Article

An Adaptive Layered Clustering Framework with Improved Genetic Algorithm for Solving Large-Scale Traveling Salesman Problems

Haiyang Xu [1] and Hengyou Lan [1,2,*]

[1] College of Mathematics and Statistics, Sichuan University of Science and Engineering, Zigong 643000, China
[2] South Sichuan Center for Applied Mathematics, Zigong 643000, China
* Correspondence: hengyoulan@163.com

Abstract: Traveling salesman problems (TSPs) are well-known combinatorial optimization problems, and most existing algorithms are challenging for solving TSPs when its scale is large. To improve the efficiency of solving large-scale TSPs, this work presents a novel adaptive layered clustering framework with improved genetic algorithm (ALC_IGA). The primary idea behind ALC_IGA is to break down a large-scale problem into a series of small-scale problems. First, the k-means and improved genetic algorithm are used to segment the large-scale TSPs layer by layer and generate the initial solution. Then, the developed two phases simplified 2-opt algorithm is applied to further improve the quality of the initial solution. The analysis reveals that the computational complexity of the ALC_IGA is between $O(n \log n)$ and $O(n^2)$. The results of numerical experiments on various TSP instances indicate that, in most situations, the ALC_IGA surpasses the state-of-the-art algorithms in convergence speed, stability, and solution quality. Specifically, the ALC_IGA can solve instances with 2×10^5 nodes within 0.15h, 1.4×10^6 nodes within 1h, and 2×10^6 nodes in three dimensions within 1.5h.

Keywords: Computational complexity analysis; High parallelizability; Improved genetic algorithm; Adaptive layered clustering framework; Large-scale traveling salesman problem

1. Introduction

As an important branch of optimization, combinatorial optimization plays a significant role in management and economics, computer science, artificial intelligence, biology, engineering, etc [1]. The traveling salesman problems (TSPs) are main subject of combinatorial optimization problems, in which the goal is to find a closed route through all the cities once, and only once. This problem is equivalent to finding a Hamilton circuit with the minimum distance. The TSP, and its variants, such as asymmetric TSPs (ATSPs) [2], clustered TSPs (CTSPs) [3], dynamic TSPs (DTSPs) [4], multiple TSPs (MTSPs) [5], wandering salesman problems (WSPs) [6], have wide applications in laser engraving [7], integrated circuit design [8], transportation [9], energy saving [10], logistics problem [11], communication engineering [12], and medical waste transportation, which is closely related to COVID-19 pandemic [13]. The TSP was first considered in mathematical format in 1930 to solve a school bus routing problem, and then spread by researchers of Rand corporation. However, these problems were first considered only dozens of cities, but with the increase of applications, the scale of the problems may exceed millions [14].

Although the description of TSP is simple, it has been proven as NP-Hard, which means that the time required to obtain the exact solution for TSP will increase exponentially when the size of the problem aggrandizes. Lots of algorithms have been developed for TSPs, they can be split into three categories: exact methods, intelligence algorithms, and heuristics algorithms. The exact solver, such as brute-force search, linear programming [15], dynamic programming [16], brand and bound [17], brand and cut [18] and cutting plane [19] are powerful tools for small scale TSPs. However, the computational complexity of exact

 doi:10.20944/preprints202302.0412.v1

algorithm is very huge that solving the instance with 85900 nodes will take over 136 CPU-years by Concorde, which is a mature exact solver for TSPs [20]. Intelligence algorithms are inspired by the nature world and have high capabilities to approximate the global optimal for optimization problems. Evolutionary algorithm (EA) [21], ant colony optimization algorithm (ACO) [22], ant colony system (ACS) [23], shuffled frog leaping algorithm (SFLA) [24], simulated annealing algorithm (SA) [25], particle swarm optimization (PSO) [26] and other well-known algorithms [27,28] are all belong to intelligence algorithms. The novel intelligence algorithm can be employed to solve the problem with 2×10^5 nodes with high quality in an hour on a retail computer, but it is still hard to tackle while the scale is larger [29]. There are two main drawbacks of intelligent algorithms: one is that they frequently converge to the local optimum; the other one is that the parameters affect the solution quality deeply but usually can only be determined empirically [30]. The main heuristic algorithms for TSPs can be grouped into Lin–Kernighan family and stem-and-cycle family, they could provide high-quality solutions for nearly 2 million cities problems [31]. For higher quality solution and less running time, some researchers combined intelligence algorithms and heuristics algorithms, see [32–34] and the reference therein.

Genetic algorithm (GA) was proposed by Holland in 1975, the basic idea stems from "survival of the fittest" in evolutionism. Most types of GAs contain three main segments: selection operator, crossover operator and mutation operator. Due to the high effectiveness and versatility of GAs, they have been widely employed to solve TSPs and other challenging optimization problems [35,36]. However, there are still several doubts to TSPs, including premature convergence, population initialization, problem encoding, etc [37].

On the other hand, crossover operators have a significant influence on the performance of GA and are a key factor in global searching and convergence speed. As a matter of fact, various crossover operators have been proposed for TSP, including partially mapped crossover (PMX) [38], ordered crossover (OX) [39], cycle crossover (CX) [40], sequential constructive crossover operator (SCX) [41], completely mapped crossover operators (CMX) [42] and others based on heuristic algorithms such as bidirectional heuristic crossover operator (BHX) [43]. Additionally, merging GAs with local search or heuristic algorithms will reveal both of their advantages, including high convergence speed and the capacity for global optimization, therefore it has been a hot topic of study [32,44,45].

While the size of TSPs are larger than 10^5, seeking the exact solution is extremely difficult, and even a small improvement in quality can take a long time, the question of how to get an acceptable approximation solution in a reasonable time is more useful in real-world applications [46]. Thus, a new series of two-layered algorithms have been proposed, the fundamental concepts of them can be divided into two categories. The first type of them is to use various clustering techniques to divide the cities into small groups, calculate the sub-TSPs within those groups, and then merge the groups into a Hamilton cycle [47–49]. The other one is to determine the start and end points for each small group after clustering firstly, and then solve the fixed start and end points TSPs, which is also called WSPs, finally combine all the groups [50]. These algorithms are much faster than algorithms without clustering and can solve 180K size TSP within a few hours [7].

The drawbacks of the above two-layered algorithms include high computational complexity, poor accuracy, and problematic parameter tuning. Naturally, two-layered algorithms can be developed to three-layered or multiple-layered, very recent works can be seen in [51] and [52]. Admittedly, in order to fully utilize all the CPUs of computers, parallelizability is becoming extremely essential for algorithms designed to solve large and complicated problems. Some parallel algorithms for TSPs can be seen in [53,54].

In this paper, in order to develop a fast, easy implementation and high parallelizability algorithm for TSPs, an adaptive layered clustering framework with improved genetic algorithm (ALC_IGA) have been supposed. The key contributions of this study are as follows:

 doi:10.20944/preprints202302.0412.v1

- An improved genetic algorithm (IGA) integrated with hybrid selection, selective BHX crossover operator and simplified 2-opt local search has been proposed, a numerical comparison of IGA, GA and ACS on TSPs shows the high performance of IGA.
- Plentiful numerical results also prove the effectiveness of the novel IGA for solving the WSPs.
- An adaptive layered clustering framework is proposed to break down a large-scale problem into a series of small-scale problems. The computational complexity of the ALC_IGA is between $O(n \log n)$ and $O(n^2)$, also the parallelability of it has been discussed.
- We show a numerical experiment for parameters tuning of the proposed ALC_IGA, the results reveal that the larger the parameter set, the higher solution quality is obtained but a longer time is required.
- Dozens of two-dimensional Euclidean instances have been tested with ALC_IGA and some two-layered algorithms, and the results show that ALC_IGA has advantages in terms of accuracy, stability and convergence speed over two-layered algorithms.
- Lots of large-scale instances ranging in size from 4×10^4 to 2×10^5 have been tested, and the results show that the parallel ALC_IGA is times faster than the other three state-of-the-art algorithms and obtains the best solution in the most cases. The results on very large-scale TSPs, with sizes ranging from 2×10^5 to 2×10^6, also demonstrate the excellent effectiveness of ALC_IGA.

The remainder of the paper is organized as follows: a brief literature review of some related concepts is presented in Section 2; the main procedures of IGA are shown in Section 3; the details ALC_IGA are discussed in Section 4; the results of experimental analyses and algorithms comparisons are shown in Section 5; A summary of this paper and future works are listed in Section 6.

2. Literature review

Numerous algorithms have been introduced for the TSPs, the well-known and typical combinatorial optimization problem. The three primary kinds are heuristic algorithms, intelligent algorithms, and exact algorithms. Considering exact algorithms cannot be used for middle-scale TSPs, which is NP-Hard, the intelligence algorithms and heuristic algorithms have been the focus of attention. However, for large-scale TSPs, the classical intelligence algorithms lose efficacy, either. The two primary methods for large-scale TSPs by intelligence algorithms are improving intelligence algorithms and partitioning the large-scale problems into smaller ones by clustering. In this section, the well-known genetic algorithm is explored, and serval approaches based on clustering for large-scale TSPs are briefly reviewed.

2.1. Genetic algorithm for TSPs

GA is one of the intelligence algorithms that is widely applied to solve both continuous and discrete optimization problems. Grefenstette et al. [55] studied GA for TSPs in detail in 1988 and provided various proposals for further work, including merging GA with other heuristic algorithms and considering the impact of parameters. In the over 40 years that have passed ever since, the GA for TSPs has tremendous advancements in terms of representation, population initialization, fitness function, selection, crossover, mutation, and integrated with other algorithms.

First, when using GA, the primary task is to find a representation that closely relates to the structure of the problem. There are five different representations of TSPs: binary, path, adjacency, ordinal, and matrix. Larranaga et al. [56] reviewed representations and operators for TSPs. They concluded that the path representation performs well under most circumstances, and lots of powerful substantial crossover and mutation operators have been developed for it.

The crossover operator plays an important role in GA. A proper crossover operator could raise the average quality of the population, which would hasten convergence and

 doi:10.20944/preprints202302.0412.v1

save time. The most popular PMX was first proposed by Goldberg and Lingle in 1985 [38], in which each offspring only uses information from each of their parents partially. Firstly, generating two random cut points, and then the portions from parents between the two cut points are swapped to generate offspring. Then the other portions are complemented orderly from the original parents. Iqbal et al. [42] presented a new CMX in 2020, which differs from prior mapping crossover operators in that it uses cycle-based cut selections at the parental genes rather than random cuts. The numerical research suggests that the new CMX outperforms well-known crossover operators such as and PMX in middle-scale instances. In 2022, Zhang et al. [35] proposed a genetic algorithm with jumping gene and heuristic operators for TSPs, where the heuristic operators include 2-opt and BHX. The key distinction between the BHX and Grefenstette's heuristic algorithm is that the BHX always chooses the candidate that is closest to the present city out of the four possible candidates. According to the numerical study, the new algorithm converges far more speedily than the CMX and other latest crossover operators.

On the other hand, because the strengths of heuristic algorithms have been shown in TSPs, numerous studies attempt to use heuristic algorithms as crossover operators for GA. Grefenstette created a probability distribution in 1987 by using the distances between the chosen city and its four nearby neighbors [57]. Then chose the next visited city at random from this distribution until all cities were visited. Ulder et al. [58] presented a genetic local search framework in 1990, which could be combined with 2-opt, Lin-Kernighan neighborhoods, or any other heuristic algorithms. They concluded that although the new algorithms might not trump simulated annealing and threshold accepting, they can nevertheless be advantageously utilized for much larger problems.

Tsai et al. [59] proposed a genetic algorithm with a neighbor-join operator in 2002, and numerical experiments suggest that the new neighbor-join operator has lower error rates than 2-opt and swap operator combined with GA in all compared instances, and is nearly as efficient as 2-opt. In 2014, Wang [32] proposed a hybrid genetic algorithm for TSPs that combined two local optimization strategies. The computation results demonstrate that the hybrid genetic algorithm can achieve higher accuracy than the GA in a reasonable amount of time. However, this method is also sensitive to parameter settings. A list-based simulated annealing algorithm combined with tour construction algorithms and enhancement algorithms was developed as a hybrid genetic algorithm by Ilin et al. in 2022 [25]. The tour is built using the nearest insertion algorithm, the cheapest insertion method, and the other two techniques, and a 2-opt local search is used to improve the tour.

2.2. Layered-based algorithms for TSPs

Even though intelligence algorithms are becoming more sophisticated, they can only solve a TSP with 2×10^5 noes in 1h by using fast C++ programming and parallel techniques [29]. Because the small-scale TSPs can be solved efficiently and precisely, some researchers attempt to cluster the large-scale TSPs into a succession of small-scale TSPs. In this section, we give a summary of the advancements produced to the clustering-based (layered-based) algorithms.

As far as is known, Ding et al. [47] may be the first to employ the well-known *k*-means clustering algorithm for TSPs. The *k*-means algorithm is used to partition the large-scale cities into several small clusters, and a two-level GA is used to generate the final tour. The low-level GA is used to find the shortest Hamilton cycle inside each cluster, and the high-level GA is utilized to determine the in and out nodes of each cluster. The numerical experiment illustrates that the new algorithm handled the 1000 cities instance in 66 seconds on Matlab, which is substantially faster than the classical genetic algorithm. Due to the uneven distribution of cities, the scales of clusters produced by *k*-means may still be quite large, leading the low-layer computation to take a long time.

In 2009, Yang et al. [50] introduced an adaptive clustering method to reduce the computational complexity of the sub-clusters. It checks whether each cluster is smaller than the specified size after *k*-means and if so, repeats *k*-means until all clusters are smaller.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Then, a GA is used to find the visited order of the clusters based on the coordinates of the clusters' centers. Finally, the clusters are connected using the nearest nodes between the adjacent traveled clusters. The numerical experiment shows that the adaptive clustering method can solve an instance with 85900 cities in 1h. Although Yang's algorithm ensures that the low layer is solved quickly, there may be too many clusters produced, resulting in slow computation of the high level.

The influences of different clustering and intelligence algorithms combined for layered algorithms were first investigated by Phienthrakul [60] in 2014. He developed a greedy cluster connection procedure and then analyzed the influence of GA and ACO based on *k*-means and Gaussian mixer models. The numerical results show that the four algorithms have only minor differences in accuracy and execution time and can be efficiently applied to large-scale TSPs.

Although the notion of using a clustering method to solve large-scale TSPs has developed and grown, the work mentioned above does not verify the algorithms' efficacy for TSPs with more than 10^5 nodes. Wu et al. [7] investigated large-scale laser engraving in 2020, which is a widely used technology in modern production and can be represented as a TSP. They suggested a new two-layered ant colony system algorithm (TLACS) based on *k*-means, in which the ACS optimizes the visited order of clusters, and the start point and the end point for each cluster are determined. After the start point and the end point of each cluster have been determined, the local traveling path of each cluster can be depicted as a WSP. The ACS will then be used to find the shortest route of each groups. Finally, all groups are connected by the order and entrance and exit nodes, and the global path is determined. The numerical experiment shows that the TLACS can solve the large-scale TSPs with 2×10^5 nodes approximately in 1h.

Naturally, based on clustering algorithms, the two-layered method could be expanded to the three-layered. This concept was realized recently by Liang et al. [51]. Firstly, they applied *k*-medoids algorithm to divide the large-scale instance into some medium-scale groups, and then applied *k*-medoids algorithm for all medium-scale groups again to divide them into small-scale groups. The authors then proposed a three-layered evolutionary optimization framework comprised of two GAs and a parallel multifactorial evolutionary algorithm (3L-MFEA-MP). Their results show that three-layered algorithms have two main advantages over two-layered algorithms. One is speeding up the computation, while the other is that the three-layered algorithms reduce path length by almost 30% on four large-scale instances.

As can be seen, the global tour generated by the two-layered or three-layered algorithms is rough and unrefined, so a further optimization phase is necessary. In 2018, Liao and Liu [61] first applied the *k*-opt algorithm to optimize the tour generated by the hierarchical hybrid algorithm, which is a method proposed by them based on ACO and density peaks clustering algorithms. Although their results demonstrate that *k*-opt will significantly improve the proposed HHA's performance, the numerical experiments only test on the medium-scale instances that no more than 3038 cities. We remark that the computational complexity of *k*-opt is usually not affordable [62], so the direct application to complex issues is not feasible.

3. IGA for TSPs and WSPs

The GA is a popular optimization algorithm and is frequently applied to TSPs. As the main idea of the adaptive layered clustering framework is to break down a large-scale problem into a series of small-scale problems, GA is suitable for these sub-tasks. However, the poor convergence speed and accuracy of the traditional GAs will increase the total time consumption of the new framework. In this section, a novel IGA is introduced to fast and precisely solve small-scale TSPs and WSPs with the following key modifications: a hybrid selection algorithm is introduced; a selective bidirectional heuristic crossover is adopted to speed up the convergence; a hybrid mutation operator is suggested to jump the local

 doi:10.20944/preprints202302.0412.v1

optimal; a simplified 2-opt is used to balance the convergence speed and global searching capability.

