Multifactorial Evolutionary Algorithm for Inter-Domain Path Computation under Domain Uniqueness Constraint

Huynh Thi Thanh Binh[†], Ta Bao Thang[†], Nguyen Binh Long[†], Ngo Viet Hoang[†], Pham Dinh Thanh[*],
[†] School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam
[*] Faculty of Natural Science and Technology, Tay Bac University, Vietnam
Email: binhht@soict.hust.edu.vn, {tabaothang97, binhlongmm99, ngoviethoang735, thanhpd05}@gmail.com

Abstract*—Nowadays, connectivity among communication devices in networks has been playing a significant role, especially when the number of devices is increasing dramatically that requires network service providers to have a better architecture of management system. One of the popular approach is to divide those devices inside a network into different domains, in which the problem of minimizing path computation in general or Inter-Domain Path Computation under Domain Uniqueness constraint (IDPC-DU) problem in specific has received much attention from the research community. Since the IDPC-DU is NP-complete, an approximate approach is usually taken to tackle this problem when the dimensionality is high. Although Multifactorial Evolutionary Algorithm (MFEA) has emerged as an effective approximation algorithm to deal with various fields of problems, there are still some difficulties to apply directly MFEA to solve the IDPC-DU problem, i.e. different chromosomes may have different numbers of genes or to construct a feasible solution not violating the problem's constraint. Therefore, to overcome these limitations, MFEA algorithm with a new solution representation based on Priority-based Encoding is introduced. With the new representation of the solution, a chromosome consists of two parts: the first part encodes the priority of the vertex while the second part encodes information of edges in the solution. Besides, the paper also proposed a corresponding decoding method as well as novel crossover and mutation operators. Those evolutionary operators always produce valid solutions. For examining the efficiency of the proposed MFEA, experiments on a wide range of test sets of instances were implemented and the results pointed out the effectiveness of the proposed algorithm. Finally, the characteristics of the proposed algorithm are also indicated and carefully analyzed.

***Index Terms*—Multifactorial Evolutionary Algorithm, Inter-Domain Path Computation under Domain Uniqueness constraint, Path Computation Element, Evolutionary Algorithm.**

I. Introduction

In the era of rapid technological development as nowadays, numerous communication devices are required to be connected to each others via a network. This demanding makes the network system to become significantly larger and should be partitioned into different domains for a better management. As a consequence, multi-domain networks have been designed to help resolving the scalability issues, since this network type is capable of minimizing the routing signals. In addition, privacy issues are also guaranteed in the multi-domain networks when sensitive intra-domain information is not revealed to neighboring domains of the competing network operators.

In some recent studies on the multi-domain networks, L. Maggi et al [1] introduced the problem of IDPC-DU. The problem addresses the multi-domain network with hierarchical Path Computation Element (h-PCE) architecture and focuses on finding an inter-domain path with minimum cost that crosses a domain at most once. The authors have proven that the IDPC-DU problem is in NP-complete class.

After suggesting that the computational bottleneck for IDPC-DU lies in the number of domains in [1], a dynamic programming approach which based on Bellman- Ford algorithm has been proposed with the complexity of $O(|V|^2 2^{|D|}|D|^2)$ to tackle the problem in combination with a novel domain clustering concept which helps to artificially reduce the number of domains.

However, as having been proved as NP-hard, the IDPC-DU problem with a large number of dimensionality can be solved more effectively by an approximation algorithm. One of the most popular approximation algorithm that has been widely studied in the scientific community is the Evolutionary Algorithm (EA) [2, 3]. Despite various advantages of EA, its weaknesses lie in the large consumption of resources and the resolution of no more than one problem at a time.

To overcome these limitations, the MFEA [4, 5] has been recently proposed and is emerging as one of the most effective variants of EA. MFEA can solve multiple independent optimization problems simultaneously using a unique representation of population in the Unified Search Space (USS). Thanks to the sharing genetic materials of individuals from multiple tasks in the USS, exchanging information among different problems inside MFEA can help to obtain the optimal solutions for each task.

Despite such advantages of the algorithm, to the best of our knowledge, there has not been any attempt at applying directly the MFEA to tackle with the IDPC-DU. Some of the difficulties can be mentioned as the MFEA has not been designed for path representation on multi-graph or to construct a solution not to violate the Domain Uniqueness (DU) constraint of the problem.

Consequently, this paper introduces a new approach based on the MFEA to solve the IDPC-DU problem. The major contributions of this work are as follows:

- Propose an effective way to encode a feasible path of the

IDPC-DU problem in USS and a corresponding decoding mechanism.

- Propose evolutionary operators for applying MFEA to solve the IDPC-DU problem.
- Analyze and evaluate experimental results on multiple test instances to show the efficiency of the proposed algorithm.

The rest of this paper is organized as follows. Section II presents the notations and definitions used for formulating problem. Section III introduces related works, while the proposed MFEA for the IDPC-DU is elaborated in section IV. Section V explains the setup of our experiments and reports the computed results. The paper concludes in section VI with discussions on the future extension of this research.

