mathematics

MDPI

Article

Balancing the Average Weighted Completion Times in Large-Scale Two-Agent Scheduling Problems: An Evolutionary-Type Computational Study

Matteo Avolio

Department of Mathematics and Computer Science, University of Calabria, 87036 Rende, Italy; matteo.avolio@unical.it

Abstract: The problem of balancing the average weighted completion times of two classes of jobs is an NP-hard scheduling problem that was very recently introduced in the literature. Interpreted as a cooperative-type two-agent single-machine problem, its applications are in various practical contexts such as in logistics for balancing the delivery times, in manufacturing for balancing the assembly lines and in services for balancing the waiting times of groups of people. The only solution technique currently existing in the literature is a Lagrangian heuristic, based on solving a finite number of successive linear assignment problems, whose dimension depends on the total number of jobs. Since the Lagrangian approach has not appeared to be particularly suitable for solving large-scale problems, to overcome this drawback, we propose to face the problem by means of a genetic algorithm. Differently from the Lagrangian heuristic, our approach is found to be effective also for large instances (up to 2000 jobs), as confirmed by numerical experiments.

Keywords: scheduling; multi-agent; genetic algorithm; large scale; local search

MSC: 90B35

check for updates

Citation: Avolio, M. Balancing the Average Weighted Completion Times in Large-Scale Two-Agent Scheduling Problems: An Evolutionary-Type Computational Study. *Mathematics* **2023**, *11*, 4034. https://doi.org/10.3390/math11194034

Academic Editors: Chin-Chia Wu, Win-Chin Lin, Jatinder N. D. Gupta and Xingong Zhang

Received: 17 August 2023
Revised: 13 September 2023
Accepted: 19 September 2023
Published: 22 September 2023

Copyright: © 2023 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Allocating resources to tasks over a given time horizon with the aim of optimizing one or more objectives is the focus of the theory of scheduling. Scheduling, as a decision-making process, plays a very important role in a lot of contexts such as manufacturing, production, transportation and distribution (see, for example, [1,2]). The difficulty in solving a scheduling problem, in general, strictly depends on the assumptions made on the problem itself. One of these assumptions concerns the number of agents involved in the considered scenario: while originally the presence of only one agent was always assumed, in the first years of the current century in [3,4], so-called multi-agent scheduling problems were introduced to take into account the possibility of having two or more agents.

In a multi-agent scenario (see [5]), since each agent has its own set of tasks (jobs) to be processed and its own objective function to be optimized, the difficulty in solving such kinds of problems is related to the fact that the agents share the same resources (machines). This has stimulated experts in this field, making the literature grow very fast in this direction. In the seminal paper [3], the authors have addressed a two-agent scheduling problem by considering the following objective functions: the maximum of regular functions (i.e., nondecreasing functions with respect to the completion times) associated with each job, the number of late jobs and the total weighted completion times. Additionally, the complexity of each problem was studied in connection with each objective function and with different structures of the processing system (single machine or shop). In [4], the authors have faced a multi-agent problem by minimizing three different objective functions: the makespan, the maximum lateness and the total weighted completion time. They proved that the

problem is polynomially solvable according to any of the three mentioned criteria, showing in addition that the problem becomes NP-hard in case a combination of them is minimized.

The multi-agent scheduling problems are applied in a lot of practical contexts, such as in telecommunication systems with two users [6], where the transmission of on-time data packets to one user is maximized, guaranteeing a certain amount of on-time data packets to the other one. Note that most of these problems are of the competitive type, since the agents compete with each other in using the common resources to optimize their own objective functions.

Differently from the more standard cases, in this work, we deal with a *cooperative*-type two-agent single-machine scheduling problem, where each agent cooperates in the same objective function aimed at balancing the average weighted completion times of two classes of jobs (one class per agent). This problem, that in the sequel we denote by BAWCT2 (Balancing the Average Weighted Completion Times with 2 agents), finds application in different contexts (see [7] for the details), such as:

- In logistics, to balance the delivery times in case a single drone is used for shipping;
- In manufacturing, to balance the production of semi-finished products characterized by high storage costs;
- In services, to balance the average waiting times of groups of people.

In particular, as mentioned in [7], the problem takes inspiration in an academic context, where a professor had the necessity to schedule the oral examinations of two groups of students belonging to two different classes, with the aim of balancing the respective average waiting times.

In passing, we highlight that similar balancing problems arise in general in the multi-agent framework whenever unfair solutions, maximizing the system utility, are unacceptable for the worse-off agent (see, for example, [8–10]). They are also present in more specific fields, such as in cloud computing when the necessity to balance the workload distribution among different servers occurs (see for example [11]).

BAWCT2 was introduced for the first time in [12], where the authors have considered its basic version, assuming that all the jobs have the same processing time (i.e., they are identical) and unitary weight. Successively, in [7], the problem was extended to the more general case, characterized by different processing times and weights. For such a general problem, the authors have proved the NP-hardness, providing, in addition, a Lagrangian relaxation-based heuristic algorithm, which consists of solving a finite number of linear assignment problems.

Apart from [7,12], to the best of our knowledge, this problem has not been explicitly considered in the literature anywhere else. Then, taking into account that solving a linear assignment problem at each iteration (as proposed in [7]) has a remarkable impact on the execution time, the main contribution of this work is mainly twofold: to propose a different technique based on the adoption of a genetic algorithm aimed at speeding up the resolution process and, at the same time, to test such approach on (new) large-scale instances of the problem.

This paper is organized as follows. In the next section, we formally state the problem, recalling also the two mathematical programming formulations (nonlinear and linear) proposed in [7]. In Section 3, we describe the used genetic algorithm, detailing the different strategies adopted in each phase. In Section 4, we present some numerical results showing the efficiency of our approach, which is able to deal with large-scale instances. Finally, in Section 5, some conclusions are drawn.

