axioms

Article

Genetic Algorithm for Scheduling Optimization Considering Heterogeneous Containers: A Real-World Case Study

Gilberto Rivera [1], Luis Cisneros [1,*], Patricia Sánchez-Solís [1], Nelson Rangel-Valdez [2] and Jorge Rodas-Osollo [1]

1 Department of Electrical and Computer Engineering, Autonomous University of Cd. Juárez, Cd. Juárez 32315, Mexico; gilberto.rivera@uacj.mx (G.R.); julia.sanchez@uacj.mx (P.S.-S.); jorge.rodas@uacj.mx (J.R.-O.)
2 Postgraduate & Research Division, National Mexican Institute of Technology/Madero Institute of Technology, Cd. Madero, Tamaulipas 89440, Mexico; nrangelva@conacyt.com
* Correspondence: luisisnerosart@gmail.com

Received: 21 September 2019; Accepted: 28 January 2020; Published: 4 March 2020

Abstract: In this paper, we develop and apply a genetic algorithm to solve surgery scheduling cases in a Mexican Public Hospital. Here, one of the most challenging issues is to process containers with heterogeneous capacity. Many scheduling problems do not share this restriction; because of this reason, we developed and implemented a strategy for the processing of heterogeneous containers in the genetic algorithm. The final product was named "genetic algorithm for scheduling optimization" (GAfSO). The results of GAfSO were tested with real data of a local hospital. Said hospital assigns different operational time to the operating rooms throughout the week. Also, the computational complexity of GAfSO is analyzed. Results show that GAfSO can assign the corresponding capacity to the operating rooms while optimizing their use.

Keywords: scheduling problem; surgery scheduling; genetic algorithm

1. Introduction

Genetic algorithms (GA) are an optimization strategy inspired by evolution. They were first introduced by Holland in 1970 [1]. GAs work by simulating the evolutionary process on a population of solutions. A typical GA generates an initial population by randomly assigning values to the decision variables of the problem. Thus, each individual of the population represents a possible solution to the optimization problem. The initial population will be submitted to a series of evolutionary operations.

The first step of the genetic process consists of determining the fitness level of every individual. Then, the best individuals are selected to go through a crossover process. The crossover consists of combining two individuals to obtain a new individual. The two chosen individuals are called "parents"and the new individual "offspring." This method hopes to create offspring with higher fitness than that of the parents. If so, offspring replaces the parents. Worse individuals are selected to undergo a mutation process. The mutation operator consists of altering the genetic information of a member of the population. If the resulted individual has better fitness, it replaces the old individual. In this kind of algorithms, iterations are called "generations," which are processed until (1) a stop condition is meet or (2) the program reaches a predefined limit of generations.

The proposals based on GAs are popular within the scientific literature (e.g., [2–4]) since GAs offer a favorable compromise between the quality of the solution and the run time spent finding it. Also, GAs have been implemented to solve real world problems [5–8].

Based on this analysis, we took the idea of the GA developed by Quiroz et al. [9], which is one of the more effective algorithms to solve bin packing problems and has also been applied to solve surgery scheduling problems [10]. The approach proposed by [9], and implemented in [10], is only capable of processing containers with homogeneous capacity. This feature often does not occur in hospitals and other organizations. Therefore, a wider range of real-world cases could be faced if the GA could process containers with heterogenous capacities. For this reason, we developed and implemented a strategy for the use of heterogeneous containers within the GA [10]. The resulted product was named "genetic algorithm for scheduling optimization" (GAfSO). GAfSO was tested using data on surgeries performed in a local hospital. According to experimental results, the proposed algorithm obtained up to 96% of the operating-room capacity for use, which represents a 45% decrease in terms of idle time compared with the organizational procedure for scheduling; as a consequence, time was reduced up to two weeks for inpatient surgery waiting lists.

Problem Definition

A scheduling problem is defined as follows:
given

a set of n activities with their durations $D = (d_1, d_2, d_3, \ldots, d_n)$, and a set of infinite containers with their capacities $C = (c_1, c_2, c_3, \ldots)$, we search for

schedule with all n activities programmed to be performed in a container such that

the capacity of the containers is not exceeded, and

idle time is minimized.

Because the number of possible schedules grows exponentially, the difficulty in solving these problems is due to the number of combinations a large set of activities generates. This fact often causes delays during the scheduling process, because the ease in which an efficient schedule is found decreases. The scientific literature additionally suggests scheduling problems belong to the NP-hard class. The use of an inefficient schedule may lead to high economic losses due to the waste of often-scarce resources and poor quality of health service. Examples of scheduling problems are scheduling courses in teaching institutions, appointments in a medical consulting room, project tasks, among others.

An important application of this problem is surgery scheduling. Population growth has incremented the number of surgeries performed in public hospitals. This fact often causes delays due to the difficulties that public budgets must attend to all of the demands of the population. Then, a key is that health agencies must find an effective schedule for performing surgeries; however, the complex nature of this problem makes it difficult to find optimal solutions [5,11,12].

Several studies related to hospital environments agree with the following terms:

- Inpatient: patients that have to stay overnight in the hospital [13].
- Outpatient: patients that do not stay overnight in the hospital [14].
- Emergency patients: patients that arrive due to emergencies [15].
- Online scheduling: patients without appointment are considered [16].
- Offline scheduling: only patients with appointment are considered [17].

Following the classification of [18], the case study described in this paper is a surgery schedule problem with offline patients with the objective function of maximize the use of operating rooms (ORs).

The rest of the document is divided as follows. In Section 2, a brief literature review is provided. Section 3 describes the components of GAfSO and the case study. In Section 4, we present the results from computational experiments. Finally, Section 5 discusses some conclusions and directions for future research.

2. A Brief Review of the Specialized Literature

Scheduling problem has been widely addressed in the literature, and there is a high number of approaches that use GAs to optimize these problems.

Budylskiy and Kvyatkovskaya considered in 2014 a multi-criteria optimization to schedule projects under constraints of cost and length. The GA they develop uses co-evolution and Pareto principles to obtain a satisfactory solution [19].

Marques et al. used in 2014 a GA for elective surgery schedule in a Portuguese hospital. Elective surgeries can be inpatient or outpatient, and no emergency patients are considered. This is the case for the hospital since it does not have emergency service. The GA is based on the biased random-key genetic algorithms (BRKGA), the main advantages of this approach are high locality and heritability. BRKGAs use a coding system to represent the solutions. The encoding scheme stores information on a matrix of days by rooms [11]. Experiments were conducted using the data of the hospital. The results of the BRKGA were compared to those of an integer linear programming approach. Results showed that the BRKGA can obtain high-quality results with a gap no more than 10%. Next year, they modified the GA to add surgery priority [6].

Azadeh et al. developed in 2015 a GA to schedule patients in emergency departments. The objective of the GA is to minimize the waiting time of the patients while scheduling according to priority treatments. The first step of the experimentation consisted of comparing the results of the GA with those of a branch and bound algorithm. Results demonstrated the capacity of the GA to obtain near-optimal solutions in a reasonable amount of time. Then, real data of an emergency department was used to test the algorithm further. A paired *t*-test demonstrated the statistical superiority of the proposed approach with respect to the system used by the emergency department [15].

