applied sciences

MDPI

*Article*

# A Hybrid Search Using Genetic Algorithms and Random-Restart Hill-Climbing for Flexible Job Shop Scheduling Instances with High Flexibility

**Nayeli Jazmin Escamilla-Serna †, Juan Carlos Seck-Tuoh-Mora *,†, Joselito Medina-Marin, Irving Barragan-Vite and José Ramón Corona-Armenta**

Área Académica de Ingeniería, Instituto de Ciencias Básicas e Ingeniería, Universidad Autónoma del Estado de Hidalgo, Carr. Pachuca-Tulancingo Km. 4.5, Pachuca 42184, Hidalgo, Mexico

* Correspondence: jseck@uaeh.edu.mx

† These authors contributed equally to this work.

**Abstract:** This work presents a novel hybrid algorithm called GA-RRHC based on genetic algorithms (GAs) and a random-restart hill-climbing (RRHC) algorithm for the optimization of the flexible job shop scheduling problem (FJSSP) with high flexibility (where every operation can be completed by a high number of machines). In particular, different GA crossover and simple mutation operators are used with a cellular automata (CA)-inspired neighborhood to perform global search. This method is refined with a local search based on RRHC, making computational implementation easy. The novel point is obtained by applying the CA-type neighborhood and hybridizing the aforementioned two techniques in the GA-RRHC, which is simple to understand and implement. The GA-RRHC is tested by taking four banks of experiments widely used in the literature and comparing their results with six recent algorithms using relative percentage deviation (RPD) and Friedman tests. The experiments demonstrate that the GA-RRHC is a competitive method compared with other recent algorithms for instances of the FJSSP with high flexibility. The GA-RRHC was implemented in Matlab and is available on Github.

**Keywords:** flexible job shop scheduling instances; genetic operators; local search methods; cellular automata



**Citation:** Escamilla-Serna, N.J.; Seck-Tuoh-Mora, J.C.; Medina-Marin, J.; Barragan-Vite, I.; Corona-Armenta, J.R. A Hybrid Search Using Genetic Algorithms and Random-Restart Hill-Climbing for Flexible Job Shop Scheduling Instances with High Flexibility. *Appl. Sci.* **2022**, *12*, 8050. https://doi.org/10.3390/app12168050

Academic Editor: Valentino Santucci

Received: 9 June 2022
Accepted: 7 August 2022
Published: 11 August 2022



## 1. Introduction

Production planning is a priority factor in modern manufacturing systems [1], where a critical aspect is the scheduling of operations and resource allocation [2]. Each industry must find a better solution for its respective production scheduling jobs to execute efficient manufacturing on time or launch new products to meet market demands satisfactorily [3].

This paper analyses the topic of task scheduling with flexible machines, also known as the flexible job shop scheduling problem (FJSSP), especially concerning instances with high flexibility, where many machines can process the same operation. The FJSSP reflects the current problem faced by manufacturing industries in the allocation of limited but highly flexible resources to perform tasks in the shortest possible time [4].

In the FJSSP, the goal is to find the most appropriate job sequencing, with each job involving operations with precedence restrictions, in an environment where several machines can perform the same operation but quite possibly in different processing times. Thus, the FJSSP is an NP-hard combinatorial problem where for a bigger number of jobs, there is an exponentially higher number of possible solutions [5]. It is thus impossible to review all solutions to find the optimal scheduling in a suitable time that minimizes the processing time of all operations [6].

Many mathematical methods have been proposed to resolve scheduling problems [7]; some works have approached the FJSSP using the branch and bound method [8], linear

programming [9], or Lagrangian relaxation [10]. These methods ensure global convergence and have worked very well in solving small instances, but their computational time makes them impractical for problems with dozens of scheduling operations [6]. That is why many researchers have chosen to move toward hybrid heuristic and metaheuristic techniques.

Over the years, distinct metaheuristic methods have been applied to solve combinatorial problems, including evolutionary algorithms using the survival of the fittest; an example is genetic algorithms (GAs) [11,12]. Other metaheuristics are ant colony optimization (ACO) [13], particle swarm optimization (PSO) [14], tabu search (TS) [15], etc. These algorithms have been adapted to different programming problems, finding good solutions with low computational time.

The FJSSP is an extension of the classic job shop scheduling problem [16]. In the inital problem, the optimal allocation of operations is sought on a set of fixed machines. In the FJSSP, several feasible machines can perform the same operation, often with distinct processing times.

Two problems must be considered to solve an instance of the FJSSP: the order of operations and the assignment of machines. In [17], the FJSSP is approached heuristically, applying dispatch rules and tabu search and further introducing 15 instances with different numbers of tasks and machines. Since this work, different heuristics and metaheuristics conducting some hybrid searches have been investigated to solve this problem.

An algorithm using TS and a simplified computation of the makespan are explained in [18], highlighting the importance of critical operations for local search. In [19], a hybrid algorithm (HA) using GA and TS is proposed to minimize the makespan. Its model maintains a good balance between exploitation and exploration. In [20], a multi-objective problem (MO-FJSP) is addressed applying the non-sorting genetic algorithm (NSGA-II) together with a bee evolutionary guide (BEG-NSGA-II) to minimize the makespan, the workload of the most-busy machine, and the total workload.

In [21], a combination of genetic algorithm and a variable neighborhood descent method is shown to optimize a multi-objective version that takes into account the makespan, the total workload, and the workload of the most-busy machine, using two methods of local search. Another hybrid algorithm using the PSO and TS is presented in [22], again for a multi-objective problem. The use of different variable neighborhoods (VNs) to refine the local search is proposed in [23] to minimize the makespan. Another algorithm combining TS and VNs is described in [24] for a different multi-target version of the FJSSP. In [25], the FJSSP is studied considering maintenance costs, where a hybrid genetic algorithm (HGA) uses a local search based on simulated annealing (SA). The combination of the Harmony Search (HS) algorithm with other heuristic and random techniques is analyzed in [26] to handle two discrete vectors, one for the sequence of operations and the other for machine allocation, to minimize makespan. Another hybrid algorithm between artificial bees and TS is presented in [27], where the quality and diversity of solutions is rated with three metrics.

Task rescheduling is investigated in [28], for which another hybrid technique was proposed using a two-stage artificial bee colony algorithm with three rescheduling strategies. Another hybrid method is introduced in [29] by applying an artificial bee colony algorithm (ABC) and a TS, introducing new reprogramming processes. In [30], a hybrid algorithm is implemented using PSO with random-restart hill-climbing (RRHC), obtaining a competitive and straightforward algorithm compared with other techniques. Another hybrid algorithm based on GA is presented in [31] for a new specification of the FJSSP that considers the human factor as a multi-objective problem. In [32], a PSO-GA hybrid algorithm is proposed to minimize the workload and the makespan. The minimization of inventory costs and total workflow are studied in [33], which involves a hybrid algorithm between the ABC algorithm and the modified migrating birds optimization (MMBO) algorithm to obtain satisfactory results. Another hybrid method is put forth in [34] based on SA and saving heuristics to minimize the energy consumption of machines, taking into account their deterioration to determine the precise processing time. In [35], the FJSSP

with fuzzy times is solved with a hybrid multi-verse optimization (HMVO) algorithm. A hybrid approach to general scheduling problems is discussed in [36], considering batch dimensions, rework, and shelf-life constraints for defective items. The algorithm applies late acceptance hill-climbing (LAHC) and analytic procedures to accelerate the process. The computational results show the benefits of using hybridization techniques. In [37], a two-stage GA (2SGA) is developed; the first stage is to choose the order of operations and the selection of machines simultaneously, and the second is to include new variants that avoid population stagnation. A hybrid algorithm that combines brain storming optimization (BSO) with LAHC is advanced in [38]; the BSO is adapted to the FJSSP to explore the search space by grouping the solutions, and the exploitation is performed with the LAHC. The hybridization of the human learning optimization (HLO) algorithm and the PSO is analyzed in [39], again applying the HLO for global search and an adaptation of the PSO to refine the local search.

The above-mentioned works suggest that hybrid approaches are still a developing research trend in solving the FJSSP and its variants, for which many of these works use GA in conjunction with another technique for local search.

However, to our knowledge, no work has applied GA operators based on a cellular automata-inspired neighborhood to explore solutions and their hybridization with the RRHC for the refinement of these solutions.

This article proposes a new hybrid technique called GA-RRHC that combines two metaheuristic techniques: the first for global search using genetic algorithm (GA) operators and a neighborhood based on concepts of cellular automata (CA) used mostly on the programming of the order of operations. As a second step, each solution is refined by a local search that applies random-restart hill-climbing (RRHC), in particular, to make the best selection of machines for critical operations, which is more convenient for problems with high flexibility. Restart is used as a simple strategy to avoid premature convergence of solutions.

The contribution of this research lies in the original use of two types of easy-to-implement operators to define a robust hybrid technique that finds satisfactory solutions to instances of the FJSSP for minimizing the processing time of all the jobs (or makespan). The GA-RRHC was implemented in Matlab and is available on Github https://github.com/juanseck/GA-RRHC (accessed on 5 August 2022).

The structure of this article is as follows: Section 2 provides the formal presentation of the FJJSP. Section 3 proposes the new GA-RRHC method explaining the genetic operators used, the CA-inspired neighborhood for the evolution of the population of solutions, and the operation of the RRHC to refine each solution. Section 4 discusses the parameter adjustment of the GA-RRHC, the comparison with six other recently published algorithms in four FJSSP datasets commonly used in the literature, making a statistical analysis based on the non-parametric Friedman test and the ranking by the relative percentage deviation (RPD). Section 5 gives the concluding comments of this manuscript.

## 2. Description of the FJSSP

The flexible job shop scheduling problem (FJSSP) consists of a set $J = \{J_1, J_2, \dots J_n\}$ of $n$ jobs and a set $M = \{M_1, M_2, \dots M_m\}$ with $m$ machines. Each job $J_i$ has $n_i$ operations $O_{J_i} = \{O_{i,1}, O_{i,2}, \dots, O_{i,n_i}\}$. Each operation $O_{i,j}$ can be performed by one machine from a set of feasible machines $M_{i,j} \subseteq M$, for $1 \leq i \leq n$ and $1 \leq j \leq n_i$. The processing time of $O_{i,j}$ in $M_k$ is represented by $p_{i,j,k}$, and $o = \sum n_i$ is the total number of operations.

It is necessary to carry out all operations to complete a job, respecting the operation precedence. The FJSSP has the following conditions: (1) At the start, all jobs and all machines are available. (2) Each operation can only be performed by one machine. (3) A machine cannot be interrupted when processing an operation. (4) Every machine can perform one operation at the same time. (5) Once defined, the order of operations cannot be changed. (6) Machine breakdowns are not considered. (7) Different jobs have no precedence

restrictions of operations. (8) The machines do not depend on each other. (9) The processing time includes the preparation of the machines and the transfer of operations.

One solution of an FJSSP instance includes two parts, a sequence of operations that respects the precedence constraints for each job, and the assignment of a feasible machine to each operation. The objective of this paper is to calculate the sequence of operations and the assignment of machines that minimize the makespan $C_{max}$, the total time needed to complete all jobs, as defined in Equation (1).

$$min\{C_{max}\} \text{ where } C_{max} = max\{C_i\}, \text{ for } 1 \leq i \leq n \tag{1}$$

$C_i$ is the time when all operations in $J_i$ are completed, subject to:

$$1 \leq i \leq n,\ 1 \leq j \leq n_i,\ 1 \leq k \leq m \text{ such that } M_k \in M_{i,j},\ p_{i,j,k} > 0 \tag{2}$$

$$s_{i,j,k} + p_{i,j,k} \leq s_{i,j+1,k} \tag{3}$$

$$X_{i,j,k} = \left\{ \begin{array}{ll} 1, & \text{iff operation } O_{i,j} \text{ is processed on } M_k \\ 0, & \text{otherwise} \end{array} \right\} \tag{4}$$

$$\sum_{k=1}^{m} X_{i,j,k} = 1 \tag{5}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n_i} X_{i,j,k} = 1 \tag{6}$$

In this formulation, the start time of operation $O_{i,j}$ in $M_k$ is $s_{i,j,k}$. The processing time of every operation greater than 0 is reviewed in Equation (2). The precedence between operations of the same job is considered in Equation (3). Equation (4) is an assignment record of one operation to a valid machine; Equation (5) represents that each operation is processed by only one machine. The constraint in Equation (6) guarantees that, at any time, every machine can process only one operation.