Throughout the paper, we will use the following notation. Given an index set L, we indicate by L_j the index of L in position j and by $P(L)$ a generic permutation of L. Moreover, in correspondence to any real number r, the nearest integer number less than or equal to r is denoted by $\lfloor r \rfloor$. Finally, given a one-dimensional array V, we denote by $V[j]$ the jth element of V.

2. Problem Definition

Let A and B be two different classes of jobs having cardinalities n_A and n_B, respectively. We indicate by $n \triangleq n_A + n_B$ the total number of jobs and by $J_A \triangleq \{1, \ldots, n_A\}$ and $J_B \triangleq \{n_A + 1, \ldots, n\}$ the index sets of A and B. For each $j \in J \triangleq J_A \cup J_B$, we denote by p_j the processing time of the jth job and by w_j the corresponding weight.

The objective of BAWCT2 is to determine a schedule of the jobs, with the aim of balancing the average weighted completion times of the two classes A and B. In particular, adopting the Graham notation [13], the problem can be stated as follows:

$$1||\bar{C}, \tag{1}$$

where

$$\bar{C} \triangleq \left| \frac{\sum_{j \in J_A} w_j C_j}{n_A} - \frac{\sum_{j \in J_B} w_j C_j}{n_B} \right|,$$

with C_j being the completion time of the jth job.

As mentioned in the previous section, to the best of our knowledge, this problem has been explicitly treated in the literature only in [7,12].

In particular, in [12], a simplified version of BAWCT2 was faced, assuming $w_j = 1$ and $p_j = p_{const}$ for $j \in J$. In such case, the problem reduces to a particular instance of the well-known subset-sum problem and it is solvable in linear time, or even in a constant time whenever the job-position assignment is not explicitly considered.

Successively, in [7], the authors have proved the NP-hardness of BAWCT2, providing, in addition, the following mathematical programming formulation of the problem as a nonsmooth quadratic assignment program:

$$\begin{cases} \min\limits_{x} & \left| \dfrac{1}{n_A} \left[\displaystyle\sum_{j \in J_A} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] - \right. \\ & \left. \dfrac{1}{n_B} \left[\displaystyle\sum_{j \in J_B} w_j \left(p_j + \sum_{t=1}^{n} \left(\sum_{l \in J} \sum_{i=1}^{t-1} p_l x_{li} \right) x_{jt} \right) \right] \right| \\ & \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ & \displaystyle\sum_{j \in J} x_{jt} = 1 \quad t = 1, \ldots, n \\ & x_{jt} \in \{0, 1\} \quad j \in J, \quad t = 1, \ldots, n, \end{cases} \tag{2}$$

where the decision variable x_{jt}, for $j \in J$ and $t = 1, \ldots, n$, is equal to 1 if the jth job is assigned to the position t and 0, otherwise, while the two groups of constraints are the classical assignment constraints imposing a bijection between jobs and positions. They have also proved that, by applying the Glover linearization technique [14] and introducing the auxiliary decision variables v and z_{jt} for $j \in J$ and $t = 1, \ldots, n$, formulation (2) reduces to the following mixed integer linear program (MILP):

$$
\begin{cases}
\min\limits_{x,v,z} & v \\
& v \geq \dfrac{1}{n_A}\left(p_A + \sum\limits_{j\in J_A}\sum\limits_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_B}\left(p_B + \sum\limits_{j\in J_B}\sum\limits_{t=1}^{n} w_j z_{jt}\right) \\
& v \geq \dfrac{1}{n_B}\left(p_B + \sum\limits_{j\in J_B}\sum\limits_{t=1}^{n} w_j z_{jt}\right) - \dfrac{1}{n_A}\left(p_A + \sum\limits_{j\in J_A}\sum\limits_{t=1}^{n} w_j z_{jt}\right) \\
& d_t x_{jt} + \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li} - z_{jt} \leq d_t \quad j \in J, \quad t = 1,\ldots,n \\
& z_{jt} \leq \sum\limits_{l\in J}\sum\limits_{i=1}^{t-1} p_l x_{li} \quad j \in J, \quad t = 1\ldots,n \\
& z_{jt} \leq d_t x_{jt} \quad j \in J, \quad t = 1,\ldots,n \\
& z_{jt} \geq 0 \quad j \in J, \quad t = 1,\ldots n \\
& \sum\limits_{t=1}^{n} x_{jt} = 1 \quad j \in J \\
& \sum\limits_{j\in J} x_{jt} = 1 \quad t = 1,\ldots,n \\
& x_{jt} \in \{0,1\} \quad j \in J, \quad t = 1,\ldots,n,
\end{cases}
\tag{3}
$$

where $p_A \triangleq \sum_{j\in J_A} w_j p_j$, $p_B \triangleq \sum_{j\in J_B} w_j p_j$ and $d_t \triangleq (t-1)\sum\limits_{j\in J} p_j$ for $t = 1,\ldots,n$.

The above MILP is at the basis of the Lagrangian heuristic proposed in [7] and based on solving successive linear assignment problems. Although this technique has provided valuable numerical results, its main limit consists in the fact that solving at each iteration a linear assignment problem has a nonnegligible computational impact in case of large-scale problems (i.e., whenever n is remarkably high). For this reason, to overcome this drawback, in the next section we propose to use a genetic algorithm aimed at speeding up the overall resolution process.

For the reader's convenience, we conclude this section by detailing in Table 1 the main characteristics of our approach with respect to [7,12], taking into account that, in [12], no type of numerical experiments was carried out.

Table 1. BAWCT2: comparison among different approaches.

