OPTIMIZATION OF MULTI-DEPOT PERIODIC VEHICLE ROUTING PROBLEM WITH TIME WINDOW

Anita AGÁRD1,*, László KOVÁCS1 and Tamás BÁNYAI2
1 University of Miskolc, Institute of Informatics, Egyetemváros, 3515, Miskolc, Hungary
2 University of Miskolc, Institute of Logistics, Egyetemváros, 3515, Miskolc, Hungary
Corresponding author, Anita Agárdi, e-mail: agardianita@iit.uni-miskolc.hu

ABSTRACT: Logistics is one of today's most important industries. It is important to store and transport raw materials, products cost-effectively. The article presents a specific delivery task, the Multi-Depot Periodic Vehicle Routing Problem with Time Window. In case of the problem several customers must be visited and must satisfy their demands. The vehicles start their route from one of the several depots, visit some customers and return to the depot from which they started their route. The customers have time window, which means that they must be visited within a predefined time interval. The periodic keyword means that the customers must be visited not once, but periodically. This means, that a periodic time is given, and the number of visits of each customer within this periodic time are also known in advance. The goal is the minimization of the length of the route. This problem is solved in this paper with construction and improvement algorithms. The presented construction algorithms are the Nearest Neighbor, Insertion Heuristics and Greedy algorithm. The presented improvement algorithms are the Firefly Algorithm, Harmony Search, Particle Swarm Optimization, Simulated Annealing and Tabu Search algorithms. Based on the test results the improvement of construction algorithms gave better performance than improving randomly generated solutions.

KEYWORDS: vehicle routing, heuristic optimization, logistics

1 INTRODUCTION

One of the most important tasks of logistics is to deliver right goods to the right place at the right time. Such a task is the vehicle routing problem, which can mean in-plant; out-plant material flow process or full supply chain management. Various versions of the Vehicle Routing Problem (VRP) have developed over the years to meet the needs of Industry 4.0. In this article the Multi-Depot Periodic Vehicle Routing Problem with Time Window is described. In case of this VRP several vehicles transport products from some depots to several customers. The customers have time window, which means that they must be visited within a time interval. Periodic problem means, that the customers must be visited not once, but periodically. The goal is the minimization of the length of the route. The Vehicle Routing Problems are NP-hard problems, various heuristics have been developed to solve the tasks. In this article construction and improvement heuristics are described. The construction algorithms construct one solution. They give locally the best steps, but with the exclusively usage of these algorithms we do not get the global optimum. The improvement algorithms improve one or more solutions iteratively. Improvement algorithms can be used to improve randomly generated solutions or solutions constructed with the construction algorithms. In this article the Nearest Neighbor, Insertion Heuristics and Greedy algorithms are used as construction algorithms, and the Simulated Annealing, Tabu Search, Particle Swarm Optimization, Firefly Algorithm and Harmony Search are used as improvement algorithms. Based on the test results the improvement of construction algorithms gave better performance than improving randomly generated solutions.

2 THE VEHICLE ROUTING PROBLEM

The Vehicle Routing Problem (VRP) is a logistical problem. In the classical problem vehicles serve the demands of the customers from a depot. The minimization of the length of the route is the goal function. During the years, the VRP has several types, constraints, for example:

- Capacity constraint [1]: the vehicles have capacity limit for the goods to be transported.
- Multi-product [2]: the vehicles transport multiple products.
- Multiple types of vehicles [3]: there are multiple types of vehicles. The vehicles can differ from their capacity limit, or they have some constraint which goods to be transported.
Environmentally-friendly (Green) problem [4]: not the minimization of the length of the route but the minimization of the fuel consumption is the goal function.
• Multi-depot [5]: there are multiple depots from which the vehicles can start their route, and after visit the customers the vehicles end their route at the same depot from which they have started their route.
• Periodic problem [6]: the customers must be visited not once, but periodically
• Time window [7]: the customers must be visited within a given interval.
• Fuzzy problem [8]: some factors (for example the time window, the demands of the customers or the travel time between the customers) are given with fuzzy numbers.
• Environmentally friendly problem [9]: not the minimization of the length of the route but the minimization of the fuel consumption is the goal function.
• Occasional driver [10]: this problem also involves the use of own vehicles and leased vehicles. The minimization of the number of leased vehicle is the goal function.
• Traffic congestion [11]: this problem also takes into account some traffic factors.
• Multi-echelon problem [12]: the goods are first delivered from the depots to satellites and then transported to additional satellites or customers.
• Perishable food delivery [13]: the vehicles transport perishable foods; therefore the goods have to be transported within a certain period of time.
• Cross docking [14]: the products are collected first, and then delivered.
• Stochastic problem [15]: some factors are not static; we know only the probability of the value of these factors.
• Location routing problem [16]: the goal is the selection of depots, from which the vehicles start their route.
• Open vehicle routing problem [17]: each vehicle starts their route from one of depots, but does not return to any depot.
• Inter-depot route [18]: the vehicles start their route from a depot, but they do not have to return to the same depot from which they started their route.
• Site-dependent problem [19]: only some type of vehicle can be visited by each customer.
• Selective problem [20]: products are delivered only to certain locations, which are the most favorable in terms of profit.