Table 1 provides an example of an FJSSP instance with three jobs, two operations per job, and three machines, where all machines can perform all operations. A possible solution to this problem is presented in the Gantt chart in Figure 1.

**Table 1.** Example of an FJSSP with 3 jobs, 3 operations per job, and 3 machines.

| Job | Op. | $M_1$ | $M_2$ | $M_3$ |
|---|---|---|---|---|
| $J_1$ | $O_{1,1}$ | 3 | 4 | 4 |
| | $O_{1,2}$ | 1 | 2 | 1 |
| $J_2$ | $O_{2,1}$ | 2 | 3 | 3 |
| | $O_{2,2}$ | 3 | 3 | 2 |
| $J_3$ | $O_{3,1}$ | 3 | 3 | 3 |
| | $O_{3,2}$ | 2 | 2 | 1 |

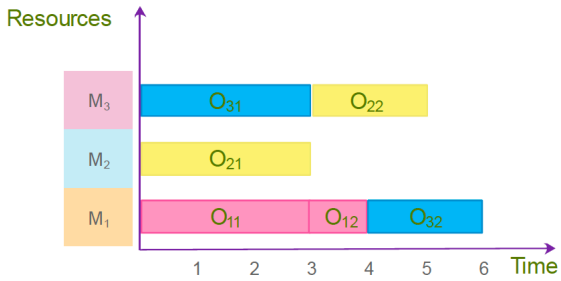


**Figure 1.** Gantt diagram of one possible solution for the FJSSP in Table 1.

## 3. Genetic Algorithm and Random-Restart Hill-Climbing (GA-RRHC)

The idea of using a genetic algorithm with a random-restart hill-climbing (GA-RRHC) algorithm is to propose a method that is easy to understand and implement and simultaneously capable of obtaining competitive results compared with other techniques in the optimization of different FJSSP instances.

The GA-RRHC uses a genetic algorithm as a global search method, mainly using job-based crossover (JBX) and precedence operation crossover (POX), and as mutation, the swapping of two operations and the shift of three-position jobs for the sequences of operations. Two-point crossover and mutation by changing feasible machines at random are used for machine allocation. These operators have been previously used to resolve instances of the FJSSP and have shown good [19] results. A contribution of this work is the method of applying these operators using a neighborhood inspired by cellular automata (CA), where each solution chooses several neighbors. Crossover and mutation are applied for each neighbor, and from all the neighboring solutions, the best one replaces the original. This idea has already been explored in the global–local neighborhood search (GLNSA) algorithm, although using different operators [40].

The local search in the GA-RRHC applies a random-restart hill-climbing (RRHC) algorithm to refine the machine selection for the critical operations of each solution. The RRHC has been used successfully in the FJSSP [30], although not in combination with a GA, which represents another contribution of this work. An advantage of RRHC is its easy implementation, unlike other techniques for discrete problems such as simulated annealing or tabu search [41].

In short, the GA-RRHC generates a random population of solutions. A neighborhood of new solutions produced with genetic operators is taken for each solution, and the best one is chosen. This new solution is refined with the RRHC. The optimization loop repeats until a limit of iterations is met, or a best solution is not calculated after a certain number of repetitions.

### *3.1. Encoding and Decoding Solutions*

Initially, the GA-RRHC generates a random population of $S_n$ solutions called smart-cells. Each smart-cell comprises two sequences, one for the operations ($OS$) and the other for the machine assigned to each operation ($MS$). Both sequences have $o$ elements. The GA-RRHC uses the decoding described in [19].

In the sequence $OS$, each job $J_i$ appears $n_i$ times (a permutation with repetitions). The sequence $MS$ has $o$ elements and is divided into $n$ parts. The $i$th part holds the machines selected to process the job $J_i$ and has as many elements as $n_i$. Figure 2 depicts the codification of the solution in the Gantt diagram of Figure 1.

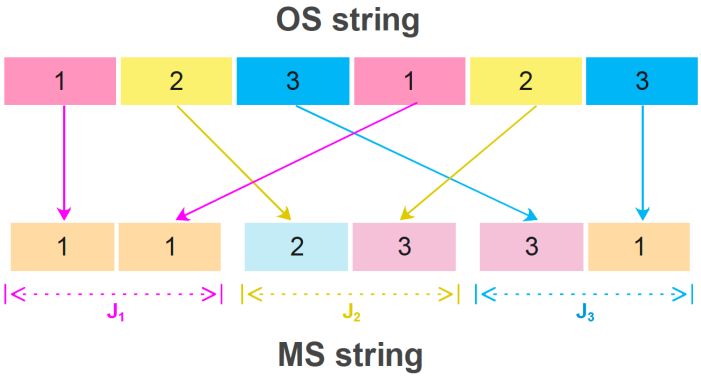


**Figure 2.** Coding of the solution in the Gantt diagram of Figure 1 in sequences $OS$ and $MS$.

To decode the solution, $OS$ is read from left to right. The $j$th occurrence of $J_i$ signifies that operation $O_{i,j}$ must be processed. This encoding allows that any permutation with

repetitions represents a rightful sequence $OS$. For each part $i$ of $MS$, the $j$th value represents the machine processing $O_{i,j}$.

### *3.2. Qualitative Description of the GA*

Each iteration of the GA used in this work involves a selection stage to refine the population of smart-cells, favoring those with lower makespan. Inspired by the CA neighborhood concept [42], for each smart-cell, a neighborhood of new solutions is produced with different crossover and mutation operators. The one with the lowest makespan is chosen as the new smart-cell. The operators used for each part of the GA are described below.

#### 3.2.1. Population Selection

Two types of selection are used in this stage: elitism and tournament. Elitism selects a proportion $E_p$ of the best smart-cells for the next iteration without change, guaranteeing that their information remains available to improve the rest of the population. Genetic operators and RRHC will not be applied in elite smart-cells. A tournament selection is used to select the rest of the smart-cells. Random pairs of smart-cells are chosen, and for each pair, the smart-cell with the lowest makespan is selected for the next iteration.

#### 3.2.2. Crossover Operators

Smart-cell crossover uses two operators for $OS$ sequences, each type of crossover is applied with 50% probability. The first is the precedence operator (POX), where the set of jobs is divided into two random subsets $J_A$ and $J_B$ such that $J_A \cup J_B = J$ and $J_A \cap J_B = \emptyset$. For two sequences $OS_1$ and $OS_2$, two new sequences $OS'_1$ and $OS'_2$ are obtained. The operations of jobs $J_A$ are placed in $OS'_1$ in the same order as in $OS_1$. The operations of $J_B$ fill the empty positions of $OS'_1$, keeping the order from left to right (seriatim) in which they appear in $OS_2$. The analogous process is carried out to form $OS'_2$ by first taking the operations of $J_A$ at the same positions as $OS_2$. The empty spaces of $OS'_2$ are filled with the operations of $J_B$ in $OS_1$ seriatim. Figure 3 exemplifies the POX crossover with three jobs and six operations.

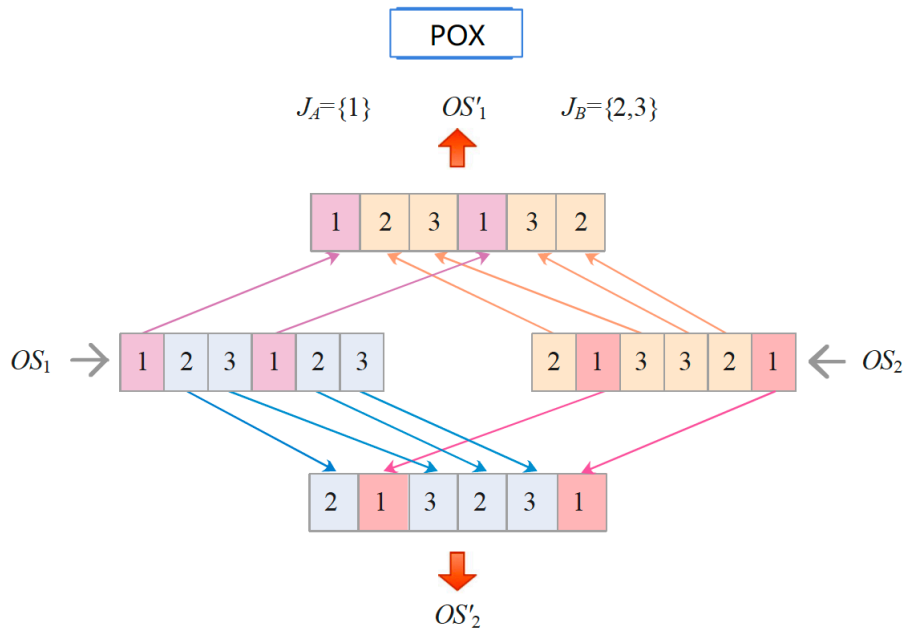


**Figure 3.** Example of a precedence operation crossover (POX).

The second operator is the job-based crossover (JBX), which also defines the subsets $J_A$ and $J_B$. From two sequences $OS_1$ and $OS_2$, $OS'_1$ is obtained in the same way. The difference lies in the specification of $OS'_2$, first taking the operations of $J_B$ in the same positions as $OS_2$.

Next, the empty spaces of $OS'_2$ are filled with the operations of $J_A$ in $OS_1$ seriatim. Figure 4 presents an example of a JBX crossover with three jobs and six operations.

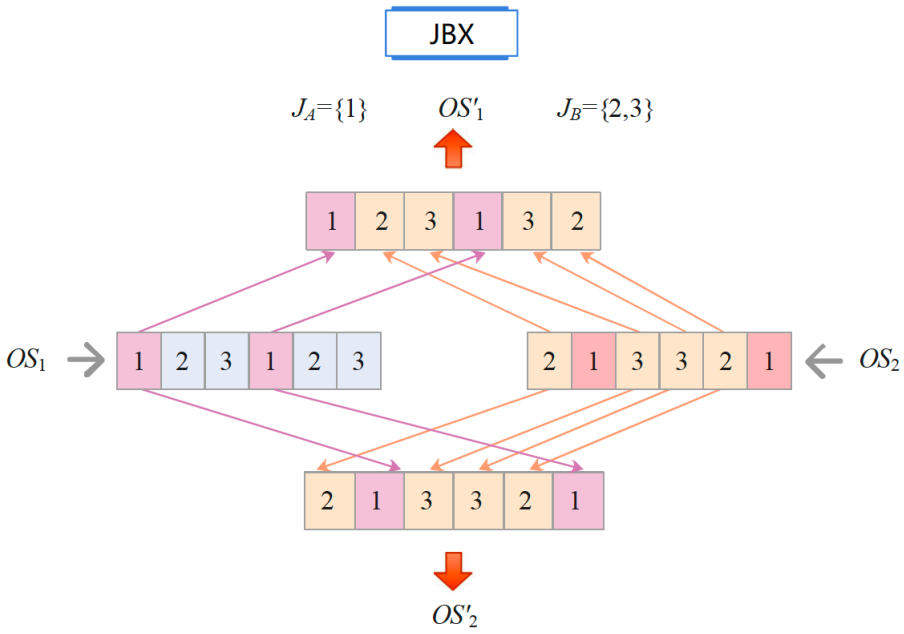


**Figure 4.** Example of a job-based crossover (JBX).

A two-point crossover is used for the sequences $MS$. For two sequences $MS_1$ and $MS_2$, two random positions $1 < a_1 < a_2 < o$ are chosen. A new sequence $MS'_1$ is obtained by taking the elements of $MS_1$ at positions $[1, a_1 - 1]$ and $[a_2 + 1, o]$ and from $MS_2$ at positions $[a1, a2]$. Similarly, a new sequence $MS'_2$ is formed by exchanging the roles of $MS_1$ and $MS_2$. Figure 5 presents a two-point cross for three machines and six operations.