Reference	Characteristics of the Problem	Algorithm Type	Maximum Number of Jobs in the Numerical Experiments
[12]	$p_j = p_{const}$ and $w_j = 1$, $j \in J$	Exact	-
[7]	General values of p_j and w_j, $j \in J$	Lagrangian heuristic	500
Our contribution	General values of p_j and w_j, $j \in J$	Genetic algorithm	2000

3. The Genetic Algorithm for BAWCT2

Genetic algorithms (GAs) are metaheuristic search approaches that are successfully applied to solve different varieties of NP-hard optimization problems (see [15]). In particular, given an optimization problem, a GA is based on the following evolution paradigm.

One starts from an initial population, where with each individual, a genotype, representing a possible solution to the problem, is associated. The quality of such a solution is measured by its fitness value, obtained by evaluating the so-called fitness function and expressing how well the solution fits the problem. In passing from a generation to the successive one, the aim is to possibly increase the overall fitness of the population, by generating, via specific genetic operators, new individuals (offspring) characterized by better fitness values.

The basic schema of GA is reported in Algorithm 1.

Algorithm 1: GA

```
1  Pop ← GenerateInitialPopulation()
2  repeat
3      NewPop ← ∅
4      repeat
5          parents ← Selection(Pop)
6          offspring ← Crossover(parents)
7          offspring ← Mutation(offspring)
8          NewPop ← NewPop ∪ offspring
9      until NewPop is complete
10     Pop ← UpdatePop(Pop,NewPop)
11 until a stopping criterion is satisfied
```

Even if in the literature there are plenty of strategies to implement the steps of Algorithm 1, each of those strategies can be, in general, adapted and tailored to the specific case, making GAs really attractive for solving many optimization problems. Some examples are [16,17], where the well-known traveling salesman problem was faced, and [18–27], where GAs were used for solving scheduling problems.

Before describing in detail how each step of Algorithm 1 is implemented for solving BAWCT2, it is worth specifying that, in our approach, at each iteration, the previous population is completely replaced by the new one, maintaining only the fittest current individual in order to preserve a monotonic behavior of the best current fitness value (line 10, Algorithm 1).

3.1. Encoding and Fitness Evaluation

The first choice in the implementation of GA is regarding the encoding of each individual, representing a feasible solution to the problem. Since we are dealing with a single machine scheduling problem, in such a case, the standard way to encode an individual I is to use a one-dimensional array V_I, such that $V_I[t] = j$ if and only if the job j is scheduled in the position t. As a consequence, according to this notation, the job processed in the position t will be denoted by $[t]$.

Another key point characterizing GAs is the definition of the fitness function, aimed at providing the fitness value of an individual. Since individuals with higher fitness are preferred by GAs, and, on the other hand, BAWCT2 is a minimization problem, we propose using the following fitness function:

$$fitness_I = 1/f(I), \tag{4}$$

where $f(I)$ is the objective function value $\bar{C}$ of BAWCT2 in correspondence to the solution represented by the individual I.

Note that using formula (4) ensures the highlighting of even slight differences in function f and, additionally, when $f(I) = 0$ (corresponding to a perfect balanced optimal solution), it trivially follows that $fitness_I \to +\infty$.

3.2. Initial Population

Once the encoding strategy is defined, the next step to be performed in GAs is the generation of the initial population. A review of the main techniques used in this phase is reported in [28].

For solving BAWCT2, we have tested the following three different techniques: *Random generation*, *Alternated generation* and *Bidirectional generation*. If, on the one hand, using the *Random generation* strategy could generate strongly unbalanced solutions due to the completely random generation of the population, on the other hand, *Alternated generation* and the *Bidirectional generation* techniques should hopefully generate better starting points, since they are based on more elaborate criteria. In particular, while the *Alternated generation* technique was used for generating a starting point in the Lagrangian relaxation approach proposed in [7], the *Bidirectional generation* is inspired by the strategy at the basis of the exact algorithm which provides an optimal solution in case $w_j = 1$ and $p_j = p_{const}$ for each $j \in J$ (see [12]).

In what follows, we formally describe the three strategies according to the above proposed encoding.

- *Random generation*. The genotype of each individual I, corresponding to a feasible solution of BAWCT2, is given by a random permutation $P(J)$ of J. In other words, $[i] = P(J)_i$ for $i = 1, \ldots, n$.
- *Alternated generation*. The genotype of each individual I is given by alternating the jobs of the two classes, until one of the two sets is completely scheduled. The remaining jobs, if any, are accommodated at the end of the sequence.
  More formally, let $P(J_A)$ and $P(J_B)$ be two random permutations of J_A and J_B, respectively. Then, for each $i = 1, \ldots, \min\{n_A, n_B\}$, we initially set

  $$[2i-1] = P(J_A)_i \text{ and } [2i] = P(J_B)_i.$$

  Successively, in case $n_A \neq n_B$, the remaining jobs of the largest class are assigned at the end of the sequence, starting from the position $2 \cdot \min\{n_A, n_B\} + 1$.
- *Bidirectional generation*. The genotype of each individual is provided by an iterative procedure, which consists of accommodating, at each iteration, two jobs of one class (alternately chosen between A and B) in the first and in the last currently available positions of the sequence, respectively.
  In particular, let $P(J_A)$ and $P(J_B)$ be two random permutations of J_A and J_B, respectively. Then, at each iteration k, for $k = 0, \ldots, \lfloor n_A/2 \rfloor + \lfloor n_B/2 \rfloor - 1$, we set either

  $$[k+1] = P(J_C)_t \quad \text{and} \quad [n-k] = P(J_C)_{t+1} \tag{5}$$

  or

  $$[k+1] = P(J_C)_{t+1} \quad \text{and} \quad [n-k] = P(J_C)_t, \tag{6}$$

  where J_C is the index set that coincides, alternately, at each iteration, with either J_A or J_B. The index t is initialized to 1 and is increased by 2 at the end of each iteration.
  Obviously, when only one of the two classes is completely scheduled, J_C remains fixed to the index set of the other class. Moreover, in case n_A and/or n_B are odd, the remaining jobs are put in the middle of the sequence, using a greedy strategy based on the evaluation of the fitness function. The fitness value is also considered in the final choice of the configuration between (5) and (6).

