Kybernetika

Nermin Kartli

Hybrid algorithms for fixed charge transportation problem

Kybernetika, Vol. 61 (2025), No. 2, 141–167

Persistent URL: http://dml.cz/dmlcz/152984

KYBERNETIKA — VOLUME *61* (2025), NUMBER 2, PAGES 141–167

HYBRID ALGORITHMS FOR FIXED CHARGE TRANSPORTATION PROBLEM

Nermin Kartli

In this paper, we consider the fixed-cost transportation problem. This problem is known to be NP-hard. Therefore, various heuristic and metaheuristic approaches have been proposed to find an approximate optimal solution. In this paper, we propose three hybrid algorithms that combine the ideas of metaheuristic and heuristic approaches in different ways. Two of the proposed algorithms consist of the sequential implementation of metaheuristic and heuristic algorithms, while the third one is a full hybrid algorithm designed by completely intertwining these two approaches. Experimental results on medium-size problems show that our proposed full hybrid algorithm provides approximately a 5% improvement over metaheuristic algorithms and a 4% improvement over heuristic algorithms. In addition, the improvement ratio increases as the size of the problem increases.

Keywords: genetic algorithms, transportation problem, fixed charge transportation problem, metaheuristic algorithms

Classification: 90C08, 90B06, 90C59, 90C10

1. INTRODUCTION

The supply chain problem refers to the global network that moves goods from producers to consumers. Interruptions and inefficiencies in this network can cause factory closures, labor shortages, and transportation bottlenecks. Natural disasters, epidemics, and regional wars are important factors that can create significant disruptions in the supply chain. These disruptions cause delays in production, higher transportation costs, and shortages in goods ranging from electronics to daily consumer goods.

Many businesses have tightly coordinated logistics and implement a "just-in-time" inventory model based on these logistics, which allows them to reduce their costs. Unexpected events disrupt this delicate balance, resulting in delays in production and stock shortages. In addition, such disruptions have a ripple effect, and the negative effects quickly spread to other businesses and consumers. This leads to inflationary pressure and a tense global economy. Accordingly, long-term solutions such as diversifying supply chains, improving logistics technology, and creating more resilient and adaptable systems are needed.

DOI: 10.14736/kyb-2025-2-0141

The transportation problem (TP) is an important part of supply chain management and aims to minimize total transportation costs by meeting supply and demand constraints. In this problem, the most cost-effective way to transport goods from multiple suppliers to multiple destinations is tried to be found. The assumption accepted for the mathematical model of the problem is that transportation costs are linearly proportional to the amount of goods transported. The transportation problem is the problem of optimizing routes to reduce costs for a company with multiple production facilities and distribution centers.

Fixed-charge transportation Problem (FCTP) is a problem in which transportation companies charge an additional fee regardless of the amount of goods transported. Although this demand is natural and common, it considerably increases the problem's difficulty, turning it into a problem that cannot be solved in polynomial time. On the other hand, companies need to be able to solve this problem to establish a balance between minimizing their costs and delivering their products efficiently.

The supply chain can be one, two, or multi-stage. Hitchcock [17] was the first to describe the transportation problem as a one-stage supply chain, sometimes referred to as the Hitchcock problem. The summary of the problem is as follows: It is desired to make a plan to transport the products from a certain number of suppliers to a given number of distributors at the least total cost.

In the classical transportation problem, transportation costs are directly proportional to the product transported. In other words, the transportation problem is a linear programming problem and Dantzig [10] made this formulation for the first time in 1947. But the classical transportation problem has 2 most important features:

1. Generally, it is desired to transport the products as a whole without being divided; mathematically, this means that the problem is an integer programming problem.

2. Every product is required to undergo transportation without any loss, meaning that the overall quantity of products held by suppliers before transportation must be equal to the total quantity of products expected to be present with distributors after transportation. This means that there is a balance condition in the problem.

In practical scenarios, transportation firms impose a fixed charge alongside a unit-based fee for each product. Including this condition, the challenge transforms into a fixed-charge transportation problem (FCTP). This problem was formulated for the first time by Hirch and Dantzig [16]. Unlike the classical version, no algorithm is known for achieving the optimal solution of FCTP within polynomial running time.

Balinski [6] first studied FCTP as an integer programming problem and proposed an algorithm to find the lower and upper bounds of the optimal solution. Subsequent studies were carried out in 2 directions:

1. Studies to improve the lower and/or upper limits of Balinski suggesting heuristic algorithms, and

2. Studies proposing various metaheuristic algorithms for FCTP. [1, 2, 3, 4, 8, 9, 42] are the studies in the first group. The studies such as [13, 18, 21, 22, 23, 28, 33, 35, 36] are in the second group. Jo et al. [23] proposed a spanning tree-based Genetic Algorithm (GA) to solve FCTP. Jawahar and Balaji [21] also used GA

to find an approximate solution of FCTP. Jawahar et al. [22] proposed to use a simulated annealing algorithm for the multi-period FCTP. Raj and Rajendran [36] used GA for two-stage FCTP. El-Sherbiny and Alhamali [13] developed a hybrid particle swarm algorithm with artificial immune learning for solving FCTP. Lotfi and Tavakkoli–Moghaddam [28] proposed GA using priority-based encoding with new operators for FCTP. Panicker et al. [33] used an ant colony optimization algorithm for two-stage FCTP. Pop et al. [35] described a hybrid algorithm that combined a steady-state GA with a local search procedure for solving two-stage FCTP. Hong et al. [18] proposed to use an ant colony optimization algorithm for two-stage FCTP.

If some parameters in the transportation problem contain uncertainty, modeling is done with interval analysis, stochastic analysis, or fuzzy sets. Many successful studies have been published in this direction recently, such as [7, 11, 14, 19, 20, 26, 29, 32, 38, 39, 40].

Recently, Kartli et al. [24, 25] proposed new heuristic algorithms for the initial feasible and optimal solutions of the FCTP. Unlike the algorithms in the first group above, Kartli et al.'s [25] algorithm solves the fixed charge problem without turning it into a classical TP problem. The main idea of this algorithm is to minimize the cost corresponding to the 2×2 dimensional square submatrices of the feasible solution looked at each step.

In this study, we consider the one-stage fixed-charge transportation problem. We propose a hybrid algorithm that combines Lotfi and Tavakkoli–Moghaddam's algorithm [28] and Kartli et al.'s [25] algorithms to find the optimal solution to the problem. Experiments show that the proposed algorithm gives better results than both the above-mentioned algorithms.

2. PROBLEM FORMULATION

There are m suppliers and n distributors. The cost of transporting one unit of goods from supplier i to distributor j is c_{ij}. In addition, if at least one unit of goods has been transported from the supplier i to the distributor j, a fixed charge a_{ij} must be paid. The capacity of the supplier i is s_i and the capacity of the distributor j is d_j. All given are positive integers and the balance condition is satisfied:

$$\sum\nolimits_{j=1}^{n} d_j = \sum\nolimits_{i=1}^{m} s_i. \tag{1}$$

Let x_{ij} denote the quantity of product transported from supplier i to distributor j. We can write this problem as follows:

Minimize the objective function

$$F(X,Y) = \sum\nolimits_{i=1}^{m} \sum\nolimits_{j=1}^{n} (c_{ij}x_{ij} + a_{ij}y_{ij}) \tag{2}$$

under constraints

$$\sum\nolimits_{j=1}^{n} x_{ij} = s_i \tag{3}$$

$$\sum\nolimits_{i=1}^{m} x_{ij} = d_j. \tag{4}$$

For all i and j

$$x_{ij} \geq 0 \tag{5}$$

$$y_{ij} = \begin{cases} 1, \text{ if } x_{ij} > 0 \\ 0, \text{ if } x_{ij} = 0. \end{cases} \tag{6}$$

3. PRIORITY-BASED GENETIC ALGORITHM

This section discusses the priority-based genetic algorithm (PbGA) proposed by Lotfi and Tavakkoli–Moghaddam [28].