3.1. Path encoding and population initialization

Path encoding is the fundamental task involved in using GA. Due to the conclusion in [56], one of the most intuitive and high-performance route encoding methods for TSPs is path representation. In path representation, all cities are encoded as unique integers and arranged into a chromosome. The position in the chromosome indicates the visited order of the city, that is for $i, j = 1, 2, \cdots, n$, if city i is the j-th element in a chromosome, then city i is the j-th to be traveled. The initial population will impact both the rate of convergence and the capacity of global searching for GA. In this study, the initial population is generated randomly, and then a 2-opt local search is applied to improve the quality of the initial population.

3.2. Fitness function and selection operator

The role of the selection operator is to choose some eligible chromosomes for the next generation; a decent selection operator will help to converge rapidly and prevent local optimal, but a poor one will not. Because the objective values of TSPs are not stable, a proper transformation for the objective values is required, which is called the fitness function[35].

Assuming there are N individuals in the population, C_i is the i individual, $L(C_i)$ represents the tour length of C_i, $f(C_i)$ denotes the fitness value of C_i. Some well-known fitness functions are as follows:

- Reciprocal-based fitness function is one of the most used fitness, it is the reciprocal of objective function value:
$$f(C_i) = 1/L(C_i). \tag{1}$$
- Linear order-based fitness function that sorts individuals in ascending order by objective function values, where $R(C_i)$ denotes the order of C_i. Then $f(C_i)$ presented by:
$$f(C_i) = \frac{N - R(C_i)}{N}. \tag{2}$$
- Nonlinear order-based fitness function also sorts the individuals, but $f(C_i)$ defined by:
$$f(C_i) = \alpha(1-\alpha)^{R(C_i)-1}, \tag{3}$$
where α is a constant in $[0.01, 0.3]$.

Some deserving individuals will be picked for the following generation once all the fitness values of individuals have been evaluated. Once all fitness values of individuals have been confirmed, some good individuals will be selected for the next generation. The most common selection method is roulette wheel selection. If M individuals must be chosen for the next generation, the main steps are as follows:

Step 1: Calculate the selection probability of C_i:
$$p(C_i) = \frac{f(C_i)}{\sum_{j=1}^{2N} f(C_j)}. \tag{4}$$

Step 2: Generate a random number P between 0 and 1.

Step 3: Select the first C_j satisfied $P \leq \sum_{h=1}^{j} p(C_h)$.

Step 4: Remove C_j from the population, then return to the first step until all N individuals are selected.

The pseudo-code of the proposed hybrid selection algorithm is shown in Algorithm 1.

Algorithm 1 Hybrid selection algorithm

Input: A set of N individuals, the number of selected requirements M, current iteration number of GA $Iter$.
Initialize parameters: $\alpha = 0.15$, and r_1, r_2 are two random numbers.
Output: A set of M selected individuals.

```
Calculate the objective value for each individuals.
if r_1 ≥ 1/Iter then
    if rand ≥ r_2 then
        Calculate fitness values by nonlinear order-based fitness function.
    else
        Calculate fitness values by linear order-based fitness function.
    end if
    Select M individuals by roulette wheel selection.
else
    Calculate fitness values by reciprocal-based fitness values.
    Select M individuals according to the smallest fitness values.
end if
```

3.3. Selective bidirectional heuristic crossover

The crossover operation is the primary role of GA in producing new offspring. As stated in Section 2.1, there are numerous crossover operators proposed for path representation. Recently, Zhang et al. [35] presented a novel BHX, and the numerical results show its excellent effectiveness in enhancing the quality of the offspring.

The drawback of the BHX is that two parents will only have one unique offspring, which will reduce the size of the population gradually. Hence, a method of enriching the population must be developed to use BHX. As all know that monogamy is not the only type of mating system in nature, polygynandry is another prevalent mating system in species that live in troupes. An individual can mate with several individuals, and the number of mates is governed by individual quality. Inspired by the polygynandry mating system, a selective bidirectional heuristic crossover (SBHX) has been developed, in which the good gene of a parent may be preserved for two or more offspring.

Assuming there are N individuals in the current population, M offspring should be created. The main steps of SBHX are as follows:

Step 1: The fitness values of individuals are computed according to the reciprocal-based fitness function Eq. (1).

Step 2: The probability that an individual will be selected is determined by Eq. (4).

Step 3: The roulette wheel selection is used to choose two individuals C_1 and C_2 based on the probability distribution.

Step 4: The start and end points are connected in C_1 and C_2, and then each chromosome becomes a ring. Let O_1 and O_2 represent the two rings.

Step 5: Randomly generate a start city s between 1 and n, and a blank offspring C_{next}.

Step 6: Starting from s in O_1, searching for the first city O_1^r that C_{next} has not yet visited on the right; O_1^l on the left. It is the same for O_2, remark the two cities as O_2^r and O_2^l.

Step 7: Compute the distance between s and the four feasible cities found by Step 6. Then choose the nearest city to s and replace s as the selected city.

Step 8: Return to Step 6. until C_{next} has been filled. Then the C_{next} is a offspring generated by C_1 and C_2.

Step 9: Continue with Step 3 until all M offspring are determined.

3.4. Mutation operator

The mutation operator is another important phase of GA. Similar to how genetic mutations never stop happening and are essential to biodiversity, the mutation operator also enriches population diversity, which prevents the GA from falling into a local optimal. Lots of swap, inversion and heuristic mutation operators have been applied in GA for TSPs, see [32,43]. Suppose that there are n cities in the i-th individual C_i, To employ the swap or inversion mutation operator, two integers p_1 and p_2 between $[1, n]$ will be generated firstly. In the swap operator, the two cities $C_i^{p_1}$ and $C_i^{p_2}$ are exchanged. In the inversion operator, the gene fragmentation between p_1 and p_2 is reversed.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

As the heuristic mutation operators usually have high computational complexity, a hybrid mutation operator combined with a swap mutation operator and inversion mutation operator is proposed in this paper. Firstly, a mutation probability is set by hand, and then if individual C_i has a chance to be mutated, the probability will control which mutation operator will be selected. The pseudo-code of the new mutation operator is shown in Algorithm 2.

Algorithm 2 Hybrid mutation operator

Input: A population of N individuals.
Initialize parameters: The probability p of mutation, the probabilities r_1 and r_2 to select of mutation operator, $r_1 \geq r_2$.
Output: The population after mutation.

```
for C_i in population do
    if rand < p then
        Randomly generated q ∈ [0, 1].
        if q > r_1 then
            The swap mutation operator is used for C_i.
        else if q > r_2 then
            The inversion mutation operator is used for C_i.
        else
            Continue.
        end if
    end if
end for
```

3.5. Simplified 2-opt local optimization

k-opt is a well-known class of local optimization algorithms, here k is an integer greater than 1. The first proposed and simplest algorithm of them is 2-opt, which was developed by Croes [63] for solving TSPs in 1958. Although k-options have better quality than 2-options when $k > 2$, they involve high computational complexity. The 2-opt local optimization applied in GA can improve the quality of the current population and speed up the convergence under suitable parameters set. However, since BHX and 2-opt are heuristic algorithms with drawbacks in searching the global optimal, combining them will almost certainly result in premature convergence. In the proposed improved genetic algorithm, a simplified 2-opt (S_2-opt) is developed to enhance the quality of individuals after mutation. The pseudo-code of the S_2-opt for GA is shown in Algorithm 3. The simplified 2-opt operator has a simple iterative structure, and only one parameter must be set. It can avoid the local optimal by setting T to a small value or achieve a fast convergence speed by setting T to large.

Algorithm 3 Simplified 2-opt for GA

Input: A population of N individuals.
Initialize parameters: Max iteration T for simplified 2-opt, the number n of cities in each individual.
Output: The optimized population.

```
for C_i in population do
    for h = 1 to T do
        Calculate the tour distance d_1 of C_i.
        Randomly generated p_1 and p_2 in [1, n].
        Inverse the gene fragment between p_1 and p_2, set as C_new.
        Compute the tour length d_2 of C_new.
        if d_1 > d_2 then
            Replace C_i by C_new.
        end if
    end for
end for
```

The main flow of the IGA can be seen in Figure 1, the stop condition of IGA is set as no improvement of solution in specified iterations. Since the WSP is a TSP with fixed start and end nodes, it can be solved as a TSP by setting the distance between the start node and

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

the end node to $-M$, where M is a large positive number [6]. With the help of this feature, the proposed IGA can also be employed to solve WSPs.

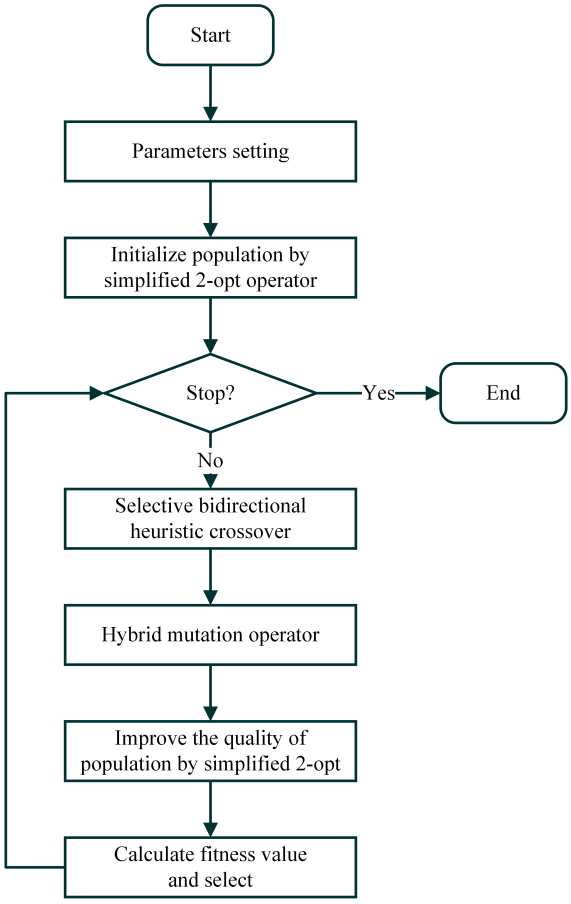


Figure 1. Flowchart of the proposed IGA.

4. The framework of ALC_IGA for large-scale TSPs

In recent years, some two-layered algorithms have been proposed, and they significantly reduce the time expenditure for large-scale TSPs. Liang et al. [51] recently proposed a three-layered algorithm with k-means and indicated that it outperforms two-layered algorithms by numerical experiments. Notwithstanding, both two-layered and three-layered algorithms may still have medium-scale or large-scale groups. Naturally, this will require a significant amount of time to solve the underlying problems. Thus, upgrading the two-layered and three-layered algorithms to the adaptive layered algorithm stands to reason.

We propose a brand-new framework for adaptive layered clustering that takes into account the IGA created in the previous section. The framework is divided into two parts: the first is applying clustering and IGA to initialize the solution, and the second is optimizing the initial solution. Based on our new algorithm, the large-scale TSPs can be transformed into solving some TSPs and WSPs that are smaller than the specified size. The processing flows are illustrated in Figure 2, and the details of solution initialization and optimization are represented subsequently in Sections 4.1 and 4.2.

4.1. Solution initialization

Assuming that there are N cities in a large-scale problem G, the cities are designated by $c_1, c_2, \cdots, c_N$, and $d(a, b)$ denotes the distance between vectors a and b, then the proposed ALC_IGA processes by following steps:

Step 1: Specify a positive integer M; if the size of the TSPs is smaller than M, then the IGA can solve it no more than T_1 seconds in most cases.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

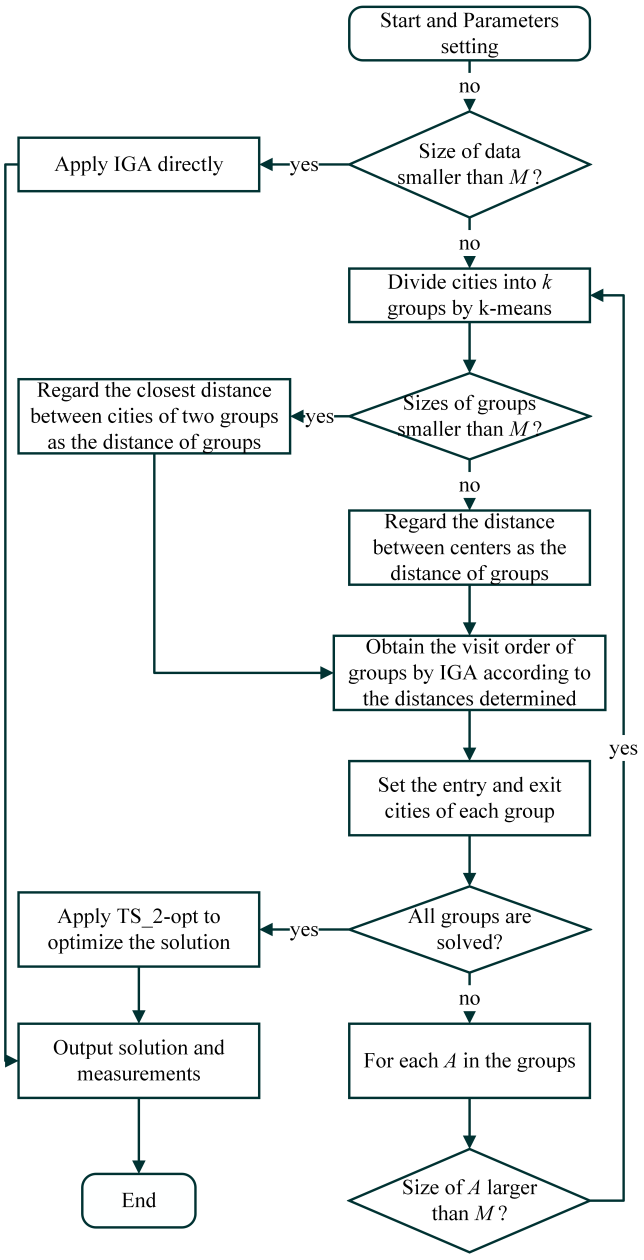


Figure 2. Main steps of the novel ALC_IGA.

Step 2: The k-means algorithm is used to cluster the N cities into k_1 clusters, where k_1 is an integer no greater than M. Then there are k_1 groups $\{G_1, G_2, \cdots, G_{k_1}\}$, and the coordinate vectors of centers for the groups are $\{V(G_1), V(G_2), \cdots, V(G_{k_1})\}$, the size of groups denote as $\{S(G_1), S(G_2), \cdots, S(G_{k_1})\}$.

Step 3: Determine the traveling order of the groups; this is referred to as a TSP. If all the sizes of the groups are less than D_1, here D_1 is a positive integer, then the distance between G_i and G_j is defined as the minimum distance between two points that belong to G_i and G_j respectively. Otherwise, the distance is set to the distance between the centers of groups. Then there is a distance matrix that can be used to solve the k_1 cities TSP using IGA. Make a note of the visited order as $\{O(G_1), O(G_2), \cdots, O(G_{k_1})\}$.

Step 4: Determine the entry city G_i^{entry} and the exit city G_i^{exit} for each group. Based on the order of visits came by Step 3., assume G_j is visited directly after G_i, then the exit of G_i is the closest city of G_i to G_j, the entry city of G_j is the closest city of G_j to G_i.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Step 5: After determining the entry and exit cities in each group, evaluating the shortest route from entry to exit of each group is a WSP. If the size of G_i is smaller than M, the IGA is used to find the optimal path $P(G_i)$ of G_i; if the size of G_i is greater than M, the k-means algorithm is applied for G_i again to reduce the computational complexity, see Step 6.

Step 6: If the size of G_h is greater than M, then divide G_h into k_{1h} groups $\{G_{h1}, G_{h2}, \cdots, G_{hk_{1h}}\}$, here $k_{1h} \leq M$, denote the coordinates of centers for each group as $\{V(G_{h1}), V(G_{h2}), \cdots, V(G_{hk_{1h}})\}$. The difference between this step and Step 2 is that the entry and exit cities of G are not specified. If G_{hi} includes G_h^{entry} and G_{hj} includes G_h^{exit}, then G_{hi} is set as the start group, and G_{hj} is set as the end group. Then finding the optimal route from G_h^{entry} to G_{hj} is a WSP, not the TSP in step 3. The distance between G_{hi} and G_{hj} is the same as defined in Step 3. As the distance is determined, the IGA is applied to find the visit order $\{O(G_{h1}), O(G_{h2}), \cdots, O(G_{hk_{1h}})\}$.

Step 7: Return to Step 4 until all the sizes of groups are smaller than M, then the visited order of each group is determined, and the optimal path of each group is recorded.

Step 8. Combine the optimum path of each group by the order in each layer from the bottom to the top layer.