It is worth noting that, for all of the above proposed techniques, the probability of obtaining two times the same schedule as the output of the generation process is very low, since they are defined in the function of random permutations. This is a crucial point, as it guarantees a *diversification* in the initial population.

3.3. Selection

Selection (see [29]) is a crucial step in designing GAs, since it defines the chance of a given individual to participate in the reproduction process. Considering that a convergence to optimal solutions is desired, it is recommended to provide a major chance to individuals characterized by high fitness values.

In this work, three different well-known selection techniques were adopted: *Roulette wheel*, *Binary tournment* and *k-tournment*.

- *Roulette wheel*. The selection is performed by simulating a roulette wheel, in which the chance of an individual to be selected is represented by a portion of the roulette. In particular, for each individual I, since the size of its portion must be proportional to the fitness value, the probability p_I to be selected is set as

$$p_I = \frac{fitness_I}{\sum_{i \in Pop} fitness_i},$$

  where Pop represents the overall population.
- *Binary tournament*. Between two randomly selected individuals, the one characterized by the highest fitness value is chosen for reproduction.
- *k-tournament*. It is the same as the *Binary tournament*, apart from the fact that k individuals, with $2 < k \leq p_{size}$, are involved in the tournament, where p_{size} is the size of the population.

3.4. Crossover

Once the parents are selected, they have to reproduce in order to generate new children (offspring). The reproduction process is simulated by using crossover operators, aimed at creating new individuals whose features are a mix of the genetic properties of the parents. From the algorithmic point of view, this phase is very important since it allows the exploration of the solution space.

In the literature, a lot of crossover operators have been proposed for the scheduling problems and, in general, for sequencing problems, such as the traveling salesman problem (for a general overview, see [30,31]).

In [26], the authors highlighted that the crossover operators, mainly proposed for the traveling salesman problem, do not perform very well in a scheduling context. As a consequence, taking into account this consideration, for solving BAWCT2, we chose to implement the following three crossover strategies: *One-point crossover*, *Two-point crossover* and *Position-based crossover*.

Given two parents I_1 and I_2, these operators can be described as follows:

- *One-point crossover.* An index $i \in [1, n]$ is randomly selected. Then, with the same probability, either the first i jobs or the last $n - i$ ones are inherited from I_1, while the remaining ones are inserted in the sequence preserving the order they have in I_2 (see Figure 1).
- *Two-point crossover.* Two indices $i, j \in [1, n]$ are randomly selected. Assuming $i \leq j$, the first i jobs and the last $n - j$ ones are inherited from I_1, while the remaining $j - i$ are inserted in the sequence preserving the order they have in I_2 (see Figure 2).
- *Position-based crossover.* An index, $k \in [1, n]$, is randomly selected and k different positions v_i, $i = 1, \ldots, k$, are sampled in the same interval $[1, n]$. Then, the jobs in positions v_i, for $i = 1, \ldots, k$, are inherited from I_1, while the remaining ones are inserted in the sequence, preserving the order they have in I_2 (see Figure 3).

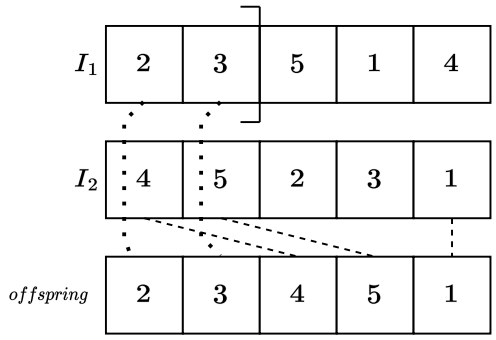

Figure 1. One-point crossover with $i = 2$.

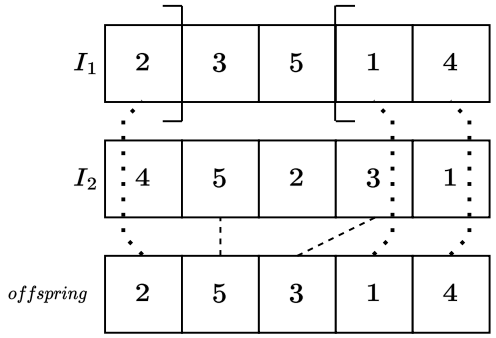

Figure 2. Two-point crossover with $i = 1$ and $j = 3$.

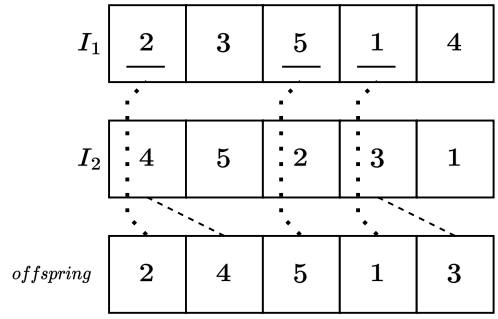

Figure 3. Position-based crossover with $k = 3$, $v_1 = 1, v_2 = 3$ and $v_3 = 4$.

3.5. Mutation

Another important operator in GAs is the mutation that, together with the crossover operator, allows the exploration of the solution space. In particular, the aim of a mutation operator is to perturb the current solution by applying random changes. Inspired by [26], in this work, the following mutation operators were tested:

- *Arbitrary two-job change.* Two positions $i, j \in [1, n]$ are randomly selected and the jobs currently scheduled in these positions are swapped (see Figure 4).
- *Shift change.* Two positions $i, j \in [1, n]$ are randomly selected, and the job in the position i is moved to the position j, shifting all the intermediate jobs (see Figure 5).

