Perturbation heuristics for the pickup and delivery traveling salesman problem

Daniëlle Hooijenga (366347)

July 4, 2015

Abstract

The Traveling Salesman Problem with Pickup and Delivery (PDTSP) is considered. Seven perturbation heuristics are applied to instances ranging from 51 to 443 vertices. The main point of perturbation heuristics is to help a local search process to move away from a local optimum. The seven heuristics fall into three categories: Instance Perturbation (IP), Algorithmic Perturbation (AP), and Solution Perturbation (SP). The main idea of these heuristics is to make perturbations to the problem instance, the method, or the solution, respectively. The optimal tours of the instances considered are known and the computational results are compared to these optima. The performances of most of the perturbation heuristics are fairly similar with, in the best case, an average deviation of around 16 percent from the optimum. Only the AP schemes perform considerably worse with an average deviation of around 60 percent from the optimal solution. Improvements in the performances are made by adjusting the initialization of the method, as well as by varying one of the user-controlled parameters of the process.

Contents

1 Introduction

This paper is concerned with the Traveling Salesman Problem with Pickup and Delivery (PDTSP), which is a variant of the well-known Traveling Salesman Problem (TSP). In the PDTSP a shortest route needs to be found along all customers, where the pickup customers need to be visited before their corresponding delivery customer. In distribution management it is commonly necessary to construct a shortest tour and making several pickups and deliveries (Renaud et al., 2002), such as in the case of urban courier services. Because the PDTSP is very hard to solve to optimality, perturbation heuristics will be used in this paper (Renaud et al., 2002). Seven of these perturbation heuristics will be implemented and the accompanying computational results will be compared. Besides, two ways in attempting to improve the performance of the heuristics are proposed. The rest of this paper is structured as follows: in Section 2 an overview of relevant literature on the discussed topic is given. Section 3 describes the problem formulation, followed by a description of the used methods in Section 4. Section 5 gives information on the data used in obtaining the computational results and these results are given in Section 6. Finally, Section 7 gives recommendations for potential further research, followed by the conclusion in Section 8.

2 Literature Review

A problem which is related to the PDTSP is the Dial-A-Ride Problem (DARP). In the DARP a vehicle, with initial position A, is called to service N customers. Each customer wants to travel from an origin to a destination. Finally, the vehicle needs to return to location A. The objective is to minimize the distance (Psaraftis, 1983). The difference between the DARP and the PDTSP is that in the DARP there may be multiple vehicles instead of one, and time windows may be present (Renaud et al., 2002). The problems are similar in the sense that in both cases the origin or the pickup customer needs to be visited before the destination or the delivery customer.

Another problem related to the PDTSP is the Vehicle Routing Problem with Backhauls. In this problem multiple vehicles are considered. The vehicles are initially located at a depot. The objective of the problem is to optimally serve a set of customers which is partitioned into two subsets of customers: linehaul and backhaul customers. Each route starts and ends at the depot. On the route all linehaul customers must be visited before any backhaul customer is visited (Mingozzi et al., 1999).

Very similar to the Vehicle Routing Problem with Backhauls is the TSP with Backhauls. In this problem the objective is to determine a least-cost Hamiltonian cycle on the graph G, such that the linehaul customers are visited before the backhaul customers (Gendreau et al., 1997).

The difference between the PDTSP and the latter two mentioned problems is that in the Vehicle Routing Problem with Backhauls and the TSP with Backhauls there is no relationship between the linehaul and the backhaul customers, whereas in PDTSP to each pickup customer corresponds exactly one delivery

customer and to every delivery customer corresponds exactly one pickup customer (Renaud et al., 2002). An exact method to solve PDTSP is described by Kalantari et al. (1985). They use a branch-and-bound method and applied this procedure to instances of up to 37 vertices. It was concluded that for larger instances heuristics should be used.

Heuristics have been proposed by, among others, Healy and Moll (1995). They propose an extension to the general local search heuristic. Furthermore, Renaud et al. (2000) propose a composite heuristic consisting of a solution construction phase and a deletion and reinsertion phase.

A perturbation heuristic, known as Instance Perturbation, was introduced by Codenotti et al. (1996). The main idea of this technique is to escape from local optima by introducing perturbations in the problem instance rather than in the solution (Renaud et al., 2002).

A different type of perturbation is the Algorithmic Perturbation (Renaud et al. 2002). The idea of AP is that perturbations are introduced in the method, rather than in the problem instance or solution. An example of this is variable neighbourhood search which is applied on the TSP with backhauls by Mladenovic and Hansen (1997).

A third type of perturbation heuristic is Solution Perturbation. In this heuristic the local optimum is modified and after that the procedure of improvement is reapplied to the perturbed solution (Renaud et al., 2002).

