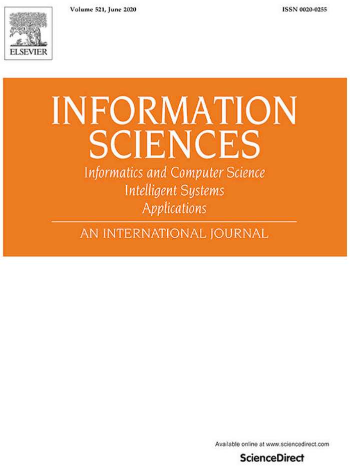


Volume 521, June 2020 ISSN 0020-0255

ELSEVIER

INFORMATION SCIENCES

Informatics and Computer Science

Intelligent Systems

Applications

AN INTERNATIONAL JOURNAL

Available online at www.sciencedirect.com

ScienceDirect

A Multifactorial Optimization Paradigm for Linkage Tree Genetic Algorithm

Huynh Thi Thanh Binh, Pham Dinh Thanh, Tran Ba Trung, Le Cong Thanh,
Le Minh Hai Phong, Ananthram Swami, Bui Thu Lam

PII: S0020-0255(20)30557-0
DOI: https://doi.org/10.1016/j.ins.2020.05.132
Reference: INS 15553

To appear in: *Information Sciences*

Received Date: 16 July 2019
Accepted Date: 30 May 2020

Please cite this article as: H.T. Thanh Binh, P.D. Thanh, T.B. Trung, L.C. Thanh, L.M. Hai Phong, A. Swami, B.T. Lam, A Multifactorial Optimization Paradigm for Linkage Tree Genetic Algorithm, *Information Sciences* (2020), doi: https://doi.org/10.1016/j.ins.2020.05.132

A Multifactorial Optimization Paradigm for Linkage Tree Genetic Algorithm

Huynh Thi Thanh Binh[a], Pham Dinh Thanh[b,d,*], Tran Ba Trung[a], Le Cong Thanh[a], Le Minh Hai Phong[a], Ananthram Swami[c], Bui Thu Lam[d]

[a]*School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam*
[b]*Faculty of Natural Science and Technology, Taybac University, Vietnam*
[c]*United States Army Research Laboratory, USA*
[d]*Le Quy Don Technical University, Vietnam*

Abstract

Linkage Tree Genetic Algorithm (LTGA) is an effective Evolutionary Algorithm (EA) to solve complex problems using the linkage information between problem variables. LTGA performs well in various kinds of single-task optimization and yields promising results in comparison with the canonical genetic algorithm. However, LTGA is an unsuitable method for dealing with multi-task optimization problems. On the other hand, Multifactorial Optimization (MFO) can simultaneously solve independent optimization problems, which are encoded in a unified representation to take advantage of the process of knowledge transfer. In this paper, we introduce Multifactorial Linkage Tree Genetic Algorithm (MF-LTGA) by combining the main features of both LTGA and MFO. MF-LTGA is able to tackle multiple optimization tasks at the same time, each task learns the dependency between problem variables from the shared representation. This knowledge serves to determine the high-quality partial solutions for supporting other tasks in exploring the search space. Moreover, MF-LTGA speeds up convergence because of knowledge transfer of relevant problems. We demonstrate the effectiveness of the proposed algorithm on two benchmark problems: Clustered Shortest-Path Tree Problem and Deceptive Trap Function. In comparison to LTGA and existing methods, MF-LTGA outperforms in quality of the solution or in computation time.

Keywords: Linkage Tree Genetic Algorithm, Multifactorial Optimization, Linkage models, Clustered Shortest-Path Tree Problem, Evolutionary Algorithm.

[*] Corresponding author.
Email addresses: binhht@soict.hust.edu.vn (Huynh Thi Thanh Binh), thanhpd05@gmail.com (Pham Dinh Thanh), batrung97@gmail.com (Tran Ba Trung), thanhcls1316@gmail.com (Le Cong Thanh), aquariuslee99@gmail.com (Le Minh Hai Phong), ananthram.swami.civ@mail.mil (Ananthram Swami), lambt@lqdtu.edu.vn (Bui Thu Lam)

1. Introduction

Linkage Tree Genetic Algorithm has been shown to scale excellently on a variety of discrete, Cartesian-space, optimization problems [1, 2]. Linkage Tree Genetic Algorithm (LTGA) determines the linkages between problem variables in the population, then clusters relevant variables to build a linkage tree. In each generation, the linkage tree is used to create crossover masks to prevent disruption between high-quality linked genes, and different partial structures of two good parent solutions can be juxtaposed to construct a new good solution. LTGA performs well in various problems: Permutation Flowshop Scheduling Problem [2, 3], Nearest-neighbor NK landscapes [4], MAX-SAT problem [5, 6], Deceptive trap function [7], Multidimensional Knapsack Problem [8], etc. and outperforms traditional Genetic Algorithm (GA). However, the linkage tree is built from a single combinatorial optimization problem without transferred knowledge from other relevant problems.

Combinatorial optimization problems in real-life like Jobs scheduling, Cloud computing etc. require solving many tasks simultaneously. Arising from the need to solve a large number of user requests in Cloud Computing, Multifactorial Evolutionary Algorithm (MFEA) proposed by Gupta, et al. [9] can solve multiple independent optimization problems simultaneously using a single population of solutions in the unified search space. The unified search space encompasses the shared knowledge for all of the tasks and the knowledge belonging to the particular optimization task. Transferring knowledge between different tasks occurs through adjusting and exchanging shared genetic material in the unified search space. In the process of transferring knowledge, good partial solutions of each task are used to support the others tasks. Leveraging the supportive genetic material requires calculating the commonality between all tasks for effective knowledge transfer.

Inspired by the idea of Multifactorial Optimization (MFO) and LTGA, we exploit the advantages of crossover mechanism of LTGA and the implicit genetic transferring of MFO. We adopt the idea that many problems are represented in the unified search space, building linkage tree for each task exploits the knowledge from the other tasks. In particular, the linkage tree indicates the distances between problem variables. These variables corresponding to each problem are the partial structure of individuals in the shared representation. The distance between two sets of variables indicates the dependence between them.

In this paper, we introduce Multifactorial Linkage Tree Genetic Algorithm (MF-LTGA) by combing the main features of LTGA and MFO: Linkage tree is used to determine the relationship between problems variables, which is used to leverage shared information among optimization problems. The assortative mating step is modified to combine crossover mechanism of LTGA and vertical cultural transmission of MFEA: A linkage tree is selected corresponding to a particular task then the crossover operator is applied to generate new offspring based on that linkage tree. The vertical cultural transmission in MF-LTGA serves to transfer the phenotype of parents to their offspring.

The effectiveness of MF-LTGA is shown in comparison to LTGA and existing algorithms on the canonical Clustered Shortest-Path Tree Problem (CluSPT) [10] and Deceptive Trap Function (DTF). The results indicate that MF-LTGA is superior to LTGA in computation time, and quality of the solution.

The main contributions of this paper are:

- We propose a mechanism combining key features of LTGA and MFO, which we call MF-LTGA.
- We modify process for building the linkage tree based on unified search space to exploit both information between problem variables and transfer knowledge of independent tasks.
- We introduce an assortative mating mechanism to enhance the compatibility between the main features of MFO and LTGA.
- We propose a crossover operator to keep the key advantages of MFO and LTGA as well as maintain population diversity.
- The experimental results show that the our algorithm is more efficient than existing methods.

This paper is organized as follows. Section 2 introduces related works. Background of MFO and LTGA is briefed in Section 3. Section 4 describes the MF-LTGA algorithm. Section 5 presents and discusses experimental results. The paper concludes in Section 6 with discussions of the future extension of this research.

2. Related works

Linkage Tree Genetic Algorithm (LTGA) was introduced by Thierens [11] and is one of the newest variants of Evolutionary Algorithm (EA) [12, 13]. LTGA learns the linkage between the problem variables and groups of variables by building a hierarchical clustered tree using a proximity distance metric. In each generation, LTGA builds a linkage tree and then uses that tree to generate new offspring. LTGA has been successfully applied to various types of problems which we review next.

In [1], Bosman et al. proposed LTGA to solve permutation optimization problems by employing a random key encoding of permutations. To evaluate the dependency between two variables, two factors were proposed: the first factor, called relative-ordering information, focuses on the order of two genes while the second factor, called adjacency information, focuses on the proximity of the two genes.

In [11], Bouter et al. applied LTGA to solve the deceptive $mk - trap$ function. The authors use the mutual information for evaluating the dependency between variables and build the linkage tree of a population of solutions.

In [14], Goldman, et al. introduced a benchmark problem, deceptive step trap problem for testing LTGA. To reduce the time complexity when calculating the entropy between all possible clusters, the authors also proposed the linkage between clusters. Instead of finding the entropy of an entire cluster, the new measure only finds the entropy between all pairs of independent problem variables in the population.

To improve the convergence of LTGA, Bosman, et al. [15] proposed Forced Improvements which is used when a solution cannot be improved. A different linkage model, Linkage Neighbors (LN) was also proposed. An advantage of the LN model compared to the Linkage Tree model is that it is well-suited to represent overlapping linkage relations.

Recently, the concept of Multifactorial Optimization (MFO) [16] has been introduced by Gupta et al. in [9] as a new optimization paradigm toward evolutionary multitasking with a single population of individuals. In contrast to traditional evolutionary search paradigm, MFO conducts evolutionary search concurrently on multiple search spaces corresponding to the optimization problems, each possessing a unique function landscape. The efficacy of MFO has been confirmed by a set of continuous and combinatorial optimization problems in [2, 17].

MFO has been applied to various algorithms which we review next. Feng, et al. [18] proposed two new mechanisms for combining Multifactorial Evolutionary Algorithm (MFEA) with Particles Warm Optimization Algorithm (PSO) (called MFPSO), Differential Evolution Algorithm (DE) (called MFDE). In the new algorithms, the authors designed new assortative mating schemes while the other components such as unified individual representation, vertical cultural transmission, etc., are kept the same as in the original MFEA.