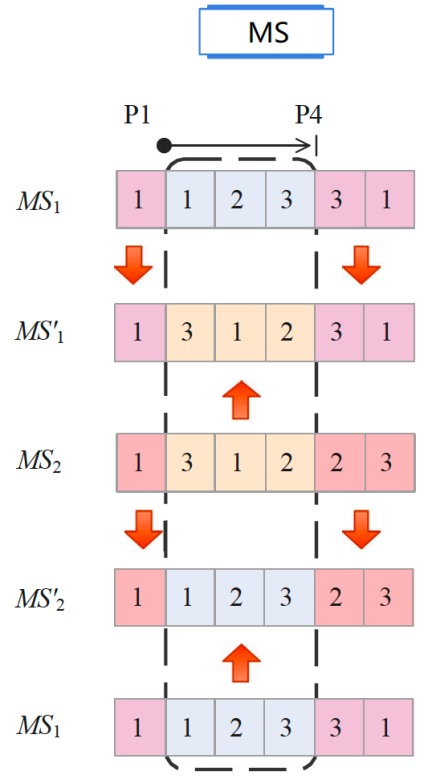


**Figure 5.** A two-point crossover example.

3.2.3. Mutation Operators

Smart-cells mutation uses two operators over the $OS$ sequences, and each type of mutation is used with 50% of probability. The first is swap mutation, where two positions of $OS$ are selected and their elements swapped to obtain $OS'$. An example is in Figure 6.

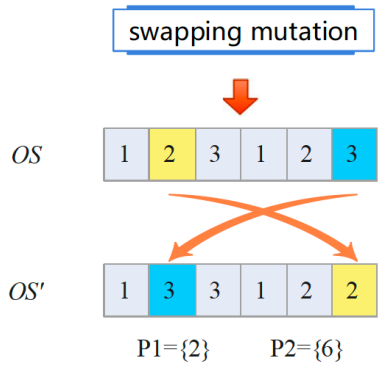


**Figure 6.** Example of swapping mutation.

The second mutation is the random change of positions for operations of three different jobs. Three positions of $OS$ belonging to different jobs are selected and their positions randomly swapped to obtain $OS'$. Three operations from different jobs are chosen since exchanging their positions in $OS$ would not generate a new solution if the operations were from the same job. Moreover, a more significant perturbation is achieved by selecting three operations. This is suitable for the exploration stage and to escape local minima, especially in cases with a more significant number of jobs and operations. This type of mutation was proposed by [19], obtaining good results. An example is depicted in Figure 7.

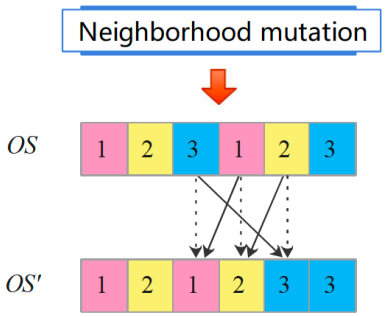


**Figure 7.** Example of three job mutation.

For the sequence $MS$, a mutation of assigned machines is applied; $\lfloor o/2 \rfloor$ random positions are chosen, and for those positions, feasible random machines, different from the initial selection, are chosen. One example is presented in Figure 8.

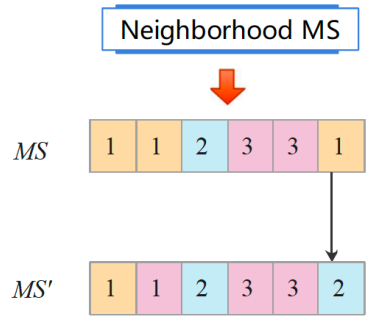


**Figure 8.** Example of machine mutation.

#### 3.2.4. CA-Type Neighborhood to Apply Genetic Operators

According to Eiben and Smith [43], an evolutionary algorithm is an optimization strategy modelled around the evolution of different biological systems. If a system shows dynamic behaviors applicable in solving complex problems, it can be a source of inspiration to define new evolutionary algorithms, regardless of whether these systems are natural or artificial.

CAs are elemental discrete dynamical systems capable of generating complex global behaviors [42,44]. The simplest model, known as elementary CA, is discrete in states and time; it consists of a linear array of cells with an initial assigned state, which can be a number or color. Each cell keeps or changes its state depending on its present state and those of its neighbors on either side, having a whole neighborhood of three cells. This process is applied synchronously to update the state of all cells.

This CA-type neighborhood has been successfully applied in different optimization algorithms [45–48]. However, to our knowledge, this type of neighborhood has not yet been applied with genetic operators for the FJSSP. In this work, to explore the solution space, each smart-cell will generate $l$ new neighboring solutions using crossover and mutation. The best of these $l$ solutions (with the smallest makespan) is selected to update to the original smart-cell. Figure 9 shows the CA-type neighborhood implemented in the proposed algorithm.

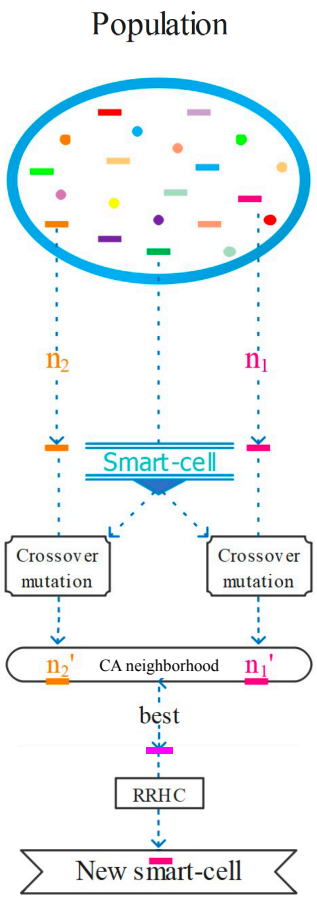


**Figure 9.** CA-type neighborhood used in the GA-RRHC. Colors represent the selection and modification of different smart-cells.

The previous genetic operators perform the global search mainly to improve the *OS* sequence of each smart-cell. Next, the local search method for optimizing the *MS* sequences of each smart-cell is described.

### *3.3. Random-Restart Hill-Climbing (RRHC)*

A local search method is intended to exploit the information of the smart-cells to decrease the value of their makespan. The general idea is to start from each smart-cell and make small changes to improve the makespan. The metaheuristic used is the random-restart hill-climbing (RRHC) [30]. The RRHC can restart the search from a solution with a makespan worse than the original one after a given number of steps to escape local minima. The whole process ends after a fixed number of steps.

The critical operations of the smart-cell are first detected to apply the RRHC. These operations define the value of the makespan, and these operations are linked by job or by a machine whose sum of processing times from the beginning to the end gives the makespan as a result, with no idle times between these operations [18].

A record of the previous task of each operation is kept when calculating the makespan to know which are the critical operations of a smart-cell, where the initial operations of each job do not have a previous operation. This record allows a fast computation of the critical operations by simply taking one of the last operations with a completion time identical to the makespan. Subsequently, previous operations on the same machine and at the same job are analyzed, taking that with end time equal to the start time of the present operation. A random pick is made if the same completion time is held by both previous operations. The procedure is repeated until an operation with no preceding operation is reached. Figure 10 presents the critical path of the solution represented in the Gantt diagram of Figure 1.

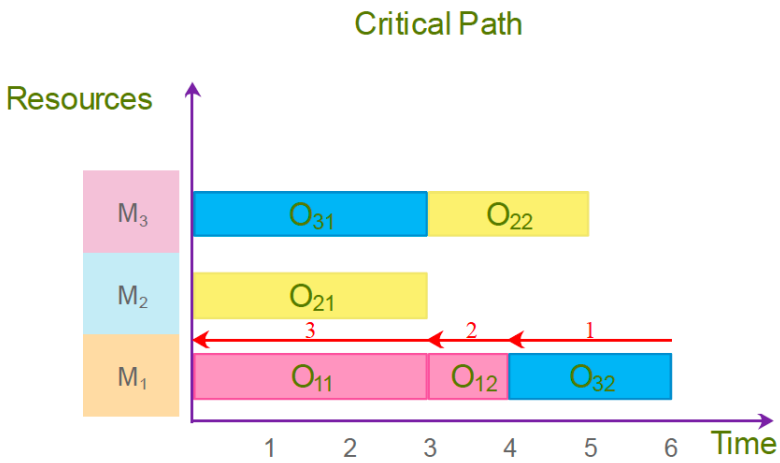


**Figure 10.** Critical path of the solution described in Figure 1.

From the set of critical operations, one is taken at random, and a different feasible machine is chosen to have a new sequence $MS'$. Additionally, another critical operation with probability $\alpha_c$ is selected, and is swapped with any other operation in $OS$ to have a new sequence $OS'$ and generate a different solution. Figure 11 shows a makespan improvement changing the machine assignment of a critical operation.

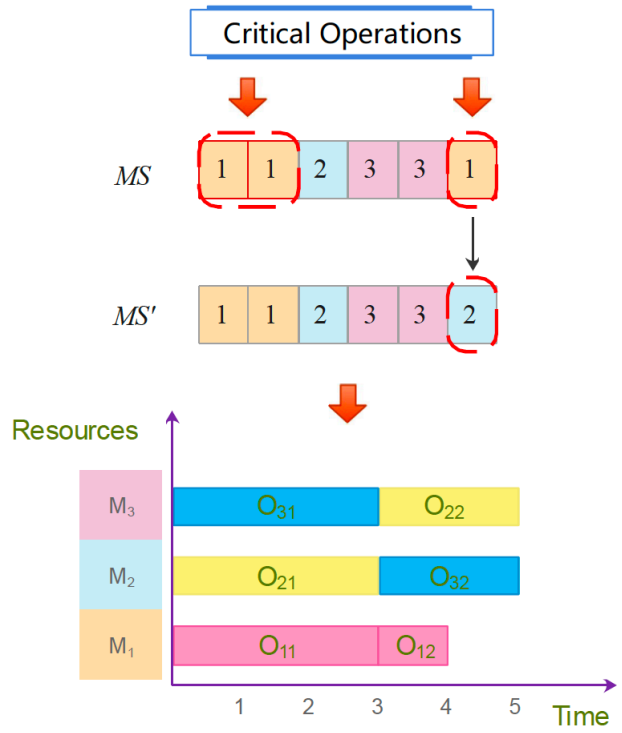


**Figure 11.** New solution obtained by selecting another random machine of a critical operation.

The RRHC is applied for $H_n$ iterations, having a pile of $H_r < H_n$ new solutions generated through the described process. If one of these solutions improves the makespan, this solution replaces the smart-cell, emptying the pile. If the makespan of the original smart-cell has not been improved after $H_r$ iterations, a new random solution is taken from the pile to restart the climbing. Since the RRHC focuses on improving machine allocation, it works best for instances with high flexibility.

### *3.4. Integration of the GA-RRHC Algorithm*

Algorithm 1 depicts the GA-RRHC pseudocode. Figure 12 illustrates the flowchart of the proposed method. After the GA-RRHC parameters are defined, the smart-cell population is generated, evaluated, and selected. GA operators are applied in a CA-like neighborhood for a global search to update every smart-cell. Next, each smart-cell is improved by a local search performed by the RRHC. Finally, the best smart-cell is returned.