3 Problem Formulation

The problem description and notation used throughout this paper are as follows. G = (V,E) is an undirected graph. V = $\{v_1,\ldots,v_n\}$ is the vertex set in this graph, with n odd, and the vertex v_1 representing a depot. The set E = $\{(v_i, v_j)$: i < j, $v_i, v_j \in V\}$ is the edge set of the graph. The set $V\backslash\{v_1\}$ is partitioned into {P,D}, where P is the set of pickup customers and D is the set of delivery customers. It holds that $|P| = |D|$ = (n-1)/2, to every pickup customer corresponds exactly one delivery customer and vice versa. Furthermore, C = (c_{ij}) is the distance matrix defined on the set E, where c_{ij} represents the distance from vertex v_i to vertex v_j. When i > j, c_{ij} should be interpreted as c_{ji}. The purpose is to find a shortest Hamiltonian cycle on this graph where the pickup customer should always be visited before the corresponding delivery customer. This problem description is as in Renaud et al. (2002).

4 Methodology

Seven perturbation heuristics will be considered. Each method used has three parts: Initialization, Perturbation, and Postoptimization. In the Initialization a feasible solution is constructed and an attempt

is made to improve this solution by means of 4-opt**, thereafter one of the seven perturbation schemes is applied, and finally, the 4-opt** procedure is applied to the solution created in the Perturbation step. In all cases, the best-known solution is kept in memory and the algorithm terminates when the best-known solution has not improved for λ successive applications of the Perturbation step and the Postoptimization step, where λ is a user-controlled parameter (Renaud et al., 2002). In order to examine the effect of this parameter on the performance of the heuristics, two values for λ will be examined, namely 5 and 20.

4-opt**

4-opt** is proposed by Renaud et al. (2002) and is an adjustment to 4-opt*, which is proposed by Renaud et al. (1996). The general idea of 4-opt is to remove four edges from the current solution and to try improving the solution by inserting four different edges. The difference between 4-opt* and 4-opt is that in the former there are only eight possible reconnections that need to be considered, in comparison to 48 possible reconnections in the standard 4-opt procedure. The procedure in 4-opt* is as follows: first consider the chain $(v_i,v_{i+1},\ldots,v_{u+i},v_{u+i+1})$ where $u \leq w$ and w is a user-controlled parameter, which will be taken as 5 as is suggested by Renaud et al. (1996). Also consider a second chain (v_k,v_{k+1},v_{k+2}), which should not have any vertex in common with the first chain. If $\min\{c_{i+1,k+1},c_{u+i,k+1}\} < \max\{c_{i,i+1},c_{u+i,u+i+1},c_{k,k+1},c_{k+1,k+2}\}$ then an attempt to make a better tour is made in the following way: first remove the edges (v_i,v_{i+1}), (v_{u+i},v_{u+i+1}), (v_k,v_{k+1}) and (v_{k+1},v_{k+2}), then the cheapest of the edges (v_{i+1},v_{k+1}) and (v_{u+i},v_{k+1}) needs to be added, as well as three other edges in order to make the tour feasible again. A simplified representation of the tour is given in Figure 1. In Figure 2 a representation of the tour is given where the specified edges have been deleted. Figure 3 and Figure 4 show the situation after either the edge (v_{i+1},v_{k+1}) or (v_{u+i},v_{k+1}) is added, respectively.