An example of 100 cities TSP and M set to 20 is shown in Figure 3. In the first layer, the cities have been divided into to groups G_1, G_2 by k-means, then the visit order found by IGA is $O(G_1) = 1$ and $O(G_2) = 2$. On the one hand, since the size of G_2 equals M, the visit route $P(G_2)$ of the 20 cities in G_2 could be solved by IGA quickly. On the other hand, because there are 80 cities in G_1, that is larger than M, so G_1 need to be divided into small groups again. Repeat the procedures until all of the group sizes are less than M, resulting in 6 groups and 4 layers being determined during the solution initialization phase. To combine the six routes, first from the bottom layer, connect $P(G_{1311})$ with $P(G_{1312})$ sequentially, and obtain $P(G_{131}) = \{P(G_{1311}), P(G_{1312})\}$. Then in the third layer, connect $P(G_{131})$ with $P(G_{132})$, then $P(G_{13}) = \{P(G_{131}), P(G_{132})\}$. Following these steps, the path for the 100 cities TSP is eventually $\{P(G_{1311}), P(G_{1312}), P(G_{132}), P(G_{11}), P(G_{12}), P(G_2)\}$.

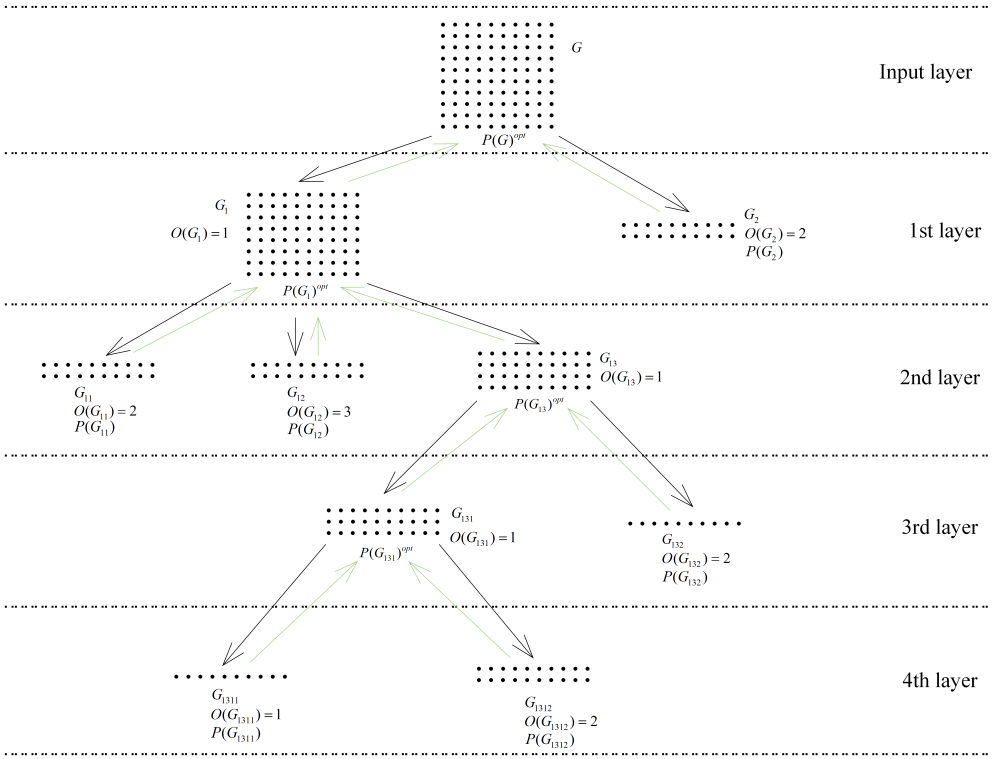


Figure 3. An example of the ALC_IGA on a 100 nodes instance. The black lines represent the solution initialization phase, and the green lines denote the solution optimization phase.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

4.2. Two phases 2-opt for solution optimization

Because of the clustering algorithm used, even if the routes in each cluster are optimal, the entire path is nearly impossible to achieve optimality. In [61], Liao and Liu first applied the 2-opt and 3-opt operators to optimize the initial route while the clustering algorithm involved, and the numerical studies show a marked improvement when k-opt is used. Nevertheless, when the number of cities in the problem is exceptionally enormous, the k-opt struggle to work.

To improve the quality of the initial solution in an affordable time, a two phases simplified 2-opt algorithm (TS_2-opt) is given in Algorithm 4. The TS_2-opt is aimed to optimize the routes and orders of all the groups which belong to a cluster at a higher layer. Once the solution is initialized, TS_2-opt is used to optimize the route of each group in the penultimate layer and repeated layer by layer until the top layer is optimized. Depicted in Figure 3, the green lines show the workflow of solution optimization. Firstly, from the bottom layer, the routes $P(G_{1311})$ and $P(G_{1312})$ are combined by TS_2-opt to the local optimal routes $P(G_{131})^{opt}$. Then the two routes in the third layer also are optimized to $P(G_{13})^{opt}$ by using TS_2-opt. Following these steps, until the final solution $P(G)^{opt}$ is obtained.

Algorithm 4 Two phases simplified 2-opt algorithm

Input: A batch of groups $\{G_{i\cdots j1}, G_{i\cdots j2}, \cdots, G_{i\cdots jh}\}$, suppose the order of them is $1, 2, \cdots, h$, and the travel routes of them $\{P(G_{i\cdots j1}), P(G_{i\cdots j2}), \cdots, P(G_{i\cdots jh})\}$.
Initialize parameters: The first phase max iteration L_1; the second phase max iteration L_2; the length selected for optimization R.
Output: An optimized route $P(G_{i\cdots j})$ for $G_{i\cdots j}$.

Compute the distance d_{bks} of the tour $\{P(G_{i\cdots j1}), P(G_{i\cdots j2}), \cdots, P(G_{i\cdots jh})\}$.
for $iter_1 = 1$ to L_1 **do**
  Randomly generated two different integers p_1, p_2 between $[2, h-1]$.
  Denote the route between $G_{i\cdots jp_1}$ and $G_{i\cdots jp_2}$ as $P_{p_1}^{p_2}$; denote the route between $G_{i\cdots j1}$ and $G_{i\cdots jp_1-1}$ as $P_1^{p_1-1}$; denote the route between $G_{i\cdots jp_2}$ and $G_{i\cdots jh}$ as $P_{p_2+1}^{h}$.
  Inverse $P_{p_1}^{p_2}$, denote the new route as $Inv(P_{p_1}^{p_2})$.
  Generate two routes P_1 and P_2, where P_1 is combined by the last R elements of $P_1^{p_1-1}$ and the first R elements of $Inv(P_{p_1}^{p_2})$; P_2 is combined by the last R elements of $Inv(P_{p_1}^{p_2})$ and the first R elements of $P_{p_2+1}^{h}$. Denote the new order of groups as $\{O(G_{i\cdots j1}), O(G_{i\cdots j2}), \cdots, O(G_{i\cdots jh})\}$, the sizes of groups is noted as $\{S(G_{i\cdots j1}), S(G_{i\cdots j2}), \cdots, S(G_{i\cdots jh})\}$.
  The Algorithm 3 with max iteration number L_2 is applied to optimize P_1 and P_2. Denote the new routes as P_1^{opt} and P_2^{opt}.
  Replace P_1 and P_2 in $\{P_1^{p_1-1}, Inv(P_{p_1}^{p_2}), P_{p_2+1}^{h}\}$ with P_1^{opt} and P_2^{opt}, respectively. Denote the new route as P_{opt}.
  Compute the distance d_{opt} of P_{opt}.
  if $d_{bks} > d_{opt}$ **then**
    Assign d_{opt} to d_{bks}.
    Divide P_{opt} into h segments $\{P_{m_1}, P_{m_2}, \cdots, P_{m_h}\}$, here $S(P_{m_k})$ is equal to $\{S(G_{i\cdots jr})|r = m_k\}$.
  end if
  Replace $\{P(G_{i\cdots j1}), P(G_{i\cdots j2}), \cdots, P(G_{i\cdots jh})\}$ by $\{P_{m_1}, P_{m_2}, \cdots, P_{m_h}\}$.
end for
Output $R = P_{opt}$.

Suppose there are three groups $\{G_{11}, G_{12}, G_{13}\}$ belong to the same higher group G_1, and the visit orders of them are $\{2, 3, 1\}$ respectively. Figure 4 illustrates the major processing of TS_2-opt in detail. Each cluster is represented by a different color, while the start and end locations are marked by larger shapes. In Step 1, the three routes are arranged by order and assume the G_{11} is chosen, then the path of G_{11} is inverted. In Step 2, the segments at the junctions of the clusters are determined according to R, where R equals 5 for simplicity. The next step is to optimize the two segments provided by Step 2. In Step 4, three new routes are generated according to Step 3 and the input routes. Once all four steps have been completed, return to Step 1 until the termination condition is met.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

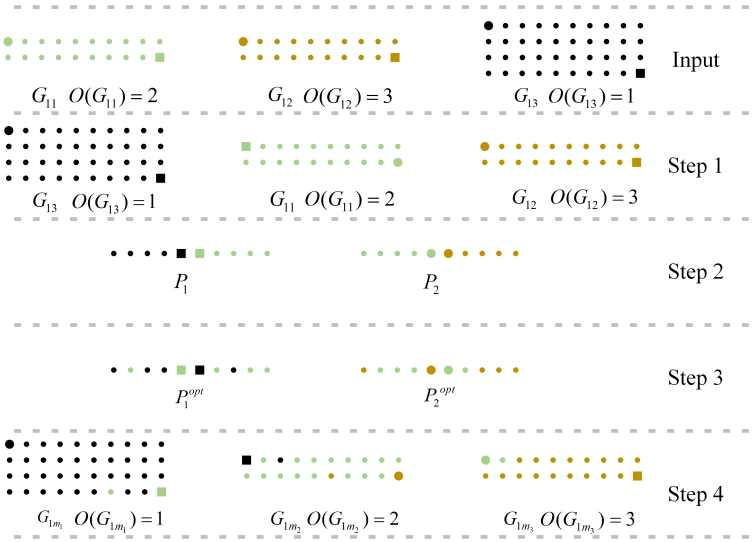


Figure 4. The major processes of TS_2-opt for optimizing three subgroups.

We remark that the purpose of the TS_2-opt is not to reach the global optimal, but rather to optimize the visit orders and junctions between groups that belong to the same group at the higher layer. Despite sacrificing some precision, the computation speed of TS_2-opt is very fast, which is critical in large-scale TSPs.

4.3. Parallelability and computational complexity analysis

We show the highly parallelizable capability of the proposed ALC_IGA. In the phase of solution initialization, the operations for clusters are independent in each layer; the operations of subgroups that do not belong to the same cluster in different layers are also independent. As an illustration, there are three tasks in the third layer shown in Figure 3, find the visit route for G_{11} and G_{12}, and apply k-means to divide G_{13} into small groups. As they are stand-alone, if there are three or more cores of the CPU, they can be computed on different cores simultaneously. Furthermore, if k-means is faster than the other two tasks, then the computations of G_{131} and G_{132} in the next layer can also be allocated to the free cores even if $P(G_{11})$ and $P(G_{12})$ are still being calculated.

In the second phase of ALC_IGA, solution optimization also can be parallelized, but the parallel effectiveness is not as high as in the first phase. Firstly, the complex calculation in solution optimization is only the optimization of the junctions, but there are only two junctions in each iteration, so parallel computing is unnecessary. Secondly, the optimization of the solution starts from the bottom and ends at the top layer, but the higher-layer optimizations must wait for lower-layer optimizations to finish. As the example shown in Figure 3, there is only one task in the fourth layer, which is connecting G_{1311} and G_{1312}. Because the route of G_{131} is not determined before the computation of the fourth layer is finished, the free cores can not be used to combine G_{131} and G_{132} in the third layer.

Notwithstanding, parallel techniques can be used in each layer to speed up computation while the scale of the problem is very large. The computational complexity of the major stages of the proposed ALC_IGA is presented in the remainder of this section.

For the sake of simplicity, it is assumed that there are n cities. First and foremost, the times of TS_2-opt used is no more than n, and the execution time of TS_2-opt is bounded if the parameters are set. Thus the complexity of the solution optimization phase is less than $O(n)$. The computational complexity analysis of solution initialization can be split into two categories: best and worst. Considering the best-case scenario, if $n = m^k$, where k is a positive integer, assume that each cluster's size equals m after clustering. Then the procedures of solution initialization include $\frac{n-1}{m(m-1)}$ times clustering and $\frac{n-1}{m-1}$ times IGA. Once if the parameters of IGA are determined, the running time of IGA is less than T_1, and the running time of IGA used in the first phase is less than $\frac{n-1}{m-1}T_1$. It is well known that

 doi:10.20944/preprints202302.0412.v1

the computational complexity of k-means is $O(pq)$, where p are the size of data and q is the number of cluster centers. Therefore the computational complexity of the first time k-means used is $O(mn)$, and the computational complexity of k-means used in the second layer is $mO(n)$, and the last layer is $m^{k-2}O(m^2m)$. According to $n = m^k$, the complexity of k-means implemented in the ALC_IGA is $O(mn \log_m \frac{n}{m})$, that is quasilinear computational complexity. Consequently, the whole computational complexity is $O(n \log n)$ for ALC_IGA in the best condition.

In the worst possible scenario, each clustering results in $m-1$ groups that each contains just one city and one group that contains all the remaining cities. Suppose $n = k(m-1)+m$, then there will be k times clustering and $k+1$ times IGA. The time of IGA applied is no more than $\frac{n-m}{m-1}T_1$, it is $O(n)$. Similar to the best-case analysis, the computational complexity of clustering in the worst condition is $O(n^2)$. Accordingly, the computational complexity of ALC_IGA in the worst condition is $O(n^2)$.

In summary, the computational complexity of the ALC_IGA ranges from $O(n \log n)$ to $O(n^2)$. The computational complexity of ALC_IGA is closer to $O(n \log n)$, however, in the majority of cases. This is supported by the numerical experiments presented in Section 5.

5. Numerical results and discussions

Four-part numerical experiments are presented in this paper to illustrate the effectiveness of ALC_IGA. First, Section 5.4 proves that IGA is substantially superior to GA and ACS in terms of accuracy and convergence speed. The implications of the primary parameter setting performance on ACL_IGA are examined in the second part. The third part proves the superiority of ALC_IGA on middle-scale benchmark datasets over two two-layered algorithms from the literature. The last part proves the excellent performance and parallelizability of the proposed ALC_IGA in comparison to some state-of-the-art algorithms.

5.1. Experimental setting

In this study, all experiments were computed on a Dell PowerEdge R620 with two Intel Xeon E5-2680V2 10-cores processors and 64.0 GB of 1066 MHz DDR3 memory under Windows 10 OS. The speed of all cores is locked to 2.80 GHz without turbo boost technology and disable hyperthreading to ensure the fairness and stability of numerical experiments. All the programs are edited and run on MATLAB R2020a, the only parallel technique used is the parallel computing toolbox in MATLAB. By default, each instance was computed 20 times under the same setting. In detail, if the algorithm is single-threaded, execute the instance on 20 cores simultaneously; if the algorithm is multi-threaded, run them one by one. The sources of GA, ACS [23], IGA, two-level genetic algorithm (TLGA) [47], TLACS [7], and ALC_IGA are published on GitHub [1], and the instances involved are also on this repository.

5.2. Benchmark instances

Numerous instances are used to study the effectiveness of the proposed IGA and ALC_IGA. The major instances come from three sources: the famous benchmark TSP datasets TSPLIB [2]; the TSP Test Data gathered by William Cook for large instances [3]; hard to solve instances of the Euclidean TSPs (TNM) [64]. The TSP Test Data used in this research can be divided into three categories: National TSPs; VLSI TSPs; Art TSPs. And the TNM data generated by the C++ source provided by the authors of [64]. A two-dimensional Santa [4] and a three-dimensional Gaia [5] with millions of nodes, also be investigated.

1 https://github.com/nefphys/tsp
2 http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
3 https://www.math.uwaterloo.ca/tsp/data/index.html
4 http://cs.uef.fi/sipu/santa/
5 https://www.math.uwaterloo.ca/tsp/star/gaia1.html

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

For various experimental tasks, the instances are classified into three categories: small-scale TSPs ($n \leq 500$), medium-scale TSPs ($500 < n \leq 4 \times 10^4$), and large-scale TSPs ($n > 4 \times 10^4$). Small-scale TSPs are used to study the effectiveness of IGA; middle-scale TSPs are employed to tune parameters and compare ALC_IGA with TLACS and TLGA in a single thread; large-scale TSPs are adopted to compare ALC_IGA with some state-of-the-art algorithms in parallel and verify the efficiency of ALC_IGA.

5.3. Evaluation criteria

The following are the evaluation criteria for the algorithmic analyses on instances:

- The minimum objective value among all runs R_{best}.
- The average objective value among all runs R_{avg}.
- The standard deviation of results among all runs R_{std}.
- The best known solution of the instance BKS.
- The deviation percentage of R_{best} is defined by:

$$PD_{best} = \frac{R_{best} - BKS}{BKS} \times 100\%. \tag{5}$$

- The deviation percentage of R_{avg} is defined by:

$$PD_{avg} = \frac{R_{avg} - BKS}{BKS} \times 100\%. \tag{6}$$

- The running time T_{Rb} in seconds while R_{best} found.
- The average of the running time in seconds among all runs T_{avg}.
- The count of the best R_{best}, R_{avg}, R_{std} and T_{avg} are denoted as C_{Rb}, C_{Ra}, C_{std}, C_{Ta}.

