Probabilistic tree-based representation for solving minimum cost integer flow problems with nonlinear non-convex cost functions

Ghasemishabankareh, Behrooz; Li, Xiaodong; Ozlen, Melih; Neumann, Frank

Ghasemishabankareh, B., Li, X., Ozlen, M., & Neumann, F. (2020). Probabilistic tree-based representation for solving minimum cost integer flow problems with nonlinear non-convex cost functions. Applied Soft Computing Journal, 86, 1–14. https://doi.org/10.1016/j.asoc.2019.105951
Document Version: Accepted Manuscript

Published Version: https://doi.org/10.1016/j.asoc.2019.105951

Probabilistic Tree-Based Representation for Solving Minimum Cost Integer Flow Problems with Nonlinear Non-convex Cost Functions

Behrooz Ghasemishabankareh[a,*], Xiaodong Li[a], Melih Ozlen[a], Frank Neumann[b]

[a]*School of Science, RMIT University, Melbourne, Australia*
[b]*School of Computer Science, The University of Adelaide, Adelaide, Australia*

Abstract

The minimum cost flow problem (MCFP) is the most generic variation of the network flow problem which aims to transfer a commodity throughout the network to satisfy demands. The problem size (in terms of the number of nodes and arcs) and the shape of the cost function are the most critical factors when considering MCFPs. Existing mathematical programming techniques often assume the cost functions to be linear or convex. Unfortunately, the linearity and convexity assumptions are too restrictive for modelling many real-world scenarios. In addition, many real-world MCFPs are large-scale, with networks having a large number of nodes and arcs. In this paper, we propose a probabilistic tree-based genetic algorithm (PTbGA) for solving large-scale minimum cost integer flow problems with nonlinear non-convex cost functions. We first compare this probabilistic tree-based representation scheme with the priority-based representation scheme, which is the most commonly-used representation for solving MCFPs. We then compare the performance of PTbGA with that of the priority-based genetic algorithm (PrGA), and two state-of-the-art mathematical solvers on a set of MCFP instances. Our experimental results demonstrate the superiority and efficiency of PTbGA in dealing with large-sized MCFPs, as compared to the PrGA

*Corresponding author

Email addresses: behrooz.ghasemishabankareh@rmit.edu.au (Behrooz Ghasemishabankareh), xiaodong.li@rmit.edu.au (Xiaodong Li), melih.ozlen@rmit.edu.au (Melih Ozlen), frank.neumann@adelaide.edu.au (Frank Neumann)

method and the mathematical solvers.

Keywords:
Minimum cost flow problem, genetic algorithm, representation scheme, mixed integer nonlinear programming, Taguchi experimental design

1. Introduction

Network flow problems have many real-world applications such as electrical and power networks, telecommunications, road and rail networks, airline services networks [1]. The shortest path problem, the maximum flow problem, the assignment problem, the transportation problem, and the minimum cost flow problem (MCFP) are different variations of the network flow problem. Among these, MCFP is the most generic network flow problem with numerous applications such as distribution problems, scheduling, optimal loading of a hopping aeroplane, and racial balancing of schools [1].

The shape of cost function (linear, nonlinear, convex, concave and non-convex) on each arc, is one of the main factors contributing to the complexity of network flow problems and more specifically MCFPs [2]. The linear MCFP is polynomially solvable [3] by using a strong polynomial algorithm [4]. Two recent surveys for solving MCFPs using linear cost functions are presented in [5, 6].

However, in many real-world problems, linearity assumption as well as linear approximation of the nonlinear functions cannot capture the real-world situations adequately. In such cases, ideally the nonlinear cost functions should be directly employed in order to accurately model the real-world scenarios [7]. The non-linearity of cost function occurs due to *economies of scale*, and it happens when the cost per unit of flow decreases by increasing flow or when there is a fixed charge for sending a flow through a new arc in the network [2]. Many studies in the literature suggest the appropriateness of employing nonlinear cost functions in the network flow problems [8, 9] and the power flow and power systems problems [10, 11, 12].

MCFP considering a concave cost function and its special case called single-source uncapacitated (SSU) MCFP with fixed-charge costs are known to be NP-hard [13]. The complexity of the concave MCFP arises because during minimisation of a concave function over a convex feasible region (defined by the network constraints) a local optimum is not necessarily a global optimum [3]. There are several studies in the literature which solved SSU

MCFP with a concave cost function in order to find a global optimum or an approximate optimum which serves as a bound on the global optimum. A dynamic programming approach, branch-and-bound method, and a linear approximation of a concave cost function were introduced to solve SSU MCFP using concave cost functions [14, 15, 16]. However, the above-mentioned algorithms are not capable of handling large-scale networks, neither efficiently. For instance, Fontes et al. [14] solved network instances only up to 19 nodes in 5 hours, and Burkard et al. [16] considered networks up to 21 nodes, but it took 13 hours.

Metaheuristic methods are another popular approach for solving the nonlinear network flow problems and more specifically MCFPs. Among these, ant colony optimisation (ACO) and genetic algorithm (GA) are most commonly-used. For example, an ACO algorithm and a hybrid ACO with local search were proposed in [3, 17] for tackling the SSU MCFPs. Both methods were able to solve the uncapacitated network instances, with the largest network instances considered being networks with just 50 nodes.

Representation plays an important role in the success of an GA. In order to deal with network flow problems, several representation schemes have been proposed such as variable-length encoding [18], fixed-length encoding [19], priority-based representation (PbR) [20], improved priority-based representation [21], and random key representation [2]. Additionally, several studies have been conducted to solve the unit commitment problems in power systems using priority list representation [10, 22, 23]. Among these, the variable-length [24] and the fixed-length [25] encoding schemes are not equipped for solving MCFP since they produce infeasible solutions.

Random key representation and PbR are frequently used for combinatorial optimisation. For instance, two genetic algorithm methods using priority-based representation were proposed to solve transportation problems in [26] and [20]. A random key representation was applied for solving resource constrained project scheduling problem [27] as well as a multi-mode resource-constrained project scheduling problem [28]. In order to deal with MCFPs, Gen et al. [29] shows that PbR can be applied for solving small-sized bi-criteria MCFPs (i.e. network with 25 nodes and 56 arcs). Whereas Fontes et al. [2] show that the random key representation can be effectively used in a hybrid GA for solving MCFPs with concave cost functions. In this case, uncapacitated networks with medium-sized instances (up to 50 nodes) were considered.

The aforementioned studies [2, 27, 26, 20, 30] have focused on the unca-

pacitated MCFPs, however, it would be desirable to solve the capacitated MCFPs directly. Although the capacitated networks can be transformed into uncapacitated ones, the transformation procedure introduces a large number of new arcs and nodes. As a result, it significantly increases the number of decision variables especially when dealing with large-scale problems [1]. Very few attempts have been made to directly handle the capacitated MCFP, with an exception in [31], where a deterministic annealing algorithm was developed for solving the capacitated MCFPs with a dense network (up to 130 variables).

In this paper we propose a probabilistic tree-based genetic algorithm (PTbGA) for solving the large-scale MCFP using nonlinear non-convex cost functions, to tackle the following issues in MCFPs: 1) networks with a large number of arcs and integer flows; 2) various shapes of the cost function; 3) dealing with a capacitated network directly. The proposed PTbGA method allows us to consider the capacitated MCFPs directly without the need of transforming a capacitated network into an uncapacitated one. As a result, PTbGA's capability to solve the large-scale problem instances is much enhanced.

Building on our preliminary work [32], where only single-source and single-sink network instances were considered, this paper provides a more detailed account of the probabilistic tree-based representation (PTbR) scheme, which provides the key ingredients allowing us to search the entire feasible search space more effectively, as compared with the conventional priority-based representation (PbR) scheme. Furthermore, we introduce an improved decoding procedure that allows more thorough sampling of the search space. Finally, we conduct systematic and extensive experiments to compare the performance of the PTbR-based GA (PTbGA) variant with the PbR-based GA (PrGA) for solving multi-source and multi-sink MCFP instances (up to 48,000 arcs), using nonlinear non-convex cost functions. We use the Taguchi method to identify the optimal parameter settings for both PTbGA and PrGA. The performances of these two GA variants are compared with those of the-state-of-the-art mathematical solvers on a set of small, medium, and large-sized network instances. Our experimental results demonstrate the superiority of the PTbGA over the PrGA and the mathematical programming solvers, especially when handling the large-scale MCFPs, as well as its robustness when using a variety of nonlinear non-convex cost functions.

The remainder of the paper is organised as follows: the definition of the MCFP is provided in Section 2. Section 3 presents the proposed PTbR scheme, and compares it with the commonly-used PbR scheme. The pro-

posed PTbGA method is presented in Section 4. Section 5 describes the problem instances used, parameter settings, and experimental results. Section 6 provides the conclusion.

Nomenclature and abbreviation

Mathematical symbols

A	A set of arcs.
$b(i)$	Amount of supply and demand for node i.
c_{ij}	Non-negative coefficient in cost function.
f_{ij}	Cost function on arc (i,j).
G	Directed network.
l_{ij}	Lower bound of arc (i,j).
m	Number of arcs.
n	Number of nodes.
$\mathcal{N}$	A set of nodes.
x_{ij}	An integer flow from node i to node j.
u_{ij}	Capacity of arc (i,j).

Notations

$Flow_p$	Amount of flow can be sent on $Path_p$.
h	Indicator for the outcome of one-sample t-test.
$No.$	Instance number.
OBJ	Cost function value obtained by mathematical solvers.
p	Path counter.
std	Standard deviation of cost function over 30 runs.
t	Average running time in seconds.
t'	Running time for mathematical solvers.
U	Capacity of each Path.

Parameters

α	Percentage of population members which are initialised randomly.
It_{max}	Maximum number of iterations.
N	Number of decoding procedure for PTbGA.
P_c	Crossover rate.
P_m	Mutation rate.
pop_size	Number of individuals in each population for GA-based methods.

Abbreviations

MCFP	Minimum cost flow problem.
MINLP	Mixed integer nonlinear programming.
NFEs	Maximum number of function evaluations.
NLP	Nonlinear programming.
OR	Operations research.
PbR	Priority-based representation scheme.
PrGA	Priority-based genetic algorithm.
PTbGA	Probabilistic tree-based genetic.
PTbR	Probabilistic tree-based representation scheme.
RPD	Relative percentage deviation.
S/N	Signal-to-noise.
WMX	Weight mapping crossover.

2. Problem definition

Let $G = (\mathcal{N}, A)$ be a directed network with a set $\mathcal{N}$ which consists of n nodes and a set A of m arcs. Each arc (i, j) has a capacity of u_{ij} and lower bound of l_{ij} which denote the maximum and minimum amount that can be sent on the arc (i, j), respectively. Each node $i \in \mathcal{N}$ is associated with an integer value $b(i)$. If $b(i)$ is positive, it shows that node i is a supply node, if $b(i)$ is negative, node i is a demand node with demand of $\mid b(i) \mid$ and finally $b(i) = 0$ shows the transshipment node i. A decision variable in an MCFP is an *integer* flow and denoted by x_{ij}. The associated cost for each flow (x_{ij}) is denoted by $f_{ij}(x_{ij})$. Fig. 1 shows an example of the MCFP with n=5 nodes and m=7 arcs, which has a supplier node ($b(1) = 10$) and a demand node ($b(5) = -10$). Generally in MCFP, we aim to send flows throughout the network to satisfy all demands by minimising the total cost (i.e., the objective function value). The formulation of the MCFP is as follows [1]:

$$Minimise : z(\vec{\mathbf{x}}) = \sum_{(i,j)\in A} f_{ij}(x_{ij}), \tag{1}$$

$$\text{s.t.} \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = b(i) \quad \forall \quad i \in \mathcal{N}, \tag{2}$$

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad \forall \quad (i,j) \in A, \tag{3}$$

$$x_{ij} \in \mathbb{Z} \quad \forall \quad (i,j) \in A, \tag{4}$$

where Eq.1 minimises the total cost within the network. Eq. 2 is a flow balance constraint and states that the difference between the total outflow (first term) and the total inflow (second term) is equal to $b(i)$. Eq. 3 ensures that the flow on each arc is between upper and lower bound, and finally Eq. 4 ensures that all flow values are integer. In this study we consider the following assumptions for the MCFP: 1) the lower bounds on all arcs are set to zero; 2) the network is directed; 3) there are no two or more arcs with the same tail or head in the network; 4) the total demand and supply in the network are equal, i.e. $\sum_{i=1}^{n} b(i) = 0$; 5) there are more than one supplier and one demand node in the network; and 6) the cost function on each arc (f_{ij}) is a nonlinear non-convex function.