3 THE PERIODIC MULTI-DEPOT VEHICLE ROUTING PROBLEM WITH TIME WINDOW

In the case of Periodic Multi-Depot Vehicle Routing Problem with Time Window (PMDVRPTW), the position of the depots is given. This is denoted by D1 and D2 in Figure 1. We also know the position, the demand, and the time window of the customers. The positions are indicated with integer numbers, the demands are indicated with D, the time windows with TW. There is also a period time. In our example in Figure 1 the period time is 4 day. We also know how much the customer must be visited within this period. This is indicated with PN. We also know the number and capacity limit of the vehicles. The vehicles start their route from one of the depots, visits some customers, than return to the depot from which they started their route. The goal is the minimization of the length of the route.

Figure 2 illustrates the solution of the Figure 1 problem. In Figure 2 in the first day the first vehicle starts the route from D1 depot, than visits customer 1, than customer 2, 4, 5 than returns to the D1 depot. The second vehicle starts the route from D2 depot, than visits customer 6, 8, 9 than returns to the D2 depot. In the second day the first vehicle starts the route from D1 depot, then visits customer 3, 5 then returns to the D1 depot. The second vehicle starts the route from D2 depot, then visits customer 6, 7, 9 then returns to the D2 depot. In the third day the first vehicle starts their route from D1 depot, then visits customer 1, 2, 4, 5 then returns to the D1 depot. The second vehicle starts their route from D2 depot, then visits customer 6, 7, 8, 9 then returns to the D2 depot. In the last day the first vehicle starts the route from D1 depot, then visits customer 3, 5 then returns to the D1 depot. The second vehicle starts their route from D2 depot, then visits customer 6, 7, 9, 10 then returns to the D2 depot.

In the following we present the mathematical model of the Periodic Multi-Depot Vehicle Routing Problem with Time Window. [44]
Fig. 1. The Periodic Multi-Depot Vehicle Routing Problem with Time Window (PMDVRPTW)

Fig. 2 The route of the 1st the 2nd the 3rd and the 4th days
The objective function is the minimization of the length of the route:

\[ Z = \sum_{i=1}^{n_c+n_d} \sum_{j=1}^{n_c+n_d} \sum_{v=1}^{n_v} \sum_{p=1}^{n_p} c_{ij} x_{ij}^v \]

One customer must be visited within a period once:

\[ \sum_{j=1}^{n_c+n_d} x_{ij}^v = 1 \quad \forall j \in CU \]  
\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v = 1 \quad \forall i \in CU \]

The vehicles must start their route from a depot and after visit the customers they must be end at the depot, from which they started their route:

\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v \leq 1 \quad \forall v \in VE, \forall i \in CU \]  
\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v \leq 1 \quad \forall v \in VE, \forall j \in CU \]

Route continuity:

\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v = \sum_{i=1}^{n_c+n_d} x_{ij}^v \quad \forall v \in VE \]

Time window:

\[ a_i \leq y_i^v \quad \forall v \in VE, \forall i \in CU \]  
\[ y_i^v + s_t \leq b_i \quad \forall v \in VE, \forall i \in CU \]

The vehicle does not exceed the capacity limit:

\[ \sum_{i=1}^{n_c+n_d} d_{i,j} x_{ij}^v \leq c_a \quad \forall v \in VE \]

There is no inter-depot route:

\[ x_{ij}^v = 0 \quad \forall v \in VE, \forall i,j \in DE \]

