Heuristic for Site-Dependent Truck and Trailer Routing Problem with Soft and Hard Time Windows and Split Deliveries

Mikhail Batsyn(✉) and Alexander Ponomarenko

Laboratory of Algorithms and Technologies for Network Analysis,
National Research University Higher School of Economics, 136 Rodionova street,
Niznhy Novgorod, Russia
{mbatsyn,aponomarenko}@hse.ru

Abstract. In this paper we develop an iterative insertion heuristic for a site-dependent truck and trailer routing problem with soft and hard time windows and split deliveries. In the considered problem a truck can leave its trailer for unloading or parking, make a truck-subtour to serve truck-customers, and return back to take the trailer. This can be done several times in one route. In our heuristic every route is constructed by sequentially inserting customers to it in the way similar to Solomon's (1987) approach developed for simple vehicle routes. Our contributions include: heuristic insertion procedure for complex truck and trailer routes with transshipment locations; efficient randomized mechanisms for choosing the first customer for insertion, for making time window violations, and for making split-deliveries; an improvement procedure shifting deliveries in a route to earlier time; an efficient approach dealing with site-dependency feature based on the transportation problem in case of arbitrary intersecting vehicle sets and a fast vehicle assignment procedure in case of nested vehicle sets.

Keywords: Truck and trailer · Site-dependent · Soft time windows · Split-deliveries · Insertion heuristic

1 Introduction

In truck and trailer routing problems there are two types of customers: trailer-customers and truck-customers. Trailer-customers can be served both from a truck and a trailer, while truck-customers can be served only from a truck. This can be, for example, small stores in a big city to which it is impossible to drive up by a road train because of narrows streets, small parking place, and other limitations. There are two possibilities to leave a trailer. It can be left at a trailer-customer for unloading and while it is unloaded a truck can serve several truck-customers making a so-called truck-subtour and then return to take the trailer. This is efficient in terms of time because truck and trailer serve customers in parallel in this case. Another possibility is to park a trailer at a

P. Pardalos et al. (Eds.): MOD 2015, LNCS 9432, pp. 65–79, 2015.
DOI: 10.1007/978-3-319-27926-8_7

special transshipment location close to some truck-customer. If there are not enough goods in a truck, before it leaves for a truck-subtour, a load transfer from the trailer to the truck can be performed. If there are not enough goods in the trailer, when it is left for unloading at a trailer-customer, then the remaining goods are unloaded from the truck before it leaves.

The most simple truck and trailer routing problem is the homogeneous fleet truck and trailer routing problem which is usually referenced in literature as TTRP (Chao, 2002). In this problem all trucks and trailers are the same, have the same capacity, same travel and fixed costs and can visit every customer without any limitations except the truck-customers limitation. A number of heuristics have been suggested for this problem: Chao (2002), Scheuerer (2006), Lin et al. (2009; 2010), Villegas et al. (2011a; 2011b). Lin et al. (2011) considered the TTRP problem with hard Time Windows (TTRPTW).

Much more difficult problems are Heterogeneous Fleet TTRP problems (HFTTRP). Different heuristics for the HFTTRP problem have been suggested by Hoff (2006), Hoff & Lokketangen (2007), Caramia & Guerriero (2010a; 2010b). Except different vehicle capacities, travel and fixed costs, for every customer there can be defined a set of vehicles which can serve it. In this case the problem is called the Site Dependent TTRP (SDTTRP). Semet (1995) developed a cluster-first route-second heuristic for the SDTTRP problem. Semet & Taillard (1993) suggested a tabu-search algorithm for the SDTTRP problem with hard time windows (SDTTRPTW). In their formulations of these problems there are no transshipment locations and a trailer can be left only for unloading at a trailer-customer. A more general formulation with transshipment locations is presented by Drexl (2011). Along with road trains there are also single-truck vehicles. The author provides a mathematical programming model for this SDTTRPTW problem and a branch-and-price algorithm to solve it.

In this paper we consider even more general TTRP problem. We add soft time windows and split-deliveries to the SDTTRPTW problem considered by Drexl (2011). An integer linear programming model for this problem is provided in Batsyn & Ponomarenko (2014). In the current paper we further develop our heuristic suggested in Batsyn & Ponomarenko (2014). The main improvements include: new approach dealing with site-dependency feature based on the transportation problem in case of arbitrary intersecting vehicle sets; new fast vehicle assignment procedure in case of nested vehicle sets; new randomized mechanism for making soft time window violations; new insertion cases for the greedy insertion procedure. We describe our iterative insertion heuristic for the considered problem, present all possible insertion cases, and provide the pseudo-code of our algorithm. There are many cases of inserting a customer to a complex truck and trailer route because such a route can have different nodes such as: depot, trailer-customers visited by a road train, truck-customers visited by a truck without a trailer, trailer-customers at which a trailer is left for unloading, transshipment locations at which a trailer is left for parking.

In our approach many different solutions are iteratively constructed with the following randomizations. We choose the first customer in every route randomly

from the most expensive (farthest) customers. Split-deliveries and soft time window violations are also made in a random way. In order to avoid making a big detour when inserting a customer, we do not insert customers for which the cost of serving it with another vehicle directly from the depot is smaller than the insertion cost. We insert customers taking into account soft time windows. If a delivery of inserted customer is started after the soft time window we apply an improvement procedure which moves all deliveries earlier in time so that instead of a late (after the soft time window) delivery at this customer we have an early (before the soft time window) delivery at some of the previous customers. This helps to compress a route and insert more customers to it.