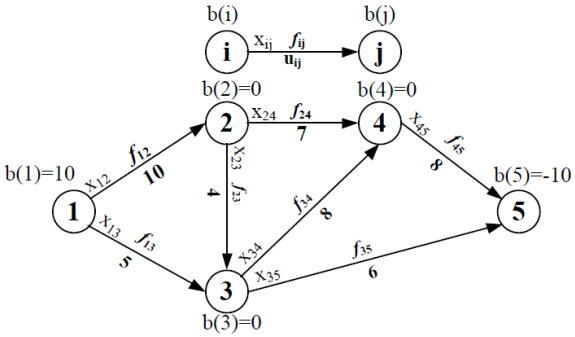


Figure 1: An example of the MCFP (n=5, m=7).

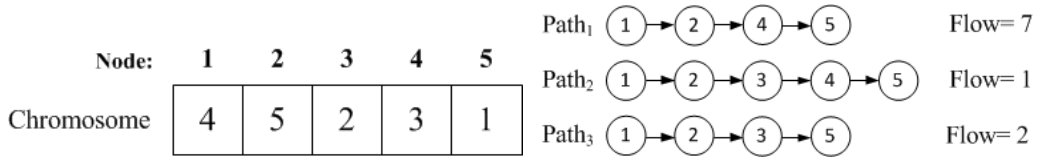
3. Representation schemes

A good representation scheme can often make an optimisation problem easier to solve. In the following, we first describe the priority-based representation (PbR) which is the most commonly-used representation method for handling MCFPs and discuss its drawbacks. Then we present the probabilistic tree based representation (PTbR) scheme for solving the MCFP and we demonstrate that this representation scheme can help the search in the right direction in feasible regions of the search space of the MCFP.

3.1. Priority-based representation (PbR)

A chromosome in the GA has two main characteristics: *locus*, the position of a gene in a chromosome; and *allele*, the possible values each gene can take. To represent a candidate solution for an MCFP, PbR lets the number of genes to be equal to n and the allele is generated randomly between 1 and n, which represents the priority of each node for constructing a path among all possible nodes [29]. Fig. 2 illustrates the PbR chromosome and its corresponding solution for the network presented in Fig. 1.

To obtain a feasible solution for a given PbR chromosome presented, a two-phase decoding procedure is followed. In phase I, a path is generated based on the priorities and the maximum possible flow is sent though the generated path in phase II. The decoding procedure starts from node 1. Based on the network presented in the Fig. 1, the possible successor nodes are nodes 2 and 3. Based on Fig. 2a, the priorities of nodes 2 and 3 are equal to 5 and 2, respectively. Since node 2 has a higher priority, node 2 is selected as the successor. Node 2's possible successors are nodes 3 and 4.



(a) An example of a chromosome for the network in Fig. 1.

(b) A feasible solution for the given chromosome.

Figure 2: The PbR chromosome and its corresponding solution.

Since the priority of node 4(3) is higher than node 3(2), node 4 is selected. Finally from node 4 we can only move to node 5. Hence, the following path is generated: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. In phase II, in order to send a feasible flow on the generated path, the capacity of the generated path is calculated: $U = \min\{u_{12} = 10, u_{24} = 7, u_{45} = 8\} = 7$. Consequently, the flow on the generated path is equal to 7. After sending the flow on the network, the upper bound (u_{ij}), supply and demand should be updated. Since the supply/demand is not equal to 0, the second path should be generated. The above procedure repeats until all demands are satisfied. Fig. 2b presents the feasible solution for the given chromosome in Fig. 2a after executing the decoding procedure.

Although PbR is commonly used in the network flow problems [29], but it has some limitations in representing the full extent of the feasible search space [32]. Fig. 3 shows an example for the network presented in Fig. 1 that PbR is unable to represent. Here the first path is generated as follows: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$. Since in Path_1 after node 1, node 2 is selected, it shows that node 2 has a higher priority than node 3. Hence, if arc (1,2) is not saturated, PbR will not allow any flow to be sent through arc (1,3), essentially blocking this possibility completely (see Path_2 in Fig. 3). This means that PbR is unable to represent a potential feasible solution such that the flow would go through arc (1,3) (as shown in Fig. 3). Another drawback for PbR is that each time a path is generated, we are supposed to send the maximum possible amount on the generated path. For instance, for the network in Fig. 1, if the first generated path is $\text{Path}_1 : 1 \rightarrow 2 \rightarrow 4 \rightarrow 5$, since $U = \min\{u_{12} = 10, u_{24} = 7, u_{45} = 8\} = 7$, we are not allowed to send a flow less than 7. These limitations would restrict a search algorithm from reaching the full extent of the feasible search space.

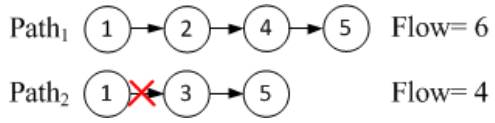


Figure 3: An example of the feasible solution that PbR fails to represent (for the network in Fig. 1).

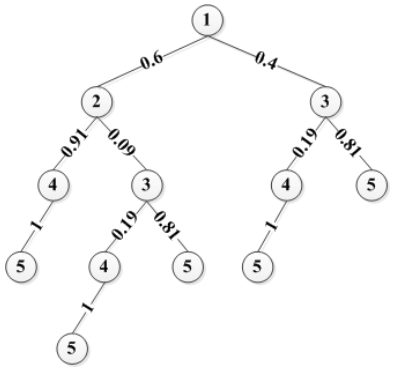


Figure 4: A tree structure for the network presented in Fig. 1

3.2. Probabilistic tree-based representation (PTbR)

To counteract the above limitations of the PbR, here we propose a probabilistic tree-based representation (PTbR) scheme, where a probability tree is adopted to represent a potential MCFP solution. Unlike the PbR scheme which is restricted to a small part of the feasible space, the PTbR is able to represent all possible feasible solutions. The PTbR can be represented as a tree structure where the probability of moving from one node to another is defined on the branches of the tree. Fig. 4 illustrates the probability tree structure for the network in Fig. 1.

The tree structure can be converted to a chromosome. Fig. 5 shows an example of a chromosome for the tree structure presented in Fig. 4. The chromosome has n-1 sub-chromosomes (SCh), m genes, and within each sub-chromosome the value of each gene (which represents the probability) is between 0 and 1 and cumulatively increases to one.

Based on Algorithm 1, in order to obtain a feasible solution from the PTbR (e.g. a chromosome), we follow two phases. In the first phase, a path is constructed and in the second phase, a feasible flow is sent through the generated path. This process is repeated until the demand is satisfied. The inputs for Algorithm 1 are chromosome, supply/demand nodes. To obtain a

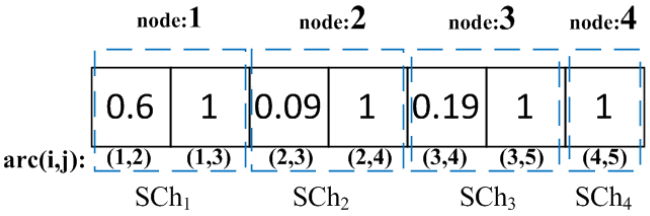


Figure 5: A chromosome for the tree structure in Fig. 4

feasible solution for the chromosome in Fig. 5, we start generating the first path from node i=1 ($\text{SCh}_{i=1}$). First, a random number is generated in [0,1] (e.g. $rand$=0.2, line 10). Based on the chromosome in Fig. 5, from node 1 (i.e. $\text{SCh}_{i=1}$) we have two outgoing arcs (1,2) and (1,3) which connect us to nodes 2 and 3 respectively. Since the generated random number is between 0 and 0.6 (i.e $0 \leq rand = 0.2 \leq 0.6$), we reach node 2 by moving through arc (1,2) and node 2 will be added to the generated path (line 11). Since we have not reached to the node n=5 (line 9) yet, from node 2 ($\text{SCh}_{i=2}$) another random number is generated ($0.09 \leq rand = 0.85 \leq 1$) and we move through arc (2,4) to reach to node 4. From node 4, the only available node is 5 (i.e. through arc (4,5)). Since we have already reached node n=5, the first phase is terminated and here is the generated path: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$.

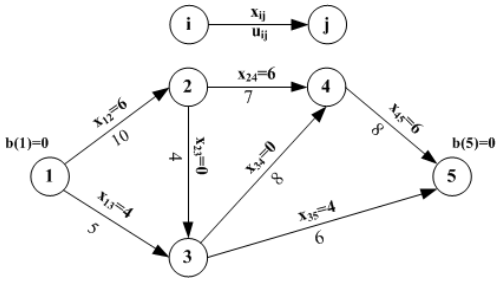
In the Phase II of Algorithm 1, we attempt to send a feasible flow through the generated path. First the capacity of the generated path is defined (line 15, $U = min\{u_{12} = 10, u_{24} = 7, u_{45} = 8\} = 7$). By generating a random integer number in $[1, min\{U = 7, b(1) = |b(n)| = 10\}]$ (line 16) a feasible flow for the constructed path is defined (e.g. $Flow_{p=1} = 6$). Since the demand has not been fully met (i.e., not equal to 0 yet) (line 3), the above procedure (Phase I and II of Algorithm 1) is repeated. The outputs for Algorithm 1 are the generated paths as well as their feasible flows. For instances, Fig. 6 shows a feasible solution with a second path generated for the chromosome presented in Fig. 5. This solution would be impossible to generate from the PbR method (as shown in Fig. 3).

4. Probabilistic tree-based GA (PTbGA)

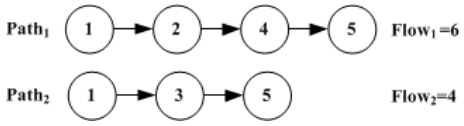
After showing the advantages of adopting PTbR over PbR in generating more freely feasible solutions for the MCFP, this section presents PTbGA (the proposed PTbR-based genetic algorithm), which incorporates PTbR

Algorithm 1 Decoding procedure for PTbR scheme

Input: *(Chromosome, Supply node (b(1)), Demand node (b(n)))*
p←1 (path counter)
while *supply* $\neq 0$ and *Demand* $\neq 0$ **do**
    $Flow_p \leftarrow 0$
    Phase I: ***Construct a path***
    i←1
    j←[ ]
    $Path_p \leftarrow${i}
    while $j \neq n$ **do**
        Generate a random number in [0,1] and define the successor node (j) based on probabilities in the sub-chromosome i (SCh_i)
        $Path_p \leftarrow Path_p \cup$ {j}
        i←j
    end while
    Phase II: ***Define a feasible flow***
    $U \leftarrow min\{u_{ij}\}$; where (i,j) are arcs in the constructed path
    $Flow_p \leftarrow$ randi $\in$ [1,$min\{U,b(1),b(n)\}$]
    update: Supply/Demand and Network
    p←p+1
end while
Output: *Path* and *Flow*



(a) Decision variables in the network.