II. Notation and definitions

The IDPC-DU problem can be stated as follows: Given a finite set of colors D and a weighted colored directed multi-graph $G = (V, E, w)$, where V is the set of nodes, E is the set of edges and a weight function $w : E \mapsto R$ assigns to each edge $e \in E$ a positive cost $w(e) \geq 0$. Let $(i, j)^k \in E$ be the k^{th} parallel edge between nodes i and j which has a weight $w^k_{i,j}$ and is assigned with a color $d \in D$. Let E^d be a domain of the graph which contains all edges having color d. Therefore, the color of each edge will determine exactly one network domain in which the edge belongs to. An edge which points from node a to node b with an associated weight w and colored by the color c, is denoted by (a, b, w, c).

Under the DU constraint, a path p on G traverses every domain at most once. In other words, once p has left a domain E^d, it does not revisit it later on. The IDPC-DU looks for a path p^* from source node s to destination node t such that:

1) The path p^* satisfies the DU constraint.
2) The sum of weights of all edges on path p^* is minimized.

$$p^* = argmin_{p \in F_{s,t}(G)} \sum_{(i,j)^k \in p} w^k_{i,j} \quad (1)$$

where $F_{s,t}(G)$ is the set of all feasible paths on graph G from node s to node t.

Figure 1 depicts a weighted colored directed multi-graph with 9 nodes (labeled from 1 through 9) and the set of edges is partitioned into three domains $(red, blue, black)$. The IDPC-DU problem searches for a path from source node 1 to destination node 9 on which the sum of weight of edges is minimized while the DU constraint must be obeyed. Path $\{(1, 2, 1, blue), (2, 7, 1, red), (7, 8, 1, black), (8, 9, 1, red)\}$ has the minimum cost but is infeasible due to entering the red domain twice, so it violates the DU constraint. A feasible path satisfying this constraint is $\{(1, 2, 1, blue), (2, 7, 1, red), (7, 3, 1, black), (3, 9, 1, black)\}$.

For a better convenience of describing our proposed algorithm in this paper, the set of edges leaving a node is called the out-edge set in which its size is the out-degree of the node. The out-edge set of a node i is denoted by $E_{i,out}$. For example, $E_{1,out}$ = $\{(1, 2, 1, blue), (1, 2, 2, black), (1, 2, 3, red), (1, 5, 5, blue), (1, 5, 1, black), (1, 6, 5, black)\}$. In addition, the subset of $E_{i,out}$ which has edges from the node i to the node j is denoted by $SE_{i,j}$. In Figure 1, $SE_{1,5} = \{(1, 5, 5, blue), (1, 5, 1, black)\}$. Finally, the maximum size of $SE_{i,j}$ is denoted by S_i, with $S_i = max_{j \in V}\{|SE_{i,j}|\}$. For example in Figure 1, $S_1 = max_{j \in V}\{|SE_{1,j}|\} = |SE_{1,2}| = 3$.

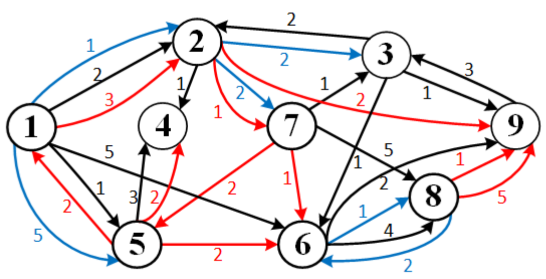

Fig. 1: An example of a weighted colored directed multi-graph

III. Related works

Along with the increasing requirements posed by significant advances in networking, path computation has become an important factor that service providers need to pay special attention to [6, 7]. In general, an optimal solution to the path computation problem with minimum cost under constraints can be obtained simply by pruning links that do not satisfy constraints and applying the shortest path algorithm on the resulting subgraph. However, this approach can be inefficient when multiple domains are involved in the network. To address this problem, the Path Computation Element (PCE) architecture was introduced, enabling one to compute the best possible path through multiple domains in the network. In [8], Sukrit Dasgupta et al presented a brief review on the significant developments of the PCE architecture and demonstrated the superior performance of the PCE compared to the per-domain approach when addressing the path computation problem.

With the development of the PCE architecture, inter-domain networks have drawn great interest from the research community over the last decade. As mentioned above, the path computation inside each domain (intra-domain) is controlled by a PCE in multi-domain networks. For handling communication among different PCEs, two architectures, which are distributed and hierarchical, are primarily envisaged. In particular, a good investigation on recent advances in path computing for distributed and hierachical architectures of PCE is provided in [9].

Several approaches has been introduced to tackle the problem of path computation in the inter-domain networks. In [10], L. Buzzi et al proposed a solution based on a combination of hierarchical routing and path computation procedures. The former method determines the sequence of domain to cross while the latter calculates the end-to-end path. Another approach to solve the path computation problem, which was proposed by D. Siracusa et al [11], is to use a Domain Sequence

Protocol. The Domain Sequence Protocol technique helps to compute the sequence of traversed domains in hierarchical PCE architectures, from that, an optimal path in multi-domain networks can be obtained.

When applying an EA to the path computation problem, one of the most difficult and essential task is how to encode a path in a graph intro a chromosome. The primary reasons to explain for this difficulty is that: a) a path does not contain a fixed number of nodes, since this value varies with the maximum number is $n-1$ for a graph of n nodes, and b) a random sequence of edges usually does not correspond to a path. In [12], M.Gen et al proposed a priority-based encoding method, which fortunately capable of representing all most every possible paths in the graph. In this encoding technique, the authors give each node a priority and add into the path the one with highest priority.