For each customer there is a set of different vehicles which can serve it. To deal with this site-dependency feature we suggest solving a special transportation problem in which we give preference to bigger vehicles. In case where these sets for all customers are nested, instead of solving the transportation problem we propose an efficient algorithm to assign a vehicle to serve a customer.

In our formulation of the problem it is permitted to violate a given number of soft time windows and to make split-deliveries to a given number of customers. To address the soft time windows feature we suggest that during building a solution it is allowed to violate a soft time window in a random way with the probability equal to the remaining number of permitted violations divided by the expected number of possible remaining violations. We allow a split-delivery only when we insert the last customer in the current route when the remaining capacity of the vehicle is not enough to serve the total demand of this customer. This is done to load vehicles as much as possible. A split-delivery is allowed in a random way with the probability equal to the remaining number of permitted split-deliveries divided by the expected number of possible remaining split-deliveries.

2 Insertion Heuristic

The following parameters are used is the pseudo-code of the algorithm.
n - the number of customers
V - the set of all customers
K - the set of all vehicles
K_i - the set of vehicles which can serve customer i
f_k - the fixed cost of using vehicle k for delivery
Q_k - the current remaining capacity of vehicle k
q_i - the current remaining demand of customer i
s_i^k - the service time spent by vehicle k when serving customer i
op_i, cl_i - the open and close time of customer i (hard time window)
er_i, lt_i - the earliest and latest time of serving customer i (soft time window)
b_j^R - the begin time of serving customer j in route R
v_R - the number of soft time window violations in route R
c_{ij}^{kl} - the travel cost of arc (i, j) for vehicle k with/without trailer ($l = 1/l = 2$)
v - the number of permitted soft time window violations
w - the current remaining number of permitted soft time window violations

σ - the number of permitted split deliveries
s - the current remaining number of permitted split deliveries
R - the current route
S - the current solution
S^* - the best solution
$f(S)$ - the total cost of the current solution
f^* - the total cost of the best solution
U - the set of all customers sorted the most expensive (farthest) customer first
$[C_{jk}]$ - the cost matrix of the transportation problem used to assign vehicles
μ - the number of the most expensive customers from which we choose randomly
λ - the preference weight of customer direct travel cost c_{0i}^{k1}

Algorithm 1. Iterative insertion heuristic

function IterativeInsertionHeuristic(N)
▷ Builds N solutions running InsertionHeuristic()
$S^* \leftarrow \varnothing, \quad f^* \leftarrow \infty$
$U \leftarrow V$ ▷ the set of all customers sorted so that U_1 has maximal c_{0i}^{k1}
$\bar{Q} \leftarrow \sum Q_k/|K|$ ▷ average vehicle capacity
$\bar{r} \leftarrow \bar{Q}/(\sum q_i/n)$ ▷ average route size
$m \leftarrow 0$ ▷ the total number of built routes
$M \leftarrow 0$ ▷ the total number of customers in these routes
$\pi \leftarrow 0$ ▷ the probability of time window violation
$v' \leftarrow v$ ▷ backup the number of allowed time window violations
$v \leftarrow n$ ▷ temporarily allow unlimited time window violations
InsertionHeuristic($U, [q_j], [Q_k]$) ▷ get a solution with unlimited tw-violations
$\pi \leftarrow (v - w)/n$ ▷ π is estimated as the ratio of time window violations to n
$v \leftarrow v'$ ▷ restore the number of allowed time window violations
for $i \leftarrow \overline{1, N}$ **do**
  $S \leftarrow$ InsertionHeuristic($U, [q_j], [Q_k]$)
  if $S \neq \varnothing$ **then**
    $m \leftarrow m + |S|, \quad M \leftarrow M + \sum_{R \in S} |R|, \quad \bar{r} \leftarrow M/m$
    if $f(S) < f^*$ **then**
      $S \leftarrow S^*, \quad f^* \leftarrow f(S)$
    end if
  end if
end for
return S^*
end function

The main function in our algorithm is IterativeInsertionHeuristic() (Algorithm 1). It makes the specified number of iterations N calling InsertionHeuristic() function and stores the best found solution in S^*. First, all customers in set V are sorted by the direct travel cost c_{0i}^{k1} from the depot so that the first customer has maximal direct cost (is the most expensive). The sorted list of customers is stored in variable U. Note that we copy parameters $U, [q_j], [Q_k]$ each time we call InsertionHeuristic() function so that it can change them without

