Complexity of combinatorial ordering genetic algorithms COFFGA and CONFGA

Cite as: AIP Conference Proceedings **2144**, 050011 (2019); https://doi.org/10.1063/1.5123127
Published Online: 23 August 2019

Huda Hallawi, and Hongmei He

Complexity Of Combinatorial Ordering Genetic Algorithms COFFGA And CONFGA

Huda Hallawi[1, a)] ,Hongmei He[2, b)]

[1] *Computer Science Department ,University of Kerbala,Iraq*
[2]*Manufacturing Department, Cranfield University, Bedford, UK*

[a)]Corresponding author: huda.f@uokerbala.edu.iq
[b)] h.he@cranfield.ac.uk

Abstract. This paper analyses the complexity of two Algorithms called COFFGA (Combinatorial Ordering First Fit Genetic Algorithm) and CONFGA (Combinatorial Ordering Next Fit Genetic Algorithm). It also identifies the parameters that affect the performance of these algorithms. The complexity of the GA depends on the problem being solved by this GA, as well as the operators of the GA itself. The complexity of COFFGA and CONFGA are analysed individually. Even of these algorithms are slightly different, they may have extremely different complexities depending on the differences in their fitness function or termination condition. To provide a provable bound on a problem, there must be a bound on the evaluation function as well as a manner by which the underlying problem is tied to the representation. Given that there is no standard complexity of the GA, and the complexity of any GA depends on the problem that being solved by this GA and its operators, then CONFGA and COFFGA are analysed with different complexities; although they built upon the same algorithm and they are used to solve the same problem (Cloud resource allocation problem), but they are different in their operators their fitness function and termination condition.

INTRODUCTION

This paper emphases the complexity of CONFGA and COFFGA [1], which are adapted for the problem: multi-capacity vector bin packing problem in application to Cloud resource allocation [2], [3], [4], [5].These algorithms are based on the sequential GA [6],[7],[8] and the detailed description of the developed algorithms in [1] is discussed in Section 2.1 and Section 2.2. The first population is generated randomly, and the fitness function is built upon the decision that was generated using fast 2-D Vector Bin Packing (VBP) heuristics: 2-D Next Fit heuristic (2-D NF) in CONFGA, and 2-D First Fit (2-D FF) in COFFGA. The design of the novel algorithms for solving the 2-D VBPP in application to Cloud resource allocation using Combinational order GA (section (2.3)5.4.1),

Previous genetic research in Cloud resource allocation and vector bin packing has used the heuristic for initializing a pool of individuals [9], [10], [11]. It also used complex encoding for representing these individuals, consequently suffering from extra difficulties in crossover and the permutation operations [9], [10], [11]. The approach of Combinatorial Ordering wrapping heuristic Genetic Algorithm in (Section) ; the heuristic is used as a part of the objective function skipping over the complexity of encoding and crossover of previous research[11],[12],[13].

One of the most important characteristics of these algorithms (CONFGA and COFFGA) is the way of encoding; it is abstracted to 1-D array and solved as a combinatorial ordering problem with an intelligent fitness function.

Section 3 describes the research methodology of analysing the complexities of CONFGA and COFFGA. Section 4 shows the experiments that are required to analyse the relation between the average number of generations and the number of Virtual Machines (VMs) in the problem. Section 5 gives the main conclusions of this research.

BACKGROUND

In terms of Cloud resource allocation there is a set of VMs, each one is a vector of different resources, such as CPU and Memory. Since Cloud computing is based on full virtualisation, then Cloud Resource allocation aims to consolidate the given VMs into minimum number of Physical resources (PMs) with the goal of minimizing the resources wastage and number of the required PMs [1],[3],[5],[6]. There is different ways to deal with Cloud resource allocation problem; some ways use multi-capacity VBP heuristics for scheduling and virtual machine packing such as NF, FF, FFD and Permutation(PP) [14], [15], [16]; others are modified natural based solution to deal with Cloud resource allocation problem such as GA and Ant-colony[11],[12],[17].

Basically, the GA is an iterative procedure which borrows the concept of survival of the fittest individual from the Darwinian theory of natural selection. The GA is able to solve complex problems depending on simulating the natural evolution of these problems. By choosing a suitable representation of the given problem and emulating biological selection and reproduction techniques the GA can effectively search a large problem domain. It has been pointed that the classical GA performs poorly when applied to BPPs [7], [18], [19]. Therefore an adapted GA is needed for use with BPPs. Bin packing is an optimisation problem which seeks to minimise the number of bins used to pack a group of items, and great effort has been made to find an optimal way of ordering the items and finding relations between the items' requirements in the VBPP.