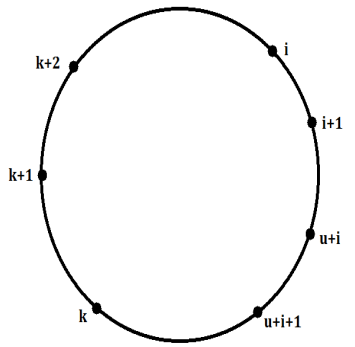

Figure 1: Representation of the tour

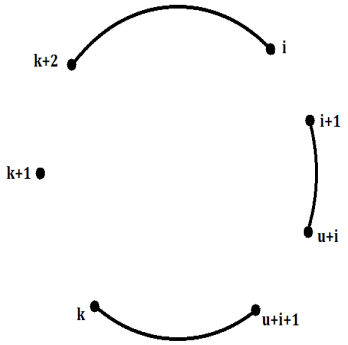

Figure 2: Representation of the tour after the specified edges have been deleted

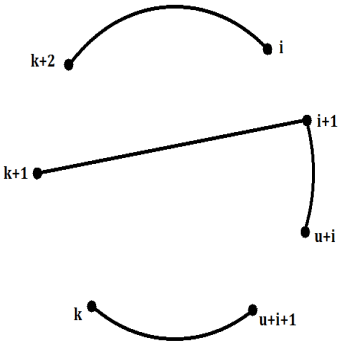

Figure 3: Tour after the edge (v_{i+1},v_{k+1}) is added

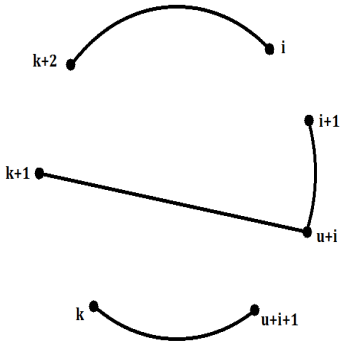

Figure 4: Tour after the edge (v_{u+i},v_{k+1}) is added

The adjustment in 4-opt** in comparison to 4-opt* is to make sure that the pickup customer always appears before the corresponding delivery customer in the tour. This is done by explicitly checking whether a proposed tour is feasible or not. Pseudocode of the 4-opt** algorithm can be found in Algorithm 1. As can be seen this algorithm uses functions to compute the possible gain and to exchange the tour. In *computeGain* the possible gain is computed by computing the cost of all possible new tours, *exchangeTour* makes the actual change; the tour with the highest gain, which is also feasible, becomes the new tour.

Algorithm 1 Pseudocode 4-opt**

```
bestGain = Inf
while bestGain > 0 do
    for u = 1 to w  do
        bestGain = Inf
        bestI = Inf
        bestK = Inf
        while bestGain > 0 do
            bestGain = 0
            for i = 1 to length(tour) do
                for k = 1 to length(tour) do
                    if min{c_{i+1,k+1},c_{u+i,k+1}}<max{c_{i,i+1},c_{u+i,u+i+1},c_{k,k+1},c_{k+1,k+2}} then
                        computeGain(i,k,u)
                        if gain > bestGain then
                            bestGain = gain
                            bestI = i
                            bestK = k
                        end if
                    end if
                end for
            end for
            if bestI != Inf && bestK != Inf then
                exchangeTour(bestI,bestK,u)
            end if
        end while
    end for
end while
```

Initialization

Step 1 (Initial subtour)

Determine the vertex pair (v_i, v_j) yielding $\max_{V_i \in P}\{c_{1i} + c_{ij} + c_{j1}\}$, where $v_j = d(v_i)$, the delivery customer corresponding to pickup customer v_i.

Set $P := P\backslash\{v_i\}$.

Step 2 (Vertex insertion)

If $P = \emptyset$, go to Step 3.

Otherwise, determine the vertex pair (v_i, v_j) yielding the best score value.

Two cases are possible:

Case 1. v_i and v_j are inserted consecutively between vertices v_k and v_l. Let F be the set of all edges in the current subtour. Then:

$$SCORE1 = \min_{v_i \in P, (v_k, v_l) \in F} \{\alpha\ c_{ki} + c_{ij} + (2 - \alpha)\ c_{jl} - c_{kl}\}$$

where $v_j = d(v_i)$ and α is a user-controlled parameter ($0 \leq \alpha \leq 2$).

Case 2. v_i and v_j are inserted between vertices v_k, v_l and v_r, v_s respectively, where (v_r, v_s) appears after (v_k, v_l) on the current subtour. Then:

$$SCORE2 = \min_{v_i \in P, (v_k, v_l), (v_r, v_s) \in F} \{\alpha\ (c_{ki} + c_{il} - c_{kl}) + (2 - \alpha)\ (c_{rj} + c_{js} - c_{rs}\}$$

where $v_j = d(v_i)$ and α is a user-controlled parameter ($0 \leq \alpha \leq 2$).
The vertex pair (v_i, v_j) yielding $\min\{SCORE1, SCORE2\}$ is then inserted in its appropriate position in the subtour. The parameter α will be taken as 1.25 as suggested by Renaud et al. (2002).

Set P := P\{v_i}.
Repeat Step 2.

Step 3 (4-opt**)
Attempt to improve the current solution by means of 4-opt**.

Alternative initialization

In order to find out whether a different starting point has an effect on the performance of the heuristics, in addition to the initialization described before, a different initialization will be used. The results between the initial initialization and the alternative initialization can then be compared to find out whether a difference in performance occurs. Step 1 and Step 3 of the initial initialization stay the same, in Step 2 instead of computing the two scores for every pair of vertices and inserting the vertex pair yielding the best score in each step, the insertion is now done by each iteration taking the first not yet inserted vertex in the set P (the set of pickup customers), where the set P is ordered on index of the vertices. The two scores that were described before are computed for the selected pickup customer with its corresponding delivery customer and the pair of vertices is then inserted at minimum cost by computing $\min\{SCORE1, SCORE2\}$.