Algorithm 2. Insertion heuristic

```
function InsertionHeuristic(U, [q_j], [Q_k])
    ▷ Builds a solution for customers U, demands [q_j], vehicle capacities [Q_k]
    ▷ Parameters U, [q_j], [Q_k] are copied and not changed in the calling function
    S ← ∅                                                    ▷ current solution
    w ← v                              ▷ remaining number of soft time window violations
    s ← σ                                  ▷ remaining number of split-deliveries
    while U ≠ ∅ do
        i ← Random(U_1, ..., U_μ)   ▷ choose random from the first μ most expensive
        k ← ChooseVehicle(i, [q_j], [Q_k])
        if k = 0 then
            return ∅                                          ▷ not enough vehicles
        end if
        InsertCustomer(U, i, 1, R, q_i, Q_k)          ▷ insert i to an empty route R
        success ← true
        while success do
            success ← InsertBestCustomer(U, [q_j], [Q_k], R, k)
        end while
        S ← S ∪ {R}
        Q_k ← 0                          ▷ remove vehicle k from further consideration
    end while
    return S
end function
```

affecting their values in the main function. We also calculate here an estimation for the average route size $\bar{r}$ equal to the average vehicle capacity $\bar{Q}$ divided by the average demand. This estimation is used only for the first iteration and for next iterations we divide the total number of customers in all constructed routes by the total number of these routes. To estimate the probability π of soft time window violation we run the insertion heuristic once with an unlimited number of permitted violations v. Then we measure this probability as the fraction of soft time window violations made in the obtained solution to the total number of customers n.

We fill in the cost matrix C_{jk} for the transportation problem using the following formula:

$$C_{jk} = \begin{cases} 0, & k \in K_j \\ \infty, & \text{otherwise} \end{cases}$$

We set $C_{jk} = \infty$ for each vehicle k which cannot serve customer j. The transportation problem is solved to check that the currently available vehicles with remaining capacities $[Q_k]$ are able to serve the remaining demands $[q_j]$ of the customers with site-dependency constraints given by vehicle sets K_j containing for each customer j the vehicles which can serve it.

If these vehicle sets are nested it is possible to check it without solving the transportation problem. It is usual that for many different customers their vehicle sets are the same: $K_{i_1} = ... = K_{i_l}$. We denote these vehicle sets as $K^j = K_{i_1} = ... = K_{i_l}$. Let $K^1 \subset K^2 \subset ... \subset K^m$ be all different nested vehicle sets. When we

Algorithm 3. Choose the best vehicle

```
function ChooseVehicle(i, [q_j], [Q_k])
    ▷ Returns the best vehicle k* for customer i, demands [q_j], vehicle capacities [Q_k]
    ▷ Parameters [q_j], [Q_k] are changed in the calling function also
    Q_max ← max(Q_k)
    [C'_jk] ← [C_jk]
    for k ∈ K_i do
        C'_ik ← Q_max/Q_k                    ▷ set smaller costs for bigger vehicles
    end for
    repeat
        [x_jk] ← TransportationProblem([C'_jk], [q_j], [Q_k])
        if [x_jk] = ∅ then
            return 0                                  ▷ not enough vehicles
        end if
        k* ← 0
        for k ∈ K_i do
            if x_ik > 0 and Q_k > Q_k* then
                k* ← k
            end if
        end for
        if x_ik* < q_i and x_ik* < Q_k* then
            C'_ik* ← ∞      ▷ cannot serve total demand q_i, let's try another vehicle
        end if
    until C'_ik* = ∞
    return k*
end function
```

assign for customer i vehicle k from its vehicle set $K_i = K^j$, and this vehicle does not belong to smaller (nested to K^j) vehicle sets: $k \notin K^{j-1}$, then there is nothing to check. Otherwise, we need to check that this assignment is feasible and the remaining vehicles capacity is enough to serve the remaining demand.

Let us denote as $Q_{K^j} = \sum_{k \in K^j} (Q_k)$ the current total capacity of vehicles in set K^j and as $q_{K^j} = \sum_{i, K_i = K^j} (q_i)$ - the current total demand of customers for which the vehicle set is K^j. We consider that vehicles in K^1 serve all customers for which $K_i = K^1$ and then they have the remaining capacity equal to $Q_{K^1} - q_{K^1}$. Then vehicles in $K^2 \backslash K^1$ together with the remaining vehicles from K^1 serve all customers for which the vehicle set is K^2, and we have the remaining capacity of vehicles equal to $(Q_{K^2 \backslash K^1} - q_{K^2}) + (Q_{K^1} - q_{K^1})$. And so on up to the largest vehicle set K^m. We precalculate all these remaining capacities for all vehicle sets from K^1 to K^m only once, and then we only update it quickly each time we add a customer to a route. If adding a customer results in a negative value for some of the remaining capacities this means that this adding is infeasible and we cannot do it. If we assign vehicle k to serve customer i, and the smallest vehicle set K^{j_0} which contains k is smaller than $K_i = K^j$, then for each set $K^{j_0}, ..., K^{j-1}$ we check that after delivering goods to customer i the remaining vehicle capacities will be enough to serve the demands $q_{K^{j_0}}, ..., q_{K^{j-1}}$.

Function InsertionHeuristic() (Algorithm 2) sequentially constructs all routes in a solution by inserting customers one by one to the constructed route. The first customer in each route is chosen randomly from the first μ customers (μ most expensive) in the set of unserved customers U. In our experiments we take $\mu = 5$ because it provides a good balance between diversification and intensification of the search. A vehicle for each constructed route is assigned in ChooseVehicle() function. Each next customer to be inserted to the current route is chosen and inserted in InsertBestCustomer() function.