5.4. Performance comparison of IGA, GA and ACS

In addition to clustering, the most time-consuming part of ALC is eliminating the sub-TSPs. That is why the IGA proposed. To illustrate that IGA is efficient on TSPs, a comparison of IGA, GA, and ACS is imperative, and 42 small-scale benchmark instances are used in this numerical comparison. The parameters setting of IGA were as follows: the population was set to 0.4 times the number of nodes; the maximum number of iterations for S_2-opt was set to 20 times the number of nodes; the parameters of selection operator, r_1 and r_2, were set to 0.15 and 0.5; the probability of mutation was set to 0.05. The population size of GA was set to 0.8 times the size of the instance and the mutation number was always set at three individuals. The parameters setting of ACS is as same as the literature [7]. Finally, the termination condition for the three compared algorithms is when there has been no improvement in the population for X iterations. In this experiment, X were set to 100, 100, and 10^4 for IGA, ACS and GA, respectively. The results of the comparison are displayed in Table 1, various evaluation criteria are considered, include R_{best}, PD_{best}, R_{avg}, PD_{avg}, R_{std}, T_{Rb}, T_{avg}, $C_{Rb}/C_{Ra}/C_{std}/C_{Ta}$ and the average value for PD_{avg}, R_{std} and T_{avg}. The best value of R_{best}, PD_{best}, R_{std} and T_{avg} are set in bold.

Table 1. Results obtained by IGA, GA and ACS on 42 small-scale instances.

Instance		IGA			GA			ACS		
Name	BKS	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}
eil51	426	**426 (0)** 426.85 (0.2)	**0.37**	1.94 **1.72**	428 (0.47) 436 (2.35)	3.18	9.02 11.42	427 (0.23) 430.95 (1.16)	4.03	3.8 2.56
berlin52	7542	**7542 (0)** 7542 (0)	**0**	1.81 **1.71**	**7542 (0)** 7836.95 (3.91)	206.62	8.85 6.14	**7542 (0)** 7600.25 (0.77)	103.39	3.56 2.3

continued on next page

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 1. *(continued).*

Instance		IGA			GA			ACS		
Name	BKS	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}
st70	675	**675 (0)** 676.65 (0.24)	**3.1**	3.49 **2.88**	**675 (0)** 689.45 (2.14)	8.3	16.28 17.56	682 (1.04) 696.4 (3.17)	7.2	5.91 4.39
pr76	108159	**108159 (0)** 108611.3 (0.42)	**465.2**	4.89 **3.6**	108936 (0.72) 113302.85 (4.76)	3423.57	18.91 20.53	112647 (4.15) 113573.65 (5.01)	657.37	7.05 7.3
eil76	538	**538 (0)** 540.3 (0.43)	**2.6**	4.12 **3.57**	549 (2.04) 558.65 (3.84)	8.43	16.76 26.14	539 (0.19) 546.25 (1.53)	4.22	7.8 9.55
rat99	1211	**1211 (0)** 1217.25 (0.52)	**5.4**	6.35 **5.92**	1230 (1.57) 1276.5 (5.41)	19.24	25.94 23.33	1229 (1.49) 1239.05 (2.32)	6.95	15.89 15.39
kroA100	21282	**21282 (0)** 21327 (0.21)	**49.28**	5.94 **5.53**	21389 (0.5) 22134.75 (4.01)	510.46	29.03 21.58	21867 (2.75) 22310.65 (4.83)	246.01	16.23 10.49
rd100	7910	**7910 (0)** 7917.3 (0.09)	**12.88**	6.04 **5.66**	7965 (0.7) 8332.3 (5.34)	181.42	29.27 40.3	8074 (2.07) 8195.65 (3.61)	80.08	16.68 23.02
eil101	629	**630 (0.16)** 636.45 (1.18)	**4.49**	7.92 **5.76**	638 (1.43) 658.35 (4.67)	7.73	29.4 22.05	635 (0.95) 661.2 (5.12)	11.26	15.31 16.56
lin105	14379	**14379 (0)** 14414.05 (0.24)	**43.05**	7.82 **5.99**	14531 (1.06) 15080.8 (4.88)	319.22	31.11 26.73	14486 (0.74) 14596.25 (1.51)	60.69	17.37 14.62
pr107	44303	**44303 (0)** 44460.9 (0.36)	**119.04**	9.49 **6.88**	44577 (0.62) 45283.25 (2.21)	728.56	33.58 38.55	44707 (0.91) 45054.75 (1.7)	198.61	15.89 13.55
pr124	59030	**59030 (0)** 59357.15 (0.55)	**270.36**	10 **11.51**	59838 (1.37) 60725.3 (2.87)	746.56	40.4 35.17	59210 (0.3) 59664.95 (1.08)	326.31	22.74 22.61
bier127	118282	**118423 (0.12)** 118982.65 (0.59)	**352.24**	14.27 **15.57**	120538 (1.91) 124348.1 (5.13)	2110.45	55.57 46.66	121306 (2.56) 122591 (3.64)	643.63	21.38 20.44
ch130	6110	**6128 (0.29)** 6178.45 (1.12)	32.35	13.4 **12.96**	6221 (1.82) 6397.35 (4.7)	87.47	55.16 66.31	6292 (2.98) 6331.55 (3.63)	**32.14**	26.49 21.52
xqf131	564	**565 (0.18)** 575.05 (1.96)	**3.71**	13.32 **11.29**	577 (2.3) 594.85 (5.47)	10.46	48.99 46.33	593 (5.14) 599.3 (6.26)	4.26	30.3 63.67
pr136	96772	**96870 (0.1)** 97810.2 (1.07)	691.4	23.05 **20.19**	97605 (0.86) 100223.55 (3.57)	1340.64	68.59 75.11	105463 (8.98) 106761.45 (10.32)	**657.71**	30.5 **19.16**
pr144	58537	**58537 (0)** 58561.15 (0.04)	**23.66**	22.16 16.88	58746 (0.36) 60252.7 (2.93)	1379.65	62.2 48.28	58701 (0.28) 58824.15 (0.49)	87.31	30.3 46.68
kroA150	26524	**26583 (0.22)** 26758.25 (0.88)	**137.74**	19.81 **18.18**	27276 (2.84) 28026.55 (5.66)	499.34	71.77 71.92	27840 (4.96) 28334.55 (6.83)	224.01	43.56 59.19
ch150	6528	**6533 (0.08)** 6556.85 (0.44)	**8.55**	14.91 **12.31**	6697 (2.59) 6914.5 (5.92)	180.44	78.56 84.22	6720 (2.94) 6758 (3.52)	28.95	35.78 29
pr152	73682	**73682 (0)** 73968.05 (0.39)	**207.17**	16.63 **16.26**	74424 (1.01) 75970.1 (3.11)	983.05	74.12 65.55	74849 (1.58) 75539.3 (2.52)	410.16	31.11 44.05
u159	42080	**42080 (0)** 42201.9 (0.29)	**185.91**	16.25 **12.35**	42396 (0.75) 42470.45 (0.93)	138.31	56.49 41.8	43582 (3.57) 44194.8 (5.03)	406.45	44.13 39.03
rat195	2323	**2332 (0.39)** 2343.25 (0.87)	**9.68**	32.84 **48.16**	2402 (3.4) 2450.75 (5.5)	31.16	119.44 93.94	2402 (3.4) 2422.45 (4.28)	9.57	71.04 90.99
d198	15780	**15885 (0.67)** 15993.45 (1.35)	**76.13**	40.34 **50.12**	15979 (1.26) 16270.4 (3.11)	179.24	163.14 147.3	16487 (4.48) 16731.7 (6.03)	188.48	63.99 93.31
kroA200	29368	**29380 (0.04)** 29526.75 (0.54)	**112.12**	31.57 **25.16**	30196 (2.82) 30935.75 (5.34)	448.98	172.31 160.72	30798 (4.87) 31320.5 (6.65)	256.21	66.66 79.73
pr226	80369	**80500 (0.16)** 80883.05 (0.64)	**255.01**	46.71 **39.63**	81124 (0.94) 84492.25 (5.13)	1789.16	168.17 154	83027 (3.31) 84005.2 (4.52)	435.43	84.47 113.67

continued on next page

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 1. *(continued).*

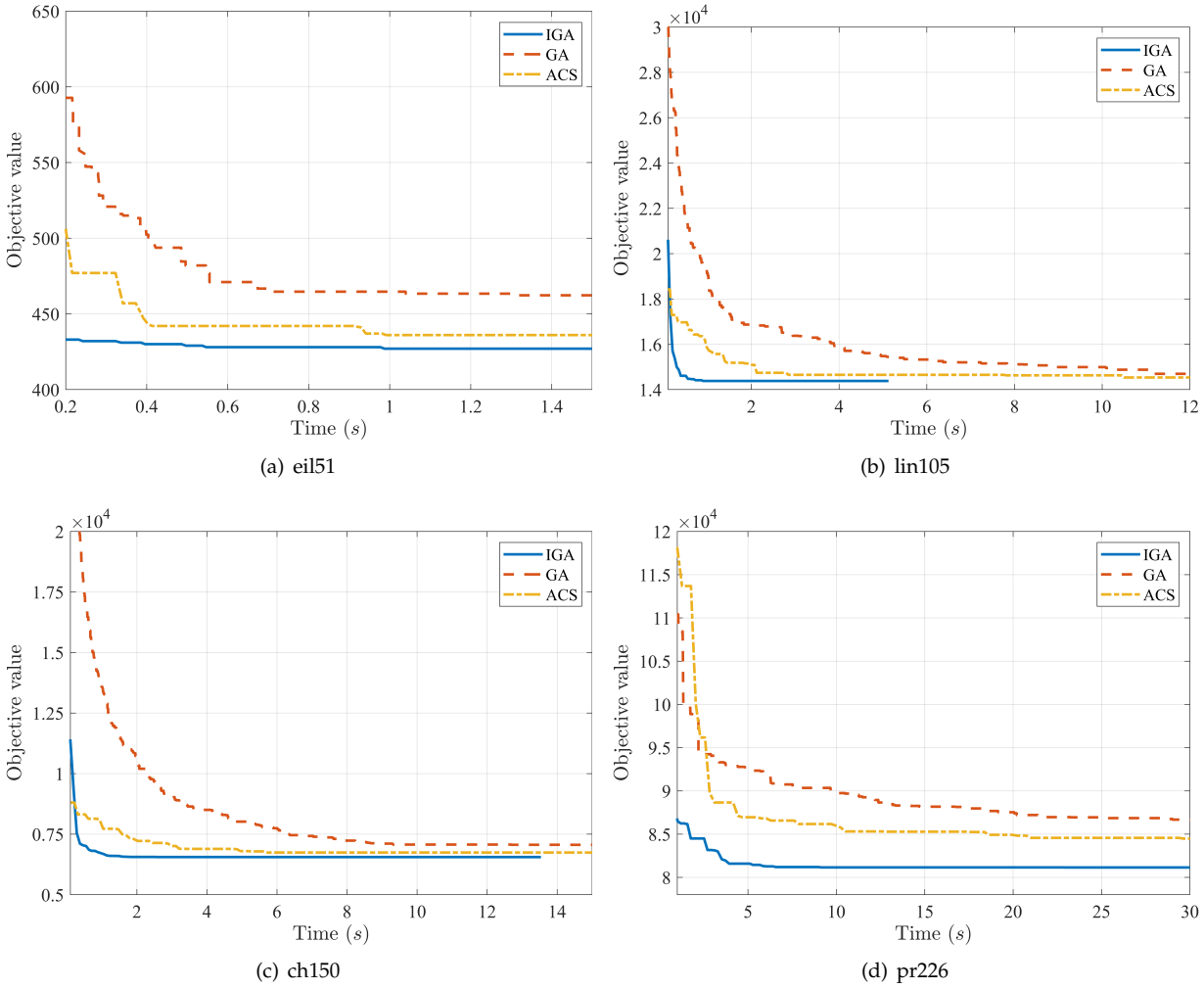
Instance		IGA			GA			ACS		
Name	BKS	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}	R_{best} (PD_{best}) R_{avg} (PD_{avg})	R_{std}	T_{Rb} T_{avg}
xqg237	1019	**1027 (0.79)** 1031.35 (1.21)	**4.04**	42.36 **34.42**	1062 (4.22) 1090.6 (7.03)	16.54	222.25 202.02	1081 (6.08) 1096.4 (7.6)	11.84	105.78 83.27
gil262	2378	**2381 (0.13)** 2392.55 (0.61)	**10.73**	70.38 **76.99**	2491 (4.75) 2541.8 (6.89)	34.6	328.5 325.98	2564 (7.82) 2594.85 (9.12)	18.17	118.42 166.12
pr264	49135	**49135 (0)** 49287.35 (0.31)	**243.7**	73.85 **92.22**	50411 (2.6) 53602.05 (9.09)	1627.27	380.4 497.31	51893 (5.61) 52451.6 (6.75)	333.2	135.88 256.38
pr299	48191	**48248 (0.12)** 48645.35 (0.94)	330.8	108.56 **91.98**	50372 (4.53) 51657.1 (7.19)	1029.18	433.79 472.81	52663 (9.28) 53056.7 (10.1)	**330.56**	182.62 221.02
lin318	42029	**42203 (0.41)** 42630.25 (1.43)	**310.79**	131.16 **168.95**	44466 (5.8) 45454.3 (8.15)	838.64	573.21 656.22	46273 (10.1) 47145.25 (12.17)	344.83	198.23 156.24
pma343	1368	**1373 (0.37)** 1379.5 (0.84)	**4.57**	125.75 **82.22**	1423 (4.02) 1450.25 (6.01)	15.67	652.51 792.98	1478 (8.04) 1512.55 (10.57)	15.32	281.64 462.81
pka379	1332	**1337 (0.38)** 1344.7 (0.95)	**5.89**	175.62 **173.24**	1390 (4.35) 1424.55 (6.95)	18.06	898.63 910.52	1416 (6.31) 1442.9 (8.33)	18.21	373.21 387.1
bcl380	1621	**1630 (0.56)** 1644.05 (1.42)	**8.52**	125.36 **94.35**	1723 (6.29) 1789.95 (10.42)	29.13	1106.53 1344.2	1732 (6.85) 1753.1 (8.15)	13.06	368.99 475.46
pbl395	1281	**1288 (0.55)** 1300.6 (1.53)	**5.57**	181.8 **184.75**	1369 (6.87) 1401.95 (9.44)	19.78	1265.45 1269.97	1427 (11.4) 1444.7 (12.78)	10.27	347.13 563.52
rd400	15281	**15350 (0.45)** 15512.55 (1.52)	**74.95**	261.87 **200.67**	15993 (4.66) 16414.55 (7.42)	196.73	1581.54 1617.67	17338 (13.46) 17519.65 (14.65)	105.81	419.85 375.42
pbk411	1343	**1359 (1.19)** 1368.15 (1.87)	**7.02**	216.66 **202.87**	1421 (5.81) 1472.55 (9.65)	24.53	1419.09 1940.95	1492 (11.09) 1518.5 (13.07)	15.07	462.24 447.8
fl417	11861	**11910 (0.41)** 11973.75 (0.95)	**49.41**	218.09 **253.43**	11993 (1.11) 12488.4 (5.29)	338.81	1548.44 1585.45	12559 (5.88) 12664.55 (6.77)	101.44	432.18 554.4
pbn423	1365	**1369 (0.29)** 1386.45 (1.57)	**8.61**	214.1 **231.72**	1459 (6.89) 1512.15 (10.78)	29.16	1508.73 1677.52	1515 (10.99) 1545.6 (13.23)	15.95	504.08 542.73
pbm436	1443	**1446 (0.21)** 1458.55 (1.08)	**7.19**	189.32 **238.13**	1538 (6.58) 1594.9 (10.53)	22.71	1881.99 2523.16	1570 (8.8) 1595 (10.53)	11.29	527.42 744.35
pr439	107217	**107666 (0.42)** 108535.5 (1.23)	**754.5**	264.02 **218.1**	110702 (3.25) 115479.95 (7.71)	2445.65	2097.05 2074.73	117852 (9.92) 120033.4 (11.95)	1099.39	464.33 463.28
pcb442	50778	**51380 (1.19)** 51597.35 (1.61)	**176.52**	332 **443.97**	54091 (6.52) 55595.1 (9.49)	990.52	1888.44 1889.08	56711 (11.68) 57762.95 (13.76)	348.22	554.25 572.78
d493	35002	**35484 (1.38)** 35750 (2.14)	**194.6**	469.09 **650.09**	36888 (5.39) 37488.9 (7.11)	336.55	3096.78 3437.53	38744 (10.69) 39710 (13.45)	412.42	771.14 753.55
Average		**0.27**	**125.45**	**90.43**	2.79	556.08	585.95	5.19	197.51	192.60
$C_{Rb}/C_{Ra}/C_{std}/C_{Ta}$		42/42/39/41			2/0/0/0			1/0/3/1		

From Table 1, the $C_{Rb}/C_{Ra}/C_{std}/C_{Ta}$ of IGA, GA and ACS are 42/42/39/41, 2/0/0/0 and 1/0/3/1 respectively. It is clear that the innovative IGA consistently produces superior results over GA and ACS. Additionally, the average computation time of IGA is the least in 97% instances, and its stability also has a far higher level than the other two algorithms. More specifically, the average PD_{best} of IGA is 0.27%, but GA and ACS are 2.79% and 5.19%, respectively 10 times and 19 times of IGA. In almost all cases, the PD_{avg} of IGA is less than 2%, but GA and ACS are often greater than 5%, especially ACS, even greater than 10% in some instances. In the view of stability, the average of the evaluation criteria R_{std} of IGA is 125.45, only 22.56% of GA and 63.52% of ACS. The average computation time

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

of IGA is 90.43 seconds, which is less than one-sixth as long as GA or half as long as ACS. The above discussion indicates that all the accuracy and the convergence speed of IGA are substantially superior to the traditional GA and ACS, which proves that the proposed IGA can reduce the computation time and improve the solution of ALC_IGA.