Xie, et al. [19] introduced a hybrid algorithm combining MFEA and PSO (call HMFEA) in which PSO plays the role of local search in the MFEA. A difference between HMFEA and the original MFEA is that the PSO is added after genetic operation of MFEA and applied to the intermediate-pop in each generation. To adjust dynamically the velocity and guarantee that the convergence velocity is not too fast, an adaptive variation adjustment factor g_α is proposed. The factor g_α is used to control the velocity of each particle.

In [20], Wen and Ting combine the MFEA with Genetic Programming (GP) for learning an ensemble of decision trees. In this algorithm, each task is associated with one run of GP. To generate diverse decision trees, their algorithm further scrambles the dataset for each task by randomly mapping the feature indexes. The tasks will then work on the dataset with different feature sequences.

Zhong. et al. [21] proposed a multifactorial GP (MFGP) paradigm toward evolutionary multitasking GP. MFGP consists of a novel scalable chromosome encoding scheme and new evolutionary mechanisms for MFO based on self-learning gene expression programming.

Although MFO and LTGA were developed for solving various types of problems, there have been no studies that combine the strengths of MFO and LTGA into a new algorithm. Therefore, this paper proposes mechanisms to take the advantages of both MFO and LTGA into a new algorithm. The experimental results demonstrate the effectiveness of the new algorithm.

3. Preliminaries

This section provides a brief background of the Multifactorial Optimization paradigm and the Linkage Tree Genetic Algorithm.

3.1. Multifactorial Optimizations

In [9], Gupta et al. introduced Multifactorial Optimization as an evolutionary multi-tasking paradigm that optimizes multiple tasks simultaneously. Unlike traditional methods, MFO solves multiple tasks within only a single task. To achieve this, individuals are represented in unified search space and MFO calculates the skill of individual and splits the population into different groups: each individual is placed in the group corresponding to the task it performs best. The ability to solve problems in multitasking environments not only allows MFO to utilize genetic materials created in a specific group but also useful for another task.

To evaluate an individual, Gupta et al. [9] define the following properties for every individual p_i in population P:

- **Factorial Cost**: The factorial cost ψ^i_j of an individual p_i on task T_j is computed by its fitness or objective value on a particular task T_j.
- **Factorial rank**: Factorial rank r^i_j is the rank of p_i on task T_j, relative to all other individuals in P.
- **Scalar Fitness**: Scalar fitness φ_i of p_i is based on its best rank over all tasks; i.e. $\varphi_i = 1/\min\{r^i_1, r^i_2, \ldots, r^i_K\}$.
- **Skill Factor**: Skill factor τ_i of p_i is the one task, amongst all other tasks in MFO, with which the individual is associated. This may be defined as $\tau_i = argmin_j\{r^i_j\}$.

In order to calculate fitness of an individual, individuals are decoded in different tasks to obtain “Factorial Cost”. Individuals are evaluated by its correlation with other individuals based on “Factorial Cost” to find the most suitable task called “Skill Factor”.

3.2. Linkage Tree Genetic Algorithm

Recently, a powerful linkage-learning EA, LTGA, was proposed by Dirk Thierens [11]. LTGA maximizes the effectiveness of the crossover operator through discovering and exploiting the relationship between problem variables during the evolutionary searching. To store linkage information, LTGA uses an additional hierarchical tree, called linkage tree. A cluster of problem variables that LTGA believes to be linked is represented by a node in the linkage tree. In each generation, the linkage tree is rebuilt by selecting a set of solutions from the current population before determining the relationship between problem variables in that set.

3.2.1. Constructing Linkage Tree

LTGA aims to identify the variables that make a dependent set, then uses an agglomerative hierarchical clustering algorithm to proceed bottom-up. Hierarchical clustering algorithm constructs Linkage information between variables, and stores it as a Linkage Tree. Each node in the Linkage Tree is a cluster of genes that are close to each other. At its first stage, the algorithm considers each gene to be a dependent cluster, before repeatedly joining the two closest clusters to create a bigger one until all genes are in the same cluster. The size of population may impact the accuracy of the information the linkage tree represents. The larger the population size, the higher the possibility of good solutions appearing in it. Therefore, the linkage tree constructed from larger population may better reflect the relations between the genes. However, for larger population, the construction of linkage tree would be more consuming in terms of computational resources and the number of evolutionary operations on each generation would be higher. Hence, it is necessary to choose an appropriate population size in order to keep a balance between the linkage information accuracy and computational resources consumption.

The details are shown in Algorithm 1:

Algorithm 1: Hierarchical Clustering

```
Input: A set of solutions from the current population
Output: A Linkage tree

Compute the proximity matrix using metric $D$.
Assign each variable to a single cluster.
Repeat until one cluster left:
    Join two closest clusters $C_i$ and $C_j$ into $C_{ij}$.
    Remove $C_i$ and $C_j$ from the proximity matrix.
    Compute the distance between $C_{ij}$ and all clusters.
    Add cluster $C_{ij}$ to the proximity matrix.
```

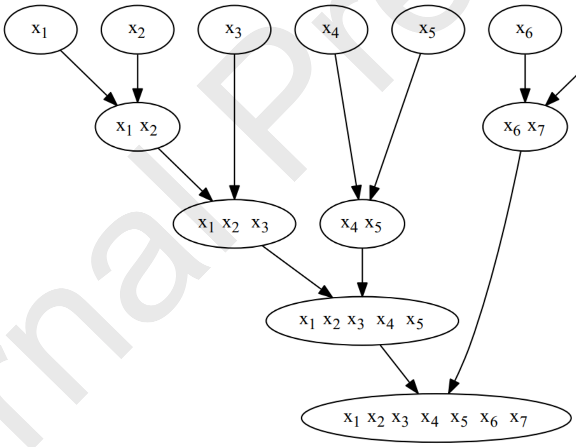


Figure 1: An example of hierarchical clustering algorithm on 7 genes.

An example of hierarchical clustering is shown in Figure 1: The first two closest genes x_1 and x_2 are joined into a cluster x_1x_2, clusters x_1, x_2 are removed from the proximity matrix. In the next iteration, LTGA considers the distances between the new cluster x_1x_2 and the other clusters, then combines the closest pair of clusters from the current population. After each generation, LTGA rebuilds the linkage tree from current population.

3.2.2. Crossover operator

Each cluster in the Linkage Tree is used as a crossover mask, the variables in a cluster are swapped between parent pair to generate two new offspring. If one of the offspring is better than its parents, then those offspring become parents for the next crossover for the remaining crossover masks. LTGA performs operations through $2l-1$ clusters in Linkage Tree, the order of visiting clusters to perform crossover operations impacts the quality of the final solution.

The outline of Linkage Tree Genetic Algorithm is presented in Figure 2.

4. Multifactorial Optimization with Linkage Tree Genetic Algorithm

In this section, we introduce the combination of Multifactorial Optimization (MFO) and Linkage Tree Genetic Algorithm (LTGA) which we call Multifactorial Linkage Tree Genetic Algorithm (MF-LTGA).

MFO is designed for conducting multitasking with the exchange of information among individuals through two key components: assortative mating and vertical cultural transmission. In a standard genetic algorithm and MFO, the solution representation and the crossover operator need to be designed to achieve good solution. However, this design will be difficult to achieve if there is insufficient domain knowledge. Different from the genetic algorithm, LTGA possesses unique solution reproduction and update operations through linkage models which learn the relationship between the problems variable through estimation of distribution.

MF-LTGA is our proposal by combining LTGA and MFO in order to capture the advantages of both algorithms to improve the quality of the solution. To hybridize MFO and LTGA, new assortative mating schemes are required. In addition, some operators like unified individual representation, vertical cultural transmission, etc., also need to change to adapt to LTGA. The workflow in Figure 3 describes the outline of our proposed algorithm, in which we maily focus on the two steps: *build linkage tree* and *perform associative mating based on linkage tree*. MF-LTGA start with a initial population of individuals which is presented in a unified search space. *The assortative mating* of MF-LTGA serves as the genetic operator to reproduce next generation as well as Multifactorial Evolutionary Algorithm (MFEA). However, *the assortative mating* of MF-LTGA is performed based on *linkage tree* which learns a probabilistic model of the current population of solutions. In addition, unlike MFEA, *vertical cultural transmission* is determined in assortative mating because it depends on tree selection. The pseudo code of MF-LTGA is presented in Algorithm 2. In what follows, the design of the MF-LTGA is detailed.

4.1. Linkage Tree Building

A key strength of LTGA is its ability to learn the relationship between the problem variables. To maintain this strength when applied to a multi-tasking environment, either a linkage tree is built for all tasks, or linkage trees are built separately for each task. Building only a single tree for all tasks can not provide the dependency between the variables because the relationship between two variables in one task might be different

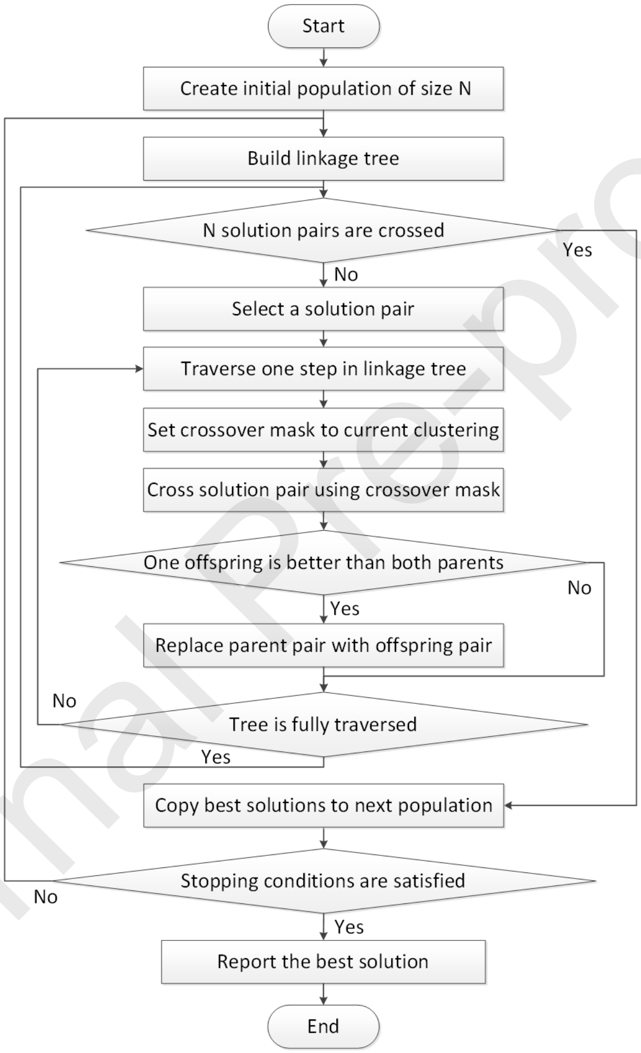

Start
Create initial population of size N
Build linkage tree
N solution pairs are crossed
Yes
No
Select a solution pair
Traverse one step in linkage tree
Set crossover mask to current clustering
Cross solution pair using crossover mask
One offspring is better than both parents
No
Yes
Replace parent pair with offspring pair
No
Tree is fully traversed
Yes
Copy best solutions to next population
Stopping conditions are satisfied
No
Yes
Report the best solution
End