(b) Feasible paths and flows.

Figure 6: An example of a feasible solution for the sample chromosome in Fig. 5.

into an GA. Our aim is to evolve a population of probabilistic trees, so that it will not only allow us to freely produce more feasible MCFP solutions, but also ultimately better solutions at the end of an optimisation run. Algorithm 2 summarises the procedure of PTbGA.

Algorithm 2 PTbGA procedure.

1: **Input:** (*MCFP network, PTbGA parameters*)
2: **Initialise** a population of *pop_size* individuals (Subsection 4.1).
3: **while** stopping condition is not satisfied **do**
4: **Genetic operators** are applied (Subsection 4.3).
5: **Decoding procedure** is performed N times on each chromosome (Algorithm 1).
6: **Evaluate** all individuals in the population (Subsection 4.4).
7: **Select** fitted individuals for next iteration (Subsection 4.4).
8: **Update** the best individual in each iteration (i.e. Best_Solution).
9: **end while**
10: **Output:** Best_Solution is reported (i.e. network contains *Path* and *Flow*).

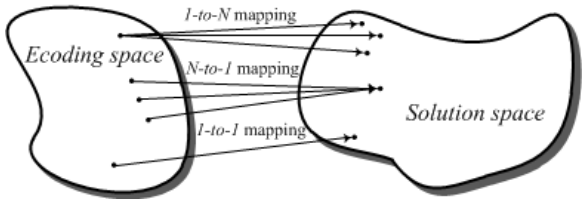


Figure 7: The mapping from the encoding space to the solution space.

4.1. Initialisation

At the start of PTbGA, a population with *pop_size* individuals is generated. Instead of applying the 100% random initialisation, α % of the population members are generated randomly and (1-α)% members of *pop_size* are seeded from the outputs of the nonlinear programming (NLP) solver, in order to give the population some good starting points. In this paper we utilised $fmincon()$ routine of the Matlab software as the NLP solver and each of its outputs is mapped to a chromosome.

4.2. Decoding procedure for PTbR

The decoding procedure is invoked on each individual to perform a mapping from a chromosome (i.e., the encoding space) to a MCFP network instance (i.e., the solution space). As shown in Fig. 7, there are three different possible mappings between the encoding space and the solution space. PTbR follows the *1-to-N* mapping, where N can be any number greater than 1. But PbR has the property of *N-to-1* mapping [29].

Since the mapping between chromosome and the solution space is 1-to-N for PTbR, by applying decoding procedure (Algorithm 1) N times on a given chromosome, N solution vectors ($\vec{\mathbf{x}}$) will be obtained. This characteristic of

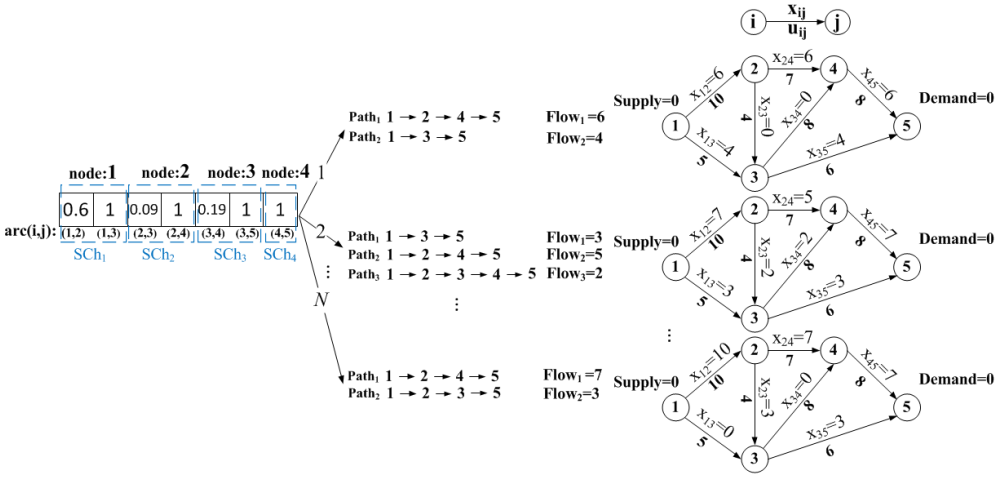


Figure 8: N times decoding procedure within PTbGA for the example chromosome in Fig. 5.

the PTbR scheme improves the searching capability of PTbGA within the feasible region and provides an algorithm a better chance to generate the better quality solutions. Hence, in PTbGA, instead of performing N=1 time decoding for a given chromosome, the decoding procedure in Algorithm 1 is applied N times for a given chromosome. The N decoding procedure for the given chromosome is shown in Fig. 8.

4.3. Crossover and mutation

In GA, search is carried out by applying genetic operators to selected individuals from the population, e.g., using crossover and mutation operators to generate new offspring from the selected parents (chromosomes) in each generation. To perform a crossover operator, two parents are randomly selected from the population and two sub-chromosomes are randomly selected within each parent (SCh_A and SCh_B), as shown in Fig. 9. Then the crossover operator is applied to swap the sub-chromosomes between the two parents. To perform mutation on each chromosome, a random parent is selected from population and a randomly chosen sub-chromosome is regenerated to create a new offspring. Note that the crossover and mutation operators always produce feasible solutions.

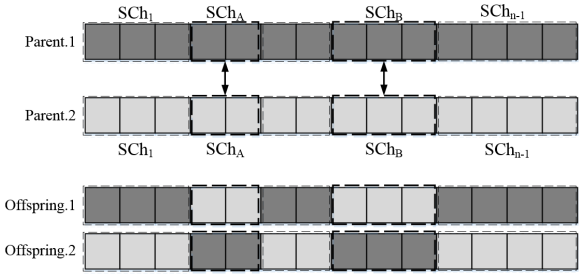


Figure 9: A two point crossover operator for PTbGA.

4.4. Fitness evaluation and selection criteria

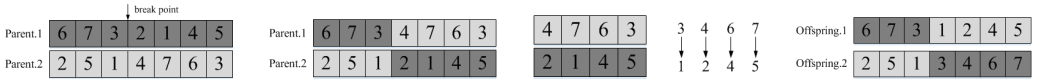
As shown in Fig. 8 and for the given chromosome, the decoding procedure presented in Algorithm 1 is applied N times which produces N solutions in the solution space. By evaluating the N solutions using Eq. 1, N objective values are obtained. To evaluate the fitness value of a given chromosome, we can either: 1) calculate the average of the objective values (Ave) for the N solutions; or 2) select the minimum ($Best$) of the N objective values. Decoding N times should provide more accurate information about the goodness of the chromosome than just decoding (or sampling) once. In Subsection 5.2, we demonstrate that the second approach (i.e. choosing the minimum ($Best$) objective value) is the most effective way of evaluating the fitness value for each chromosome. Finally, after obtaining a fitness value for each individual probabilistic tree, the tournament selection procedure is applied to select the fitter individuals from the population for the next iteration.

4.5. Stopping criteria

There are two stopping criteria for the PTbGA method: 1) reach the maximum number of function evaluations (NFEs); 2) no improvement in the best individual for β successive iterations. Either of these conditions is satisfied first, the algorithm terminates and the best individual (Best_Solution) is reported as the output.

In this paper we compare the proposed PTbGA with the priority-based GA (PrGA). The PrGA method is obtained from [29] which used priority-based representation as the encoding scheme for MCFPs. A weight mapping crossover (WMX) and inversion mutation were applied as genetic operators and tournament selection was chosen as the selection procedure [29, 33].

A weight mapping crossover (WMX) is proposed as an extension of one-cut crossover for permutation representation [34]. To apply WMX crossover,



(a) Select a break point. (b) Swap sub-strings between parents. (c) Sort and map the right sub-strings. (d) Generate offspring.

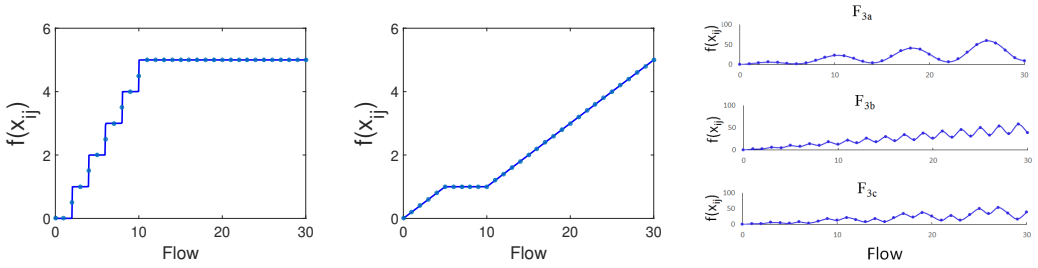
Figure 10: An example of a weight mapping crossover (WMX) for PrGA.

two parents are randomly selected from the population and a random breakpoint is chosen, as shown in Fig. 10a. Then the sub-strings on the right side of the breakpoint are swapped between parents (Fig. 10b). Sub-strings located on the right side of the break point are sorted and mapping between those sub-strings is done (Fig. 10c). Finally, the offspring based on the mapping is generated as shown in Fig. 10d. To perform inversion mutation, a parent, as well as two break points within the chromosome are randomly selected. Then the sub-string between two selected break points is inverted and an offspring is generated.

5. Experimental Studies

In this section, we first describe the MCFP instances and cost functions that have been adopted, followed by some discussion about the mathematical solver packages used in our experiments. We then present the parameter setting, experimental result analysis on the performances of PrGA, PTbGA, and mathematical solvers in solving the MCFP instances.

Since we want to solve the MCFP instances using nonlinear non-convex functions, a set of nonlinear non-convex functions commonly-used in the literature has been adopted [35, 36, 37]. Michalewicz et al. [37] categorised these nonlinear cost functions as 1) piece-wise linear cost functions; 2) multimodal (nonlinear non-convex) cost functions; 3) smooth cost functions which are mostly used for Operations Research (OR) problems. In this paper we chose the nonlinear non-convex and arc-tangent approximation of the piecewise linear cost functions to compare the performances the PTbGA, PrGA and two mathematical solver packages. The formulation of these functions are as follows [37, 35, 36]:



(a) F_1 (b) F_2 (c) F_{3a}, F_{3b} and F_{3c}

Figure 11: Shapes of different cost functions.

$$\begin{aligned} F_1 : f(x_{ij}) = c_{ij}\big(&arctan(P_A(x_{ij} - S))/\pi + 0.5 + \\ &arctan(P_A(x_{ij} - 2S))/\pi + 0.5 + \\ &arctan(P_A(x_{ij} - 3S))/\pi + 0.5 + \\ &arctan(P_A(x_{ij} - 4S))/\pi + 0.5 + \\ &arctan(P_A(x_{ij} - 5S))/\pi + 0.5\big). \end{aligned} \tag{5}$$

$$\begin{aligned} F_2 : f(x_{ij}) = c_{ij}\big(&(x_{ij}/S)(arctan(P_B x_{ij})/\pi + 0.5) + \\ &(1 - x_{ij}/S)(arctan(P_B(x_{ij} - S))/\pi + 0.5) + \\ &(x_{ij}/S - 2)(arctan(P_B(x_{ij} - 2S))/\pi + 0.5)\big). \end{aligned} \tag{6}$$

$$F_3 : f(x_{ij}) = \quad 100 \times c_{ij}\big(x_{ij}(sin\Big(\frac{5\pi x_{ij}^w}{4S}\Big) + 1.3)\big). \tag{7}$$

Note that P_A and P_B are set to 1000, S is set to 5 and c_{ij} is non-negative coefficient [14, 36]. To examine the robustness of PTbGA, the parameter w in the cost function F_3 is set to 1, 2 and 3, to generate F_{3a}, F_{3b} and F_{3c} functions, respectively. These cost functions F_1 to F_3 are illustrated in Fig. 11.