**Algorithm 1:** Pseudocode of the GA-RRHC

**Result:** Smart-cell with minimum makespan
Set GA-RRHC parameters;
Initialize $S_n$ initial smart-cells at random;
Calculate the makespan of each smart-cell ;
**do**
  Apply elitism and tournament to obtain a refined population;
  Using GA crossover and mutation operators, generate a CA-like neighborhood for each solution; the best neighbor replaces the smart-cell;
  Improve the machine assignment of critical operations in each smart-cell applying RRHC;
**while** *(Stagnation number* $< G_b$ *or iteration number* $< G_n$*)*;
Return the best smart-cell;

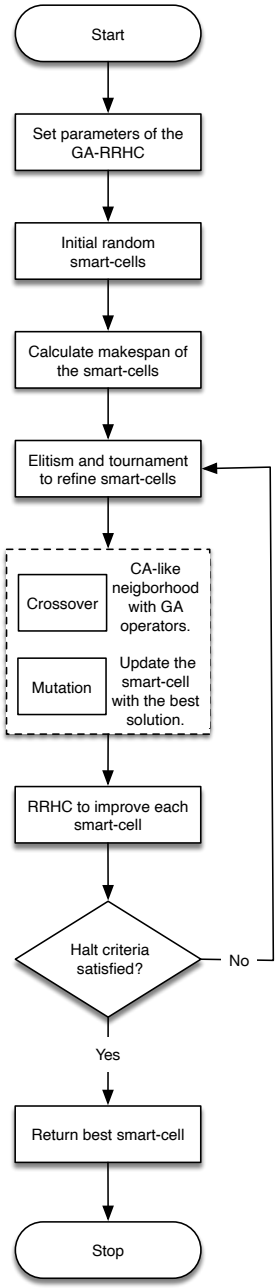


**Figure 12.** GA-RRHC flowchart.

## 4. Results of Experiments

The GA-RRHC was coded in Matlab R2015a (TM) on an Intel Xeon W machine with 128 GB of RAM and running at 2.3 GHz. The source code is available on Github https://github.com/juanseck/GA-RRHC (accessed on 5 August 2022). Four datasets were taken to test the effectiveness of the GA-RRHC. These datasets have been widely used in the specialized literature, with instances having different degrees of flexibility. A flexibility rate between 0 and 1 is specified as $\beta$ = (flexibility average/number of machines). A high value of $\beta$ means that the same operation can be processed by more machines.

The first experiment takes the Kacem dataset [49], with five instances of which four have a value of $\beta = 1$ (full flexibility). The BRdata dataset is used for the second experiment and consists of 10 instances, from 10 to 20 jobs and 4 to 15 machines, with partial flexibility ($\beta \leq 0.35$) [17]. The third and fourth datasets are the Rdata and Vdata datasets, with each having 43 problems going from 6 to 30 jobs and 6 to 15 machines. The Rdata set has a maximum value of $\beta \leq 0.4$, while all instances of the *Vdata* set have a rate of $\beta = 0.5$,

which means that half of the machines can perform each operation [50]. These datasets are available at https:/people.idsia.ch/~monaldo/fjsp.html (accessed on 5 August 2022).

### 4.1. GA-RRHC Parameter Tuning

The GA-RRHC has nine parameters that control its operation. The total number of iterations is $G_n$; $G_b$ is the limit of stagnation iterations, $S_n$ is the number of smart-cells, $E_p$ is the proportion of elite smart-cells, and $l$ is the number of neighbors of each smart-cell generated by the genetic operators. Further, $\alpha_m$ is the probability of mutation of a solution, $H_n$ is the number of RRHC iterations, $H_r$ are the iterations to restart the RRHC, and $\alpha_c$ is the probability of moving a critical operation in the RRHC.

For the first three parameters ($G_n = 250, G_b = 50, S_n = 100$), the values used in [19,38] were employed as a reference since these works obtained good results in minimizing the makespan. These parameter values are typically used in specialized publications and are comparable to those utilized by methods employed in the following sections to compare the performance of the GA-RRHC.

To adjust the other parameters, different levels of each parameter were taken to minimize the *mt*20 instance of the Vdata dataset and the best combination of parameters was chosen.

For $E_p$, the values 0.02 and 0.04 were tested, the parameter $l$ was tested between 2 and 3, and the values 0.1 and 0.2 were tested for $\alpha_m$. For the RRHC, the values between 80 and 100 and between 30 and 40 were taken to tune $H_n$ and $H_r$ respectively. For $\alpha_c$, the values 0.025 and 0.05 were proved. In this way, 64 different combinations of parameters were taken; for each combination, 30 independent runs were performed, selecting the set of parameters with the least average makespan. Table 2 shows the GA-RRHC parameters to analyze its results in the rest of the instances.

**Table 2.** Parameters selected for the execution of the GA-RRHC.

| | | |
|---|---|---|
| $G_n$ | total iterations of the algorithm | 250 |
| $G_b$ | limit of stagnation iterations | 50 |
| $S_n$ | number of smart-cells | 100 |
| $E_p$ | proportion of elite smart-cells | 0.02 |
| $l$ | neighbors of each smart-cell | 3 |
| $\alpha_m$ | mutation probability | 0.1 |
| $H_n$ | iterations of the RRHC | 100 |
| $H_r$ | iterations to restart the RRHC | 30 |
| $\alpha_c$ | probability of moving a critical operation in the RRHC | 0.05 |

### 4.2. Comparison with Other Methods

Six algorithms published between 2016 and 2021 were used to compare the GA-RRHC performance. These algorithms include the global–local neighborhood search algorithm (GLNSA) [40], the hybrid algorithm (HA) [19], the greedy randomized adaptive search procedure (GRASP) [51], the hybrid brain storm optimization and late acceptance hill-climbing (HBSO-LAHC) [38], the improved Jaya algorithm (IJA) [52], and the two-level PSO (TlPSO) [53].

Comparing the execution times of the different algorithms is not appropriate since these were implemented in different architectures and languages and with different programming skills. That is why this work compares these algorithms using their computational complexity concerning the total operation number ($\mathcal{O}(o)$). To use a standard notation for all algorithms, the number of solutions will be represented by $X$, the iteration number by $G_n$, the number of local search iterations by $H_n$, and $m$ is the number of machines. The CA-inspired neighborhood algorithms (GA-RRHC and GLNSA) satisfy that $X = S_n l$, where $l$ is the number of neighbors in the global search.

The GLNSA uses elitism to select solutions and generates $l$ neighbors with insertion, swapping, path-relinking operators, a machine mutation, and a tabu search on $S_n$ solutions.

The GRASP calculates a Gantt chart and then applies a greedy local search which is quadratic with respect to $o$. The HA applies four genetic operations for each solution and then a TS to select different machines for the critical operations. The HBSO-LAHC uses a clustering of solutions, which requires calculating the distance between them, and then applies four strategies and three neighborhoods for each solution, with one of them on critical operations. For the local search, a hill-climbing algorithm with late acceptance for $H_n$ steps is applied in each solution. The IJA is a modified Java algorithm that applies three exchange procedures to each solution and a local search with the random exchange of blocks of critical operations. Finally, the TlPSO uses two modifications of the PSO: the first is applied to improve the order of operations, and in each iteration of the first PSO, another PSO is used for the allocation of machines.

Table 3 presents the algorithms ordered from least to most complex, with $S_n < X < H_n$ since $H_n$ usually has a high value concerning obtaining a better local search. This table shows that the GA-RRHC has computational complexity comparable to recently proposed state-of-the-art algorithms.

**Table 3.** Computational complexity of the methods used in the experiments.

| Method | Complexity | Rank |
|---|---|---|
| TlPSO | $\mathcal{O}(o(G_n(2X + G_n2X)))$ | 1 |
| GA-RRHC | $\mathcal{O}(o(G_n(S_n + X + S_nH_nm)))$ | 2 |
| GLNSA | $\mathcal{O}(o(G_n(S_n + X + S_nH_nm)))$ | 2 |
| IJA | $\mathcal{O}(o(G_n(3X + XH_nm))$ | 3 |
| HA | $\mathcal{O}(o(G_n(4X + XH_nm)))$ | 4 |
| HBSO-LAHC | $\mathcal{O}(o(G_n(4X + XH_nm)))$ | 4 |
| GRASP | $\mathcal{O}(o^2(G_n * H_n))$ | 5 |

This analysis only considers the complexity inherent in modifying the order of operations and the machine assignment in a solution. The computational complexity for calculating the makespan or tracking the critical operations is not considered since all the algorithms use them. Thus, the analysis only focuses on the computational processes that make each method different.

### *4.3. Kacem Dataset*

In every experiment described in this work, the selected methods were tested with the same datasets as those exposed in their references, making the presented analysis reliable. For each instance, 30 independent runs were executed and the smallest makespan obtained was taken. The best values reported in their respective papers were taken for the other algorithms.

For the Kacem dataset, the HBSO-LAHC does not report results, and only the GLNSA and IJA report complete results. Table 4 presents the outcomes of the GA-RRHC, where $n$ indicates the number of jobs, $m$ the number of machines, and $\beta$ the flexibility rate of each instance. These problems have total flexibility and start with fewer jobs and machines until they reach a high dimensionality.

**Table 4.** Kacem dataset results.

| Instance | n × m | $\beta$ | GA-RRHC | GLNSA | GRASP | HA | IJA | TlPSO |
|---|---|---|---|---|---|---|---|---|
| K1 | 4 × 5 | 1. | 11 | 11 | – | – | 11 | 11 |
| K2 | 8 × 8 | 0.81 | 14 | 14 | 14 | 14 | 14 | 14 |
| K3 | 10 × 7 | 1 | 11 | 11 | – | – | 11 | – |
| K4 | 10 × 10 | 1 | 7 | 7 | 7 | 7 | 7 | 7 |
| K5 | 15 × 10 | 1 | 11 | 11 | 11 | 11 | 11 | – |

Figure 13 presents the number of best results obtained for each algorithm in this dataset. Table 4 exposes that the GA-RRHC calculates the best makespan values besides the GLNSA and IJA, confirming the satisfactory operation of the GA-RRHC for instances with high flexibility.

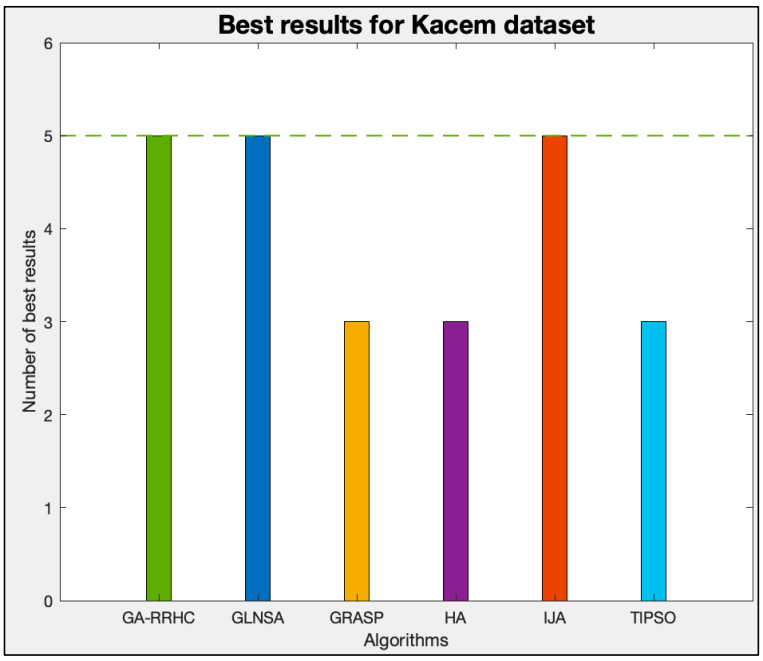


**Figure 13.** Number of best results per algorithm for the Kacem dataset.

### *4.4. Brandimarte Dataset*

For the rest of the experiments, the relative percentage deviation (*RPD*) and Friedman's non-parametric test were employed to compare the GA-RRHC with the other methods [54]. The *RPD* is defined in Equation (7); *BOV* is the best-obtained makespan by each algorithm, and *BKV* is the best-known makespan for each problem.

$$RPD = \frac{BOV - BKV}{BOV} \times 100 \tag{7}$$

Table 5 provides the results of the GA-RRHC compared with the other methods for the BRdata dataset. In each case, the smallest obtained makespan is marked with *.