Additionally, considering any set of adjacent jobs as a *batch*, we also defined the following mutation operator:

- *Arbitrary batch change.* Given two random positions $i, j \in [1, n]$, assume $i \leq j$. Then, an integer s (the batch size) is randomly determined in the interval $[1, \min\{j - i, n + 1 - j\}]$ and, for $k = 0, \ldots, s - 1$, the jobs in the positions $i + k$ and $j + k$ are swapped (see Figure 6).

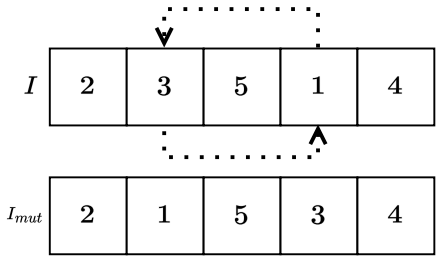

Figure 4. Arbitrary two-job change with $i = 2$ and $j = 4$.

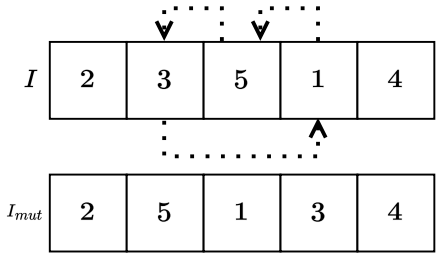

Figure 5. Shift change with $i = 2$ and $j = 4$.

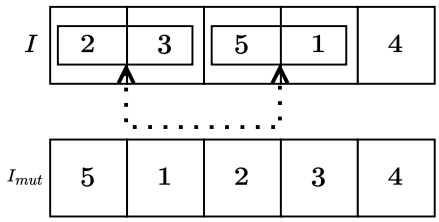

Figure 6. Arbitrary batch change with $i = 1$, $j = 3$ and $s = 2$.

3.6. Local Search

The aim of the local search, that we immediately apply after the mutation, is to improve the fitness value of a new offspring and to possibly avoid a premature convergence.

In a lot of scenarios considered in our numerical experiments, the local search has turned out to be very effective, since, starting from a good solution produced in the previous phases, it has often provided a perfect balancing between A and B, i.e., a global solution located in the current neighborhood.

In particular we have adopted the following strategy, proposed in [7] and detailed in Algorithm 2. Given an initial offspring O_{init}, for each of the $\frac{n(n-1)}{2}$ couples of indices (i, j), with $i < j$, we check whether, by swapping the jobs in the positions i and j, a decreasing in the objective function of BAWCT2 (i.e., a better fitness value) is obtained. If so, the exchange is executed.

Algorithm 2: Local search

```
Input: offspring O_init
for i ← 1 to n − 1 do
    for j ← i + 1 to n do
        O ← Swap(O_init,i,j)
        Δ ← fitness_O - fitness_{O_init}
        if Δ > 0 then
            O_init ← O
        end
    end
end
```

3.7. The Overall Algorithm

The overall GA, proposed for solving BAWCT2, is named GA-BAWCT2 and is detailed in Algorithm 3 (see also Figure 7 for a graphical description). The input parameters are the following:

- p_{size}: size of the population;
- p_c: probability of performing the crossover;
- p_m: probability of performing the mutation;
- GenerateIndividual: adopted strategy for generating the initial population;
- Selection: adopted strategy for the selection operator;
- Crossover: adopted strategy for the crossover operator;
- Mutation: adopted strategy for the mutation operator;
- $maxIter$: maximum number of iterations.

Algorithm 3: GA-BAWCT2

Input: p_{size}, p_c, p_m, $maxIter$

1 $Pop \leftarrow \emptyset$
2 **for** $i \leftarrow 1$ ***to*** p_{size} **do**
3 $\quad Pop_i \leftarrow$ GenerateIndividual()
4 $\quad$ Evaluate(Pop_i)
5 $\quad Pop \leftarrow Pop \cup \{Pop_i\}$
6 $I_{best} \leftarrow$ Best(Pop)
7 iter $\leftarrow 0$
8 **while** $fitness_{I_{best}} < +\infty$ *and* $iter < maxIter$ **do**
9 $\quad NewPop \leftarrow \emptyset$
10 $\quad$ **for** $i \leftarrow 1$ ***to*** p_{size} **do**
11 $\quad\quad parents \leftarrow$ Selection(Pop)
12 $\quad\quad$ **if** $rand([0,1]) < p_c$ **then**
13 $\quad\quad\quad offspring \leftarrow$ Crossover($parents$)
14 $\quad\quad$ **else**
15 $\quad\quad\quad offspring \leftarrow parents_1$
16 $\quad\quad$ **if** $rand([0,1]) < p_m$ **then**
17 $\quad\quad\quad offspring \leftarrow$ Mutation($offspring$)
18 $\quad\quad$ Evaluate($offspring$)
19 $\quad\quad offspring \leftarrow$ LocalSearch($offspring$)
20 $\quad\quad NewPop \leftarrow NewPop \cup \{offspring\}$
21 $\quad toDiscard \leftarrow O \in NewPop$ s.t. $\exists P \in NewPop : fitness_P \geq fitness_O$
22 $\quad Pop \leftarrow NewPop \cup \{I_{best}\} \setminus \{toDiscard\}$
23 $\quad I_{best} \leftarrow$ Best(Pop)
24 $\quad$ iter $\leftarrow$ iter $+1$

The following comments on Algorithm 3 are in order. Given the current population (Pop), initialized at Steps 1–5, at each iteration we create a new population ($NewPop$), by randomly selecting $2p_{size}$ numbers in the interval $[0, 1]$ to decide whether, in generating the new current individual, the crossover and mutation have to be applied. In case the crossover is not carried out, the offspring coincides with the first parent, selected at Step 11. Once an offspring is generated, we proceed with the eventual mutation (Step 17, depending on p_m) and we evaluate the corresponding fitness value (Step 18), in order to perform the local search (Step 19). Then, the new current population (Step 22) is obtained by substituting, in the new population, one individual (different from the best one) with the best one of the old population. In this way, we guarantee the monotonicity of the fitness function in corresponding to the best individual (I_{best}) of the current population, which, at the end of the algorithm, constitutes a heuristic solution to the problem.