Ortiz et al. in 2015, implemented several optimization techniques to solve a scheduling problem. They aimed to schedule courses at a university. The methods were compared with each other. Those techniques are genetic, memetic, and immune system algorithms. After comparing all the algorithms, the authors concluded that the GA offers several advantages [20].

Xiang et al. used, in 2015, ant colony optimization (ACO) to solve a surgery scheduling problem. Using a two-level model, the algorithm is capable of obtaining a satisfactory solution within a reasonable amount of time. It should be noted that the ACO considers the specialty of the surgeon and multiple phases associated with surgical procedures [21].

Figuera and Cavalcanti chose in 2016 to use a direct representation of chromosomes in their GA. This feature means that the position of the gen in the chromosome indicate on which day of the week the surgery is scheduled [22].

Li and Gao proposed in 2016 a hybrid approach between a GA and a tabu search, named HA, for the flexible job scheduling problem (FJSP). The classical JSP consists of scheduling a set of n jobs in a set of m machines, each job consists of op operations. Each machine can process one operation of a job at the same time. Each job must be processed by a specific route of machines. However, it is assumed that there is not flexibility of resources (machines and tools), and modern manufacturing enterprises have flexible systems. The FJSP was introduced to match these situations, allowing to assign jobs to any machine. The results of the algorithm were compared with six important benchmark instances, which includes 201 open problems. The HA achieves a high level of performance, obtaining new best solutions for several benchmark problems [23].

Latorre-Nuñes et al. developed in 2016 several tools for surgery scheduling. These tools optimize the matching between surgeries and surgical equipment, and the assignation of beds at the same time. Reducing the bottleneck present in sequential optimization. Thus, obtaining a “truly” optimal solution [24].

Rivera et al. developed in 2017 a GA to schedule surgeries in operating rooms. This paper arises from the necessity of a local hospital, given the growth of demand in surgery procedures. The demand surpassed 385 surgeries by month, exceeding the capacity of the hospital to respond to all patients efficiently. A genetic algorithm was developed and reached an increase of 45% with respect to the previous scheduling process [1].

Cheng-Yang et al. constructed in 2018 two algorithms to solve the flow shop with proportional permutations (PPF) problem. The algorithm schedule activities with variable maintenance (VMA). Said algorithms obtained optimal solutions for the PPF problem [25].

Rahmani et al. developed in 2019 a genetic algorithm to solve the open-shop scheduling problem (OSSP). In the OSSP, n jobs are processed by m machines in a not deterministic way. Each job has n operations that must be processed in a deterministic time interval by a machine. Each machine can process one operation of a job. The goal is to find an optimal schedule for the completion of the n jobs. The performance of the GA was compared against an ACO approach and an imperialist competitive algorithm (ICA). The algorithms were tested using small, medium, and large-sized benchmark instances. Results showed the GA was able to find better solutions in shorter computational time [26].

Guo et al. proposed in 2019 a GA based on column generation to solve an electric bus scheduling problem. As stated by Guo et al., bus companies are incrementing the use of electric vehicles because of their low transportation cost and low noise. The multi-depot electric bus vehicle scheduling problem (EVSP) is considered. Column generation (CG) is used to create initial solutions for the GA to optimize. The GA-CG approach was compared against the branch-and-price (BP) algorithm using sixteen instances based on real data. Results showed that the GA-CG obtained quality solutions in a short amount of time [27].

Mahdi et al. proposed in 2019 a BRKGA for the integrated scheduling of manufacturing, transport, and storage/retrieval operations in flexible manufacturing systems (FMSs). In an FMS environment, a set of machines, automated vehicles and automated storage/retrieval systems cooperate to perform jobs (in this context, a job consists of a set of manufacturing operations). To the authors' knowledge, the FMS had not been addressed in an integrated way with heuristic approaches. Here, the chromosome has two parts: the operations sequence and the vehicle assignment. The results of the BRKGA were compared to the optimal solution obtained by a MILP model. The proposed approach reached several exact solutions and maintained the gap between the MILP model under 2% [28].

Although many studies employ GAs to solve scheduling problems, there are approaches in similar optimization problems that have shown good results. This is the case for the bin packing problem (BPP). In the BPP a set of objects is to be pack in containers with limited capacity. The GA developed in [10] promotes the transmission of the best genes while avoiding local optimum. However, this approach is not able to process containers with heterogeneous capacities, narrowing the scope of the GA. In this paper, we developed and implemented a strategy for the use of heterogeneous containers within the GA, the final algorithm was named genetic algorithm for scheduling optimization (GAfSO). A comparison was made with one of the best algorithms to the moment, results showed promising results.

3. Development

In this section, we describe the GA, the developed strategy, and its application inthe case study.

3.1. Genetic Algorithm

In this section, we describe the components of the genetic algorithm developed.

3.1.1. Genetic Encoding

We utilize a representation based on groups to model a schedule. Gens are used to model time-spaces in which activities can be scheduled. Each gen has a length that cannot be exceeded. A chromosome contains a group of gens. In other words, each chromosome represents a possible schedule of the activities. In this representation, the chromosomes may differ in size depending on the number of ORs they use in the solution. Figure 1 exemplifies this encoding scheme.

Activity	Length
1	100
2	40
3	100
4	90

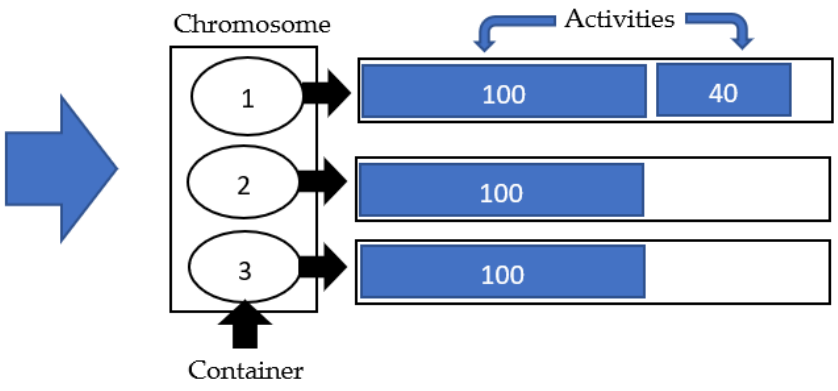


Figure 1. Genetic coding.

3.1.2. Fitness Function

The objective of GAfSO is to find a schedule that uses the minimum number of operating rooms as possible. Considering the similarities between the scheduling problem and the BPP, we chose a fitness function used on BPP. The fitness function is used to maximize the usage of ORs; therefore, it also minimizes the number of ORs used [29]. The function is expressed by Equation (1):

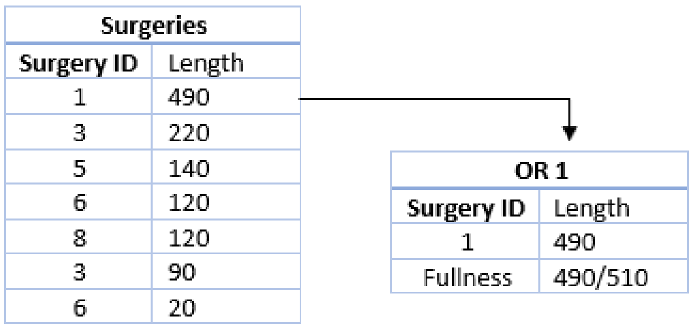
$$F = \frac{\sum_{i=1}^{N}\left(\frac{F_i}{c_i}\right)^k}{N} \tag{1}$$

where N is the number of used ORs in the solution, F_i is the sum of all the surgeries to be perform in bin i, c_i is the capacity of the ith-OR, and k is a constant ($k > 1$).