Algorithm 4. Insert the customer to the route

function InsertCustomer(U, i, p, R, q_i, Q_k)
▷ Inserts customer i to position p in route R
▷ Parameters U, R, q_i, Q_k are changed in the calling function also
$r \leftarrow |R|$ ▷ the size of route $R = (R_1, ..., R_r)$
$R \leftarrow (R_1, ..., R_{p-1}, i, R_p, ..., R_r)$ ▷ insert customer i to position p in route R
$q \leftarrow \min(q_i, Q_k)$ ▷ the delivered demand
$Q_k \leftarrow Q_k - q$ ▷ update the remaining vehicle capacity
$q_i \leftarrow q_i - q$ ▷ update the remaining customer demand
if $q_i = 0$ **then**
$U \leftarrow U \setminus \{i\}$ ▷ update the unserved customers list
end if
end function

Function ChooseVehicle() (Algorithm 3) chooses the biggest vehicle feasible for customer i. It is done by setting transportation costs C_{ik} such that the bigger vehicles feasible for i have smaller costs and thus the optimal solution of the transportation problem will assign as much demand as possible to the biggest vehicle. If the biggest vehicle cannot serve the total demand q_i due to the site-dependency constraints though its capacity is enough, we forbid assignment of this vehicle by setting $C'_{ik} = \infty$ and solve the transportation problem again.

Function InsertCustomer() (Algorithm 4) inserts customer i to the specified position p in route R served by vehicle k. The remaining capacity Q_k of this vehicle and the remaining demand q_i of this customer are decreased by the value of the delivered demand q. If this customer has no remaining demand after this operation it is removed from the set of unserved customers U.

Function InsertBestCustomer() (Algorithm 5) finds the unserved customer which has the lowest insertion cost and inserts it to the best position in the current route. Precisely we take into account the insertion cost c decreased by the direct customer cost c_{0i}^{k1} multiplied by weight λ. This is done to give preference to the most expensive (farthest) customers because it is usually more optimal to insert such customers to the solution earlier. In our experiments we take $\lambda = 1$ since it gives the best results in average.

Before choosing the best customer we decide in a random way if we allow soft time window violation and split-delivery for this customer. In average every

Algorithm 5. Insert the customer with the lowest insertion cost

function InsertBestCustomer($U, [q_j], Q_k, R, k$)
▷ Inserts the best customer from U to route R served by vehicle k
▷ Parameters $[q_j]$ store demands, Q_k - the remaining free capacity of vehicle k
▷ Parameters $U, [q_j], Q_k, R$ are changed in the calling function also
$\bar{s} = |U|/\bar{r}$ ▷ the expected number of remaining splits (1 split per route)
$\bar{w} = |U| \cdot \pi$ ▷ the expected number of remaining time window violations
$split \leftarrow (\text{Random}([0,1]) < s/\bar{s})$ ▷ allow split-delivery with probability $s/\bar{s}$
$violate \leftarrow (\text{Random}([0,1]) < w/\bar{w})$ ▷ allow tw-violation with probability $w/\bar{w}$
$i^* \leftarrow 0$ ▷ best customer to insert to route R
$p^* \leftarrow 0$ ▷ best position in route R to insert this customer
$c^* \leftarrow \infty$ ▷ insertion cost for this customer
for $i \in U$ **do**
if $k \notin K_i$ **then** ▷ skip customers which cannot be served by this vehicle k
continue
end if
▷ If split-deliveries are not allowed skip all them except inevitable ones
if (! *split*) & $(q_i > Q_k)$ & $(q_i < \max_{k \in K_i}(Q_k))$ **then**
continue
end if
▷ Check that after serving customer i by vehicle k the remaining
▷ vehicles are able to serve the remaining customers
$q \leftarrow \min(q_i, Q_k)$ ▷ the delivered demand
$Q_k \leftarrow Q_k - q$ ▷ try decreasing the remaining vehicle capacity
$q_i \leftarrow q_i - q$ ▷ try decreasing the remaining customer demand
$[x_{jk}] \leftarrow$ TransportationProblem$([C_{jk}], [q_j], [Q_k])$
$Q_k \leftarrow Q_k + q$ ▷ restore the remaining vehicle capacity
$q_i \leftarrow q_i + q$ ▷ restore the remaining customer demand
▷ If the remaining vehicles are not able to serve the remaining customers
if $[x_{jk}] = \varnothing$ **then**
continue ▷ skip such a customer
end if
$p^* \leftarrow 0$ ▷ the best position in route R to insert customer i
$c \leftarrow$ GetInsertionCost$(R, i, k, p, violate)$ ▷ find the best place for i in R
$c_0 \leftarrow c_{0i}^{k1} + f_k/|R|$ ▷ estimation for the cost of serving i directly from depot
if $c_0 < c$ **then**
continue ▷ it is cheaper to serve this customer directly from the depot
end if
$c \leftarrow c - \lambda \cdot c_{0i}^{k1}$ ▷ give preference $(\lambda \cdot c_{0i}^{k1})$ to farthest customers
if $c < c^*$ **then**
$c^* \leftarrow c, \quad i^* \leftarrow i, \quad p^* \leftarrow p$
end if
end for
if $c^* = \infty$ **then**
return false
end if
InsertCustomer$(U, i^*, p^*, R, q_{i^*}, Q_k)$
if $Q_k = 0$ **then**
return false ▷ return false to stop building this route
end if
return true
end function