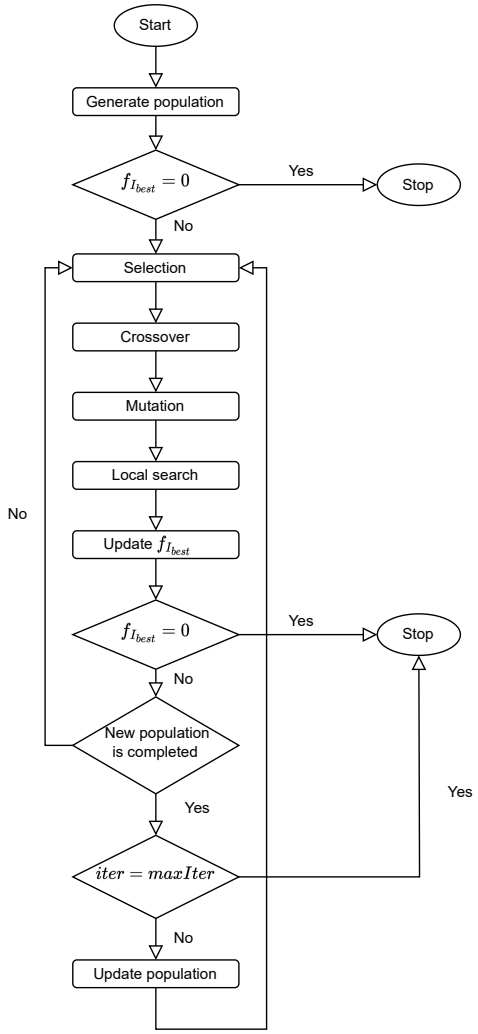

Figure 7. Algorithm GA-BAWCT2: a graphical description.

Finally, we observe that the algorithm terminates either by reaching the maximum number of iterations or because a perfect balancing (i.e., $f(I_{best}) = 0$) has been determined, with the latter corresponding to $fitness_{I_{best}} = +\infty$.

4. Numerical Experiments

Algorithm 3 (GA-BAWCT2) was implemented in Java (version 14.02) and run on a Windows 11 system, characterized by 16 GB of RAM and a 2.30 GHz Intel Core i7 processor. For each run, the time limit was fixed to 3600 s, as in [7].

We recall that a preliminary step in the implementation of GAs is the parameters setting, which is crucial since it strongly affects the performance of the algorithm. Moreover, because a vast number of possible settings has to be considered, this is not a trivial step.

In particular, apart from the maximum number of iterations ($maxIter$) that we have set equal to 1000 as in [7], our heuristic is defined in a function of three numerical input parameters (p_{size}, p_c, p_m) and four categorical parameters (the inner procedures *Initial population*, *Selection*, *Crossover* and *Mutation*), the latter to be chosen among the different strategies proposed in the previous section and summarized in Table 2. On the possible values of the numerical parameters, our proposal is reported in Table 3 (three different values for each parameter), resulting in a total of seven parameters and $3^7 = 2187$ combinations for the overall setting of the algorithm. Due to the huge number of these combinations, the parameters' setting was simplified by adopting the strategy described in the next subsection.

Table 2. Algorithm GA-BAWCT2: Domains of the categorical parameters.

Parameters	**Proposed Values**		
Initial population	Random	Alternated	Bidirectional
Selection	Roulette	Binary tournament	k-tournament
Crossover	One-point	Two-point	Position-based
Mutation	Arbitrary two-job	Shift	Arbitrary batch

Table 3. Algorithm GA-BAWCT2: Domains of the numerical parameters.

Parameters	**Proposed Values**		
p_{size}	20	50	100
p_c	0.75	0.85	0.95
p_m	0.05	0.25	0.5

4.1. Parameters' Setting

Initially, the three numerical parameters (p_{size}, p_c and p_m) were fixed to the median of the proposed values, i.e., 50, 0.85 and 0.25, respectively. Then, in correspondence to such values of the numerical parameters, we have considered all the 3^4 combinations of the four categorical parameters, and, for each combination, we ran the code on 30 randomly generated instances of the problem. Such instances were obtained similarly to [7], as follows. We considered three different scenarios in correspondence to $n_A = n_B = k$, with $k \in \{100, 150, 250\}$, and, for each scenario, we generated ten instances by uniformly sampling p_j and w_j in the interval $[1, 3n]$. Taking into account that the algorithm was able to solve to optimality (i.e., $fitness_{I_{best}} = +\infty$) all of these instances, the execution time was considered as the performance measure. The optimal setting of the categorical parameters, i.e., the one in correspondence to which we have obtained the lowest average execution time, resulted in the following:

- Initial population: Bidirectional generation;
- Selection: Binary tournament;
- Crossover: Two-point crossover;
- Mutation: Shift change.

Successively, in correspondence with the above fixed best configuration of the categorical parameters, we similarly proceeded to determine the optimal setting of the numerical parameters by considering all the 3^3 combinations. We obtained:

- $p_{size} = 20$;
- $p_c = 0.85$;
- $p_m = 0.5$.

For the sake of completeness, in Figures 8 and 9, the main effects' plots of the categorical and of the numerical parameters, respectively, are reported, using as a metric the execution time.