Perturbation

Perturbation scheme IP1

Step 1 (Vertex moves)

Move each vertex $v_i \in V\backslash\{v_1\}$ with probability β, which will be taken to be 0.5 as proposed by Renaud et al. (2002). When vertex v_i is selected for a move it is randomly relocated within a circle of radius γc_{1i} centered at its current position. Then the modified distance matrix C' is computed and the tour length updated. The temporary new location of the selected vertex is determined by randomly generating a distance between zero and the maximum distance allowed, randomly generating a distance to move vertically, and finally randomly picking a direction in which to move (to the right and up, to the right and down, to the left and up, to the left and down) by randomly picking an integer between 1 and 4, each representing one of the four possible directions.

Step 2 (4-opt**)

Apply 4-opt** to the perturbed instance.

Step 3 (Mapping)

Map each vertex of the tour obtained at the end of Step 2 onto its initial position and update the tour length.

Perturbation scheme IP2

Step 1 (Vertex moves)

Consider the current solution (v_1,v_{i_1},v_{i_2},...,v_{i_n},v_1). For t = 2,...,n, randomly relocate vertex v_{i_t} with probability β, which will be taken as 1.0 as proposed by Renaud et al. (2002), in the crown centered at v_{i_t-1}, determined by the two radii c_{i_t,i_t-1} and $(1+\gamma)c_{i_t,i_t-1}$. The parameter γ will be taken as 0.2 as proposed by Renaud et al. (2002). The temporary new location of the selected vertex is determined in the same way as was done in perturbation scheme IP1, however, now the distance is randomly generated between the minimum distance and the maximum distance the vertex can move.

Step 2 and Step 3 are the same as in IP1.

Perturbation scheme AP1

Step 1 and Step 3 of the initialization stay the same.

In Step 2, instead of seeking the vertex pair minimizing min{SCORE1, SCORE2}, this choice is now made randomly among all non-inserted vertices. The selected pair is then inserted at minimum cost by computing min{SCORE1, SCORE2} with $\alpha = 1$.

Perturbation scheme AP2

Same as AP1 but by using a randomly selected value of α in [0,2].

Perturbation scheme SP1

The current solution is perturbed by randomly removing between δ and θ vertices from the current tour and reinserting each of them in a random but feasible position in the tour. The parameters δ and θ will be taken to be 10 and 15, respectively, as suggested by Renaud et al. (2002).

Perturbation scheme SP2

As in SP1, between δ and θ vertices are randomly removed from the current tour. However, reinsertions are now performed by using a least insertion length criterion, while maintaining feasibility. The removed vertices are sequentially reinserted between consecutive vertices in order to minimize the additional distance arising from the reinsertion, while maintaining feasibility. The parameters δ and θ will here be taken as 0.10n and 0.15n, respectively, as suggested by Renaud et al. (2002), where n is the number of vertices, and the parameters will be rounded to the nearest integer.

Perturbation scheme SP3

In SP3 two new solutions are created. This is done by combining solution S1 produced by SP1 with the best known solution S2, different from S1. A cross-over position u is randomly selected between $\lceil \pi n \rceil$ and $\lceil \sigma n \rceil$. π will be taken to be 0.3, and σ will be 0.6 as proposed by Renaud et al. (2002), n is the number of vertices. To generate a new solution S', the first u vertices of S1 are kept in this order, and S' is completed by considering the vertices of S2. Whenever a vertex of S2 not already present in S' is identified it is introduced after the vertices of S', in the same sequence in which they appear in S2. A second solution S" is created by keeping the first u vertices of S2 and completing S" by using the vertices of S1. Either S' or S" may then yield a new best known solution.

Postoptimization

4-opt** is applied to the solution obtained at the end of Step 2 or, when SP3 is used, to the two solutions S' and S". Note that this step is not needed for AP1 and AP2 as in these perturbation schemes nothing is changed after the 4-opt** procedure is applied in the initialization.

5 Data

The data which will be used to test the performance of the heuristics consists of 16 Euclidean instances obtained from TSPLIB (Reinelt, 1991), ranging from 51 to 442 vertices. The distances are rounded to the nearest integer. To be able to evaluate the performance of the heuristics, the optimal tours belonging to the instances are considered. The pickup and delivery pairs are formed in the following way: the first vertex is considered to be the depot. Thereafter, the first vertex in the tour not yet visited is considered

a pickup customer and the corresponding delivery customer is randomly selected from the set of all unselected vertices. If the instance has an even number of vertices, one vertex is added as follows: a vertex already present in the tour is randomly selected. The extra vertex is added between this vertex and the one following in the tour. The additional vertex is placed on the middle of the line between the selected vertex and the next one.

6 Results

An overview of all parameters used in the execution of the perturbation heuristics can be found in Table 1.