The number of required customer visits in a period must be satisfied:

\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v = \sum_{i=1}^{n_c+n_d} x_{ij}^v = v_i \quad \forall j \in CU \]  
\[ \sum_{i=1}^{n_c+n_d} x_{ij}^v = \sum_{i=1}^{n_c+n_d} x_{ij}^v = v_i \quad \forall i \in CU \]

4 OPTIMIZATION ALGORITHMS

4.1 Construction Algorithms

The construction algorithms build one solution. They make locally the best steps, but they stuck to local optimum in most of the cases. The implemented algorithms are the Nearest Neighbor algorithm, Insertion Heuristics algorithm family (Nearest Insertion, Farthest Insertion, Cheapest Insertion, and Arbitrary Insertion) and Greedy algorithm. Figure 3 illustrated the pseudo code of the Nearest Neighbor algorithm, Figure 4 the Insertion Heuristics and Figure 5 the Greedy algorithm.

BEGIN PROCEDURE
The first city is taken randomly.
WHILE not all cities have been selected DO
IF Nearest Insertion THEN
Take the next (unselected) city which is the nearest to the selected cities.
ELSE IF Farthest Insertion THEN
Take the next (unselected) city which is the farthest to the selected cities.
ELSE IF Cheapest Insertion THEN
Take the next (unselected) city which insertion is the cheapest.
ELSE IF Arbitrary Insertion THEN
Take the next (unselected) city arbitrary.
END IF
Insert the selected city to the tour so that the insertion cost is minimal.
END WHILE
END PROCEDURE

Fig. 3 The pseudo code of the Nearest Neighbor algorithm [44]

BEGIN PROCEDURE
Sort the edges by length.
WHILE the tour does not contain all nodes DO
Select the shortest edge, which does not create a tour, in which the peak of any node is greater than 2.
END WHILE
END PROCEDURE

Fig. 4 The pseudo code of the Insertion Heuristics algorithm family [45-49]

BEGIN PROCEDURE
The first city is taken randomly.
WHILE not all cities have been selected DO
END PROCEDURE

Fig. 5 The pseudo code of the Greedy algorithm [50]

4.2 Improvement Algorithms

The improvement algorithms improve iteratively one or more possible solutions. The Simulated Annealing and Tabu Search algorithms operate on one solution, the Particle Swarm Optimization, Firefly Algorithm and the Harmony Search operates on the population of solutions.

4.2.1 Simulated Annealing

Simulated Annealing (SA) simulates the cooling of metals. In metallurgy, cooling is a metal-hardening, starch process. The material is then heated to high temperature and then gradually cooled to bring the material into a low-energy crystalline state. [51]

During Simulated Annealing we take random steps. If we get a better solution during the step, we always accept it. If we get a worse solution, we only accept it with a certain probability. Probability decreases with ΔE exponentially. Even with reduced T temperatures, it is less likely to accept worse solutions. Accepting bad solutions at higher T values (at the start of the algorithm) is more likely to occur, while T decreases become more unlikely. Accepting the worse solutions also allows us to get out of the local optimum. In case of sufficiently
slow cooling, the algorithm finds the global optimum. [51]

BEGIN PROCEDURE
Start with a solution. Initially this will be the current solution. Mark this as $S_{\text{actual}}$.

WHILE termination criteria is not met DO

WHILE $L$ is not reached DO

Create a neighbor of $S_{\text{actual}}$. Mark this as $S_{\text{neig}}$.

Calculate $\Delta E$ with the following formula:
$$\Delta E = E(S_{\text{neig}}) - E(S_{\text{actual}}).$$

IF $\Delta E < 0$ THEN DO

$S_{\text{actual}} = S_{\text{neig}}$.

ELSE DO

END IF

END WHILE

END WHILE

END PROCEDURE

Fig. 6 The pseudo code of the Simulated Annealing algorithm [51]

The parameters of the algorithm are: $0 < \alpha < 1$ is the cooling factor, $T > 0$ is the current temperature and $L$ is the length of the process.

In case of the Simulated Annealing we have chosen the 2-opt [52] as a neighborhood operator. In case of this operator two edges are changed.