The algorithm consists of Decoding, Opex, Pex, OpexMutation, and Main functions. The meanings of the variables used for the input of these functions are as follows:

m : number of suppliers
n : number of distributors
A : fixed charges matrix
C : transportation cost matrix for the unit goods
S : capacity vector of the suppliers
D : capacity vector of the distributors
V : chromosome representing a feasible solution

Apart from the inputs of the problem, the algorithm has 2 parameters such as *pop_size* and *max_gen*. The parameter *pop_size* indicates the number of feasible solutions (number of populations) handled in each step, and *max_gen* indicates the maximum number of iterations (stopping criterion). If there are m suppliers and n distributors in the given problem, the algorithm initially generates chromosomes $V_1, V_2, \ldots, V_{pop\_size}$. Each chromosome is a random permutation of the set $\{1, 2, \ldots, m+n\}$.

3.1. Decoding function

The Decoding function produces a feasible solution corresponding to the given chromosome V. This function does this by performing the following operations: Initially, all cells of the feasible solution $X = (x_{ij})$ are assigned zeros. Then the index k of the maximum number contained in the chromosome V is found. If $k \leq m$, then we assign $i^* = k$ and then for every $j = 1, 2, \ldots, n$ satisfying the condition $V[m+j] \neq 0$ we calculate the numbers

$$W_{i^*,j} = c_{i^*,j} + \frac{a_{i^*,j}}{\min\{s_{i^*}, d_j\}}.$$

Let the smallest value of these numbers be attained for the index $j = j^*$. Otherwise, that is, if $k > m$, then we assign $j^* = k - m$ and then for every $i = 1, 2, \ldots, m$ satisfying the condition $V[i] \neq 0$ we calculate the numbers

$$W_{i,j^*} = c_{i,j^*} + \frac{a_{i,j^*}}{\min\{s_i, d_{j^*}\}}.$$

Let the smallest value of these numbers be attained for the index $i = i^*$. In the next step, we assign

$$x_{i^*,j^*} \quad = \quad \min\{s_{i^*}, d_{j^*}\}$$

$$\begin{aligned} s_{i^*} &= s_{i^*} - x_{i^*,j^*} \\ d_{j^*} &= d_{j^*} - x_{i^*,j^*}. \end{aligned}$$

If $s_{i^*} = 0$ then we put $V[i^*] = 0$; if $d_{j^*} = 0$ then we put $V[m + j^*] = 0$. If after all these operations there is still a nonzero value among the first m coordinates of the chromosome V, we repeat everything starting from calculating the index k.

After finding a feasible solution X corresponding to the given chromosome V with the help of the Decoding function, the transportation cost is calculated according to formula 2. The pseudo-code of the Decoding function is given in Algorithm 1.

Algorithm 1: Decoding(V, m, n, A, C, S, D).

```
X ← 0
cont ← 1
while cont = 1 do
    k ← argmax(V[1], ..., V[m + n])
    if k ≤ m then
        istar ← k
        for j ← 1 to n do
            W[istar, j] ← C[istar, j] + A[istar, j]/min{S[istar], D[j]}
        jstar ← argmin(W[istar, 1], ..., W[istar, n])
    else
        jstar ← k − m
        for i ← 1 to m do
            W[i, jstar] ← C[i, jstar] + A[i, jstar]/min{S[i], D[jstar]}
        istar ← argmin(W[1, jstar], ..., W[m, jstar])
    X[istar, jstar] ← min{S[istar], D[jstar]}
    S[istar] ← S[istar] − X[istar, jstar]
    D[jstar] ← D[jstar] − X[istar, jstar]
    if S[istar] = 0 then
        V[istar] ← 0
    if D[jstar] = 0 then
        V[m + jstar] ← 0
    cont ← 0
    for i ← 1 to m do
        if V[i] > 0 then
            cont ← 1
            break
f ← transportation cost for X according to (1)
return X and f
```

Example 3.1. We apply the Decoding function to the example from Balinski given in Table 1. In this table, the last column shows the capacities of the suppliers, and the bottom row shows the capacities of the distributors. The first of the numbers given at the intersection of the row s_i and the column d_j shows c_{ij}, and the second shows a_{ij}.

	d_1	d_2	d_3	
s_1	2;10	3;30	4;20	10
s_2	3;10	2;30	1;20	30
s_3	1;10	4;30	3;20	40
s_4	4;10	5;30	2;20	20
	20	50	30	

Tab. 1: Input parameters of the Balinski example.

V	s	d	$W_{i^*,j}$ or W_{i,j^*}	i^*	j^*	x_{i^*,j^*}
$7,3,2,5 \| 1,4,6$	$10,30,40,20$	$20,50,30$	$3,6,6$	1	1	10
$0,3,2,5 \| 1,4,6$	$0,30,40,20$	$10,50,30$	$-,1.6,3.6,3$	2	3	30
$0,0,2,5 \| 1,4,0$	$0,0,40,20$	$10,50,0$	$5,6.5,-$	4	1	10
$0,0,2,5 \| 0,4,0$	$0,0,40,10$	$0,50,0$	$-,8,-$	4	2	10
$0,0,2,0 \| 0,4,0$	$0,0,40,0$	$0,40,0$	$-,3.75,-,-$	3	2	40

Tab. 2: Step-by-step application of the Decoding function.

For this problem and the chromosome $V = [7,3,2,5,1,4,6]$, the Decoding function finds the following initial solution according to the operations performed in Table 2:

$$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 0 & 30 \\ 0 & 40 & 0 \\ 10 & 10 & 0 \end{bmatrix}.$$

The transportation cost for this solution is 400.

3.2. Opex function

The Opex function takes 2 chromosomes V_1 and V_2 as input and produces 2 new chromosomes u_1 and u_2 from them. The pseudocode of this function is given in Algorithm 2:

Algorithm 2: Opex(V_1, V_2, m, n).

```
1  u1[1...m+n] ← 0
2  u2[1...m+n] ← 0
3  p ← random[1, m+n]
4  u1[1...p] ← V1[1...p]
5  u2[1...p] ← V2[1...p]
6  t1 ← Sort(V1[p+1...m+n])
7  t2 ← Sort(V2[p+1...m+n])
8  k ← m+n−p
9  for i ← 1 to k do
10     for j ← 1 to k do
11         if t1[i] = V1[p+j] then
12             q1[i] ← j
13             break
14     for j ← 1 to k do
15         if t2[i] = V2[p+j] then
16             q2[i] ← j
17             break
18 for i ← 1 to k do
19     u1[p+i] ← t1[q2[i]]
20     u2[p+i] ← t2[q1[i]]
21 return u1 and u2
```

The Opex function performs the following operations: First, the number p, a cutoff point, is selected randomly. The first p coordinates of the given chromosomes V_1 and V_2 are transferred to the chromosomes u_1 and u_2 as it is. Then, all the coordinates beginning from $(p+1)^{th}$ of the chromosomes V_1 and V_2 are sorted in ascending order and assigned to the t_1 and t_2, respectively. In addition, the indices of all coordinates before they are sorted (counting $(p+1)^{th}$ as the first coordinate) are assigned to the q_1 and q_2 arrays, respectively. Finally, the values of t_1 in q_2 are written to the coordinates of u_1 starting from $(p+1)^{th}$ coordinate, and the values of t_2 in q_1 are written to the coordinates of u_2 starting from $(p+1)^{th}$ coordinate.

Example 3.2. Let the chromosomes V_1 and V_2 are given as follows: V_1=$[5, 6, 7, 4, 3, 2, 1]$ and V_2=$[4, 3, 6, 5, 1, 7, 2]$. Assume that p is equal to 2. Then after steps 3-4 of the Algorithm 2 we have $u_1 = [5, 6, 0, 0, 0, 0, 0]$ and $u_2 = [4, 3, 0, 0, 0, 0, 0]$. After the steps 5-6 the sorted arrays are $t_1 = [1, 2, 3, 4, 7]$ and $t_2 = [1, 2, 5, 6, 7]$. In steps 7-20, the algorithm created the corresponding arrays q_1 and q_2 as follows: $q_1 = [5, 4, 3, 2, 1]$ and $q_2 = [3, 5, 2, 1, 4]$. Therefore, we get the new chromosomes $u_1 = [5, 6, 3, 7, 2, 1, 4]$ and $u_2 = [4, 3, 7, 6, 5, 2, 1]$.

