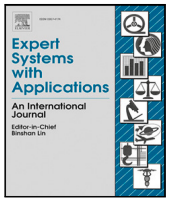
Expert Systems With Applications 240 (2024) 122472

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



# A multioperator genetic algorithm for the traveling salesman problem with job-times

Pablo Gutiérrez-Aguirre, Carlos Contreras-Bolton *

*Departamento de Ingeniería Industrial, Universidad de Concepción, Concepción, Chile*

ARTICLE INFO

*Keywords:*
Traveling salesman problem
Scheduling problem
Genetic algorithm
Multioperators

ABSTRACT

This paper addresses the traveling salesman problem with job-times (TSPJ). TSPJ considers two sets of equal size, a set of tasks and a set of vertices, where a traveler must visit each vertex exactly once, return to the starting vertex, and perform a unique task at each vertex. Each task is assigned to only one vertex, and each task is completed with a job-time that depends on each vertex. Thus, the objective is to minimize the time of the last task performed. However, due to its NP-hardness, existing algorithms do not optimally solve the TSPJ larger instances. Therefore, we propose an approach based on a multioperator genetic algorithm (MGA) that utilizes various initial population procedures, crossover and mutation operators. MGA applies five initial population procedures to generate a diverse population, and employs three crossover operators and six mutation operators, with four mutations focused on diversification and two designed to aid intensification. Furthermore, to improve the quality of the best individual found, a local search is used in every generation, and generational replacement with elitism is considered when generating a new population. MGA is evaluated on four sets of instances ranging in size from 17 to 1200 vertices: tsplib, small, medium, and large. Our approach outperforms the state-of-the-art algorithms on the four instance sets. This performance is attributed to the synergistic effect of the multioperators of crossover and mutation, along with effective parameter tuning.

## 1. Introduction

The traveling salesman problem (TSP) is a classic problem in operations research that has been extensively studied since the early 19th century, with applications mainly in transportation and logistics. The TSP involves finding the minimum cost route that passes through a number of customers the salesman must visit exactly once, starting and returning to the same starting point (Applegate et al., 2007). Despite its seemingly simple description, solving the TSP is a challenging task as it belongs to the class of NP-complete problems (Karp, 1972). Due to the TSP's complexity and significance in real-world applications, it continues to attract significant research attention, and numerous approaches have been proposed to tackle it. Consequently, the TSP remains a significant challenge today, especially when considering its several extensions and variants, such as (Cacchiani et al., 2023; He & Hao, 2023; Mardones et al., 2023; Pralet, 2023).

The scheduling problem (SP) is another classic problem in operations research that plays a critical role in manufacturing. SP consists of assigning tasks to resources over time, subject to various constraints, with the goal of optimizing one or more objectives. Manufacturing companies, for instance, must meet delivery dates promised to customers, and any non-compliance can result in significant financial losses. The SP encompasses various cases involving sequential, parallel, related, and independent machines, each with different objectives, such as minimizing makespan, maximum lateness, total tardiness, completion time, among others (Pinedo, 2016). Given the complexity and importance of the SP, it remains an active area of research, and researchers continue to propose numerous approaches to solving it.

Some variants of the TSP are related to the SP, such as a case where a machine has a set of jobs and sequence-dependent service times, and the objective is to minimize the makespan or completion time of the last task. This problem is called the single machine scheduling problem (SMSP) and is equivalent to the TSP, wherein the sequence dependent times are the travel times between vertices (Lawler et al., 1993).

Another SP that is related to the TSP is the unrelated parallel machine scheduling problem (UPMSP). UPMSP consists of scheduling a set of jobs, each available at time zero, on a set of unrelated machines without preemption. The time between machines will be the travel time, and the time to complete the tasks will be the job-time. UPMSP can be formulated as a variant of the TSP called the TSP with job-times (TSPJ), which was recently defined by Mosayebi et al. (2021). TSPJ consists of a set of tasks and a set of vertices, both of the same size,

* Corresponding author.
*E-mail addresses:* pgutierrez2018@udec.cl (P. Gutiérrez-Aguirre), carlos.contreras.b@udec.cl (C. Contreras-Bolton).

https://doi.org/10.1016/j.eswa.2023.122472
Received 13 April 2023; Received in revised form 15 September 2023; Accepted 4 November 2023
Available online 14 November 2023

where a traveler must move through all the vertices, visiting each vertex exactly once and returning to the starting vertex. Furthermore, the traveler must perform a unique task at each vertex without repeating any tasks, and the time required to complete each task depends on the vertex. The objective is to minimize the time of the last task performed.

The TSPJ has applications in various fields, such as petroleum extraction, where proactive maintenance or reactive repairs dictate when to stop extraction. Another application is in automated manufacturing environments where machining centers can process workpieces without oversight. TSPJ also finds applications in equipment management for disaster recovery and highly automated harvesting (Mosayebi et al., 2021).

Although state-of-the-art algorithms can provide good solutions for existing instances of the TSPJ, their performance deteriorates as the instances grow in size due to the mixed nature of TSPJ, which combines elements of both TSP and SP. To address this problem, this paper proposes a multioperator genetic algorithm for the TSPJ. The main contributions of this study can be summarized as follows:

- We tested the existing mixed integer linear programming (MILP) model with additional subtour elimination constraints on all instance sets in a general-purpose solver, obtaining upper and lower bounds (not existing) for all the instances.
- A multioperator genetic algorithm is proposed that works with different initial population procedures, mutation, and crossover operators. Specifically, with five procedures to generate the initial population, three classic crossover operators based on permutations, and six mutation operators, two of which are used to intensify the solution, and four are used to diversify. In addition, effective parameter tuning of our approach is performed.
- Extensive computational experiments are conducted on all instance sets to demonstrate the effectiveness of our algorithm. The results show that our approach outperforms existing algorithms in the literature. Moreover, additional experiments are reported to shed light on the multioperators' influences and parameter tuning on the algorithm's performance.

The remaining sections of this document are organized as follows: Section 2 reviews the literature on similar problems. In Section 3, we formally define the problem, together with the mathematical model. Section 4 details the multioperator genetic algorithm, explaining its functioning and each of its operators. Section 5 presents and discusses the extensive computational experiments. Finally, in Section 6, we give concluding remarks and propose future research.

## 2. Related work

In this section, we review the main works on the SMSP and UPMSP, due to near with the TSPJ. Additionally, we review variants of TSP that integrate with SP.

SMSP has been extensively studied, and two approaches have been used to address it: exact and heuristic methods (Chen et al., 1998; Martinelli et al., 2022). Similarly, UPMSP has been widely researched using various approaches, including exact and heuristic algorithms (Durasević & Jakobović, 2023; Moser et al., 2022).

In the TSP literature exists an extension called time-dependent TSP (TDTSP), which considers the time dependency of travel times, taking into account the variability of these times (Fox et al., 1980). In several papers (Bigras et al., 2008; Cacchiani et al., 2020; Picard & Queyranne, 1978), the arc cost depends on the arc's position in the TSP tour. Therefore, TDTSP may be stated as an SP in which a set of jobs must be processed on a single machine at a minimum cost.

Another variant of TSP is the minimum latency problem (MLP) (Blum et al., 1994), which is also known by several other names as the traveling repairman problem (Tsitsiklis, 1992), the deliveryman problem (Fischetti et al., 1993), the TSP with cumulative costs (Bianco et al., 1993), and the school bus driver problem (Will, 1993). These problems deal with a repairman who wants to visit their customers to minimize the overall waiting times of the customers located in vertices. Thus, these problems can be interpreted as an SMSP with sequence-dependent processing times, where the total flow time of the work must be minimized. Note that in the MLP, the objective is to minimize the total latency time for all customers, while in the TSP, the focus is on minimizing the travel time of a single traveling salesman.

Mosayebi et al. (2021) recently proposed the TSPJ and a MILP model for tackling this new problem. The authors also introduced four hybridized versions of classic heuristics for the TSP that are adapted for the TSPJ. These heuristics are based on the nearest neighbor heuristic, reverse assignment, 2-opt, and local search improvement. Finally, the authors evaluated the model and four heuristics on 310 benchmark instances and employed the CPLEX solver to solve the MILP model. The computational results showed that the CPLEX solver can solve the MILP model optimally only for small instances, while the heuristic algorithms proved effective for medium and large instances.

Integrating the SP and the TSP involves optimizing both scheduling and routing decisions simultaneously, considering constraints such as delivery deadlines, resource availability, and the TSP objective function. This can lead to more efficient use of resources, reduced travel times, and improved delivery performance. However, this integrated consideration has been mainly with routing problems since it has wide-ranging applications in different industries and domains, such as transportation and logistics, manufacturing, healthcare, and emergency response. For instance, in transportation and logistics, integrated problems can optimize the delivery routes of vehicles, reduce travel times, and improve delivery efficiency (Chen et al., 2022; Chevroton et al., 2021; Lam & Ip, 2019; Liang et al., 2023). In manufacturing, integrated problems can optimize the scheduling of production processes and the routing of materials, components, and finished goods (Berghman et al., 2023; Bourreau et al., 2022; Fu et al., 2017; Lyu et al., 2019; Zou et al., 2018). In healthcare, integrated problems can optimize the scheduling of medical procedures and routing patients, medical equipment, and supplies (Di Mascolo et al., 2021; Fikar & Hirsch, 2017; Yadav & Tanksale, 2022; Yu et al., 2021). Finally, in emergency response, the integrated problems can optimize the dispatch of emergency responders and the routing of emergency vehicles to reduce response times and save lives (Bodaghi et al., 2020; Maya-Duque et al., 2016; Moreno et al., 2019; Özdamar & Ertem, 2015).