3.1.3. Initial Population

To generate the initial population, we applied a two-phase process in order to schedule a set of n surgeries. During the first phase, those surgeries that are greater than 50% of the capacity of the ORs are selected to be scheduled. Because none of these $\tilde{n}$ selected surgeries can share the same OR, an OR is created to schedule each single surgery. During this phase, a set $\tilde{n}$ of selected surgeries is scheduled in independent ORs. This methodology is called First-Fit with $\tilde{n}$ preassigned objects (FF-$\tilde{n}$). The rest of the n-$\tilde{n}$ activities are scheduled during phase two using a First-Fit (FF) methodology. The remaining n-$\tilde{n}$ surgeries are selected at random to schedule. FF goes throughout all the ORs until the surgery can be scheduled. If none of the ORs have enough space to perform the surgery, then a new OR will be created, and the surgery will be assigned to it. Figure 2 shows an example of this process.

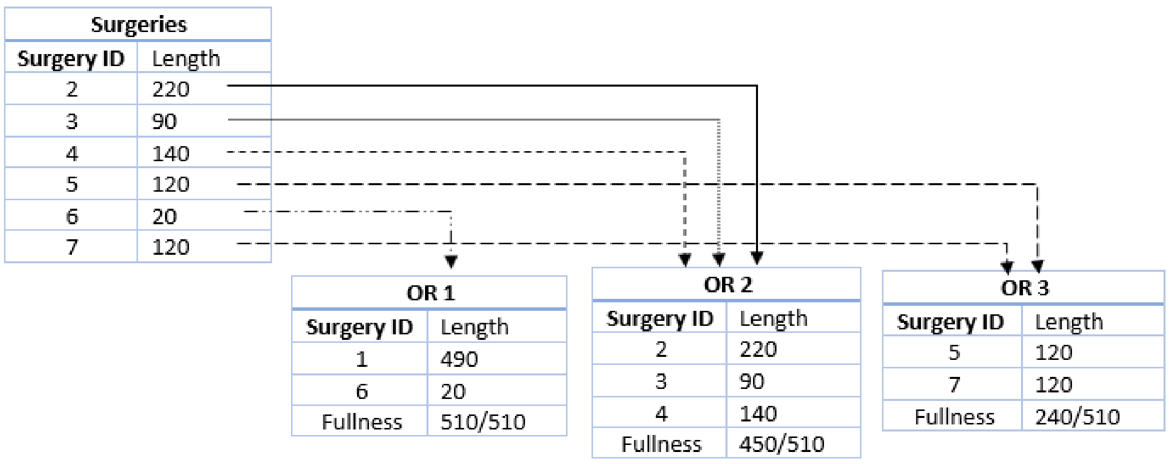
In Figure 2, a set of seven surgeries is going to be scheduled. The duration of the surgeries is given by the number of minutes it takes to perform them. In this example, the surgeries are to be assigned to ORs with a maximum capacity of 510 min. In Figure 2a, surgeries are ordered by their duration time before applying FF-$\tilde{n}$. FF-$\tilde{n}$ schedules Surgery 1 in an OR, in this case, only this surgery has a duration longer than 255 min (50% of the OR capacity). In Figure 2b, the rest of the surgeries are merged at random and scheduled using the FF methodology. In this case, since OR 1 does not have enough capacity to schedule Surgery 2, a second OR was added to schedule Surgery 2. Then, surgeries 3 and 4 were added to the OR 2 because OR1 did not have enough capacity for either. Next, because neither of the ORs had enough capacity to schedule Surgery 5, a new OR was added to schedule the activity. Surgery 6 was added to OR1, reaching its fullness. Finally, Surgery 7 was schedule in OR3. The result is a schedule conform by three ORs.



Surgeries	
Surgery ID	**Length**
1	490
3	220
5	140
6	120
8	120
3	90
6	20

OR 1	
Surgery ID	**Length**
1	490
Fullness	490/510

(**a**)



Surgeries	
Surgery ID	**Length**
2	220
3	90
4	140
5	120
6	20
7	120

OR 1	
Surgery ID	**Length**
1	490
6	20
Fullness	510/510

OR 2	
Surgery ID	**Length**
2	220
3	90
4	140
Fullness	450/510

OR 3	
Surgery ID	**Length**
5	120
7	120
Fullness	240/510

(**b**)

Figure 2. Construction of a solution, (**a**) FF-ñ applied to a set of surgeries; (**b**) FF applied to the rest of the surgeries.

3.1.4. Selection

Parents for crossover are chosen from two groups. One is the elite group, and the other has some non-elite individuals to promote diversity. The elite group contains those individuals that have the highest fitness. The non-elite group contains individuals selected at random. It may be possible that some individuals are selected for both groups; however, it is not allowed to perform the crossover if the same individual is chosen from both groups as parent. In Figure 3, an example of the two groups is given.

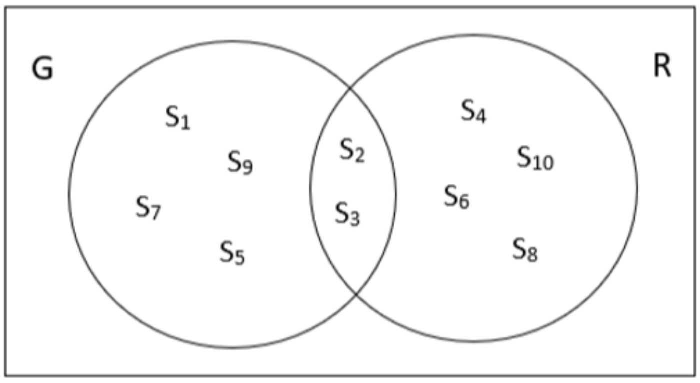


Figure 3. Selection for crossover.

In Figure 3, group G represents the elite group, and the group R represents the non-elite group. In this example, group G was first created with solutions one, two, three, five, seven, and nine. After the creation of group G took place, group R was created by selecting solutions at random. As we can see, solutions two and three are also present in group R. If one of these solutions is chosen from group G to participate in the crossover, then the algorithm does not select the same solution from group R.

Selection for mutation is done in two phases: (1) Calculate the number of individuals that will be mutated, and (2) select the calculated number of individuals from the population. Phase 1 uses some equations to calculate how many individuals are going to be mutated. These equations are given in Section 3.1.6. Phase 2 is also divided into two phases: (2.1) First, those individuals that reach a limit of generations are selected, (2.2) then, the worst individuals of the population are then selected to reach the number of individuals calculated in Phase 1.