3.3. Pex function

The Pex function takes 2 chromosomes such as V_1 and V_2 as input and produces 2 new chromosomes u_1 and u_2 from them. The problem's size that is the number of suppliers and distributors is the input data for this function. The pseudocode of the function is given in Algorithm 3:

Example 3.3. Let $V_1 = [5, 6, 3, 7, 2, 1, 4]$, $V_2 = [4, 3, 7, 6, 5, 2, 1]$ and $m = 4$, $n = 3$. Initially, $u_1 = [0, 0, 0, 0, 0, 0, 0]$ and $u_2 = [0, 0, 0, 0, 0, 0, 0]$. In the first 2 for loops, it is checked whether the values after the m^{th} coordinate of V_1 exist after the m^{th} coordinate of V_2. In this example, the value 2, which is the fifth coordinate of V_1, is the sixth coordinate of V_2, so the value 2 is written to the sixth coordinate of u_1 and the fifth coordinate of u_2. Similarly, the value 1 is written at the seventh coordinate of u_1 and the sixth coordinate of u_2. After these 2 for loops it becomes $u_1 = [0, 0, 0, 0, 0, 2, 1]$ and $u_2 = [0, 0, 0, 0, 2, 1, 0]$. Apart from that, the appropriate indices of V_1 and V_2 are reset, that is $V_1 = [5, 6, 3, 7, 0, 0, 4]$ and $V_2 = [4, 3, 7, 6, 5, 0, 0]$.

In the next 2 for loops, starting from the $(m+1)^{th}$ coordinate of V_1 and V_2, looking at the values that are both non-zero, these values are written into the arrays q_1 and q_2, and the appropriate values of V_1 and V_2 are reset. In this example, since V_1 for $i = 5, 6$ and V_2 for $j = 6, 7$ are zero, only $i = 7$ and $j = 5$ will both be non-zero. So $k = 1$, $q_1 = 4$, $q_2 = 5$ and also $V_1[7] = 0, V_2[5] = 0$.

In the next 2 loops, the values of the arrays q_1 and q_2 are searched in the first m coordinates of V_2 and V_1, respectively, and written to u_1 and u_2 in their index. In this example, the number 4 is in the first place in V_2, so $u_1[1] = 4$. The number 5 is in the 1st place in V_1, so $u_2[1] = 5$. After this step it becomes $u_1 = [4, 0, 0, 0, 0, 2, 1]$ and $u_2 = [5, 0, 0, 0, 2, 1, 0]$. In the next stage, zeros after m^{th} coordinate of u_1 and u_2 are replaced with the values q_2 and q_1, respectively. After this step; $u_1 = [4, 0, 0, 0, 5, 2, 1]$ and $u_2 = [5, 0, 0, 0, 2, 1, 4]$. Finally, the corresponding values of V_1 and V_2 are written as they are into places of u_1 and u_2 equal to 0, respectively. In this example, we obtain $u_1 = [4, 6, 3, 7, 5, 2, 1]$ and $u_2 = [5, 3, 7, 6, 2, 1, 4]$.

3.4. OpexMutation function

The OpexMutation function takes a chromosome V and produces a chromosome u from it. The dimensions of the problem are also inputs to this function. The pseudocode of this function is given in Algorithm 4.

Example 3.4. Let $V = [5, 3, 7, 6, 2, 1, 4]$, $m = 4$ and $n = 3$. Also let the random numbers be $p = 3$, $k_1 = 2$ and $k_2 = 5$. Initially it becomes $u = V = [5, 3, 7, 6, 2, 1, 4]$. Starting from the $k_1 = 2$nd coordinate, we sort the $p = 3$ elements of V and write them to t_1. Similarly, starting from the $k_2 = 5$th coordinate, we sort the $p = 3$ elements of V and write them to t_2. In result, we get $t_1 = [3, 6, 7]$ and $t_2 = [1, 2, 4]$. The indices of these elements in V (appropriately after k_1 and k_2) are kept in the q_1 and q_2 arrays, respectively. Accordingly, $q_1 = [1, 3, 2]$ and $q_2 = [2, 1, 3]$. In the last loop of the pseudocode, the values of t_1 in q_2 are written to the elements of u after $k_1 - 1$, and the values of t_2 in q_1 are written to the elements after $k_2 - 1$, respectively. In result, it becomes $u = [5, 6, 3, 7, 1, 4, 2]$.

Algorithm 3: Pex(V_1, V_2, m, n).

```
u1[1 ... m+n] ← 0; u2[1 ... m+n] ← 0; k ← 0
for i ← m+1 to m+n do
    for j ← m+1 to m+n do
        if V1[i] = V2[j] then
            u1[j] ← V2[j]; u2[i] ← V1[i]
            V1[i] ← 0; V2[j] ← 0
            break
for i ← m+1 to m+n do
    for j ← m+1 to m+n do
        if V1[i] * V2[j] > 0 then
            k ← k+1
            q1[k] ← V1[i]; q2[k] ← V2[j]
            V1[i] ← 0; V2[j] ← 0
            break
for i ← 1 to k do
    for j ← 1 to m do
        if V1[j] = q2[i] then
            u1[j] ← q1[i]
        if V2[j] = q1[i] then
            u2[j] ← q2[i]
k ← 0; l ← 0
for i ← m+1 to m+n do
    if u1[i] = 0 then
        k ← k+1; u1[i] ← q2[k]
    if u2[i] = 0 then
        l ← l+1; u2[i] ← q1[l]
for i ← 1 to m do
    if u1[i] = 0 then
        u1[i] ← V1[i]
    if u2[i] = 0 then
        u2[i] ← V2[i]
return u1 and u2
```

3.5. Main function

The results and processing time of the PbGA algorithm proposed by Lotfi and Tavakkoli–Mogaddam depend on the order in which the crossover and mutation operations are performed on the chromosomes and the selection strategy of new and old chromosomes.

Algorithm 4: OpexMutation(V, m, n).

```
p ← random[1, min{m, n}]
k1 ← random[1, m − p + 1]
k2 ← m + random[1, n − p + 1]
u ← V
t1 ← Sort(V[k1, ..., k1 + p − 1])
t2 ← Sort(V[k2, ..., k2 + p − 1])
for i ← 1 to p do
    for j ← 1 to p do
        if t1[i] = V[k1 + j − 1] then
            q1[i] ← j
            break
    for j ← 1 to p do
        if t2[i] = V[k2 + j − 1] then
            q2[i] ← j
            break
for i ← 1 to p do
    u[k1 + i − 1] ← t1[q2[i]]
    u[k2 + i − 1] ← t2[q1[i]]
return u
```

The main function written by the authors in the studies is given below in Algorithm 5.

In the Algorithm 5, $P(t)$ and $PN(t)$ represent a *pop_size* number of old and new chromosomes at step t. The authors used the roulette wheel and elitist strategy as selection methods in the algorithm. We direct readers to the study [12] on these methods.

Algorithm 5: Main$(m, n, A, C, S, D, \mu, \lambda, max_gen, pop_size)$.

```
t ← 1
initialize P(t) by the Decoding function
evaluate P(t)
while t ≤ max_gen do
    PN(t) ← crossover P(t) by Opex and Pex and mutate by OpexMutation
      functions
    evaluate PN(t)
    select P(t + 1) from P(t) and PN(t) by the μ + λ, roulette wheel selection
      and elitist strategy
    t ← t + 1
```

As mentioned above, when genetic algorithm-style algorithms are applied to a problem, the results obtained depend on how new chromosomes are produced and the order in which this production is carried out. It is also very important which old and new chromosomes are accepted as the chromosomes of the next iteration.

Unfortunately, in this algorithm, it is not clear which order of operations on the chromosomes is given priority and which selection strategy is given priority.

This function has been tried in many versions; the pseudocode of the version with the best results is given in Algorithm 6.

In this algorithm, $V[p]$ indicates the chromosome in the pth population; that is, for each p, $V[p]$ is a vector of size $m + n$.

Similarly, $X[p]$ indicates the feasible solution corresponding to the chromosome $V[p]$ in the pth population, that is, for each p, $X[p]$ is a matrix of size $m \times n$. $F[p]$ shows the transportation cost of the solution $X[p]$.