Figure 2: The outline of Linkage Tree Genetic Algorithm

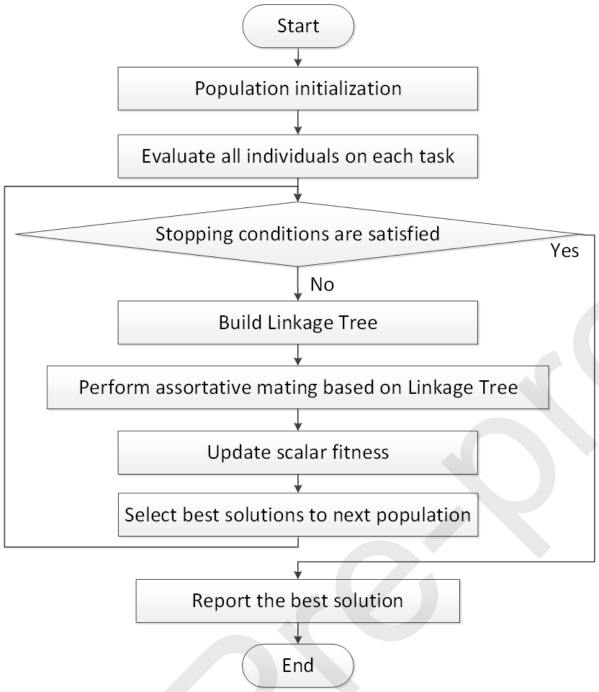

Start
Population initialization
Evaluate all individuals on each task
Stopping conditions are satisfied
Yes
No
Build Linkage Tree
Perform assortative mating based on Linkage Tree
Update scalar fitness
Select best solutions to next population
Report the best solution
End

Figure 3: The scheme of Multifactorial Linkage Tree Genetic Algorithm

Algorithm 2: Multifactorial Linkage Tree Genetic Algorithm (MF-LTGA)

```
Generate an initial population of size $n$;
Evaluate all individuals on every task in the multi-tasking environment, and
  obtain the skill factor of each individual;
$t \leftarrow 0$;
while stopping conditions are not satisfied do
    Build linkage tree                                ▹ Refer to Algorithm 3;
    Perform assortative mating based on linkage model on current-pop to
      generate an intermediate-pop                    ▹ Refer to Algorithm 4;
    Update the scalar fitness of all the individuals in intermediate-pop  ▹ Refer
      subsection 3.1;
    Select the fittest individuals from intermediate-pop to form the next
      generation;
    $t \leftarrow t + 1$;
end
```

from that in another task. Therefore, this paper applies the second approach. The pseudo code of the building linkage tree in MF-LTGA is given in Algorithm 3.

In Algorithm 3, for each task, we firstly generate a selected population including individuals whose Skill Factor is in that task. Next, each individual in the selected population is decoded to a solution for this task which will be added to a population, called task-population. Finally, linkage tree is built based on task-population in the same way in [11].

Algorithm 3: Linkage tree building in MF-LTGA

```
Input: $P$: A population of individuals in unified search space; $k$: Number of tasks.
Output: Linkage tree $T_i$ for task $i, i = 1, \dots, k$.
foreach task $i$ do
    Choose individuals from $P$ which have skill factor of task $i$ to generate selected-population ($C$);
    Decode all individuals of selected-population ($C$) to generate task-population ($P_i$);
    Build the linkage tree ($T_i$) based on task-population ($P_i$);
end
return $T_i, i = 1, \dots, k$
```

4.2. Assortative Mating

The pseudo code of the assortative mating in MF-LTGA is given in Algorithm 4. Firstly, current-population is partitioned into pairs of individuals which are considered as parents. Next, with each pairs of parents, we select randomly a single task for evaluation, because evaluating every individual for every problem being solved will often be computationally too expensive. However, comparisons and evaluations only on selected task may lead to a loss on good individuals of unselected tasks. These individual that has unselected skill factor might be an outstanding solution for that particular task and could produce good offspring on that task. Therefore, we need to create a backup population which contains the individuals that does not have their skill factor tasks selected. Next, the pair of parents will have crossover operator applied based on the tree of the selected task to generate offspring. As a result, the offspring imitate selected task, so that *vertical cultural transmission* is integrated into the assortative mating. Finally, offspring-pop and backup-pop are concatenated to form an intermediate-pop.

4.3. Crossover Operator

In this part, we will clarify the crossover operator based on linkage tree. The pseudo code of the crossover operator is presented in Algorithm 5. In the new crossover operator, we traverse the linkage tree top-down to set the crossover mask. With each mask, parent pair is crossed using crossover mask to generate a pair of offspring which is evaluated on the selected task to compete with the parent pair. If one of the children is better

Algorithm 4: Assortative Mating in MF-LTGA

```
Input:
  • $P$: A population of individuals in unified search space;
  • $k$: Number of tasks;
  • Linkage tree $T_i$ for task $i, i = 1, \ldots, k$.
Output: A new population of individuals in unified search space
Build a set of selected parents ($S$) by randomly partitioning population ($P$)
  into pairs of individuals;
foreach pair ($p_i, p_j$) in $S$ do
    $\tau_i \leftarrow$ skill factor of $p_i$;
    $\tau_j \leftarrow$ skill factor of $p_j$;
    $\tau \leftarrow \tau_i$;
    if $\tau_i \neq \tau_j$ then
        $\tau \leftarrow$ A random skill factor from $\{\tau_i, \tau_j\}$;
        if $\tau = \tau_i$ then
            Add $p_j$ to backup population ($B$);
        end
        else
            Add $p_i$ to backup population ($B$);
        end
    end
    Parents $p_i$ and $p_j$ crossover in task $\tau$ based on $T_\tau$ to generate two offspring
      $o_i$ and $o_j$                        ▹ Refer to Algorithm 5;
    Two offspring $o_i$ and $o_j$ imitates selected skill factor $\tau$ ;
    Add best individual of $\{o_i, o_j\}$ to offspring population ($O$);
end
Concatenate offspring-pop and backup-pop to form an intermediate-pop
  ($O \cup B$);
return intermediate-pop;
```

than both parents then the offspring pair replaces the parent pair, and MF-LTGA continues to traverse the linkage tree with the new pair. If none of the two children is better than their parents, MF-LTGA continues its tree traversal with the parent pair. Each individual p_i is assigned with a positive integer ne_i reflecting the number of generations that individual has existed in the population. However, an individual could exist in the population for many generations without producing any better solutions. Therefore, we propose a mechanism for replacing these bad individuals by new individuals. More specifically, such an individual will be punished whenever it fails to improve; once its punishment record reaches a certain limit, called individual-punishment-threshold (IPT), it is replaced by a new individual. By eliminating this kind of unimprovable individuals, IPT has positive impact on the diversity of the population as well as time consumption. The value selection of IPT is discussed in details in section 5.

Algorithm 5: Crossover Operator in MF-LTGA

```
Input:
    • Parents $p_i$ and $p_j$;
    • $max_p$: individual-punishment-threshold;
    • Linkage tree $T_\tau$ for task $\tau$;
Output: Two offspring $o_i$ and $o_j$;
is_improved ← False;
foreach node $t_i$ in $T_\tau$ do
    Set crossover mask to node $t_i$;
    Cross $p_i$ and $p_j$ using crossover mask to generate offspring $o_i$ and $o_j$;
    if one offspring better than both parents then
        Replace parents $p_i$ and $p_j$ by offspring $o_i$ and $o_j$;
        is_improved ← True;
    end
end
if !is_improved then
    $ne_i \leftarrow ne_i + 1$;
    if $ne_i > max_{p_i}$ then
        Replace parents $p_i$ by a new random individual;
        $ne_i \leftarrow 0$;
    end
    $ne_j \leftarrow ne_j + 1$;
    if $ne_j > max_{p_j}$ then
        Replace parents $p_j$ by a new random individual;
        $ne_j \leftarrow 0$;
    end
end
return Two offspring $o_i$ and $o_j$;
```

5. Simulation results

We evaluate the performance of MF-LTGA on two canonical problems: Clustered Shortest-Path Tree Problem (CluSPT) [10, 22] and Deceptive Trap Function (DTF) [4, 11]. These two problems are described in detail later in this section.

5.1. Evaluation criteria

We focus on the following criteria to assess the quality of the output of the algorithms.

Criteria	
Average (Avg)	Average function value over all runs.
Best-found (BF)	Best function value achieved over all runs.
Num.Opt	The number of instances in which the optimal solution is found.
Num.Eval	Number of evaluations to success.

We compare the performance of algorithms via a normalized difference. More specifically, let C_A and C_B denote the performance of algorithms A and B under metric C; then the relative performance of algorithm A relative to that of algorithm B is defined as

$$PI(A, B) = \frac{C_B - C_A}{C_B} * 100\%$$

As examples: C could denote the cost of the best solution found by an algorithm, or the average number of evaluations needed to obtain a solution.