3.1.5. Crossover

A process of crossover selects two solutions, referred to as parent solutions, and combines them in order to create two new solutions, referred to as offspring solutions. It is expected that an optimal solution to the scheduling problem has most of the ORs at almost fully used. Therefore, the chromosomes of the parents with ORs at full contribute to reproduce offspring with a high fitness. For this reason, we choose to utilize a crossover method that allows for the combination of the parents to a gen level. This gives more probability for the preservation of the better gens [9].

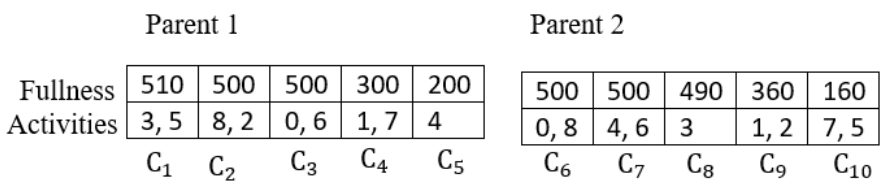
Crossover compares the ORs of the parent in parallel, after ordering them in descending order. This increases the probability that the offspring inherits the best ORs. For each pair of ORs, the best is inherited to the new solution, followed by the other OR. Then, ORs with duplicated surgeries are erased, and their surgeries released. While this is happening, the released surgeries are stored in a temporary list. When this process finishes, the surgeries that are already in the solution are removed from the list. Finally, the remaining surgeries are reschedule using First-Fit Decreasing methodology, which will be described next.

First-Fit Decreasing (FFD)

This methodology sorts a list of surgeries descending according to the duration of them. Then, each surgery is scheduled in the first OR that has enough space to store it. If there is no such case, the surgery is scheduled in a new OR that is added to the solution.

Numerical Example

We will view the application of the crossover operator with an example of a scheduling problem with a set of nine surgeries with the next durations (200, 160, 200, 490, 200, 20, 300, 140, 300), and ORs with a capacity of 510 min. Figure 4 shows the two chromosomes that we will be using for the example.


Parent 1

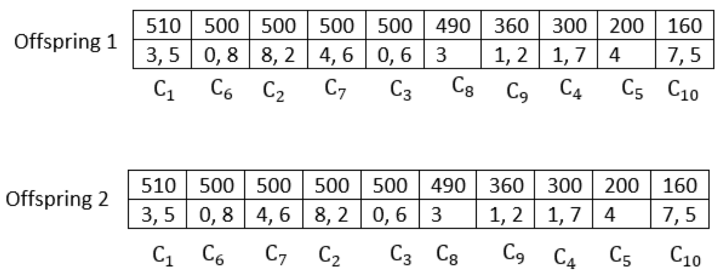
Fullness	510	500	500	300	200
Activities	3, 5	8, 2	0, 6	1, 7	4
	C_1	C_2	C_3	C_4	C_5

Parent 2

500	500	490	360	160
0, 8	4, 6	3	1, 2	7, 5
C_6	C_7	C_8	C_9	C_{10}

Figure 4. Parent for crossover.

The first step is to combine the gens of the parents. Two offspring solutions will be created, one combining Parent 1 with Parent 2, and the other combining Parent 2 with Parent 1. Figure 5 shows both offspring solutions.



Offspring 1

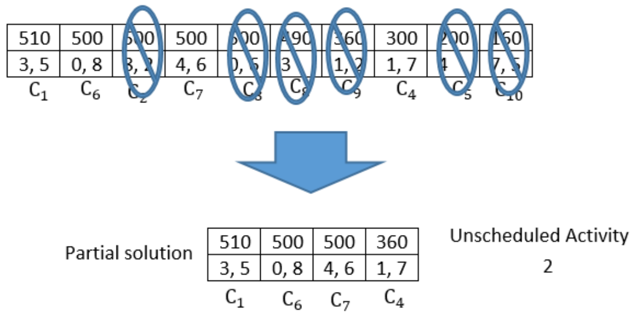
510	500	500	500	500	490	360	300	200	160
3, 5	0, 8	8, 2	4, 6	0, 6	3	1, 2	1, 7	4	7, 5
C_1	C_6	C_2	C_7	C_3	C_8	C_9	C_4	C_5	C_{10}

Offspring 2

510	500	500	500	500	490	360	300	200	160
3, 5	0, 8	4, 6	8, 2	0, 6	3	1, 2	1, 7	4	7, 5
C_1	C_6	C_7	C_2	C_3	C_8	C_9	C_4	C_5	C_{10}

Figure 5. Offspring solutions.

In Figure 5, we can see how some surgeries are repeated in the solutions. For example, in offspring solution 1, surgery 8 is scheduled in OR 6 and 2. The next step in the crossover methodology is to erase all of the ORs with repeated activities. In this example, ORs 2, 7, 8, 9, and 10 are eliminated. After removing those ORs, activity 2 was also erased from the solution. In Figure 6, we can see the resultant individual.



510	500	500	500	500	490	360	300	200	160
3, 5	0, 8	8, 2	4, 6	0, 6	3	1, 2	1, 7	4	7, 5
C_1	C_6	C_2	C_7	C_3	C_8	C_9	C_4	C_5	C_{10}

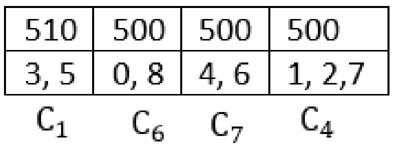
Partial solution

510	500	500	360
3, 5	0, 8	4, 6	1, 7
C_1	C_6	C_7	C_4

Unscheduled Activity
2

Figure 6. Elimination of repeated activities.

In order to complete the solution, FFD is used in order to reschedule the activities that were eliminated from the solution. In this example, Activity 2 was assigned to OR 5. Figure 7 shows the final solution.



510	500	500	500
3, 5	0, 8	4, 6	1, 2,7
C_1	C_6	C_7	C_4

Figure 7. Offspring solution generated with crossover.

3.1.6. Mutation

The ORs from a solution are ordered according to their usage, descending. The less filled ORs are eliminated, and their activities are rescheduled using FF. Given a solution with m ORs, of which t are unfilled, the number of ORs n_b to be eliminated is given by Equation (2). The elimination ratio ε is defined by Equation (3), p_ε is the elimination probability defined by Equation (4), and k is a parameter that defines the rate of change of ε and p_ε with respect to t.

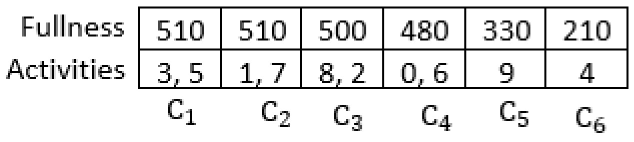
$$n_b = (t)(\varepsilon)(p_\varepsilon) \tag{2}$$

$$\varepsilon = \frac{2 - \frac{t}{m}}{t^{1/k}} \tag{3}$$

$$p_\varepsilon = 1 - U\left(0, \frac{1}{t^{1/k}}\right) \tag{4}$$

Numerical Example

Given a set of ten activities with durations (300, 250, 300, 490, 210, 20, 180, 260, 200) scheduled in ORs with a capacity of 510 min, Figure 8 illustrates a solution to mutate.