4.2.2 Tabu search

The tabu search maintains a so-called tabu list that contains the solutions that were involved in the last few steps. As you proceed, you can only move to the neighbor of the current solution that is not in the tabu list. The tabu step is modified in each iteration. We insert a new element at the end, and if it is full, we remove the oldest item. Steps of the algorithm: [53]

BEGIN PROCEDURE
Taking a possible solution. At this point, this will be the optimal solution, so mark it as $S_{\text{best}}$, which is inserted into the tabu list.

WHILE termination criteria is not met DO

Forming $S_{\text{best}}$ neighbors and choosing the best one that is not included in the tabu list. Let’s mark this as $S_{\text{neig}}$.

Insert $S_{\text{neig}}$ as the last item in the tabu list. (If the tabu list is full, delete the first item from the tabu list.)

IF $S_{\text{neig}}$ is better than $S_{\text{best}}$, DO

$S_{\text{best}} = S_{\text{neig}}$.

END IF

END WHILE

END PROCEDURE

Fig. 7 The pseudo code of the Tabu Search algorithm

The input of the algorithm is the size of the tabu list. This greatly influences the accuracy of the solution and the time of calculation.

In case of the Tabu Search we have chosen the 2-opt [52] as a neighborhood operator. In case of this operator two edges are changed.

4.2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) maintains potential solutions, i.e., a population of particles. The algorithm moves the particles through simple mathematical formulas in the search space. The movement of particles is driven by the best search space positions found. The best search space positions are refreshed when particles are found with better points. The movement of the particles is determined by their best known position and the best known position of the swarm.

Description of the algorithm: [54]

Given the following function: $f: \mathbb{R}^D \rightarrow \mathbb{R}$

Search for: $x_{\text{opt}} = \min f(x)$

Where $\mathbb{R}^D$ denotes the search space defined by the vector $x$, which indicates the possible solutions. $x_{\text{opt}}$ denotes the optimal solution, which means maximizing the function $f$.

During PSO, each solution is called a particle that represents a point in the $D$ dimensional space. $D$ denotes the number of parameters to be optimized. So the $i$. position of the particle can be indicated with the vector $x_i$.

$$x_i = [x_{i1} x_{i2} x_{i3} ... x_{id}]$$

The population consists of $N$ possible solutions that form the swarm:

$$X = \{x_1 x_2 x_3 ... x_N\}$$

In order to find the optimal solution to the problem, we can define a displacement (so iteratively changing the position of the swarm). The movement is given with the following formula:

$$x_i(t + 1) = x_i(t) + v_i(t + 1)$$

where $t$ and $t + 1$ mean two iterations during the algorithm. $v_i$ is the velocity vector of $i$. particle. Speed vectors control the path of particles in the search space. The management of the speed is based on three criteria:

- Prevent (or control) the sudden change of direction by means of the inertia associated with the movement of the particles.

- Consider the best position of the particles so far using the cognitive component of the algorithm.

- The social component is responsible for the relationship between the movement of particles and the swarm.

Based on these, the $i$. particle displacement (velocity) is given by the formula:
where $p_i$ is the best position of the $i$-th particle and $g$ is the best position for the swarm. The acceleration constants $c_1$ and $c_2$, which are real numbers, are usually in the interval $[0,4]$. These are also called cognitive and social coefficients. These determine the length of the steps of the particles relative to the best of their own and of the swarm position. $w$ denotes the inertia weight, and $R_1$ and $R_2$ are random real numbers from the $[0,1]$ interval. This means that social and cognitive components have a stochastic influence on updating speed.

BEGIN PROCEDURE
Initialization the positions of the particles, i.e. $x_i(0)$, with random vector.
Initialization the best positions of the particles, i.e. $p_i(0)$, with the starting position of the particles, i.e., $x_i(0)$.
Calculating the fitness values of the particles and initialize the best particle, i.e. $g$.
WHILE termination criteria is not met DO
Updating the speed of all particles using the formula:
$$v_i(t + 1) = w v_i(t) + c_1(p_i(t) - x_i(t))R_1 + c_2(g - x_i(t))R_2$$
Updating the particle position using the following formula:
Calculating the fitness values of the particles.
Updation the best position of the particle, i.e. $p_i$, if the current position, i.e. $x_i(t + 1)$ is better than the best position of the particle so far.
Updation the best position of the swarm, i.e. $g$, if the current position, i.e. $x_i(t + 1)$ is better than the current best position of the swarm.
END WHILE
The best solution at the end of the algorithm is $g$.
END PROCEDURE