Design of Combinatorial Ordering Heuristic in Wrap Genetic Algorithm

This approach has been developed to treat vector bin packing as an optimisation ordering problem, then using an adapted genetic wrapping heuristic algorithm to solve it [1]. It uses the cooperation of a GA with traditional multi-dimension packing heuristics to produce packing solutions under multiple constraints. The GA evolves to find a new packing solution. The decision is configured by finding the best VM order to be packed by an heuristic objective function. The packing solution catches the minimum number of servers and least resource wastage; finding the best order is achieved by evaluating the chromosomes using a fast multi-dimensional heuristic as an objective function. On the other hand, using the heuristic as a part of the objective function will be useful in constructing the packing solution across the available servers. In the box below the proposed hybrid Multi-capacity Combinatorial Ordering GA Procedure is shown in more detail.

Multi-Capacity Combinatorial Ordering GA Procedure

```
1   Initially, $J$=0 ; $VMs$=0; $T$=0; // number of jobs; number of VMs; termination condition
2   Initially, OptSolution = NULL;
    //To host a number of jobs with multi-capacity demands in hosted servers:
3   Initialize a descriptive file that identifies *number of servers*, *capacity of the servers' resources*, *number of jobs* [services],
    and the *requirements of each job* [service].
4   Call INITIALIZE VMs
5   Call FIND PLACEMENT USING COMBINATORIAL ORDERING GA
6   Call PLACE  VMs
```

```
//INITIALIZE VMs
1   While $J$ < *number of jobs*: // from descriptive file
2   Begin
3       $J$ = $J$+1;
4       Encapsulate $Job_J$ into a proper $VM$.
5       $VMs$= $VMs$+1;
6   End
```

```
//FIND PLACEMENT USING COMBINATORIAL ORDERING GA
1   Initiate *Termination Condition*, *Pool Size;*
2   $ChromLength$= $VMs$;
3   $Pool$ = $N$;
4   While $T$ < *Termination Condition*:
5   Begin
6       $N$ = *Pool Size*;
7       $T$ = $T$+1;
8       While $N \neq 0$: //Find Fitness for all Pool Chromosomes
9       Begin
10          Fetch *servers' capacities* and *VM's requirements*;
11          Pack the given $VMs$ order using Multi-capacity vector bin packing heuristic;
12          Use packing function to convert the chrome VMs order to PackingSolution;
13          Calculate the chromosome fitness using the fineness function in (5.2.3);
14          $N$ = $N$ – 1;
15          bestSolution = findBestOrder(Pool);
16      End
17      For $i$=1..*PoolSize*/2; // apply selection and crossover
18      Begin
19          [chrom1, chrom2] = Mini-Roulette; (5.2.4);.
20          [chd1, chd2] = Crossover(chrom1, chrom2); (5.2.5);
21          Save(newPool, chd1, chd2);
22      End
23      Pool=newPool;
24  End
```

```
//PLACE VMs
1   If (OptSolution=NULL or *fit*(OptSolution)<*fit*(BestSolution)
2       OptSolution = BestSolution;
3       OptPacking = PackingSolution(BestSolution);
4       Apply OptPacking (VMs, Servers);
```

Problem Modelling and Algorithms Development

The procedure describes the way to apply the proposed Combinatorial Ordering GA to the Cloud resource allocation problem. It is divided into three main functions: INITIALIZE VMs, FIND PLACEMENT USING COMBINATORIAL ORDERING GA, and PLACE VMs. Each function is dedicated to a specific purpose.

INITIALIZE VMs describes the initialization of the VMs according to the input workload of each job of the given workload and should be encapsulated into a proper VM according to the job requirements.

FIND PLACEMENT USING COMBINATORIAL ORDERING GA function produces the deploying decision of the pre-initialized VMs. It presents the proposed Combinatorial Ordering GA that wraps a fast heuristic as an objective function to make the optimal packing solution. It starts with creating a pool of N chromosomes; each chromosome comprises a random VM order.

The length of each chromosome is set to the number of VMs created for the jobs given in the workload. The chromosomes will be evaluated using the objective function of a fast multi-capacity heuristic according to the given chromosome order; each VM of the given chromosome is an index to N capacities associated to this VM. The packing solution will be generated for each chromosome based on the VMs' capacities and the VMs' order using a multi-capacity vector bin packing heuristic, and then a fitness value will be given to the chromosome using the fitness function developed in [1] The output of this function will be a chromosome's fitness function and chromosome packing solution. As with any other GA this GA will be evolve until reaching the termination condition.

Finally, the PLACE VMs function deploys the packing solution associated with the best solution generated by the previous function onto Cloud servers. All of the related GA functions including chromosome encoding, selection, mutation, fitness function, and termination condition are described in the sections below.