The function rW used in the pseudocode indicates the roulette wheel selection method adapted for minimization problems.

The input of this function is the value of the function F to be minimized and the number pop_size that represents the population number, and the output is the index of a value taken by the function F.

The function rW works as follows: The probabilities of each value taken by the function F are calculated and written to the roulette wheel.

Then, this roulette wheel is randomly rotated, and the index of the value F that provides the probability that this probability is obtained is returned. Note that since we are looking at the minimization problem, if $F[i] < F[j]$, the probability of $F[i]$ will be greater than the probability of $F[j]$.

The main idea of Algorithm 6 is:

First, we generate a pop_size number of initial chromosomes and corresponding initial solutions.

We calculate the transportation cost for each initial solution.

Then, we find 2 new chromosomes (2 children) by applying the Opex function to the 2 old chromosomes (parents) selected with the roulette wheel.

We generate the solutions corresponding to the children's chromosomes and calculate transportation costs.

If the child's transportation cost is lower than the parent's, we write the child instead of the parent.

Then we do the same operations for the Pex and OpexMutation functions and repeat all these operations for the number pop_size.

Algorithm 6: PbGA$(m, n, A, C, S, D, max_gen, pop_size)$.

```
for p ← 1 to pop_size do
    Produce chromosome V[p]
    (X[p], F[p]) ← Decoding(V[p])
for g ← 1 to max_gen do
    for p ← 1 to pop_size do
        irw1 ← rW(F, pop_size)
        irw2 ← irw1
        while irw2 = irw1 do
            irw2 ← rW(F, pop_size)
        for k ← 1 to 3 do
            V1 ← V[irw1]
            V2 ← V[irw2]
            if k = 1 then
                (u1, u2) ← Opex(V1, V2)
            else
                if k = 2 then
                    (u1, u2) ← Pex(V1, V2)
                else
                    u1 ← Opex_Mutation(V1)
                    u2 ← Opex_Mutation(V2)
                (X1, f1) ← Decoding(u1)
                (X2, f2) ← Decoding(u2)
                if F[irw1] > f1 then
                    F[irw1] ← f1; X[irw1] ← X1; V[irw1] ← u1
                if F[irw2] > f2 then
                    F[irw2] ← f2; X[irw2] ← X2; V[irw2] ← u2
imin ← 1
for p ← 1 to pop_size do
    if F[imin] > F[p] then
        imin ← p
return F[imin] and X[imin]
```

4. KARTLI ET AL.'S HEURISTIC ALGORITHM

Recently, Kartli et al. [25] have proposed a new heuristic algorithm for the optimal solution of FCTP. At the heart of this algorithm is the Minimization function. This function does the following:

The authors consider all possible 2×2 submatrices that have a positive product of the elements of at least one diagonal of the feasible solution and calculate the carrying cost

for this submatrix. Then they subtract the minimum element of the positive diagonal from the elements of this diagonal and add to the elements of the other diagonal, thus yielding another 2×2 dimensional submatrix. If the transportation cost for the new submatrix is less than the cost for the old submatrix, then they keep the new 2×2 submatrix in memory. After looking at all possible 2×2 submatrices, they write the 2×2 matrix with the minimum transportation cost in place of the corresponding 2×2 submatrix of the feasible solution. As a result of these processes, another feasible solution with less transportation cost is found and the same processes are repeated for this feasible solution. The Minimization function stops working when there is no reduction for any of the 2×2 submatrices of the considered feasible solution.

The pseudocode of the Minimization function is given in Algorithm 6. The function $FQ(i1, i2, j1, j2, X)$ in this algorithm calculates the objective function value for the appropriate 2×2 submatrix of the matrix X.

Kartli et al.'s algorithm also has two parameters as in the genetic algorithm, one of these parameters is pop_size, which is the number of initial feasible solutions, and the second is max_nd, which indicates the maximum number of non-decreases in cost. Initially, a pop_size number of initial feasible solutions are generated.

Any of the existing initial solution algorithms for FCTP or even classical TP can be used to create initial solutions. Note that since all the constraints of classical TP and FCTP are the same, a feasible solution to one of these two problems will also be a feasible solution to the other.

The Minimization function is run for each initial solution. The number of solutions for which the Minimization function cannot reduce the value of the optimization function is kept in a counter variable, and the solution with the least transportation cost is found. If the value of the counter variable does not reach the max_nd, the pop_size number initial solutions are generated again and the same operations are repeated. When the value of the counter variable reaches max_nd, the algorithm stops working and writes the solution with the least transportation cost.

The pseudocode of the Kartli et al.'s algorithm is given in Algorithm 8. In this algorithm, the variables $fprebest$ and $fbest$ represent the smallest transportation costs obtained in previous and current iterations, respectively. Similarly, variables $Xprebest$ and $Xbest$ denote the best solution obtained in previous and current iterations, respectively.

Algorithm 7: Minimization(X, f, C, A, m, n).

```
df ← −1
while df < 0 do
    df ← 0; k ← 0
    for i ← 1 to m do
        for j ← 1 to n do
            if X[i, j] > 0 then
                Irow[k] ← i; Jcol[k] ← j
                k ← k + 1
    for i ← 1 to k do
        i1 ← Irow[i]; j1 ← Jcol[i]
        for j ← 1 to k do
            i2 ← Irow[j]; j2 ← Jcol[j]
            if (i1 ≠ i2)&(j1 ≠ j2) then
                min ← X[i1, j1]
                if min > X[i2, j2] then
                    min ← X[i2, j2]
                SQ[1, 1] ← X[i1, j1] − min; SQ[1, 2] ← X[i1, j2] + min
                SQ[2, 1] ← X[i2, j1] + min; SQ[2, 2] ← X[i2, j2] − min
                f1 ← FQ[i1, i2, j1, j2, X]; f2 ← FQ[1, 2, 1, 2, SQ]
                df1 ← f2 − f1
                if df1 < df then
                    df ← df1; f3 ← f + df
                    i1m ← i1; i2m ← i2
                    j1m ← j1; j2m ← j2
                    SQM ← SQ
    if df < 0 then
        X[i1m, j1m] ← SQM[1, 1]; X[i1m, j2m] ← SQM[1, 2]
        X[i2m, j1m] ← SQM[2, 1]; X[i2m, j2m] ← SQM[2, 2]
        f ← f3
return f and X
Function FQ(i1, i2, j1, j2, X)
f ← C[i1, j1] ∗ X[i1, j1] + C[i1, j2] ∗ X[i1, j2]
f ← f + C[i2, j1] ∗ X[i2, j1] + C[i2, j2] ∗ X[i2, j2]
if X[i1, j1] > 0 then
    f ← f + A[i1, j1]
if X[i1, j2] > 0 then
    f ← f + A[i1, j2]
if X[i2, j1] > 0 then
    f ← f + A[i2, j1]
if X[i2, j2] > 0 then
    f ← f + A[i2, j2]
return f
```

Algorithm 8: Kartli et al. [25].

```
count ← 0; fprebest ← ∞
while count < max_nd do
    fbest ← ∞
    for poppar ← 1 to pop_size do
        V ← random.permutation[1, m + n]
        X, f1 ← Decoding(V, C, A, S, D, m, n)
        f, X ← Minimization(X, f1, C, A, m, n)
        if fbest > f then
            fbest ← f
            Xbest ← X
    if fprebest > fbest then
        fprebest ← fbest
        Xprebest ← Xbest
    count ← count + 1
return fprebest, Xprebest
```

5. PROPOSED HYBRID ALGORITHMS

This section proposes 3 new hybrid algorithms: H1, H2, and H3.