A set of 36 capacitated network instances are randomly generated with $n = \{5, 10, 20, 40, 60, 80, 100, 150, 200, 250, 400, 500\}$ nodes and presented in Table 2, where *No.* denotes the instance number. Note that for each node size n, we generate 3 different instances with m arcs. The network instances are grouped as small-sized instances (5 and 10 nodes), medium-sized instances (20-60 nodes) and large-sized instances (80-500 nodes). We are not able to utilise the existing benchmark in the literature, since all existing instances are uncapacitated networks, and the largest size of the

Table 2: A set of 36 randomly generated network instances.

small-sized problems						medium-sized problems								
No.	n	m	*No.*	n	m	*No.*	n	m	*No.*	n	m	*No.*	n	m
1		7	4		35	7		85	10		263	13		717
2	**5**	9	5	**10**	32	8	**20**	104	11	**40**	374	14	**60**	724
3		7	6		24	9		93	12		336	15		663
large-sized problems														
No.	n	m	*No.*	n	m	*No.*	n	m	*No.*	n	m	*No.*	n	m
16		1306	19		1894	22		3770	25		7704	28		12027
17	**80**	1278	20	**100**	2141	23	**150**	4205	26	**200**	7546	29	**250**	12134
18		1235	21		2063	24		4590	27		7768	30		11771
No.	n	m	*No.*	n	m									
31		30247	34		47029									
32	**400**	30210	35	**500**	47554									
33		29556	36		48839									

networks are 50 nodes, however in this paper we aim to solve the large-scale capacitated network instances with up to 500 nodes and 48,000 arcs.

5.1. Mathematical programming methods and challenges

The MCFP using nonlinear non-convex cost functions is categorised as mixed integer nonlinear programming (MINLP) problems. There are limited mathematical solver packages which are able to solve MINLPs such as CPLEX, Couenne, Baron, LINDOGlobal and AlphaECP [38, 39, 40]. However, each of these solvers has some limitations. For instance, CPLEX is only able to solve quadratic optimisation problems and it is unable to model and solve general MINLPs. BARON [41] cannot handle the trigonometric functions $sin(x)$, $cos(x)$, while Couenne is not able to handle the *arctangent* function. Among these solvers, AlphaECP and LINDOGlobal are the only ones which are able to handle general MINLPs [39, 40]. Considering all the limitations that mathematical programming methods encountered, in this paper we choose to compare PTbGA and PrGA's results with those of LINDOGlobal and AlphaECP.

5.2. Parameter settings

In this subsection we study the behaviour of different parameters and operators of the PTbGA and PrGA. One way of doing that is to use a full factorial design to study the behaviour of parameters and operators [42, 43, 44]. When the number of factors increases, the full factorial design becomes inefficient [45]. The most important parameters for PTbGA are: population size

Table 3: Factors and their levels in the proposed PTbGA.

Factors	Symbols	Level	Factors	Symbols	Levels	Factors	Symbols	Levels	Factors	Symbols	Levels	Factors	Symbols	Levels
	A(1)	1		B(1)	20		C(1)	0.80		D(1)	0.10	Fitness	E(1)	*Ave*
N decoding	A(2)	5	*pop_size*	B(2)	40	P_c	C(2)	0.85	P_m	D(2)	0.15		E(2)	*Best*
	A(3)	10		B(3)	60		C(3)	0.90		D(3)	0.20			
	A(4)	20		B(4)	80		C(4)	0.95		D(4)	0.25			

Table 4: Factors and their levels in the PrGA.

Factors	Symbols	Levels	Factors	Symbols	Levels	Factors	Symbols	Levels
	A(1)	100		B(1)	0.80		C(1)	0.10
pop_size	A(2)	200	P_c	B(2)	0.85	P_m	C(2)	0.15
	A(3)	300		B(3)	0.90		C(3)	0.20

(*pop_size*), N times decoding (or sampling) for each chromosome, crossover rate (P_c), mutation rate (P_m). One important consideration for PTbGA is how we evaluate the fitness value of a chromosome, either using the average of N objective values (*Ave*) or using the minimum (*Best*) of the N objective values (Subsection 4.4). For PrGA, the most important parameters are *pop_size*, P_c and P_m. These parameters for PTbGA and PrGA and their levels are presented in Tables 3 and 4. As shown in Table 3, for PTbGA, we consider four 4-level factors and one 2-level factor with 30 network instances which runs five times [45]. Hence, the total number of runs for a full factorial design for PTbGA is $36 \times 4^4 \times 2^1 \times 5 = 92,160$, and this value for PrGA is $36 \times 3^3 \times 5 = 4,860$.

Instead of running the full factorial design test, several experimental designs have been suggested to reduce the number of experiments [46]. Among these, the Taguchi method has been successfully applied for a systematic approach in optimisation [47]. Using the Taguchi design for our experiments, we try to evaluate different combinations of parameter values. We also try to maximise the performance of the signal-to-noise (S/N) ratios by solving a partial set of experiments (instead of the full set of experiments) utilising the orthogonal arrays [48]. In our experiments, the desirable value (response variable) is denoted by *signal*, and the undesirable (standard deviation) value is denoted by *noise*. The S/N ratio of the minimisation objective is as follows [49]:

$$S/N \quad ratio = -10 \log_{10}(objective function)^2. \tag{8}$$

In Taguchi methods, adopting the proper orthogonal array for the chosen problem can be a challenging task [50]. For PTbGA, we have four 4-level factors and one 2-level factor. Hence, the appropriate orthogonal array for

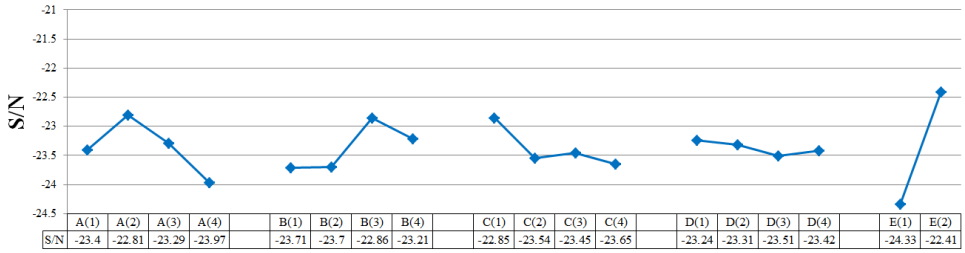
this method is L16. In L16 (4^5) orthogonal array, five 4-level factors are considered. But in our PTbGA we have four 4-level and one 2-level (the last factor is fitness). To assign the 2-level factor to the 4-level column from the orthogonal array L16, each of these levels is required to be replicated twice. It is essential to notice that, after applying these techniques, the obtained array remains orthogonal and this method is called means of adjustment techniques [51]. For PrGA, since we have three 3-level factors, the appropriate orthogonal array is L9 (3^3).

Since we have L16 and L9 orthogonal arrays, there are 16 and 9 different combinations of control factors for PTbGA and PrGA, respectively. Both PrGA and PTbGA are implemented in MATLAB on a PC with Intel(R) Core(TM) i7-6500U 2.50 GHz processor with 8 GB RAM. Each algorithm is evaluated on a set of 36 instances (Table 2) using cost function F_{3a} for each combination. In order to have a fair comparison, for both PTbGA and PrGA, a fixed computational budget for each trial is considered. Since the scale of an objective function value for each instance may be different, the relative percentage deviation (RPD) is used for each instance and calculated by [51]:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}}, \tag{9}$$

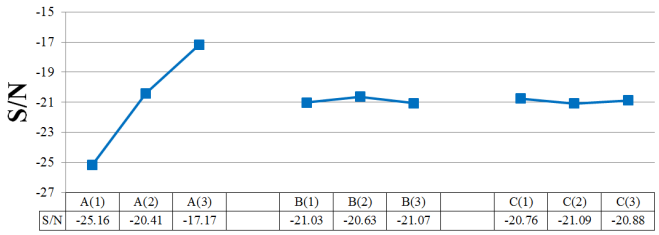
where Alg_{sol} and Min_{sol} are the obtained objective value and the best solution found for each replication of a trial over a given instance, respectively. First, we calculate the average RPD for each trial. Then, by using Eq. 8, the S/N ratio is obtained for each trial. Finally, the S/N ratios of trials at each level of factors are averaged and the results are shown in Figs. 12 and 13 for PTbGA and PrGA, respectively. For instance, as shown in Fig. 13, to obtain the value of -17.17 for factor A at level 3 (i.e. A(3)), we should calculate the average of S/N ratios for all trials in which factor A at level 3. As shown in Fig. 12 and based on Table 3, the best values for factors A, B, C, D, and E are 2, 3, 1, 1, and 2 respectively. These results demonstrate that the N=5 times decoding, pop_size=60, P_c=0.8, P_m=0.1, and assigning the *Best* (i.e., the minimum) objective value among N objective values have the highest performance for PTbGA. As shown in Fig. 13 and based on Table 4 for PrGA, the best values for A, B, and C are 3, 2, and 1 respectively (i.e. pop_size=300, P_c=0.85 and P_m=0.1).

The other parameters for PTbGA and PrGA are set as follows: maximum number of iterations (It_{max}=200) and maximum number of function evalu-



	A(1)	A(2)	A(3)	A(4)		B(1)	B(2)	B(3)	B(4)		C(1)	C(2)	C(3)	C(4)		D(1)	D(2)	D(3)	D(4)		E(1)	E(2)
S/N	-23.4	-22.81	-23.29	-23.97		-23.71	-23.7	-22.86	-23.21		-22.85	-23.54	-23.45	-23.65		-23.24	-23.31	-23.51	-23.42		-24.33	-22.41

Figure 12: Mean S/N ratio plot for each level of the factors in PTbGA.



	A(1)	A(2)	A(3)		B(1)	B(2)	B(3)		C(1)	C(2)	C(3)
S/N	-25.16	-20.41	-17.17		-21.03	-20.63	-21.07		-20.76	-21.09	-20.88

Figure 13: Mean S/N ratio plot for each level of the factors in PrGA.

ations ($NFEs$=100,000). In order to have an adequate level of randomness in the initialisation phase, α is set to 95%. If the results are not improved in $\beta = 30$ successive iterations, the algorithm is terminated. The parameter for mathematical solvers (AlphaECP and LINDOGlobal) is running time which is set to 3600 seconds (s) for problem instances up to n=100 nodes and 7200s for larger problem instances.

5.3. Results and analysis

After defining the parameters and operators for PTbGA and PrGA, we use a set of 36 network instances using cost functions F_1, F_2, F_{3a}, F_{3b} and F_{3c} to evaluate PTbGA, PrGA, AlphaECP, and LINDOGlobal. The results are provided in Tables 5 to 9. *std* and t (for PTbGA and PrGA) denote the standard deviation of the results and the average of running time in seconds respectively, and the *mean* represents the average of objective function values over 30 runs. The t' and OBJ for LINDOGlobal and AlphaECP denote the running time and the objective value, respectively. "NF" denotes that the mathematical solver cannot find any feasible solution within the time limit. For LINDOGlobal and AlphaECP on network instances up to 100 nodes, the time limit is set to 3600s, and on instances with n={150,200,250,400,500}, it is set to 7200s. In some cases the solvers find the optimal solution be-

Table 5: Result comparison of PTbGA with other algorithms using cost function F_1.