Algorithm 6. Calculate the best cost of inserting the customer to the route

```
function GetInsertionCost(R, i, k, p*, violate)
    ▷ Inserts customer i to route R served by vehicle k, returns position p* and cost
    ▷ Parameter p* (best insertion position) is changed in the calling function also
    c* ← ∞
    r ← |R|                                          ▷ the size of route R = (R_1, ..., R_r)
    v_R ← Violations(R)
    for p ← 1, r + 1 do
        R' ← (R_1, ..., R_{p-1}, i, R_p, ..., R_r)          ▷ insert customer i to position p
        Δ* ← 0
        for j ∈ {i, R_p, ..., R_r} do
            l ← lt_j          ▷ the latest possible begin time is the time window end lt_j
            ▷ For the second and next split-deliveries violations are not counted
            if NextSplitDelivery(j, R') then
                l ← cl_j − s_j^k          ▷ the latest possible is close time minus service time
            end if
            Δ ← b_j^{R'} − l     ▷ how greater the begin time than the latest possible time
            if Δ > Δ* then
                Δ* ← Δ,    j* ← j
            end if
        end for
        ▷ Try to shift all deliveries earlier by Δ*
        if (! ShiftEarlier(j*, R', Δ*)) then  ▷ if cannot satisfy hard time windows
            continue
        end if
        v_{R'} ← Violations(R')
        if v_{R'} > v_R and violate = false then       ▷ skip if R' has more tw-violations
            continue
        end if
        c ← CostDelta(i, p, R)
        if c < c* then
            c* ← c,    p* ← p
        end if
    end for
    return c*
end function
```

route should have one split-delivery to fully use the capacity of the vehicle. So we take the expected number of possible future split-deliveries $\bar{s}$ equal to the average number of the remaining routes in this solution. And this value is estimated as the number of unserved customers $|U|$ divided by the average route size $\bar{r}$. The expected number of possible future violations of soft time windows $\bar{w}$ is equal to the number of unserved customers $|U|$ multiplied by the probability π of soft time window violation. To provide uniform occurrence of split-deliveries and soft time window violations during construction of a solution we allow a split-delivery and a violation with probabilities $s/\bar{s}$ and $w/\bar{w}$ correspondingly.

Algorithm 7. Count the number of time window violations in the route

```
function Violations(R)
    ▷ Returns the number of time window violations in route R
    v_R ← 0
    r ← |R|                                   ▷ the size of route R = (R_1, ..., R_r)
    for p ← 1, r do
        j ← R_p
        ▷ For the second and next split-deliveries violations are not counted
        if NextSplitDelivery(j, R) then
            continue
        end if
        if b_j^R < er_j or b_j^R > lt_j then   ▷ begin time b_j^R violates time window [er_j, lt_j]
            v_R ← v_R + 1
        end if
    end for
    return v_R
end function
```

For every customer we check that after serving it by the current vehicle the remaining vehicle capacities will be enough to serve the remaining demands. This is done by solving the transportation problem with the cost matrix $[C_{jk}]$. The best position p^* to insert a customer is determined by function GetInsertionCost(). If the insertion cost of a customer is greater than the cost of serving this customer directly from the depot with another vehicle, then such customer is not inserted to the current route. This is done to avoid big detours when inserting customers.

Function GetInsertionCost() (Algorithm 6) finds the best position to insert the given customer to the current route which provides the lowest insertion cost. It tries to insert the customer to every position in the route. If after insertion this or next customers in the route have late deliveries (deliveries after the soft time window), then function ShiftEarlier() is called to shift all the deliveries earlier so that some of the previous deliveries become early deliveries (deliveries before the soft time window). This procedure reduces the total time of the route and thus allows inserting more customers to it.

Function ShiftEarlier() moves the delivery for the specified customer j^* earlier by the specified time Δ^*. This requires moving earlier deliveries for some of the previous customers in the route and for some of the next customers. It moves the deliveries taking into account waiting time and split deliveries, because two vehicles cannot serve one customer at the same time. If it is not possible to move the deliveries earlier so that all hard time windows are satisfied, then ShiftEarlier() function returns $false$. If for example waiting time at some previous customer is greater than Δ^*, then this time is simply decreased by Δ^* and any other previous customers do not need to be moved.

Function NextSplitDelivery() checks if the delivery to the specified customer j in route R is the second or next split delivery to this customer. This is needed because it is allowed to violate the soft time window for all split deliveries