Fullness	510	510	500	480	330	210
Activities	3, 5	1, 7	8, 2	0, 6	9	4
	C_1	C_2	C_3	C_4	C_5	C_6

Figure 8. Solution to mutate.

In this example, the number of ORs is m = 6, the number of unfilled ORs is t = 4, the change rate k = 2 and a random number U (0, 0.62) = 0.5. The elimination proportion is U = 0.839, the probability proportion p_ε = 0.5, and the number of ORs to eliminate n_b = 2. The mutation operator will sort the ORs in descendent order of their fullness. So, ORs 5 and 6 are eliminated, and their activities (9 and 4) are left unassigned. Figure 9 shows this phase.

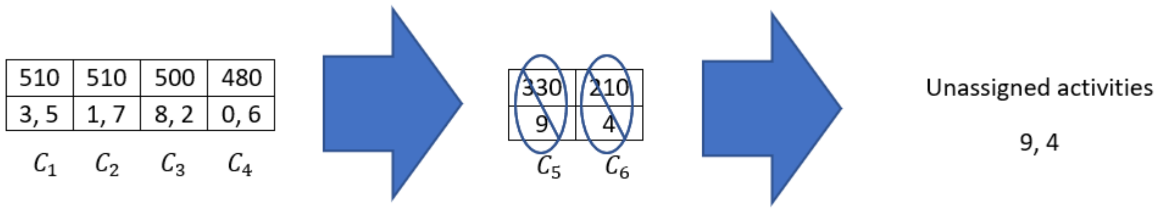
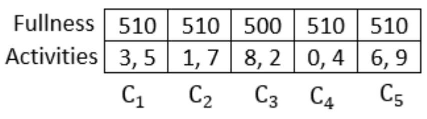


Figure 9. Elimination of ORs.

To finalize the process, a rearrangement methodology, based on the dominance criterion, is used to reschedule the unassigned activities. This allows for the exchange between unscheduled and scheduled activities. In this example, Activity 6 from OR 4 is exchanged with Activity 4 because this arrangement achieves the full use of the OR. Then, a new OR is created, and activities 6 and 9 are scheduled together. In Figure 10, the final solution is presented.



Fullness	510	510	500	510	510
Activities	3, 5	1, 7	8, 2	0, 4	6, 9
	C_1	C_2	C_3	C_4	C_5

Figure 10. Final mutated solution.

3.1.7. Stop Condition

Within GAfSO, the end of the evolution process happens when one of the following conditions are met:

The algorithm generates a solution with the same or a smaller number of ORs previously calculated, named theorical optimum.

The evolution process reaches the maximum number of allowed generations.

The theorical optimum determines which would be the minimal quantity of ORs needed to schedule a set of activities. This value is determined by Equation (5)

$$x(t,i) = \begin{cases} 1 + x(t - c_i, i + 1) & \text{if } t = 0, \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where:

t is the summation of the duration of all activities to schedule.

i is a control parameter, and

c_i is the capacity of the ith OR.

3.2. Strategy for the Use of Heterogeneous ORs

Based on the study of Figuera and Cavalcanti [22], we choose a direct representation of the chromosomes. This model determines the capacity of each OR based on the position within the array of ORs. The model was selected due to its advantages for implementation in the algorithm and its flexibility for different conditions of variable capacity.

GAfSO was modified to assign the capacity of the ORs before their processing. Given an n number of activities to schedule, each new solution will be created with n ORs. This is done because, clearly, there cannot be more ORs than surgeries. The capacity of each OR will be assigned based on its position. Then the surgeries will be scheduled maximizing the usage of OR using n as the upper bound and the theorical optimum as the lower bound. This strategy allows for the modeling of several constrains regarding heterogeneous capacity. Thus, allowing GAfSO to work in a wider range of case studies. Next, the usage of the strategy is exemplified using a real case study.

Case Study

We followed the same case study presented in [10]. This case study was constructed upon data provided by a local hospital. The hospital has four ORs with the same usage time. The usage time of the OR is 510 min, except the Fridays where it changes to 450 min. This situation is modeled by Equation (6), where i is the identifier of the OR.

$$c_i = \begin{cases} 510 & \text{if } (i-1) \bmod 20 < 16, \\ 450 & \text{otherwise.} \end{cases} \tag{6}$$

ORs are used from 8:30 am. to 4:50 pm. on Fridays, and from 8:30 am. to 5:00 pm. the rest of the week. Equation (6) monitors the day of the week by counting the ORs. Since the hospital has four ORs, each group of four ORs represents a day. Using this information GAfSO is able to identify those ORs that conform each Friday and correctly assign their capacity.

3.3. Computational Complexity

In this section we analyze the computational complexity of each operator of GAfSO analyzing the worst-case scenario if applicable. For the sake of efficiency, insertion sort was used to get sorted short sets, i.e., all sets related to the population size (whose worst case is in O(n^2)), whilst merge sort was applied to large sets, i.e., all related sets to the input data, chiefly the number of surgeries (whose worst case is in $O\big(n \log_2 n\big)$).

Lemma 1. *Calculation of fitness has a complexity of* $|S| + 2$.

Proof. Fitness is obtained by the summation of all surgery durations for a given solution (|S|). Then, the summation is elevated to a constant k and the result divided by the number of OR of the given solution. Each operation is performed once for the fitness calculation of a solution. □

Lemma 2. *Creation of the initial population has a complexity of* $p((n-\widetilde{n})\log_2(n-\widetilde{n}) + ((n-\widetilde{n})\cdot f(OR+1)) + (n\log_2 n + 2\widetilde{n}))$.

Proof. The first step in the creation of the initial population is FF-$\tilde{n}$. In this methodology, surgeries are sorted in descending order of their durations ($n \log_2 n$). Then, an OR is created to schedule each $\tilde{n}$ surgery ($2\tilde{n}$). This gives us a complexity of $n \log_2 n + 2\tilde{n}$ for FF-$\tilde{n}$. The second step is to apply FF to the rest of the surgeries, here the worst-case scenario would be that none of the OR has enough capacity to hold the activities because of a heterogeneous capacity constraint. First, the rest of the surgeries are sorted at random $(n-\widetilde{n})\log_2(n-\widetilde{n})$. Then, each surgery is scheduled in a new OR after going throughout all the ORs $(n-\tilde{n})\cdot f(OR+1)$. Therefore, FF has a complexity of $(n-\widetilde{n})\log_2(n-\widetilde{n}) + ((n-\widetilde{n})\cdot f(OR+1)$. Finally, this process is repeated to create each individual for a population size of p. □

Lemma 3. *Selection for crossover has a computational complexity of* $2\Big(p^2 + \ (n\cdot pc)\Big)\cdot(n\cdot pc)$.