In Figure 5, the convergence speeds of IGA, GA, and ACS are compared under four instances which sizes ranging from 51 to 226. It can be observed that the convergence speed of IGA in the initial stage is much faster than that of GA and ACS. This is due to the heuristic crossover SBHX and the local search S_2-opt combined in IGA.



(a) eil51 (b) lin105

(c) ch150 (d) pr226

Figure 5. Comparison of the convergence speed of IGA, GA and ACS on 4 instances.

We know that the suggested IGA can be utilized to solve WSP as stated in Section 3, with just a minor adjustment to the distance between the start and end cities. In this part, to validate the effectiveness of IGA for WSP, the 42 instances in Table 1 are reinvestigated. The start and end cities of these instances were determined using the first and last elements of the best known solutions provided by TSPLIB and TSP Test Data, and the distances between start and end cities were set to -10^5 . The benchmark algorithm is the famous TSP solver LKH proposed by Keld Helsgaun [6]. The results, which include R_{best}, PD_{Best}, R_{avg}, PD_{avg}, R_{worst}, R_{std}, T_{Rb} and T_{avg} are shown in Table 2.

It is clear from Table 2 that the IGA can produce the solution of WSP with a high level of accuracy. We note that all PD_{best} are lower than 1% and 18 out of 42 are as good as LKH. The PD_{best} of 25 out of 42 instances produced by IGA are less than 0.1%, and all the PD_{best}

[6] http://webhotel4.ruc.dk/ keld/research/LKH/

 doi:10.20944/preprints202302.0412.v1

are lower than 1%. The outcomes on WSPs are even superior to those of IGA on TSPs in some aspects. In detailed, the average of PD_{best}, R_{std}, and T_{avg} are 0.2%, 134.28 and 81.83 respectively. By comparison, they are 0.27%, 125.45, and 90.43 on TSPs, that indicating that the IGA is able to find better solutions on WSPs in a shorter time than on TSPs. Especially on d493, the average execution time T_{avg} of IGA on WSPs is only 473.19, whereas it is 650.09 on TSPs.

According to the aforementioned analyses, the proposed IGA significantly outperforms GA and ACS in terms of convergence speed, solution quality, and stability. Additionally, on the WSP, which more often appeared in ALC_IGA, IGA also performs very well.

Table 2. Results obtained by IGA on 42 small-scale WSPs.

Instance		IGA							
Name	LKH	R_{best}	PD_{best}	R_{avg}	PD_{avg}	R_{worst}	R_{std}	T_{Rb}	T_{avg}
eil51	420	420	0	420.95	0.23	426	2.09	1.72	1.91
berlin52	7387	7387	0	7387	0	7387	0	1.77	1.95
st70	666	666	0	669.05	0.46	675	3.19	2.94	3.51
pr76	104443	104443	0	104856.4	0.4	105375	469.08	3.56	4.15
eil76	530	530	0	532.95	0.56	535	1.5	3.84	4.1
rat99	1207	1211	0.33	1217.35	0.86	1225	4.6	6.14	7.27
kroA100	21106	21106	0	21262.5	0.74	21509	92.99	6.26	6.46
rd100	7787	7787	0	7796.2	0.12	7947	35.53	5.42	6.58
eil101	629	629	0	631.8	0.45	637	2.44	5.89	6.9
lin105	14336	14336	0	14400.7	0.45	14509	60.77	6.55	7.5
pr107	39270	39270	0	39413.85	0.37	39729	134.11	7.83	10.84
pr124	58810	58810	0	58898.9	0.15	59030	78.16	9.73	12.5
bier127	117393	117650	0.22	118336.9	0.8	119236	594.61	10.87	12.81
ch130	6028	6075	0.78	6119.35	1.52	6201	40.47	12.69	13.33
xqf131	529	529	0	535.6	1.25	541	3.7	9.83	10.74
pr136	96386	96475	0.09	97392.7	1.04	99228	862.65	17.2	22.12
pr144	56126	56126	0	56134.65	0.02	56162	13.41	14.29	16.14
kroA150	26387	26390	0.01	26594.2	0.79	26975	164.62	24.62	19.39
ch150	6498	6498	0	6528.1	0.46	6591	19.52	15.77	15.25
pr152	64215	64215	0	64459.35	0.38	65335	335.55	13.21	19.92
u159	41797	41797	0	41925.8	0.31	42410	179.63	12.93	16.2
rat195	2260	2260	0	2264.7	0.21	2297	8.42	19.28	24.15
d198	12804	12855	0.4	12914.7	0.86	13019	48.92	61.13	48.05
kroA200	29206	29218	0.04	29411.5	0.7	29688	121.34	28.33	35.48
pr226	78587	78637	0.06	79045.9	0.58	80116	378.15	39.01	49.5
xqg237	1004	1012	0.8	1021.4	1.73	1032	5.53	35.45	46.88
gil262	2375	2378	0.13	2396.7	0.91	2415	10.32	66.78	72.32
pr264	46430	46430	0	46914.8	1.04	47922	439.57	70.32	64.26
pr299	47534	47563	0.06	48069.9	1.13	48544	275.39	133.62	112.19
lin318	41608	41704	0.23	42139.8	1.28	42714	266.71	179.63	119.53
pma343	1323	1326	0.23	1336.5	1.02	1357	9.26	146	125.9
pka379	1267	1269	0.16	1282.8	1.25	1312	11.47	155.4	153
bcl380	1606	1609	0.19	1623.9	1.11	1660	12.9	95.49	121.71
pbl395	1277	1284	0.55	1292.65	1.23	1311	6.71	141.11	157.49
rd400	15192	15310	0.78	15435.5	1.6	15620	79.69	157.5	209.83
pbk411	1337	1348	0.82	1367.8	2.3	1380	7.64	274.59	203.49
fl417	11414	11423	0.08	11464.45	0.44	11679	54.26	274.92	225.02
pbn423	1361	1362	0.07	1382.6	1.59	1407	10.21	243.59	196.58
pbm436	1420	1431	0.77	1446.15	1.84	1460	8.36	186.44	179.57
pr439	104810	104957	0.14	105786.2	0.93	106390	383.21	322.24	271.09
pcb442	50331	50734	0.8	51205.2	1.74	51654	252.19	333.2	327.85
d493	32897	33097	0.61	33363.95	1.42	33722	154.92	510.58	473.19
Average		-	0.20	-	0.86	-	134.38	87.33	81.83

 doi:10.20944/preprints202302.0412.v1

5.5. Parameters tuning for ALC_IGA

The solution initialization phase of ALC_IGA shown in Section 4.1 shows that the main parameters of ALC_IGA in the first phase only is M, which limits the time required to solve TSP or WSP less than T_1. The results from the previous section show that, under ordinary situations, the IGA can handle TSPs with less than 100 nodes in 6 seconds and solve TSPs with less than 150 nodes in 20 seconds. Consequently, a decent M shouldn't go beyond 150 too much. In order to choose a favorable M for ALC_IGA to balance the computation time and quality of solution, numerical comparison of M was set to 50, 100, and 150 on 45 instances are considered in this section. These instances are medium-scale, which sizes ranging from 1.3×10^3 to 2.5×10^4. Due to the fact that the distribution of nodes greatly affects the clustering effect, in order to fairly study the influence of M on the results of ALC_IGA, a variety of instances come from TSPLIB, TSP Test Data and TNM data were studied in this experiment. In the following of this paper, the termination condition of IGA is set to when there has been no improvement in the population for 30 iterations, and the other parameters are as same as in the last section. Denote the ALC_IGA with $M = 50, 100, 150$ as ALC_IGA50, ALC_IGA100, ALC_IGA150 respectively, the major five evaluation criteria R_{best}, PD_{best}, R_{avg}, PD_{avg}, T_{avg} and $C_{Rb}/C_{Ra}/C_{Ta}$ of the results are presented in Table 3.

Table 3. Comparison of results obtained by ALC_IGA with M setting to 50, 100, 150 respectively.

Instance		ALC_IGA50			ALC_IGA100			ALC_IGA150		
Name	BKS	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}
rl1323	270199	9.63	14.83	**14.67**	10.07	11.7	25.36	**5.99**	**9.84**	32.49
dca1389	5085	10.05	11.56	**16.05**	**5.17**	8.32	27.7	6.12	**7.49**	52.5
fl1400	20127	**3.42**	**7.89**	**15.86**	6.04	10.17	22.66	6	9.16	39.92
u1432	152970	5.62	7.06	**15.99**	4.73	5.62	24.17	**4.41**	**5.29**	45.62
fl1577	22249	10.78	14.05	**18.35**	8.88	12.32	29.87	**7.38**	**11.37**	41.37
fnb1615	4956	8.35	10.05	**18.32**	7.14	8.82	35	**5.31**	**7.49**	47.57
d1655	62128	8.44	9.78	**19.13**	5.43	6.93	30.27	**3.17**	**4.22**	53.45
vm1748	336556	8.46	9.74	**21.64**	6.4	7.71	42.32	**5.15**	**6.86**	63.85
u1817	57201	9.37	10.89	**20.25**	7.47	9.53	32.81	**6.98**	**8.81**	71.13
dkd1973	6421	7.16	8.24	**23.86**	**5.17**	**6.14**	40.96	6.73	7.99	54.72
Tnm2002	37029600	**7.33**	**10.74**	**21.7**	8.4	13.72	28.08	9.36	14.93	49.52
d2103	80450	12.56	15.9	**25.49**	10.56	12.7	45.11	**9.03**	**10.69**	70.55
bva2144	6304	7.92	9.68	**24.25**	5.9	7.49	39.01	**4.6**	**5.65**	68.61
u2319	234256	2.64	3.2	**25.39**	1.8	2.25	41.08	**1.76**	**2.16**	69.86
pr2392	378032	8.48	9.83	**30.69**	7.85	9.22	46.79	**6.65**	**8.31**	121.08
pcb3038	137694	8.13	9	**37.59**	6.42	7.33	72.23	**5.56**	**6.5**	112.85
ltb3729	11821	9.86	11.07	**42.9**	6.97	8.54	67.82	**5.74**	**7.16**	114.96
fl3795	28772	13.79	16.04	**43.27**	10.4	12.85	68.44	**9.23**	**12.3**	100.94
Tnm4000	74858233	**4.73**	**7.55**	**42.1**	8.88	12.05	62.52	10.88	16	86
fnl4461	182566	6.95	7.72	**56.3**	5.35	5.9	108.01	**4.52**	**5.29**	160.13
bgf4475	13221	13.46	15.06	**51.97**	10.51	11.6	84.75	**9.15**	**10.43**	121.42
fea5557	15445	12.35	13.34	**64.25**	8.73	9.6	99.42	**8.11**	**8.87**	172.33
rl5915	565530	17.75	19.14	**66.7**	12.81	14.58	108.87	**11.43**	**12.86**	158.25

continued on next page

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 3. *(continued).*

Instance		ALC_IGA50			ALC_IGA100			ALC_IGA150		
Name	BKS	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}
rl5934	556045	15.77	17.94	**68.37**	12.59	13.82	107.29	**10.5**	**11.73**	151.87
Tnm6001	112708118	7.68	9.76	**62.41**	**5.96**	**9.05**	89.06	9.32	12.26	120.36
xsc6880	21535	12.77	13.91	**79.76**	10.25	11.04	130.57	**9.05**	**9.64**	191.37
bnd7168	21834	11.88	12.63	**87.6**	8.41	9.31	142.97	**7.73**	**8.56**	226.37
lap7454	19535	13.87	14.62	**87.4**	9.89	10.67	127.22	**8.96**	**9.59**	204.57
Tnm8002	150561446	12.85	14.74	**88.22**	**6.24**	**8.01**	112.69	7.29	10.04	132.49
ida8197	22338	10.89	12.53	**96.35**	9.21	10.01	160.35	**7.66**	**9**	240.39
dga9698	27724	14.88	15.82	**116.08**	11.08	12.23	190.04	**9.55**	**10.49**	272.45
Tnm10000	188414262	20.6	23.02	**103.41**	**5.13**	**6.97**	127.58	5.94	8.93	160.26
xmc10150	28387	13.61	14.51	**113.46**	10.75	11.77	191.94	**9.47**	**10.4**	284.55
rl11849	923288	14.22	15.03	**141.87**	10.8	11.47	224.3	**9.31**	**10.18**	358.89
usa13509	19982859	9.81	10.93	**165.83**	8.26	8.82	318.67	**6.69**	**7.18**	492.82
xvb13584	37083	11.16	11.85	**155.75**	8.48	9.13	236.12	**7.77**	**8.27**	373.29
brd14051	469385	8.07	8.52	**174.4**	5.92	6.18	334.87	**5.16**	**5.49**	552.78
d15112	1573084	7.94	8.43	**190.14**	6.13	6.62	349.63	**5.54**	**5.79**	598.85
xia16928	52838	13.24	13.85	**194.99**	8.77	9.53	312.44	**8.26**	**8.74**	477.84
pjh17845	48083	11.19	12	**204.99**	8.22	8.88	324.65	**7.63**	**8.38**	524.2
d18512	645238	8.06	8.39	**233.68**	6.47	6.86	438.17	**5.27**	**5.57**	720.64
Tnm20002	377692238	15.16	23.22	**209.17**	5.44	6.58	268.69	**4.88**	**6.42**	379.48
ido21215	63501	12.57	13.18	**246.85**	9.77	10.25	401.68	**8.8**	**9.14**	656.89
lsb22777	60977	13.35	13.83	**268.06**	9.85	10.71	409.42	**8.76**	**9.8**	660.7
bbz25234	69335	12.08	12.69	**290.48**	9.45	9.98	482.27	**8.5**	**8.99**	746.58
Average		10.64	12.31	**91.02**	7.96	9.4	148.09	**7.23**	**8.76**	231.93
$C_{Rb}/C_{Ra}/C_{Ta}$		3/3/45			5/4/0			37/38/0		

From Table 3, the $C_{Rb}/C_{Ra}/C_{Ta}$ of the ALC_IGA50, ALC_IGA100, ALC_IGA150 are 3/3/45, 5/4/0 and 37/38/0 respectively. As can be seen, the ALC_IGA50 is the fastest, whereas the ALC_IGA150 algorithm usually produces the best results. When the size of instance is less than 2×10^3, ALC_IGA50 has the minimum PD_{best} and PD_{avg} on fl1400, ALC_IGA10 has the lowest PD_{best} on dca1389 and dkd1973. However, the PD_{best} and PD_{avg} of ALC_IGA150 on the three instances are all less than 10%, this is still a respectable result. When the instance size is large than 2×10^3, the ALC_IGA50 and ALC_IGA100 only perform better than the ALC_IGA150 on TNM instances. Concerning specifics, the ALC_IGA50 works well on Tnm2002 and Tnm4000, the ALC_IGA100 excels on Tnm6001, Tnm8002, and Tnm10000, but the ALC_IGA150 provided the best result on the large instance of Tnm20002. The results of ALC_IGA150 are therefore superior to those of ALC_IGA50 and ALC_IGA100 in TSPLIB and TSP Test Data, and it is still a suitable approach for TNM data. The average of PD_{best} and PD_{avg} for the three algorithms shown at the bottom of Table 3 also support this.

Furthermore, considering the algorithms' running time, the mean of T_{avg} of ALC_IGA50 is 91.02, which is three-fifths of the time taken by ALC_IGA100 and two-fifths of ALC_IGA-150. This indicates that the fastest algorithm is ALC_IGA50, and the ratio of running time hardly changes with the size of the instance. However, even the slowest proposed ALC_IGA150 could handle the 10^4 nodes instance with just approximately 10% deviation

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

percentage in the same amount of running time as the IGA, which can only solve the instance with a size of roughly 400 nodes. The fastest ALC_IGA50, which is more than 60 times faster than the IGA, can deal with 2.5×10^4 nodes in the same amount of time. Thus the high efficiency of ALC_IGA has been verified.