Fig. 8 The pseudo code of the basic Particle Swarm Optimization algorithm [54]

BEGIN PROCEDURE
Creating a particles corresponding to the number of particles. The position of a particle, that is, $x_{id}(t)$ is a permutation. The permutations are either randomly generated or obtained by a construction algorithm. Thus, the best position of the particles is the initialization of $p_{id}$ with the best initial permutation.
Then, for each particle is randomly generated velocity, i.e. $v_{id}(t)$. The velocity means in [55] paper presented swap sequence.
WHILE termination criteria is not met DO
The global best particle, i.e., $g_d$, is determined by evaluating the particles.
FOR all particles DO
To update the particle velocity using the formula:
$$v_{id}(t + 1) = v_{id}(t) \oplus \alpha(p_{id} - x_{id}(t)) \ominus \beta(g_d - x_{id}(t)).$$
In the formula, $v_{id}(t + 1)$ represents the new velocity (swap sequence), $p_{id} - x_{id}(t)$ the basic swap sequence (shown in [55]) between the best and current position (permutation) of the particle, while $g_d - x_{id}(t)$ provides the basic swap sequence between the position of the global best particle and the current position of the particle. $\alpha, \beta \in [0,1]$ are random numbers. With these, we do not complete the entire exchange sequences, only some of them. So we do the swapping operations only with some probability. The operation $\oplus$ here also refers to the sequential execution of the swap sequences.
Creating a Basic Swap Sequence from $v_{id}(t + 1)$ based on [55] paper.
Updation the actual permutation of the particle by performing the new basic swap sequence (velocity) on the permutation, i.e. $x_{id}(t + 1) = x_{id}(t) + v_{id}(t + 1)$.
Updation $p_{id}$, so the best position of the particle if the current position, i.e. $x_{id}(t + 1)$ is better than the best position ever.
Updation $x_{id}(t + 1)$ which is the best position of the swarm if the current position, i.e. $x_{id}(t + 1)$ is better than the best position of the swarm.
END FOR
END WHILE
RETURN with $g_d$ so the position of the best particle
END PROCEDURE

Fig. 9 The pseudo code of the implemented Particle Swarm Optimization algorithm

4.2.4 Firefly Algorithm
The firefly algorithm is based on the social behavior of fireflies. The following four rules define the algorithm: [56]
1. Unisexes, a firefly with all its light attracts all other fireflies regardless of gender.
2. The attraction of fireflies is proportional to their brightness and depends on the distance between them. The brighter firefly attracts less bright.
3. If the firefly is not brighter then the firefly will move randomly.
4. The brightness of fireflies is influenced by their fitness value.

The algorithm requires the following formulas: [56]
- $I$: The brightness of fireflies. In the simplest case $I = f(x)$.
- $\beta$: attraction of fireflies $\beta = \beta_0 e^{-\gamma r^2}$.
- $r$: Euclidean distance.
$$r_{ij} = \sqrt{\sum_{k=1}^{d}(x_{ik} - x_{kj})^2}$$
In the formula, $d$ is the number of dimensions.
• $x_i$: new positions (movements) of the individuals: $x_i = x_i + \beta (x_j - x_i) + \alpha \xi_i$. In the formula, $\xi_i$ is a random number and $\alpha$ is a scaling parameter.

BEGIN PROCEDURE
Generating initial fireflies.
WHILE termination criteria is not met DO
FOR all fireflies pairs DO
  Determine the distance ($\gamma$) of two fireflies. The less bright firefly moves toward the brighter firefly. The position of the less bright firefly is determined by the formula: $x_i = x_i + \beta (x_j - x_i) + \alpha \xi_i$. Determining attractively by distance, i.e. $\beta = \beta_0 e^{-\gamma^2}$
  Evaluating new solutions, calculating light intensities ($I$).
END FOR
END WHILE
END PROCEDURE

BEGIN PROCEDURE
Creating a permutation corresponding to the number of fireflies. A firefly will mean a permutation. Evaluating the fireflies. The best solution is stored.
FOR termination criteria is not met DO
  FOR all the possible firefly pairs DO
    We determine which firefly is the brighter of the two fireflies. Determine the distance between the two fireflies, i.e. $x_{ij}$. The distance is given by the number of exchanges between the two permutations, i.e. the size of the Basic Swap Sequence [55].
    The less bright fireflies will move toward the brighter fireflies. This is accomplished by generating a random number between 2 and $x_{ij}$. On the permutation of the less glossy firefly, the 2-opt operation is performed as much the random number [57]. If you get a better solution with the permutation, the result will be the new firefly; otherwise the old permutation will remain. The permutation of the brighter firefly will not change.
    We will update the best solution if the new permutation gives better results than the best solution so far.
  END FOR
END FOR
END PROCEDURE

Fig. 10 The pseudo code of the Firefly algorithm

During the algorithm, the user should specify $\alpha, \beta$ and $\gamma$ control parameters.