To evaluate the performance of the MF-LTGA in solving the CluSPT and the DTF, we implemented three sets of experiments.

- In the first set, the quality of the solutions obtained by the C-MFEA [23] and E-MFEA [24] on each instance were compared with those obtained by MF-LTGA.

- In the second set, various experiments were performed to analyze the impact of possible influencing factors.

- In the third set, analyze the effective of MF-LTGA on instances of the Deceptive Trap Function.

This paper uses the decoding method and evolutionary operators in [23].

Each problem instance was evaluated 30 times for the CluSPT and 10 times for the DTF on Intel Core i7-3.60GHz, 16GB RAM computer using Windows 8 64-bit. The source codes of LTGA and MF-LTGA were written in the Python.

The simulation parameters include population size = 100, number of evaluations = 10^6, probability of random mating = 0.5, mutation rate = 0.05 and number of tasks = 2.

5.2. *Deceptive Trap Function*

The Deceptive Trap Function is a well-known canonical benchmark for LTGA. With m-Trap Functions, the number of local optima of the deceptive trap function is $2^m - 1$. A notable point for this problem is that a hillclimbing algorithm quickly becomes trapped in one of the local optima while Genetic Algorithm (GA) will quickly converge to the deceptive local optima [11].

5.2.1. *Problem formulation*

The Deceptive mk-Trap Function (mk-DTF) is a binary, additively decomposable function composed of m trap functions DT_i, each defined on a separate group of k bits (the total length is $l = mk$). The cost of mk-DTF is defined as:

$$f_{mk-DTF}(x_1 \ldots x_l) = \sum_{i=0}^{m-1} DT_i(x_{ik} \ldots x_{ik+k-1}) \tag{1}$$

with $x_i \in \{0, 1\}$

Call u the number of bits in such a group that are equal to 1:

$$DT_i(x_{ik} \ldots x_{ik+k-1}) = \begin{cases} k, & \text{if } u = k \\ k - 1 - u, & \text{otherwise} \end{cases} \tag{2}$$

Clearly, the array of all 1-bits is the global optimal solution of the mk-DTF and all schemata of order less than k are deceptive.

5.2.2. *Experimental setup*

The MF-LTGA is tested on deceptive functions with trap length $k = 4, 5, 6$. The number of blocks m varies from 5 to 30 with increments of 5, the problem length thus varies from 20 to 180. The details of problem instances and parameters are presented in Table 1.

Table 1: mk-DTF Instances and Parameters for Evaluating MF-LTGA and LTGA

k	3	3	3	3	3	3	4	4	4
m	5	10	15	20	25	30	5	10	15
Problem length	15	30	45	60	75	90	20	40	60
Population size	128	128	128	128	128	128	128	128	128
k	4	4	4	5	5	5	5	5	5
m	20	25	30	5	10	15	20	25	30
Problem length	80	100	120	25	50	75	100	125	150
Population size	128	128	128	256	256	256	256	256	256

Each problem instance was evaluated 10 times on Intel Core i7-3.60GHz, 16GB RAM computer using Windows 8 64-bit.

5.2.3. *Experimental results*

Table 2 presents the results obtained by two algorithms LTGA and MF-LTGA. The results in this table indicate that LTGA slightly outperforms MF-LTGA: LTGA gets optimal solutions in all 100 tests while MF-LTGA only finds optimal results in 99 out of 100 tests.

Table 2: mk-DTF Results Obtained By MF-LTGA and LTGA

		LTGA		MF-LTGA		
k	**m**	**Num.Opt**	**Num.Eval**	**Num.Opt**	**Num.Eval**	**P.Imp**
3	5	10	4,228.0	10	2,783.2	34.2%
3	10	10	13,664.8	10	8,908.8	34.8%
3	15	10	24,552.0	10	13,780.8	43.9%
3	20	10	34,007.6	10	23,434.8	31.1%
3	25	10	52,244.0	10	28,179.2	46.1%
3	30	10	57,387.2	10	39,765.2	30.7%
4	5	10	7706.4	10	4,408	42.8%
4	10	10	25615.2	10	17,113.2	32.2%
4	15	10	39010.8	10	28,367.2	27.3%
4	20	10	64053.2	10	48,695.6	24.0%
4	25	10	85021.2	10	48,549.6	42.9%
4	30	10	111479.2	10	69,353.2	37.8%
5	5	10	20342.4	10	13,900.8	31.7%
5	10	10	66150.0	10	51,959.6	25.1%
5	15	10	123994.4	10	65,327.2	47.3%
5	20	10	176378.4	9	14,1609.6	19.7%
5	25	10	231582.4	10	128,116.8	44.7%
5	30	10	284411.2	10	196,322.4	31.0%

Num.Opt: The number of optimal solutions are found. (Total number of instances = 10)

Num.Eval: The Average number of evaluations to success.

P.Imp: The percentage of differences between the average number of evaluations to success.

However, the number of evaluations needed to successfully find the optimal solutions of MF-LTGA is smaller than that of LTGA on every instance, and nearly 18% fewer on average. This difference climbs up to 47.3% on the test case $k = 5, m = 15$. Detailed comparisons are given in Figure 4 in which Num.Evals denotes number of evaluations needed to successfully produce the optimal solutions.

5.2.4. *Analysis of influential factors*

In this experiment, we investigate the Individual Punishment Threshold (IPT) parameter when applying MF-LTGA to solve multiple deceptive trap problems. Experimental results point out the suitable IPT value for solving these problems. We selected the IPT value based on the fraction of instances in which the optimal result is achieved

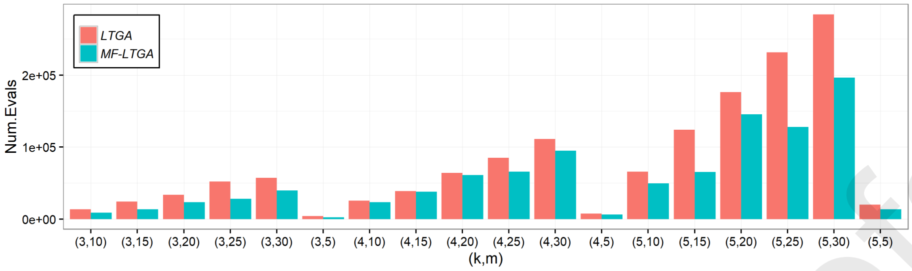


Figure 4: Comparing number of evaluations needed to find the optimal solutions

and the average number of evaluations needed to find the optimal solution. We performed experiments on IPT values ranging from 1 to the number of generations that our algorithm supposedly needed to find the desired optimal result when the punishment mechanism has not been applied. The result that is found when no threshold is used as the baseline to find the best IPT parameter. The results obtained when applying punishments were compared with the baseline in terms of the fraction of instances in which the optimal solution was found and average number of evaluations.

For each generation in MF-LTGA, pairs of individuals are chosen from the current population as parents, and the crossover operator is applied to generate new offspring. If the offspring are better than their parents then the parents, will be immediately replaced by their offspring. A parent existing in the population over many generations indicates that it has a low chance of producing better individuals. Intuitively, such a bad parent must be replaced by a new one. In MF-LTGA, the IPT parameter represents the maximum number of generations an individual can stay in the population. If the value of IPT is high then bad individuals will have a high chance to exist in many generations. However, if the IPT parameter is too low, individuals would tend to be quickly replaced, which could results in potential loss of good individuals and an unstable population. Learning the distribution from an unstable population could slow down the process of finding the optimal solutions.

Table 3 presents the number of instances (out of a total of 10 trials) in which the optimal solution was found and average number of evaluations for different IPT values. The results show that when the IPT value is 1, meaning an individual only exists for one generation, the optimal solution is found less often than with larger IPT values and the number of evaluations is 13.43% higher on average compared with when the IPT is larger. In particular, when the punishment value is 1, the optimal solution is found ony in 6 of 10 instances for both $k = 4, m = 10$ and $k = 4, m = 20$ cases (cells in bold).

The most suitable value for IPT is either 2 or 4. The results show that the optimal solution is always found and, compared to when the IPT value is larger, the numbers of evaluations for IPT of 2 and 4 are only 0.6% or 0.7% higher on average, respectively. In the Deceptive Trap $k = 5, m = 20$ problem, the optimal solution is always found because the bad solutions are replaced by new random solutions after 3 generations. As a result, the MF-LTGA can learn to escape local optima. Suitable IPT values will reinforce the search space exploration power of MF-LTGA.

Table 3: The number of found optimal values and average number of evaluations for different Individual-punishment-threshold

IPT		**1**		**2**		**3**		**4**		**5**		**6**		**7**		**8**	
k	m	Op	N.Eval	Op	N.Eval	Op	N.Eval	Op	N.Eval	Op	N.Eval	Op	N.Eval	Op	N.Eval	Op	N.Eval
3	5	10	3108	10	2749.6	10	2755.2	10	2783.2	10	2783.2	10	2783.2	10	2783.2	10	2783.2
3	10	10	10440	10	8676.8	10	9338	10	8908.8	10	8908.8	10	8908.8	10	8908.8	10	8908.8
3	15	10	14643.2	10	14062.4	10	14027.2	10	13780.8	10	13780.8	10	13780.8	10	13780.8	10	13780.8
3	20	10	23246	10	23765.2	10	23836	10	23434.8	10	23434.8	10	23434.8	10	23434.8	10	23434.8
3	25	10	28327.2	10	28179.2	10	28179.2	10	28179.2	10	28179.2	10	28179.2	10	28179.2	10	28179.2
3	30	10	39658.4	10	39800.8	10	39765.2	10	39765.2	10	39765.2	10	39765.2	10	39765.2	10	39765.2
4	5	10	5700	10	5069.2	10	4734.8	10	4408	10	4408	10	4408	10	4408	10	4408
4	10	**6**	25142	10	15490.8	10	15912	10	17113.2	10	16536	10	16551.6	10	16426.8	10	16567.2
4	15	10	28744.8	10	27163.6	10	27352.4	10	28367.2	10	28013.2	10	28013.2	10	28013.2	10	28013.2
4	20	**6**	51666	10	45914.8	9	44169.78	10	48695.6	9	44380.44	9	44380.44	10	48664	9	44380.44
4	25	10	50371.2	10	48114	10	48985.2	10	48549.6	10	48549.6	10	48549.6	10	48549.6	10	48549.6
4	30	9	85045.33	10	72352	10	72780.4	10	69353.2	10	69353.2	10	69353.2	10	69353.2	10	69353.2
5	5	10	13785.6	10	13977.6	10	14083.2	10	13804.8	10	13900.8	10	13900.8	10	13900.8	10	13900.8
5	10	7	58352	10	48274.8	10	52136	10	51959.6	10	51587.2	10	50293.6	10	50411.2	10	50705.2
5	15	10	78884	10	67399.2	10	66807.2	10	65327.2	10	65327.2	10	65327.2	10	65327.2	10	65327.2
5	20	7	166489.7	*10*	136620	9	125092	*10*	141609.6	9	136048	9	133144	9	133628	9	133628
5	25	10	139475.2	10	133027.2	10	130249.6	10	128116.8	10	128116.8	10	128116.8	10	128116.8	10	128116.8
5	30	9	238532.4	10	200792.4	10	197931.6	10	196322.4	10	196322.4	10	196322.4	10	196322.4	10	196322.4

Op: The number of optimal solutions are found. (Total number of instances = 10)

N.Eval: The Average number of evaluations to success.