Figure 6 displays the deviation percentage of each run among all instances. It is noteworthy that for all three algorithms, most of the deviation percentages are under 20%. In particular, the deviation percentages of the ALC_IGA100 and ALC_IGA150 are less than 10% in the majority of instances. Furthermore, the figure also reveals that the ALC_IGA100 and ALC_IGA150 have many overlapping regions, indicating that the performance of the two algorithms is roughly equivalent.

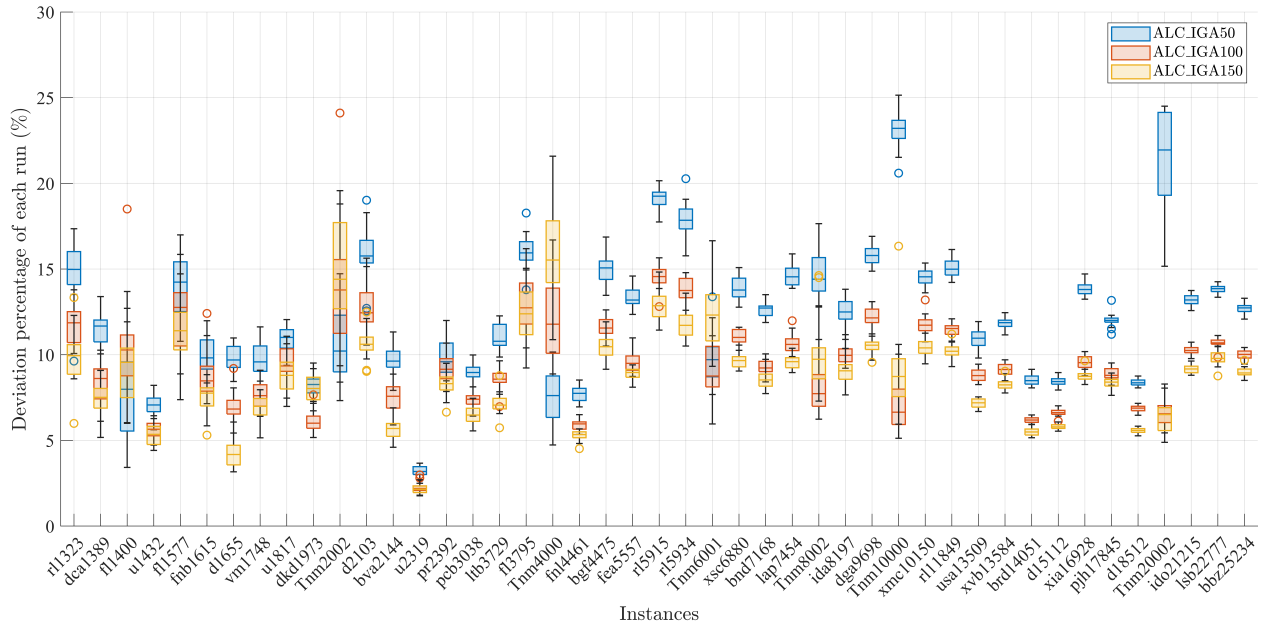


Figure 6. The deviation percentage of each run on 45 medium-scale instances with M setting to 50, 100, 150.

Additionally, the relationship between the running time of ALC_IGA and the value of M is taken into account. The average execution time for the instances of the three algorithms is plotted in Figure 7 in different colors. In order to discuss the computational complexity of the algorithms, the exponential curve fitting for each group is calculated. Due to the computation time of ALC_IGA150 is larger than the other two, its slope shown in the figure is undoubtedly the steepest. The approximated time complexities of ALC_IGA50, ALC_IGA100 and ALC_IGA150 are $O(n^{0.9992})$, $O(n^{0.9958})$ and $O(n^{1.02})$ respectively, which are all extremely close to the linear computational complexity $O(n)$. With 95% confidence bounds, the upper bound of the computational complexity for ALC_IGA50 is 1.0326, and the other two are 1.0963 and 1.151. The statistical outcomes of curve fitting are shown in Table 4. It can be seen that all three fitting models have high confidence, especially the R^2 of ALC_IGA50 is over 0.99. The above results prove the computational complexity analysis of the proposed ALC_IGA in Section 4.3.

Table 4. The exponential curve fitting $a \cdot n^b$ of the running time of ALC_IGA while M setting to 50, 100, 150.

M	a	b	SSE	R^2	Adjusted R^2	RMSE
50	0.0118 ± 0.0038	0.9992 ± 0.0334	1705	0.9938	0.9936	6.297
100	0.0198 ± 0.0192	0.9958 ± 0.1005	41246	0.9459	0.9446	30.97
150	0.0247 ± 0.0314	1.02 ± 0.131	167900	0.9146	0.9126	62.49

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

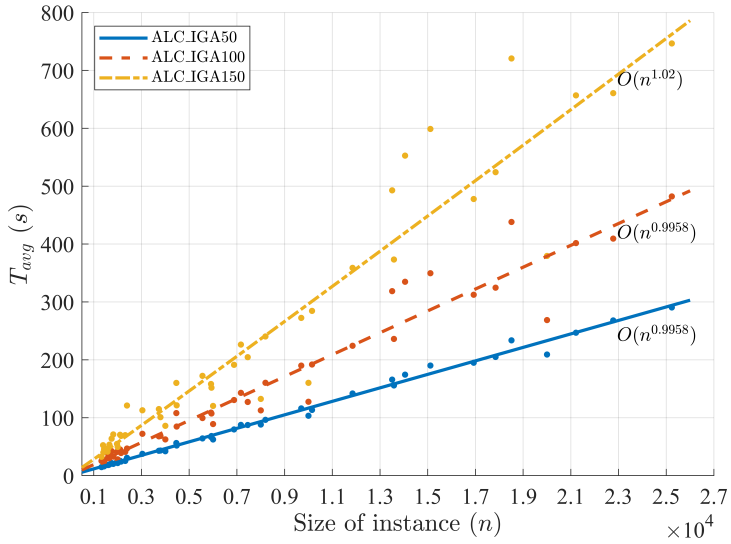


Figure 7. Computational complexity analysis of the proposed ALC_IGA.

To sum up, the quality of the solution obtained by ALC_IGA has a strong relationship with the data distribution and the value of M. On the other hand, the larger M is set, the longer the computation time required by ALC_IGA according to the numerical experiments. In most cases, setting M to 100 is a typical compromise choice to balance computation time and quality.

5.6. ALC_IGA compared with two-layered algorithms

The effectiveness of ALC_IGA on medium-scale problems was confirmed in Section 5.5, although it is unclear whether it is superior to the other layered algorithms. To illustrate the performance of ALC_IGA, the proposed ALC_IGA is compared with two typical algorithms, which are TLGA [47] and TLACS [7]. The TLGA and TLACS are re-coded in Matlab, and to be fair, the running time and the solution quality are improved to be better than the literature. The main parameters were set as follows: the M of ALC_IGA was set to 100; the numbers of cluster centers of TLACS and TLGA were automatically adjusted according to the size of the instance; the termination conditions of ALC_IGA, TLACS, TLGA were that when there has been no improvement of the solution for 30, 30, 100 iterations respectively. All of the algorithms are implemented in single-thread. There are 45 medium-scale instances whose sizes ranging from 1×10^3 to 4×10^5 are investigated in this experiment.

Table 5. Comparison of results obtained by ALC_IGA, TLACS and TLGA on medium-scale instances.

Instance		ALC_IGA			TLACS			TLGA		
Name	BKS	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}
vm1084	239297	**5.88**	**7.72**	21.32	12.03	13.63	**13.59**	53.63	66.69	92.61
d1291	50801	**8.76**	**10.65**	20.89	14.08	16.34	**15.32**	58.1	65.88	61.76
rl1323	270199	**10.01**	**11.46**	24.65	17.88	20.35	**13.86**	70.35	79.18	69.01
fl1400	20127	4.57	9.74	**23.93**	**4.25**	**6.85**	28.78	54.8	74.71	147.38
fl1577	22249	**7.79**	13.08	29.34	10.19	**12.27**	**18.86**	86.64	98.19	82.13
d1655	62128	**5.03**	**6.54**	29.27	13.15	14.32	**21.17**	50.89	61.03	115.34
vm1748	336556	**6.7**	**7.64**	31.65	12.74	14.19	**22.95**	64.18	76.08	108.67
u1817	57201	**7.88**	**9.49**	33.76	10.88	12.34	**19.89**	54.99	61.58	102.33

continued on next page

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 5. *(continued).*

Instance		ALC_IGA			TLACS			TLGA		
Name	BKS	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}	PD_{best}	PD_{avg}	T_{avg}
d2103	80450	**10.58**	**12.45**	44.22	19.26	21.76	**24.17**	59.41	68.84	166.26
u2152	64253	**8.06**	**9.37**	39.2	12.12	13.45	**28.12**	57.26	63.54	144.79
u2319	234256	**1.84**	**2.3**	41.32	4.3	5.09	**31.11**	32.87	36.72	150.83
pr2392	378032	**7.17**	**9.13**	44.87	10.99	13.38	**37.39**	53.73	62.36	132.17
pcb3038	137694	**6.66**	**7.38**	76.63	12.17	13.25	**48.31**	51.59	57.4	175.13
fl3795	28772	**11.53**	**12.98**	**66.47**	13.01	14.28	112.8	101.58	116.31	275.55
dkf3954	12538	**9.02**	**9.89**	76.13	14.47	16.08	**68.82**	61.99	67.22	247.99
Tnm4000	74858233	8.58	12.59	59.12	**3.59**	**5.17**	**44.85**	259.8	298.2	214.53
fnl4461	182566	**5.53**	**5.93**	112.01	10.1	10.75	**90.24**	47.59	52.67	240.54
ca4663	1290319	**8.61**	**10.45**	**100.84**	14.37	16.41	155.92	76.7	92.73	378.54
xqd4966	15316	**5.56**	**6.53**	**100.23**	11.07	12.45	105.53	71.28	94.91	349.59
fqm5087	13029	**5.52**	**6.56**	**99.07**	11.15	12.03	99.23	81.88	94.76	325.78
fea5557	15445	**8.93**	**9.84**	**106.72**	14.06	15.72	111.2	63.68	74.72	417.61
rl5915	565530	**14.14**	**15.1**	**103.29**	20.18	22.21	113.81	75.64	85.77	386.15
rl5934	556045	**12.66**	**13.9**	**105.45**	19.4	20.16	107.38	72.67	84.82	374.8
tz6117	394718	**6.86**	**7.63**	**136.46**	13.17	14.13	205.99	66.47	73.76	429.67
xsc6880	21535	**9.94**	**11.14**	**132.75**	15.76	17.26	150.23	64.88	72.96	495.94
bnd7168	21834	**8.18**	**9.13**	**139.69**	14.7	16.02	163.16	63.15	70.91	518.01
lap7454	19535	**9.76**	**10.75**	**128.89**	15.9	16.71	172.89	67.42	74.47	594.09
ida8197	22338	**9.19**	**9.92**	**152.68**	14.87	15.74	190.42	61.98	72.42	610.65
dga9698	27724	**11.3**	**12.18**	**176.72**	17.14	17.88	256.85	71.31	77.97	690.84
Tnm10000	188414262	5.43	7.57	**132**	**1.96**	**3.05**	163.07	393.56	458.1	722.56
xmc10150	28387	**10.9**	**11.68**	**175.01**	16.45	17.23	265.42	72.03	77.04	734.97
rl11849	923288	**10.35**	**11.43**	**224.34**	15.54	16.63	359.35	69.73	75.44	933.61
usa13509	19982859	**8.21**	**8.65**	**295.5**	13.61	14.53	664.7	66.37	71.67	1378.6
brd14051	469385	**5.72**	**6.11**	**342.91**	10.96	11.71	528.3	50.36	58.39	1232.43
d15112	1573084	**6.1**	**6.44**	**356.77**	11.02	11.92	641.19	52.75	57.46	1443.81
it16862	557315	**8.55**	**9.11**	**361.73**	12.7	13.39	790.6	63.35	75.35	1547.82
d18512	645238	**6.59**	**6.84**	**434.83**	11.1	11.71	795.83	52.15	57.02	1722.17
boa28924	79622	**11.19**	**11.83**	**529.35**	15.76	16.41	1473.54	79.98	86.43	2760.68
Tnm30001	566973296	8.06	8.68	**417.43**	**1.18**	**1.78**	905.14	640.39	730.39	2924.84
pbh30440	88313	**11.33**	**11.77**	**585.99**	15.9	16.33	1685.75	72.34	80.03	3306.87
xib32892	96757	**10.34**	**10.84**	**613.21**	15.07	15.63	1897.12	76.96	83.16	3252.86
fry33203	97240	**11.44**	**11.79**	**617.37**	15.2	16.01	1992.54	76.68	82.39	3600.66
bby34656	99159	**9.67**	**10.19**	**647.45**	14.92	15.38	2192.23	70.47	77.38	3866
pba38478	108318	**10.7**	**11.21**	**732.33**	15.34	15.89	2614.56	73.06	79.11	4093.7
ics39603	106819	**11.97**	**12.54**	**725.33**	16.4	16.81	2584.34	76.36	83.37	4318.36
Average		**8.51**	**9.74**	**209.98**	12.89	14.1	489.48	89.84	102.43	1020.86
$C_{Rb}/C_{Ra}/C_{Ta}$		41/40/30			4/5/15			0/0/0		

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

As is shown in Table 5, the evaluation criteria $C_{Rb}/C_{Ra}/C_{Ta}$ of ALC_IGA is 41/40/30, the $C_{Rb}/C_{Ra}/C_{Ta}$ of TL-ACS is 4/5/15 and $C_{Rb}/C_{Ra}/C_{Ta}$ of TLGA is 0/0/0. First of all, it is pointed out that TLGA has no advantage in all instances compared with the other two algorithms in terms of solution quality and convergence speed. The TLACS obtained 4 best PD_{best} and 5 best PD_{avg} among all 45 instances. In detail, TLACS outperforms ALC_IGA on fl1400 and fl1577, but ALC_IGA defeats TLACS on fl3795. The other three instances where TLACS performs better are all hard-to-solve instances [64]. That is because the fewer clusters generated, the better solution produced, which is according to the results in Section 5.5. The average of PD_{best} and PD_{avg} for ALC_IGA are 8.51 and 9.74, while for TLACS and TLGA, they are 12.89 and 14.10, 88.84 and 102.43, respectively. The analyses above verify that the accuracy of ALC_IGA is superior to TLACS and TLGA in all scenarios except for TNM instances.

From Table 5, the average values of T_{avg} of ALC_IGA, TLACS and TLGA are 209.98, 489.48 and 1020.86 seconds. It can be seen that the proposed ALC_IGA is much faster than the other two algorithms. In detail, when the size of the instance is less than 4.5×10^3, TLACS is faster than ALC_IGA in most cases. When the size of the instance is between 4.5×10^3 and 10^4, the running time of ALC_IGA and TLACS are very close. When the size of the instance is larger than 10^4, the proposed ALC_IGA has huge advantages, especially when the problem size is greater than 3×10^4, the computation time of ALC_IGA is less than one-third of TLACS and less than one-fifth of TLGA.

Figure 8 converts a large amount of data in Table 5 into an explicit image. The real lines represent the PD_{avg} and T_{avg} of ALC_IGA. It is closer to the horizontal axis, which means that the ALC_IGA has high performance on accuracy and convergence speed. The results of run time for ALC_IGA, TLACS and TLGA with exponential curve fitting are $O(n^{0.945})$, $O(n^{1.611})$ and $O(n^{1.221})$. It reveals that the gap in computation time between ALC_IGA and the other two algorithms will increase as the size of the problem increases.

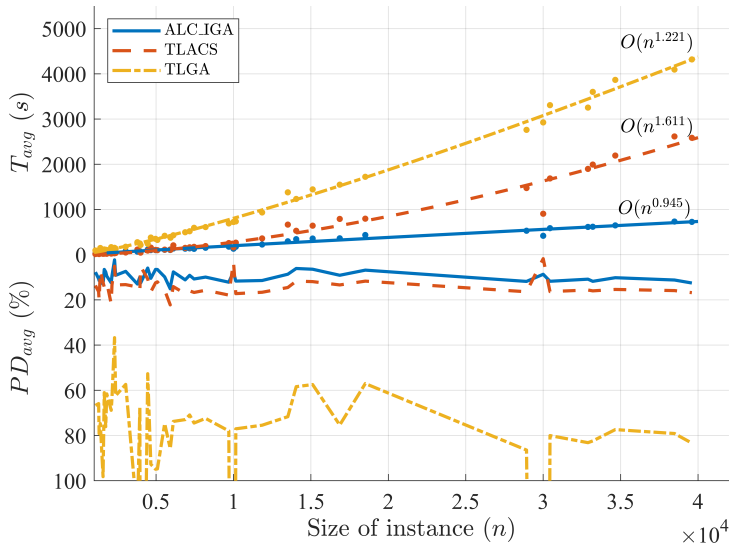


Figure 8. Results analysis of the proposed algorithm, TLGA and TLACS.