BEGIN PROCEDURE
Creating a permutation corresponding to the number of fireflies. A firefly will mean a permutation.
Evaluating the fireflies. The best solution is stored.
FOR termination criteria is not met DO
  FOR all the possible firefly pairs DO
    We determine which firefly is the brighter of the two fireflies.
    Determine the distance between the two fireflies, i.e. $x_{ij}$. The distance is given by the number of exchanges between the two permutations, i.e. the size of the Basic Swap Sequence [55].
    The less bright fireflies will move toward the brighter fireflies. This is accomplished by generating a random number between 2 and $x_{ij}$. On the permutation of the less glossy firefly, the 2-opt operation is performed as much the random number [57]. If you get a better solution with the permutation, the result will be the new firefly; otherwise the old permutation will remain. The permutation of the brighter firefly will not change.
    We will update the best solution if the new permutation gives better results than the best solution so far.
  END FOR
END FOR
END PROCEDURE

Fig. 11 The pseudo code of the implemented Firefly algorithm

4.2.5 Harmony Search

The Harmony Search algorithm simulates the natural processes of musical performances. The aim of the algorithm is to find the perfect harmony in the band. If the music played by the musicians is in harmony with each other, the musicians will note that harmony. In the algorithm, a good parameter that provides a high-fitness solution along with other parameters is included in the harmony memory matrix. [58]

BEGIN PROCEDURE
Initializing the parameters of the algorithm.
BEGIN PROCEDURE
Initializing the music memory.
WHILE termination criteria is not met DO
  Improving new music.
  Updating the music memory.
END WHILE
END PROCEDURE

Fig. 12 The pseudo code of the Harmony Search algorithm

Initializing the parameters [58]
The following parameters must be specified by the user: number of solutions generated (HMS), acceptance rate (HMCR), pitch adjustment rate (PAR), and stop condition e.g. number of iterations.

Initializing music memory [58]
Initializing music memory means that we generate random solutions that match the harmony memory size. Each $x$ solution consists of $N$ elements. For each solution we calculate the fitness function. The following formula represents the general structure of music memory:

$$HM = \begin{bmatrix} x_1^{max} & x_2^{max} & \cdots & x_N^{max} \\ x_1^{min} & x_2^{min} & \cdots & x_N^{min} \end{bmatrix}$$

Improving new music [58]
The new solution uses $x = (x_1, x_2, \ldots, x_n)$ two parameters:

- Harmony memory consideration rate (HMCR)
- Pitch adjustment rate (PAR)

With these two parameters, the algorithm can improve solutions locally and globally. The new solution is:

$$x_i \in \left\{ \begin{array}{c} [X_s^{x_1}, x_2^{x_1}, \ldots, x_N^{x_1}] \text{ with HMCR probability} \\ X_s \text{ with (1-HMCR) probability} \end{array} \right\}$$

where $X_s$ denotes all possible elements for each variable.

For each variable, the memory decides whether to tune or not. The second parameter, the pitch adjustment rate (PAR), controls the search for better solutions. We use the PAR parameter as follows:

$$x_i = \begin{cases} x_i + bw \ast u(-1,1) \text{ with PAR probability} \\ x_i \text{ with (1 – PAR) probability} \end{cases}$$

102
In the formula, \( bw \) is the bandwidth of the distance and \( U(-1,1) \) is a random number between -1 and 1.

**Updating music memory [58]**

Newly generated music will replace a worse music stored in the music memory. For this, of course, the worst music that music memory stores should be better.

**Stop condition test [58]**

The algorithm may end after a certain iteration or if it finds an acceptable solution (corresponding to the exit condition).

**BEGIN PROCEDURE**

We create a permutation corresponding to the size of the harmony memory. One harmony will mean a permutation.