5.1. Hybrid algorithm H1

The algorithm H1 solves FCTP as follows: First, a certain number of feasible solutions are found with the PbGA algorithm, and then these solutions are sent to the Minimization function of the Kartli et al.'s algorithm as the initial feasible solutions. Among the obtained solutions, the solution that gives the smallest value to the objective function is accepted as the nearly optimal solution. The running time of the algorithm H1 with the same parameters will be slightly greater than the PbGA algorithm. If we compare Kartli et al.'s algorithm with the PbGA algorithm, we can see that both algorithms have advantages over each other. Like every metaheuristic algorithm, the PbGA algorithm does not perform a direct minimization process; all of these algorithms select the best one among a large number of feasible solutions. Kartli et al.'s algorithm performs a minimization process for each feasible solution, which makes it more advantageous than the PbGA algorithm. On the other hand, like every metaheuristic algorithm, PbGA also obtains new feasible solutions from the initially randomly selected feasible solutions more intelligently than Kartli et al.'s algorithm, which is a heuristic algorithm. In the algorithm H1, the Minimization function, which stands at the heart of Kartli et al.'s algorithm, was applied where the PbGA algorithm stopped, and better solutions were found.

The pseudocode of the algorithm H1 is given in Algorithm 9. This pseudocode assumes that the global variables X and F keep the feasible solutions and transportation costs found in the PbGA algorithm for all populations. Here, for each number p satis-

fying the condition $1 \leq p \leq pop\_size$, the variable $X[p]$ is a matrix of size $m \times n$ and represents a feasible solution to the problem.

Algorithm 9: H1

```
fbest, Xbest ← PbGA(m, n, A, C, S, D, max_gen, pop_size)
for p ← 1 to pop_size do
    f1 ← F[p]
    X1 ← X[p]
    f, X1 ← Minimization(X1, f1, C, A, m, n)
    if f < fbest then
        f ← fbest
        Xbest ← X1
return f and Xbest
```

5.2. Hybrid algorithm H2

To design the algorithm H2, we modify the main function of the PbGA algorithm as follows: Sort the feasible solutions at each step based on the value they give to the objective function in ascending order, eliminate the same solutions, and keep the *pop_size* number of feasible solutions. When the problem size is small, we can sort with insertion sort or bidirectional insertion sort algorithms [31], otherwise, with one of the optimal sorting algorithms [5]. By applying Opex, Pex and OpexMutation functions to these solutions circularly (1st to 2nd, 2nd to 3rd,...,*pop_size* $-$ 1 to *pop_size* we get new feasible solutions. Then, keep the best *pop_size* number of these solutions in memory and continue these operations until the stopping criterion is met. In a metaheuristic algorithm, only keeping the best solutions in memory at each step may cause us to get stuck at local extrema. Therefore, it is difficult to expect that our modified PbGA algorithm will give better results than Lotfi and Tavakkoli–Moghaddam's PbGA algorithm. However, our aim in the algorithm H2 is to obtain feasible solutions to send to the Minimization function of Kartli et al.'s algorithm. Therefore, it is not imaginary to assume that there are problems for which the algorithm H2 will achieve better results than the algorithm H1.

The algorithm H2 solves FCTP as follows: First, a certain number of feasible solutions are found with the Modified PbGA algorithm, and then these solutions are sent to the Minimization function of Kartli et al.'s algorithm as initial solutions. Among the solutions obtained, the solution that gives the smallest value to the objective function is accepted as the nearly optimal solution. The running time of the algorithm H2 for the same parameters will naturally, be slightly greater than the PbGA algorithm. The pseudocode of the algorithm H2 is given in Algorithm 10. In this pseudocode, it was assumed that all the first chromosomes produced were different from each other.

Algorithm 10: H2

```
fbest, Xbest ← MPbGA(m, n, A, C, S, D, max_gen, pop_size)
for p ← 1 to pop_size do
    f1 ← F[p]; X1 ← X[p]
    f, X1 ← Minimization(X1, f1, C, A, m, n)
    if f < fbest then
        f ← fbest; Xbest ← X1
return f and Xbest
Function MPbGA(m, n, A, C, S, D, max_gen, pop_size)
for p ← 1 to pop_size do
    Produce chromosome V[p]
    (X[p], F[p]) ← Decoding(V[p])
for g ← 1 to max_gen do
    F0 ← F; X0 ← X; V0 ← V
    F ← Sort(F)
    for p ← 1 to pop_size do
        for p0 ← 1 to pop_size do
            if F[p] = F0[p0] then
                break
        X[p] ← X0[p0]; V[p] ← V0[p0]
    V[pop_size + 1] ← V[1]
    for p ← 1 to pop_size − 1 do
        for k ← 1 to 3 do
            V1 ← V[p]; V2 ← V[p + 1]
            if k = 1 then
                (u1, u2) ← Opex(V1, V2)
            else
                if k = 2 then
                    (u1, u2) ← Pex(V1, V2)
                else
                    u1 ← Opex_Mutation(V1)
                    u2 ← Opex_Mutation(V2)
                (X1, f1) ← Decoding(u1)
                (X2, f2) ← Decoding(u2)
                if F[p] > f1 then
                    F[p] ← f1; X[p] ← X1; V[p] ← u1
                if F[p + 1] > f2 then
                    F[p + 1] ← f2; X[p + 1] ← X2; V[p + 1] ← u2
imin ← 1
for p ← 1 to pop_size do
    if F[imin] > F[p] then
        imin ← p
return F[imin] and X[imin]
```

5.3. Hybrid algorithm H3

The algorithm H3 is completely hybrid. It combines the iterative minimization feature of the heuristic Kartli et al.'s algorithm with the metaheuristic PbGA algorithm's feature of obtaining new feasible solutions from existing feasible solutions by crossover and mutation in a smarter. This algorithm uses the Decoding, Opex, Pex, OpexMutation functions of the PbGA algorithm and the Minimization functions of the Kartli et al.'s algorithm. First, we find *pop_size* number of feasible solutions with the help of the Decoding function. Then, we send these solutions to the Minimization function to assign the best value obtained to the variable $fbest$ and the best solution that achieves this value to the variable $Xbest$. We also list all the feasible solutions in increasing order according to their transportation cost values. Then, we apply the Opex, Pex, and OpexMutation functions to the chromosomes corresponding to all feasible solutions in a loop, respectively, with a circular rule, until the stopping condition is met. We send the new chromosome resulting from each application to the Decoding function, find a new feasible solution, and add this solution according to the transportation cost value, if this value differs from the existing ones, without breaking the order among the feasible solutions. We also send this new feasible solution to the Minimization function and update the values of the variables $fbest$ and $Xbest$ according to the result obtained. We keep the best solutions in memory as many as *pop_size* and perform the same operations until the stopping criterion is met. The pseudo-code of the algorithm H3 is given in Algorithm 11. In this algorithm, as in the algorithms H1 and H2, *pop_size* number chromosomes are marked with the array V. That is, for each number p, the variable $V[p]$ is an $m+n$ dimensional vector, and this vector is a permutation array of the $1, 2, \ldots, m+n$. In the algorithm, the variable $X[p]$ is a feasible solution generated by the Decoding function of the chromosome $V[p]$, and $F[p]$ is the transportation cost corresponding to this solution.

In steps 1 – 8 of the pseudocode, *pop_size* random chromosomes are generated, solutions corresponding to these chromosomes are found, and the transportation costs of these solutions are calculated.

In steps 9 – 15, the values of the objective function, the appropriate solution, and the appropriate chromosome are copied to the variables $F0$, $X0$, and $V0$, respectively, and the array F is sorted. Moreover, the arrays X and V are also sorted by finding the indices of the numbers in the array F in the array $F0$.

In steps 16 – 35, the following is done: Pex, Opex, OpexMutation functions are circularly applied to the chromosomes. After each application, each new chromosome obtained is sent to the Decoding function to find a new feasible solution and its transportation cost. If the transportation cost of the new feasible solution is different from the previous ones, this solution is added to the feasible solutions list. In addition, the new feasible solution is sent to the Minimization function, and the values of the variables $fbest$ and $Xbest$ are updated according to the result obtained.

In steps 37 – 42 of the pseudo-code, the Insert function is described, which adds the new feasible solution, its transportation cost, and the chromosome that produces it to the appropriate place in the sorted list.