5.7. Results on large-scale TSP instances

In this section, to investigate the performance of ALC_-IGA on large-scale instances, the new ALC_IGA is compared to three state-of-the-art algorithms, which are TLACS [7], an accelerating genetic algorithm evolution via ant-based mutation and crossover (ER-ACO) [28] and a 3L-MFEA-MP [51]. The ALC_IGA and TLACS were implemented in Matlab R2022a and parallelized by the parallel computing toolbox in Matlab. The ER-ACO was set on an AMD Ryzen 2700 CPU with 16 threads in parallel. The parallel 3L-MFEA-MP was coded in Python, and it was implemented on a supercomputer with a 24-core Intel Xeon CPU and 96 GB RAM. The sizes of the 15 involved instances range from 4×10^4 to 2×10^5.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 6. Comparison of ALC_IGA and three state-of-the-art algorithms on large-scale instances.

Instance	BKS	Algorithms	R_{best}	PD_{best}	R_{avg}	PD_{avg}	T_{avg}
rbz43748	125183	ALC_IGA	**138336**	**10.51**	**138780**	**10.86**	**78.97**
		TLACS	143707	14.8	144783	15.66	460.23
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
fht47608	125104	ALC_IGA	**138369**	**10.6**	**138854**	**10.99**	**90.39**
		TLACS	143328	14.57	144080	15.17	500.51
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
fna52057	147789	ALC_IGA	**162347**	**9.85**	**162900**	**10.22**	**89.73**
		TLACS	170295	15.23	170813	15.58	545.47
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
bna56769	158078	ALC_IGA	**174110**	**10.14**	**175110**	**10.77**	**121.35**
		TLACS	181703	14.95	182421	15.4	604.53
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
dan59296	165371	ALC_IGA	**183301**	**10.84**	**183803**	**11.15**	**112.64**
		TLACS	190994	15.49	191471	15.78	607.85
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
Tnm80002	1513392208	ALC_IGA	1719287088	13.6	1815094672	19.94	**145.72**
		TLACS	**1521978113**	**0.57**	1528977655	1.03	876.21
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
Tnm90001	1702667051	ALC_IGA	1900341576	11.61	2038420433	19.72	**161.17**
		TLACS	**1712186024**	**0.56**	1717989072	0.9	949.71
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
Tnm100000	1891945975	ALC_IGA	2107195713	11.38	2237645170	18.27	**171.85**
		TLACS	**1902231611**	**0.54**	1910148253	0.96	1497.72
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
mona-lisa100K	5757191	ALC_IGA	**5930206**	**3.01**	5934489	3.08	**235.13**
		TLACS	6401529	11.19	6417896	11.48	1657.04
		ER-ACO	-	7.99	-	8.9	1792.95
		3L-MFEA-MP	6513686	13.34	6525173	13.34	1030.72
sra104815	251342	ALC_IGA	**276998**	**10.21**	**277851**	**10.55**	**212.9**
		TLACS	288535	14.8	289519	15.19	1562.05
		ER-ACO	-	-	-	-	-
		3L-MFEA-MP	-	-	-	-	-
vangogh120K	6543609	ALC_IGA	**6742349**	**3.04**	6746733	3.1	**314.21**
		TLACS	7332648	12.06	7344261	12.24	2269.41
		ER-ACO	-	8.66	-	9.22	1975.97
		3L-MFEA-MP	7423925	13.55	7430063	13.55	1256.78
venus140K	6810665	ALC_IGA	**7018375**	**3.05**	7021104	3.09	**341.17**
		TLACS	7638796	12.16	7647611	12.29	3262.63
		ER-ACO	-	8.33	-	8.72	2496.99
		3L-MFEA-MP	7718441	13.41	7724201	13.41	1518.13
pareja160K	7619953	ALC_IGA	**7854282**	**3.08**	7858881	3.14	**428.04**
		TLACS	8623198	13.17	8629465	13.25	3734.21
		ER-ACO	-	8.47	-	9.47	3049.45
		3L-MFEA-MP	-	-	-	-	-

continued on next page

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 6. *(continued).*

Instance	BKS	Algorithms	R_{best}	PD_{best}	R_{avg}	PD_{avg}	T_{avg}
courbet180K	7888731	ALC_IGA	**8148232**	**3.29**	8150953	3.32	**498.64**
		TLACS	8940877	13.34	8956732	13.54	4454.45
		ER-ACO	-	8.37	-	9.83	3666.29
		3L-MFEA-MP	-	-	-	-	-
earring200K	8171677	ALC_IGA	**8454565**	**3.46**	8460779	3.54	**522.74**
		TLACS	-	-	-	-	-
		ER-ACO	-	9.18	-	9.83	4236.65
		3L-MFEA-MP	9365519	14.65	9368743	14.65	2382.31

The results and five evaluation criteria R_{best}, PD_{best}, R_{avg}, PD_{avg} and T_{avg} are shown in Table 6. Compared to ALC_IGA with TLACS, the advantage of ALC_IGA in running time is apparent again. The running time of ALC_IGA is roughly one-sixth of TLACS when the problem size is around 5×10^4, but when the size approaches 2×10^5, the running time of it is just one-ninth of TLACS. The performance of ALC_IGA is better than TLACS in most conditions, but TLACS works pretty well on TNM instances.

There are four instances compared with 3L-MFEA-MP, results shown in Table 6 reveal that the performance of it is very close to TLACS, the difference between them in terms of PD_{best} and PD_{avg} is about 2%. While compared with ALC_IGA, the 3L-MFEA-MP is far worse than it in terms of convergence speed and solution quality. On the involved six instances, the PD_{best} and PD_{avg} of the novel intelligence algorithm ER-ACO exceed ALC_IGA by 2.5 times. Additionally, the proposed ALC_IGA runs significantly faster than ER-ACO.

Figure 9 shows the average computation time and deviation percentages of the four algorithms. It is clear that ALC_IGA performs well in most situations and is significantly faster than the others. According to the results illustrated in Section 5.5, the only drawback of ALC_IGA is on TNM instances, which can be improved by setting M larger.

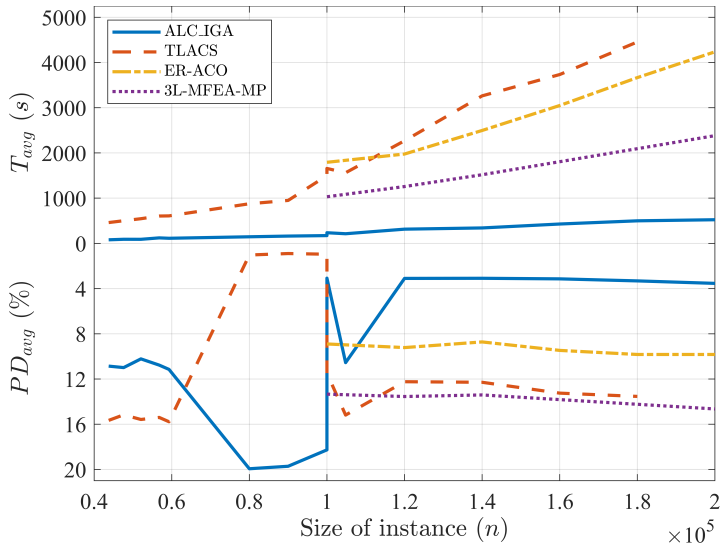
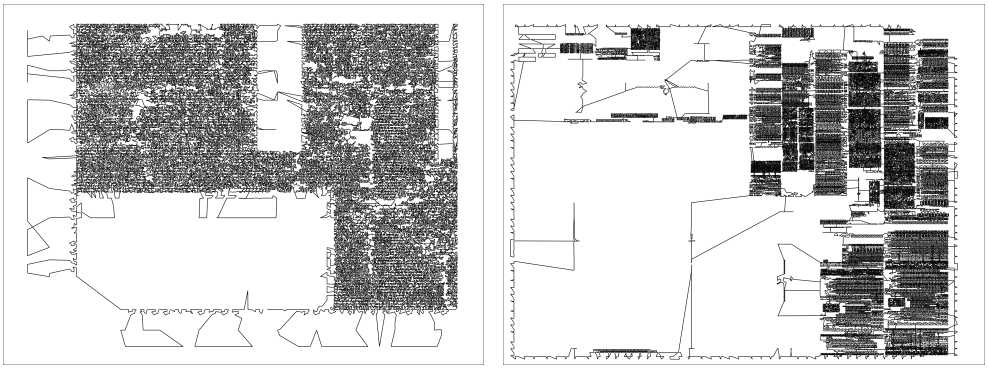


Figure 9. The results of the compared algorithms on large-scale TSPs.

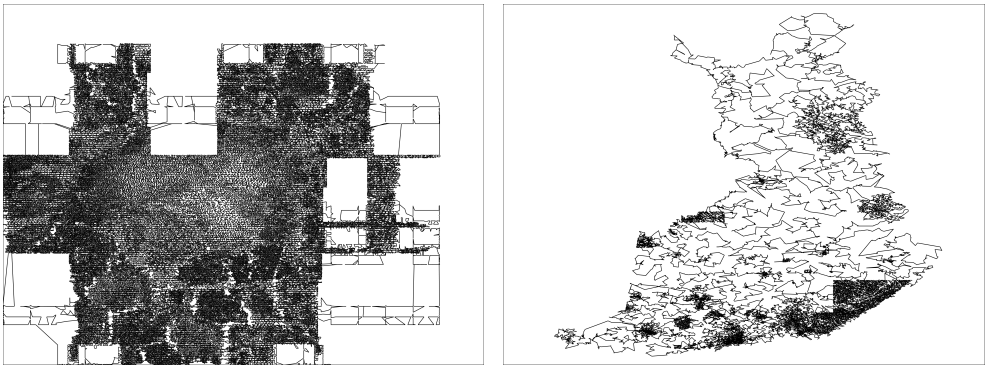
Finally, the results of ALC_IGA under M set to 50, 100 and 150 for five huge instances are also given. The ara238025, lra498378 and lrb744710 are three instances containing hundreds of thousands of nodes, which are the very large-scale integration instances of TSP Test Data. The Santa, which has 1437195 cities, as a benchmark instance for large-scale TSPs, has been investigated thoroughly by several well-known solvers in [65]. Gaia was published by William Cook in 2019 and includes two million coordinates of stars.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

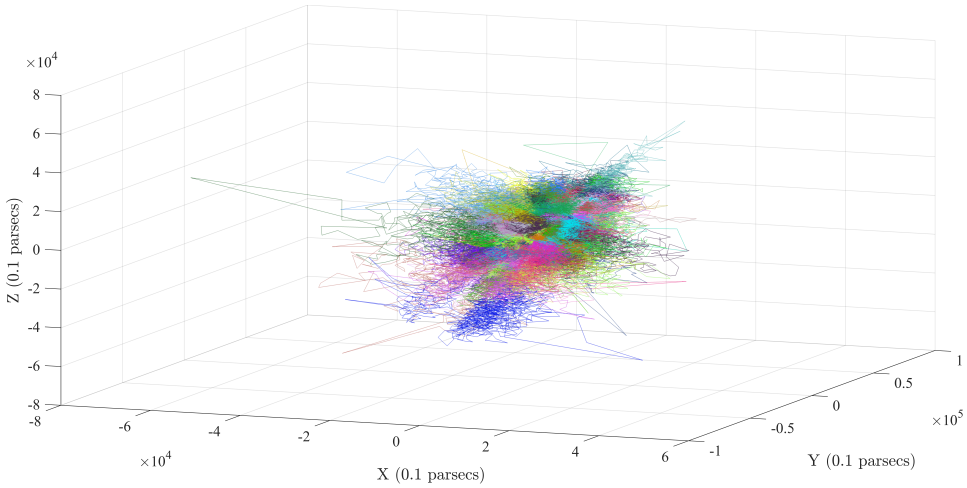
Five evaluation criteria and the average of them are presented in Table 7. It shows again that the larger the M set, the better solution obtained and the longer computation time needed. For ALC_IGA50, ALC_IGA100, and ALC_IGA150, the average of PD_{best} are 13.882, 11.064, and 10.304, respectively, which is extremely close to the average of PD_{avg}. This illustrates the strong stability of ALC_IGA, which the average of R_{std} has also proven. While M was set to 50 or 100, the 1.4×10^6 nodes instance can be handled within 1h on our implement, and even the large three-dimensional Gaia can be fixed within 1.5h. Figure 10 depicts the best solutions obtained by the ALC_IGA with $M = 100$.



(a) ara238025 (b) lra498378



(c) lrb744710 (d) santa1437195



(e) gaia2079471

Figure 10. Visualization of the best solutions obtained by the ALC_IGA with $M = 100$ on five large instances over 2×10^5 nodes.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 February 2023 doi:10.20944/preprints202302.0412.v1

Table 7. Results obtained by the ALC_IGA on five large instances over 2×10^5 nodes.

Instance	BKS	M	R_{best}	PD_{best}	R_{avg}	PD_{avg}	R_{std}	T_{avg}	T_{Rb}
ara238025	578761	50	649841	12.28	653160	12.85	1534	**242.59**	**250.69**
		100	634357	9.61	637414	10.13	1001	392.4	390.87
		150	**630357**	**9.17**	**631805**	**9.17**	**905**	621.9	587.38
lra498378	2168039	50	2504137	15.5	2511139	15.83	3620	**586.65**	**561.38**
		100	2424156	11.81	2431562	12.15	4526	799.82	822.77
		150	**2398861**	**10.92**	**2404857**	**10.92**	**4126**	1447.49	1241.13
lrb744710	1611232	50	1803710	11.95	1806807	12.14	1553	**832.53**	**856.56**
		100	1773389	10.06	1775519	10.2	1402	1164.27	1209.86
		150	**1756006**	**9.09**	**1757731**	**9.09**	**1199**	1728.67	1718.95
santa1437195109284000		50	126452359	15.71	126870650	16.09	282274	**2355.69**	**2502.04**
		100	122732785	12.31	123183399	12.72	282164	3403.22	3101.24
		150	**121831057**	**11.76**	**122134133**	**11.76**	**189127**	5022.14	4812.72
gaia2079471	288843524	50	329200974	13.97	329395175	14.04	106820	**5010.2**	**4865**
		100	322144985	11.53	322360796	11.6	117896	5225.05	5377.99
		150	**319244386**	**10.58**	**319408762**	**10.58**	**86203**	7891.54	7694.04
Average		50	-	13.882	-	14.19	79160.2	**1805.53**	**1807.13**
		100	-	11.064	-	11.36	81397.8	2196.952	2180.546
		150	-	**10.304**		**10.304**	**56312**	3342.348	3210.844

6. Conclusions and future research

The existing layered algorithms might encounter obstacles when solving large-scale TSPs: the subtask small-scale TSPs solved slowly; the number of cluster centers in the upper layer may be enormous; the single cluster in the bottom layer may have an overwhelming number of nodes; the quality of the final solution is poor.

In this study, aiming at solving large-scale TSPs with millions of nodes fast, the ALC_IGA with high parallelizability is proposed. In the first phase, ALC_IGA ensures that all sub-TSPs and sub-WSPs are smaller than the specified size through k-means repeatedly applied, thereby reducing the computation time. In the second phase, the TS_2-opt is developed to rapidly improve the initial solution. The IGA is also proposed for small-scale TSPs and WSPs, with the following significant modifications: the polygynandry-inspired SBHX is designed for high convergence speed; the S_2-opt for balancing convergence speed and falling into local optimum is created. According to the study, the computational complexity of ALC_IGA is between $O(n \log n)$ and $O(n^2)$.

The numerical results on 42 instances show that the proposed IGA is better than both GA and ACS in terms of convergence speed and accuracy, and it performs better on WSP than on TSP. According to the numerical results on lots of instances from diverse sources, in most conditions, ALC_IGA outperforms TLGA, TLACS, 3L-MFEA-MP and the novel ER-ACO in terms of precision, stability and computation speed. The worst situation of ALC_IGA is on the hard-to-solve TSP instances, where the errors are still less than 20% and can be improved by adjusting the parameters.

To improve the performance of ALC_IGA, future research may focus on optimizing the initial solution of ALC_IGA, investigating the influence of the different clustering algorithms adopted, and improving the performance on the hard-to-solve TSPs. The ALC_IGA can also be extended to solve large-scale ATSPs, CTSPs, DTSPs and other related problems.