In recent literature, MFEA [4, 13] has been emerging as an effective framework that can solve a wide range of optimization problems. In [14], R. Chandra et al applied the MFEA to train neural networks with the required tasks are neural network topologies with different number of hidden neurons. Recently, the MFEA has also been applied to solve some problems related to path cost minimization on graph, one of which is the Clustered Shortest-Path Tree Problem (CluSPT) [15]. In applying the MFEA to tackle this problem, P.D. Thanh et al [16] took the advantages of Cayley Code to encode a solution of CluSPT into the USS and proposed new evolutionary operators based on this type of encoding.

Although the MFEA has been developed for solving various optimization problem, to the best of our knowledge, there has not been any prior attempt to apply the MFEA to solve the problem of IDPC-DU, that is, to find a minimum cost path that crossed a domain at most one in the multi-domain network. Hence, this paper focuses on proposing effective evolutionary operators and a novel method of representing a path in IDPC-DU in order to apply the MFEA for addressing the problem.

IV. Proposed Algorithm

MFEA is used to find the solutions of K IDPC-DU problems $G^i = (V^i, E^i, w^i, D^i)$, $i = 1, \dots, K$ where V^i, E^i, w^i and D^i are set of vertices (nodes), set of edges, weight matrix and set of domains of graph, respectively.

A. *Encoding path in IDPC-DU graph*

In the routing problem, representing a valid path is a critical and relatively complex problem. A good coding strategy will provide many benefits for solving complicated multi-constraint problems. One traditional approach of encoding a path between two vertices s and t in the graph is to represent it simply as a list of vertices or edges that passes from s to t. However, with this approach, the length of the path is not fixed since many potential nodes can be selected and appears as intermediate nodes in solution path. Therefore, the evolutionary operators can be very complex or ineffective when implementing on this encoding, especially in incomplete graphs. Moreover, because the path is randomly generated, constraints are easy to be violated and may be uncontrollable. Therefore, the obtained path may not terminate at the destination node, leading to an invalid path.

Fortunately, there is another approach to encrypt the path between two vertices, which is Priority-based Encoding. A path is now represented by an array of integers of fixed size N, where N is the total number of nodes in the graph. The value of each element in the array is the priority of that node in the graph. The higher the priority, the earlier the node is visited and then, the order of nodes on the path from $s \to t$ can be interpreted as the order of priority of those nodes in the graph. Every possible path in the graph can be represented by this encoding, and this strategy has been applied in its basic form to show its effectiveness in solving routing path problems in simple undirected graph [12]. In this work, to represent a solution for the IDPC-DU problem, a new solution representation (NSR) based on the priority-based encoding is introduced for the directed and multi-constrained graph, which takes the advantages of the flexibility of this encoding method.

Because input graph of the IDPC-DU problem is a multi-graph in which 2 nodes may be connected by many edges, if only using the Priority-based Encoding to represent a path, a given gene can only be decoded into a sequence of nodes that the path passes through. Consequently, edges connecting those nodes are not yet determined since this encoding can associate with many paths. Therefore, we propose a many-to-one path encoding consisting of two parts as follows:

- **Node Priority**: An array π has N elements where N is the number of nodes in the graph and the value of i^{th} element presents the priority of the i^{th} node. The higher the value of a node is, the more priority that this node is passed through. An example of the priority of nodes in the input graph G as in Figure 1 is shown in the white array of Figure 2. According to this priority, a path from node 1 to node 9 on the input graph G is $\{1, 2, 7, 3, 9\}$.
- **Edge Index**: An array σ also has N elements where N is the number of nodes in the input graph. The value of the i^{th} element in the array represents the index of the edge in the out-edge set $E_{i,out}$ that the path will take to reach another node. The blue array in Figure 2 describes an example for this representation. In this example, the path will leave node 1 by the 1^{st} out-edge $(1, 2, 1, blue)$, leave node 2 by the 3^{rd} out-edge $(2, 7, 1, red)$, etc. A special case in this example is node 4 that has no edge coming out of it, so the value of the fourth element in the array is '-'.

A complete representation of the path is illustrated in Figure 2.

1	2	3	4	5	6	7	8	9	Position: **Node ID**
9	8	6	4	2	3	7	1	5	Value: **Priority**
1	3	2	-	2	2	1	1	1	Value: **Out-Edge Index**

Fig. 2: An representation of the path in graph

For each representation, the path from node s to node t

will be constructed by visiting the nodes in their orders of priority according to a proposed Growing Path Algorithm (GPA). The basic idea of the GPA is that from the node i, the algorithm visits a node j having the highest priority by the k^{th} edge between them with k is an out-edge index in the subset $SE_{i,j}$ that satisfies the DU constraint. With this strategy, each encoding corresponds to a path and every possible single path in a graph can be represented by at least an encoding.

Details of the GPA are described in Algorithm 1. For example, from the encoding illustrated in Figure 2, a corresponding path from node 1 to 9 of the given graph in Figure 1 can be obtained, which is $(1, 2, 1, blue), (2, 7, 1, red), (7, 3, 1, black)$ and $(3, 9, 1, black)$ with a total cost of 4.