Algorithm 11: H3

```
1  fbest ← ∞
2  for p ← 1 to pop_size do
3      Produce chromosome V[p]
4      (X[p], F[p]) ← Decoding(V[p])
5      f1 ← F[p]; X1 ← X[p]
6      f, X1 ← Minimization(X1, f1, C, A, m, n)
7      if fbest > f then
8          fbest ← f; Xbest ← X1
9  F0 ← F; X0 ← X; V0 ← V
10 F ← Sort(F)
11 for p ← 1 to pop_size do
12     for p0 ← 1 to pop_size do
13         if F[p] = F0[p0] then
14             break
15     X[p] ← X0[p0]; V[p] ← V0[p0]
16 for g ← 1 to max_gen do
17     V[pop_size + 1] ← V[1]
18     for p ← 1 to pop_size do
19         for k ← 1 to 3 do
20             V1 ← V[p]; V2 ← V[p + 1]
21             if k = 1 then
22                 (u1, u2) ← Opex(V1, V2)
23             else
24                 if k = 2 then
25                     (u1, u2) ← Pex(V1, V2)
26                 else
27                     u1 ← Opex_Mutation(V1); u2 ← Opex_Mutation(V2)
28                 (X1, f1) ← Decoding(u1); (X2, f2) ← Decoding(u2)
29                 Insert(X1, f1, F, X, V, u1); Insert(X2, f2, F, X, V, u2)
30                 f, X1 ← Minimization(X1, f1, C, A, m, n)
31                 if fbest > f then
32                     fbest ← f; Xbest ← X1
33                 f, X2 ← Minimization(X2, f2, C, A, m, n)
34                 if fbest > f then
35                     fbest ← f; Xbest ← X2
36 return fbest and Xbest
37 Function Insert(X1, f1, F, X, V, u)
38 k ← pop_size
39 while k > 0 and F[k] > f1 do
40     F[k + 1] ← F[k]; k ← k − 1
41 if k < pop_size and f1 > F[k] then
42     F[k + 1] ← f1; V[k + 1] ← u; X[k + 1] ← X1
```

6. EXPERIMENTAL RESULTS

The experiments are carried out on a computer with qualifications CPU i9 12th generation 16 core 32 GB RAM GPU 3090 RTX in a Python 3.0 environment. Lotfi and Tavakkoli–Moghaddam [28] produced 6 random problems with different dimensions; they applied their algorithm to these problems and compared the results obtained with the algorithm of Jo et al. [23]. The dimensions and inputs of these problems are given in Table 1.

Problem	Size	Suppliers	Distributors
1	4×5	$35, 27, 45, 37$	$33, 27, 22, 28, 34$
2	5×10	$45, 27, 65, 37, 30$	$23, 17, 13, 28, 20, 21, 21, 20, 26, 15$
3	10×10	$30, 17, 27, 29, 20, 18, 11, 15, 14, 23$	$23, 17, 13, 28, 20, 21, 21, 20, 26, 15$
4	10×20	$45, 27, 65, 37, 30, 38, 31, 35, 24, 24$	$16, 17, 12, 18, 20, 21, 21, 20, 16, 15,$ $20, 14, 13, 22, 17, 24, 15, 18, 17, 20$
5	20×30	$28, 27, 35, 26, 30, 27, 31, 22, 24, 24,$ $30, 36, 19, 32, 23, 33, 20, 16, 27, 28$	$23, 17, 12, 18, 20, 21, 21, 20, 16, 15,$ $20, 14, 13, 22, 17, 24, 15, 18, 24, 22$ $14, 13, 14, 12, 15, 20, 20, 18, 19, 21$
6	30×50	$45, 27, 65, 37, 30, 38, 31, 35, 24, 24,$ $30, 36, 29, 32, 23, 33, 30, 24, 27, 28,$ $34, 22, 27, 23, 31, 25, 23, 37, 39, 36$	$23, 17, 12, 18, 20, 21, 21, 20, 16, 15,$ $20, 14, 13, 22, 17, 24, 15, 18, 24, 22,$ $14, 13, 14, 12, 15, 20, 20, 18, 19, 21,$ $14, 23, 12, 20, 21, 24, 24, 23, 26, 28,$ $19, 17, 23, 21, 16, 24, 17, 23, 22, 20$

Tab. 3: Input data of the problems.

The matrices A and C determining the transportation cost of these problems are taken from the study [28].

Algorithms are applied to Problems 1-6 for 10 times. Upper-left sub-matrices of matrices C and A are taken based on the dimensions of each problem as costs. For example, 4×5 dimension matrices C and A are taken as below for Problem 1.

$$C = \begin{bmatrix} 25 & 14 & 34 & 46 & 45 \\ 10 & 47 & 14 & 20 & 41 \\ 22 & 42 & 38 & 21 & 46 \\ 36 & 20 & 41 & 38 & 44 \end{bmatrix}$$

$$A = \begin{bmatrix} 850 & 610 & 620 & 900 & 780 \\ 870 & 920 & 630 & 540 & 900 \\ 780 & 550 & 550 & 630 & 940 \\ 890 & 710 & 830 & 870 & 930 \end{bmatrix}.$$

Problem	*pop_size*	*max_gen/max_nd*	PbGA	Kartli et al.	H1	H2	H3
1	10	300/30	9291	**9168**	**9168**	**9168**	**9168**
	10	500/50	9222	**9168**	**9168**	**9168**	**9168**
2	10	300/30	**12718**	**12718**	**12718**	**12718**	**12718**
	10	500/50	**12718**	**12718**	**12718**	**12718**	**12718**
3	20	300/30	13987	**13923**	**13923**	**13923**	**13923**
	20	500/50	13934	**13923**	**13923**	**13923**	**13923**
	30	500/50	13934	**13923**	**13923**	**13923**	**13923**
4	20	500/50	22095	21926	21926	21926	21926
	20	700/70	22205	**21908**	21926	22013	21926
	30	700/70	22095	**21908**	**21908**	21926	**21908**
5	30	500/50	33466	32616	32568	32138	**31828**
	30	700/70	33075	32440	32460	32133	32124
	40	700/70	33214	32583	32368	32098	31868
6	30	500/50	55104	53860	53329	52844	52946
	40	700/70	54821	54072	53169	52612	52509
	40	1000/100	54677	54195	53414	52660	52828
	50	1000/100	55024	54138	53274	52791	**52136**

Tab. 4: Comparison of the best values obtained for objective function with algorithms applied (After 10 trials).

Comparative results of the algorithms are given in Table 4 – 7.

Table 4 compares the best results obtained by the algorithms. For Problem 1, the best value obtained by the PbGA algorithm was 9222, while all other algorithms reached the value of 9168. For Problem 2, the best result obtained by all algorithms was 12718. For Problem 3, the best value found by the PbGA algorithm was 13934, while all other algorithms found a value of 13923.

The best values found in Problem 4 were 22095 with PbGA, 21926 with H2, and 21908 with the other algorithms. In larger problems, Problem 5 and Problem 6, the algorithm H3 was superior. In Problem 5, PbGA found the value of 33075. In this problem, the best value was 32440 with the Kartli et al.'s algorithm, 32368 with the algorithm H1, and 32098 with the algorithm H2, while H3 reached the value of 31828. In Problem 6, the best values are as follows: PbGA algorithm 54677, Kartli et al.'s algorithm 53860, algorithm H1 53169, algorithm H2 52612, and algorithm H3 52136.

Table 5 describes the average values obtained when the algorithms were run 10 times for each problem. All other algorithms except PbGA obtained the best value in all 10 times in Problem 1 and Problem 2, so their average values were equal to their best values. While the best values of the PbGA algorithm in Problem 1 and Problem 2 are 9222 and 12718, respectively, the average values are 9285 and 12721, respectively. In Problem 3, the average value of the Kartli et al.'s algorithm and the algorithm H3 is equal to the best value, while the other algorithms, especially the PbGA algorithm, are far from the best value. While the algorithm that obtains the best average value in Problem 4 is the Kartli et al.'s algorithm the results of the algorithm H3 are better in larger Problems 5 and 6.