Author Contributions: Conceptualization, software, investigation, data curation and writing—original draft preparation, H.-Y.X.; methodology, validation, formal analysis and writing—review and editing, H.-Y.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

 doi:10.20944/preprints202302.0412.v1

Data Availability Statement: The data involved in this study are available on https://github.com/nef-phys/tsp.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

TSPs	traveling salesman problems
ALC_IGA	Adaptive layered clustering framework with improved genetic algorithm
ATSPs	Asymmetric TSPs
CTSPs	Clustered TSPs
DTSPs	Dynamic TSPs
MTSPs	Multiple TSPs
WSPs	Wandering salesman problems
EA	Evolutionary algorithm
ACO	Ant colony optimization algorithm
ACS	Ant colony system
SFLA	Shuffled frog leaping algorithm
SA	Simulated annealing algorithm
PSO	Particle swarm optimization
GA	Genetic algorithm
PMX	Partially mapped crossover
OX	Ordered crossover
CX	Cycle crossover
SCX	Sequential constructive crossover operator
CMX	Completely mapped crossover operators
BHX	Bidirectional heuristic crossover operator
IGA	Improved genetic algorithm
TLACS	Two-layered ant colony system algorithm
3L-MFEA-MP	The three-layered evolutionary optimization framework
SBHX	Selective bidirectional heuristic crossover
S_2-opt	Simplified 2-opt
TS_2-opt	Two phases simplified 2-opt algorithm
TLGA	Two-level genetic algorithm
TNM	Hard to solve instances of the Euclidean TSPs
ER-ACO	Accelerating genetic algorithm evolution via ant-based mutation and crossover

References

1. Zgurovsky, M.Z.; Pavlov, A.A. *Combinatorial optimization problems in planning and decision making: Theory and applications*; Springer: Cham, 2018. https://doi.org/10.1007/978-3-319-98977-8.
2. Öncan, T.; Altınel, I.K.; Laporte, G. A comparative analysis of several asymmetric traveling salesman problem formulations. *Comput. Oper. Res.* **2009**, *36*, 637–654. https://doi.org/10.1016/j.cor.2007.11.008.
3. Chisman, J.A. The clustered traveling salesman problem. *Comput. Oper. Res.* **1975**, *2*, 115–119. https://doi.org/10.1016/0305-0548(75)90015-5.
4. Groba, C.; Sartal, A.; Vázquez, X.H. Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. *Comput. Oper. Res.* **2015**, *56*, 22–32. https://doi.org/10.1016/j.cor.2014.10.012.
5. Cheikhrouhou, O.; Khoufi, I. A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Comput. Sci. Rev.* **2021**, *40*. https://doi.org/10.1016/j.cosrev.2021.100369.
6. Gutin, G.; Punnen, A.P. *The traveling salesman problem and its variations*; Springer: New York, 2006. https://doi.org/10.1007/b101971.
7. Wu, Z.; Wu, J.; Zhao, M.; Feng, L.; Liu, K. Two-layered ant colony system to improve engraving robot's efficiency based on a large-scale TSP model. *Neural Comput. Appl.* **2021**, *33*, 6939–6949. https://doi.org/10.1007/s00521-020-05468-4.
8. Castellani, M.; Otri, S.; Pham, D.T. Printed circuit board assembly time minimisation using a novel bees algorithm. *Comput. Ind. Eng.* **2019**, *133*, 186–194. https://doi.org/10.1016/j.cie.2019.05.015.
9. Crişan, G.C.; Pintea, C.M.; Calinescu, A.; Pop Sitar, C.; Pop, P.C. Secure traveling salesman problem with intelligent transport systems features. *Log. J. IGPL.* **2021**, *29*, 925–935. https://doi.org/10.1093/jigpal/jzaa035.

10. Cacchiani, V.; Contreras-Bolton, C.; Escobar-Falcón, L.M.; Toth, P. A matheuristic algorithm for the pollution and energy minimization traveling salesman problems. *Int. Trans. Oper. Res.* **2021**, *30*, 655–687. https://doi.org/10.1111/itor.12991.
11. Baniasadi, P.; Foumani, M.; Smith-Miles, K.; Ejov, V. A transformation technique for the clustered generalized traveling salesman problem with applications to logistics. *European J. Oper. Res.* **2020**, *285*, 444–457. https://doi.org/10.1016/j.ejor.2020.01.053.
12. Wei, Z.; Xia, C.; Yuan, X.; Sun, R.; Lyu, Z.; Shi, L.; Ji, J. The path planning scheme for joint charging and data collection in WRSNs: A multi-objective optimization method. *J. Netw. Ccomput. Appl.* **2020**, *156*, 102565. https://doi.org/doi.org/10.1016/j.jnca.2020.102565.
13. Eren, E.; Tuzkaya, U.R. Safe distance-based vehicle routing problem: Medical waste collection case study in COVID-19 pandemic. *Comput. Ind. Eng.* **2021**, *157*, 107328. https://doi.org/10.1016/j.cie.2021.107328.
14. Xu, L.; Geman, D.; Winslow, R.L. Large-scale integration of cancer microarray data identifies a robust common cancer signature. *BMC Bioinform.* **2007**, *8*, 275. https://doi.org/10.1186/1471-2105-8-275.
15. Roberti, R.; Toth, P. Models and algorithms for the asymmetric traveling salesman problem: An experimental comparison. *EURO J. Transp. Logist.* **2012**, *1*, 113–133. https://doi.org/10.1007/s13676-012-0010-0.
16. Chauhan, C.; Gupta, R.; Pathak, K. Survey of methods of solving TSP along with its implementation using dynamic programming approach. *Int. J. Comput. Appl.* **2012**, *52*, 12–19. https://doi.org/10.5120/8189-1550.
17. Volgenant, T.; Jonker, R. A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation. *European J. Oper. Res.* **1982**, *9*, 83–89. https://doi.org/10.1016/0377-2217(82)90015-7.
18. Fischetti, M.; Salazar González, J.J.; Toth, P. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Oper. Res.* **1997**, *45*, 378–394. https://doi.org/10.1287/opre.45.3.378.
19. Miliotis, P. Using cutting planes to solve the symmetric travelling salesman problem. *Math. Program.* **1978**, *15*, 177–188. https://doi.org/10.1007/BF01609016.
20. Bazylevych, R.; Kuz, B.; Kutelmakh, R.; Dupas, R.; Prasad, B.; Haxhimusa, Y.; Bazylevych, L. A parallel ring method for solving a large-scale traveling salesman problem. *Int. J. Inf. Technol. Comput. Sci.* **2016**, *8*, 5. https://doi.org/10.5815/ijitcs.2016.05.01.
21. Ali, I.M.; Essam, D.; Kasmarik, K. A novel design of differential evolution for solving discrete traveling salesman problems. *Swarm Evol. Comput.* **2020**, *52*, 100607. https://doi.org/10.1016/j.swevo.2019.100607.
22. Deng, W.; Xu, J.; Zhao, H. An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. *IEEE Access* **2019**, *7*, 20281–20292. https://doi.org/10.1109/access.2019.2897580.
23. Dorigo, M.; Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1997**, *1*, 53–66. https://doi.org/10.1109/4235.585892.
24. Huang, Y.; Shen, X.; You, X. A discrete shuffled frog-leaping algorithm based on heuristic information for traveling salesman problem. *Appl. Soft Comput.* **2021**, *102*, 107085. https://doi.org/10.1016/j.asoc.2021.107085.
25. Ilin, V.; Simić, D.; Simić, S.D.; Simić, S.; Saulić, N.; Calvo-Rolle, J.L. A hybrid genetic algorithm, list-based simulated annealing algorithm, and different heuristic algorithms for travelling salesman problem. *Log. J. IGPL.* **2022**. https://doi.org/10.1093/jigpal/jzac028.
26. Zhong, Y.; Lin, J.; Wang, L.; Zhang, H. Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem. *Swarm Evol. Comput.* **2018**, *42*, 77–88. https://doi.org/10.1016/j.swevo.2018.02.017.
27. Xu, Y.; Che, C. A Brief Review of the Intelligent Algorithm for Traveling Salesman Problem in UAV Route Planning. In Proceedings of the 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC), 2019, pp. 1–7. https://doi.org/10.1109/ICEIEC.2019.8784651.
28. Chitty, D.M. Accelerating Genetic Algorithm Evolution via Ant-based Mutation and Crossover for Application to Large-scale TSPs. In Proceedings of the Proceedings of the Genetic and Evolutionary Computation Conference Companion. Association for Computing Machinery, 2022, pp. 2046–2053. https://doi.org/10.1145/3520304.3534020.
29. Skinderowicz, R. Improving ant colony pptimization efficiency for solving large TSP instances. *Appl. Soft Comput.* **2022**, *120*. https://doi.org/10.1016/j.asoc.2022.108653.
30. Karafotias, G.; Hoogendoorn, M.; Eiben, Á.E. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Trans. Evol. Comput.* **2014**, *19*, 167–187. https://doi.org/10.1109/TEVC.2014.2308294.
31. Rego, C.; Gamboa, D.; Glover, F.; Osterman, C. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European J. Oper. Res.* **2011**, *211*, 427–441. https://doi.org/10.1016/j.ejor.2010.09.010.
32. Wang, Y. The hybrid genetic algorithm with two local optimization strategies for traveling salesman problem. *Comput. Ind. Eng.* **2014**, *70*, 124–133. https://doi.org/10.1016/j.cie.2014.01.015.
33. Zhou, Y.; Luo, Q.; Chen, H.; He, A.; Wu, J. A discrete invasive weed optimization algorithm for solving traveling salesman problem. *Neurocomputing* **2015**, *151*, 1227–1236. https://doi.org/10.1016/j.neucom.2014.01.078.
34. Créput, J.C.; Koukam, A. A memetic neural network for the Euclidean traveling salesman problem. *Neurocomputing* **2009**, *72*, 1250–1264. https://doi.org/10.1016/j.neucom.2008.01.023.
35. Zhang, P.; Wang, J.; Tian, Z.; Sun, S.; Li, J.; Yang, J. A genetic algorithm with jumping gene and heuristic operators for traveling salesman problem. *Appl. Soft Comput.* **2022**, *127*, 109339. https://doi.org/10.1016/j.asoc.2022.109339.
36. Jain, R.; Singh, K.P.; Meena, A.; Rana, K.B.; Meena, M.L.; Dangayach, G.S.; Gao, X. Application of proposed hybrid active genetic algorithm for optimization of traveling salesman problem. *Soft Comput.* **2022**. https://doi.org/10.1007/s00500-022-07581-z.

37. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools Appl.* **2021**, *80*, 8091–8126. https://doi.org/10.1007/s11042-020-10139-6.
38. Goldberg, D.E.; Lingle, R. Alleles, Loci, and the Traveling Salesman Problem. In Proceedings of the Proceedings of the 1st International Conference on Genetic Algorithms, 1985, pp. 154–159. https://doi.org/10.5555/645511.657095.
39. Deep, K.; Mebrahtu, H. New variations of order crossover for travelling salesman problem. *Int. J. Comb. Optim. Prob. Inf.* **2011**, *2*, 2–13.
40. Hussain, A.; Muhammad, Y.S.; Nauman Sajid, M.; Hussain, I.; Mohamd Shoukry, A.; Gani, S. Genetic algorithm for traveling salesman problem with modified cycle crossover operator. *Comput. Intell. Neurosci.* **2017**, *2017*, 7430125. https://doi.org/10.1155/2017/7430125.
41. Zakir, H.A. Genetic algorithm for the traveling salesman problem using sequential constructive crossover operator. *Int. J. Biom. Bioinf.* **2010**, *3*, 96–105.
42. Iqbal, Z.; Bashir, N.; Hussain, A.; Cheema, S.A. A novel completely mapped crossover operator for genetic algorithm to facilitate the traveling salesman problem. *Comput. Math. Methods* **2020**, *2*, e1122. https://doi.org/10.1002/cmm4.1122.
43. Zhang, P.; Wang, J.; Tian, Z.; Sun, S.; Li, J.; Yang, J. A genetic algorithm with jumping gene and heuristic operators for traveling salesman problem. *Appl. Soft Comput.* **2022**, *127*, 109339. https://doi.org/10.1016/j.asoc.2022.109339.
44. Alipour, M.M.; Razavi, S.N.; Feizi Derakhshi, M.R.; Balafar, M.A. A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. *Neural Comput. Appl.* **2018**, *30*, 2935–2951. https://doi.org/10.1007/s00521-017-2880-4.
45. Ganesan, V.; Sobhana, M.; Anuradha, G.; Yellamma, P.; Devi, O.R.; Prakash, K.B.; Naren, J. Quantum inspired meta-heuristic approach for optimization of genetic algorithm. *Comput. Ind. Eng.* **2021**, *94*, 107356. https://doi.org/10.1016/j.compeleceng.2021.107356.
46. Huerta, I.I.; Neira, D.A.; Ortega, D.A.; Varas, V.; Godoy, J.; Asín-Achá, R. Improving the state-of-the-art in the traveling salesman problem: An anytime automatic algorithm selection. *Expert Syst. Appl.* **2022**, *187*. https://doi.org/10.1016/j.eswa.2021.115948.
47. Ding, C.; Cheng, Y.; He, M. Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale TSPs. *Tsinghua Sci. Technol.* **2007**, *12*, 459–465. https://doi.org/10.1016/S1007-0214(07)70068-8.
48. Anaya Fuentes, G.E.; Hernández Gress, E.S.; Seck Tuoh Mora, J.C.; Medina Marín, J. Solution to travelling salesman problem by clusters and a modified multi-restart iterated local search metaheuristic. *PloS one* **2018**, *13*, e0201868. https://doi.org/10.1371/journal.pone.0201868.
49. Anantathanavit, M.; Munlin, M. Using K-means radius particle swarm optimization for the travelling salesman problem. *IETE Tech. Rev.* **2016**, *33*, 172–180. https://doi.org/10.1080/02564602.2015.1057770.
50. Yang, J.; Yang, J.; Chen, G. Solving Large-Scale TSP Using Adaptive Clustering Method. In Proceedings of the 2009 Second International Symposium on Computational Intelligence and Design, 2009, Vol. 1, pp. 49–51. https://doi.org/10.1109/ISCID.2009.19.
51. Liang, A.; Yang, H.; Sun, L.; Sun, M. A three-layered multifactorial evolutionary algorithm with parallelization for large-scale engraving path planning. *Electronics* **2022**, *11*, 1712. https://doi.org/10.3390/electronics11111712.
52. Yu, J.; You, X.; Liu, S. Dynamically induced clustering ant colony algorithm based on a coevolutionary chain. *Knowl.-Based Syst.* **2022**, *251*, 109231. https://doi.org/10.1016/j.knosys.2022.109231.
53. Honda, K.; Nagata, Y.; Ono, I. A parallel genetic algorithm with edge assembly crossover for 100,000-city scale TSPs. In Proceedings of the 2013 IEEE congress on evolutionary computation. IEEE, 2013, pp. 1278–1285. https://doi.org/10.1109/CEC.2013.6557712.
54. Wang, Z.; Shen, Y.; Li, S.; Wang, S. A fine-grained fast parallel genetic algorithm based on a ternary optical computer for solving traveling salesman problem. *J. Supercomput.* **2022**. https://doi.org/10.1007/s11227-022-04813-9.
55. Grefenstette, J.; Gopal, R.; Rosmaita, B.; Van Gucht, D. Genetic algorithms for the traveling salesman problem. In Proceedings of the Proceedings of the first International Conference on Genetic Algorithms and their Applications; Lawrence Erlbaum, , 1985; pp. 160–168.
56. Larranaga, P.; Kuijpers, C.M.H.; Murga, R.H.; Inza, I.; Dizdarevic, S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.* **1999**, *13*, 129–170. https://doi.org/10.1023/A:1006529012972.
57. Davies, L. *Genetic Algorithms and Simulated Annealing*; Morgan Kaufmann: Los Altos, 1987. https://doi.org/10.1017/S0263574700004215.
58. Ulder, N.L.; Aarts, E.H.; Bandelt, H.J.; Van Laarhoven, P.J.; Pesch, E. Genetic Local Search Algorithms for the Traveling Salesman Problem. In Proceedings of the International Conference on Parallel Problem Solving from Nature; Springer, , 1990; pp. 109–116. https://doi.org/10.1007/BFb0029740.
59. Tsai, H.K.; Yang, J.M.; Kao, C.Y. Solving Traveling Salesman Problems by Combining Global and Local Search Mechanisms. In Proceedings of the Proceedings of the Evolutionary Computation on 2002; IEEE, , 2002; pp. 1290–1295. https://doi.org/10.1109/CEC.2002.1004429.
60. Phienthrakul, T. Clustering evolutionary computation for solving travelling salesman problems. *Int. J. Adv. Comput. Sci. Inf. Technol.* **2014**, *3*, 243–262.
61. Liao, E.; Liu, C. A hierarchical algorithm based on density peaks clustering and ant colony optimization for traveling salesman problem. *IEEE Access* **2018**, *6*, 38921–38933. https://doi.org/10.1109/access.2018.2853129.

 doi:10.20944/preprints202302.0412.v1

62. Englert, M.; Röglin, H.; Vöcking, B. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. *Algorithmica* **2014**, *68*, 190–264. https://doi.org/10.1007/s00453-013-9801-4.
63. Croes, G.A. A method for solving traveling-salesman problems. *Oper. Res.* **1958**, *6*, 791–812.
64. Hougardy, S.; Zhong, X. Hard to solve instances of the Euclidean traveling salesman problem. *Math. Program. Comput.* **2021**, *13*, 51–74. https://doi.org/10.1007/s12532-020-00184-5.
65. Mariescu-Istodor, R.; Fränti, P. Solving the large-scale TSP problem in 1h: Santa Claus challenge 2020. *Front. Robot. AI* **2021**, *8*, 689908. https://doi.org/10.3389/frobt.2021.689908.