## 3. Traveling salesman problem with job-times

Formally, the TSPJ can be defined as an undirected complete graph, $G = (V, A)$ where $V = \{0, 1, \dots, n\}$ is a set of vertices with 0 as the depot and $\{1, \dots, n\}$ as the vertices to visit. While the set of vertices without the depot is $\overline{V} = V \setminus \{0\}$. $A$ is a set of arcs, where $A = \{(i, k) : i, k \in V, i \neq k\}$. Each arc $(i, k) \in A$ has associated a nonnegative value $d_{ik}$ that represents the travel distance from vertex $i$ to vertex $k$. In addition, $J = \{1, \dots, n\}$ is a set of jobs to be performed, where each job has a processing time $p_{ij}, i \in \overline{V}, j \in J$ and no jobs are performed in the depot. The objective is to find a route from vertex 0 passing through the $n$ vertices, visiting each vertex exactly once, and returning to the start, i.e., a Hamiltonian circuit. Thus, the Hamiltonian circuit must consider performing the $n$ jobs so that each job is assigned to only one vertex, without repeating, minimizing the maximum completion time (makespan).

Fig. 1 illustrates an example of the TSPJ featuring a depot, four vertices, and four jobs. In this example, we can see the travel times on each arc and the processing times at each vertex as arrows. Each arrow shows the job number and its corresponding job-time. The route and the assigned jobs of the feasible solution for the TSPJ are marked in blue. Moreover, Table 1 provides details of the feasible solution of Fig. 1, where the traveler begins the route from the depot, visits vertex 1, initiates job 3, continues to vertex 2, starts job 4, moves to vertex 3,

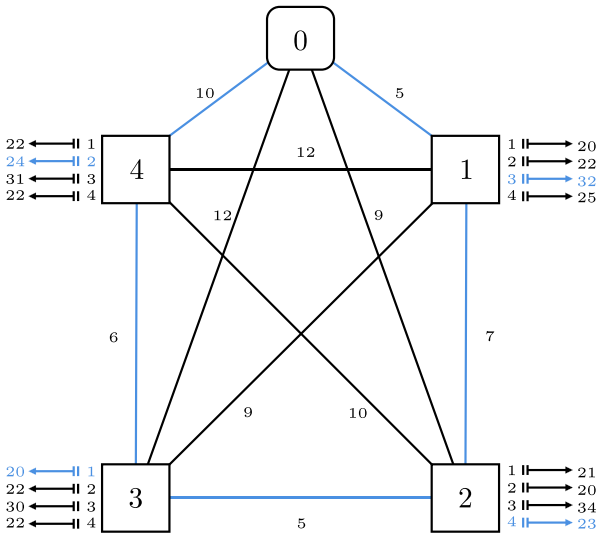



**Fig. 1.** Example of an instance with a depot, four vertices and four jobs.

**Table 1**
The computing of $C_{max}$ from the example.

| Vertices sequence | 0 | 1 | 2 | 3 | 4 | 0 |
|---|---|---|---|---|---|---|
| Jobs assignment | – | 3 | 4 | 1 | 2 | – |
| TSP route to vertex | 0 | 5 | 12 | 17 | 23 | 33 |
| Job-time | 0 | 32 | 23 | 20 | 24 | 0 |
| Completion time | 0 | 37 | 35 | 37 | **47** | 33 |

**Table 2**
Nomenclature of the MILP model.

| | |
|---|---|
| **Sets** | |
| $V$ | Set of vertices $\{0, 1, \dots, n\}$. |
| $\overline{V}$ | Set of vertices without depot $\{1, \dots, n\}$. |
| $J$ | Sets of jobs $\{1, \dots, n\}$. |
| $A$ | Set of arcs $\{(i,k) : i,k \in V, i \neq k\}$. |
| **Parameters** | |
| $d_{ik}$ | Travel distance associated the arcs $(i,k) \in A$. |
| $p_{ij}$ | Processing time of job $j \in J$ at vertex $i \in \overline{V}$. |
| **Decision variables** | |
| $x_{ik}$ | Represents the travel from vertex $i$ to vertex $k$ $((i,k) \in A)$. |
| $y_{ij}$ | Represents the assignment of job $j$ to vertex $i$ $(j \in J,\ i \in \overline{V})$. |
| $t_i$ | Represents the actual time of starting the job at vertex $i \in V$. |
| $C_{max}$ | Maximum completion time of the jobs in all vertices. |

begins job 1, continues to vertex 4, initiates the last job 2, and finally returns to the depot. The total travel time for this solution is 33 units, and the maximum completion time is 47.

The TSPJ can be formulated as a MILP model that was introduced by Mosayebi et al. (2021). The formulation uses the following variables: $x_{ik}$, $(i,k) \in A$ is a binary variable that takes the value 1 if the vertex $k$ is visited immediately after that vertex $i$, and value 0 otherwise. $y_{ij}$ is a binary decision variable that denotes the assignment of job $j \in J$ to vertex $i \in \overline{V}$. In addition, $t_i$ is a nonnegative continuous decision variable representing the arrival time and also the time to start work of the job assigned to vertex $i \in V$. Finally, $C_{max}$ is a nonnegative continuous decision variable that keeps the maximum completion time of the jobs in all vertices. To ease presentation, we summarize the sets, the parameters, and the decision variables used for the MILP model and list them in Table 2. Therefore, the MILP model for the TSPJ reads as follows:

$$\text{minimize } C_{max} \tag{1}$$

subject to:

$$C_{max} \geq t_i + \sum_{j \in J} p_{ij} y_{ij} \qquad \forall i \in \overline{V} \tag{2}$$

$$C_{max} \geq t_i + d_{i0} x_{i0} \qquad \forall i \in \overline{V} \tag{3}$$

$$\sum_{i \in \overline{V}} y_{ij} = 1 \qquad \forall j \in J \tag{4}$$

$$\sum_{j \in J} y_{ij} = 1 \qquad \forall i \in \overline{V} \tag{5}$$

$$\sum_{k \in \overline{V} : i \neq k} x_{ik} = 1 \qquad \forall i \in V \tag{6}$$

$$\sum_{i \in \overline{V} : i \neq k} x_{ik} = 1 \qquad \forall k \in V \tag{7}$$

$$t_i + d_{ik} - M(1 - x_{ik}) \leq t_k \qquad \forall i \in V, k \in \overline{V} : i \neq k \tag{8}$$

$$x_{ik} \in \{0,1\} \qquad \forall (i,k) \in A \tag{9}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in \overline{V}, j \in J \tag{10}$$

$$t_i \geq 0 \qquad \forall i \in V \tag{11}$$

The objective function (1) minimizes the maximum completion time, which is computed by constraints (2) and (3). Constraint (2) guarantees that the makespan is greater or equal to the starting time of every vertex plus its processing time. Constraint (3) ensures that the makespan is greater or equal to the completion time of the last operation plus the travel distance required to return to the depot. Constraints (4) and (5) ensure that assign only one job to each vertex. Constraints (6) and (7) impose have exactly one outgoing arc from each vertex and one ingoing arc for each vertex, respectively. Constraints (8) follow the sequence and the time of starting the jobs in each vertex, when the traveler moves from vertex $i$ to vertex $j$, the starting time of the job at vertex $j$ has to be greater or equal than the time of starting job at vertex $i$ plus the travel-time between vertices $i$ and $j$. $M$ is a fixed large number and can be defined as $M = \sum_{i \in \overline{V}} \max_{j \in J} \{p_{ij}\}$. Finally, constraints (9)–(11) define the domain of the decision variables.

In order to strengthen model (1)–(11), we add an improved lower bound of the maximum completion time given by constraint (12).

$$C_{max} \geq \min_{i \in \overline{V}, j \in J} \{p_{ij}\} + \sum_{i \in V} \sum_{k \in \overline{V} : i \neq k} d_{ik} x_{ik} \tag{12}$$

### 3.1. Additional subtour elimination constraints

We consider three additional subtour elimination constraints, which are the Miller, Tucker and Zemlin (MTZ) formulation (Miller et al., 1960), the Desrochers and Laporte (DL) formulation (Desrochers & Laporte, 1991), and the Gavish and Graves (GG) formulation (Gavish & Graves, 1978), in order to enhance the performance of the model. In other words, these constraints can be used as valid inequalities in the model (1)–(12), they are not needed to eliminate subtours since constraints (8) inherently prevent them.

#### 3.1.1. MTZ formulation

The MTZ formulation considers a dummy variable $u_i (i \in \overline{V})$, denoting the position in which vertex $i$ is visited in the tour. Thus, constraints (13) enforce that only a single tour covers all vertices instead of two or more disjointed tours collectively covering all vertices. Constraints (14) require $u$ variables as nonnegative and define their upper bounds.

$$u_i - u_k + n x_{ik} \leq n - 1 \qquad \forall i \in \overline{V}, k \in V : i \neq k \tag{13}$$

$$0 \leq u_i \leq n - 1 \qquad \forall i \in \overline{V} \tag{14}$$

#### 3.1.2. DL formulation

The DL formulation is an improved MTZ formulation that results in a stronger linear programming relaxation. Therefore, this formulation also considers a dummy variable $u_i (i \in \overline{V})$ with the lifted constraints defined by Desrochers and Laporte (1991). Thus, constraints (15) are

facet defining, while constraints (16) and (17) are valid inequalities, and constraints (18) as the domain of the decision variables.

$$u_i - u_k + (n-1)x_{ik} + (n-3)x_{ki} \leq n-2 \qquad \forall i \in \overline{V}, k \in V : i \neq k \qquad (15)$$

$$u_i \geq 1 + (n-3)x_{i0} + \sum_{k \in \overline{V}, i \neq k} x_{ki} \qquad \forall i \in \overline{V} \qquad (16)$$

$$u_i \leq n-1-(n-3)x_{0i} - \sum_{k \in \overline{V}, i \neq k} x_{ik} \qquad \forall i \in \overline{V} \qquad (17)$$

$$1 \leq u_i \leq n-1 \qquad \forall i \in \overline{V} \qquad (18)$$

#### 3.1.3. GG formulation

The GG formulation considers $g_{ik} (i \in \overline{V}, k \in V)$ a non-negative variable denoting the number of arcs in a path from depot 0 to arc $(i,k) \in A$. Thus, constraints (19) require that the number of arcs in a path from a depot 0 to a vertex $i$ (whatever successor vertex is selected from $i$) must be one unit larger than the number of arcs in a path from depot 0 to the predecessor of vertex $i$ (whatever vertex it is). While the constraints (20) require $g$ variables to be non-negative and restrict their value to be 0, if arc $(i,k)$ is not selected, and at most $n$, if it is selected.

$$\sum_{k \in V : i \neq k} g_{ik} - \sum_{k \in V : i \neq j} g_{ki} = 1 \qquad \forall i \in \overline{V} \qquad (19)$$

$$0 \leq g_{ik} \leq n x_{ik} \qquad \forall i \in \overline{V}, k \in V : i \neq k \qquad (20)$$

## 4. Multioperator genetic algorithm

Genetic algorithms (GA) are effective metaheuristic algorithms that belong to the family of evolutionary algorithms. GA is based on the evolution of a population of individuals, corresponding to solutions to the considered problem. Each solution goes through three operators called selection, crossover, and mutation to search for the best solution (Eiben & Smith, 2015; Talbi, 2009). However, classic GA uses only single operators for crossover and mutation, disregarding the potential synergy of multioperators. Since researchers have found that the multioperators based on hybrid heuristics generate better solutions, taking advantage of utilizing the benefits of different heuristic approaches together (Asadujjaman et al., 2022; Contreras-Bolton et al., 2016; Elsayed et al., 2011; Liu et al., 2022). This section presents our proposed algorithm, which consists of a multioperator genetic algorithm (MGA). In addition, we describe the evolutionary process of the MGA, solution representation, initial population generation, crossover, and mutation operators.