Evaluate the elements of the harmony memory and sort them in order. The best solution will be the first element of the harmony memory, while the worst will be the last one.

**WHILE** termination criteria is not met **DO**

Creation a new harmony based the following ways [58]:
- Randomly selecting a city, it will be the first element of the permutation.
- **WHILE** not all cities is selected **DO**
  - **WHILE** the selected city is selected **DO**
    - **IF** probability < \( HMCR \) **DO**
      - Taking randomly selected element of the music memory i.e. a city.
    - **ELSE** **DO**
      - Taking the city closest to the last city of new music.
    **END IF**
  - **ELSE IF** probability < \( 1 - PAR \) **DO**
    - Taking a randomly generated city
  - **ELSE** **DO**
    - Taking the city closest to the last city of new music
  **END IF**
- **END IF**

**END WHILE**

Evaluation the new music.

**IF** new music is better than the worst element of the harmony **DO**

Instead of the worst element, insert new music into the music memory and sort the music memory.

**END IF**

**END WHILE**

**RETURN** the best element of the harmony memory.

**END PROCEDURE**

**5 TEST RESULTS**

<table>
<thead>
<tr>
<th><strong>Table 1. Parameters of the Firefly algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of iterations</strong></td>
</tr>
<tr>
<td><strong>Number of firesflies</strong></td>
</tr>
<tr>
<td>( \alpha )</td>
</tr>
<tr>
<td>( \beta )</td>
</tr>
<tr>
<td>( \gamma )</td>
</tr>
<tr>
<td><strong>Generated solutions</strong></td>
</tr>
<tr>
<td>Arbitrary Insertion: 10</td>
</tr>
<tr>
<td>Cheapest Insertion: 10</td>
</tr>
<tr>
<td>Farthest Insertion: 10</td>
</tr>
<tr>
<td>Nearest Insertion: 10</td>
</tr>
<tr>
<td>Greedy: 10</td>
</tr>
<tr>
<td>Nearest Neighbor: 10</td>
</tr>
<tr>
<td>Random: 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 2. Parameters of the Harmony Search algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of iterations</strong></td>
</tr>
<tr>
<td><strong>HMS</strong></td>
</tr>
<tr>
<td>( HMCR )</td>
</tr>
<tr>
<td>( PAR )</td>
</tr>
<tr>
<td><strong>Generated solutions</strong></td>
</tr>
<tr>
<td>Arbitrary Insertion: 10</td>
</tr>
<tr>
<td>Cheapest Insertion: 10</td>
</tr>
<tr>
<td>Farthest Insertion: 10</td>
</tr>
<tr>
<td>Nearest Insertion: 10</td>
</tr>
<tr>
<td>Greedy: 10</td>
</tr>
<tr>
<td>Nearest Neighbor: 10</td>
</tr>
<tr>
<td>Random: 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 3. Parameters of the Particle Swarm Optimization algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of iterations</strong></td>
</tr>
<tr>
<td><strong>Number of particles</strong></td>
</tr>
<tr>
<td>( \alpha )</td>
</tr>
<tr>
<td>( \beta )</td>
</tr>
<tr>
<td>Minimum initial velocity</td>
</tr>
<tr>
<td>Maximum initial velocity</td>
</tr>
<tr>
<td><strong>Generated solutions</strong></td>
</tr>
<tr>
<td>Arbitrary Insertion: 10</td>
</tr>
<tr>
<td>Cheapest Insertion: 10</td>
</tr>
<tr>
<td>Farthest Insertion: 10</td>
</tr>
<tr>
<td>Nearest Insertion: 10</td>
</tr>
<tr>
<td>Greedy: 10</td>
</tr>
<tr>
<td>Nearest Neighbor: 10</td>
</tr>
<tr>
<td>Random: 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Table 4. Parameters of the Simulated Annealing algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of iterations</strong></td>
</tr>
<tr>
<td>( \alpha )</td>
</tr>
<tr>
<td>Temperature (T)</td>
</tr>
<tr>
<td>Length (L)</td>
</tr>
<tr>
<td><strong>Generated solutions</strong></td>
</tr>
<tr>
<td>Arbitrary Insertion: 1</td>
</tr>
<tr>
<td>Cheapest Insertion: 1</td>
</tr>
<tr>
<td>Farthest Insertion: 1</td>
</tr>
<tr>
<td>Nearest Insertion: 1</td>
</tr>
<tr>
<td>Greedy: 1</td>
</tr>
<tr>
<td>Nearest Neighbor: 1</td>
</tr>
</tbody>
</table>