No.	n	m	PTbGA			PrGA			LINDOGlobal		AlphaECP		*Gap*	h
			t	*mean*	*std*	t	*mean*	*std*	t'	*OBJ*	t'	*OBJ*		
small-sized instances														
1		7	32	**0.0038**	0.00E+00	26	4.0018	0.00E+00	1	**0.0038**	2	4.0017	0.0%	0
2	5	9	28	**1.0033**	2.28E-16	27	4.0015	9.11E-16	2	**1.0033**	4	4.0015	0.0%	0
3		7	35	**1.0027**	2.28E-16	30	4.0012	0.00E+00	1	**1.0027**	1	3.0016	0.0%	0
4		35	57	**0.0098**	1.26E-05	65	4.0083	5.32E-05	6	**0.0098**	55	2.0097	0.0%	0
5	10	32	73	**0.0092**	1.47E-05	65	2.5085	2.14E-04	10	**0.0092**	17	3.0080	0.0%	0
6		24	60	**1.0071**	4.56E-16	66	3.5065	4.56E-16	112	**1.0071**	250	2.5073	0.0%	0
medium-sized instances														
7		85	120	0.0244	1.30E-05	116	2.0245	2.69E-05	3600	**0.0243**	150	0.0254	-0.4%	-1
8	20	104	130	**0.0307**	2.93E-06	128	0.0311	0.00E+00	575	**0.0307**	162	0.0308	0.0%	0
9		93	95	**1.5270**	1.19E-05	99	2.5276	2.40E-04	3600	**1.5270**	135	6.5256	0.0%	0
10		263	168	0.0705	3.16E-06	165	0.2474	2.44E-01	3600	**0.0704**	3600	0.0707	-0.1%	-1
11	40	374	195	**0.1036**	4.27E-17	195	0.1043	2.07E-05	3600	**0.1036**	3600	0.1037	0.0%	0
12		336	207	**0.0913**	2.85E-17	197	0.0938	1.56E-04	3600	**0.0913**	3600	0.0926	0.0%	0
13		717	261	**0.2065**	4.26E-06	262	3.7306	1.12E-01	3600	**0.2065**	3600	0.2079	0.0%	0
14	60	724	273	**0.1893**	2.01E-05	266	0.4662	4.43E-01	3600	**0.1893**	3600	0.1908	0.0%	0
15		663	262	**0.1874**	5.70E-17	261	0.214	1.12E-01	3600	**0.1874**	3600	0.1881	0.0%	0
large-sized instances														
16		1306	466	**0.3469**	1.69E-05	439	5.2462	2.61E-01	3600	**0.3469**	3600	0.8494	0.0%	0
17	80	1278	450	0.3334	1.37E-05	436	1.5602	3.02E-01	3600	**0.3333**	3600	0.3343	0.0%	-1
18		1235	485	**0.3359**	9.48E-06	462	1.3374	2.28E-04	3600	**0.3359**	3600	2.3361	0.0%	0
19		1894	629	**0.4970**	1.39E-05	620	0.5486	1.54E-01	3600	0.4971	3600	0.4973	0.0%	1
20	100	2141	641	**6.0648**	6.08E-06	645	6.5654	3.08E-05	3600	**6.0648**	3600	6.0649	0.0%	0
21		2063	580	**0.5561**	1.57E-05	585	2.5568	1.51E-04	3600	**0.55610**	3600	0.5574	0.0%	0
22		3770	1067	**0.9845**	2.37E-05	1061	1.8852	3.08E-01	7200	NF	7200	0.9851	0.1%	1
23	150	4205	1679	**1.112**	1.49E-04	1670	9.158	1.54E-01	7200	NF	7200	1.6131	45.1%	1
24		4590	1160	**1.1731**	2.28E-16	1136	1.1738	1.20E-04	7200	**1.1731**	7200	2.6729	0.0%	0
25		7704	1703	**2.0237**	1.91E-05	1698	2.0253	1.64E-04	7200	NF	7200	2.0242	0.0%	1
26	200	7546	1683	**1.9875**	2.20E-04	1657	6.8599	7.93E-01	7200	NF	7200	3.4853	75.4%	1
27		7768	1781	**1.9900**	6.68E-05	1770	9.8616	2.75E-01	7200	NF	7200	3.9902	100.5%	1
28		12027	2680	**3.2414**	1.24E-04	2666	3.2418	1.64E-04	7200	NF	7200	3.2417	0.0%	1
29	250	12134	2632	**3.1387**	5.52E-04	2617	7.1371	1.03E-04	7200	NF	7200	4.1395	31.9%	1
30		11771	2419	**3.0327**	4.68E-16	2391	3.5352	5.29E-04	7200	NF	7200	**3.0327**	0.0%	0
31		30247	3048	**11.2415**	1.29E-01	2959	23.7160	2.86E+00	7200	NF	7200	26.7962	138.4%	1
32	400	30210	3062	**7.8540**	6.99E-13	3035	13.1003	3.53E-01	7200	NF	7200	13.8495	76.3%	1
33		29556	2969	**7.6520**	9.11E-16	3037	9.6828	1.29E-01	7200	NF	7200	10.1493	32.6%	1
34		47029	3512	**12.1615**	1.12E-04	3485	14.6596	2.55E-04	7200	NF	7200	14.6602	20.5%	1
35	500	47554	3561	**12.5239**	2.62E-04	3588	12.5242	7.15E-05	7200	NF	7200	32.5245	159.7%	1
36		48839	3471	**12.8661**	1.30E-04	3479	14.6661	3.16E-01	7200	NF	7200	15.3649	19.4%	1

fore reaching the time limit or other stopping criterion is met (e.g., on NLP sub-problems it stops improving), and consequently the algorithms are terminated.

In order to compare the performance of PTbGA with PrGA (two GA variants method), the t-test with the significance level of 0.05 is performed. The PTbGA outperforms PrGA in all instances using different cost functions (F_1-F_{3c}). It is clearly evident that the superior performance of the PTbGA comes from utilising PTbR in its procedure, instead of using PbR in PrGA. For instance, in Table 5, PrGA cannot find optimal solutions for all small instances (No.1-6) using function F_1. It is apparent that PbR is restricted

Table 6: Result comparison of PTbGA with other algorithms using cost function F_2.

No.	n	m	PTbGA			PrGA			LINDOGlobal		AlphaECP		*Gap*	h
			t	*mean*	*std*	t	*mean*	*std*	t'	*OBJ*	t'	*OBJ*		
small-sized instances														
1		7	24	**3.9997**	4.56E-16	25	**3.9997**	9.11E-16	1	**3.9997**	2	**3.9997**	0.0%	0
2	5	9	31	**3.9997**	0.00E+00	33	**3.9997**	9.11E-16	9	**3.9997**	2	**3.9997**	0.0%	0
3		7	31	**3.9996**	4.56E-16	28	3.9997	0.00E+00	2	**3.9996**	1	**3.9996**	0.0%	0
4		35	80	**5.9992**	1.82E-15	82	6.1994	2.73E-15	631	**5.9992**	39	7.1996	0.0%	0
5	10	32	56	**6.7994**	9.11E-16	61	**6.7994**	9.11E-16	325	**6.7994**	28	8.9991	0.0%	0
6		24	69	**7.7994**	4.56E-15	65	7.7995	9.11E-16	246	**7.7994**	21	7.7995	0.0%	0
medium-sized instances														
7		85	127	**9.1987**	3.65E-15	124	9.399	3.65E-15	3600	**9.1987**	85	12.7981	0.0%	0
8	20	104	106	**8.5986**	3.65E-15	115	8.5989	2.85E-05	3600	**8.5986**	142	**8.5986**	0.0%	0
9		93	252	**10.7987**	1.82E-15	251	11.8288	7.32E-02	3600	**10.7987**	123	13.5980	0.0%	0
10		263	170	**9.7988**	1.15E-04	163	9.9389	1.31E-01	3600	10.1988	3600	10.5986	4.1%	1
11	40	374	199	**6.5987**	9.11E-16	208	7.7488	4.89E-01	3600	**6.5987**	3600	6.7990	0.0%	0
12		336	183	**8.9993**	0.00E+00	181	10.009	2.86E-01	3600	**8.9993**	3600	10.3982	0.0%	0
13		717	274	**9.7990**	3.07E-13	270	10.7987	5.99E-05	3600	**9.7990**	3600	20.9981	0.0%	0
14	60	724	254	**10.8687**	1.40E-02	258	**10.8687**	3.13E-01	3600	11.1985	3600	14.3985	3.0%	1
15		663	244	**5.8390**	1.39E-01	242	5.9992	3.84E-01	3600	6.5988	3600	9.7985	13.0%	1
large-sized instances														
16		1306	414	10.5985	0.00E+00	411	11.9187	7.69E-01	3600	12.5983	3600	**10.5976**	0.0%	-1
17	80	1278	413	**12.398**	1.82E-15	422	12.6084	2.94E-01	3600	**12.3980**	3600	12.5981	0.0%	0
18		1235	413	**12.6084**	5.21E-01	413	13.0787	2.93E-01	3600	13.3985	3600	13.1984	4.7%	1
19		1894	783	**8.1987**	4.27E-05	527	9.4386	3.59E-01	3600	9.1983	3600	9.1984	12.2%	1
20	100	2141	679	**17.9483**	7.85E-03	552	**17.9483**	8.89E-02	3600	17.9979	3600	19.3974	0.3%	1
21		2063	677	**15.4181**	6.33E-02	536	15.4385	3.08E-01	3600	15.7970	3600	17.5980	2.5%	1
22		3770	1033	**9.3986**	7.18E-05	982	10.1088	3.92E-01	3600	9.3987	7200	13.3987	0.0%	1
23	150	4205	1124	**15.1481**	2.23E-01	937	15.3981	6.09E-01	3600	16.7982	7200	15.1981	0.3%	0
24		4590	1088	**7.6291**	4.37E-01	1314	8.3985	3.12E-05	3600	8.3985	7200	8.3983	10.1%	1
25		7704	1415	**10.3988**	5.47E-15	1585	10.8088	6.31E-01	7200	**10.3988**	7200	10.9985	0.0%	0
26	200	7546	1821	**13.7981**	8.54E-05	1487	15.2389	1.26E+00	7200	NF	7200	13.9978	1.4%	1
27		7768	1439	**15.7282**	2.62E-01	1587	17.4483	5.65E-01	7200	NF	7200	17.1976	9.3%	1
28		12027	2394	**13.0486**	5.06E-01	2543	13.1978	1.54E-04	7200	NF	7200	13.5966	4.2%	1
29	250	12134	2292	**15.7986**	5.03E-01	2294	16.3182	1.93E-01	7200	NF	7200	17.5968	11.4%	1
30		11771	2861	**9.3980**	6.23E-05	3083	10.2088	6.27E-01	7200	NF	7200	9.5979	2.1%	1
31		30247	3105	**11.5452**	3.58E-01	3197	24.9752	4.37E+00	7200	NF	7200	26.39478	128.6%	1
32	400	30210	3097	**17.4518**	1.37E+00	2998	34.054	5.06E+00	7200	NF	7200	37.792297	116.6%	1
33		29556	3096	**11.0387**	8.18E-01	3146	21.0264	4.40E+00	7200	NF	7200	23.596053	113.8%	1
34		47029	3573	**14.0783**	4.59E-01	3513	23.5957	3.65E-15	7200	NF	7200	23.595671	67.6%	1
35	500	47554	3418	**8.7857**	8.19E-01	3475	22.5954	8.29E-10	7200	NF	7200	22.595416	157.2%	1
36		48839	3493	**12.025**	4.46E-01	3577	22.5954	1.84E-08	7200	NF	7200	22.595416	87.9%	1

to a limited subset of the feasible region. As a result PrGA could not find optimal solutions for the small-sized instances.