Problem	*pop_size*	*max_gen/max_nd*	PbGA	Kartli et al.	H1	H2	H3
1	10	300/30	9298	**9168**	**9168**	**9168**	**9168**
	10	500/50	9285	**9168**	**9168**	**9168**	**9168**
2	10	300/30	12721	**12718**	**12718**	**12718**	**12718**
	10	500/10	12741	**12718**	**12718**	**12718**	**12718**
3	20	300/30	14050	**13923**	13964	13951	**13923**
	20	500/50	14019	**13923**	13947	13947	**13923**
	30	500/50	14005	**13923**	13924	13929	**13923**
4	20	500/50	22287	21957	22121	22106	22089
	20	700/70	22392	21940	22101	22128	22096
	30	700/70	22288	**21940**	22136	22096	22011
5	30	500/50	33952	32940	32935	32648	32391
	30	700/70	33587	32820	32774	32570	32521
	40	700/70	33625	32942	32755	32573	**32321**
6	30	500/50	55849	54627	54119	53701	53383
	40	700/70	55670	54536	53690	53451	**53008**
	40	1000/100	55580	54584	53948	53183	53306
	50	1000//100	55536	54534	53875	53357	53108

Tab. 5: Comparison of the average values obtained for objective function with algorithms applied (After 10 trials).

Problem	*pop_size*	*max_gen/max_nd*	PbGA	Kartli et al.	H1	H2	H3
1	10	300/30	9304	**9168**	**9168**	**9168**	**9168**
	10	500/50	9304	**9168**	**9168**	**9168**	**9168**
2	10	300/30	12748	**12718**	**12718**	**12718**	**12718**
	10	500/50	12818	**12718**	**12718**	**12718**	**12718**
3	20	300/30	14195	**13923**	14195	14135	**13923**
	20	500/50	14070	**13923**	14019	14103	**13923**
	30	500/50	14065	**13923**	13934	13987	**13923**
4	20	500/50	22507	22124	22198	22198	22198
	20	700/70	22532	22035	22198	22198	22198
	30	700/70	22436	**22013**	22198	22198	22155
5	30	500/50	34294	33239	33309	33031	33024
	30	700/70	33902	33211	33061	33241	32924
	40	700/70	33909	33248	33041	32838	**32710**
6	30	500/50	56578	55024	54729	54679	53973
	40	700/70	56194	54898	54249	53857	**53786**
	40	1000/100	56294	54929	54587	54260	53992
	50	1000/100	56393	54822	54167	53858	53871

Tab. 6: Comparison of the worst values obtained for objective function with algorithms applied (After 10 trials).

Table 6 describes the worst-case results obtained when the algorithms are applied to Problems 1-6. In Problems 1 and 2, the results of all algorithms except PbGA are equal to their best results. In Problem 3, the results of Kartli et al.'s and H3 algorithms equal the best result. In Problem 4, the best of the worst values are obtained by Kartli et al.'s algorithm. In Problems 5 and 6, the clear superiority belongs to algorithm H3.

Problem	*pop_size*	*max_gen/max_nd*	PbGA	Kartli et al.	H1	H2	H3
1	10	300/30	1.03	0.08	1.04	0.88	0.90
	10	500/50	1.81	0.14	1.81	1.49	1.53
2	10	300/30	2.41	0.30	2.41	2.12	2.25
	10	500/50	3.67	0.52	3.67	3.33	3.43
3	20	300/30	7.04	1.31	7.05	6.36	6.33
	20	500/50	12.64	2.38	12.64	11.09	11.19
	30	500/50	18.20	3.32	18.20	14.88	15.60
4	20	500/50	24.50	8.77	24.50	21.08	22.78
	20	700/70	35.51	12.28	35.51	31.42	31.44
	30	700/70	52.52	17.73	52.52	45.88	46.86
5	30	500/50	101.43	52.84	101.44	85.68	86.05
	30	700/70	136.41	70.96	136.42	113.00	119.76
	40	700/70	184.21	92.49	184.22	150.89	155.96
6	30	500/50	232.22	188.73	232.77	203.06	229.41
	40	700/70	443.51	354.48	443.56	386.31	424.63
	40	1000/100	637.57	517.74	637.80	558.07	580.83
	50	1000/100	789.75	623.68	789.80	673.24	742.41

Tab. 7: Comparison of the running times of the algorithms applied (After 10 trials).

Table 7 discusses the running times of the algorithms. Experimental results show that Kartli et al.'s algorithm works faster than others. The processing times of algorithms H1, H2, H3 are comparable to the running time of PbGA, algorithm H1 reaches the result in slightly more time than PbGA, while algorithms H2, H3 reach the result in slightly less time.

7. CONCLUSION

In this study, we propose three hybrid algorithms H1, H2, and H3 to find the best solution to the fixed-charge transportation problem. Our aim in designing these algorithms is to combine the ability of metaheuristic algorithms to intelligently generate new solutions from existing solutions with the direct minimization feature of heuristic algorithms. In the algorithms H1 and H2, we did this as follows: First, we applied the PbGA (Modified PbGA) algorithm to solve the problem and sent the solutions obtained by these algorithms as input to the Minimization function of the Kartli et al. algorithm. In the algorithm H3, the advantages of metaheuristic and heuristic algorithms against each other are combined in the specific case of PbGA and Kartli et al.'s algorithms. Experimental results show that the algorithm H3 is superior to other algorithms.

In future studies, the algorithm H3 can be modified as follows: If the solution obtained after applying the Minimization function is different from the initial solution,

the chromosome corresponding to this solution can be found and added to the chromosome list. In addition, hybrid algorithms consisting of the combination of different metaheuristic and heuristic algorithms can be designed.

Recently, it has been observed that the teaching-learning based optimization (TLBO) algorithm [37], a metaheuristic algorithm, gives reasonable results for solving many problems [15]. In the study [27], it has been shown that when this algorithm is applied to FCTP, it produces solutions that are comparable to the PbGA algorithm in less running time. For example, a hybrid algorithm designed with the help of TLBO and Kartli et al.'s algorithm may give better results in a faster time. Therefore, studies can be done in this direction. Dragonfly optimization algorithm [30] is also one of the newly tried algorithms for FCTP [34]. The results of the hybrid algorithm that can be created with the help of this algorithm may also be a matter of curiosity. While this study is under review, an important study for 2-stage FCTP was published by Shivani et al. [41]. They present a feasibility restoration particle swarm optimizer with chaotic maps to solve 2-stage FCTP. In future studies, a hybrid algorithm of a heuristic approach can be designed with this proposed method.

(Received November 11, 2024)

REFERENCES

[1] V. Adlakha and K. Kowalski: On the fixed-charge transportation problem. Omega *27* (1999), 3, 381–388. DOI:10.1016/S0305-0483(98)00064-4

[2] V. Adlakha and K. Kowalski: A simple heuristic for solving small fixed-charge transportation problems. Omega *31* (2003), 3, 205–211. DOI:10.1016/S0305-0483(03)00025-2

[3] V. Adlakha, K. Kowalski, and R. R. Vemuganti: Heuristic algorithms for the fixed-charge transportation problem. Opsearch *43* (2006), 132–151. DOI:10.1007/BF03398770

[4] V. Adlakha, K. Kowalski, and B. Lev: A branching method for the fixed charge transportation problem. Omega *38* (2010), 5, 393–397. DOI:10.1016/j.omega.2009.10.005

[5] S. E. Amrahov, Y. Ar, B. Tugrul, B. E. Akay, and N. Kartli: A new approach to Mergesort algorithm: Divide smart and conquer. Future Generation Computer Systems *157* (2024), 330–343. DOI:10.1016/j.future.2024.03.049

[6] M. L. Balinski: Fixed-cost transportation problems. Naval Research Logistics Quarterly *8* (1961), 1, 41–54. DOI:10.1002/nav.3800080104

[7] A. Biswas, S. Roy, and S. P. Mondal: Evolutionary algorithm based approach for solving transportation problems in normal and pandemic scenario. Applied Soft Computing *129* (2022), 109576. DOI:10.1016/j.asoc.2022.109576

[8] H. I. Calvete, C Gale, J. A. Iranzo, and P. Toth: A matheuristic for the two-stage fixed-charge transportation problem. Computers Oper. Res. *95* (2018), 113–122. DOI:10.1016/j.cor.2018.03.007

[9] O. Cosma, P. C. Pop, and D. Dănciulescu: A novel matheuristic approach for a two-stage transportation problem with fixed costs associated to the routes. Computers Oper. Res. *118* (2020), 104906. DOI:10.1016/j.cor.2020.104906

[10] G. B. Dantzig: Linear programming. Oper. Res. *50* (2002), 1, 42–47. DOI:10.1287/opre.50.1.42.17798

[11] A. Ebrahimnejad: New method for solving fuzzy transportation problems with LR flat fuzzy numbers. Inform. Sci. *357* (2016), 108–124. DOI:10.1016/j.ins.2016.04.008

[12] A. E. Eiben and A. E. Smith: Introduction to Evolutionary Computing. Springer-Verlag, Berlin, Heidelberg 2015.