Algorithm 1: Growing Path Algorithm

Input: A multi-graph $G = (V, E, w)$, $E = \{E^1, \ldots, E^D\}$; A source node s; A destination node t; an individual $I = (\pi_1, \pi_2, \ldots, \pi_N | \sigma_1, \sigma_2, \ldots, \sigma_N)$.
Output: A path p and its cost

```
1  begin
2      $H \leftarrow \emptyset$      ▷ Set of domains that path $p$ has left;
3      $d \leftarrow$ -1   ▷ The domain that the path $p$ is visiting;
       /* $c(v)$ is cost of path from $s$ to $v$    */
4      $c(s) \leftarrow 0$; $c(t) \leftarrow \infty$;
5      $curr \leftarrow s$    ▷ $curr$ is the node that $p$ is visiting;
6      $visited[v] \leftarrow$ false $\forall v \in V$; $p \leftarrow \emptyset$;
7      while $h \neq t$ do
8          $visited[h] \leftarrow$ true;
9          $Adj(h) \leftarrow$ the set of edges that connect $h$ to
             other unvisited nodes in $G$, such that each
             edge's domain is not in $H$;
10         if $Adj(h) = \emptyset$ then Break ;
11         $v \leftarrow$ the node has the highest priority $\pi_v$ and
             connects to $h$ by an edge in $Adj(h)$;
12         $E_{h,v} \leftarrow$ the set of edges connecting two nodes
             $h$ and $v$ and belong to $Adj(h)$;
13         $k \leftarrow \sigma_v \ \% \ |E_{h,v}| + 1$;
14         $e \leftarrow$ the $k^{th}$ edge in $E_{h,v}$; $p.append(e)$;
15         $d' \leftarrow$ the domain of $e$;
16         if $d \neq -1$ and $d' \neq d$ then H $\leftarrow$ H $\cup$ {d} ;
17         $d \leftarrow d'$; $c(v) \leftarrow c(h) + w(e)$; $h \leftarrow v$;
18     return $p$ and $c(t)$;
```

B. Individual Representation in Unified Search Space

In MFEA, the representations of individual of each task are combined into a common representation and evolutionary operators are performed on it.

The USS for tasks is a graph which is constructed as follows:

- The number of domains D in the graph is the largest number of domains in all tasks, $D = max(|D^1|, |D^2|, \ldots, |D^K|)$ where $|D^i|$ is the number of domains of the i^{th} task.
- The number of nodes in the graph is the largest number of nodes in all tasks. $N = max(N^1, N^2, \ldots, N^K)$ where N^i is the number of nodes of the i^{th} task.
- The maximum number of out-edges from the node i to another node is the maximum of those in all tasks. $S_i = max(S_i^1, S_i^2, \ldots, S_i^K)$ with S_i^j is the maximum number of out-edge from the node i to another node in the j^{th} task.

An individual $I = (\pi_1, \pi_2, \ldots, \pi_N | \sigma_1, \sigma_2, \ldots, \sigma_N)$ in USS consists of two parts and each part is an array of N elements. The first part contains values representing the priority of the corresponding nodes in the graph. In the second part, the i^{th} element in the array is the index of edge that the path leaves node i in the graph.

C. Individual Initialization Method

With the desire to represent all possible paths between two vertices s and t of the graph and to be able to explore all possible solutions in the search space, the priority and the out-edge indexes of all nodes in graph are generated randomly. Let the representation of an individual in the USS be:

$$I = (\pi_1, \pi_2, \ldots, \pi_N | \sigma_1, \sigma_2, \ldots, \sigma_N)$$

where $\{\pi_1, \pi_2, ..., \pi_N\}$ is a permutation of $(1, 2, ..., N)$ and each element in the second half part is a random number from 1 to the maximum number of out edges of the nodes respectively. The initialization method is described in Algorithm 2.

Algorithm 2: Individual initialization method

Input: The number of nodes N and a set $\{S_1, S_2, \ldots, S_N\}$
Output: An individual in unified search space

```
1  begin
2      $\pi \leftarrow$ Shuffle the array of $\{1, 2, \ldots, N\}$ randomly;
3      for $i \leftarrow 1$ to $N$ do
4          if $S_i = 0$ then  $\sigma_i \leftarrow$ '-' ;
5          else  $\sigma_i \leftarrow$ a random number from 1 to $S_i$ ;
6      I $\leftarrow \{\pi \mid \sigma\}$;
7      return I;
```

D. Crossover Operator

This paper describes a crossover operator based on the Partially Mapped Crossover (PMX) [17] and Two-Point Crossover (TPX) [18] operators, in which the PMX is used in the first part while TPX is used in the other part of the individual.

In the first part, the PMX is applied since this operator is efficient for the permutation representation and tends to maintain good characteristics of important similarities about the position and the order of elements from the parent to the child generation. In the other part, we choose to perform the TPX because each element has a specific upper bound S_i leading to swapping values between elements is inappropriate. Proposed method is detailed in Algorithm 3, and examples to describe how these operators work in parent's chromosomes are illustrated in Figure 3 and Figure 4.