**Algorithm 1:** Pseudocode of the MGA

```
   Output: Solution
1  g ← 0
2  for t ← 0 to N^i do
3  |  P_t^g ← initial-population()
4  |  evaluate-individual(P_t^g)
5  repeat
6  |  g ← g + 1
7  |  for t ← 0 to N^i do
8  |  |  (a, e) ← selection(P^{g-1})
9  |  |  P_t^g ← crossover(P_a^{g-1}, P_e^{g-1})
10 |  |  P_t^g ← mutation(P_t^g)
11 |  for t ← 0 to N^i do
12 |  |  evaluate-individual(P_t^g)
13 |  LSJA(P_b^g), b is best individual of P^g
14 |  P^g ← elitism(P^{g-1})
15 until g ≤ N^g;
```

### 4.1. Overall algorithm

The functioning of MGA is described in Algorithm 1 and starts generating a population of individuals ($P^g$) of size $N^i$ by five initial population procedures and the evaluation of generated individuals (lines 1–4). Subsequently, the algorithm's main loop consists of the evolutionary process, presented in lines 5–15, which is responsible for generating a new population from the current one via operators of selection, crossover, and mutation (lines 7–10). The selection operator is made in a tournament of $N^t$ individuals (Eiben & Smith, 2015) in line 8. The crossover considers three operators (line 9), and six operators participate in the mutation (line 10). The resulting offspring constitute the new population, which is evaluated in lines 11–12. In order to further improve the quality of the best individual of the current generation, a local search (LSJA, see Section 4.5 for details) is applied to it in line 13. Finally, we consider a generational replacement with elitism is performed in line 14, where the newly generated population replaces the entire previous population. Meanwhile, elitism consists of replacing the $\rho^e$ percent of the worst individuals in the current population with the best individuals of the prior population. This elitism strategy ensures that the best solutions found so far are not lost. The evolutionary process is repeated until the maximum number of generations $N^g$ is reached, at which point MGA returns the best solution found during the evolutionary process.

### 4.2. Solution representation

Each individual $t$ in the population $P_t^g$ ($g \in \{0, \ldots, N^g\}$) consists of two chromosomes, $P_t^g = \{\pi^1, \pi^2\}$, which work together to generate a feasible solution. Specifically, the first chromosome $\pi^1$ represents a Hamiltonian route and is a permutation of vertices where the $l$th position can take a value $l \in \overline{V}$. The second chromosome $\pi^2$ represents the jobs assignment to vertices in the Hamiltonian route. Therefore, the $l$th position of $\pi^2$ is assigned to the vertex in the $l$th position of $\pi^1$, thereby associating each vertex with a job. Thus, the pairing of $(\pi_i^1, \pi_i^2)$, where $l \in \overline{V}$, generates a feasible solution that assigns each job to a vertex in the Hamiltonian route. Finally, to evaluate the individual $t$, the constructed route must start and end at the depot (vertex 0) and is evaluated with the objective function given by Eq. (21).

$$f(P_t^g) = \max\{\ p_{\pi_0^1,\pi_0^2} + d_{0,\pi_0^1}, \max_{i \in \{1,\ldots,n-2\}} \{p_{\pi_i^1,\pi_i^2} + d_{0,\pi_0^1} + \sum_{l=1}^{i} d_{\pi_{l-1}^1,\pi_l^1}\}, d_{0,\pi_0^1} + \sum_{l=0}^{n-2} d_{\pi_l^1,\pi_{l+1}^1} + d_{\pi_{n-1}^1,0}\} \qquad (21)$$

Fig. 2 illustrates the chromosome representation used in the example presented in Fig. 1. On the left-hand side, the first chromosome ($\pi^1$) depicts the Hamiltonian route, while the right-hand side shows the second chromosome ($\pi^2$) that assigns jobs to vertices. To construct a feasible solution from these chromosomes of example, we first use the $\pi^1$ chromosome to obtain the route 1, 2, 3, 4. Next, we assign each vertex of the route to the corresponding job indicated by the $\pi^2$ chromosome. In this case, vertex 1 is assigned job 3, vertex 2 is assigned job 4, and so on, resulting in the vertex-job pairings $(1,3), (2,4), (3,1), (4,2)$. Finally, we add the depot at the beginning and end of the route to obtain the feasible solution shown in Fig. 1.

### 4.3. Initial population

As each individual has two chromosomes, there are procedures to build routes and others for the jobs assignment. There are three procedures to construct the route: (i) random route (RR), (ii) nearest neighbor (NN), and (iii) TSP solver (TSPS). These procedures are chosen randomly based on probabilities $\alpha_1$, $\alpha_2$, and $\alpha_3$, respectively. Similarly, there are two procedures to build the jobs assignment: (iv) random

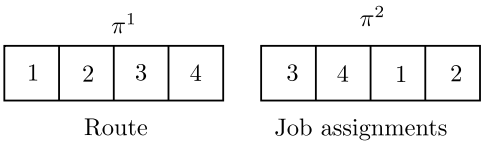



**Fig. 2.** Individual representation.

assignment permutation (RAP) and (v) nearest neighbor for jobs assignment (NNJA). The selection of the job assignment procedure is also based on probabilities $\alpha_4$, and $\alpha_5$, respectively. Next, the description of the procedures.

(i) RR: The route is generated randomly. This procedure is applied with probability $\alpha_1$.
(ii) NN: The tour is generated by applying the well-known nearest neighbor heuristic (Flood, 1956), starting from a random vertex. This procedure is applied with probability $\alpha_2$.
(iii) TSPS: Since the computing times increase as size increases. An exact method is employed if the instance size is smaller than 150 vertices. Otherwise, a heuristic method is used. This procedure is applied with a probability $\alpha_3$.

- Exact method: Route is obtained by solving the TSP through the classical model of two indices using a general-purpose solver. Therefore, the integer linear programming model for the TSP reads as follows:

$$\text{minimize} \quad \sum_{(i,j)\in A} \overline{x}_{ik} d_{ik} \tag{22}$$

$$\text{subject to} \quad \sum_{k\in V: i\neq k, i<k} \overline{x}_{ik} = 2 \quad \forall i \in V \tag{23}$$

$$\overline{x}_{ik} \in \{0,1\} \quad \forall (i,j) \in A \tag{24}$$

where $\overline{x}_{ik}((i,j) \in A)$ determines whether or not the salesman travels directly from vertex $i$ to vertex $k$, the TSP is solved through (22), (23) and (24) relaxed model by adding explicit subtour elimination constraints proposed by Dantzig, Fulkerson, and Johnson (DFJ) formulation (Dantzig et al., 1954), as shown in constraints (25). Constraints (25) ensure that any subset of vertices $S$ of size $2 \leq |S| \leq |V|-1$ must have at least one outgoing arc. These DFJ subtour elimination constraints are added iteratively until the route ($\pi^1$) no longer contains any subtours (Gutin & Punnen, 2007).

$$\sum_{i\in S} \sum_{k\in S: i\neq k} \overline{x}_{ik} \leq |S| - 1 \quad \forall S \subset V : 2 \leq |S| \leq |V| - 2 \tag{25}$$

- Heuristic method: Route is obtained by the Lin-Kernighan-Helsgaun (LKH) algorithm (Helsgaun, 2000, 2017), which is a state-of-the-art algorithm widely used for the TSP. The LKH algorithm is run for 10,000 iterations to obtain the route.

(iv) RAP: The jobs assignment is generated randomly. This procedure is applied with probability $\alpha_4$.
(v) NNJA: This algorithm receives the route created previously as input, then it starts in the last vertex, assigning the job with the shortest processing time available. Thus, the jobs with the largest processing time will be in the first places of the tour. This algorithm is applied with a probability $\alpha_5$.

Note above procedures is chosen according to a randomly generated value ($\omega$) between 0 and 1 that matches some given probabilities of the procedures for route and jobs assignment, which are determined by the tuning procedure later. Thus, the ranges of the probabilities for RR ($0 \leq \omega \leq \alpha_1$), NN ($\alpha_1 < \omega \leq \alpha_1 + \alpha_2$), TSPS ($1 - \alpha_3 < \omega \leq 1$), where these probabilities are determined by the conditions $\alpha_1 + \alpha_2 + \alpha_3 = 1$. While, RAP ($0 \leq \omega \leq \alpha_4$), and NNJA ($1 - \alpha_5 < \omega \leq 1$) with $\alpha_4 + \alpha_5 = 1$.