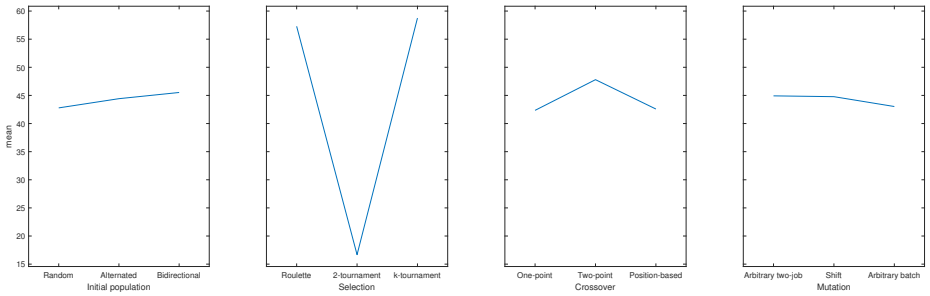

Figure 8. Algorithm GA-BAWCT2: Main effects' plots of categorical parameters.

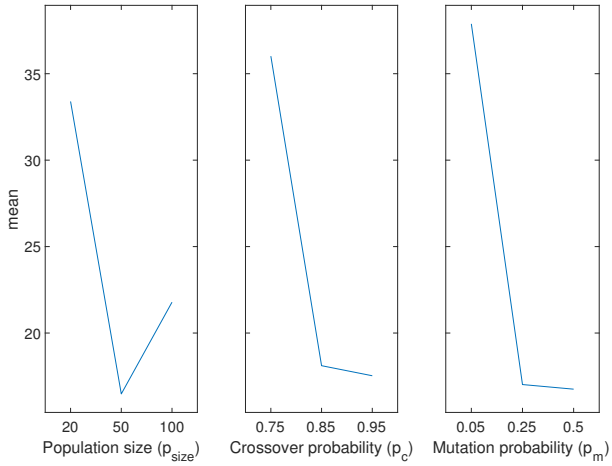

Figure 9. Algorithm GA-BAWCT2: Main effects' plots of numerical parameters.

4.2. Testing Phase

In correspondence with the above parameters' setting, the code was tested on some of the instances used in [7] (see Table 5 therein), and on some new larger randomly generated instances. We recall, in fact, that our main purpose is to show the effectiveness of GA, especially in solving large-scale problems.

The dataset used in Table 5 of [7] consists of six scenarios, each of them composed of 20 instances. These scenarios were defined by taking $n_A = n_B = k$, with $k \in \{30, 50, 100, 150, 200, 250\}$, and by randomly sampling p_j and w_j from the uniform interval $[1, 3n]$. Our results are reported in Table 4, in comparison with the results obtained by the authors in [7], exactly on the same machine, where the GA-BAWCT2 code was further run. In particular, for both the algorithms and for each scenario, we report:

- The average execution time (in seconds);
- The average best incumbent, i.e., the average of the best objective function values of BAWCT2, found by the algorithm;
- The number of instances certainly solved to optimality, i.e., how many times the algorithm exited with a zero objective function value.

Table 4. Algorithm GA-BAWCT2: numerical results on medium instances, in comparison with the Lagrangian heuristic proposed in [7].

		Lagrangian Heuristics			**GA-BAWCT2**		
n_A	n_B	**Time (s)**	**Best Incumbent**	**#Solved to Opt**	**Time (s)**	**Best Incumbent**	**#Solved to Opt**
30	30	1.08	0	20/20	0.026	0	20/20
50	50	12.01	0	20/20	0.101	0	20/20
100	100	144.58	0	20/20	1.35	0	20/20
150	150	737.13	0	20/20	3.45	0	20/20
200	200	1467.99	0.001	16/20	11.97	0	20/20
250	250	1847.42	0.005	4/20	42.72	0	20/20

Looking at the results, it is evident that Algorithm GA-BAWCT2 overcomes the Lagrangian heuristic: in fact, differently from the Lagrangian approach, our algorithm is able to solve to optimality all of the 120 instances, and moreover, in terms of execution time, it is clearly the winner with a difference of two orders of magnitude.

In order to test our approach on more and larger instances, we randomly generated a new large-scale dataset consisting of seven scenarios, each of them constituted by ten

instances. In such case, a maximum number of jobs equal to 2000 is considered, taking again p_j and w_j as random integer numbers from a uniform distribution on the interval $[1, 3n]$. The obtained results are reported in Table 5, from which we can observe that Algorithm GA-BAWCT2 was certainly able solve to optimality in more than 50% of the overall instances, providing, for each scenario, an average value of the incumbent that is less then 0.001.

Table 5. Algorithm GA-BAWCT2: numerical results on large instances.

n_A	n_B	Time (s)	Best Incumbent	#Solved to Opt
500	500	619.81	0	10/10
500	600	2581.38	0.0002	5/10
500	700	2370.27	0.0001	5/10
500	800	2589.24	0.0002	4/10
500	900	3220.57	0.0002	4/10
500	1000	1984.54	0.0002	8/10
1000	1000	2426.04	0.0006	6/10

5. Conclusions

In this paper, a genetic algorithm was adopted for balancing the average weighted completion times of two classes of jobs in a single-machine scheduling problem.

For implementing each phase, different techniques were tested, with some of them taken from the literature, while others were introduced for the first time in this work.

The numerical results obtained on a dataset taken from the literature demonstrated the efficiency of the new proposal, which is able to significantly speed up the resolution process with respect to the state of the art. Additional numerical experiments also proved that our approach is able to effectively deal with large-scale instances of up to 2000 jobs.

