
10:         else
11:             Remove i from route r
12:     Add route r to solution s
13: until (All customers have been inserted OR |s| = |Vehicles|)
```

Algorithm 2 Hill climbing

```
1: Given a route r
2: repeat
3:     for (Each possible pair of locations) do
4:         if (The latter location is more urgent in its upper time window bound) then
5:             Swap the current two locations in r to get a new route r'
6:             Calculate cost(r) and cost(r')
7:             if (cost(r) − cost(r') > 0) then
8:                 r ← r'
9: until (Done) {Stop when no improvement achieved in the previous pass}
```

1.1 Different approaches

As we found out that we did not found initial solutions for the larger instances we have tried some different approaches to solve this problem.

1.1.1 Removing the customer with the largest distance

The sequential construction algorithm adds random customers to the route and after that the route gets improved by the hill climbing algorithm. We removed the customer which we had just added if the new route is infeasible. Instead of removing the just added customer we tried to remove the customer which needed the most time to travel to in the route. For every customer and it's destination we summed the distance from the last node to the customer/destination and from the customer/destination to the next node. We removed the customer which needed the most time. In experiments we found out that this approach was no improvement compared to the one we had before.

1.1.2 Removing a random customer from the route

Another method we tried was to remove a random customer instead of the added customer from the route if the new route from the hill climbing algorithm is infeasible. The idea was to be able to remove customers from the route again after they have been successfully added before. One problem of our main approach is that when we add a customer which is bad for the route but still results in a feasible route we might not be able to add other customers because of the *bad* customer we already added. The random removing strategy should be able to remove such *bad* customers. In tests this approached performed much worse than the one we decided to take.

1.1.3 Parallel construction of the routes

Inspired by [$HoMu12$] we tried different approaches by constructing the routes in parallel. For this we initialized all possible routes at the beginning. Before we started adding the customers to the routes, we sorted the customers by several different criteria. Dependant on the criteria we chose a customer and add it to a route. We start with

the first route and use the hill climbing algorithm to improve the route. If the route is feasible we mark the customer as added. If it is not feasible we remove the customer again and try to add it to the next route. We tried the following methods to sort the customers:

- Ascending by lower time windows
- Ascending by upper time windows
- Ascending by lower time windows, if equal ascending by upper time windows
- Ascending by difference between lower and upper time windows

For the smaller instances all parallel construction methods seems to find a initial solution but none of them was faster than the sequential construction algorithm. Also they found even less solution for the larger instances.

2 Large neighborhood search

The large neighborhood search in Algorithm 3 tries to improve a given initial solution s. The approach was introduced in [$JaHe11$]. The algorithm gets as parameters $maxSize$ (the maximum number of customers to be removed), $range$ (to increase the neighborhood size progressively), $iterations$ (the number of iterations) and $probability$ (the probability to accept a worse solution).
In the first step we set the $current$ solution to the given solution s in line 2. Then we have three nested for loops in which we produce a new solutions new by removing randomly $i + j$ customers from the $current$ solution. We add the customer to a randomly selected route r in the new solution and always try to improve the route r by calling the hill climbing algorithm. After the insertion of the removed customer we generate a random number pr between 0 and 1 in line 10. If the new solution is feasible we check if the cost function of the new solution has a better value than the cost function of the $current$ solution. If this is the case or if pr is smaller than $probability$ we set $current = new$. Also we check if the cost function value of the $current$ solution is smaller than the cost function value of the given solution s (line 14). If it is better we update our solution s by setting it to our $current$ solution.

Algorithm 3 LNS $(maxSize, range, iterations, probability)$

```
1:  Given a solution s
2:  current ← s
3:  for (i ← 2; i ≤ masSize − range; i ← i + 1) do
4:      for (j ← 0; j ≤ range; j ← j + 1) do
5:          for (k ← 0; k < iterations; k ← k + 1) do
6:              new ← Remove randomly i + j customer in current
7:              for (All removed customer) do
8:                  Add customer to a randomly selected route r in new
9:                  Call HC (Algorithm 2) to improve route r
10:             pr ← random number between 0 and 1
11:             if (new is feasible solution) then
12:                 if (cost(new) < cost(current) OR pr < probability) then
13:                     current = new
14:                     if (cost(current) < cost(s)) then
15:                         s = current
```

3 Relevant work in the literature

In the literature nearly all possible meta heuristics have been tested for the dial-a-ride problem. [$JaHe11$] introduced a large neighborhood search for the problem. It seems to work well which is the reason why we decided to implement this meta heuristic. A different approach was introduced in [$CoLa03$] where they used a tabu search heuristic for the dial-a-ride problem. In [$Parragh\ et\ al.\ 10$] they used a variable neighborhood search to solve the problem and in [$Jørgensen\ et\ al.\ 07$] genetic algorithms were introduced to solve the dial-a-ride problem. Every approach has its advantages and disadvantaged but in most cases the large neighborhood search gets the best solutions in the literature.

4 How to compile and run the code

Our approach is implemented in C# using Microsoft Visual Studio Enterprise 2015 under Windows. To compile and run the code, simply open the solution file *ORB.DARP.sln* and run the project using *F5* (debug) or *Shift + F5* (no debug). A copy of Microsoft Visual Studio Enterprise 2015 can be downloaded at Dreamspark UPB.
Another method to run the code, is to open a command line window and enter the path to the compiled *orbdar.exe* with the additional parameters.