5.3. *Clustered Shortest-Path Tree Problem*

5.3.1. *Problem formulation*

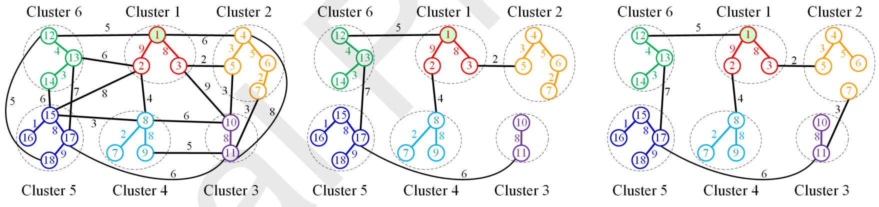
We let $G = (V, E, w)$ represent a simple, connected and undirected graph, with vertex set V, edge set E, and non-negative edge weights w. An edge between vertices u and v is denoted by (u, v), and its weight is denoted by $w(u, v)$.

For a vertex subset U, the sub-graph of G induced by U is denoted by $G[U]$. A collection $C = \{C_i | 1 \leq i \leq k\}$ of subsets of V is a partition of V if the subsets are mutually disjoint and their union is exactly V. A path in G is simple if no vertex appears more than once on the path. This paper only considers simple paths.

For a given spanning tree T of $G = (V, E, w)$ and $u, v \in V$, let $d_T(u, v)$ denote the shortest path length between u and v on T.

The CluSPT problem [10] is defined as follows:

Input:	- A weighted undirected graph $G = (V, E, w)$. - Vertex set V is partitioned into k clusters $C_1, C_2, ..., C_k$. - A source vertex $s \in V$.
Output:	- A spanning tree T of G. - Sub-graph $T[C_i](i = 1, \ldots, k)$ is a connected graph.
Objective:	$\sum_{v \in V} d_T(s, v) \rightarrow \min$



(a) The original graph (b) A valid solution to CluSPT (c) An invalid solution to CluSPT

Figure 5: An example of valid and invalid solutions for the CluSPT

Figure 5 illustrates the cases of valid and invalid solutions of CluSPT. Figure 5(a) shows the input graph G with 6 clusters, 18 vertices and vertex 1 as source vertex. Figure 5(b) presents a valid solution of CluSPT. In Figure 5(c), the vertex 6 and vertex 7 in cluster 2 are not connected, so this solution violates the second condition of the output of the CluSPT problem.

5.3.2. *Problem instances*

For assessment of the proposed algorithms' performance, we created instances for CluSPT from Clustered Traveling Salesman Problem (CluTSP) instances [25, 26] by adding the information of the source vertex. The main reason for building CluSPT instances from CluTSP instances was that CluTSP instances have been shown to be suitable for clustered problems in general [26].

All tested instances are available via [27]

5.3.3. Experimental results

Comparison between the performance of existing algorithms and that of MF-LTGA.

In this section, we compare the results obtained by AAL [22], C-MFEA [23], E-MFEA [24], and LTGA with those achieved by MF-LTGA. Tables 5, 6 and 7 illustrate the results obtained by these algorithm on instances of Type 1, Type 5 and Type 6. In table 7, the symbol "-" indicates that the corresponding instances were not executed by C-MFEA. In these tables, a cell in red and italics means that the result of the algorithm exceeds that of MF-LTGA on that instance. Moreover, if a result is marked in bold then it is the best result for that instance. For example, the result of LTGA for 50kroA100 is in italic red because it is better that that of MF-LTGA; meanwhile, the best performed for that instance is AAL and its result is in bold.

The results in Table 5, 6 and 7 point out that both single-tasking (ST) and multitasking (MT) outperform AAL, E-MFEA and C-MFEA in most test cases. In particular, MT outperformed the three existing algorithms on all test cases in Type 5. Table 4 summarizes the comparison results among AAL, E-MFEA, C-MFEA, ST and MT on the benchmarks.

The experimental results point out that MT is also better than ST on approximately 83% of the test cases i.e., 14 out of 22 Type 1 instances, 12 out of 12 Type 5 instances and 17 out of 18 Type 6 instances. Maximum PI(MT, ST) are 58.8% (for Type 1), 23.7% (for Type 5) and 3.2% (for Type 6).

The experimental results obtained by AAL, C-MFEA, E-MFEA and MF-LTGA on Type 1 instances are shown in Table 5. On this set of instances, MT outperforms ST on 14 out of 22 instances. C-MFEA and E-MFEA outperforms ST on 6 and 16 out of 22 instances. E-MFEA outperforms MT on 1 out of 22 instances whereas C-MFEA is worse than MT on all instances.

The results on Type 6 instances are displayed in Table 7. ST outperformed C-MFEA on 12 out of 12 instances and outperformed E-MFEA on 18 out of 18 instances.

Table 4: Comparison of results obtained by MF-LTGA and the existing algorithms.

Algorithm		**Type 1**	**Type 5**	**Type 6**	**Total**
	Number of instances in a Type	22	12	18	52
AAL	Number of instances on which MF-LTGA outperformed AAL	18	12	18	48
	Maximum PI(MF-LTGA, AAL)	66.4%	73.7%	69.0%	
C-MFEA	Number of instances on which MF-LTGA outperformed C-MFEA	22	12	18	52
	Maximum PI(MF-LTGA, C-MFEA)	55.3%	25.6%	33.1%	
E-MFEA	Number of instances on which MF-LTGA outperformed E-MFEA	21	12	17	51
	Maximum PI(MF-LTGA, E-MFEA)	28.6%	30.4%	34.8%	
LTGA	Number of instances on which MF-LTGA outperformed LTGA	14	12	17	43
	Maximum PI(MF-LTGA, LTGA)	58.8%	23.7%	3.2%	

Table 5: Results Obtained By E-MFEA, C-MFEA, LTGA and MF-LTGA on Instances In Type 1

	AAL	**C-MFEA**		**E-MFEA**		**LTGA**		**MF-LTGA**	
Instances	Avg	BF	Avg	BF	Avg	BF	Avg	BF	Avg
10berlin52	82619.1	48569.8	57519.6	46185.8	46707.8	44291.9	44979.3	44059.2	**44624.7**
10eil51	4489.1	1891.7	2336.6	2008.3	2039.4	2751.7	3337.9	1738.5	**1826.6**
10eil76	7220.5	2489.5	3286.5	2775.4	2973.3	4440.9	5887.5	2347.3	**2424.9**
10kroB100	384046.7	170695.2	227972.8	198181.6	218275.1	317039.7	373333.9	148479.3	**155902.1**
10pr76	1042986	632704.5	785555.1	643903.6	665835.1	1060518.5	1237905.1	545368.0	**570075.2**
10rat99	22402.3	8937.1	11470.7	10427.6	10792.8	15805.0	18368	7928.2	**8357.2**
15eil51	2833.5	1922.0	2724.1	1662.9	***1781.4***	2020.7	*2116.3*	2013.5	2127.2
15eil76	8147	3773.0	4752	3349.0	3402.8	3351.2	3454.6	3336.9	**3398.7**
15pr76	1627671.1	833734.0	1165803.3	772173.1	***787889***	807555.5	836711	810496.6	828216.5
15st70	10074.1	5171.8	6500.6	4972.1	5117.7	4403.6	***4516.2***	4451.3	4541.6
25eil101	12327.8	6852.3	8695.7	5192.4	5248.4	4863.7	5010.3	4859.9	**4928.3**
25kroA100	324160	266798.7	377454.1	164038.4	***167528.9***	166260.1	173517.3	164732.2	168833.1
25lin105	211355.4	182650.4	262420.6	106500.2	***107524.2***	138695.6	143329.3	136815.2	139356.5
25rat99	16747	12931.7	17533.2	9234.7	*9375.8*	8143.5	***8427.3***	9876.1	10017.7
50eil101	*9195.2*	9461.1	12689.9	3978.2	***3991.2***	8016.7	*8242.8*	10067.0	10113.8
50kroA100	*319270*	451952.5	618957.9	173626.5	***176321.3***	444438.5	*525951.5*	526297.5	526913.4
50kroB100	*311384.2*	450713.7	599175.7	137956.1	***138761.9***	341053.0	*472108.3*	538220.7	538973.3
50lin105	*273675.6*	309399.9	474219.6	147782.5	***148306.8***	331727.6	*385223.6*	392315.4	392632.5
5eil51	4397.7	1524.0	***1663.4***	1830.6	1942.3	3050.3	3728	1771.9	1813.5
5eil76	6539.8	2689.2	2961.6	3210.7	3297.5	5597.0	6770	2741.8	**2904.8**
5pr76	1688544.9	609766.6	652318.1	660629.3	700987.2	594603.1	***632258.1***	597900.7	638376.1