Two algorithms, CONFGA and COFFGA, were developed in [1]using the above procedure. CONFGA uses Next Fit (NF algorithm is the quickest algorithm among the other heuristic as it requires O(N) time [20]) as a packing solution governor, whereas COFFGA employs the multi-capacity FF heuristic as a packing governor instead. NF is able to deal with D-capacity requirement items instead of one requirement items. The FF Algorithm is also an approximation heuristic that is used with multi-capacity vector bin packing problems, but it removes the restriction of NF as it allows the current item to be packed in any non-empty bin which can accommodate the item[18], [21]. It is also known as a fast heuristic with O(N logN) time complexity, where N is the number of items to be packed [22], [23]. The algorithms (CONFGA and COFFGA) were implemented as a group of strongly connected functions. Some of these functions are already present in LibGA [6],[7], such as pool initialization, crossover and mutation. Some others are developed in [1], for example: objective function, chromosome encoding, main function, VMs initialisation from input files, and Packing function.

RESEARCH METHODOLOGY

Since CONFGA and COFFGA are new algorithms, the time complexity of both were analysed. This section analyses the complexity of COFFGA and CONFGA algorithms and identifies the parameters that affect the performance of these algorithms. Given that the complexity of the GA depends on the problem being solved by this GA, as well as the operators of the GA itself; therefore, the time complexity of both algorithms was conducted by analysing the key aspects of theses algorithms in terms of the computational complexity. The complexity of COFFGA and CONFGA are analysed individually CONFGA and COFFGA were implemented using a number of functions and operators that work in the LibGA development package. All these functions and operators were analysed in details below:

Chromosome encoding

In the problem of Multi Capacity Cloud Resource allocation, each Chromosome is encoded as a string of integers that represents a random order of VMs. Each VM in the chromosome is an index to M capacities associated to this VM. In this research, two capacities are considered. The consolidation decision will be generated using one of the algorithms COFFGA or COFNGA. The main parameters of the given problem are:

N: the number of VMs; number of inputs; the length of the individual

Ts: the Total number of servers generated by the fitness function.

P: is the population size
G: is the number of generations

The length of each chromosome equals to the number of VMs need to be deployed over the available servers. The time for chromosome encoding mainly depends on the number of VMs (N), it does not exceed the O(N), the complexity of a population of chromosomes is equal to O(PN), N is constant and P also is constant; therefore O(PN) is also a constant. Both CONFGA and COFFGA have the same complexity of encoding.

Fitness Function

The quality of individuals is evaluated by fitness functions; the developed fitness function comprises a 2-D VBP heuristic as packing governor. The Vector bin packing problem is NP hard, furthermore the 2-D the vector bin packing problem is known to be APX-hard which means that there is no asymptotic PTAS for the problem, unless P = NP [24], [25]. The packing decision will be generated for each chromosome using a 2-D vector bin packing heuristic that is encapsulated in the fitness function. The packing decision is a set of vertices $\{(VM_5, S_1), (VM_2, S_1), (VM_N, S_2), (VM_7, S_2), \ldots\}$. The developed fitness function is identified in formula 1 [1]:

$$f(O) = \mathrm{Ts} * \mathrm{R}_{\mathrm{tot}}, \quad (1)$$

Ts is generated by the 2-D VBP heuristic, consequently the time of the fitness function of COFFGA is different from CONFGA. Since the former uses the 2-D FF in its fitness function, the time of the 2-D FF is O(N log N) [25],[1164]. Whereas, the latter algorithm uses 2-D NF heurist to generate the total number of required servers. NF is a very fast heuristic that works with time equal to O(N) [24], [25]. In regard to the time complexity of $\mathrm{R}_{\mathrm{tot}}$ it has the same complexity for CONGA and COFFGA that is equal to O(Ts^2), depending on the following formulas:

$$\mathrm{R}_{\mathrm{tot}} = \mathrm{R}_{\mathrm{cpu}} + \mathrm{R}_{\mathrm{MEM}}, \quad (2)$$

$$R_{\mathrm{CPU}} = \sum_{\mathrm{i}=1}^{Ts} \frac{\mathrm{SRCi} - \sum_{\mathrm{k}=0}^{\mathrm{a}} \mathrm{C}_{\mathrm{k,i}}}{\mathrm{SRCi}}, \quad (3)$$

$$\mathrm{R}_{\mathrm{MEM}} = \sum_{i=1}^{Ts} \frac{\mathrm{SRMi} - \sum_{\mathrm{K}=0}^{\mathrm{a}} \mathrm{M}_{\mathrm{k,i}}}{\mathrm{SRMi}}, \quad (4)$$

Finally, the time of the COFFGA fitness function is (O(N log N)+O(Ts^2)), and the time complexity of the CONFGA fitness function is (O(N)+O(Ts^2)).