Proof. Two groups of solutions are created for crossover. Each group has $n\cdot pc$ individuals, where pc is the probability for crossover ($0 < pc < 1$). The first one is an elite group that is comprised of the best solutions of the population. To create this group the population is sorted in descendent order of fitness (p^2). Then, the first $n*pc$ individuals are selected ($n\cdot pc$). The second group is comprised of random individuals. For this, the population is sorted at random and then the $n\cdot pc$ individuals are selected ($n\cdot pc$). This process is repeated $n\cdot pc$ times. □

Lemma 4. *Crossover has a computational complexity of* $(n\cdot pc)\cdot(|S_i| + \ |SR|^2 + (|SE_i| + \ |SR_i|)^2 \ + \ |SE|^2 + |SR|^2 + |SH_i| + (n-\widetilde{n})\log_2(n-\widetilde{n}) + ((n-\widetilde{n})\cdot f(OR+1))$.

Proof. The first step of the crossover is to sort the individuals from the random group in descending order of their fitness $|SR|^2$. The next step is to create the new individual using the ORs of both solutions. This step takes $|SE|$ or $|SR|$ depending on which solution has more ORs. During the next step, ORs with repeated surgeries are eliminated $|SH|$ and their surgeries are rescheduled using FF $(n-\widetilde{n})^2 + ((n-\widetilde{n})\cdot f(OR+1)$. Finally, the fitness for the new individual is calculated. □

Lemma 5. *Selection for mutation has a computational complexity of* $n \ \log_2 \ n + (n-lim)\log_2(n-lim) + ((n\cdot pm) - lim)$.

Proof. In the selection for mutation, a number of $n\cdot pm$ individuals are selected. pm is the probability for mutation $pm < 1$. First, the population is sorted by descending order of their age (number of generations)— $n \ \log_2 \ n$. Those individuals that reach a limit of age are selected for mutation (lim). The worst-case scenario would be that the selected individuals do not reach $n\cdot pm$. So, the next step is to sort the rest of the population by ascending order of their fitness $(n-lim)\log_2(n-lim)$. Finally, individuals are selected until $n\cdot pm$ is reached ($n\cdot pm - lim$). □

Lemma 6. *Mutation has a computational complexity of* $(n\cdot pm)\cdot(|SM|^2 + c + ((n-\widetilde{n})\log_2(n-\widetilde{n}) + ((n-\widetilde{n})\cdot f(OR+1)))$.

Proof. First, ORs of a solution are sorted by ascending order of their fullness $|SM|^2$. Then, a number of c ORs are eliminated and their activities are rescheduled using FF, whose complexity is $(n-\widetilde{n})\log_2(n-\widetilde{n}) + ((n-\widetilde{n})\cdot f(OR+1)$. This process is done for each selected solution ($n\cdot pm$). □

Lemma 7. *GAfSO has a computational complexity of* $O(n^2 \ \log_2 \ n)$, *where n is the number of activities to be scheduled.*

Proof. Initial population and theorical optimum are performed one time, while the rest of the processes are done iteratively. The worst-case scenario would be $\tilde{n}=0$ and $pm=1$, then the complexity of GAfSO would be in $O(n^2 \ \log_2 \ n)$—see Lemma 6. □

4. Results

In this section we describe the test instances used, the experimental conditions and the results obtained by GAfSO.

4.1. Test Instances

We used the instances presented in [1]. These are four instances proportionated by a local hospital. Each instance corresponds to the surgery schedule of a month. The four instances are described in Table 1.

Table 1. Test instances.

Instance	Number of Surgeries	Average Length of Surgery	Longest Surgery	Shortest Surgery
1	238	103 min	420 min	45 min
2	210	125 min	395 min	40 min
3	192	122 min	380 min	40 min
4	221	124 min	510 min	40 min

The hospital has four ORs. A group of four ORs represents a scheduled day. The ORs have a capacity of 510 min except for Fridays when their capacity is 450 min. Our goal is to schedule all surgeries using ORs with variable capacity while maintaining the solution quality by GAfSO.

4.2. Experimental Conditions

We implemented GAfSO under the next conditions.

Software: GAfSO was codded using the programming language C# and compiled in Visual Studio 2017 under Windows 10 of 64 bits. The following Table 2 lists the parameter values used in GAfSO.

Table 2. Genetic algorithm for scheduling optimization (GAfSO) parameters.

Parameter	Value
Population size	100
Elite population size	10
Maximum number of generations	500
Solution lifespan	10
Individuals for selection	80
Individuals for mutation	17

Hardware: All the experimentation was done using a computer with Windows 10 of 64 bits, Intel Core i7 3.4 GHz and 8GB RAM.

4.3. Results

GAfSO was able to find a solution in every instance while assigning the corresponding capacity to each operating room. In order to measure GAfSO solution quality, we run the algorithm thirty times. Figure 11 shows the usage percentage of the ORs.

In this experiment, the algorithm obtained an average of 95.33% in OR usage and a variance of 1.13. Besides measuring the quality of solutions, we also measure the execution time of the program through the runs. Figure 12 shows this information. In most of the cases, the solution was founded in the first generations.

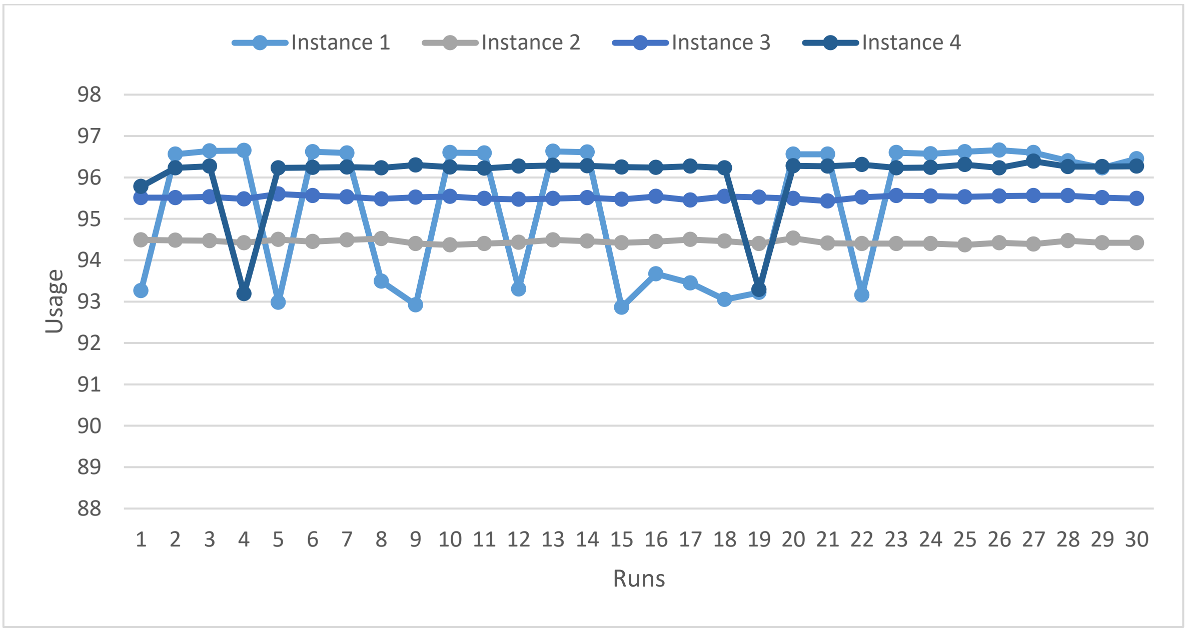


Figure 11. Usage of operation rooms.