**Table 5.** BRdata dataset results.

| Instance | n × m | $\beta$ | BKV | BSO-LAHC | GA-RRHC | GLNSA | GRASP | HA | IJA | TIPSO |
|---|---|---|---|---|---|---|---|---|---|---|
| MK01 | 10 × 6 | 0.2 | 36 | 40 * | 40 * | 40 * | 40 * | 40 * | 40 * | 40 * |
| MK02 | 10 × 6 | 0.35 | 24 | 26 * | 26 * | 26 * | 26 * | 26 * | 27 | 26 * |
| MK03 | 15 × 8 | 0.3 | 204 | 204 * | 204 * | 204 * | 204 * | 204 * | 204 * | 204 * |
| MK04 | 15 × 8 | 0.2 | 48 | 60 * | 60 * | 60 * | 60 * | 60 * | 60 * | 60 * |
| MK05 | 15 × 4 | 0.15 | 168 | 173 | 172 * | 173 | 172 * | 172 * | 172 * | 173 |
| MK06 | 10 × 15 | 0.3 | 33 | 61 | 58 | 58 | 64 | 57 * | 57 * | 60 |
| MK07 | 20 × 5 | 0.3 | 133 | 141 | 139 * | 139 * | 139 * | 139 * | 139 * | 139 * |
| MK08 | 20 × 10 | 0.15 | 523 | 523 * | 523 * | 523 * | 523 * | 523 * | 523 * | 523 * |
| MK09 | 20 × 10 | 0.3 | 299 | 307 * | 307 * | 307 * | 307 * | 307 * | 307 * | 307 * |
| MK10 | 20 × 15 | 0.2 | 165 | 204 | 198 | 205 | 205 | 197 * | 197 * | 205 |

It can be seen in Table 5 that the GA-RRHC calculates the least makespan in eight cases, behind the HA and IJA, and with the same optimal results as the GRASP. Figure 14 shows the number of best results obtained for each algorithm in this dataset.

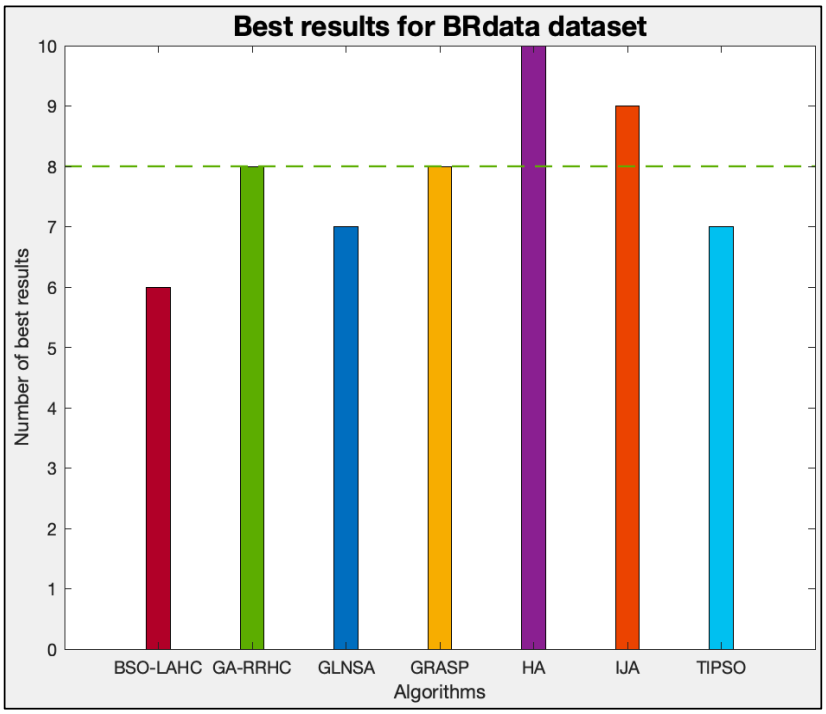


**Figure 14.** Number of best results per algorithm for the BRdata dataset.

Table 6 shows the ranking of each method based on the average *RPD*, as well as the value $p$ of the non-parametric Friedman pairwise test, comparing the GA-RRHC with each of the other algorithms. The GA-RRHC secured the second place among the compared algorithms, obtaining a value of $p < 0.05$ with the BSO-LAHC and GRASP.

**Table 6.** Algorithm ranking and Friedman test value for BRdata dataset.

| **Algorithm:** | **BSO-LAHC** | **GA-RRHC** | **GLNSA** | **GRASP** | **HA** | **IJA** | **TlPSO** |
|---|---|---|---|---|---|---|---|
| Average *RPD*: | 11.3881 | 10.6710 | 11.0121 | 11.4890 | 10.5289 | 11.2665 | 11.2017 |
| Rank: | 6 | 2 | 3 | 7 | 1 | 5 | 4 |
| *p*-value: | 0.0455 | ~ | 0.1573 | 0.0435 | 0.1643 | 0.0803 | 0.0833 |

Figure 15 gives the signed difference between the average RPD of the GA-RRHC against the other methods in pairs. A negative difference means an inferior performance in contrast to the GA-RRHC.

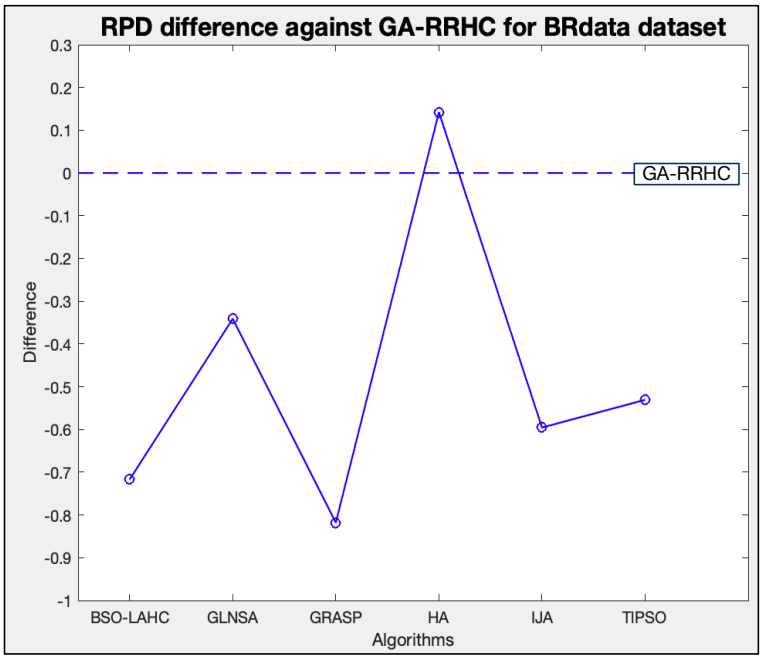


**Figure 15.** Comparison of the RPD difference per algorithm for the BRdata dataset.

This analysis indicates that the GA-RRHC is statistically competitive with the other four best algorithms for optimizing the BRdata dataset.

### *4.5. Rdata Dataset*

This experiment takes the 43 instances of the Rdata dataset, with a flexibility rate $\beta$ ranging from 0.13 to 0.4. The results generated by the GA-RRHC are compared in Table 7 with those obtained by the GLNSA, HA, and IJA, which are the methods that report results for this dataset. Figure 16 depicts the number of best results obtained for each algorithm in this dataset, where the GA-RRHC calculated the 23 best values, behind the HA and IJA.

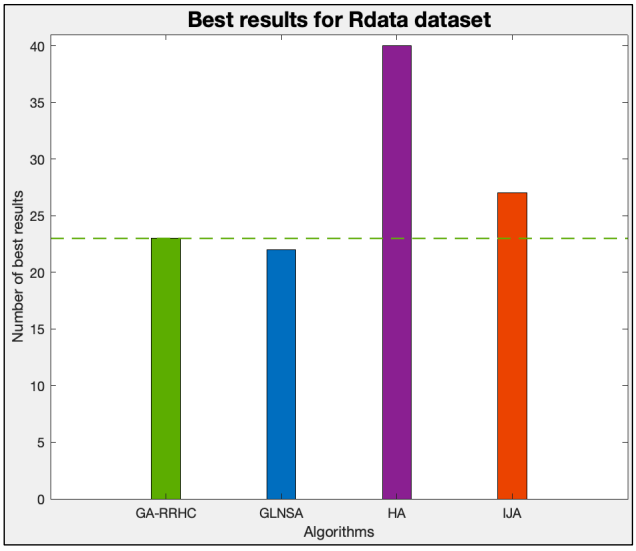


**Figure 16.** Number of best results per algorithm for the Rdata dataset.

Table 8 sorts each algorithm with its average *RPD* and the comparative Friedman test. From the results, it can be noted that the GA-RRHC ranked third overall, as might be expected in instances with low flexibility. The statistical analysis shows no significant difference with the IJA.