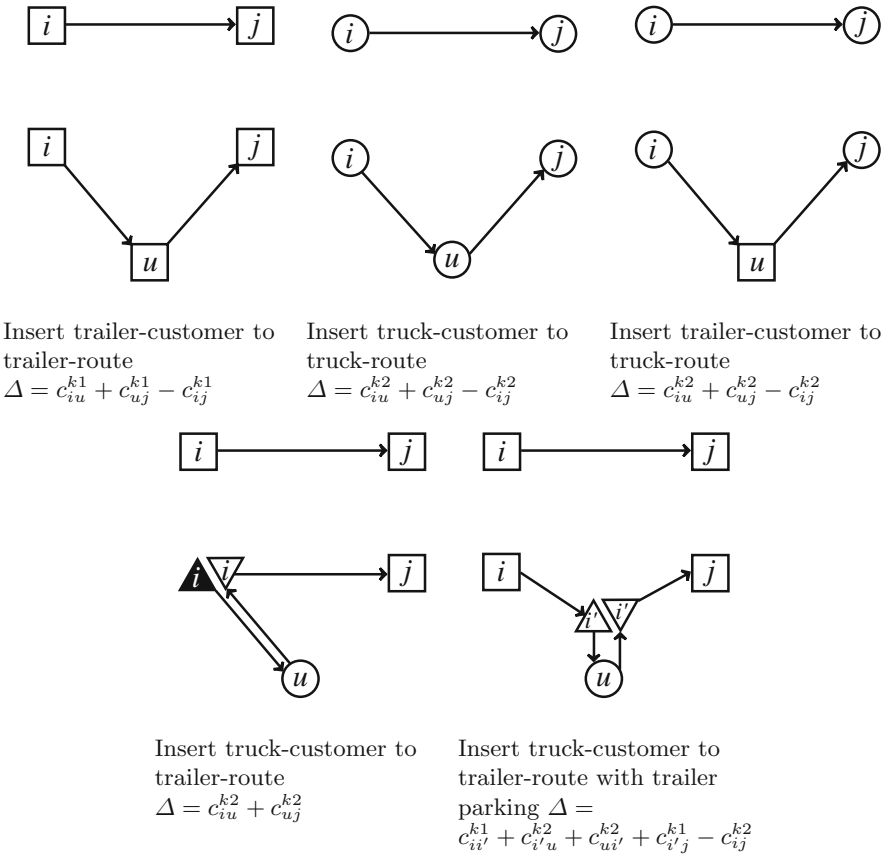
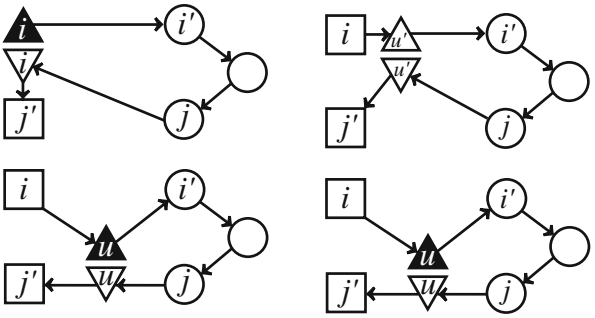


Fig. 1. Simple insertion cases

except one. So we do not count soft time window violations for the second and next split deliveries.

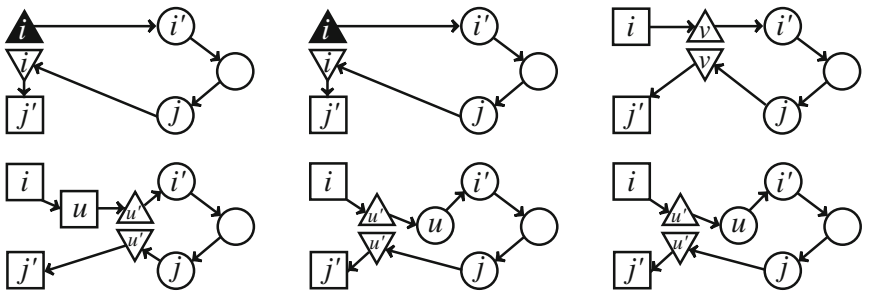
Function CostDelta() returns the insertion cost for all possible cases. These cases together with the corresponding cost deltas are shown in Figs. 1, 2, 3. The following icons are used in these figures: □, a node visited by a road train; ◯, a node visited by a truck without the trailer; △, a node at which a trailer is left for parking; ▲, a node at which a trailer is left for unloading; ▽, a node at which a trailer is connected back to a truck. In each of these figures a new customer u is inserted to a route. The original route is shown above and the route after inserting this customer is shown below in each figure. An expression to calculate the cost delta is also provided.

Function Violations() (Algorithm 7) counts the number of soft time window violations made in the given route. Note that in case of split deliveries the soft time window should not be violated only for the first split delivery to a customer. All the next split deliveries to this customer can be outside the soft time window.



Leave trailer at new customer
$\Delta = c^{k1}_{iu} + c^{k2}_{ui'} + c^{k2}_{ju} + c^{k1}_{uj'} - c^{k2}_{ii'} - c^{k2}_{ji} - c^{k1}_{ij'}$

Leave trailer at new customer and remove parking
$\Delta = c^{k1}_{iu} + c^{k2}_{ui'} + c^{k2}_{ju} + c^{k1}_{uj'} - c^{k1}_{iu'} - c^{k2}_{u'i'} - c^{k2}_{ju'} - c^{k1}_{u'j'}$



Leave trailer at parking place after new trailer-customer
$\Delta = c^{k1}_{iu} + c^{k1}_{uu'} + c^{k2}_{u'i'} + c^{k2}_{ju'} + c^{k1}_{u'j'} - c^{k2}_{ii'} - c^{k2}_{ji} - c^{k1}_{ij'}$