In the following we present the test result. Test results are created with four dataset.
Fig. 14 The test results of the first dataset

Fig. 15 Test results of the second dataset
Based in the first test result the Tabu Search algorithm and Simulated Annealing algorithm was equal effective. The Firefly Algorithm, the Particle Swarm Optimization and the Harmony Search algorithms gave a bit better solution than Tabu Search and Simulated Annealing algorithm in case of improving construction algorithms. Improving the solutions of Greedy, Arbitrary Insertion, Cheapest Insertion and Nearest Insertion is worth than improving solutions of other construction algorithms and improving randomly generated solutions.
The results of the second test run are similar to the first. During this test result, we can also see the big difference in improving randomly generated solutions and solutions made by construction algorithms.

The third test run is similar to the previous ones, so here the improvement of the construction algorithms proved to be significantly better than the improvement of the randomly generated solutions. Improvement of construction algorithms has been with 1000-1500 route units better than improving randomly generated solutions.

According to the worst results of the fourth test run, we can get better results with up to 2000 units of road if we improve the results of construction algorithms and not randomly generated solutions.

6 CONCLUSION

In this article we described the Periodic Multi-Depot Vehicle Routing Problem with Time Window. We described the mathematical model of this problem. After that we presented construction and improvement algorithms. The construction algorithms are the Nearest Neighbor, Construction Heuristics (Nearest Insertion, Farthest Insertion, Cheapest Insertion and Arbitrary Insertion) and Greedy algorithms. The improvement algorithms are the Firefly Algorithm, Harmony Search, Particle Swarm Optimization, Simulated Annealing and Tabu Search algorithms. Based on the test results the improvement of construction algorithms gave better performance than improving randomly generated solutions.

7 REFERENCES


[47] isye.gatech.edu: Traveling Salesman Problem Insertion Algorithms Nearest Insertion, forrás: https://www2.isye.gatech.edu/~mgoetsch/cali/VEHI CLE/TSP/TSP009__HTM (2019.07.02.)


8 NOTATION

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation of the symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>nc</td>
<td>Number of customers</td>
</tr>
<tr>
<td>$CU = { cu_1, ..., cu_{nc} }$</td>
<td>Customers</td>
</tr>
<tr>
<td>$D = { d_1, ..., d_{nc} }$</td>
<td>Demands of each customers</td>
</tr>
<tr>
<td>nv</td>
<td>Number of vehicles</td>
</tr>
<tr>
<td>$VE = { ve_1, ..., ve_{nv} }$</td>
<td>Vehicles</td>
</tr>
<tr>
<td>$CA = { ca_1, ..., ca_{nv} }$</td>
<td>Capacity of vehicles</td>
</tr>
<tr>
<td>nd</td>
<td>Number of depots</td>
</tr>
<tr>
<td>$DE = { de_1, ..., de_{nd} }$</td>
<td>Depots</td>
</tr>
<tr>
<td>np</td>
<td>Number of periods</td>
</tr>
<tr>
<td>$TW = { tw_1, ..., tw_{np} }$</td>
<td>The time windows of the customers, where $tw_i = { a_{ij}, b_{ij} }$</td>
</tr>
<tr>
<td>$ST = { st_1, ..., st_{np} }$</td>
<td>Service times of the customers</td>
</tr>
<tr>
<td>$VI = { vi_1, ..., vi_{nc} }$</td>
<td>The frequency of visits of customers in the period</td>
</tr>
<tr>
<td>$p$</td>
<td>Day of the period index</td>
</tr>
<tr>
<td>$i, j$</td>
<td>Vertex indices (depot, customer)</td>
</tr>
<tr>
<td>$\nu$</td>
<td>Vehicle index</td>
</tr>
<tr>
<td>$d$</td>
<td>Depot index</td>
</tr>
<tr>
<td>$x_{ij}^{vp}$</td>
<td>Decision variable. 1 if the $\nu$. vehicle travels from $i$. customer to $j$. customer directly in the $p$. day. 0 else</td>
</tr>
<tr>
<td>$\chi_i$</td>
<td>Arrival time to the $i$. node</td>
</tr>
</tbody>
</table>