**Table 7.** Rdata dataset results, best makespan values are marked with *.

| Instance | n × m | $\beta$ | BKV | GA-RRHC | GLNSA | HA | IJA |
|---|---|---|---|---|---|---|---|
| mt06 | 6 × 6 | 0.33 | 47 | 47 * | 47 * | 47 * | 47 * |
| mt10 | 10 × 10 | 0.2 | 686 | 686 * | 686 * | 686 * | 686 * |
| mt20 | 20 × 5 | 0.4 | 1022 | 1022 * | 1022 * | 1024 | 1024 |
| la01 | 10 × 5 | 0.4 | 570 | 571 | 571 | 570 * | 571 |
| la02 | 10 × 5 | 0.4 | 529 | 530 * | 530 * | 530 * | 530 * |
| la03 | 10 × 5 | 0.4 | 477 | 477 * | 477 * | 477 * | 477 * |
| la04 | 10 × 5 | 0.4 | 502 | 502 * | 502 * | 502 * | 502 * |
| la05 | 10 × 5 | 0.4 | 457 | 457 * | 457 * | 457 * | 457 * |
| la06 | 15 × 5 | 0.4 | 799 | 799 * | 799 * | 799 * | 799 * |
| la07 | 15 × 5 | 0.4 | 749 | 749 * | 749 * | 749 * | 749 * |
| la08 | 15 × 5 | 0.4 | 765 | 765 * | 765 * | 765 * | 765 * |
| la09 | 15 × 5 | 0.4 | 853 | 853 * | 853 * | 853 * | 853 * |
| la10 | 15 × 5 | 0.4 | 804 | 804 * | 804 * | 804 * | 804 * |
| la11 | 20 × 5 | 0.4 | 1071 | 1071 * | 1071 * | 1071 * | 1071 * |
| la12 | 20 × 5 | 0.4 | 936 | 936 * | 936 * | 936 * | 936 * |
| la13 | 20 × 5 | 0.4 | 1038 | 1038 * | 1038 * | 1038 * | 1038 * |
| la14 | 20 × 5 | 0.4 | 1070 | 1070 * | 1070 * | 1070 * | 1070 * |
| la15 | 20 × 5 | 0.4 | 1089 | 1089 * | 1089 * | 1090 | 1090 |
| la16 | 10 × 10 | 0.2 | 717 | 717 * | 717 * | 717 * | 717 * |
| la17 | 10 × 10 | 0.2 | 646 | 646 * | 646 * | 646 * | 646 * |
| la18 | 10 × 10 | 0.2 | 666 | 666 * | 666 * | 666 * | 666 * |
| la19 | 10 × 10 | 0.2 | 647 | 700 * | 700 * | 700 | 702 |
| la20 | 10 × 10 | 0.2 | 756 | 756 * | 756 * | 756 * | 760 |
| la21 | 15 × 10 | 0.2 | 808 | 850 | 852 | 835 * | 854 |
| la22 | 15 × 10 | 0.2 | 737 | 770 | 774 | 760 * | 760 * |
| la23 | 15 × 10 | 0.2 | 816 | 850 | 854 | 840 * | 852 |
| la24 | 15 × 10 | 0.2 | 775 | 810 | 826 | 806 * | 806 * |
| la25 | 15 × 10 | 0.2 | 752 | 800 | 803 | 789 * | 803 |
| la26 | 20 × 10 | 0.2 | 1056 | 1070 | 1075 | 1061 * | 1061 * |
| la27 | 20 × 10 | 0.2 | 1085 | 1100 | 1109 | 1089 * | 1109 |
| la28 | 20 × 10 | 0.2 | 1075 | 1090 | 1096 | 1079 * | 1081 |
| la29 | 20 × 10 | 0.2 | 993 | 999 | 1008 | 997 * | 997 * |
| la30 | 20 × 10 | 0.2 | 1068 | 1088 | 1096 | 1078 * | 1078 * |
| la31 | 30 × 10 | 0.2 | 1520 | 1521 * | 1527 | 1521 * | 1521 * |
| la32 | 30 × 10 | 0.2 | 1657 | 1667 | 1667 | 1659 * | 1659 * |
| la33 | 30 × 10 | 0.2 | 1497 | 1500 | 1504 | 1499 * | 1499 * |
| la34 | 30 × 10 | 0.2 | 1535 | 1539 | 1540 | 1536 * | 1536 * |
| la35 | 30 × 10 | 0.2 | 1549 | 1553 | 1555 | 1550 * | 1555 |
| la36 | 15 × 15 | 0.13 | 1016 | 1050 | 1053 | 1028 * | 1050 |
| la37 | 15 × 15 | 0.13 | 989 | 1092 | 1093 | 1074 * | 1092 |
| la38 | 15 × 15 | 0.13 | 943 | 995 | 999 | 960 * | 995 |
| la39 | 15 × 15 | 0.13 | 966 | 1030 | 1034 | 1024 * | 1031 |
| la40 | 15 × 15 | 0.13 | 955 | 998 | 997 | 970 * | 993 |

**Table 8.** Algorithm ranking and Friedman test value for Rdata dataset.

| Algorithm: | GA-RRHC | GLNSA | HA | IJA |
|---|---|---|---|---|
| Average *RPD*: | 1.5761 | 1.7768 | 1.0872 | 1.5155 |
| Rank: | 3 | 4 | 1 | 2 |
| *p*-value: | ~ | 0.0001 | 0.0001 | 0.9195 |

Figure 17 shows the difference between the GA-RRHC and the other methods for the average RPD. A negative value means again an inferior performance compared to the GA-RRHC.

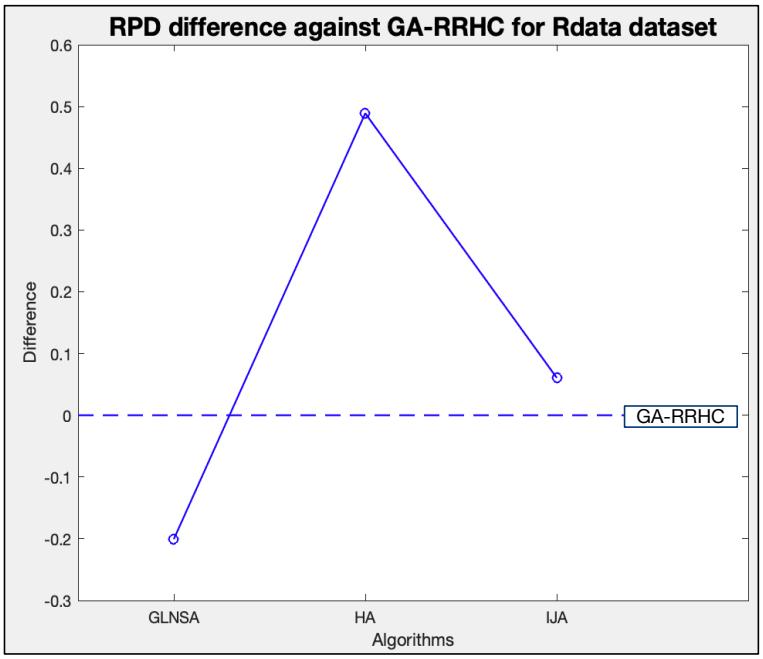


**Figure 17.** Comparison of the RPD difference per algorithm for the Rdata dataset.

This experiment verifies that the GA-RRHC performs comparably to the IJA for problems with low flexibility and outperforms the GLNSA.

### *4.6. Vdata Dataset*

This experiment takes the 43 instances of the Vdata dataset, all of which have a flexibility rate $\beta = 0.5$. The results generated by the GA-RRHC have been compared again with those obtained by the GLNSA, HA, and IJA (Table 9).

**Table 9.** Vdata dataset results, best makespan values are marked with *.

| Instance | n × m | $\beta$ | BKV | GA-RRHC | GLNSA | HA | IJA |
|---|---|---|---|---|---|---|---|
| mt06 | 6 × 6 | 0.5 | 47 | 47 * | 47 * | 47 * | 47 * |
| mt10 | 10 × 10 | 0.5 | 655 | 655 * | 655 * | 655 * | 655 * |
| mt20 | 20 × 5 | 0.5 | 1022 | 1022 * | 1022 * | 1022 * | 1024 |
| la01 | 10 × 5 | 0.5 | 570 | 570 * | 570 * | 570 * | 571 |
| la02 | 10 × 5 | 0.5 | 529 | 529 * | 529 * | 529 * | 529 * |
| la03 | 10 × 5 | 0.5 | 477 | 477 * | 477 * | 477 * | 477 * |
| la04 | 10 × 5 | 0.5 | 502 | 502 * | 502 * | 502 * | 502 * |
| la05 | 10 × 5 | 0.5 | 457 | 457 * | 457 * | 457 * | 457 * |
| la06 | 15 × 5 | 0.5 | 799 | 799 * | 799 * | 799 * | 799 * |
| la07 | 15 × 5 | 0.5 | 749 | 749 * | 749 * | 749 * | 749 * |
| la08 | 15 × 5 | 0.5 | 765 | 765 * | 765 * | 765 * | 765 * |
| la09 | 15 × 5 | 0.5 | 853 | 853 * | 853 * | 853 * | 853 * |
| la10 | 15 × 5 | 0.5 | 804 | 804 * | 804 * | 804 * | 804 * |
| la11 | 20 × 5 | 0.5 | 1071 | 1071 * | 1071 * | 1071 * | 1071 * |
| la12 | 20 × 5 | 0.5 | 936 | 936 * | 936 * | 936 * | 936 * |

**Table 9.** *Cont.*

| Instance | n × m | $\beta$ | BKV | GA-RRHC | GLNSA | HA | IJA |
|---|---|---|---|---|---|---|---|
| la13 | 20 × 5 | 0.5 | 1038 | 1038 * | 1038 * | 1038 * | 1038 * |
| la14 | 20 × 5 | 0.5 | 1070 | 1070 * | 1070 * | 1070 * | 1070 * |
| la15 | 20 × 5 | 0.5 | 1089 | 1089 * | 1089 * | 1089 * | 1089 * |
| la16 | 10 × 10 | 0.5 | 717 | 717 * | 717 * | 717 * | 717 * |
| la17 | 10 × 10 | 0.5 | 646 | 646 * | 646 * | 646 * | 646 * |
| la18 | 10 × 10 | 0.5 | 663 | 663 * | 663 * | 663 * | 665 |
| la19 | 10 × 10 | 0.5 | 617 | 617 * | 617 * | 617 * | 618 |
| la20 | 10 × 10 | 0.5 | 756 | 756 * | 756 * | 756 * | 758 |
| la21 | 15 × 10 | 0.5 | 800 | 804 * | 806 | 804 * | 806 |
| la22 | 15 × 10 | 0.5 | 733 | 737 * | 737 * | 738 | 738 |
| la23 | 15 × 10 | 0.5 | 809 | 813 * | 813 * | 813 * | 813 * |
| la24 | 15 × 10 | 0.5 | 773 | 777 * | 777 | 777 * | 778 |
| la25 | 15 × 10 | 0.5 | 751 | 754 * | 754 | 754 * | 754 * |
| la26 | 20 × 10 | 0.5 | 1052 | 1053 * | 1054 | 1053 * | 1054 |
| la27 | 20 × 10 | 0.5 | 1084 | 1085 * | 1085 * | 1085 * | 1085 * |
| la28 | 20 × 10 | 0.5 | 1069 | 1070 * | 1070 * | 1070 * | 1070 * |
| la29 | 20 × 10 | 0.5 | 993 | 994 * | 994 * | 994 * | 994 * |
| la30 | 20 × 10 | 0.5 | 1068 | 1069 * | 1069 * | 1069 * | 1069 * |
| la31 | 30 × 10 | 0.5 | 1520 | 1520 * | 1520 * | 1520 * | 1521 |
| la32 | 30 × 10 | 0.5 | 1657 | 1658 * | 1658 * | 1658 * | 1658* |
| la33 | 30 × 10 | 0.5 | 1497 | 1497 * | 1497 * | 1497 * | 1497 * |
| la34 | 30 × 10 | 0.5 | 1535 | 1535 * | 1535 * | 1535 * | 1535 * |
| la35 | 30 × 10 | 0.5 | 1549 | 1549 * | 1549 * | 1549 * | 1549 * |
| la36 | 15 × 15 | 0.5 | 948 | 948 * | 948 * | 948 * | 950 |
| la37 | 15 × 15 | 0.5 | 986 | 986 * | 986 * | 986 * | 986 * |
| la38 | 15 × 15 | 0.5 | 943 | 943 * | 943 * | 943 * | 943 * |
| la39 | 15 × 15 | 0.5 | 922 | 922 * | 922 * | 922 * | 922 * |
| la40 | 15 × 15 | 0.5 | 955 | 955 * | 955 * | 955 * | 956 |

Figure 18 presents the number of best results obtained for each algorithm in this dataset, where the GA-RRHC calculated the 43 best values, the same as the HA and superior to the GLNSA and IJA.