### 4.4. Crossover

Three different crossover operators are applied to both chromosomes: order-based crossover (OX) (Davis, 1985), partially matched crossover (PMX) (Goldberg & Lingle, 1985), and uniform partially matched crossover (UPMX) (Cicirello & Smith, 2000). The probabilities of using each of these operators ($\beta_1$, $\beta_2$, and $\beta_3$, with $\beta_1+\beta_2+\beta_3 = 1$) are determined by a tuning procedure. Here is a brief description of each operator:

- OX: This operator selects a random subset of genes from one parent and inserts them into the offspring in the same relative order as in the parent. The remaining genes are then added to the offspring in the order in which they appear in the other parent, avoiding any duplicates.
- PMX: This operator starts by selecting two random cutpoints defining a gene section to be exchanged between parents. Then, the two exchanged sections are matched such that any duplicate genes are swapped with their corresponding duplicates in the opposite section. Once the matching is completed, the offspring is created by copying the remaining genes from the parents, excluding any duplicates that are already present in the exchanged sections.
- UPMX: This operator is similar to PMX, but instead of selecting a contiguous subset of genes, it randomly selects a subset of genes from both parents with equal probability and maps them following the same rules as in PMX.

### 4.5. Mutation

Six different mutation operators can be applied, four operators are diversification, and two remaining operators are intensification. The operators that diversify the solution are (i) vertex exchange mutation (VEM) (Banzhaf, 1990), (ii) job exchange mutation (JEM), (iii) reverse mutation (RM) (Fogel & Atmar, 1990), and (iv) successor mutation (SM). While the operators that intensify the solution are (v) 2-opt (Croes, 1958; Morton & Land, 1955), and (vi) local search for jobs assignment (LSJA). In addition, the mutation operators: VEM, RM, SM, and 2-opt work on the route ($\pi^1$), and JEM and LSJA work for the jobs assignment ($\pi^2$). Next, the description of the operators.

(i) VEM: This operator consists of a random exchange of two vertices.
(ii) JEM: This operator consists of a random exchange of two jobs.
(iii) RM: This operator reverses the subtour order between two randomly chosen vertex.
(iv) SM: This operator performs an exchange between a random vertex and its successor.
(v) 2-opt: This operator is a well-known local search heuristic that improves routes by iteratively exchanging two arcs. Our implementation uses the version that terminates when the first improvement is found.
(vi) LSJA: This operator is described in Algorithm 2 and first identifies the vertex $u \in \overline{V}$ with the current maximum completion time (lines 1–2). It then proceeds to iteratively exchange the job assigned to vertex $u$ with those of the remaining vertices $v \in \overline{V} : u \neq v$ in an attempt to improve the completion time (lines 3–8). The operator begins from the first vertex and works its way through the rest of the vertices. In order to determine

whether a job exchange results in an improvement, the operator checks whether the completion time at vertex $u$ decreases while the completion time at vertex $v$ stays the same or decreases as well (line 6). To efficiently recalculate the maximum completion time, the operator only needs to compute the new completion time at vertices $u$ and $v$, and take the maximum of those values as the new makespan.

**Algorithm 2:** Pseudocode of the LSJA

```
Input : π¹, π²
1  f_u ← makespan(π¹, π²)
2  u ← get-index(f_u)
3  for v ← 1 to n do
4      if v ≠ u then
5          f̄_u, f̄_v ← makespan(π¹, π²) by swapping tasks u and v
6          if f̄_u < f_u and f̄_v < f_u then
7              swap(π², u, v)
8              break
```

The mutation operators for the route and jobs assignment can be executed together or separately, according to the $\delta_1$ and $\delta_2$. Furthermore, each operator has a probability of being used, represented by $\delta_3, \ldots, \delta_8$. The functioning of probabilities of the mutation operators is described in Algorithm 3. Note that the probabilities are tuning parameters, and the sum of the probabilities is determined by the conditions $\delta_3 + \delta_4 + \delta_5 + \delta_6 = 1$ and $\delta_7 + \delta_8 = 1$.

### 4.6. Computational complexity analysis of MGA

When assessing the computational time of any metaheuristic algorithm, it is crucial to account for the computational complexity related to the algorithm's design and implementation. Thus, the computational complexity of the proposed MGA can be mainly attributed to four factors: the initial population, fitness function evaluation, crossover operator, and mutation operator. The computational complexities of all these operators are detailed in Table 3. Consequently, the computational complexity of the initial population is predominantly influenced by the TSPS, in particular, using the LKH algorithm since this one is used in medium and large instances. Therefore, it is applied to the entire population in the worst-case scenario, resulting in a complexity of $O(N^i \times n^{2.2})$, according to Helsgaun (2000). Subsequently, the computational complexity of the crossover operators is identical for all operators, as all share the same computational complexity. Thus, in the worst-case scenario, any operator is applied to every individual in the population in each generation, resulting in a complexity of $O(N^g \times N^i \times n)$. Similarly, the mutation operators are applied to all individuals of the population in each generation, and its complexity is primarily determined by the 2-opt operator, so, under the worst-case scenario, it yields a complexity of $O(N^g \times N^i \times n^2)$. Meanwhile, the fitness function's complexity stands at $O(N^g \times N^i \times n)$ as it computes the fitness for every individual in the population in each generation. Finally, the overall computational complexity of the MGA can be expressed as $O(N^i \times n^{2.2} + N^g \times N^i \times (2n + n^2))$.

## 5. Computational experiments

The models and algorithms were implemented in Python 3.9. The models were solved using the general-purpose solver Gurobi 9.5.0 and the LKH algorithm 3.0, implemented in C (Helsgaun, 2017). Additionally, the four heuristics proposed by Mosayebi et al. (2021) were obtained from Mosayebi (2020) and implemented in Python 3.9.

**Algorithm 3:** Pseudocode of the Mutation operator

```
Input : P_t^g
1  ω ← random(0, 1)
2  if ω ≤ δ_1 then
3      ω ← random(0, 1)
4      if ω ≤ δ_3 then
5          P_t^g.π¹ ← VEM(P_t^g.π¹)
6      else if ω ≤ δ_3 + δ_4 then
7          P_t^g.π¹ ← RM(P_t^g.π¹)
8      else if ω ≤ δ_3 + δ_4 + δ_5 then
9          P_t^g.π¹ ← SM(P_t^g.π¹)
10     else                                  /* ω ≤ δ_3 + δ_4 + δ_5 + δ_6 */
11         P_t^g.π¹ ← 2-opt(P_t^g.π¹)
12 ω ← random(0, 1)
13 if ω ≤ δ_2 then
14     ω ← random(0, 1)
15     if ω ≤ δ_7 then
16         P_t^g.π² ← LSJA(P_t^g.π¹, P_t^g.π²)
17     else                                  /* ω ≤ δ_7 + δ_8 */
18         P_t^g.π² ← JEM(P_t^g.π²)
```

The experiments were conducted on the supercomputing infrastructure of the NLHPC, utilizing 2 × Intel Xeon Gold 6152 processors with 2.10 GHz, each with 22 cores and running 768 GB RAM. All experiments were run with a single thread using the CentOS Linux 7 operating system (64-bit). We have made all benchmarking instances, along with the detailed results for each set of instances and the source code, available online at the following URL[1].

### 5.1. Benchmarking instances

We considered four sets of instances proposed by Mosayebi et al. (2021), where the job processing time in different vertices is generated randomly and must be between 50 to 80 percent of the optimal tour length. The sets of instances are the following:

- tsplib: This set comprises classic instances from the TSPLIB (Reinelt, 1991) containing between 17 to 101 vertices, consists ten instances.
- small: This set includes 100 instances with distances between vertices ranging from 10 to 50 and sizes between 40 to 50 vertices.
- medium: This set contains 100 instances with distances between vertices ranging from 50 to 200 and sizes between 400 to 500 vertices.
- large: This set comprises 100 instances with distances between vertices ranging from 200 to 500 and sizes between 1000 to 1200 vertices.

### 5.2. Parameter settings

Our MGA considers 22 parameters: $\alpha_1$, $\alpha_2$, $\alpha_3$, $\alpha_4$, and $\alpha_5$ correspond to the probabilities of the callings of the procedures to construct the initial solution; $\beta_1$, $\beta_2$, and $\beta_3$ are the probabilities of crossover operators; $\delta_1$ to $\delta_8$ are the probabilities of mutation operators; $N^i$ is the population size; $N^g$ is the number of generations; $N^t$ is the size of the tournament of the selection operator. Finally, $\rho^c$, $\rho^m$, and $\rho^e$ are the probability of crossover, mutation, and elitism, respectively.

Given the stochastic nature of evolutionary algorithms and the large number of parameters involved, accurate parameter tuning is crucial

[1] https://github.com/pabloojavier/TSPJ_git

**Table 3**
The computational complexities of initial population, crossover, and mutation procedures.

| Initial population | Computational complexity | Crossover operator | Computational complexity | Mutation operator | Computational complexity |
|---|---|---|---|---|---|
| RR | $O(n)$ | OX | $O(n)$ | VEM | $O(n)$ |
| NN | $O(n^2)$ | PMX | $O(n)$ | JEM | $O(n)$ |
| TSPS | $O(n^{2.2})$ | UPMX | $O(n)$ | RM | $O(n)$ |
| RAP | $O(n)$ | – | – | SM | $O(n)$ |
| NNJA | $O(n^2)$ | – | – | 2-opt | $O(n^2)$ |
| – | – | – | – | LSJA | $O(n)$ |