[13] M. M. El-Sherbiny and R. M. Alhamali: A hybrid particle swarm algorithm with artificial immune learning for solving the fixed charge transportation problem. Computers Industr. Engrg. *64* (2013), 2, 610–620. DOI:10.1016/j.cie.2012.12.001

[14] M. Hakim and R. Zitouni: An approach to solve a fuzzy bi-objective multi-index fixed charge transportation problem. Kybernetika *60* (2024), 3, 271–292. DOI:10.14736/kyb-2024-3-0271

[15] E. Hazrati Nejad, S. Yigit-Sert, and S. Emrah Amrahov: An effective global path planning algorithm with teaching-learning-based optimization. Kybernetika *60* (2024), 3, 293–316. DOI:10.14736/kyb-2024-3-0293

[16] W. M. Hirsch and G. B. Dantzig: The fixed charge problem. Naval Res. Logist. Quarterly *15* (1968), 3, 413–424. DOI:10.1002/nav.3800150306

[17] F. L. Hitchcock: Distribution of a product from several sources to numerous locations. J. Math. Physics *20* (1941), 224–230. DOI:10.1002/sapm1941201224

[18] J. Hong, A. Diabat, V. V. Panicker, Sand . Rajagopalan: A two-stage supply chain problem with fixed costs: An ant colony optimization approach. Int. J. Product. Econom. *204*, (2018), 214–226. DOI:10.1016/j.ijpe.2018.07.019

[19] A. Hosseini and M. S. Pishvaee: Capacity reliability under uncertainty in transportation networks: An optimization framework and stability assessment methodology. Fuzzy Optim. Decision Making *21* (2022), 3, 479–512. DOI:10.1007/s10700-021-09374-9

[20] J. Jansi Rani, A. Manivannan, and S. Dhanasekar: Interval valued intuitionistic fuzzy diagonal optimal algorithm to solve transportation problems. Int. J. Fuzzy Systems *25* (2023), 4, 1465–1479. DOI:10.1007/s40815-022-01446-1

[21] N. Jawahar and A. N. Balaji: A genetic algorithm for the two-stage supply chain distribution problem associated with a fixed charge. Eur. J. Oper. Res. *194* (2009), 2, 496–537. DOI:10.1016/j.ejor.2007.12.005

[22] N. Jawahar, A. Gunasekaran, and N. Balaji: A simulated annealing algorithm to the multi-period fixed charge distribution problem associated with backorder and inventory. Int. J. Prod. Res. *50* (2012), 9, 2533–2554. DOI:10.1080/00207543.2011.581013

[23] J. B. Jo, Y. Li, and M. Gen: Nonlinear fixed charge transportation problem by spanning tree-based genetic algorithm. Computers Industr. Engrg. *53* (2007), 2, 290–298. DOI:10.1016/j.cie.2007.06.022

[24] N. Kartlı, E. Bostancı, and M. S. Guzel: A new algorithm for the initial feasible solutions of fixed charge transportation problem. In: 2022 7th International Conference on Computer Science and Engineering (UBMK), IEEE 2022, pp. 82–85. DOI:10.1109/ubmk55850.2022.9919524

[25] N. Kartli, E. Bostanci, and M. S. Guzel: A new algorithm for optimal solution of fixed charge transportation problem. Kybernetika *59* (2023), 1, 45–63. DOI:10.15625/2615-9023/18488

[26] N. Kartli, E. Bostanci, and M. S. Guzel: Heuristic algorithm for an optimal solution of fully fuzzy transportation problem. Computing *106* (2024), 10, 3195–3227. DOI:10.1007/s00607-024-01319-5

[27] N. Kartli: A Metaheuristic Algorithm for the Fixed Charge Transportation Problem. In 2024 9th International Conference on Computer Science and Engineering (UBMK) IEEE (2024) 1030–1033. DOI:10.1109/UBMK63289.2024.10773580

[28] M. M. Lotfi and R. Tavakkoli–Moghaddam: A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems. Appl. Soft Comput. *13* (2013), 5, 2711–2726. DOI:10.1016/j.asoc.2012.11.016

[29] D. Mardanya and S. K. Roy: New approach to solve fuzzy multi-objective multi-item solid transportation problem. RAIRO Oper. Res. *57* (2023), 1, 99–120. DOI:10.1051/ro/2022211

[30] S. Mirjalili: Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. Neural Comput. Appl. *27* (2016) 1053–1073. DOI:10.1007/s00521-015-1920-1.

[31] A. S. Mohammed, S. E. Amrahov, and F. V. Celebi: Bidirectional conditional insertion sort algorithm; An efficient progress on the classical insertion sort. Future Generation Computer Systems *71* (2017), 102–112. DOI:10.1016/j.future.2017.01.034

[32] A. Mondal and S. K. Roy: Behavioural threeway decision making with Fermatean fuzzy Mahalanobis distance: Application to the supply chain management problems. Appl. Soft Computing *151* (2024), 111182. DOI:10.1016/j.asoc.2023.111182

[33] V. V. Panicker, R. Vanga, and R. Sridharan: Ant colony Optimization algorithm for distribution-allocation problem in a two-stage supply chain with a fixed transportation charge. Int. J. Prod. Res. *51* (2013), 3, 698–717. DOI:10.1080/00207543.2012.658118

[34] A. N. S. Paojiyah, A. P. Az'zahra, V. F. Aulia, and E. R. Wulan: Penerapan Dragonfly Optimization Algorithm (DOA) untuk Menyelesaikan Fixed Charge Transportation Problem (FCTP). KUBIK: Jurnal Publikasi Ilmiah Matematika *9* (2024), 2, 187–197.

[35] P. C. Pop, C. Sabo, B. Biesinger, B. Hu, and G. R. Raidl: Solving the two-stage fixed charge transportation problem with a hybrid genetic algorithm. Carpathian J. Math. *33* (2017), 3, 365–371.

[36] K. A. A. D. Raj and C. Rajendran: A genetic algorithm for solving the fixed-charge transportation model: two-stage problem. Comput. Oper. Res. *39* (2012), 9, 2016–2032. DOI:10.1016/j.cor.2011.09.020

[37] R. V. Rao, V. J. Savsani, and D. Vakharia: Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Computer-aided Design *43* (2011), 303–315. DOI:10.1016/j.cad.2010.12.015

[38] B. Saikia, P. Dutta, and P. Talukdar: An advanced similarity measure for Pythagorean fuzzy sets and its applications in transportation problem. Artif. Intell. Rev. *56* (2023), 11, 12689–12724. DOI:10.1007/s10462-023-10421-7

[39] S. Sandhiya and A. Dhanapal: Solving neutrosophic multi-dimensional fixed charge transportation problem. Contemp. Math. *5* (2024), 3, 3601–3624. DOI:10.37256/cm.5320244927

[40] G. Singh and A. Singh: Extension of particle swarm optimization algorithm for solving transportation problem in fuzzy environment. Appl. Soft Comput. *110* (2021), 107619. DOI:10.1016/j.asoc.2021.107619

[41] Shivani, D. Chauhan, and D. Rani: A feasibility restoration particle swarm optimizer with chaotic maps for two-stage fixed-charge transportation problems. Swarm Evolutionary Comput. *91* (2024), 101776. DOI:10.1016/j.swevo.2024.101776

[42] M. Sun, J. E. Aronson, P. G. Mckeown, and D. Drinka: Tabu search heuristic procedure for the fixed charge transportation problem. Eur. J. Oper. Res. *106* (1998), 2–3, 411–456.

Nermin Kartli, Computer Engineering Department, Ankara University, Ankara. Turkey.
e-mail: nermin.kartli@gmail.com