Figure 3 illustrates how PMX works in the Node-Priority Segment. First, it selects uniformly at random 2 cut points which partition the parents' Node-Priority Segment into 3 sections (left, middle and right). The middle section between

Algorithm 3: Proposed Crossover operator

```
Input: Two parents $I_1 = (\pi^1|\sigma^1)$ and $I_2 = (\pi^2|\sigma^2)$;
Output: Two Offsprings $O_1, O_2$;
1 begin
2     $\pi^{1*}, \pi^{2*} \leftarrow PMX\_Crossover(\pi^1, \pi^2)$;
3     $\sigma^{1*}, \sigma^{2*} \leftarrow Two\_Point\_Crossover(\sigma^1, \sigma^2)$;
4     $O_1 \leftarrow \{\pi^{1*}|\sigma^{1*}\}$; $O_2 \leftarrow \{\pi^{2*}|\sigma^{2*}\}$;
5     return $O_1, O_2$;
```

these cut points define the mappings (i.e. $4 \leftrightarrow 3, 2 \leftrightarrow 8$ and $3 \leftrightarrow 5$), and elements in this section of a parent are copied directly into the corresponding offspring. Then, the remaining of this offspring is filled up by copying elements of the other parent. If an element is already present in the offspring, it is replaced according to those mappings defined in the middle section. For example, in the 2^{nd} position, since value 8 in Parent 1 has appeared in the middle section of Offspring 2 and there exists a mapping $8 \leftrightarrow 2$, the 2^{nd} element of Offspring 2 is chosen to be 2. Analogously, we find offspring as shown.

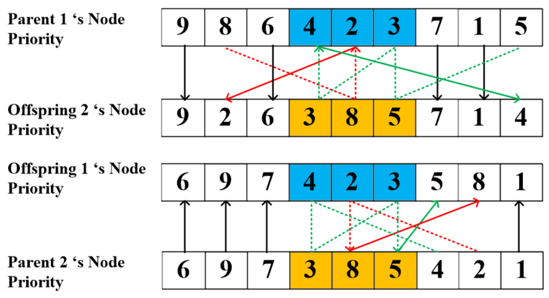
Parent 1 's Node Priority
9 8 6 4 2 3 7 1 5
Offspring 2 's Node Priority
9 2 6 3 8 5 7 1 4
Offspring 1 's Node Priority
6 9 7 4 2 3 5 8 1
Parent 2 's Node Priority
6 9 7 3 8 5 4 2 1

Fig. 3: An example describes PMX Crossover Operator works in the Node-Priority Segment.

Figure 4 presents how Two-point crossover operator works in the Edge-Index Segment. Two cutting-points are chosen randomly in the segment and the offsprings are built by swapping the section between these cut points.

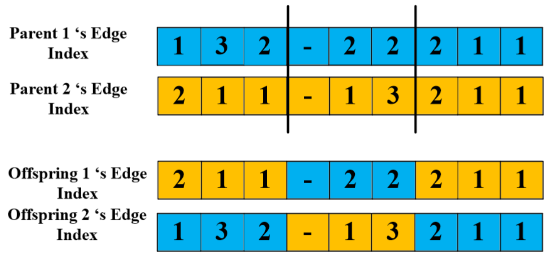

Fig. 4: An example describes how Two-Point Crossover Operator works in the Edge-Index Segment.

E. Mutation Operator

Similar to the crossover operator, the mutation operator is also performed on both segments of genes, in which

- Swap Mutation operator [17] in the first segment with the idea of randomly selecting two positions and swapping the values of those two positions for each other;
- One-Point mutation in the rest segment by choosing randomly a position and changing its value.

Details of the proposed method is shown in Algorithm 4 with an example of a mutation in genes in Figure 5. In this example, the priority values of the 3^{rd} and 6^{th} nodes are swapped, and the out-edge index in the 7^{th} position is changed from 1 to 2.

Algorithm 4: Proposed Mutation operator

```
Input: An individual $I = (\pi|\sigma)$; a set $\{S_1, S_2, ..., S_N\}$
Output: A new individual $I^* = (\pi^*|\sigma^*)$
1 begin
2     $\pi^* \leftarrow \pi$; $\sigma^* \leftarrow \sigma$;
3     $i, j \leftarrow$ Choose two random positions in $\pi$;
4     Swap($\pi_i^*$, $\pi_j^*$);
5     $i \leftarrow$ Choose a random position in $\sigma$;
6     $\sigma_i^* \leftarrow$ Random a number from 1 to $S_i$;
7     return $I^*$;
```

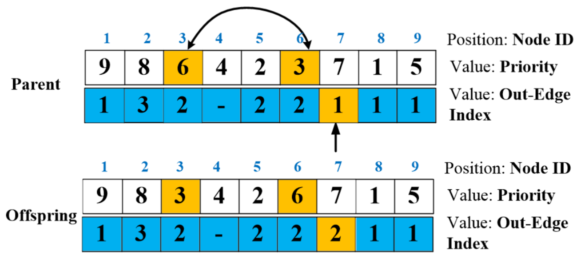

Fig. 5: An example illustrates Mutation Operator on a representation.

F. Decoding method

A representation associated with a task T_j (the number of nodes N_j) built from a unified representation I = (π | σ) is denoted by $I^j = (\pi^j|\sigma^j)$. The first segment (π^j) is built by just choosing all integers that are not larger than N_j from π ($\pi = \{\pi_1, \pi_2, ..., \pi_N\}$) and keeping them in the same relative order. The other segment is constructed by getting the first N_j elements of the corresponding array $\sigma^j = \{\sigma_1, \sigma_2, ..., \sigma_{N_j}\}$.