Example: *orbdar.exe [optional: 60] gen_10_2_75_8_10_1.darp*

5 Experimental investigation of our approaches components and performance

We tested different parameters for our hill climbing algorithm and for our LNS algorithm. In [$HoMu12$] they showed that the hill climbing algorithm performed best for large weights on time window violations. Because of this our manual experiments focused on large values for it. For time window violation weights smaller than 0.5 the performance decreased a lot. For too large values (greater than 0.95) it takes 4-5 times longer than with 0.80. The 10 customer instances need between 1 and 3 seconds to find an initial solution. The 20 customer instances vary a lot because of the random factor in our approach. It mostly takes between 1 and 10 minutes. The best result to find a solution in good runtime we used the weights 0.01 on route duration, 0.8 on time window violations and 0.19 on capacity violations. For the LNS we tested several different parameters. The maxSize is dependent of the size of the instance. We found out that $\lfloor number\ of\ customers/number\ of\ vehicles \rfloor$ worked well. We set the *range* parameter always to 1 to have more possibilities in searching large search space of the neighborhoods. With a larger number of iterations we found better solutions in the average but it also needed more runtime. Too small number of iterations resulted in the opposite. A value of 50000 performed best to find good solutions without taking too much time. This could be improved if we would stop the LNS when in a number of iterations x the solution did not improve more than p %.
We tried different values for the probability to accept worse solutions. Accepting worse solutions helps us to make jumps in the search space so that we can overcome local optima. We have chosen a value of 0.25 as for larger values we accepted too many worse solutions and with too small values we got stuck in a local optimum too often. In experimental results we tested the performance of the LNS. We found out that the above parameters performed best. Therefor we used for the hill climber wight of 0.01 for the route duration, 0.80 for the time window violations and 0.19 for the capacity violations. In the LNS we used the following parameters: $maxSize = \lfloor number\ of\ customers/number\ of\ vehicles \rfloor$, $range = 1$, $iterations = 5000$, $probability = 0.25$.
We tested every instance a 100 times. From the results in Table 1 and Figure 1 we can see that the LNS could always improve the initial solutions. The maximum improvement was for the instance gen_10_2_75_8_10_1 where we had 10 customers and 2 vehicles. The improvement was 31.87% in average. For some instances the average initial solution was nearly the best solution we could find. Probably the optimal solution is found for the 10 customer instances because all project groups found the same best solution in the leaderboard. Because of this the LNS could not further improve the solution. In the more significant 20 customer instances we have improvements between 8.07% and 27.20%. This shows that our LNS might perform quiet well. For even more significant results we should have tested it for larger instances, but our sequential construction algorithm was unable to find initial solutions for the larger instances.

Instance	Index	AVG initial solution	AVG best solution	AVG improvement
gen_10_2_75_8_10	1	1387	945	31.87%
	2	1229	1226	0.24%
	3	1231	1097	10.89%
	4	1339	1268	5.30%
	5	1175	1137	3.23%
gen_8_2_75_8_10	1	2524	2288	9.35%
	2	2417	1963	18.78%
	3	2256	2074	8.07%
	4	2982	2172	27.20%
	5	2188	1919	12.29%

Table 1: Performance comparison of our LNS approach

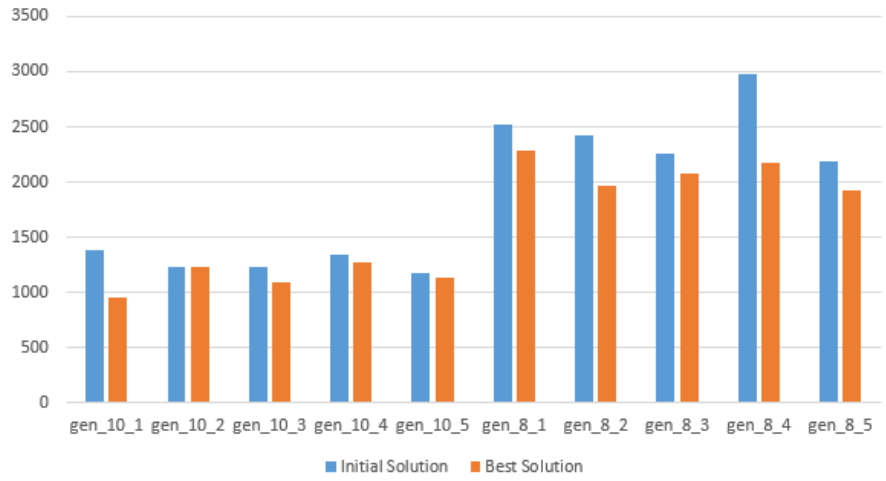

Figure 1: Performance of our LNS approach

6 Literature

[$HoMu12$] M. I. Hosny and C. L.Mumford, "Constructing initial solutions for the multiple vehicle pickup and delivery problem with time windows", Journal of King Saud University, Computer and Information Sciences, vol. 24, no. 1, pp. 59–69, 2012.

[$JaHe11$] S. Jain , P. Van Hentenryck, "Large neighborhood search for dial-a-ride problems", In: Principles and practice of constraint programming, Notes in computer science, vol. 6876. Springer, 2011.

[$CoLa03$] J.-F. Cordeau, G. Laporte, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem", Transportation Research Part B 37: 579–594, 2003

[*Parragh et al.* 10] S.N. Parragh, K.F. Doerner, R.F. Hartl, "Variable neighborhood search for the dial-a-ride problem", Computers & Operations Research 37 (6),1129–1138, 2010.

[*Jørgensen et al.* 07] R.M. Jørgensen , J. Larsen, K.B. Bergvinsdottir, "Solving the dial-a-ride problem using genetic algorithms", J Oper Res Soc 58:1321–1331, 2007.