We also compare the performance of PTbGA with LINDOGlobal and AlphaECP by performing a one-sample t-test with the significance level set to 0.05. If PTbGA has statistically better or worse performance than that of the mathematical solvers, the parameter h is set to 1 and -1 respectively, otherwise h is set to 0. To calculate the value of *Gap* between PTbGA and other mathematical solvers' results, the following equation is used: $Gap = \big((min\{OBJ_{AlphaECP}, OBJ_{LINDOGlobal}\} - mean_{PTbGA})/mean_{PTbGA}\big) \times 100$. In Tables 5-9, the best objective values among LINDOGlobal, AlphaECP and the average of 30 runs for the PTbGA and PrGA are presented in boldface.

Table 7: Result comparison of PTbGA with other algorithms using cost function F_{3a}.

No.	n	m	PTbGA			PrGA			LINDOGlobal		AlphaECP		*Gap*	h
			t	*mean*	*std*	t	*mean*	*std*	t'	*OBJ*	t'	*OBJ*		
small-sized instances														
1	5	7	24	**12.4142**	3.65E-15	25	**12.4142**	3.65E-15	1	**12.4142**	1	17.7868	0.0%	0
2		9	31	**12.4142**	3.65E-15	30	16.6142	0.00E+00	1	**12.4142**	2	17.7868	0.0%	0
3		7	32	**12.4142**	3.65E-15	27	**12.4142**	3.65E-15	1	**12.4142**	1	**12.4142**	0.0%	0
4	10	35	62	23.3427	3.74E-01	60	23.7017	3.23E-01	578	**22.6355**	112	29.5655	-3.0%	-1
5		32	63	33.7005	1.46E-14	62	39.0863	0.00E+00	498	**33.2152**	119	48.0589	1.4%	-1
6		24	56	**24.5503**	7.29E-15	59	52.865	1.46E-14	147	**24.5503**	134	49.9939	0.0%	0
medium-sized instances														
7	20	85	173	**49.9574**	0.00E+00	105	56.8499	4.43E-01	3600	**49.9574**	3600	96.5076	0.0%	0
8		104	81	**50.0995**	0.00E+00	101	51.0446	1.46E+00	3600	**50.0995**	3600	69.2579	0.0%	0
9		93	104	**53.8000**	2.19E-14	117	54.2162	0.00E+00	3600	**53.8000**	3600	55.5370	0.0%	0
10	40	263	165	**43.9655**	7.29E-15	149	51.2448	6.86E-01	3600	**43.9655**	3600	85.9451	0.0%	0
11		374	191	**40.1289**	7.29E-15	175	44.2344	5.34E-01	3600	**40.1289**	3600	126.8589	0.0%	0
12		336	108	**48.0355**	1.50E-14	161	61.9575	2.58E+00	3600	48.0360	3600	75.4873	0.0%	1
13	60	717	251	**54.0853**	1.46E-14	245	61.9999	4.62E-01	3600	**54.0853**	3600	128.8355	0.0%	0
14		724	203	**50.5289**	1.46E-14	242	71.3269	4.62E+00	3600	**50.5289**	3600	84.7289	0.0%	0
15		663	265	**40.3431**	1.46E-14	232	52.5299	2.40E+00	3600	**40.3432**	3600	62.9218	0.0%	1
large-sized instances														
16	80	1306	404	**53.9005**	7.29E-15	405	70.2128	3.93E+00	3600	**53.9005**	3600	116.4883	0.0%	0
17		1278	397	**59.5147**	1.46E-14	359	68.4995	4.48E+00	3600	**59.5147**	3600	115.0995	0.0%	0
18		1235	362	**61.0782**	7.29E-15	385	71.9623	3.83E+00	3600	64.6284	3600	66.8711	5.8%	1
19	100	1894	584	**54.0711**	2.92E-14	501	70.6492	2.74E+00	3600	**54.0711**	3600	163.9421	0.0%	0
20		2141	626	81.2045	3.43E-01	531	85.2692	2.71E+00	3600	**79.5005**	3600	137.5025	-2.1%	-1
21		2063	534	**81.1964**	3.38E-01	541	84.3428	4.36E+00	3600	81.2721	3600	114.8010	1.0%	0
22	150	3770	985	**59.2212**	4.50E+00	1019	87.8434	3.30E+00	7200	91.0721	7200	112.6650	53.8%	1
23		4205	948	**59.9465**	2.40E+00	1071	91.9019	3.66E+00	7200	148.8142	7200	95.3878	59.1%	1
24		4590	1036	**69.7926**	2.52E+00	994	87.3404	1.75E+00	7200	96.7432	7200	91.5421	31.2%	1
25	200	7704	1505	**74.7676**	3.49E+00	1407	104.8014	2.65E+00	7200	178.8802	7200	110.0944	47.2%	1
26		7546	1418	**86.943**	9.33E+00	1533	120.7734	1.33E+01	7200	NF	7200	251.8740	189.7%	1
27		7768	1513	**78.9748**	3.50E+00	1648	86.3431	1.46E-14	7200	86.3432	7200	151.3330	9.3%	1
28	250	12027	2361	**97.4219**	6.13E+00	2119	118.1286	6.66E+00	7200	NF	7200	318.8519	227.3%	1
29		12134	2064	**88.5444**	5.63E+00	2015	107.5992	1.54E+00	7200	NF	7200	110.4081	24.7%	1
30		11771	2096	**71.4761**	5.99E+00	2054	78.0109	2.61E+00	7200	104.9594	7200	86.1289	20.5%	1
31	400	30247	3058	**65.5005**	4.88E+00	3112	158.7264	4.51E+01	7200	NF	7200	181.95736	177.8%	1
32		30210	2950	**105.9835**	6.95E+00	3022	282.7109	8.75E+01	7200	NF	7200	327.53503	209.0%	1
33		29556	3026	**59.4162**	4.60E+00	3110	160.9017	5.79E+01	7200	NF	7200	203.03755	241.7%	1
34	500	47029	3597	**72.6626**	4.29E+00	3543	186.7592	6.21E+00	7200	NF	7200	189.78478	161.2%	1
35		47554	3528	**63.7279**	5.24E+00	3592	163.8156	4.65E+00	7200	NF	7200	166.08019	160.6%	1
36		48839	3415	**61.8586**	3.07E+00	3432	144.5961	6.73E+00	7200	NF	7200	147.87511	139.1%	1

Table 10 summarises the results of PTbGA, LINDOGlobal and AlphaECP for solving all instances of different sizes using all nonlinear non-convex cost functions. We count the number of "win-draw-lose"(W-D-L) among these methods. For each instance in the table (i.e., each row), if the value of h is 1, 0 or -1, we add the number of wins, draws and loses, respectively. For instance, as shown in Table 5 and for large-sized problems, the W-D-L score is 14-6-1, meaning that PTbGA wins 14 times, draws 6 times, and loses 1 time for solving the large-sized instances using cost function F_1 as compared with the minimum of the LINDOGlobal and AlphaECP objective values. The aforementioned calculation is considered for all the results in Tables 5-9, and summarised in Table 10.

Table 8: Result comparison of PTbGA with other algorithms using cost function F_{3b}.

No.	n	m	PTbGA			PrGA			LINDOGlobal		AlphaECP		*Gap*	h
			t	*mean*	*std*	t	*mean*	*std*	t'	*OBJ*	t'	*OBJ*		
small-sized instances														
1		7	36	**40.4142**	0.00E+00	25	**40.4142**	0.00E+00	84	**40.4142**	156	49.2639	0.0%	0
2	5	9	23	**40.4142**	0.00E+00	26	43.2426	1.46E-14	480	**40.4142**	395	57.7492	0.0%	0
3		7	24	**40.4142**	0.00E+00	26	**40.4142**	0.00E+00	106	**40.4142**	85	**40.4142**	0.0%	0
4		35	50	**58.0213**	1.31E-14	59	63.7826	8.80E-01	3600	**58.0213**	3600	78.2284	0.0%	0
5	10	32	63	**63.8142**	7.29E-15	60	71.3596	5.09E+00	3600	**63.8142**	3600	96.7056	0.0%	0
6		24	43	**69.8355**	1.56E-14	55	85.7563	0.00E+00	3600	**69.8355**	3600	72.6640	0.0%	0
medium-sized instances														
7		85	126	75.8569	2.44E-14	112	90.135	2.92E-14	3600	**75.6284**	3600	110.1137	-0.3%	-1
8	20	104	151	67.2292	1.52E+00	102	67.9208	0.00E+00	3600	**65.4569**	3600	83.3848	-2.6%	-1
9		93	97	97.1268	1.48E+00	119	114.9492	1.84E-14	3600	**96.0640**	3600	174.0639	-1.1%	-1
10		263	243	77.0225	5.20E-01	174	87.7858	1.48E+00	3600	**74.4426**	3600	132.6487	-3.3%	-1
11	40	374	223	**47.0695**	1.84E-01	188	59.9811	1.64E+00	3600	49.0355	3600	74.5350	4.2%	1
12		336	201	**74.917**	1.35E+00	180	84.4553	4.95E+00	3600	75.6284	3600	211.2853	0.9%	1
13		717	328	**86.9664**	2.76E-01	256	104.2283	1.06E+00	3600	88.7208	3600	120.3770	2.0%	1
14	60	724	326	**81.6177**	1.41E+00	288	90.2898	2.19E+00	3600	84.4782	3600	125.9317	3.5%	1
15		663	232	**48.3425**	4.13E-01	266	55.9722	2.97E+00	3600	53.2782	3600	106.5564	10.2%	1
large-sized instances														
16		1306	407	**107.0965**	2.29E+00	420	118.6993	1.86E+00	3600	112.1208	3600	188.2985	4.7%	1
17	80	1278	433	**95.9933**	1.20E+00	411	102.5468	1.84E+00	3600	198.8833	3600	137.1635	42.9%	1
18		1235	472	114.1646	1.78E+00	408	123.9676	2.72E+00	3600	**112.9421**	3600	147.0629	-1.1%	-1
19		1894	610	**65.904**	5.36E-01	549	72.3289	2.85E+00	3600	126.8558	3600	128.1777	92.5%	1
20	100	2141	560	**157.3409**	6.74E+00	556	168.0308	7.53E+00	3600	242.7624	3600	199.8843	27.0%	1
21		2063	525	**120.9239**	4.75E-01	514	131.5095	3.48E+00	3600	244.6335	3600	204.1837	68.9%	1
22		3770	937	**86.9786**	7.16E+00	951	151.0684	6.39E+00	7200	285.0914	7200	154.1776	77.3%	1
23	150	4205	943	**158.4398**	2.99E+00	1175	221.2248	1.15E+01	7200	297.4548	7200	344.0700	87.7%	1
24		4590	975	**67.1359**	3.33E+00	1059	101.6343	5.11E+00	7200	202.2122	7200	160.5198	139.1%	1
25		7704	1448	**91.0785**	4.16E+00	1456	179.9828	1.14E+01	7200	NF	7200	154.1777	69.3%	1
26	200	7546	1427	**165.536**	1.32E+01	1355	338.5548	1.81E+01	7200	NF	7200	211.1056	27.5%	1
27		7768	1502	**175.6161**	6.11E+00	1489	289.4556	5.94E+00	7200	NF	7200	300.0548	70.9%	1
28		12027	2331	**102.806**	4.41E+00	2076	208.2315	9.57E+00	7200	NF	7200	453.7210	341.3%	1
29	250	12134	2283	**149.8248**	3.72E+00	2106	227.9566	8.40E+00	7200	NF	7200	195.5056	30.5%	1
30		11771	1973	**99.6941**	9.16E+00	2055	189.4755	1.22E+01	7200	NF	7200	161.6132	62.1%	1
31		30247	3030	**110.9487**	3.60E+00	3049	214.449	4.23E+01	7200	NF	7200	234.69037	111.5%	1
32	400	30210	2984	**170.0742**	7.53E+00	3043	284.6509	7.73E+01	7200	NF	7200	345.98478	103.4%	1
33		29556	3020	**97.5071**	6.73E+00	3105	220.0146	4.04E+01	7200	NF	7200	234.14113	140.1%	1
34		47029	3542	**121.0351**	6.15E+00	3507	208.8335	6.22E+00	7200	NF	7200	215.21219	77.8%	1
35	500	47554	3345	**70.5963**	4.38E+00	3371	176.6044	3.82E+00	7200	NF	7200	179.04062	153.6%	1
36		48839	3437	**109.0453**	1.56E+00	3498	188.5628	6.22E+00	7200	NF	7200	215.21219	97.4%	1