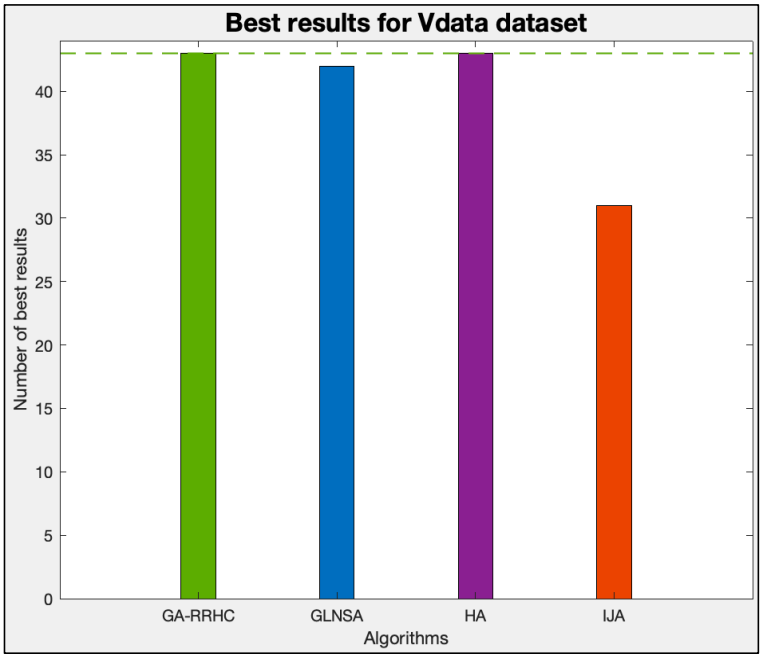


**Figure 18.** Number of best results per algorithm for the Vdata dataset.

Table 10 shows the ranking, the average *RPD*, and the non-parametric Friedman pairwise test values of each algorithm. The results show that the GA-RRHC obtained the first place concerning the RPD of the 43 problems, and the results indicate a significant difference with the IJA.

**Table 10.** Algorithm ranking and Friedman test value for Vdata dataset.

| **Algorithm:** | **GA-RRHC** | **GLNSA** | **HA** | **IJA** |
|---|---|---|---|---|
| Average *RPD*: | 0.0693 | 0.0772 | 0.0724 | 0.1177 |
| Rank: | 1 | 3 | 2 | 4 |
| *p*-value: | ~ | 0.1573 | 0.3173 | 0.0005 |

Figure 19 depicts the difference of the average RPD between the GA-RRHC and the other methods.

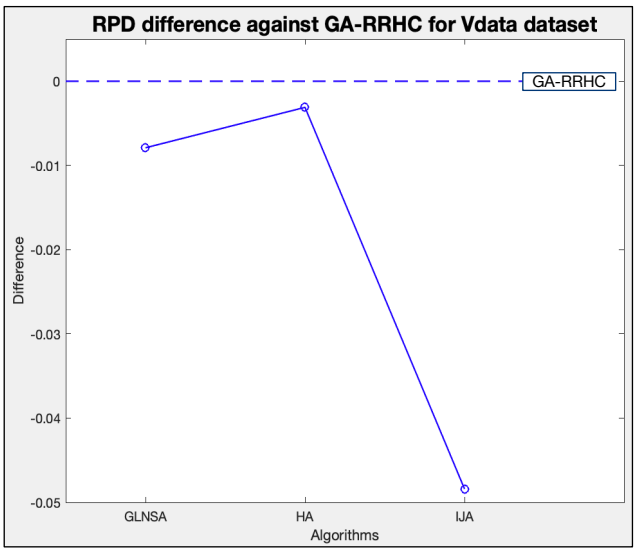


**Figure 19.** Comparison of the RPD difference per algorithm for the Vdata dataset.

### *4.7. Generated Large Dataset*

The previous experiments proved the correct performance of the GA-RRHC. However, the largest instance considers only 30 jobs, 10 machines, and 300 operations, which can be insufficient in real-world cases. Consequently, three randomly generated large instances were studied using the parameters established by [17,55]. These instances are named VL01, VL02, and VL03, considering from 50 to 80 jobs, from 20 to 50 machines, and from 704 to 2773 operations, with a flexibility rate $\beta = 0.75$. Since these instances have not been used before in previous studies, we only apply the GLNSA for comparison, whose code is available in Github https://github.com/juanseck/GLNSA-FJSP-2020 (accessed on 5 August 2022).

Table 11 shows that GA-RRHC performs much better than GLNSA, with enhanced performance for an increasing problem dimension. Figure 20 presents three examples, one for each VL instance, of the optimization process achieved by the GA-RRHC. In the Gantt charts, it is remarkable how the idle times decrease from initial random solutions and the continuous convergence to calculate the minimum makespan.

**Table 11.** Experiment with large instances.

| Instance | n × m | *o* | GLNSA | | GA-RRHC | |
|---|---|---|---|---|---|---|
| | | | **Best** | **Avg.** | **Best** | **Avg.** |
| VL01 | 50 × 20 | 704 | 592 | 617.3 | 551 | 570.9 |
| VL02 | 60 × 30 | 1246 | 759 | 781.3 | 705 | 717.8 |
| VL03 | 80 × 50 | 2773 | 1155 | 1183.1 | 1041 | 1058.4 |

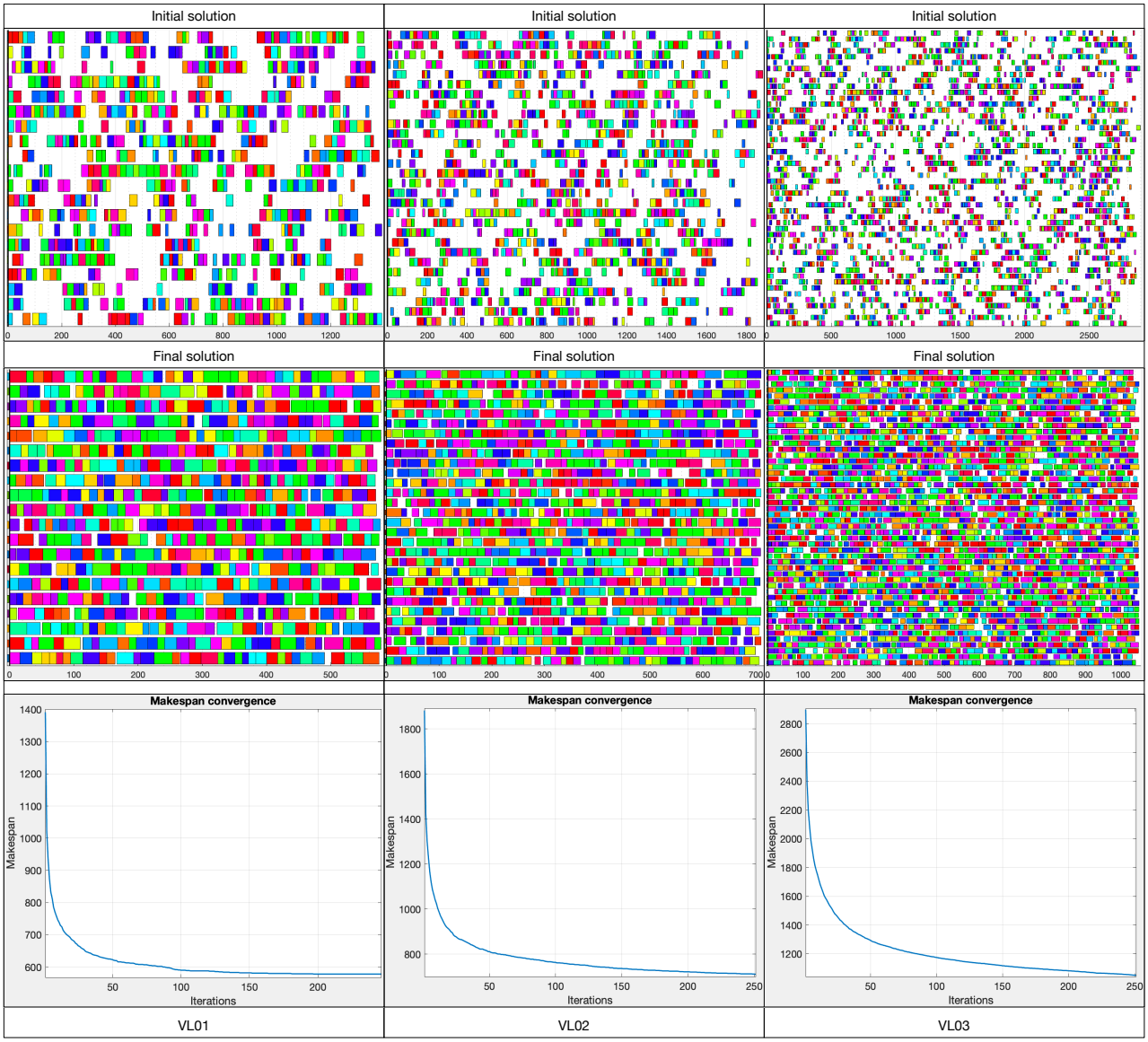


**Figure 20.** Comparison of the RPD difference per algorithm for the Vdata dataset.

These results corroborate that the GA-RRHC exhibits a performance comparable to recent algorithms recognized for their robustness in optimizing this type of problem, mainly for instances with high flexibility, and with competitive computational complexity.

## 5. Conclusions

This paper has described a hybrid algorithm that applies a global search with genetic crossover and mutation operators in a CA-like neighborhood. These operators mainly optimize the sequence of operations. Random-restart hill-climbing performs a local search to allocate the best machine to each critical operation. This GA-RRHC feature makes it suitable for FJSSP instances with high flexibility.

The CA-like neighborhood allows the concurrent application of genetic operations, which gives the GA-RRHC a better exploration capability. Hill-climbing uses a similar number of iterations compared to other algorithms and is easy to implement, reflecting a satisfactory complexity. Four popular datasets (covering 101 problems) were used for the numerical experimentation of the GA-RRHC. The results expose good performance compared to recent algorithms taken as reference, especially for instances with high flexibility.

The GA-RRHC opens a new way to apply CA-like neighborhoods that concurrently apply the exploration and exploitation operators to solve task scheduling problems; for instance, the flowshop, the job shop, or the open shop cases.

As a possible future work, it is suggested to use another type of exploitation technique such as simulated annealing, similar to the one proposed in [56], to reduce the complexity of the local search. Other variants of scaling algorithms can be tested for optimizing the sequences of operations to solve FJSSP instances with low flexibility, in addition to extending this methodology for the optimization of multi-objective problems.

**Author Contributions:** Conceptualization, N.J.E.-S. and J.C.S.-T.-M.; methodology, N.J.E.-S., J.C.S.-T.-M. and J.M.M.; software, N.J.E.-S. and J.C.S.-T.-M.; validation, J.M.-M., I.B.-V. and J.R.C.-A.; formal analysis, N.J.E.-S., J.C.S.-T.-M. and J.M.-M.; investigation, N.J.E.-S., J.C.S.-T.-M. and I.B.-V.; resources, J.M.-M. and J.C.S.-T.-M.; data curation, N.J.E.-S., J.C.S.-T.-M.; writing—original draft preparation, J.C.S.-T.-M., I.B.-V. and J.R.C.-A.; writing—review and editing, J.C.S.-T.-M., I.B.-V. and J.R.C.-A. visualization, N.J.E.-S., J.C.S.-T.-M. and J.M.M.; supervision, J.M.-M. and J.R.C.-A.; project administration, J.M.-M. and J.C.S.-T.-M.; funding acquisition, J.M.-M. and J.C.S.-T.-M. All authors have read and agreed to the published version of the manuscript.

**Funding:** This study was supported by the Autonomous University of Hidalgo (UAEH) and the National Council for Science and Technology (CONACYT) with projects numbers CB-2017-2018-A1-S-43008 and F003/320109. Nayeli Jazmin Escamilla Serna was supported by CONACYT grant number 1013175.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The GA-RRHC source code is available on Github https://github.com/juanseck/GA-RRHC (accessed on 5 August 2022). The datasets taken to test the GA-RRHC are available at https:/people.idsia.ch/~monaldo/fjsp.html (accessed on 5 August 2022).

**Conflicts of Interest:** The authors declare that they have no known competing financial interest or personal relationships that could have appeared to influence the work reported in this paper.