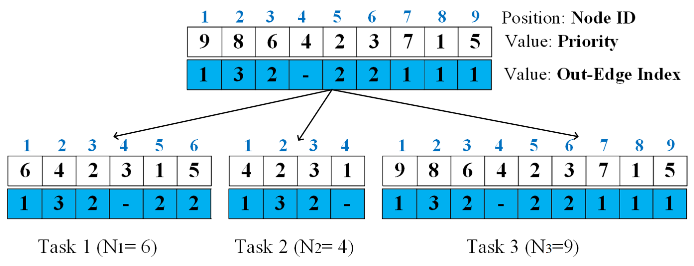

Fig. 6: An example of decoding a specific representation of each Task from the unified representation.

An example of the decoding method is shown in Figure 6 with the number of nodes of individual in USS, in Task 1, Task 2 and Task 3 are 9, 6, 4 and 9 respectively.

V. Computational results

A. Problem instances

To evaluate the proposed algorithm, on account of no instances were available for IDPC-DU, two sets of test instances are generated. To generate a test instance, we first passed three parameters: number of nodes, number of domains and number of edges. After that, we created an array of distinct nodes and an array of distinct domains which satisfy a condition that the number of nodes is greater than the length of domain array. Source node and terminal node are the first and the last nodes in the node array, respectively. From the above arrays, we merged them to construct a valid solution path called p. Each edge of p was assigned to weight 1, except for the out-edge of the source node which is set to weight 2. To add noise to the test instance, for every single node in p, we randomly added several edges from that node to some other nodes not in p. Moreover, some one-weight edges are created between nodes not in the solution path. These traps would make simple greedy algorithms harder to find optimal solution. Eventually, we randomly generated edges with greater values of weight than those of edges in p. This method guaranteed that p is the optimal solution of the instance.

There are two sets of instances created, a small set (the number of nodes varies from 10 to 45 with an increment of 5) and a large set (the number of node varies from 50 to 100 with an increment of 10). The number of nodes is then combined with the domain size by ratio of 0.5, 1.0 and 2.0. All instances are available in [19].

B. Experimental criteria

Criterias for assessing the quality of the output of the algorithms are presented in Table I.

TABLE I: Criterias for assessing the quality of the output of the algorithms

Avg	The average function value over all runs.
BF	The best function value achieved over all runs.
NIB	Number of instances on which multitasking outperformed single-tasking.
NIE	Number of instances with Avg value of multitasking equals to that of single-tasking
RPD	The difference between the average costs of two algorithms [20]

C. Experimental setup

To evaluate the performance of new MFEA for the IDPC-DU, two sets of experiments are implemented.

- On the first set, two algorithms including single-task (traditional EA) and multi-task (proposed MFEA) were implemented.
- On the second set, we conducted an experiment (C-experiment) for evaluating the effect of the parameters on the convergence trends of each task in generations.

Each scenario was simulated for 30 times on the computer (Intel Core i7 - 3.60GHz, 16GB RAM), with a population size of 100 individuals evolving through 500 generations, the rmp is 0.5 and the mutation rate is 0.05. The source codes were installed by Java language.

D. Experimental results

1) Comparison between the performances of multi-task and single task:

The experimental results show that proposed multitasking exceeded single-tasking on most instances. Particularly, results obtained by multitasking are better than those obtained by single-task on 17 of 18 instances in Set 1. Table II presents the achieved results of 2 test sets in five fields, in which the field *NIB* indicates the number of instances on which average results found by multi-task outperform those found by single-task, while *NIE* shows how many instances in the group have the average results obtained by multi-task which are equal to those of single-task. A highlight indicated in Table II is that the average *RPD(multi-task, single task)* of instances in Set 1 and Set 2 are 13.2% and 4.6% respectively.

TABLE II: Comparison of results obtained by multi-task and single-task

Sets	Number of instances in a Set	NIB	NIE	Maximum RPD	Average RPD
Set 1	24	18	6	41.7	13.2
Set 2	18	17	1	15.8	4.6

The detail of comparison between results obtained by multitasking and single-tasking is presented in the Table III and Table IV. In these tables, italic, red cells on the *Avg* column of a particular algorithm mean that on those instances, that algorithm outperforms the other one. Experimental results on these tables show that the output produced by multi-task on Set 1 was better than that of single-task on 18 of 24 instances. The biggest gap recorded in this group was 41.7% on instance *Idpc_10x5x425* and the instance with the smallest difference was instance *Idpc_30x15x10025* on which multi-task performed better than single-task only 0.2%. Regarding Set 2, the results also showed that multi-task surpassed single-task on 17 of 18 instances. The gap percentage on instances of Set 2 varied between 1.6% (on instance *Idpc_100x50x461319*) and 2.7% (on instance *Idpc_60x120x434337*).

2) C-experiment:

To compare the convergence trends of objective functions between single-tasking and multitasking, we use the functions in [4] for computing the normalized objectives and averaged normalized objectives.

Figure 7 depicts the average convergence trends of the single-tasking (ST) and multitasking (MT) of instances *Idpc_100x100x1000000* and *Idpc_50x100x285357*. The figure shows that multitasking outperformed single-tasking in which multitasking converges faster than single-tasking on all generations.