IP1	$\beta = 0.5$	$\gamma = 0.2$	$\lambda = 5, 20$
IP2	$\beta = 1.0$	$\gamma = 0.2$	$\lambda = 5, 20$
AP1	$\alpha = 1$	$\lambda = 5, 20$	
AP2	$\alpha \in [0, 2]$	$\lambda = 5, 20$	
SP1	$\delta = 10$	$\theta = 15$	$\lambda = 5, 20$
SP2	$\delta = 0.10$n	$\theta = 0.15$n	$\lambda = 5, 20$
SP3	$\pi = 0.3$	$\sigma = 0.6$	$\lambda = 5, 20$

Table 1: Parameters used in the execution of the heuristics

The algorithms that were described in Methodology were implemented in MATLAB 8.3. The structure of this section is as follows: first the results for the method with the initial initialization will be given, with $\lambda = 5$, directly followed by $\lambda = 20$. After that the results for the method using the alternative initialization will be given, also first with $\lambda = 5$, followed by $\lambda = 20$.

The first column in the tables gives the number of vertices in the instances used; the second column gives the number of instances belonging to this group. Value/opt represents the ratio of the value found by the heuristic over the optimal value of the tour. The time is given in seconds. Both the value/opt ratios and the computation times are averages over the number of instances as given in the second column. If an instance would takes longer than 6 hours to complete, the program is forcibly terminated after 6 hours (21 600 seconds) and the best known solution up to that point is retained.

As the AP heuristics do not rely on which initialization is used, these results are equal for both initializations. Therefore, the results will only be presented in the section Initial initialization.

Initial initialization with $\lambda = 5$

The results obtained using the initialization as described by Renaud et al. (2002) can be found in Table 2, Table 3, and Table 4.

n	nr of instances	IP1		IP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.12	265	1.18	421
100-199	8	1.19	1169	1.21	3110
200-299	2	1.15	17168	1.15	18791
300-499	1	1.40	21600	1.40	21600
All	16	1.19	4163	1.22	5376

Table 2: Results of the IP heuristics, using $\lambda = 5$

n	nr of instances	AP1		AP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.29	347	1.27	373
100-199	8	1.43	1808	1.46	2327
200-299	2	1.94	21600	2.01	21600
300-499	1	4.28	21600	4.58	21600
All	16	1.63	5062	1.67	5330

Table 3: Results of the AP heuristics, using $\lambda = 5$

n	nr of instances	SP1		SP2		SP3	
		Value/opt	Time	Value/opt	Time	Value/Opt	Time
50-99	5	1.13	389	1.14	127	1.08	102
100-199	8	1.20	756	1.22	215	1.17	421
200-299	2	1.15	6133	1.15	3570	1.15	946
300-499	1	1.40	21600	1.40	21600	1.40	10973
All	16	1.20	2616	1.21	1936	1.17	1046

Table 4: Results of the SP heuristics, using $\lambda = 5$

Except for AP1 and AP2 the heuristics seem to perform approximately equal, with an average deviation of around 20 percent from the optimum.

It can be seen that SP3 performs the best on average with an average deviation of 17 percent from the optimal solution. Moreover, SP3 is also the heuristic having the lowest computation times. The performance of this heuristic can be assigned to the use of the best found solution so far.

The worst performing heuristic is AP2, directly followed by AP1; with value/opt ratios of 1.67 and 1.63, respectively. This may be explained by the fact that in the initialization used for all other perturbation schemes the vertex pair (v_i,v_j) with the lowest insertion cost is determined and inserted in the appropriate position. In the initialization of the AP schemes this is done by randomly selecting a vertex pair and inserting it in the position yielding the lowest possible cost.

Another difference between the AP schemes and the other perturbation schemes is that for IP and SP after the initialization the perturbation takes place and 4-opt** is applied again, whereas AP is based on the initialization and is not changed anymore after 4-opt** has taken place in the initialization.

Also, the computation times are rather high for the AP schemes which results in the three largest instances to be terminated after 6 hours of computation time. The execution of the heuristics was not yet done at this time and as a result the value/opt ratios get as high as 4.58. When leaving out the three largest instances, that is, only considering the instances with between 50 and 199 vertices, the value/opt ratios of AP1 and AP2 drop to 1.38 and 1.39, respectively. In this case the heuristics are still the worst performing among all heuristics.

Even though IP1 performs on average better than IP2, the latter has longer computation times. This might be caused by the fact that in IP2 all vertices are moved to a temporary position, whereas in IP1 every vertex has a probability of 0.5 to be moved. This implies that in IP2 for every vertex a new position needs to be generated, whereas in IP1 this only needs to be done for approximately half of the vertices.