**Table 4**
Set of parameters.

| Parameter | Description | Tested values | Final |
|---|---|---|---|
| $\alpha_1$ | RR procedure probability | [0, 1] | 0.236 |
| $\alpha_2$ | NN procedure probability | [0, 1] | 0.112 |
| $\alpha_3$ | TSPS procedure probability | [0, 1] | 0.652 |
| $\alpha_4$ | RAP procedure probability | [0, 1] | 0.497 |
| $\alpha_5$ | NNJA procedure probability | [0, 1] | 0.503 |
| $\beta_1$ | OX crossover operator probability | [0, 1] | 0.429 |
| $\beta_2$ | PMX crossover operator probability | [0, 1] | 0.363 |
| $\beta_3$ | UMPX crossover operator probability | [0, 1] | 0.208 |
| $\delta_1$ | Probability of executing route mutation operators | [0, 1] | 0.121 |
| $\delta_2$ | Probability of executing assignment mutation operators | [0, 1] | 0.883 |
| $\delta_3$ | VEM mutation operator probability | [0, 1] | 0.339 |
| $\delta_4$ | RM mutation operator probability | [0, 1] | 0.253 |
| $\delta_5$ | SM mutation operator probability | [0, 1] | 0.395 |
| $\delta_6$ | 2-opt mutation operator probability | [0, 1] | 0.013 |
| $\delta_7$ | LSJA mutation operator probability | [0, 1] | 0.914 |
| $\delta_8$ | JEM mutation operator probability | [0, 1] | 0.086 |
| $N^t$ | Population size | {25, 50, 75, 100} | 100 |
| $N^g$ | Number of generations | 500 | 500 |
| $N^t$ | Tournament size | {2, 3, 4, 5, 6} | 2 |
| $\rho^c$ | Crossover probability | [0, 1] | 0.327 |
| $\rho^m$ | Mutation probability | [0, 1] | 0.717 |
| $\rho^e$ | Elitism probability | [0.01, 0.2] | 0.147 |

to achieve good computational performance. This is especially true for the crossover and mutation probabilities, which must be carefully balanced to produce synergistic effects and obtain good-quality solutions (Contreras-Bolton & Parada, 2015). In this study, we employed an effective method for tuning these parameters, namely irace, which is an automatic configuration tool implemented in R (López-Ibáñez et al., 2016). To tune the parameters, we selected a subset of instances based on preliminary results from the MGA, which included the five worst-performing instances for the tsplib (gr24, bays29, gr48, eil51, and eil76), five instances for the small set (34, 77, 81, 97, and 100), five instances for the medium set (1, 14, 18, 29, and 58), and five instances for the large set (1, 9, 14, 47, and 78). Then, during each irace call to an instance, we set two stop conditions: a time limit of 1800 s and a fixed number of generations ($N^g$) at 500. If either condition is exceeded, MGA is terminated. This is because we noticed, based on preliminary experiments, that irace tended to select the highest possible value for this parameter, as larger values led to better results. However, our aim was also to have short computing times. Finally, the irace tuning process took approximately 49.24 h using 40 threads on the supercomputing mentioned above. The parameter names, descriptions, ranges where values were tested and values obtained through irace tuning are listed in Table 4.

### 5.3. MILP results

The first experiment aims to compare the performance of four different subtour elimination constraint formulations for the MILP model. Specifically, we compare the performance of the basic model (BM) without additional subtour elimination constraints (constraints (1)–(12)), and the same model with three different subtour elimination constraints: the MTZ formulation (constraints (13)–(14)), the DL formulation (constraints (15)–(18)), and the GG formulation (constraints (19)–(20)). Additionally, all the models start from a feasible solution to obtain the largest number of upper and lower bounds for all instances. This is because it can be difficult for the models to find such solutions, particularly for medium and large instances. Therefore, the feasible solution is built using the LKH algorithm and NNJA procedure.

Table 5 presents the performance of the mathematical models on the benchmarking instances. The first column shows the name of the instance set, while the second column shows the best known result (BKS). The BKS cost is calculated as the minimum cost obtained among all the reported models and algorithms with each instance. The following four groups of columns correspond to the four mathematical models evaluated, and provide information on the upper and lower bounds found, the optimality percentage gap (Laporte & Toth, 2022) (i.e., the percentage gap between the best upper bound and the best lower bound found at the end of the solving process), and the corresponding computing time (expressed in seconds). The tsplib, small, and medium instances were executed with a time limit of one hour per instance. In contrast, the large instances were allowed a two-hour time limit due to their complexity since the lower bounds and feasible solutions were not obtained in the first hour of running for some of the large instances.

During the initial execution of the four models using the tsplib and small instances, GG emerged as the most efficient model in terms of computing time and gap. DL, BM, and MTZ followed in order of their computing times, as they produced the same number of optimal solutions. Out of the 110 instances corresponding to the tsplib and small instances, GG was the only model that could solve all the instances optimally. Contrary to expectations, GG performed poorly when applied to the medium and large instances. Notably, BM demonstrated the best performance within the medium instances set (although DL came close), while DL was highlighted in the large instances. However, none of the models achieved optimal solutions for the medium and large instances. In summary, the DL-based formulation emerged as the most effective model overall. Nonetheless, it is important to note that the proposed models can only handle small instances. Therefore, using

**Table 5**
Results for the all formulations.

| Instance | BKS | BM | | | | MTZ | | | | DL | | | | GG | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | UB | LB | Gap | Time | UB | LB | Gap | Time | UB | LB | Gap | Time | UB | LB | Gap | Time |
| Tsplib | 3 729.79 | 3 729.79 | 3 729.79 | 0.00 | 68.21 | 3 729.79 | 3 729.79 | 0.00 | 74.83 | 3 729.79 | 3 729.79 | 0.00 | 49.21 | 3 729.79 | 3 729.79 | 0.00 | 14.52 |
| Small | 279.23 | 279.23 | 279.21 | 0.01 | 66.31 | 279.23 | 279.20 | 0.01 | 94.04 | 279.23 | 279.20 | 0.01 | 57.64 | 279.23 | 279.23 | 0.00 | 4.25 |
| Medium | 3 394.18 | 3 394.43 | 3 283.61 | 3.37 | 3600.55 | 3 394.54 | 3 254.81 | 4.30 | 3602.96 | 3 394.46 | 3 281.91 | 3.43 | 3600.38 | 3 394.94 | 2 865.70 | 19.20 | 3600.08 |
| Large | 13 297.97 | 13 298.32 | 11 620.81 | 19.30 | 7202.94 | 13 298.32 | 9 172.35 | 71.02 | 7204.93 | 13 297.98 | 11 686.48 | 14.16 | 7203.04 | 13 298.32 | 10 665.44 | 24.69 | 7201.20 |
| Overall | 5 594.95 | 5 595.15 | 5 018.26 | 7.32 | 3508.59 | 5 595.18 | 4 219.14 | 24.30 | 3519.17 | 5 595.05 | 5 038.89 | 5.67 | 3505.16 | 5 595.31 | 4 575.27 | 14.16 | 3486.12 |

**Table 6**
Performance of MGA against MILP models.

| Instance | BKS | MGA | | | | | MILP-DL-Large | | | | MILP-DL-Short | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Best | Avg | Best gap | Avg gap | Time | Best | Gap | Best gap | Time | Best | Gap | Best gap | Time |
| Tsplib | 3 729.79 | 3 790.16 | 3 934.86 | 1.63 | 4.77 | 8.88 | 3 729.79 | 0.00 | 0.00 | 49.21 | 3 767.68 | 3.74 | 3.39 | 9.50 |
| Small | 279.23 | 285.39 | 285.58 | 2.21 | 2.27 | 10.00 | 279.23 | 0.01 | 0.00 | 57.64 | 280.13 | 0.81 | 0.31 | 11.62 |
| Medium | 3 394.18 | 3 394.91 | 3 395.22 | 0.02 | 0.03 | 95.41 | 3 394.46 | 3.43 | 0.01 | 3600.38 | 3 549.36 | 17.93* | 4.51 | 154.99 |
| Large | 13 297.97 | 13 298.31 | 13 299.37 | 0.00 | 0.01 | 332.23 | 13 297.98 | 14.16 | 0.00 | 7203.04 | 16 125.71 | – | 21.27 | 400.00 |
| Overall | 5 594.95 | 5 599.23 | 5 604.40 | 0.77 | 0.90 | 141.46 | 5 595.05 | 5.67 | 0.00 | 3505.16 | 6 558.70 | 7.02* | 8.53 | 183.08 |

a heuristic approach to solve this problem efficiently appears to be the more practical choice.

### 5.4. MGA performance

To assess the MGA's performance, we first evaluated it against the most effective model overall, DL-based formulation. It is important to note that the DL-based formulation (here referred to as MILP-DL-Large) was executed by a one-hour time limit per instance for the tsplib, small, and medium instance sets and with a two-hour time limit for the large instances. Conversely, the MGA runs ten separate executions on each instance, each with different randomization seeds. Thus, to ensure a fair and equitable comparison, we reran the DL-based formulation (referred to as MILP-DL-Short) and adjusted its time limits based on the respective instance set sizes. These modified time limits were set at 20 s for tsplib and small instances, 150 s for medium instances, and 400 s for large instances.

The results of these experiments are presented in Table 6. The first column shows the name of the instance set, while the second column shows BKS. The next group of columns corresponds to the following five columns display the MGA results: best, which is the best result obtained in the ten runs; avg, which corresponds to the average obtained in the ten executions; the best and avg percentage gap (i.e., the percentage gap between the (average or best) cost and the BKS); and the corresponding computing time (expressed in seconds). Finally, the subsequent two sets of columns correspond to MILP-DL-Large and MILP-DL-Short, providing four columns each: upper bounds found (best result); optimality percentage gap, best gap (the percentage gap between the UB and BKS), and the corresponding computing time (expressed in seconds).

When considering the best gap metric, we observe that MILP-DL-Large consistently achieves the best results across all instance sets. In contrast, MGA performs almost the same in medium instances and slightly better in large instances (detected in the best cost). However, MILP-DL-Large needs extensive computing time to achieve this level of performance. Therefore, a fair comparison is to set a time limit for the MILP model that closely aligns with the computing time required by MGA or slightly surpasses it. Under these adjusted time limits, the performance of MILP-DL-Short experiences a considerable decline, causing MGA to outperform it in the tsplib, medium, and large instance sets. Note that the medium instance set is marked with an asterisk (*), indicating that the average optimality percentage gap for MILP-DL-Short was computed for certain instances, specifically, 60 out of 100 instances, as it could not find lower bounds within the given time limit. In the case of large instances, instances marked with (–) indicate that MILP-DL-Short could not compute any instances within the given time limit. Finally, the overall row is affected for both cases, considering only 170 out of 310 instances.

Secondly, we evaluate the performance of the MGA against the four heuristics (LSH-I, LSH-II, LSH-III, and LSH-IV) presented in Mosayebi et al. (2021) and also consider MILP-DL-Short in the comparison. To ensure a fair comparison, we ran all heuristics on the same abovementioned computer. The MGA was executed ten times on each instance using different seeds, while the four heuristics were only run once since they are deterministic heuristics.

