1 Solution Method

Describe Local search algorithms and refer some papers. These algorithms can have difficulties moving from one solution space to another, even when this is cosidered in the metaheuristics.

A way to deal with this could be to, refer to some paper that tries to implement some form of heuristic to search larger neighbourhoods. In this paper we proposed a heuristic is based on Shaw (1997) Large Neighbourhood Search (LNS)

2 ALNS on PDPMTWF

Describe here the difference between this model and standard methods like ALNS and other LNS for PDPTW

2.1 Request shuffeling

To quickly do series of local searches and reach locally optimal solutions we have designed the following heuristics to efficiently search in local neighbourhood.

2.1.1 2-opt

The 2-opt operator used in this paper is based on the 2-opt-L operator from ref: Carcasse2006. It selects a random vehicle with more than 2 orders (it breaks out after not finding any fitting vehicle after o tires, where o is the amount of orders in the problem). For the selected vehicle it divides up the route of the vehicle in 3 parts. All orders up until the index i of the vehicle route are inserted normally. Orders from the index j+1 until the end of the route are also inserted normally. then finally orders from the index i+1 until index j are inserted in inverse order. If the new route has a smaller cost than the original route the route the route is remembered as the new best route. When all reverses have been performed on the current route, the best route is selected as the new route. This operation is continued until no improvement can be made ie. the best possible schedule for the selected vehicle has been found.

2.1.2 Swap first fit

This operator simply swaps the pickup and delivery of two orders first time it fits. This operator is very efficient and jumps randomly around the solution space.

2.2 Removal and reinsertion heuristics

Removal and reinsertion heuristics are well reserached tools used in solving PDP. We have implemented and adapted the most succesful heuristics and created some new ones to help solve our specific problem.

2.2.1 Remove Non-Clustered and insert Clustered

This heuristic tries to remove orders that are bundled together but belong to different clusters. It then tries to bundle orders together in the best possible way. The orders pickup and delivery locations are all assorted into clusters based on the distance between them. We use a hirarchial single linkage clustering algorithm to cluster locations together into k clusters. (TODO: make algorithm table for herarchial single linkage algorithm) Then we compare each set of k cluster to eachother using the siluette coefficient, and keep the best one. (TODO: make algorithm table for siluette coefficient)

To choose which order to remove we have ranked the orders based on how many clusters are shared within a vehicles schedule. If an orders pickup and delivery shares no cluster with any other pickup or delivery nodes the rank is 0. If the order shares the same cluster with both pickup and delivery node as all other pickup and delivery nodes within a vehicle it is given a rank 1. This leads us to quite easily differenciate between a well clustered order and a not well clustered one. The orders are sorted based on their rank and we choose the order to pickup that has the lowest rank, using some randomization, based on the value p. We choose the order on the position k^p in the rank where k is a random number. We have chosen $p = 4$ in this paper.

To reinsert the orders we have chosen to try and maximize the cluster rank we introduced above. This means we find the vehicle where the rank is the highest and insert the order in the best possible position in the chosen vehicle.

2.2.2 Remove Worst and Insert Greedy

Removing orders in the most costful positions and reinserting it in its cheapest (greedy) position seems to be a reasonable way of moving towards a better solution. We therefore propose a heuristic that removes the orders with the highest cost C_i. The C_i is calclated as the increase in a vehicles schedule cost with the chosen order. We remove orders again based on the same randomness factor explained above. We do this by first ranking the orders based on cost and choose the order in the k^p position.

The reinsertion is using a greedy algorithm where it simply checks to find the best possible possition for the order and inserts it there.

2.2.3 Remove Similar and Regret Insert

The removal heuristic used in this operator is based on Shaw (1997) with slight modifications based on our PDPMTWF problem. It removes orders that share specific similar qualities, as the basic idea is that by replacing these orders by eachother we find new, hopefully better, solutions. Another good reason to remove similar orders is that it could be advantagous to reinsert these orders togther on the same vehicle since they share alot of the same properties in regards to distance, time etc. We define a relatedness factor r_{ij} which represents how much the order i is related to the order j. The lower the value of r_{ij} the more the two orders i and j are related. The relatedness of two orders we base here on the following properties: a distance property, a weight property, an overlapping timewindow property, a property indicating if the same vehicles can be used to serve each request, and finally if the orders belong to the same factory.

The relatedness factor is given by the following equation:

$$r_{ij} = \psi(D_{ij} + D_{(i+n)(j+n)}) + \omega|Q_i - Q_j| + \phi(1 - \frac{|V_i \cap V_j|}{max(|V_i|, |V_j|)}) + \tau G_{ij} + \chi(U_{ij} + U_{(i+n)(j+n)}) \quad (1)$$

D_{ij}, Q_i, are defined in the problem formulation section and these values have been normalised to result in a value between [0..1]. V_i is the set of vehicles that can serve order i. The parameter G_{ij} is 1 if i belong to another factory than j and 0 if they belong to the same factory. U_{ij} is the timewindows at the pickup and delivery location, equal to 0 if the two orders have identical time windows and 1 if not. It corresponds to the sum of overlapping time windows divided by the overlapping span of the two time window sets. It can be formulated as follows

$$U_{ij} = 1 - \frac{\sum\limits_{\substack{p\in\pi_i \\ o\in\pi_j \\ \underline{T_{ip}}\le\overline{T_{jo}} \\ \underline{T_{jo}}\le\overline{T_{ip}}}} (\min(\overline{T_{ip}}, \overline{T_{jo}}) - \max(\underline{T_{ip}}, \underline{T_{jo}}))}{\max(\max\limits_{p\in\pi_i}\overline{T_{ip}}, \max\limits_{o\in\pi_j}\overline{T_{jo}}) - \min(\min\limits_{p\in\pi_i}\underline{T_{ip}}, \min\limits_{o\in\pi_j}\underline{T_{jo}}) - \sum\limits_{\substack{p\in\pi_i \\ o\in\pi:q_j \\ \underline{T_{ip}}\ge\overline{T_{j(o-1)}} \\ \underline{T_{jo}}\ge\overline{T_{i(p-1)}}}} (\min(\underline{T_{ip}}, \underline{T_{jo}}) - \max(\overline{T_{i(p-1)}}, \overline{T_{j(o-1)}}))} \tag{2}$$

Here $\overline{T_{ip}}$ and $\underline{T_{ip}}$ are the upper and lower time windows defined in the problem formulation. Thus the relatedness measure is given a value $0 \le r_{ij} \le 2\psi + \omega + \phi + \tau + \chi$. We have chosen the following values in this paper $\psi = 0.7$, $\omega = 1.0$, $\phi = 0.8$, $\tau = 0.3$, $\chi = 0.3$.

The insertion part of this operator tries to improve on the insert greedy by calculating a regret value, c_i^*, which represents how much is lost by inserting this order in its second best position.

2.2.4 Remove Randomly and insert First Fit

Removes randomly between 2 orders and 10 percent of the amount of orders and reinserts them randomly in their first possible position. This operator is used to jump from one neighbourhood to another and is trying to search for possible solutions regardless of the cost they produce.

2.3 Choosing heuristic

2.4 Adaptive Weights

2.5 Stopping and Acceptance Criteria

2.6 Wild operator

If we for 500 iterations find no improvement in the solution we are assuming that we are stuck. In this case we need to do something to get unstuck. We have designed our algorithm so that in this case we run 100 iterations apart from the normal iterations, where we use the most random operators from our selection and accept every solution generated. This will allow us to move away from the solution space into another neighbourhood to begin our search here.

References

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK*, 1997.