Also between SP1 and SP2 significant differences in computation times can be seen. Both heuristics achieve on average the same results, but SP1 has longer computation times. This is surprising as in SP1 the removed vertices are inserted in a random (but feasible) position in the tour. In SP2 the vertices are reinserted using a least length insertion criterion, which implies that for every (feasible) possibility the extra length created by adding the vertex needs to be computed before inserting the vertex.

Initial initialization with $\lambda = 20$

Table 5, Table 6, and Table 7 give the computational results when making use of the initial initialization in combination with λ equal to 20.

n	nr of instances	IP1		IP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.07	852	1.18	1526
100-199	8	1.16	5213	1.25	9775
200-299	2	1.15	21600	1.15	21600
300-499	1	1.40	21600	1.40	21600
All	16	1.15	6923	1.22	9414

Table 5: Results of the IP heuristics, using $\lambda = 20$

n	nr of instances	AP1		AP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.22	1398	1.25	1361
100-199	8	1.42	9800	1.42	10202
200-299	2	1.94	21600	2.01	21600
300-499	1	4.28	21600	4.58	21600
All	16	1.60	9387	1.64	9567

Table 6: Results of the AP heuristics, using $\lambda = 20$

n	nr of instances	SP1		SP2		SP3	
		Value/opt	Time	Value/opt	Time	Value/Opt	Time
50-99	5	1.11	1686	1.13	443	1.05	332
100-199	8	1.15	7850	1.23	1512	1.12	1431
200-299	2	1.15	20576	1.15	15136	1.15	3174
300-499	1	1.40	21600	1.40	21600	1.40	21600
All	16	1.16	8374	1.20	4136	1.12	2566

Table 7: Results of the SP heuristics, using $\lambda = 20$

When comparing the results using $\lambda = 20$ to the results where $\lambda = 5$ was used, it can be seen that the heuristics perform slightly better. Only IP2 did not improve on the average deviation from the optimum. Both IP1 and SP1 show an improvement of 4 percent, SP2 has improved by 1 percent, and SP3 shows the largest improvement with a decrease of 5 percent in the average value/opt ratio. As a result, SP3 is still the best performing heuristic, and it also again has the lowest computation times. The AP schemes both show an improvement of 3 percent, but they are still the worst performing heuristics on average.

From the fact that an improvement is made when using $\lambda = 20$, we can deduct that improvements take place also after the time that for 5 times no improvement has been found. This seems to indeed be the case as now for more instances an improvement is made at all, in comparison to the case when $\lambda =$

5. When $\lambda = 5$, an improvement is made by the heuristics in 42.5 percent of the cases. When $\lambda = 20$, this number rises to 58.8 percent.

Moreover, when the first improvement has been made after λ exceeded 5, the process gets another chance to make more improvements; λ is reset to zero and the heuristic might improve the best known solution again, whether it is before or after $\lambda = 5$ is reached again. An example of this can be seen in Figure 5, where the cost has been plotted against the number of iterations of the Perturbation and Postoptimization step. The first improvement takes place after λ reached the value of 12, thereafter 12 more improvements are made after values of λ varying between 1 and 15.

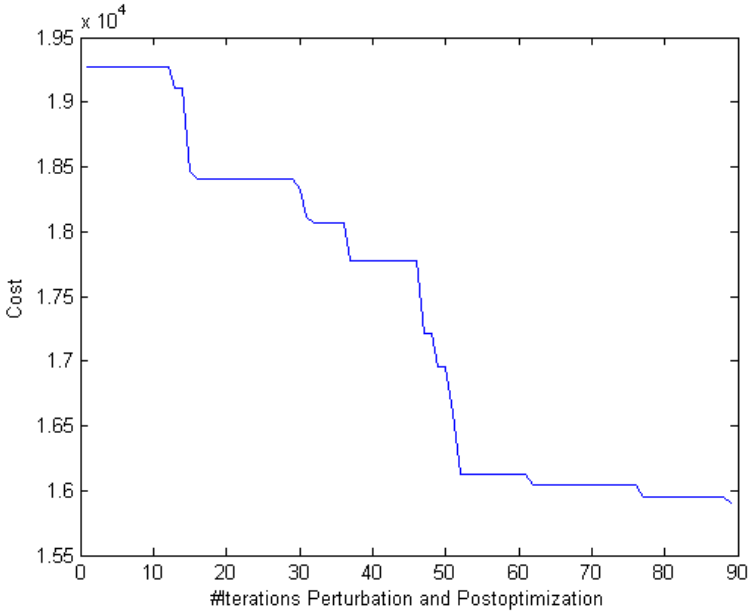

Figure 5: Example of how the cost decreases throughout the process

Alternative initialization with $\lambda = 5$

As mentioned before, here only the results for the IP and SP schemes are shown, as the results for the AP schemes do not rely on the initialization used. The results of the IP and SP heuristics using the alternative initialization and $\lambda = 5$ can be found in Table 8 and Table 9.