The results of the MGA, MILP-DL-Short and the four heuristics are summarized in Table 7. The first column lists the set of instances, and the following five columns display the MGA results: best, which is the best result obtained in the ten runs; avg, which corresponds to the average obtained in the ten executions; the best and avg percentage gap (i.e., the percentage gap between the (average or best) cost and the BKS); and the corresponding computing time (expressed in seconds). The subsequent three columns show the results of MILP-DL-Short: best, gap (same as above), and computing time (expressed in seconds). This format is then repeated for the columns of LSH-I, LSH-II, LSH-III, and LSH-IV algorithms. Finally, the last row shows the averages of each of the abovementioned columns. It is worth noting that the tsplib set contains ten instances while the other instances contain 100 each.

MGA outperforms MILP-DL-Short, LSH-I, LSH-II, LSH-III, and LSH-IV. In the first set of instances, MGA achieves a lower percentage best gap than all four heuristics when comparing based on the best gap. Specifically, MGA achieves the best gap of 1.63%, while the best performing heuristic, LSH-III, achieves 2.61%. Regarding avg gap, MGA is superior to three of the four heuristics. However, MGA has a higher computing time than the four heuristics. Similar results are observed for the small instances set, where MGA performs better than all four heuristics in the best and avg gap metrics but not in computing time. Specifically, MGA obtains 2.21% and 2.27% in the best and avg gap, respectively. Again the best performing heuristic is LSH-III achieves 3.61%. However, MILP-DL-Short surpassed MGA with the best gap of 0.31%, making it the only set of instances where this occurred. MGA performs similarly to the previous set in the medium instances set, but the best performing heuristics are LSH-III and LSH-IV. Note the MGA obtains the best gap of 0.00% and an avg gap very close to 0.01%. MGA outperforms all four heuristics and MILP-DL-Short for the large instances set in both the best gap and avg gap metrics. Moreover, it

**Table 7**
Performance of MGA against MILP-DL-Short and four heuristics.

| Instance | MGA | | | | | MILP-DL-Short | | | LSH-I | | | LSH-II | | | LSH-III | | | LSH-IV | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Best | Avg | Best gap | Avg gap | Time | Best | Best gap | Time | Best | Best gap | Time | Best | Best gap | Time | Best | Best gap | Time | Best | Best gap | Time |
| Tsplib | 3 790.16 | 3 934.86 | 1.63 | 4.77 | 8.88 | 3 767.68 | 3.39 | 9.50 | 3 823.32 | 2.74 | 0.25 | 3 955.35 | 5.68 | 0.32 | 3 819.61 | 2.61 | 0.38 | 3 838.23 | 2.77 | 0.46 |
| Small | 285.39 | 285.58 | 2.21 | 2.27 | 10.00 | 280.13 | 0.31 | 11.62 | 289.93 | 3.81 | 0.22 | 295.00 | 5.63 | 0.21 | 289.34 | 3.61 | 0.35 | 289.39 | 3.63 | 0.34 |
| Medium | 3 394.91 | 3 395.22 | 0.00 | 0.01 | 95.41 | 3 549.36 | 4.51 | 154.99 | 3 551.98 | 4.62 | 22.91 | 3 557.01 | 4.77 | 25.13 | 3 544.75 | 4.41 | 45.70 | 3 544.88 | 4.41 | 40.74 |
| Large | 13 298.31 | 13 299.37 | 0.00 | 0.01 | 332.23 | 16 125.71 | 21.27 | 400.00 | 13 908.57 | 4.59 | 214.48 | 13 928.34 | 4.74 | 228.20 | 13 892.40 | 4.47 | 424.67 | 13 882.74 | 4.39 | 375.47 |
| Overall | 5 599.23 | 5 604.40 | 0.77 | 0.89 | 141.46 | 6 558.70 | 8.53 | 183.08 | 5 849.29 | 4.29 | 76.66 | 5 863.19 | 5.07 | 81.80 | 5 841.44 | 4.11 | 151.86 | 5 838.98 | 4.10 | 134.39 |

**Table 8**
The statistical significance levels of $p$-values obtained.

| Instance | $p$-value | | | | |
|---|---|---|---|---|---|
| | MILP-DL-Short | LSH-I | LSH-II | LSH-III | LSH-IV |
| Tsplib | $2.07 \times 10^{-1}$ | $6.63 \times 10^{-2}$ | $3.91 \times 10^{-3}$ | $1.10 \times 10^{-1}$ | $3.82 \times 10^{-2}$ |
| Small | $2.60 \times 10^{-17}$ | $6.57 \times 10^{-8}$ | $6.32 \times 10^{-17}$ | $6.64 \times 10^{-7}$ | $2.86 \times 10^{-7}$ |
| Medium | $5.35 \times 10^{-7}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ |
| Large | $3.89 \times 10^{-18}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ | $3.90 \times 10^{-18}$ |

has better computing time than the best performing heuristic, LSH-IV. Finally, it is worth noting that the best gap metric for large instances is 0.00%, and the avg gap is very close to 0.01%. These results suggest that MGA is an efficient algorithm for solving these instances, especially for large instances where it widely outperforms all four heuristics and MILP-DL-Short.

Overall, MGA outperforms the four heuristics and MILP-DL-Short when considering both the best and average gap metrics. Specifically, MGA achieves the best gap of 0.77% and an average gap of 0.89%. In comparison, the best performing heuristic is LSH-IV, which has a gap of 4.10%, closely followed by LSH-III with a gap of 4.11%. In terms of computing times, MGA only is faster than LSH-III. While MGA requires more computing time in small instances set, this is compensated by the fact that it has approximately the same processing time as LSH-III in large instances. It is worth noting that the computing time of MGA is expected to be greater than LSH-III. Since MGA is a population-based metaheuristic and requires highly time-consuming to work with its population of solutions in each iteration. In contrast, LSH-III is only a heuristic. However, MGA has a slightly shorter time than LSH-III. These results suggest that MGA is an efficient algorithm in terms of computing time required to find good-quality solutions, especially for the medium and large instances sets where it outperforms all four heuristics and MILP-DL-Short.

To determine whether the differences in the results obtained are statistically significant, we used a nonparametric test for pairwise comparison procedures, Wilcoxon's signed-ranks test. This test allows us to compare the means of two populations by testing the null hypothesis that they are equal against the alternative hypothesis that they are not (Carrasco et al., 2020; Derrac et al., 2011; García et al., 2009). We use the results of the best gap obtained from the algorithms for applying Wilcoxon test. Table 8 presents the $p$-values for the test, with the first column representing the set of instances, and the following four columns representing the $p$-values for the comparison of MGA with MILP-DL-Short and the corresponding heuristics (LSH-I, LSH-II, LSH-III, and LSH-VI). For example, the second column shows the $p$-value of the comparison between MGA and MILP-DL-Short for each set of instances, and so on. The results of the test indicate that MGA has statistically significant differences with each heuristic in the small, medium, and large instances at a level of significance of 5%. However, for tsplib, the null hypothesis is only validated for LSH-II, and for small, there is an exception with MILP-DL-Short since the significant differences are favor of it. This suggests that MGA performs significantly better than the four heuristics in most cases, further supporting the conclusion that MGA is an effective algorithm for solving these instances.

### 5.5. Importance of the multioperators and an effective parameter tuning

Our previous experiments revealed that combining multioperators in GA with effective parameter tuning leads to a positive synergy, which produces a trade-off between high-quality solutions and computing times. To further investigate this relationship, we conducted additional experiments. We compared our MGA against eight versions of less-tuned MGA, four of which use single operators, while the other four use multioperators but with percentages of the parameters proportional. For clarity, we define a single operator as a GA that employs only one crossover operator for both chromosomes and a single mutation operator for both chromosomes.

To ensure a fair comparison, we conducted the additional experiments by running all algorithms ten times, independently, with different random seeds for each instance and using given time limits for each set of instances. Each iteration has a different computational effort for each variant; thereby, the previous terminating condition based on the number of generations ($N^g$) is now changed for a given time limit. Therefore, the sets of time limits were determined based on the sizes of the sets of instances. We established time limits of 20, 20, 150, and 400 s for the tsplib, small, medium, and large sets, respectively. The results of the experiments are shown in Table 9, where the first column describes the eight versions of less-tuned MGA and MGA (fully-tuned) in the last row. The second through sixth columns show the best gap, average gap, average number of iterations achieved in each instance (max iterations), average of the iterations in which the best result is found (best iterations), and $p$-value of the Wilcoxon test. Below are the details of the eight versions of the algorithms used in this section. MGA1, MGA2, MGA3, and MGA4 correspond to the use of single operators, while MGA5, MGA6, MGA7, and MGA8 use multioperators. Note that the remaining parameters not mentioned for all eight versions are the same as those used in MGA.

- MGA1 uses the OX crossover operator and SM mutation operator for both chromosomes. The probability of mutation is the same for both chromosomes, with a value of $\delta_1 = \delta_2 = 0.5$.
- MGA2 uses the PMX crossover operator and SM mutation operator for both chromosomes, with the same probability of mutation as MGA1.
- MGA3 uses the OX crossover operator and LSJA mutation operator for both chromosomes, with the same probability of mutation as MGA1 and MGA2.
- MGA4 uses the OX crossover operator, and the mutation operators are the same as the ones used in MGA for both chromosomes. But all operators have the same probability of occurrence, with a value of $\delta_1 = \delta_2 = 0.5$.