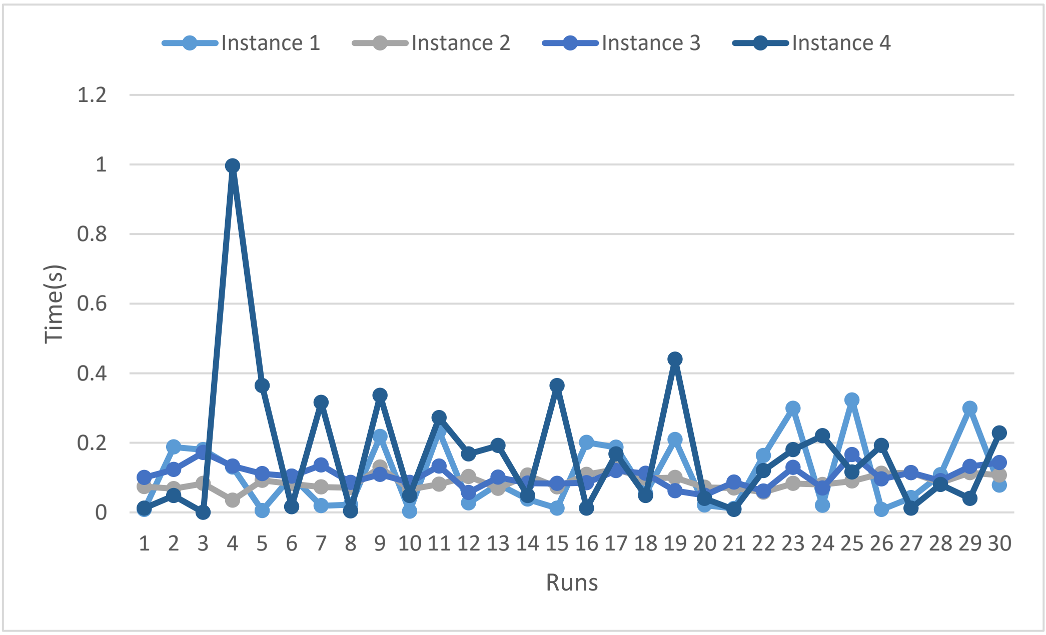


Figure 12. Execution time.

In this experiment, GAfSO reach an average of 0.117 s. With a variance of 0.013.

4.4. Results Comparison

As stated earlier, the GA developed in [1] was not capable of processing ORs with heterogeneous capacities. because of this reason, to schedule the surgeries of the hospital, the method showed in Figure 13 was used.

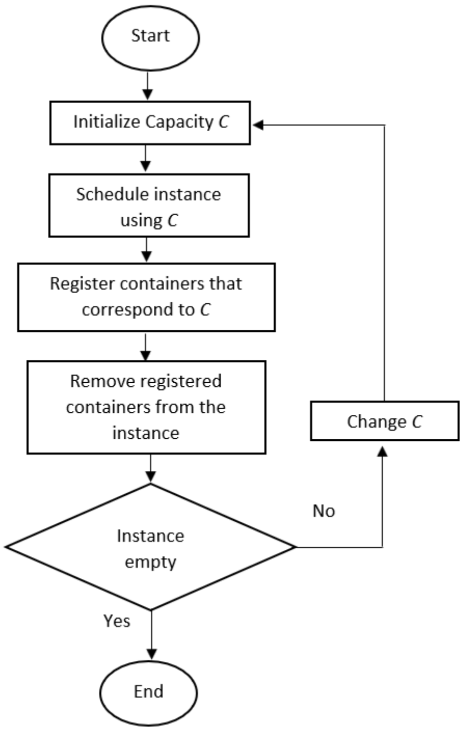

Start
Initialize Capacity *C*
Schedule instance using *C*
Register containers that correspond to *C*
Remove registered containers from the instance
Instance empty
No
Change *C*
Yes
End

Figure 13. GA strategy for multiple capacities.

We will describe how to use the methodology of Figure 13 with the case study. Here, the values for the capacities will be $p = 510$, and $S = 450$. In the first step, C will be initialized with p. Then, the GA solves the instance. After that, the part of the schedule that corresponds to Monday to Thursday will be stored and then eliminated from the instance. If there are surgeries left in the instance, C changes to the other capacity, in this case S. Then, the process is repeated, except that S is considered, and only ORs corresponding to Friday are stored and then removed from the instance.

Table 3 shows a comparison between the results obtained by the GA developed in [1] and GAfSO. Both programs solve all instances thirty times; in the case of the GA, five executions were needed to solve the instance.

Table 3. Results comparison

	Instance 1		**Instance 2**		**Instance 3**		**Instance 4**		**Average**	
	GA [1]	**GAfSO**	**GA**	**GAfSO**	**GA**	**GAfSO**	**GA**	**GAfSO**	**GA**	**GAfSO**
Fitness	96.33	95.54	95.2	94.4	96.1	95.5	94.26	96.04	94.81	95.3
Number of ORs used in the solution	50	50	54	54	48	48	56	56	52	52
Number of days used in the solution	12.5	12.5	13.5	13.5	12	12	14	14	13	13
Number of executions	5	1	5	1	4	1	5	1	4.75	1
Average computational time (s)	0.03	0.1	0.012	0.08	0.035	0.107	0.043	0.169	0.02	0.11

5. Conclusions

In [10], a genetic algorithm for BPP was developed. Because of its performance, this genetic algorithm was implemented in a scheduling problem [1]. However, the GA is only capable of processing containers with homogeneous capacity, narrowing the scope of possible applications for it. We developed and implemented a strategy for the usage of heterogeneous containers within the genetic algorithm. The resulting algorithm was named "genetic algorithm for scheduling optimization"

(GAfSO). The flexibility of the strategy enables the usage of GAfSO in other scheduling problems with varying conditions for heterogeneous capacity. The computational complexity of GAfSO is analyzed.

Experiments were conducted using four instances provided by a local hospital. Each instance corresponds to the surgery schedule of one month. The hospital uses two capacities for the operating rooms throughout the week. The first experiment consisted of the schedule of the surgeries performed in each instance. Results show that GAfSO is capable of reaching up to a 96% usage of ORs. Besides, GAfSO obtained the schedule in one execution, instead of the multiple executions the GA previously needed to reach the solution. This characteristic of GAfSO simplifies in great measure the process of scheduling a set of activities and reduces the run time. Future research will consist of comparing GAfSO with other proposals from the literature, and performance analysis using benchmark instances.

Author Contributions: All authors contributed significantly in writing this paper in the following manner: conceptualization and methodology, G.R. and P.S.-S.; validation, N.R.-V.; formal analysis, J.R.-O.; Software development, L.C.; writing—original draft preparation, L.C.; writing—review and editing, P.S.-S. and N.R.-V.; supervision, G.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors gratefully acknowledge the support of the *EUREKA Community* and the Autonomous University of Ciudad Juárez (División Multidisciplinaria de Ciudad Universitaria).