5st70	10270.5	4589.7	4959.6	4795.3	5171.7	4629.6	4833.4	4579.3	**4749.2**

Table 6: Results Obtained By E-MFEA, C-MFEA, LTGA and MF-LTGA on Instances In Type 5

	AAL	**C-MFEA**		**E-MFEA**		**LTGA**		**MF-LTGA**	
Instances	Avg	BF	Avg	BF	Avg	BF	Avg	BF	Avg
10i120-46	227078.7	105754.4	125137.1	119168.5	120920.6	98147.5	***101674.5***	100270.8	102508.8
10i45-18	59797.3	26942.8	32663.5	27164.4	28538.5	23391.5	25074.7	22890.4	**24546.8**
10i60-21	89193.4	37640.0	45427.1	43125.9	45389.1	34424.5	37144	34511.7	**36262.2**
10i65-21	102156.3	41053.9	49824.3	46651.5	47955.3	38910.7	41307.7	39380.1	**41141.1**
10i70-21	103674.7	41892.8	55760.2	49875.8	51532.2	39365.4	41532.5	40070.0	**41512.5**
10i90-33	128438.3	55361.9	65493.1	63225.8	65633.3	54636.8	57450.7	55393.4	**57338.6**
5i45-18	59161.2	15511.5	17007	20042.8	22345.5	14884.3	15571.6	14884.3	**15551.8**
5i60-21	63252.8	29797.9	34613	35099.9	36474.5	28842.3	30737	28850.1	**30548**
5i75-22	73544.5	34867.3	38705.1	37992.7	40668.9	35525.8	38580	36277.0	**38021.5**
5i90-33	114467.8	53230.6	***55888.2***	62701.2	65622.1	53563.1	*56096.9*	53818.2	56841.2
7i60-21	85486.3	37690.6	41532.3	44669.6	46337.4	37364.1	39637.2	37447.5	**39601.2**
7i65-21	82949.6	35878.8	40222.5	45237.3	47211.2	36520.3	38528.6	35570.9	**38100.9**

Table 7: Results Obtained By E-MFEA, C-MFEA, LTGA and MF-LTGA on Instances In Type 6

	AAL	**C-MFEA**		**E-MFEA**		**LTGA**		**MF-LTGA**	
Instances	Avg	BF	Avg	BF	Avg	BF	Avg	BF	Avg
12eil51-3x4	4209.1	2185.0	2691.5	1922.1	1960.9	1730.5	1829.5	1721.3	**1799.5**
12eil76-3x4	7665.9	3065.7	3896.1	3352.2	3449.3	2775.7	2926.3	2757.4	**2874.8**
4berlin52-2x2	70649.9	23635.3	24751	35413.1	37121.5	23291.1	24580.5	23330.2	**24190.6**
4eil51-2x2	5187.5	1909.5	2053.9	2545.3	2641.1	1898.5	2014.3	1900.5	**1996.9**
4eil76-2x2	6882.8	2949.1	***3179.9***	4319.3	4517.2	3007.7	3263.5	3084.2	3255.8
4pr76-2x2	1607169.6	446862.4	***480043.8***	688228.2	762880.2	445424.1	505575.5	470046.8	497917.4
6pr76-2x3	1269147.1	656978.3	736743.5	741847.3	771563.7	670286.1	***696701.5***	655575.4	697341.8
6st70-2x3	8147.4	3508.1	4244.1	3880.5	4057	3505.1	3745.3	3574.2	**3728.5**
9eil101-3x3	9560.8	3320.3	4345.8	4281.2	4585	3303.5	3515.3	3380.2	**3511.7**
9eil51-3x3	4642.4	2106.5	2630.8	2127.0	2182.7	1937.0	2046.6	1952.3	**2038.4**
9eil76-3x3	6659.6	3401.8	4048	3599.5	3730.8	3089.7	3245.6	3070.5	**3220.7**
9pr76-3x3	1344518.1	642796.6	783056.1	713966.7	749030.6	565527.5	591679.5	572387.3	**591205.8**
12st70-3x4	10072.3	-	-	4750.9	4795.7	4213.2	4341.5	4225.0	**4315**
15pr76-3x5	1316199.6	-	-	601015.9	***623645.7***	787601.4	829304.8	774271.4	808278.1
16eil51-4x4	3123.2	-	-	1371.3	***1425.9***	1780.5	1834.2	1756.5	1795.2
16eil76-4x4	6625.7	-	-	2314.0	***2374.9***	2548.0	2799	2556.1	2708.9
16lin105-4x4	309572.4	-	-	179729.7	179729.7	162114.6	170731.4	162677.5	**166291.5**
16st70-4x4	8073.9	-	-	3560.4	3560.4	3197.4	3405.5	3258.0	**3356.6**

Convergence trends.

We use the functions in [9] for computing the normalized objectives and averaged normalized objectives, and analyze the convergence trends of the proposed algorithm. 410

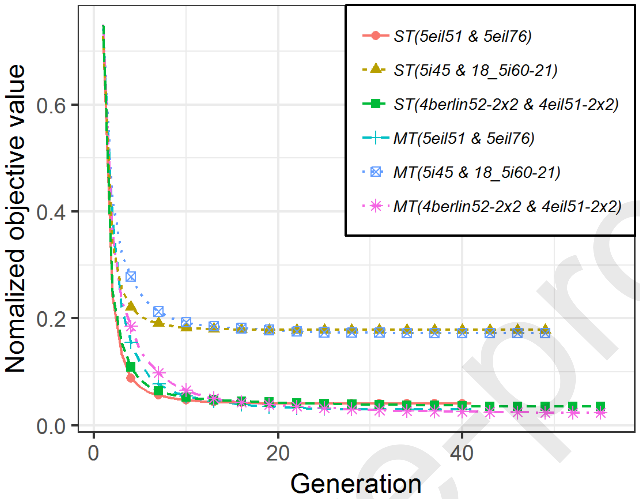
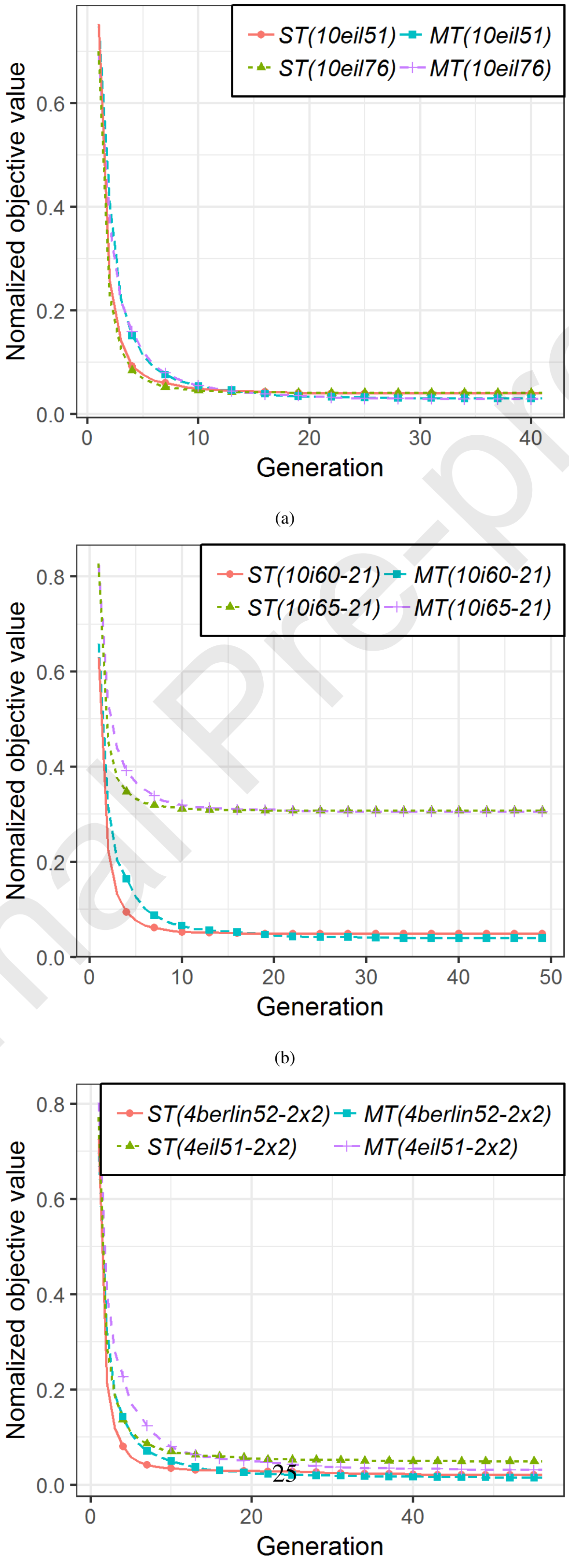


Figure 6: Convergence trends of $\tilde{f}$ in multi-tasks and serial single task for instances 10eil51 and 10eil76 in Type 1; instances 10i60-21 and 10i65-21 in Type 5; instances 4berlin52-2x2 and 4eil51-2x2 in Type 6.

Figure 6 illustrates the convergence trends of the ST and MT for instances 10eil51 and 10eil76 in Type 1; instances 10i60-21 and 10i65-21 in Type 5; instances 4berlin52-2x2 and 4eil51-2x2 in Type 6. These figures point out that the convergence rate of MT is faster than that of ST in most test cases.

415 A notable point in Figure 6 is that the numbers of evaluations of each generation are proportional to the dimensionalities of the instances. Moreover, in this experiment, the number of evaluations of each generation is a constant parameter. Due to this reason, the number of generations among instances might vary.



(a)

(b)

(c)

Figure 7: Comparing convergence trends of $\tilde{f}_1$ and $\tilde{f}_2$ in multi-tasks and serial single task for instances 10eil51 and 10eil76 in Type 1; instances 10i60-21 and 10i65-21 in Type 5; instances 4berlin52-2x2 and 4eil51-2x2 in Type 6.