**Table 9**
Results of the fully-tuned MGA versus versions of less-tuned MGA.

| Algorithm | Best gap | Avg gap | Max iterations | Best iterations | $p$-value |
|---|---|---|---|---|---|
| MGA1 | 0.82 | 1.12 | 1229.86 | 459.39 | $3.14 \times 10^{-12}$ |
| MGA2 | 0.83 | 1.10 | 1307.30 | 497.80 | $4.38 \times 10^{-16}$ |
| MGA3 | 0.80 | 0.88 | 891.83 | 176.21 | $9.81 \times 10^{-4}$ |
| MGA4 | 3.09 | 3.48 | 526.54 | 110.49 | $1.28 \times 10^{-19}$ |
| MGA5 | 0.78 | 0.98 | 1055.84 | 298.98 | $1.79 \times 10^{-4}$ |
| MGA6 | 8.88 | 9.61 | 148.77 | 15.19 | $3.32 \times 10^{-36}$ |
| MGA7 | 2.99 | 3.54 | 517.16 | 111.36 | $2.90 \times 10^{-19}$ |
| MGA8 | 2.97 | 3.39 | 554.54 | 112.56 | $1.88 \times 10^{-19}$ |
| MGA | 0.76 | 0.88 | 867.07 | 199.84 | – |

- MGA5 uses the same crossover operators as MGA (OX, PMX, and UPMX) but with equal probability of occurrence. While mutation operator, the SM operator is used for the route chromosome and LSJA is used for the jobs assignment. Both chromosomes have the same probability of mutation, with a value of $\delta_1 = \delta_2 = 0.5$.
- MGA6 uses the same crossover operators as MGA (OX, PMX, and UPMX) but with equal probability of occurrence. While mutation operator, the 2-opt mutation operator used for the route chromosome and LSJA used for the jobs assignment. The mutation probability is the same for both chromosomes, with a value of $\delta_1 = \delta_2 = 0.5$.
- MGA7 uses the same crossover operators as MGA (OX, PMX, and UPMX) but with an equal probability of occurrence. While mutation operator, for both chromosomes is the same as the ones used in MGA, but all operators have the same probability of occurrence, also including $\delta_1$ and $\delta_2$.
- MGA8 uses the OX and PMX crossover operators with equal probability of occurrence. While mutation operators are the same as those used in MGA for both chromosomes. The mutation probability of all operators has the same probability of occurrence. In addition, $\delta_1 = \delta_2 = 0.5$.

The findings of the additional experiments highlight the critical importance of tuning the crossover and mutation operators in MGA. MGA exhibits superior performance and operator balance compared to the eight less-tuned MGA variants. Notably, the single operator versions outperform MGA in terms of the number of iterations but suffer from premature convergence and worse gap metrics. Note that MGA3 is the best version of the single operators and gets the same average gap as MGA but worse in the best gap. This is due to the use of a mutation operator based on LSJA.

Among the multioperator versions, MGA5 stands out, achieving similar results to MGA in the best gap metric while offering higher average gap values. This success is attributed to MGA5's alignment with MGA's operator proportions, particularly its use of LSJA in 100% of job assignments, enabling more iterations and better solutions. Conversely, MGA6, MGA7, and MGA8 have higher computational complexity due to imbalanced operator usage. While MGA6 favors the 2-opt operator, leading to suboptimal job assignment. Moreover, MGA7 and MGA8 assign equal probabilities to operators, resulting in a lack of balance between solution quality and the number of iterations. This highlights that effective tuning of crossover and mutation operators is imperative, as removing this tuning generates worse solutions. Finally, it is worth emphasizing that the two best versions of less-tuned MGA consistently exhibit a high participation rate of the operator introduced in this work, LSJA.

Wilcoxon's signed-ranks test is adopted to show the difference in the mean solutions is statistically significant for all experiments, using a significance level of 5%. Notably, MGA3 and MGA5 exhibit the closest performance in terms of solution quality. Overall, these results suggest that synergy among multioperators can be further enhanced with effective parameter tuning, such as the one employed in MGA, to achieve a good trade-off between solution quality and the number of iterations.

The above behavior can be verified in Fig. 3(a) illustrates the convergence curves of MGA and all its variants for instance 58 in the large set with a random seed value of 0. Each curve in the plot represents the best individual of each generation but is reported in seconds. It can be observed that MGA4, MGA6, MGA7 and MG8 only manage to complete 113, 9, 127 and 78 generations, respectively, within the given time limit of 400 s due to their high computational complexity. On the other hand, MGA1 and MGA2 complete a larger number of generations (616 and 757, respectively, within 316.34 s and 354 s) and produce better results than MGA4, MGA6, MGA7 and MG8, particularly MGA1. However, they seem to require more generations to converge. MGA5 converges to a local optimum in generation 529 within 269.89 s. Meanwhile, both MGA3 and MGA converge to the best known solution, but MGA achieves faster convergence within 206.39 s compared to MGA3, which converges within 257.19 s. Fig. 3(b) shows a zoomed-in view of MGA, MGA1, MGA2, MGA3, and MGA5, in order to better visualize the details of their convergences.

## 6. Conclusion

This work proposed a multioperator genetic algorithm that employs various initial population procedures, crossover, and mutation operators to address the TSPJ. Our algorithm used five procedures to generate the initial population, and three crossover operators, which are classical operators for the permutation. In addition, six different mutation operators, where four operators focused on diversification, and two were designed to aid intensification. The results of the computational experiments show that the MGA outperforms the state-of-the-art algorithms on four sets of instances, with 310 instances in total ranging in size from 17 to 1200 vertices. Our results demonstrate that the engine behind the generation of high-quality solutions is the use of multioperators with effective parameter tuning, which creates a synergistic effect that balances solution quality and computing time. This is exemplified by the superior performance of MGA, which outperforms GAs using both single operators and multioperators with traditional proportions.

Several aspects could be explored in future research. Firstly, it is still challenging to efficiently address medium and large instances, so a new formulation model could be modeled and solved by an exact algorithm taking advantage of the new formulation. Secondly, our proposed approach could provide a warm start to improve the performance of an exact algorithm. Thirdly, a hybridization with a single-solution-based metaheuristic approach that operates in the search space of the route or job assignment. Lastly, to extend the proposed algorithm to solve recent variants of the TSP.

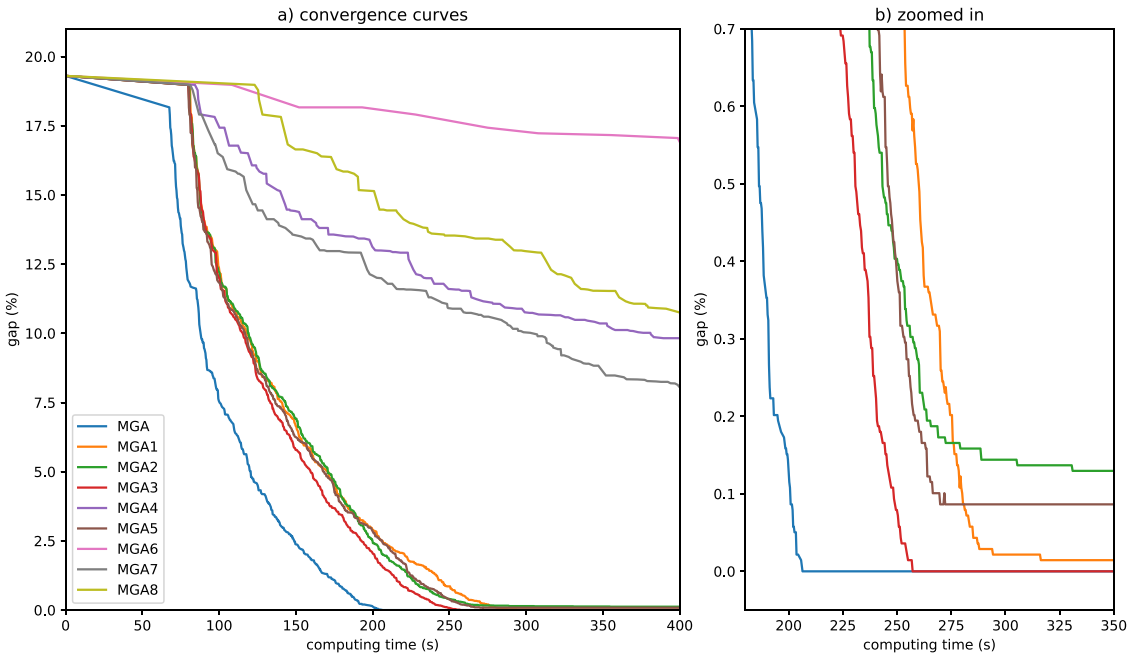



**Fig. 3.** Convergence curves of all algorithms in the instance 58 in the large set.

## CRediT authorship contribution statement

**Pablo Gutiérrez-Aguirre:** Conceptualization, Methodology, Software, Validation, Investigation, Formal analysis, Data curation, Writing – original draft, Writing – review & editing. **Carlos Contreras-Bolton:** Conceptualization, Methodology, Validation, Investigation, Formal analysis, Data curation, Writing – original draft, Writing – review & editing, Supervision.

## Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Carlos Contreras-Bolton reports a relationship with University of Concepción that includes: funding grants.

## Data availability

We have shared the link to our data/code in the manuscript.

## Acknowledgments

This research was partially supported by the supercomputing infrastructure of the NLHPC (ECM-02). The authors thank the anonymous reviewers for their invaluable comments and suggestions, which have significantly enhanced our work.

## References

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2007). *The traveling salesman problem: A computational study*. Princeton, NJ, USA: Princeton University Press.

Asadujjaman, M., Rahman, H. F., Chakrabortty, R. K., & Ryan, M. J. (2022). Multi-operator immune genetic algorithm for project scheduling with discounted cash flows. *Expert Systems with Applications*, *195*, Article 116589.

Banzhaf, W. (1990). The "molecular" traveling salesman. *Biological Cybernetics*, *64*(1), 7–14.

Berghman, L., Kergosien, Y., & Billaut, J.-C. (2023). A review on integrated scheduling and outbound vehicle routing problems. *European Journal of Operational Research*, *311*(1), 1–23.

Bianco, L., Mingozzi, A., & Ricciardelli, S. (1993). The traveling salesman problem with cumulative costs. *Networks*, *23*(2), 81–91.

Bigras, L.-P., Gamache, M., & Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization*, *5*(4), 685–699.

Blum, A., Chalasani, P., Coppersmith, D., Pulleyblank, B., Raghavan, P., & Sudan, M. (1994). The minimum latency problem. In *Proceedings of the twenty-sixth annual ACM symposium on theory of computing, STOC '94* (pp. 163—171). New York, NY, USA: Association for Computing Machinery.