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

References

1. Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*; Addison-Wesley Professional: Boston, MA, USA, 1989.
2. Agarwal, M.; Srivastava, G.M.S. A Genetic Algorithm Inspired Task Scheduling in Cloud Computing. In Proceedings of the IEEE International Conference on Computing, Communication and Automation(ICCCA), Greater Noida, India, 29–30 April 2016.
3. Sheng, X.; Li, Q. Templeta-based Genetic Algorithm for QoS-aware Task Scheduling in Cloud Computing. In Proceedings of the 2016 International Conference on Advanced Cloud and BigData (CBD), Chengdu, China, 13–14 August 2016.
4. Gao, X.M.; Yang, Y.; Wu, H.Z. Genetic algorithm for scheduling double different size crane system with different truck ready times. In Proceedings of the 2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), Bali, Indonesia, 4–7 December 2016.
5. Marques, I.; Captivo, M.E. Bicriteria elective surgery scheduling using an evolutionary algorith. *Oper. Res. Health Care* **2015**, *7*, 14–26. [CrossRef]
6. Fügener, A.; Hans, E.W.; Kolisch, R.; Kortbeek, N.; Vanberkel, P. Master surgery scheduling with consideration of multiple downstream. *Eur. J. Oper. Res.* **2014**, *239*, 227–236. [CrossRef]
7. Ma, L.; Gong, M.; Lui, J.; Cai, Q.; Jiao, L. Multi-level learning based memetic algorithm for community detection. *Appl. Soft Comput.* **2014**, *19*, 121–133. [CrossRef]
8. Mia, L.; Li, J.; Lin, Q.; Gong, M.; Coello, C.A.C.; Zhong, M. Cost-Aware Robust Control of Signed Networks by Using a Memetic Algorithm. *IEEE Trans. Cybern.* **2019**, 1–14. [CrossRef] [PubMed]
9. Quiroz-Castellanos, M.; Cruz-Reyes, L.; Torres-Jimenez, J.; Gómez, C.; Fraire-Huacuja, H.J.; Alvim, A.C. A grouping genetic algorithm with controlled gene transmission for the bin packing problem. *Comput. Oper. Res.* **2015**, *55*, 52–64. [CrossRef]
10. Rivera, G.; Rodas-Osollo, J.; Bañuelos, P.; Quiroz, M.; Lopez, M. A Genetic Algorithm for surgery Scheduling Optimization in a Mexican Public Hospital. In *Recent Advances in Artificial Intelligence Research and Development*; IOS Press: Amsterdam, The Netherlands, 2017; pp. 269–274.
11. Conforti, D.; Guerriero, F.; Guido, R. *A MultiObjective Block Scheduling Model for the Managment of Surgical Operating Rooms: New Solution Approaches via Genetic Algorithms*; Health Care Management (WHCM): Venice, Italy, 2010.
12. Marques, I.; Captivo, M.E.; Pato, M.V. Scheduling elective surgeries in a Portuguese hospital using a genetic heuristic. *Oper. Res. Health Care* **2014**, *3*, 59–72. [CrossRef]

13. Zhao, L.; Chien, C.F.; Gren, M. A bi-objective genetic algorithm for intelligent rehabilitation scheduling considering therapy precedence constraints. *J. Intell. Manuf.* **2018**, *29*, 873–988. [CrossRef]
14. Azadeh, A.; Baghersad, M.; Farahani, M.H.; Zarrin, M. Semi-online patient scheduling in pathology laboratories. *Artif. Intell. Med.* **2015**, *64*, 217–226. [CrossRef] [PubMed]
15. Azadeh, A.; Farahani, M.H.; Torabzadeh, S.; Baghersad, M. Scheduling prioritized patients in emergency department laboratories. *Comput. Methods Programs Biomed.* **2014**, *117*, 61–70. [CrossRef] [PubMed]
16. Braaksma, A.; Kortbeek, N.; Post, G.F.; Nollet, F. Integral multidisciplinary rehabilitation treatment planning. *Oper. Res. Health Care* **2014**, *3*, 145–159. [CrossRef]
17. Bikker, I.A.; Kortbeek, N.; vanOs, R.M.; Boucherie, R.J. Reducing access times for radiation treatment by aligning the doctor's schemes. *Oper. Res. Health Care* **2015**, *7*, 111–121. [CrossRef]
18. Marynissen, J.; Demeulemeester, E. *Literature Review on Integrated Hospital Scheduling Problems*; Technical Report; Faculty of Economics and Business, KU Leuven: Leuven, Belgium, 2016.
19. Budylskiy, A.V.; Kvyatkovskaya, I.Y. Using coevolution genetic algorithm with Pareto principles to solve project scheduling problem under duration and cost constraints. *J. Inf. Organ. Sci.* **2014**, *38*, 1–9.
20. Ortiz-Aguilar, L.; Carpio-Valadez, J.M.; Puga-Soberanes, H.J.; Díaz-González, C.L.; Lino-Ramirez, C.; Soria-Alcaraz, J.A. Comparativa de algoritmos bioinspirados aplicados al problema de calendarización de horarios [Comparison of bioinspired algorithms applied to the schedule scheduling problem]. *Res. Comput. Sci.* **2015**, *94*, 33–43.
21. Xiang, W.; Yin, J.; Lim, G. An ant colony optimization approach for solving an operating room surgery scheduling problem. *Comput. Ind. Eng.* **2015**, *85*, 335–345. [CrossRef]
22. Marchesi, J.F.; Cavalcanti, M.A. *A Genetic Algorithm Approach for the Master Surgical Schedule Problem; IEEE conference on Evolving and Adaptive Intelligent Systems (EAIS)*; IEEE: Natal, Brazil, 2016.
23. Li, X.; Gao, L. An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *Int. J. Prod. Econ.* **2016**, *174*, 93–110. [CrossRef]
24. Latorre-Nuñez, G.; Lüer-Villagra, A.; Marianov, V.; Obreque, C.; Ramis, F.; Neriz, L. Scheduling operating rooms with consideration of all resources, postanesthesia beds and emergency surgeries. *Comput. Ind. Eng.* **2016**, *97*, 248–257.
25. Chen-Yang, C.; Kuo-Ching, Y.; Hsia-Hsiang, C.; Jia-Xian, L. Optimization algorithms for proportionate flow shop scheduling problems with variable maintenance activities. *Comput. Ind. Eng.* **2018**, *117*, 164–170. [CrossRef]
26. Hosseinabadi, A.A.R.; Vahidi, J.; Saemi, B.; Sangaiah, A.K.; Elhoseny, M. Extended Genetic Algorithm for solving open-shop scheduling problem. *Soft Comput.* **2019**, *23*, 5099–5116. [CrossRef]
27. Guo, C.; Wang, C.; Zuo, X. A Genetic Algorithm based Column Generation Method for Multi-Depot Electric Bus Vehicle Scheduling. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO2019), Prague, Czech Republic, 13–17 July 2019.
28. Mahdi, S.; Fontes, D.M.; Fontes, F.C. A BRK GA for the Integrated Scheduling Problem in FMSs. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO19), Prague, Czech Republic, 13–17 July 2019.
29. Falkenauer, E.; Delchambre, A. A genetic algorithm for bin packing and line balancing. In Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 12–14 May 1992.