Refer to Figure 8 for a better understanding of the improved performance as a consequence of multitasking and single-tasking. The figure depicts the convergence trends correspond-

TABLE III: Results obtained by algorithms in Set 1

	EA			MFEA		
Instances	BF	Avg	Rs	BF	Avg	Rs
Idpc_10x10x1000	13	13	1	7	*12.2*	1
Idpc_10x20x2713	12	12	1	12	12	1
Idpc_10x5x425	12	12	1	7	*7*	1
Idpc_15x15x3375	16	16	1	10	*10*	1
Idpc_15x30x12111	16	16	1	10	*10*	1
Idpc_15x7x1504	18	18	1	18	18	1
Idpc_20x10x2492	22	23.5	1	22	*22*	1
Idpc_20x20x8000	21	21	1	15	*15.4*	1
Idpc_20x40x26104	21	21	1	15	*15.2*	3
Idpc_25x12x4817	27	27.9	1	27	*27*	1
Idpc_25x25x15625	26	26	1	18	*24.1*	1
Idpc_25x50x57147	26	26	1	18	*24.7*	3
Idpc_30x15x10025	33	33.1	1	33	*33*	2
Idpc_30x30x27000	33	34	1	32	*32*	6
Idpc_30x60x89772	32	32	2	32	32	8
Idpc_35x17x13934	38	44.5	1	38	*38*	2
Idpc_35x35x42875	39	39	1	37	*37*	6
Idpc_35x70x123585	36	36	3	28	*35.5*	8
Idpc_40x20x18485	42	45.7	1	42	*42*	4
Idpc_40x40x64000	42	42.1	2	42	*42*	10
Idpc_40x80x130681	42	42	3	42	42	21
Idpc_45x22x43769	48	49	1	47	*47*	4
Idpc_45x45x91125	46	46	3	46	46	10
Idpc_45x90x322081	46	46	7	46	46	21

Rs: Running time of algorithms in seconds

TABLE IV: Results obtained by algorithms in Set 2

	EA			MFEA		
Instances	BF	Avg	Rs	BF	Avg	Rs
Idpc_100x100x1000000	103	104.2	21	102	*102*	114
Idpc_100x200x2296097	101	105.9	38	101	*101*	50
Idpc_100x50x461319	115	121.2	9	102	*102.1*	56
Idpc_50x100x285357	52	52.6	7	52	*52*	42
Idpc_50x25x38961	53	60.8	1	53	*53*	11
Idpc_50x50x125000	53	53	4	52	*52*	29
Idpc_60x120x434337	61	61.2	11	61	*61*	42
Idpc_60x30x99470	64	68.8	2	62	*62*	11
Idpc_60x60x216000	63	63.3	5	62	*62*	29
Idpc_70x140x923343	73	73.7	20	72	*72*	90
Idpc_70x35x120810	72	72	3	72	72	23
Idpc_70x70x343000	71	72.2	6	71	*71*	40
Idpc_80x160x1490468	81	81.4	24	81	*81*	90
Idpc_80x40x175762	82	85.8	4	82	*82*	23
Idpc_80x80x512000	85	88.4	10	82	*82*	40
Idpc_90x180x1644367	91	92.1	33	91	*91*	50
Idpc_90x45x260195	95	102	6	93	*93.2*	56
Idpc_90x90x729000	91	91.7	13	91	*91*	114

Rs: Running time of algorithms in seconds

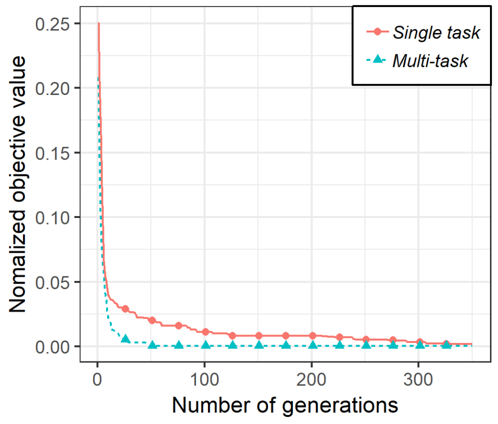

Fig. 7: Convergence trends of $\tilde{f}$ in multi-task and serial single task for instances Idpc_100x100x1000000 and Idpc_50x100x285357

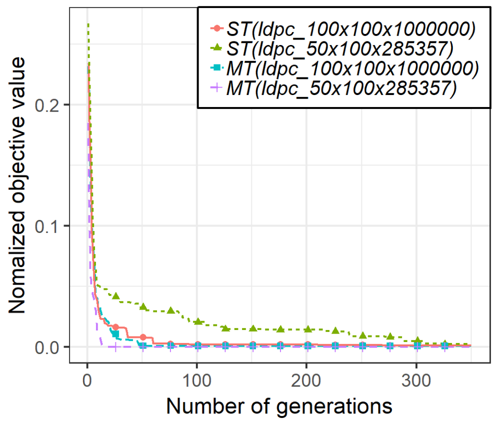

Fig. 8: Comparing convergence trends of $\tilde{f}_1$ and $\tilde{f}_2$ in multi-tasks and serial single task for instances *Idpc_100x100x1000000* and *Idpc_50x100x285357*

ing to each individual task. According to Figure 8, the convergence rate of multi-task on instance *Idpc_100x100x1000000* was slightly slower in comparison with single-task. An explanation for this phenomenon is that evolutionary multitasking does not necessarily guarantee improved performance for every task [21] in the MFEA. Some tasks are positively impacted, while other tasks may be negatively impacted by the implicit genetic transfer available during multitasking. In contrast, on instance *Idpc_50x100x285357*, the convergence rate of each task in multi-task is much faster than that of the corresponding task in single-task.