Leave trailer at parking place before new truck-customer
$\Delta = c^{k1}_{iu'} + c^{k2}_{u'u} + c^{k2}_{ui'} + c^{k2}_{ju'} + c^{k1}_{u'j'} - c^{k2}_{ii'} - c^{k2}_{ji} - c^{k1}_{ij'}$

Leave trailer at new parking place before new truck-customer
$\Delta = c^{k1}_{iu'} + c^{k2}_{u'u} + c^{k2}_{ui'} + c^{k2}_{ju'} + c^{k1}_{u'j'} - c^{k1}_{iv} - c^{k2}_{vi'} - c^{k2}_{jv} - c^{k1}_{vj'}$

Fig. 2. Insertion cases with leaving trailer at another node

3 Computational Experiments

For computational experiments we used several real-life instances with the number of customers from 55 to 300 (input data for all instances can be provided by request). All the experiments for real-life instances have been performed on Intel Core i7 machine with 2.2 GHz CPU and 8 Gb of memory. A comparison with an exact solution is possible only for very small instances of about 10 customers. Such a comparison for the first version of our heuristic can be found in Batsyn & Ponomarenko (2014).

The objective function values obtained by different algorithms are presented in Table 1. When an algorithm is not able to obtain a feasible solution it is shown as "–" in the table. We compare the suggested heuristic with an iterative greedy insertion heuristic which iteratively applies Solomon (1987) insertion procedure

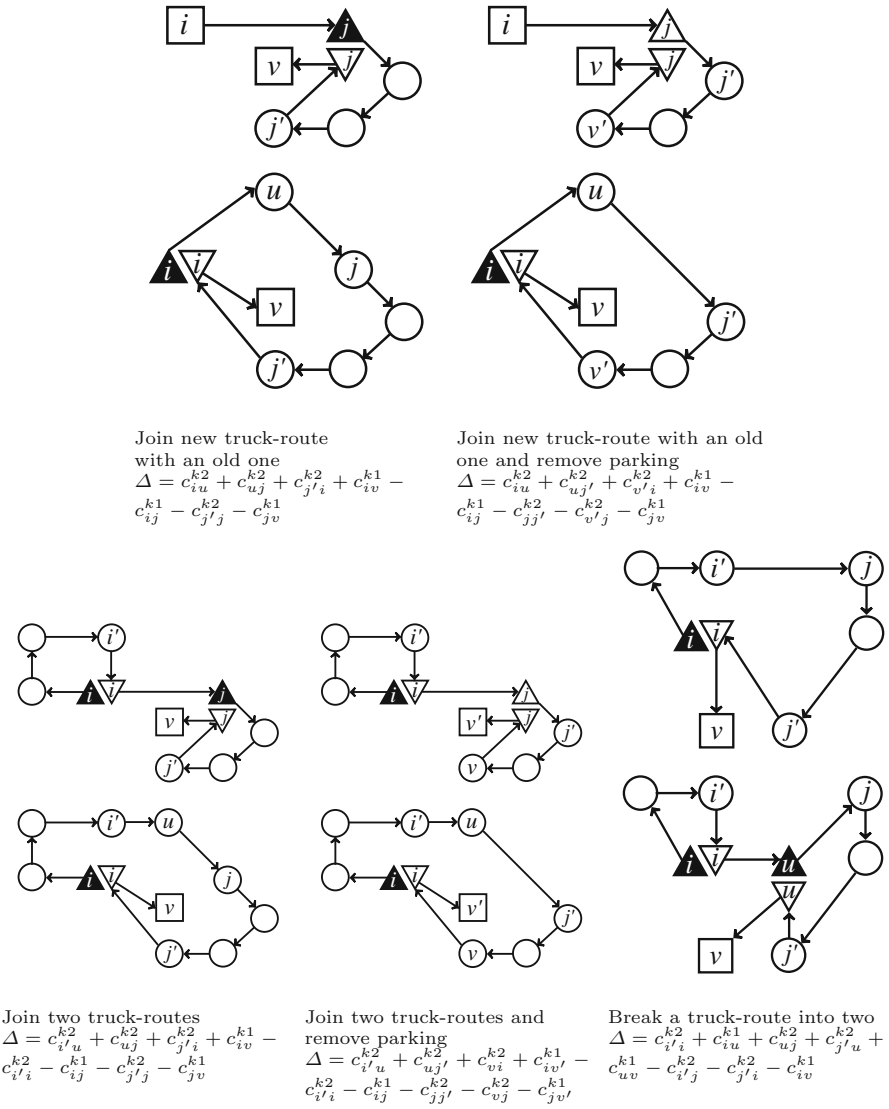


Fig. 3. Insertion cases with joined and broken truck-routes

using our insertion cases. To provide different solutions on all iterations the first customer in every route is chosen randomly from the remaining unserved customers. The solutions obtained by this approach for 100 and 100k iterations are shown in columns 2 and 3 of Table 1. The solutions of Batsyn & Ponomarenko (2014) algorithm for 100 and 100k iterations are presented in columns 4 and 5. Finally, the solutions found by the suggested new heuristic for 100 and 100k iterations are reported in columns 6 and 7.