The results show that the optima are approached somewhat better when using the alternative initialization, rather than the initial initialization. The biggest difference can be seen in the performance of IP2, where the result is on average 4 percent closer to the optimum than when using the initial initialization. Now both IP1 and SP3 perform the best on average.

n	nr of instances	IP1		IP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.09	79	1.11	342
100-199	8	1.15	696	1.17	2096
200-299	2	1.30	18412	1.30	21600
300-499	1	1.35	21600	1.35	21600
All	16	1.16	4024	1.18	5205

Table 8: Results of the IP heuristics, using $\lambda = 5$

n	nr of instances	SP1		SP2		SP3	
		Value/opt	Time	Value/opt	Time	Value/Opt	Time
50-99	5	1.10	149	1.10	51	1.09	36
100-199	8	1.16	707	1.17	192	1.15	366
200-299	2	1.30	5673	1.30	2259	1.30	1595
300-499	1	1.35	21600	1.35	20962	1.35	10294
All	16	1.17	2459	1.18	1704	1.16	1037

Table 9: Results of the SP heuristics, using $\lambda = 5$

Again the results of all perturbation heuristics, except for the AP schemes, are rather similar. All average value/opt ratios are now around 1.17 instead of 1.20 when using the initial initialization with $\lambda = 5$.

The relative ranking of the different heuristics stays equivalent, except that IP1 now performs just as good as SP3, and IP2 performs on average equally well as SP2.

Alternative initialization with $\lambda = 20$

The results that were obtained using the alternative initialization and $\lambda = 20$ can be found in Table 10 and Table 11.

n	nr of instances	IP1		IP2	
		Value/opt	Time	Value/opt	Time
50-99	5	1.08	663	1.10	1287
100-199	8	1.12	4066	1.17	9083
200-299	2	1.30	21600	1.30	21600
300-499	1	1.35	21600	1.30	21600
All	16	1.14	6291	1.18	8994

Table 10: Results of the IP heuristics, using $\lambda = 20$

n	nr of instances	SP1		SP2		SP3	
		Value/opt	Time	Value/opt	Time	Value/Opt	Time
50-99	5	1.10	700	1.09	305	1.08	187
100-199	8	1.14	4000	1.16	1609	1.11	1295
200-299	2	1.30	21578	1.30	15674	1.29	9520
300-499	1	1.35	21600	1.35	21600	1.35	21600
All	16	1.16	6266	1.17	4209	1.14	3246

Table 11: Results of the SP heuristics, using $\lambda = 20$

The same as was seen before when applying the initial initialization, an improvement is also observed when increasing λ from 5 to 20 in the case of the alternative initialization. However, this improvement is smaller than in the case of the initial initialization. The improvement is at most 2 percent, whereas the initial initialization accomplished improvements of up to 5 percent. The only heuristic not showing any improvement in performance is IP2, this results in the fact that IP2 now performs on average worse than SP2 rather than equally well, as was the case when $\lambda = 5$.

When comparing the number of times an improvement is found, it becomes clear that less often an improvement is found when using the alternative initialization than when using the initial initialization. With $\lambda = 5$ an improvement is made in only 21.3 percent of the cases, when using $\lambda = 20$ this number rises to 38.8 percent of the cases. However, for the initial initialization these percentages were 42.5 and 58.8.

Even though an improvement is made less often when the alternative initialization is used rather than the initial initialization, the results when using the alternative initialization are still slightly better. This may be due to the fact that the cost of the tour found after the initialization are on average 4.8 percent lower when using the alternative initialization.

Discussion

When the results are compared to the results presented by Renaud et al. (2002) it can be seen that the results of Renaud et al. (2002) are considerably better. One major explanation for this could be that for the parameter λ a value of 50 is used by Renaud et al. (2002) compared to 5 and 20 used in obtaining the results presented here. The reason for using a smaller value for λ is the longer computation times. This can be due to the use of sparse matrices in the coding of the heuristics. To represent the current tour an nxn matrix is used, with n the number of vertices, where an element is 1 if this edge is present in the current tour and 0 otherwise. This means that n elements are 1 and n^2 − n elements are 0. This makes the implementation rather inefficient.

The difference between the ratios given by Renaud et al. (2002) and the results presented here is generally around 11 percent. However, the difference between the ratios of the AP heuristics is around 56 percent. This could possibly be due to a misinterpretation of the heuristics, or a different implementation of the algorithms.