As shown in Table 5, LINDOGlobal could not find any feasible solutions for problems instances with n=150, 200, 250, 400 and 500 (except No. 24). In terms of solution quality, PTbGA is better than or equal to AlphaECP on all large-sized instances except instance No.17. The performance *Gap* between PTbGA and AlphaECP becomes significantly greater for the instances with n=400 and 500. AlphaECP got stuck in local optima for small instances No.1-6, and could not find the optimal solutions. For function F_2 and when a size of the problem is increased, LINDOGlobal is not able to find any feasible solution even after 7200s. PTbGA has better quality and efficiency than AlphaECP on all large-sized instances except No.16. Also worth noting that, PTbGA is superior than PrGA on all instances using F_1 and F_2 functions.

Table 9: Result comparison of PTbGA with other algorithms using cost function F_{3c}.

No.	n	m	PTbGA			PrGA			LINDOGlobal		AlphaECP		*Gap*	h
			t	*mean*	*std*	t	*mean*	*std*	t'	*OBJ*	t'	*OBJ*		
small-sized instances														
1	5	7	21	**17.7868**	3.65E-15	19	**17.7868**	3.65E-15	13	**17.7868**	5	**17.7868**	0.0%	0
2		9	23	**17.7868**	3.65E-15	21	**17.7868**	3.65E-15	104	**17.7868**	36	**17.7868**	0.0%	0
3		7	21	**17.7868**	3.65E-15	20	**17.7868**	3.65E-15	14	**17.7868**	15	**17.7868**	0.0%	0
4	10	35	45	33.6934	7.64E-01	56	35.4376	1.46E-14	3600	**32.9299**	3600	45.5650	-2.3%	-1
5		32	70	36.3853	7.03E-01	66	42.9655	1.46E-14	3600	**35.5299**	3600	44.0590	-2.4%	-1
6		24	59	**41.3595**	4.68E-01	60	52.865	0.00E+00	3600	41.5513	3600	58.5221	0.5%	0
medium-sized instances														
7	20	85	161	55.8904	1.60E+00	111	57.0152	0.00E+00	3600	**50.1726**	3600	75.5723	-10.2%	-1
8		104	116	**44.6081**	0.00E+00	103	45.6321	1.93E+00	3600	48.4365	3600	53.2294	8.6%	1
9		93	126	**50.2162**	0.00E+00	115	**50.2162**	0.00E+00	3600	**50.2162**	3600	67.5513	0.0%	0
10	40	263	124	**49.3124**	1.95E+00	151	49.4919	1.23E+00	3600	55.2365	3600	89.5320	12.2%	1
11		374	174	**40.4579**	7.29E-15	183	46.1681	1.15E+00	3600	47.0284	3600	75.0300	16.2%	1
12		336	133	55.0127	1.31E+00	170	58.6588	3.22E+00	3600	**51.2223**	3600	97.5218	-9.6%	-1
13	60	717	214	54.9165	6.67E-01	242	56.4198	5.61E-01	3600	**54.7797**	3600	241.5177	-0.2%	0
14		724	255	**65.2548**	4.44E+00	259	67.4152	1.88E+00	3600	67.5076	3600	138.8497	3.5%	1
15		663	259	**37.5143**	1.55E+00	219	51.2203	1.74E+00	3600	38.7716	3600	83.2873	3.4%	1
large-sized instances														
16	80	1306	459	**70.4999**	1.95E+00	430	86.2443	1.91E+00	3600	254.2510	3600	130.8650	85.6%	1
17		1278	467	**75.448**	3.29E+00	463	81.3351	1.59E+00	3600	99.9421	3600	137.2071	32.5%	1
18		1235	448	**79.1008**	4.28E+00	470	90.531	3.70E+00	3600	106.2355	3600	144.4630	34.3%	1
19	100	1894	554	**53.8924**	9.74E-01	577	59.1795	3.43E+00	3600	90.2711	3600	169.4852	67.5%	1
20		2141	584	**83.1025**	0.00E+00	628	83.9481	2.16E+00	3600	190.7137	3600	129.8974	56.3%	1
21		2063	530	**84.5746**	1.92E+00	544	95.6089	3.90E+00	3600	221.1848	3600	140.2101	65.8%	1
22	150	3770	1142	**60.9084**	3.19E+00	1091	106.6349	1.02E+01	7200	195.9086	7200	71.9787	18.2%	1
23		4205	996	**70.7362**	2.61E+00	1045	147.9236	6.35E+00	7200	NF	7200	158.7746	124.5%	1
24		4590	1043	**50.1445**	1.04E+00	1168	83.7778	3.70E+00	7200	NF	7200	128.9888	157.2%	1
25	200	7704	1503	**70.6881**	4.17E+00	1609	147.6372	5.86E+00	7200	NF	7200	134.6457	90.5%	1
26		7546	1538	**99.5808**	4.39E+00	1423	260.8419	1.44E+01	7200	NF	7200	336.5219	237.9%	1
27		7768	1576	**96.6772**	4.90E+00	1508	218.5723	8.10E+00	7200	NF	7200	171.0406	76.9%	1
28	250	12027	2355	**90.4287**	3.96E+00	2375	185.9695	1.53E+01	7200	NF	7200	132.9645	47.0%	1
29		12134	2378	**94.4638**	4.46E+00	2280	179.9448	1.17E+01	7200	NF	7200	182.2233	92.9%	1
30		11771	2058	**67.5253**	4.79E+00	2208	160.9879	5.34E+00	7200	NF	7200	165.0782	144.5%	1
31	400	30247	3057	**62.2175**	4.12E+00	3094	144.0554	5.24E+01	7200	NF	7200	177.57258	185.4%	1
32		30210	3080	**109.1538**	8.18E+00	3175	245.5707	7.20E+01	7200	NF	7200	305.65278	180.0%	1
33		29556	3042	**55.0122**	5.12E+00	3114	137.442	5.14E+01	7200	NF	7200	170.58882	210.1%	1
34	500	47029	3554	**77.4947**	4.84E+00	3585	164.1868	1.00E+01	7200	NF	7200	171.37157	121.1%	1
35		47554	3515	**57.7303**	5.27E+00	3529	136.1355	1.57E+00	7200	NF	7200	137.66396	138.5%	1
36		48839	3495	**61.6489**	2.07E+00	3537	136.921	2.94E+00	7200	NF	7200	138.79592	125.1%	1

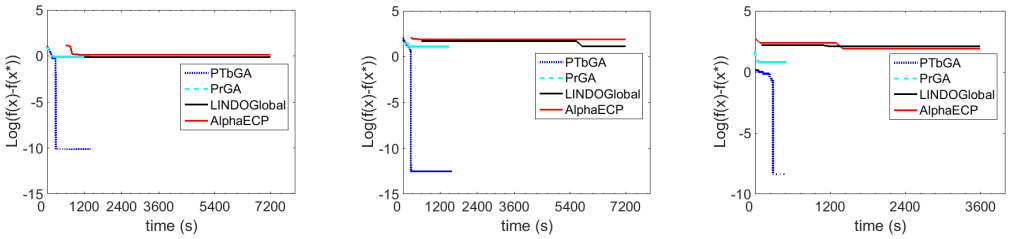
Table 10: Comparing PTbGA's performances with LINDOGlobal and AlphaECP on $F_1, F_2, F_{3a}, F_{3b}, F_{3c}$.

	Small-sized	medium-sized	Large-sized	Total score
Functions	W-D-L	W-D-L	W-D-L	W-D-L
F_1	0-6-0	0-7-2	14-6-1	14-19-3
F_2	0-6-0	3-6-0	17-3-1	20-15-1
F_{3a}	0-4-2	2-7-0	16-4-1	18-15-3
F_{3b}	0-6-0	5-0-4	20-0-1	25-6-5
F_{3c}	0-4-2	5-2-2	21-0-0	26-6-4

5.4. Robustness

To examine if PTbGA is robust to the different shapes of a cost function, we use cost function F_3 (Eq. 7), choosing three different values for the

parameter w in Eq. 7. As shown in Fig. 11c, by increasing the parameter w from 1, to 2 and 3, the number of peaks and valleys (local optima) are increased gradually in functions F_{3b}, F_{3c}. Dealing with these cost functions will be a challenging task. A robust optimisation algorithm should be able to handle this sort of highly non-convex shaped cost functions, without degrading their performances. As can be seen from Tables 7, 8, 9, LINDOGlobal has increasing difficulty in finding feasible solutions on instances with 150, 200, 250, 400 and 500 nodes. Furthermore, on all large-sized instances PTbGA outperforms AlphaECP using functions F_{3a}, F_{3b} and F_{3c}. It is evident that the mathematical solvers are sensitive to the non-convex shapes introduced in the cost functions F_{3a}, F_{3b}, and F_{3c}. In contrast, PTbGA's performance is much more robust with respect to these cost functions.



(a) F_2 on instance No.25. (b) F_{3a} on instance No.27. (c) F_{3b} on instance No.21.

Figure 14: Convergence graphs for PTbGA, PrGA, LINDOGlobal and AlphaECP.

5.5. Efficiency

The convergence speeds of PTbGA on large-sized problem instances as compared to that by PrGA and mathematical solvers are presented in Fig. 14. As shown in Fig. 14a, LINDOGlobal cannot find any feasible solution until about 1200s. Once a feasible solution is found by LINDOGlobal, it cannot be further improved and the convergence curve becomes a straight line (Figs. 14a). Similar results are observed for LINDOGlobal considering functions F_{3a} and F_{3b} (Figs. 14b,c). Although AlphaECP can find a feasible solution faster than LINDOGlobal, the found solution cannot be improved dramatically within the time limit. Unlike the mathematical approach, PTbGA and PrGA are able to converge faster. In particular, PTbGA is able to find better quality solutions than that of PrGA (Fig. 14).