Table 1. Computational experiments

Instance	Simple greedy, 100	Simple greedy, 100k	Batsyn & Ponomarenko (2014), 100	Batsyn & Ponomarenko (2014), 100k	New heuristic, 100	New heuristic, 100k
Novosibirsk	104026	103660	104148	103660	103757	103660
Udmurtia	–	–	1312720	1277967	1222630	1218939
Bashkortostan01	–	–	960130	941629	916792	902353
Bashkortostan07	–	–	1074651	1060665	1036272	1028024
Bashkortostan08	–	–	1027655	1021944	995317	984917
Bashkortostan09	–	–	1012611	1002596	974641	965984
Bashkortostan10	–	–	957746	945515	939019	929642
Novgorod03	–	–	–	–	1190419	1185121
Novgorod04	–	–	–	–	1082395	1073438
Novgorod05	–	–	–	–	1138652	1128973
Novgorod06	–	–	–	–	1184675	1171648
Novgorod07	–	–	–	–	1243691	1229926
Kropotkin	–	–	–	–	717075	708625

The simple greedy heuristic is able to find a feasible solution only for the first small instance of 55 customers. This is due to random assignment of vehicles and treating soft time windows as hard ones. The heuristic of Batsyn & Ponomarenko (2014) is not able to find a feasible solution for 7 instances because it always chooses the biggest available vehicle which can serve the first customer in the route and does not analyze that later a vehicle of this type will be needed for other customers which could not be served by other vehicles. The suggested heuristic obtains feasible solutions for all instances and provides the lowest costs.

4 Conclusions

In this paper we have suggested an iterative greedy randomized heuristic which efficiently addresses such features of real-life vehicle routing problems as heterogeneous fleet of vehicles, site-dependency feature, complex truck-and-trailer routes with transshipment locations, soft time windows along with hard time windows, split deliveries. To the best of our knowledge this is one of the most general formulations considered in literature for truck and trailer routing problems. We have implemented our algorithm and integrated it to the software system of a big retail company. And it shows better results on real-life instances than their experienced staff could get using their old software which partly automates construction of routes.

Currently we are working on further improvements of the suggested approach. We consider different neighbourhood structures for local search. The idea is to allow infeasible solutions during local search by penalizing them in the

objective function. This makes it possible to use many different simple and efficient neighbourhoods. However due to the penalties local search will often get stuck in bad, but feasible solutions. To overcome this problem we are going to apply the tabu search approach.

Acknowledgments. The authors are supported by LATNA Laboratory, NRU HSE, RF government grant, ag. 11.G34.31.0057.

References

Batsyn, M., Ponomarenko, A.: Heuristic for a real-life truck and trailer routing problem. Procedia Comput. Sci. **31**, 778–792 (2014). The 2nd International Conference on Information Technology and Quantitative Management, ITQM 2014

Caramia, M., Guerriero, F.: A heuristic approach for the truck and trailer routing problem. J. Oper. Res. Soc. **61**, 1168–1180 (2010a)

Caramia, M., Guerriero, F.: A milk collection problem with incompatibility constraints. Interfaces **40**(2), 130–143 (2010b)

Chao, I.M.: A tabu search method for the truck and trailer routing problem. Comput. Oper. Res. **29**(1), 33–51 (2002)

Drexl, M.: Branch-and-price and heuristic column generation for the generalized truck-and-trailer routing problem. J. Quant. Methods Econ. Bus. Adm. **12**(1), 5–38 (2011)

Hoff, A.: Heuristics for rich vehicle routing problems. Ph.D. Thesis. Molde University College (2006)

Hoff, A., Lokketangen, A.: A tabu search approach for milk collection in western Norway using trucks and trailers. In: The Sixth Triennial Symposium on Transportation Analysis TRISTAN VI, Phuket, Tailand (2007)

Lin, S.-W., Yu, V.F., Chou, S.-Y.: Solving the truck and trailer routing problem based on a simulated annealing heuristic. Comput. Oper. Res. **36**(5), 1683–1692 (2009)

Lin, S.-W., Yu, V.F., Chou, S.-Y.: A note on the truck and trailer routing problem. Expert Syst. Appl. **37**(1), 899–903 (2010)

Lin, S.-W., Yu, V.F., Lu, C.-C.: A simulated annealing heuristic for the truck and trailer routing problem with time windows. Expert Syst. Appl. **38**, 15244–15252 (2011)

Scheuerer, S.: A tabu search heuristic for the truck and trailer routing problem. Comput. Oper. Res. **33**, 894–909 (2006)

Semet, F., Taillard, E.: Solving real-life vehicle routing problems efficiently using tabu search. Ann. Oper. Res. **41**, 469–488 (1993)

Semet, F.: A two-phase algorithm for the partial accessibility constrained vehicle routing problem. Ann. Oper. Res. **61**, 45–65 (1995)

Solomon, M.M.: Algorithms for the vehicle routing and scheduling problem with time window constraints. Oper. Res. **35**, 254–265 (1987)

Villegas, J.G., Prins, C., Prodhon, C., Medaglia, A.L., Velasco, N.: A GRASP with evolutionary path relinking for the truck and trailer routing problem. Comput. Oper. Res. **38**(9), 1319–1334 (2011a)

Villegas, J.G., Prins, C., Prodhon, C., Medaglia, A.L., Velasco, N.: Heuristic column generation for the truck and trailer routing problem. In: International Conference on Industrial Engineering and Systems Management IESM2011, Metz, France (2011b)