## Abbreviations

The following abbreviations were used in this research:

| | |
|---|---|
| FJSSP | Flexible job shop scheduling problem |
| GA | Genetic algorithm |
| RRHC | Random-restart hill-climbing algorithm |
| CA | Cellular automata |
| POX | Precedence operation crossover |
| JBX | Job-based crossover |
| OS | Operation sequence |
| MS | Machine sequence |

## References

1. Chen, H.; Ihlow, J.; Lehmann, C. A genetic algorithm for flexible job-shop scheduling. *IEEE Int. Conf. Robot. Autom.* **1999**, *2*, 1120–1125. [CrossRef]
2. Pinedo, M.L. *Scheduling Theory, Algorithms, and Systems*, 5th ed.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 1–670. [CrossRef]
3. Amjad, M.K.; Butt, S.I.; Kousar, R.; Ahmad, R.; Agha, M.H.; Faping, Z.; Anjum, N.; Asgher, U. Recent research trends in genetic algorithm based flexible job shop scheduling problems. *Math. Probl. Eng.* **2018**, *2018*, 9270802. [CrossRef]

4. Pezzellaa, F.; Morgantia, G.; Ciaschettib, G. A genetic algorithm for the flexible job-shop scheduling problem. *Comput. Oper. Res.* **2008**, *35*, 3202–3212. [CrossRef]
5. Garey, M.; Johnson, D.; Sethi, R. The complexity of flowshop and jobshop scheduling. *Math. Oper. Res.* **1976**, *1*, 117–129. [CrossRef]
6. Qing-dao-er ji, R.; Wang, Y. A new hybrid genetic algorithm for job shop scheduling problem. *Comput. Ind. Eng.* **2012**, *1*, 2291–2299. [CrossRef]
7. Yu, Y. A research review on job shop scheduling problem. *E3S Web Conf.* **2021**, *253*, 02024. [CrossRef]
8. Morrison David, R.; Jacobson, S.H.; Sauppe, J.J.; Sewell, E.C. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discret. Optim.* **2016**, *19*, 79–102. [CrossRef]
9. Mallia, B.; Das, M.; Das, C. Fundamentals of transportation problem. *Int. J. Eng. Adv. Technol. (IJEAT)* **2021**, *10*, 90–103. [CrossRef]
10. Che, P.; Tang, Z.; Gong, H.; Zhao, X. An improved Lagrangian relaxation algorithm for the robust generation self-scheduling problem. *Math. Probl. Eng.* **2018**, *2018*, 6303596. [CrossRef]
11. Holland, J.H. *Adaptation in Natural and Artificial Systems*; MIT Press Ltd.: Cambridge, MA, USA, 1975; p. 232.
12. Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*; Addison-Wesley: Reading, MA, USA, 1989.
13. Dorigo, M. Optimization, Learning and Natural Algorithms. Ph.D. Thesis, Politecnico di Milano, Milano, Italy, 1992.
14. Kennedy, J.; Eberhart, R. Particle swarm pptimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [CrossRef]
15. Glover, F. Tabu seach—Part I. *ORSA J. Comput. Summer* **1989**, *1*, 190–206. [CrossRef]
16. Brucker, P.; Schlie, R. Job-shop scheduling with multi-purpose machines. *Computing* **1990**, *45*, 369–375. [CrossRef]
17. Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Ann. Oper. Res.* **1993**, *41*, 157–183. [CrossRef]
18. Mastrolilli, M.; Gambardella, L.M. Effective neighbourhood functions for the flexible job shop problem. *J. Sched.* **2000**, *3*, 3–20. [CrossRef]
19. Li, X.; Gao, L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *J. Prod. Econ.* **2016**, *174*, 93–110. [CrossRef]
20. Deng, Q.; Gong, G.; Gong, X.; Zhang, L.; Liu, W.; Ren, Q. A bee evolutionary guiding nondominated sorting genetic algorithm II for multiobjetive flexible job shop scheduling. *Comput. Intell. Neurosci.* **2017**, *27*, 1–20. [CrossRef]
21. Gao, J.; Sun, L.; Gen, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Comput. Oper. Res.* **2008**, *35*, 2892–2907. [CrossRef]
22. Zhang, G.; Shao, X.; Li, P.; Gao, L. An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2009**, *56*, 1309–1318. [CrossRef]
23. Amiri, M.; Zandieh, M.; Yazdani, M.; Bagheri, A. A variable neighbourhood search algorithm for the flexible job-shop scheduling problem. *Int. J. Prod. Res.* **2010**, *48*, 5671–5689. [CrossRef]
24. Li, J.q.; Pan, Q.k.; Liang, Y.C. An effective hybrid tabu search algorithm for multi-objective flexible job-shop scheduling problems. *Comput. Ind. Eng.* **2010**, *59*, 647–662. [CrossRef]
25. Dalfarda, V.M.; Mohammadi, G. Two meta-heuristic algorithms for solving multi-objective flexible job-shop scheduling with parallel machine and maintenance constraints. *Comput. Math. Appl.* **2012**, *64*, 2111–2117. [CrossRef]
26. Yuan, Y.; Xu, H.; Yang, J. A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Appl. Soft Comput.* **2013**, *13*, 3259–3272. [CrossRef]
27. Li, J.Q.; Pan, Q.K.; Tasgetiren, M.F. A discrete artificial bee colony algorithm for the multi-objective flexible job-shop scheduling problem with maintenance activities. *Appl. Math. Model.* **2014**, *38*, 1111–1132. [CrossRef]
28. Gao, K.Z.; Suganthan, P.N.; Chua, T.J.; Chong, C.S.; Cai, T.X.; Pan, Q.K. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Syst. Appl.* **2015**, *42*, 7652–7663. [CrossRef]
29. Li, X.; Peng, Z.; Du, B.; Guo, J.; Xu, W.; Zhuang, K. Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems. *Comput. Ind. Eng.* **2017**, *113*, 10–26. [CrossRef]
30. Rodriguez Kato, E.R.; de Aguiar Aranha, G.D.; Tsunaki, R.H. A new approach to solve the flexible job shop problem based on a hybrid particle swarm optimization and random-restart hill climbing. *Comput. Ind. Eng.* **2018**, *125*, 178–189. [CrossRef]
31. Gong, G.; Deng, Q.; Gong, X.; Liu, W.; Ren, Q. A new double flexible job-shop scheduling problem integrating processing time, green production, and human factor indicators. *J. Clean. Prod.* **2018**, *174*, 560–576. [CrossRef]
32. Sreekara Reddy, M.; Ratnam, C.; Rajyalakshmi, G.; Manupati, V. An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. *Measurement* **2018**, *114*, 78–90. [CrossRef]
33. Meng, T.; Pan, Q.K.; Sang, H.Y. A hybrid artificial bee colony algorithm for a flexible job shop scheduling problem with overlapping in operations. *Int. J. Prod. Res.* **2018**, *56*, 5278–5292. [CrossRef]
34. Wu, X.; Shen, X.; Li, C. The flexible job-shop scheduling problem considering deterioration effect. *Comput. Ind. Eng.* **2019**, *135*, 1004–1024. [CrossRef]
35. Lin, J.; Zhu, L.; Wang, Z.J. A hybrid multi-verse optimization for the fuzzy flexible job-shop scheduling problem. *Comput. Ind. Eng.* **2019**, *127*, 1089–1100. [CrossRef]
36. Goerler, A.; Lalla-Ruiz, E.; Voß, S. Late acceptance hill-climbing matheuristic for the general lot sizing and scheduling problem with rich constraints. *Algorithms* **2020**, *13*, 138. [CrossRef]

37. Defersha, F.M.; Rooyani, D. An efficient two-stage genetic algorithm for a flexible job-shop scheduling problem with sequence dependent attached/detached setup, machine release date and lag-time. *Comput. Ind. Eng.* **2020**, *147*, 106605. [CrossRef]
38. Alzaqebah, M.; Jawarneh, S.; Alwohaibi, M.; Alsmadi, M.K.; Almarashdeh, I.; Mohammad, R.M.A. Hybrid brain storm optimization algorithm and late acceptance hill climbing to solve the flexible job-shop scheduling problem. *J. King Saud Univ. Comput. Inf. Sci.* **2020**, *34*, 2926–2937. [CrossRef]
39. Ding, H.; Gu, X. Hybrid of human learning optimization algorithm and particle swarm optimization algorithm with scheduling strategies for the flexible job-shop scheduling problem. *Neurocomputing* **2020**, *414*, 313–332. [CrossRef]
40. Escamilla-Serna, N.; Seck-Tuoh-Mora, J.C.; Medina-Marin, J.; Hernandez-Romero, N.; Barragan-Vite, I.; Corona Armenta, J.R. A global-local neighborhood search algorithm and tabu search for flexible job shop scheduling problem. *PeerJ Comput. Sci.* **2021**, *7*, e574. [CrossRef] [PubMed]
41. Jacobson, S.H.; Yücesan, E. Analyzing the performance of generalized hill climbing algorithms. *J. Heuristics.* **2004**, *10*, 387–405. [CrossRef]
42. McIntosh, H.V. *One Dimensional Cellular Automata*; Luniver Press: Bristol, UK 2009.
43. Eiben, A.E.; Smith, J.E. Introduction to evolutionary computing. *Nat. Comput. Ser.* **2015**, *2*, 287. [CrossRef]
44. Kari, J. Theory of cellular automata: A survey. *Theor. Comput. Sci.* **2005**, *334*, 3–33. [CrossRef]
45. Shi, Y.; Liu, H.; Gao, L.; Zhang, G. Cellular particle swarm optimization. *Inf. Sci.* **2011**, *181*, 4460–4493. [CrossRef]
46. Lagos-Eulogio, P.; Seck-Tuoh-Mora, J.C.; Hernandez-Romero, N.; Medina-Marin, J. A new design method for adaptive IIR system identification using hybrid CPSO and DE. *Nonlinear Dyn.* **2017**, *88*, 2371–2389. [CrossRef]
47. Seck-Tuoh-Mora, J.C.; Medina-Marin, J.; Martinez-Gomez, E.S.; Hernandez-Gress, E.S.; Hernandez-Romero, N.; Volpi-Leon, V. Cellular particle swarm optimization with a simple adaptive local search strategy for the permutation flow shop scheduling problem. *Arch. Control Sci.* **2019**, *29*, 205–226.
48. Hernández-Gress, E.S.; Seck-Tuoh-Mora, J.C.; Hernández-Romero, N.; Medina-Marín, J.; Lagos-Eulogio, P.; Ortíz-Perea, J. The solution of the concurrent layout scheduling problem in the job-shop environment through a local neighborhood search algorithm. *Expert Syst. Appl.* **2020**, *144*, 113096. [CrossRef]
49. Kacem, I.; Hammadi, S.; Borne, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.* **2002**, *32*, 1–13. [CrossRef]
50. Hurink, J.; Jurisch, B.; Thole, M. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Oper. Res. Spektrum* **1994**, *15*, 205–215. [CrossRef]
51. Baykasoğlu, A.; Madenoğlu, F.S.; Hamzadayı, A. Greedy randomized adaptive search for dynamic flexible job-shop scheduling. *J. Manuf. Syst.* **2020**, *56*, 425–451. [CrossRef]
52. Caldeira, R.H.; Gnanavelbabu, A. Solving the flexible job shop scheduling problem using an improved Jaya algorithm. *Comput. Ind. Eng.* **2019**, *137*, 106064. [CrossRef]
53. Zarrouk, R.; Bennour, I.E.; Jemai, A. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. *Swarm Intell.* **2019**, *13*, 145–168. [CrossRef]
54. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [CrossRef]
55. Sun, L.; Lin, L.; Li, H.; Gen, M. Large scale flexible scheduling optimization by a distributed evolutionary algorithm. *Comput. Ind. Eng.* **2019**, *128*, 894–904. [CrossRef]
56. Sajid, M.; Jafar, A.; Sharma, S. Hybrid Genetic and Simulated Annealing Algorithm for Capacitated Vehicle Routing Problem. In Proceedings of the 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), Waknaghat, India, 6–8 November 2020; pp. 131–136.