Future research could be devoted to the extension of the genetic algorithm to the bi-objective case, by taking into account the additional minimization of the average weighted completion times of one of the two agents.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Pinedo, M.L. *Planning and Scheduling in Manufacturing and Services*; Springer: New York, NY, USA, 2009.
2. Pinedo, M.L. *Scheduling: Theory, Algorithms, and Systems*, 6th ed.; Springer: Cham, Switzerland, 2022.
3. Agnetis, A.; Mirchandani, P.B.; Pacciarelli, D.; Pacifici, A. Scheduling problems with two competing agents. *Oper. Res.* **2004**, *52*, 229–242. [CrossRef]
4. Baker, K.R.; Smith, J.C. A multiple-criterion model for machine scheduling. *J. Sched.* **2003**, *6*, 7–16. [CrossRef]
5. Agnetis, A.; Billaut, J.C.; Gawiejnowicz, S.; Pacciarelli, D.; Soukhal, A. *Multiagent Scheduling: Models and Algorithms*; Springer: Berlin/Heidelberg, Germany, 2014.
6. Arbib, C.; Smriglio, S.; Servilio, M. A competitive scheduling problem and its relevance to UMTS channel assignment. *Networks* **2004**, *44*, 132–141. [CrossRef]
7. Avolio, M.; Fuduli, A. A Lagrangian heuristics for balancing the average weighted completion times of two classes of jobs in a single-machine scheduling problem. *EURO J. Comput. Optim.* **2022**, *10*, 100032. [CrossRef]
8. Agnetis, A.; Chen, B.; Nicosia, G.; Pacifici, A. Price of fairness in two-agent single-machine scheduling problems. *Eur. J. Oper. Res.* **2019**, *276*, 79–87. [CrossRef]
9. Bahel, E.; Trudeau, C. Stability and fairness in the job scheduling problem. *Games Econ. Behav.* **2019**, *117*, 1–14. [CrossRef]
10. Zhang, Y.; Zhang, Z.; Liu, Z. The price of fairness for a two-agent scheduling game minimizing total completion time. *J. Comb. Optim.* **2022**, *44*, 2104–2122. [CrossRef]
11. Gohil, B.N.; Patel, D.R. Load balancing in cloud using improved gray wolf optimizer. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6888. [CrossRef]

12. Avolio, M.; Fuduli, A. A subset-sum type formulation of a two-agent single-machine scheduling problem. *Inf. Process. Lett.* **2020**, *155*, 105886. [CrossRef]
13. Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Kan, A.H.G.R. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discret. Math.* **1979**, *5*, 287–326.
14. Glover, F. Improved linear integer programming formulations of nonlinear integer problems. *Manag. Sci.* **1975**, *22*, 455–460. [CrossRef]
15. Kramer, O. Studies in Computational Intelligence. In *Genetic Algorithm Essentials*; Springer: Cham, Switzerland, 2017; Volume 679, pp. 3–84.
16. Larrañaga, 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. [CrossRef]
17. Potvin, J.. Genetic algorithms for the traveling salesman problem. *Ann. Oper. Res.* **1996**, *63*, 339–370. [CrossRef]
18. Abreu, L.R.; Cunha, J.O.; Prata, B.A.; Framinan, J.M. A genetic algorithm for scheduling open shops with sequence-dependent setup times. *Comput. Oper. Res.* **2020**, *113*, 104793. [CrossRef]
19. Fan, J.; Zhang, C.; Liu, Q.; Shen, W.; Gao, L. An improved genetic algorithm for flexible job shop scheduling problem considering reconfigurable machine tools with limited auxiliary modules. *J. Manuf. Syst.* **2022**, *62*, 650–667. [CrossRef]
20. Gonçalves, J.F.; De Magalhães Mendes, J.J.; Resende, M.G.C. A hybrid genetic algorithm for the job shop scheduling problem. *Eur. J. Oper. Res.* **2005**, *167*, 77–95. [CrossRef]
21. Liu, Z.; Wang, J.; Zhang, C.; Chu, H.; Ding, G.; Zhang, L. A hybrid genetic-particle swarm algorithm based on multilevel neighbourhood structure for flexible job shop scheduling problem. *Comput. Oper. Res.* **2021**, *135*, 105431. [CrossRef]
22. Pezzella, F.; Morganti, G.; Ciaschetti, G. A genetic algorithm for the flexible job-shop scheduling problem. *Comput. Oper. Res.* **2008**, *35*, 3202–3212. [CrossRef]
23. Phu-ang, A.; Thammano, A. Memetic algorithm based on marriage in honey bees optimization for flexible job shop scheduling problem. *Memetic Comput.* **2017**, *9*, 295–309. [CrossRef]
24. Raeesi N., M.R.; Kobti, Z. A memetic algorithm for job shop scheduling using a critical-path-based local search heuristic. *Memetic Comput.* **2012**, *4*, 231–245. [CrossRef]
25. Tan, M.; Yang, H.; Su, Y. Genetic algorithms with greedy strategy for green batch scheduling on non-identical parallel machines. *Memetic Comput.* **2019**, *11*, 439–452. [CrossRef]
26. Yu, C.; Semeraro, Q.; Matta, A. A genetic algorithm for the hybrid flow shop scheduling with unrelated machines and machine eligibility. *Comput. Oper. Res.* **2018**, *100*, 211–229. [CrossRef]
27. Zhang, G.; Hu, Y.; Sun, J.; Zhang, W. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm Evol. Comput.* **2020**, *54*, 100664. [CrossRef]
28. Kazimipour, B.; Li, X.; Qin, A.K. A review of population initialization techniques for evolutionary algorithms. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, 6–11 July 2014; pp. 2585–2592.
29. Yadav, S.L.; Sohal, A. Comparative study of different selection techniques in genetic algorithm. *Int. J. Eng. Sci. Math.* **2017**, *6*, 174–180.
30. Poon, P.; Carter, J. Genetic algorithm crossover operators for ordering applications. *Comput. Oper. Res.* **1995**, *22*, 135–147. [CrossRef]
31. Umbarkar, A.; Sheth, P. Crossover operators in genetic algorithms: A review. *ICTACT J. Soft Comput.* **2015**, *6*, 1083–1092.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.