6. Conclusion

In this paper, we have proposed a probabilistic tree-based representation (PTbR) and incorporated it into a genetic algorithm (PTbGA) for solving large-scale MCFP using nonlinear non-convex cost functions. After identifying the drawbacks of the well-known priority-based representation scheme, we have proposed the PTbR by introducing an improved decoding procedure that allows a more thorough sampling of the search space. This new representation scheme PTbR can be executed N times (i.e., decoding N times) to allow for better exploration of the feasible search space of an MCFP. This is in sharp contrast to the limitations imposed by the priority-based representation scheme (PbR). We have conducted systematic and extensive experiments to compare the performance of the PTbR-based GA (PTbGA) variant, the PbR-based GA (PrGA) and two mathematical solver packages for solving MCFP using nonlinear cost functions. In order to identify the optimal parameter settings for both PTbGA and PrGA, the Taguchi method has been applied. PTbGA, PrGA, LINDOGlobal, and AlphaECP have been evaluated on a set of 36 randomly generated MCFP instances considering various nonlinear non-convex cost functions. Our results demonstrate that PTbGA outperforms PrGA on all instances using different cost functions. Unlike mathematical solvers that either got stuck in local optima or cannot find any feasible solutions after 2 hours for large-scale instances, PTbGA can still find high-quality solutions without sacrificing on efficiency. As shown in convergence graphs (Fig. 14), PTbGA converges to a better solution extremely faster than PrGA and mathematical solver packages. Furthermore, PTbGA is more robust to the different shapes of a cost function, as compared to LINDOGlobal and AlphaECP.

References

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, Network flows: theory, algorithms, and applications, Prentice Hall , pp.4–6, 1993 (1993).

[2] D. B. Fontes, J. F. Gonçalves, Heuristic solutions for general concave minimum cost network flow problems, Networks 50 (1) (2007) 67–76 (2007).

[3] M. S. Monteiro, D. B. Fontes, F. A. Fontes, Concave minimum cost network flow problems solved with a colony of ants, Journal of Heuristics 19 (1) (2013) 1–33 (2013).

[4] L. A. Vegh, A strongly polynomial algorithm for a class of minimum-cost flow problems with separable convex objectives, SIAM Journal on Computing 45 (5) (2016) 1729–1761 (2016).

[5] A. Sifaleras, Minimum cost network flows: Problems, algorithms, and software, Yugoslav Journal of Operations Research 23 (1) (2013) 3–17 (2013).

[6] P. Kovács, Minimum-cost flow algorithms: an experimental evaluation, Optimization Methods and Software 30 (1) (2015) 94–127 (2015).

[7] S. Yan, Y. Shih, C. Wang, An ant colony system-based hybrid algorithm for square root concave cost transhipment problems, Engineering Optimization 42 (11) (2010) 983–1001 (2010).

[8] F. Xie, R. Jia, Nonlinear fixed charge transportation problem by minimum cost flow-based genetic algorithm, Computers & Industrial Engineering 63 (4) (2012) 763–778 (2012).

[9] A. S. Amiri, S. A. Torabi, R. Ghodsi, An iterative approach for a bi-level competitive supply chain network design problem under foresight competition and variable coverage, Transportation Research Part E: Logistics and Transportation Review 109 (2018) 99–114 (2018).

[10] A. Trivedi, D. Srinivasan, S. Biswas, T. Reindl, A genetic algorithm–differential evolution based hybrid framework: case study on unit commitment scheduling problem, Information Sciences 354 (2016) 275–300 (2016).

[11] H. R. Bouchekara, A. Chaib, M. A. Abido, R. A. El-Sehiemy, Optimal power flow using an improved colliding bodies optimization algorithm, Applied Soft Computing 42 (2016) 119–131 (2016).

[12] K. Ayan, U. Kılıç, Artificial bee colony algorithm solution for optimal reactive power flow, Applied soft computing 12 (5) (2012) 1477–1482 (2012).

[13] G. M. Guisewite, P. M. Pardalos, Algorithms for the single-source uncapacitated minimum concave-cost network flow problem, Journal of Global Optimization 1 (3) (1991) 245–265 (1991).

[14] D. B. Fontes, E. Hadjiconstantinou, N. Christofides, A dynamic programming approach for solving single-source uncapacitated concave minimum cost network flow problems, European Journal of Operational Research 174 (2) (2006) 1205–1219 (2006).

[15] D. B. Fontes, E. Hadji constantinou, N. Christofides, A branch-and-bound algorithm for concave network flow problems, Journal of Global Optimization 34 (1) (2006) 127–155 (2006).

[16] R. E. Burkard, H. Dollani, P. T. Thach, Linear approximations in a dynamic programming approach for the uncapacitated single-source minimum concave cost network flow problem in acyclic networks, Journal of Global Optimization 19 (2) (2001) 121–139 (2001).

[17] M. S. Monteiro, D. B. Fontes, F. A. Fontes, An ant colony optimization algorithm to solve the minimum cost network flow problem with concave cost functions, in: Proceedings of the 13th annual conference on Genetic and evolutionary computation, ACM, 2011, pp. 139–146 (2011).

[18] Y.-H. Zhang, Y.-J. Gong, T.-L. Gu, Y. Li, J. Zhang, Flexible genetic algorithm: A simple and generic approach to node placement problems, Applied Soft Computing 52 (2017) 457–470 (2017).

[19] G. Aiello, G. La Scalia, M. Enea, A multi objective genetic algorithm for the facility layout problem based upon slicing structure encoding, Expert Systems with Applications 39 (12) (2012) 10352–10358 (2012).

[20] F. G. Tari, Z. Hashemi, A priority based genetic algorithm for nonlinear transportation costs problems, Computers & Industrial Engineering 96 (2016) 86–95 (2016).

[21] B. Ghasemishabankareh, M. Ozlen, X. Li, K. Deb, A genetic algorithm with local search for solving single-source single-sink nonlinear non-convex minimum cost flow problems, Soft Computing (2019) 1–17 (2019).

[22] T. Senjyu, K. Shimabukuro, K. Uezato, T. Funabashi, A fast technique for unit commitment problem by extended priority list, IEEE Transactions on Power Systems 18 (2) (2003) 882–888 (2003).

[23] C. Chung, H. Yu, K. P. Wong, An advanced quantum-inspired evolutionary algorithm for unit commitment, IEEE Transactions on Power Systems 26 (2) (2011) 847–854 (2011).

[24] C. W. Ahn, R. S. Ramakrishna, A genetic algorithm for shortest path routing problem and the sizing of populations, IEEE transactions on evolutionary computation 6 (6) (2002) 566–579 (2002).

[25] N. Shimamoto, A. Hiramatsu, K. Yamasaki, A dynamic routing control based on a genetic algorithm, in: IEEE International conference on neural networks, IEEE, 1993, pp. 1123–1128 (1993).

[26] M. Lotfi, R. Tavakkoli-Moghaddam, A genetic algorithm using priority-based encoding with new operators for fixed charge transportation problems, Applied Soft Computing 13 (5) (2013) 2711–2726 (2013).

[27] J. J. d. M. Mendes, J. F. Gonçalves, M. G. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, Computers & Operations Research 36 (1) (2009) 92–109 (2009).

[28] V. Van Peteghem, M. Vanhoucke, A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem, European Journal of Operational Research 201 (2) (2010) 409–418 (2010).

[29] M. Gen, R. Cheng, L. Lin, Network models and optimization: Multiobjective genetic algorithm approach, Springer Science & Business Media, 2008 (2008).

[30] D. B. Fontes, J. F. Gonçalves, A multi-population hybrid biased random key genetic algorithm for hop-constrained trees in nonlinear cost flow networks, Optimization Letters 7 (6) (2013) 1303–1324 (2013).

[31] C. Dang, Y. Sun, Y. Wang, Y. Yang, A deterministic annealing algorithm for the minimum concave cost network flow problem, Neural networks 24 (7) (2011) 699–708 (2011).

[32] B. Ghasemishabankareh, M. Ozlen, F. Neumann, X. Li, A probabilistic tree-based representation for non-convex minimum cost flow problems, in: Parallel Problem Solving from Nature–PPSN XV: 15th International

Conference, Coimbra, Portugal on 8-12 September 2018, Proceedings, Springer, 2018 (2018).

[33] D. E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of genetic algorithms, Vol. 1, Elsevier, 1991, pp. 69–93 (1991).

[34] J.-E. Lee, M. Gen, K.-G. Rhee, Network model and optimization of reverse logistics by hybrid genetic algorithm, Computers & Industrial Engineering 56 (3) (2009) 951–964 (2009).

[35] U. Klanšek, Solving the nonlinear discrete transportation problem by minlp optimization, Transport 29 (1) (2014) 1–11 (2014).

[36] U. Klanšek, M. Pšunder, Solving the nonlinear transportation problem by global optimization, Transport 25 (3) (2010) 314–324 (2010).

[37] Z. Michalewicz, G. A. Vignaux, M. Hobbs, A nonstandard genetic algorithm for the nonlinear transportation problem, ORSA Journal on Computing 3 (4) (1991) 307–316 (1991).

[38] S. Burer, A. N. Letchford, Non-convex mixed-integer nonlinear programming: a survey, Surveys in Operations Research and Management Science 17 (2) (2012) 97–106 (2012).

[39] Y. Lin, L. Schrage, The global solver in the lindo api, Optimization Methods & Software 24 (4-5) (2009) 657–668 (2009).

[40] T. Westerlund, R. Pörn, Solving pseudo-convex mixed integer optimization problems by cutting plane techniques, Optimization and Engineering 3 (3) (2002) 253–280 (2002).

[41] M. Tawarmalani, N. V. Sahinidis, A polyhedral branch-and-cut approach to global optimization, Mathematical Programming 103 (2005) 225–249 (2005).

[42] R. Al-Aomar, et al., A ga-based parameter design for single machine turning process with high-volume production, Computers & Industrial Engineering 50 (3) (2006) 317–337 (2006).

[43] F. Jabbarizadeh, M. Zandieh, D. Talebi, Hybrid flexible flowshops with sequence-dependent setup times and machine availability constraints, Computers & Industrial Engineering 57 (3) (2009) 949–957 (2009).

[44] B. Ghasemishabankareh, N. Shahsavari-Pour, M.-A. Basiri, X. Li, A hybrid imperialist competitive algorithm for the flexible job shop problem, in: Australasian Conference on Artificial Life and Computational Intelligence, Springer, 2016, pp. 221–233 (2016).

[45] M. Hajiaghaei-Keshteli, S. Molla-Alizadeh-Zavardehi, R. Tavakkoli-Moghaddam, Addressing a nonlinear fixed-charge transportation problem using a spanning tree-based genetic algorithm, Computers & Industrial Engineering 59 (2) (2010) 259–271 (2010).

[46] R. Al-Aomar, Incorporating robustness into genetic algorithm search of stochastic simulation outputs, Simulation Modelling Practice and Theory 14 (3) (2006) 201–223 (2006).

[47] M. S. Phadke, Quality engineering using robust design, Prentice Hall PTR, 1995 (1995).

[48] J.-T. Tsai, W.-H. Ho, T.-K. Liu, J.-H. Chou, Improved immune algorithm for global numerical optimization and job-shop scheduling problems, Applied Mathematics and Computation 194 (2) (2007) 406–424 (2007).

[49] B. Naderi, M. Zandieh, A. K. G. Balagh, V. Roshanaei, An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness, Expert systems with Applications 36 (6) (2009) 9625–9633 (2009).

[50] S. Bolboacă, L. Jäntschi, Design of experiments: Useful orthogonal arrays for number of experiments from 4 to 16, Entropy 9 (4) (2007) 198–232 (2007).

[51] S. Molla-Alizadeh-Zavardehi, M. Hajiaghaei-Keshteli, R. Tavakkoli-Moghaddam, Solving a capacitated fixed-charge transportation problem by artificial immune and genetic algorithms with a prüfer number representation, Expert Systems with Applications 38 (8) (2011) 10462–10474 (2011).