The major convergence trends of those algorithms in Figure 6 is that MT converges slower than ST for initial generations but MT surpasses ST in later generations which means that the implicit genetic transferring among tasks in evolutionary multitasking paradigm improves the convergence speed of MT in comparison with ST

Figure 7 provides insight into the improved performance as a consequence of MT. The figure depicts the convergence trends corresponding to each individual task, which is somewhat similar to that of MT and ST in Figure 6 when the convergence rate of each task in MT is better than the corresponding task in ST in later generations.

5.3.4. Analysis of influential factors

Parameters, such as population size or crossover rate, play an important role in meta heuristic algorithms. Therefore, we perform experiments by applying different parameter values on each instance of the dataset to find the best parameters for the proposed algorithm. In this study, a 24-instance dataset is created by randomly selecting 8 instances of each a type.

The ideal situation would be to find parameters that can produce superior results for all instances. Unfortunately, the instances are very diverse in size and structure, so finding perfect parameters settings across all the data is unrealistic. Therefore, the default parameters settings for the algorithm will be selected by evaluating each parameter with the following metrics:

- **Time-of-best (TOB)**: For each parameter, a set P of its values will be evaluated on the dataset. The TOB of one such value is the number of instances for which that value achieved the best result among all values in P.
- **Distance-to-optimal (DTO)**: We also compute distance-to-optimal of each value of the parameter, which is the average distance between the result obtained by that value and the best result of all values, according to the following formula:

$$DTO(p) = \frac{\sum_{d \in D} \dfrac{r(d,p) - \min\limits_{p' \in P} r(d,p')}{\min\limits_{p' \in P} r(d,p')}}{|D|} \tag{3}$$

  where:

  - P is a set of values of the parameter and p is one of those values.
  - D is the dataset and d is one instance of it.
  - $r(d, p)$ is the result of our algorithm on instance d with the value p of the parameter being considered.

Some values of the parameter can be extremely good in many instances but extremely bad in others, which leads to high TOB and high DTO. Conversely, some values might show acceptable results for all instances but scarcely be the best in any of them, signifying low TOB and low DTO. Hence, we would want to choose the value that satisfy both conditions: high TOB and low DTO. In this paper, we use both of the above metrics to evaluate two important parameters: population size, individual punishment threshold and dimension of the input graph.

Impact of Population Size.

Population size plays an important role in population-based meta-heuristic algorithms. Therefore, we perform experiments to study the effect of population size on the algorithm's effectiveness. We consider six different population sizes: 50, 75, 100, 150, 200, 300 on the chosen 24-instance dataset.

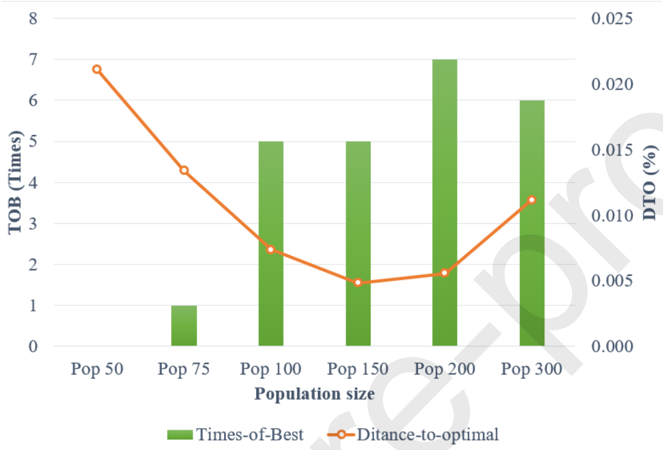


Figure 8: Impact of Population Size

For each population size, both metrics - TOB and DTO - are used for evaluation, the result is shown in Figure 8.

Figure 8 shows that the TOB increases continuously as the population size increases and reaches its peak at population size of 200. On the other hand, the DTO decreases significantly until the population reaches 150, before increasing by a small amount when the population size is 200, and surges when it is 300. These results can be explained by the common impact of population size on the result as well as the time-consuming problem of population-based meta-heuristics algorithms. Small populations are more likely to experience loss of diversity over time because the mating between individuals with similar genetic structures occurrs regularly. Therefore, the search space of the algorithm is narrowed, which usually leads to poor results as shown by the result for the population size of 50, 75 or 100 in this case. In contrast, a larger population would often means more evaluations and operations executed, which cost higher computational resources like time and space, leading to poor results if under the constraint of number of evaluations or computational time. For our algorithm, the population size of 200 is chosen because it has both best TOB and second best DTO. Although population size of 150 has smaller DTO, it is only a marginal difference while its TOB is obviously lower. Similarly, although population size of 300 has almost the same TOB as 200 (lower by only 1), it has significantly higher DTO (twice that of population size 200).

Impact of Individual Punishment Threshold.

IPT is also an important parameter in our algorithm. IPT is the maximum allowed number of subsequent generations for which that individual has not led to improved offspring. To analyze the impact of IPT, we performed experiments by applying different IPT values ranging from one to seven. Times-of-best and Distance-to-optimal of these values were calculated and the result is shown in Figure 9.

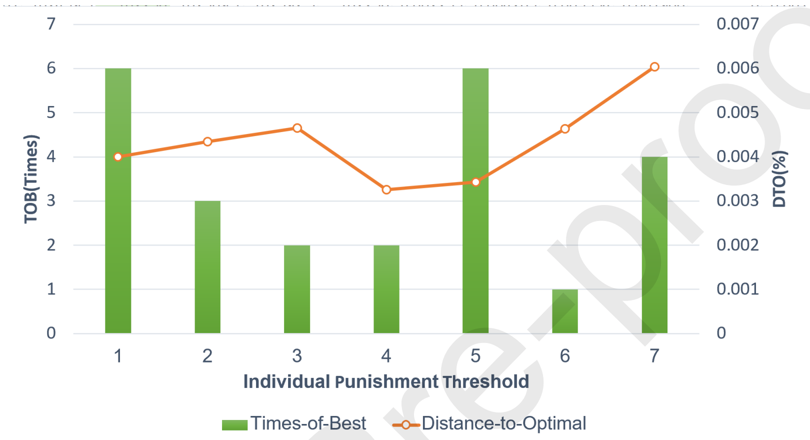
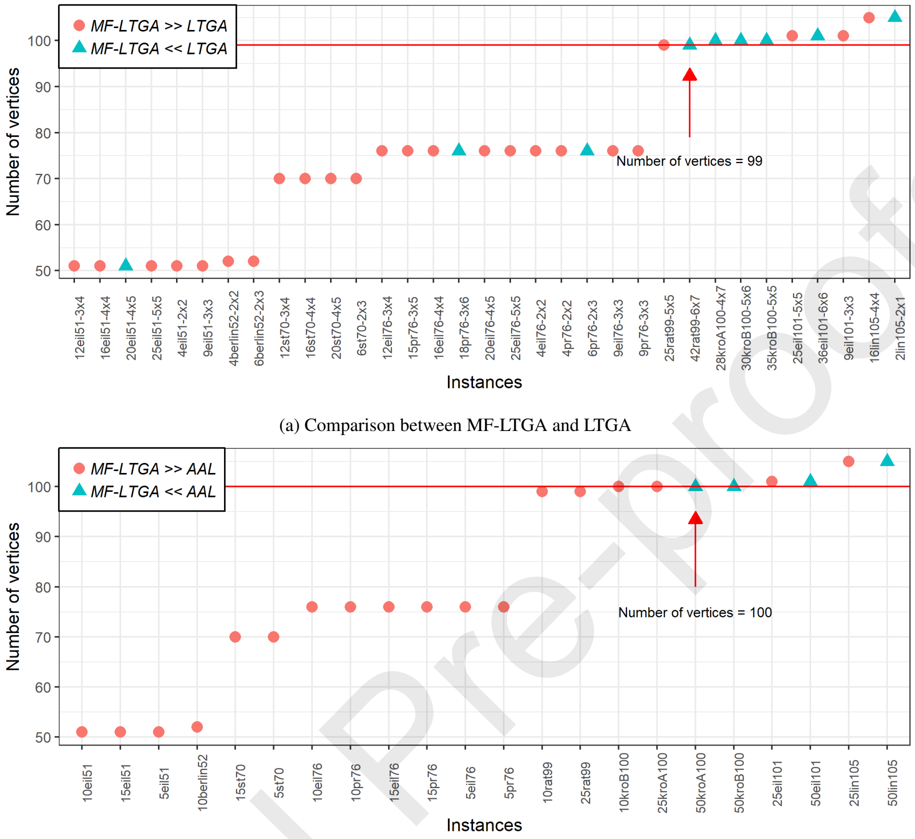


Figure 9: Impact of the parameter IPT

Figure 9 points out that IPT of 4 produced result with the best distance-to-optimal; however, its times-of-best is one of the lowest. Meanwhile, the results of IPT of 1 and 5 not only have the highest TOB but also have acceptable DTO - 3rd and 2nd among all seven IPT values, respectively. Although both 1 and 5 are promising values, IPT of 5 is selected as default setting because small IPT value like 1 may lead to the constant replacements of individuals, which in turn makes it easy for the population to lose promising genetic structures.

Impact of the Dimension of input data.

The dimension of input data often has remarkable impacts on the results so we decided to analyze how the number of graph vertices influenced the results obtained by MF-LTGA comparing to AAL and LTGA. Figure 10, Figure 11 and Figure 12 illustrate the relationship between number of vertices and algorithms' performance on instances of Type 1, Type 5 and Type 6, respectively. In these figures, the round symbol "A >> B" means algorithm A outperformed algorithm B on the instance denoted by that symbol's column, whereas the triangle symbol "A << B" means the opposite.