VI. Conclusion

This paper introduces an algorithm based on MFEA to deal with the IDPC-DU problem. In the approach, a solution representation based on an improvement of the Priority-based Encoding is proposed. The new encoding method can present IDPC-DU solutions with different lengths by individuals of fixed sizes and always guarantees to construct a valid solution when the decoding operator is applied. Furthermore, the paper proposes novel crossover and mutation operators corresponding to the proposed representation method. The proposed algorithm is evaluated on 2 different data sets, and experimental results show that the algorithm performed well in most instances, especially those instances with large dimensionalities.

Acknowledgment

This research was sponsored by the U.S. Army Combat Capabilities Development Command (CCDC) Pacific and CCDC Army Research Laboratory (ARL) under Contract Number W90GQZ-93290007. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the CCDC Pacific and CCDC ARL and the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] L. Maggi, J. Leguay, J. Cohen, and P. Medagliani, "Domain clustering for inter-domain path computation speed-up," *Networks*, vol. 71, no. 3, pp. 252–270, 2018.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.

[3] H. T. T. Binh, P. D. Thanh, and T. B. Thang, "New approach to solving the clustered shortest-path tree problem based on reducing the search space of evolutionary algorithm," *Knowledge-Based Systems*, vol. 180, pp. 12 – 25, 2019.

[4] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

[5] H. T. T. Binh, P. D. Thanh, T. B. Trung, and L. P. Thao, "Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem," in *Evolutionary Computation (CEC), 2018 IEEE Congress on*. IEEE, 2018, pp. 819–826.

[6] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag, "Adaptive fastest path computation on a road network: a traffic mining approach," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 794–805.

[7] M. K. M. Ali and F. Kamoun, "Neural networks for shortest path computation and routing in computer networks," *IEEE transactions on neural networks*, vol. 4, no. 6, pp. 941–954, 1993.

[8] S. Dasgupta, J. C. De Oliveira, and J.-P. Vasseur, "Path-computation-element-based architecture for interdomain mpls/gmpls traffic engineering: overview and performance," *IEEE Network*, vol. 21, no. 4, pp. 38–45, 2007.

[9] F. Paolucci, F. Cugini, A. Giorgetti, N. Sambo, and P. Castoldi, "A survey on the path computation element (pce) architecture," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1819–1841, 2013.

[10] L. Buzzi, M. C. Bardellini, D. Siracusa, G. Maier, F. Paolucci, F. Cugini, L. Valcarenghi, and P. Castoldi, "Hierarchical border gateway protocol (hbgp) for pce-based multi-domain traffic engineering," in *2010 IEEE International Conference on Communications*. IEEE, 2010, pp. 1–6.

[11] D. Siracusa, S. Grita, G. Maier, A. Pattavina, F. Paolucci, F. Cugini, and P. Castoldi, "Domain sequence protocol (dsp) for pce-based multi-domain traffic engineering," *Journal of Optical Communications and Networking*, vol. 4, no. 11, pp. 876–884, 2012.

[12] M. Gen, R. Cheng, and D. Wang, "Genetic algorithms for solving shortest path problems," in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*. IEEE, 1997, pp. 401–406.

[13] P. D. Thanh, H. T. T. Binh, and T. B. Trung, "An efficient strategy for using multifactorial optimization to solve the clustered shortest path tree problem," *Applied Intelligence*, 2020.

[14] R. Chandra, A. Gupta, Y.-S. Ong, and C.-K. Goh, "Evolutionary multi-task learning for modular training of feed-forward neural networks," in *International Conference on Neural Information Processing*. Springer, 2016, pp. 37–46.

[15] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, and G. Proietti, "Hardness, approximability, and fixed-parameter tractability of the clustered shortest-path tree problem," *Journal of Combinatorial Optimization*, pp. 1–20, 2019.

[16] P. D. Thanh, D. A. Dung, T. N. Tien, and H. T. T. Binh, "An effective representation scheme in multifactorial evolutionary algorithm for solving cluster shortest-path tree problem," in *Evolutionary Computation (CEC), 2018 IEEE Congress on*. IEEE, 2018, pp. 811–818.

[17] E. E. Agoston, *Introduction to Evolutionary Computing*. Berlin, Springer-Verlag, 2003.

[18] X. Yu and M. Gen, *Introduction to evolutionary algorithms*. Springer Science & Business Media, 2010.

[19] P. D. Thanh, T. B. Thang, N. V. Hoang, and N. B. Long, "Inter-domain path computation under domain uniqueness constraint instances," *Mendeley Data*, vol. v3, 2020. [Online]. Available: http://dx.doi.org/10.17632/t726xwcjf9.3

[20] P. Pop, O. Matei, and C. Pintea, "A two-level diploid genetic based algorithm for solving the family traveling salesman problem," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2018, pp. 340–346.

[21] Y. Yuan, Y.-S. Ong, A. Gupta, P. S. Tan, and H. Xu, "Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with tsp, qap, lop, and jsp," in *Region 10 Conference (TENCON), 2016 IEEE*. IEEE, 2016, pp. 3157–3164.