When looking at the relative ranking of the performance of the different heuristics of both Renaud et al. (2002) and the results presented here, it can be observed that these are rather similar. In all cases the top three best performing heuristics consist of IP1, SP1 and SP3.The only noticeable difference is that in the results from Renaud et al. (2002) AP2 is the fourth best performing heuristic, whereas in the results given in this paper this is the worst performing heuristic.

Two different initializations were considered. The alternative initialization showed better results both when $\lambda = 5$ and when $\lambda = 20$. However, the results when using the initial initialization approached the results of the alternative initialization better when using $\lambda = 20$. It is not known what will happen to both results when increasing λ further. Therefore, when the effect of the initialization is to be studied, it can be beneficial to also examine the different initializations with a larger variety of values for λ.

7 Further Research

In order to improve the performance of the perturbation heuristics presented in this paper, a larger variety of values for the user-controlled variables could be considered. For example, as was shown an increase in λ results in a better performance, this could be extended by increasing λ further than was done in this paper. Also other values and combinations of all other user-controlled parameters may be considered. As was mentioned before, the implementation was done rather inefficiently. In order to be able to solve the instances with a larger value of λ within a reasonable amount of time, a different implementation should be considered.

The use of a different initialization was considered in this paper and showed some change in performance. To examine this effect to a larger extent, still an other initialization could be tried. Any heuristic could be used to form an initial tour, on which then an attempt can be made to improve the solution by using the perturbation heuristics. An example of a possible heuristic would be the Double Cycle heuristic (Renaud et al., 2000), in which first two Hamiltonian cycles are found for the pickup customers and the delivery customers separately. To create a feasible tour, the depot, pickup customers, and delivery customers are connected using a least cost criterion. In this way all pickup customers are visited before any delivery customer is visited, and thus the tour is feasible. Then this tour can be improved by applying an optimization procedure like 4-opt** and by applying the perturbation heuristics.

Finally, instead of 4-opt**, a different optimization technique could be used. One example is the Lin-Kernighan heuristic (Lin and Kernighan, 1973). This heuristic is based on the idea that it is a drawback

that the k in k-opt procedures need to be decided on in advance. In this heuristic for every move the k with the best compromise between computation time and the quality of the solution is tried to be found.

8 Conclusion

In this paper seven perturbation heuristics have been considered of which the main goal is to help a local search process to move away from a local optimum. Each heuristic made use of an Initialization, a Perturbation, and a Postoptimization step. Next to the initial initialization also an alternative initialization was proposed. The computational results of the heuristics were compared, with the two different initializations and with two different values for the user-controlled parameter λ. The best results were reached using the alternative initialization in combination with λ equal to 20, in this case the results that were found deviated on average around 16 percent from the optimal solutions. It has also been shown that an increase in λ gives an increase in performance independent of which initialization is used.

References

Codenotti, B., Manzini, G., Margara, L., Resta, G. (1996), Perturbation: an efficient technique for the solution of very large instances of Euclidean TSP, INFORMS Journal on Computing, Vol. 8, pp. 125-133

Gendreau, M., Hertz, A., Laporte, G. (1997), An approximation algorithm for the traveling salesman problem with backhauls, Operations Research, Vol. 45, pp. 639-641

Healy, P., Moll, R. (1995), A new extension of local search applied to the dial-a-ride problem, European Journal of Operational Research, Vol. 83, pp. 83-104

Kalantari, B., Hill, A., Arora, S. (1985), An algorithm for the traveling salesman problem with pickup and delivery customers, European Journal of Operational Research, Vol. 22, pp. 377-386

Lin, S., Kernighan, B. (1973), An effective heuristic algorithm for the traveling salesman problem, Operations Research, Vol. 21 (2), pp. 498-516

Mingozzi, A., Giorgi, S., Baldacci, R. (1999), An exact method for the vehicle routing problem with backhaul, Transportation Science, Vol. 33, pp. 315-329

Mladenovic, N., Hansen, P. (1997), Variable neighbourhood search, Computers & Operations Research, Vol.24, pp. 1097-1100

Psaraftis, H. (1983), Analysis of an O(N^2) heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem, Transportation Research, Vol. 17B (2), pp. 130-145

Reinelt, G. (1991), TSPLIB: A traveling salesman problem library, ORSA Journal on Computing, Vol. 3, pp. 376-384

Renaud, J., Boctor, F., Laporte, G. (1996), A fast composite heuristic for the symmetric traveling salesman problem, INFORMS Journal on Computing, Vol. 8, pp. 134-143

Renaud, J., Boctor, F., Ouenniche, I. (2000), A heuristic for the pickup and delivery traveling salesman problem, Computers & Operations Research, Vol. 27, pp. 905-916

Renaud, J., Boctor, F., Laporte, G. (2002), Perturbation heuristics for the pickup and delivery traveling salesman problem, Computers & Operations Research, Vol. 29, pp. 1129-1141