Bodaghi, B., Shahparvari, S., Fadaki, M., Lau, K. H., Ekambaram, P., & Chhetri, P. (2020). Multi-resource scheduling and routing for emergency recovery operations. *International Journal of Disaster Risk Reduction*, *50*, Article 101780.

Bourreau, E., Garaix, T., Gondran, M., Lacomme, P., & Tchernev, N. (2022). A constraint-programming based decomposition method for the Generalised Workforce Scheduling and Routing Problem (GWSRP). *International Journal of Production Research*, *60*(4), 1265–1283.

Cacchiani, V., Contreras-Bolton, C., Escobar-Falcón, L. M., & Toth, P. (2023). A matheuristic algorithm for the pollution and energy minimization traveling salesman problems. *International Transactions in Operational Research*, *30*(2), 655–687.

Cacchiani, V., Contreras-Bolton, C., & Toth, P. (2020). Models and algorithms for the traveling salesman problem with time-dependent service times. *European Journal of Operational Research*, *283*(3), 825–843.

Carrasco, J., García, S., Rueda, M., Das, S., & Herrera, F. (2020). Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm and Evolutionary Computation*, *54*, Article 100665.

Chen, Y., Lan, H., Wang, C., & Jia, X. (2022). An integrated distribution scheduling and route planning of food cold chain with demand surge. *Complex & Intelligent Systems*, *9*, 475—491.

Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In D.-Z. Du, & P. M. Pardalos (Eds.), *Handbook of combinatorial optimization: Volume1–3* (pp. 1493–1641). Boston, MA: Springer US.

Chevroton, H., Kergosien, Y., Berghman, L., & Billaut, J.-C. (2021). Solving an integrated scheduling and routing problem with inventory, routing and penalty costs. *European Journal of Operational Research*, *294*(2), 571–589.

Cicirello, V. A., & Smith, S. F. (2000). Modeling GA performance for control parameter optimization. In *Proceedings of the 2nd annual conference on genetic and evolutionary computation, GECCO '00* (pp. 235—242). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Contreras-Bolton, C., Gatica, G., Barra, C. R., & Parada, V. (2016). A multi-operator genetic algorithm for the generalized minimum spanning tree problem. *Expert Systems with Applications*, *50*, 1–8.

Contreras-Bolton, C., & Parada, V. (2015). Automatic combination of operators in a genetic algorithm to solve the traveling salesman problem. *PLOS ONE*, *10*(9), 1–25.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations Research*, *6*(6), 791–812.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, *2*(4), 393–410.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th international joint conference on artificial intelligence - Volume 1, IJCAI '85* (pp. 162–164). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, *1*(1), 3–18.

Desrochers, M., & Laporte, G. (1991). Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints. *Operations Research Letters*, *10*(1), 27–36.

Di Mascolo, M., Martinez, C., & Espinouse, M.-L. (2021). Routing and scheduling in Home Health Care: A literature survey and bibliometric analysis. *Computers & Industrial Engineering*, *158*, Article 107255.

Durasević, M., & Jakobović, D. (2023). Heuristic and metaheuristic methods for the parallel unrelated machines scheduling problem: a survey. *Artificial Intelligence Review*, *56*(4), 3181–3289.

Eiben, A., & Smith, J. (2015). *Introduction to evolutionary computing* (2nd ed.). Berlin, Heidelberg: Springer Berlin Heidelberg.

Elsayed, S. M., Sarker, R. A., & Essam, D. L. (2011). Multi-operator based evolutionary algorithms for solving constrained optimization problems. *Computers & Operations Research*, *38*(12), 1877–1896.

Fikar, C., & Hirsch, P. (2017). Home health care routing and scheduling: A review. *Computers & Operations Research*, *77*, 86–95.

Fischetti, M., Laporte, G., & Martello, S. (1993). The delivery man problem and cumulative matroids. *Operations Research*, *41*(6), 1055–1064.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, *4*(1), 61–75.

Fogel, D. B., & Atmar, J. (1990). Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, *63*, 111–114.

Fox, K. R., Gavish, B., & Graves, S. C. (1980). Technical note—An n-constraint formulation of the (time-dependent) traveling salesman problem. *Operations Research*, *28*(4), 1018–1021.

Fu, L.-L., Aloulou, M. A., & Triki, C. (2017). Integrated production scheduling and vehicle routing problem with job splitting and delivery time windows. *International Journal of Production Research*, *55*(20), 5942–5957.

García, S., Molina, D., Lozano, M., & Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special Session on Real Parameter Optimization. *Journal of Heuristic*, *15*(6), 617–644.

Gavish, B., & Graves, S. C. (1978). *The travelling salesman problem and related problems*: *Technical report*, Massachusetts Institute of Technology, Operations Research Center.

Goldberg, D. E., & Lingle, R. (1985). AllelesLociand the traveling salesman problem. In *Proceedings of the 1st international conference on genetic algorithms* (pp. 154–159). USA: L. Erlbaum Associates Inc..

Gutin, G., & Punnen, A. (2007). *Combinatorial optimization*: *vol. 12*, *The traveling salesman problem and its variations* (p. 749). Boston, MA: Springer US.

He, P., & Hao, J.-K. (2023). Memetic search for the minmax multiple traveling salesman problem with single and multiple depots. *European Journal of Operational Research*, *307*(3), 1055–1070.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106–130.

Helsgaun, K. (2017). *An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems*: *Technical report*, (pp. 24–50). Roskilde: Roskilde University.

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), *Complexity of computer computations. the IBM research symposia series* (pp. 85–103). Boston, MA: Springer US.

Lam, C., & Ip, W. (2019). An integrated logistics routing and scheduling network model with RFID-GPS data for supply chain management. *Wireless Personal Communications*, *105*, 803–817.

Laporte, G., & Toth, P. (2022). A gap in scientific reporting. *4OR*, *20*, 169–171.

Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H., & Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. In *Logistics of production and inventory* (pp. 445–522). North-Holland, Amsterdam: Elsevier.

Liang, Y., Wang, X., Luo, Z., & Zhang, D. (2023). Integrated optimisation of loading schedules and delivery routes. *International Journal of Production Research*, *61*(16), 5354–5371.

Liu, J., Anavatti, S., Garratt, M., & Abbass, H. A. (2022). Multi-operator continuous ant colony optimisation for real world problems. *Swarm and Evolutionary Computation*, *69*, Article 100984.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., & Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, *3*, 43–58.

Lyu, X., Song, Y., He, C., Lei, Q., & Guo, W. (2019). Approach to integrated scheduling problems considering optimal number of automated guided vehicles and conflict-free routing in flexible manufacturing systems. *IEEE Access*, *7*, 74909–74924.

Mardones, B., Gatica, G., & Contreras-Bolton, C. (2023). A metaheuristic for the double traveling salesman problem with partial last-in-first-out loading constraints. *International Transactions in Operational Research*, *30*(6), 3904–3929.

Martinelli, R., Mariano, F. C. M. Q., & Martins, C. B. (2022). Single machine scheduling in make to order environments: A systematic review. *Computers & Industrial Engineering*, *169*, Article 108190.

Maya-Duque, P. A., Dolinskaya, I. S., & Sörensen, K. (2016). Network repair crew scheduling and routing for emergency relief distribution problem. *European Journal of Operational Research*, *248*(1), 272–285.

Miller, C., Zemlin, R., & Tucker, A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, *7*(4), 326–329.

Moreno, A., Munari, P., & Alem, D. (2019). A branch-and-Benders-cut algorithm for the Crew Scheduling and Routing Problem in road restoration. *European Journal of Operational Research*, *275*(1), 16–34.

Morton, G., & Land, A. H. (1955). A contribution to the 'travelling-salesman' problem. *Journal of the Royal Statistical Society. Series B.*, *17*(2), 185–194.

Mosayebi, M. (2020). *The traveling salesman problem with job-time, multi-circuit, and variations* (Ph.D. thesis), USA: University of Rhode Island.

Mosayebi, M., Sodhi, M., & Wettergren, T. A. (2021). The Traveling Salesman Problem with Job-times (TSPJ). *Computers & Operations Research*, *129*, Article 105226.

Moser, M., Musliu, N., Schaerf, A., & Winter, F. (2022). Exact and metaheuristic approaches for unrelated parallel machine scheduling. *Journal of Scheduling*, *25*, 507–534.

Özdamar, L., & Ertem, M. A. (2015). Models, solutions and enabling technologies in humanitarian logistics. *European Journal of Operational Research*, *244*(1), 55–65.

Picard, J.-C., & Queyranne, M. (1978). The time-dependent traveling salesman problem and its application to the tardiness problem in one-machine scheduling. *Operations Research*, *26*(1), 86–110.

Pinedo, M. L. (2016). *Scheduling: Theory, algorithms, and systems* (5th ed.). Springer International Publishing.

Pralet, C. (2023). Iterated maximum large neighborhood search for the traveling salesman problem with time windows and its time-dependent version. *Computers & Operations Research*, *150*, Article 106078.

Reinelt, G. (1991). TSPLIB—A traveling salesman problem library. *ORSA Journal on Computing*, *3*(4), 376–384.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation* (p. 618). Hoboken, NJ: JohnWiley & Sons, Inc..

Tsitsiklis, J. N. (1992). Special cases of traveling salesman and repairman problems with time windows. *Networks*, *22*(3), 263–282.

Will, T. (1993). *Extremal results and algorithms for degree sequences of graphs* (Ph.D. thesis), Urbana-Champaign, USA: University of Illinois.

Yadav, N., & Tanksale, A. (2022). An integrated routing and scheduling problem for home healthcare delivery with limited person-to-person contact. *European Journal of Operational Research*, *303*(3), 1100–1125.

Yu, X., Shen, S., & Wang, H. (2021). Integrated vehicle routing and service scheduling under time and cancellation uncertainties with application in nonemergency medical transportation. *Service Science*, *13*(3), 172–191.

Zou, X., Liu, L., Li, K., & Li, W. (2018). A coordinated algorithm for integrated production scheduling and vehicle routing problem. *International Journal of Production Research*, *56*(15), 5005–5024.