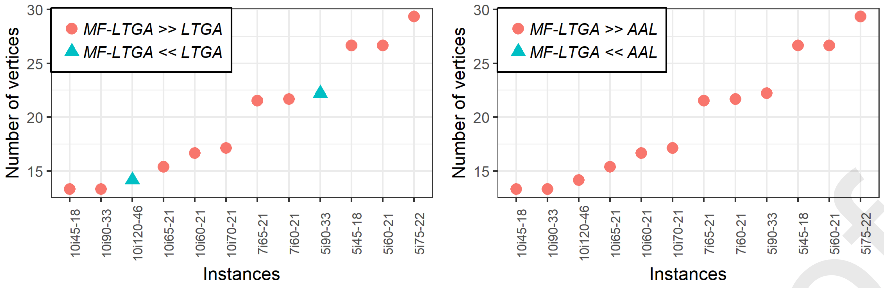


(a) Comparison between MF-LTGA and LTGA

(b) Comparison between MF-LTGA and AAL

Figure 10: The scatter of the instances and the number of vertices on Type 1

With respect to the instances of Type 1, Figure 10(a) shows that MF-LTGA exceed LTGA when the number of vertices is less than 99. But once that number reaches 99, LTGA tends to prevail over MF-LTGA. Figure 10(b) points out that the results obtained 505 by MF-LTGA are completely superior to ones obtained by AAL when the number of vertices is less than 100. After that 100 mark, the results obtained by the two algorithms are balanced.

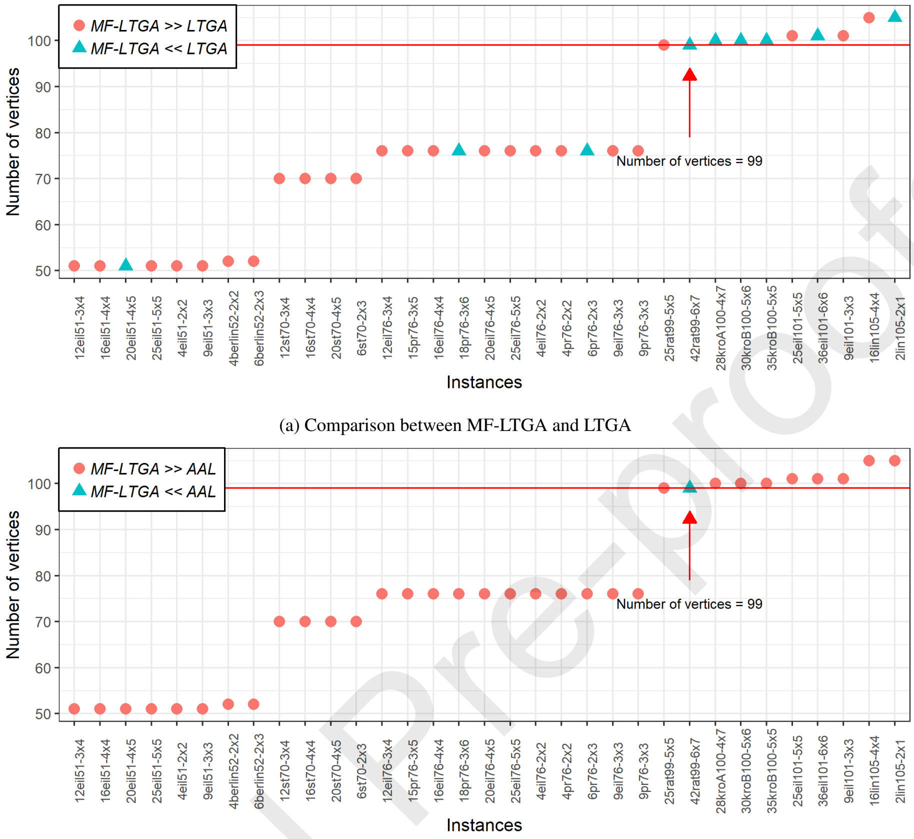


(a) Comparison between MF-LTGA and LTGA

(b) Comparison between MF-LTGA and AAL

Figure 11: The scatter of the instances and the number of vertices on Type 5

On instances in Type 5, Figure 11(a) and Figure 11(b) show that MF-LTGA surpasses both algorithms LTGA and AAL on almost all test cases. Particularly, MF-LTGA defeats AAL on every test case of this type.



(a) Comparison between MF-LTGA and LTGA

(b) Comparison between MF-LTGA and AAL

Figure 12: The scatter of the instances and the number of vertices on Type 6

The impact of the number of vertices on the comparison between MF-LTGA and LTGA on instances of Type 6 is similar to that on instances of Type 1. In other words, results obtained by MF-LTGA are better than results obtained by LTGA when the number of vertices is less than 99 but the opposite happens when the number of vertices is greater than or equal to 99 (Figure 12(a)). On the other hand, between MF-LTGA and AAL, the former shows dominance on all test cases except for one when the number of vertices reaches 99 - instance 42rat99-6x7 is the only test case where AAL outperforms MF-LTGA.

6. Conclusion

This paper introduced a mechanism for combining LTGA and MFO. The novel algorithm kept the main features of both LTGA and MFO, and described new methods for building Linkage Tree Model, Assortative Mating and Crossover Operator. These three new methods are the cornerstones in merging the features of MFO and LTGA.

Assortative Mating serves to reproduce new generation of population from the unified search space of all tasks, which is achieved by applying Crossover Operator on pairs of individuals of possibly different tasks based on the Linkage Tree Model.

The experimental results show that the proposed algorithms are more effective in solving the canonical CluSPT and DTF compared with some other existing metaheuristics. The results also illustrate the impact of individual-punishment-threshold, population size, and input data dimension on different kinds of problems. These parameters have significant task-dependent impact on the results. However, since MF-LTGA requires building a linkage tree for every task, its time complexity and computational resources consumed are significant.

Several theoretical aspects of the MF-LTGA will be investigated in more detail. In the future, we will focus on methods for constructing only one Linkage Tree Model for all tasks. This will potentially improve the algorithm's performance in terms of computational complexity.

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] P. A. Bosman, N. H. Luong, D. Thierens, Expanding from Discrete Cartesian to Permutation Gene-pool Optimal Mixing Evolutionary Algorithms, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, ACM, 2016, pp. 637–644.

[2] A. Bouter, T. Alderliesten, C. Witteveen, P. A. Bosman, Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2017, pp. 705–712.

[3] G. H. Aalvanger, N. H. Luong, P. A. N. Bosman, D. Thierens, Heuristics in Permutation GOMEA for Solving the Permutation Flowshop Scheduling Problem, in: A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, D. Whitley (Eds.), Parallel Problem Solving from Nature – PPSN XV, Springer International Publishing, 2018, pp. 146–157.

[4] D. Thierens, P. A. Bosman, Evolvability analysis of the linkage tree genetic algorithm, in: International Conference on Parallel Problem Solving from Nature, Springer, 2012, pp. 286–295.

[5] R. de Bokx, D. Thierens, P. A. Bosman, In Search of Optimal Linkage Trees, in: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, 2015, pp. 1375–1376.

[6] K. L. Sadowski, P. A. Bosman, D. Thierens, On the usefulness of linkage processing for solving MAX-SAT, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation, ACM, 2013, pp. 853–860.

[7] D. Thierens, The linkage tree genetic algorithm, in: International Conference on Parallel Problem Solving from Nature, Springer, 2010, pp. 264–273.

[8] J. P. Martins, C. M. Fonseca, A. C. Delbem, On the performance of linkage-tree genetic algorithms for the multidimensional knapsack problem, Neurocomputing 146 (2014) 17–29.

[9] A. Gupta, Y.-S. Ong, L. Feng, Multifactorial evolution: toward evolutionary multitasking, IEEE Transactions on Evolutionary Computation 20 (3) (2016) 343–357.

[10] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, G. Proietti, On the Clustered Shortest-Path Tree Problem., in: ICTCS, 2016, pp. 263–268.

[11] D. Thierens, Linkage tree genetic algorithm: first results, in: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation, ACM, 2010, pp. 1953–1958.

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

[13] T. Back, Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms, Oxford university press, 1996.

[14] B. W. Goldman, D. R. Tauritz, Linkage tree genetic algorithms: variants and analysis, in: Proceedings of the 14th annual conference on Genetic and evolutionary computation, ACM, 2012, pp. 625–632.

[15] P. A. Bosman, D. Thierens, Linkage neighbors, optimal mixing and forced improvements in genetic algorithms, in: Proceedings of the 14th annual conference on Genetic and evolutionary computation, ACM, 2012, pp. 585–592.

[16] Y.-S. Ong, A. Gupta, Evolutionary multitasking: a computer science view of cognitive multitasking, Cognitive Computation 8 (2) (2016) 125–142.

[17] P. A. Bosman, D. Thierens, On measures to build linkage trees in LTGA, in: International Conference on Parallel Problem Solving from Nature, Springer, 2012, pp. 276–285.

[18] L. Feng, W. Zhou, L. Zhou, S. Jiang, J. Zhong, B. Da, Z. Zhu, Y. Wang, An empirical study of multifactorial pso and multifactorial de, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 921–928.

[19] T. Xie, M. Gong, Z. Tang, Y. Lei, J. Liu, Z. Wang, Enhancing evolutionary multifactorial optimization based on particle swarm optimization, in: Evolutionary Computation (CEC), 2016 IEEE Congress on, IEEE, 2016, pp. 1658–1665.

[20] Y.-W. Wen, C.-K. Ting, Learning ensemble of decision trees through multifactorial genetic programming, in: Evolutionary Computation (CEC), 2016 IEEE Congress on, IEEE, 2016, pp. 5293–5300.

[21] J. Zhong, L. Feng, W. Cai, Y.-S. Ong, Multifactorial Genetic Programming for Symbolic Regression Problems, IEEE Transactions on Systems, Man, and Cybernetics: Systems (99) (2018) 1–14.

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

[23] P. D. Thanh, D. A. Dung, T. N. Tien, 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.

[24] H. T. T. Binh, P. D. Thanh, T. B. Trung, 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.

[25] K. Helsgaun, Solving the Clustered Traveling Salesman Problem Using the Lin-Kernighan-Helsgaun Algorithm, Computer Science Research Report (142) (2011) 1–16.

[26] M. Mestria, L. S. Ochi, S. de Lima Martins, GRASP with path relinking for the symmetric euclidean clustered traveling salesman problem, Computers & Operations Research 40 (12) (2013) 3218–3229.

[27] P. D. Thanh, CluSPT instances, Mendeley Data v2, http://dx.doi.org/10.17632/b4gcgybvt6.2, 2018. doi:http://dx.doi.org/10.17632/b4gcgybvt6.2.