Selection

The proposed algorithms use the adopted mini roulette strategy; roulette-wheel selection scheme is common in the traditional GA. A probability, $P(i)$, is used to decide the selection operator. The mini-selection roulette is similar to the classic roulette selection, where each individual is assigned a slice of a circular roulette wheel with $P(i)$, $P(i) = \frac{\acute{F}(\iota)}{\sum_{j=1}^{P} \acute{F}(\jmath)}$, but the size of the slice of the roulette wheel is, the exception in the mini-selection roulette wheel strategy is that chromosomes are assigned a fitness$\acute{F}(\iota)$, which is inversely proportional to the total fitness function of the given pool, $F(i)$. It is equal to: $\acute{F}(\iota) = \frac{\sum_{j=1}^{P} F(j)}{F(i)}$. Accordingly, the time complexity of the mini roulette section algorithm equals O(P).

Crossover

The Crossover operator that was mainly used with the developed algorithms is Order1 crossover, which is abbreviated to OX [27]. To analyse the complexity of OX, the procedure will be described to support the time complexity that is assigned to OX.

The input is a pair of parents, P_1 and P_2, which will be used to generate two new individuals, $C_1$and C_2. The algorithm randomly assigns two cutting sites i and j in P_1, here i = 4 and j = 6. Then, the substring $P_1(i) \cdots P_1(j)$ is copied into the new individual with index of C_1 (i) $\cdots$ C_1 (j). Then, P_2 is swept circularly from j + 1 onward to complete the missing holes of C_1.on the other hand, C_1 is also filled circularly from j + 1. The example in Figure 18 shows how OX creates the first child C_1. The other child, C_2, will be generated by exchanging the roles of P_1 and P_2.The whole procedure is implemented in O(N) [28]. This crossover will be applied to the best half of individuals in the given pool.

Rank	1	2	3	4	5	6	7	8	9
				i=4		j=6			
$\mathbf{P_1}$	1	3	2 |	6	4	5 |	9	7	8
$\mathbf{P_2}$	3	7	8 |	1	4	9 |	2	5	6
$\mathbf{C_1}$	8	1	9	6	4	5	2	3	7
$\mathbf{C_2}$	2	6	5	1	4	9	7	8	3

Figure 1: Order 1 Procedure Applied to One Parent

Termination Condition

The termination criterion is a critical parameter that affects the running time and the final decision of the developed algorithms. From the experiments, it has been concluded that, the number of generations for the multi-capacity Cloud resource allocation problem depends on the number of VMs and the corresponding requirements of these VMs from one side. From the other side, it depends on the randomness of the pool generation. Since the problem is a combinatorial order problem, the initial pool of individuals greatly affects the speed of algorithms convergence.

There are two termination conditions that have been used. One is that the GA process will be terminated when the chance of improvement equals to '0', then after 50 generations the algorithm will converge. The evolution procedure will be repeated until the best solution shows no further improvement. This termination does not work effectively within CONFGA, but it can work with COFFGA. The second termination condition is by specifying a maximal number of evolution generations, and then evolution will run for a long time, but might get better solutions. This termination condition was specified by making a number of experiments in Section 4.

EXPERIMENTS AND RESULTS

To identify the termination condition for both algorithms (CONFGA and COFFGA) about 200 experiments were done with different problem sizes from (20-340) VMs. These experiments are conducted with a goal of identifying the relationship between the number of generations for convergence and the size of the input. It is useful to draw a mathematical relationship between maximum number of required generations and the size of the input. Identifying the number of maximum generations over different problem instances is a clever way to accelerate the tested algorithm: it prevents unnecessary generations that algorithms spent without improving the minimum fitness value. Therefore, mathematical relationship helps to find a suitable termination condition for each of the tested algorithms, and then finding the complexity over these conditions. All experiments were carried out using the data developed in [1] and the GA parameters as in table 1

Table 1: The Chosen Parameters for CONFGA and COFFGA

Genetic Parameters	Magnitude
Crossover probability	1.0
Population Size	75
Type of crossover	Order or asexual
Selection type	Mini Roulette

Analysing the Relation between Number of Generations and the Problem Size over CONFGA

Experiments were carried out with the goal of identifying the relationship between the number of generations and the minimum fitness values within CONFGA. CONFGA was tested over 26 problem instances for three times. Table 2 shows the number of generations over three runs. It also specifies the minimum, maximum and average fitness values for the problem instances. It shows which generation related to which fitness value.

The table's data is analysed in detail: it illustrates that CONFGA needs a different number of generations to reach to the minimum fitness value for different runs of the tested problem instance over the all problem cases. Clearly, there is a considerable difference in the minimum fitness value for the same problem instance over different runs; correspondingly the resultant packing solutions of these fitness values are varied. For example the problem of size 280 VMs comes with 21,390 as the maximum resultant fitness value of CONFGA over the three runs, and 14,058 as the minimum fitness value for the same problem instance over these runs. The corresponding number of required servers of the given fitness values are 24 and 26.

In order to analyse the main factors that affect CONFGA's convergence of the given problem, the results are analysed against fitness values the maximum fitness value and the minimum fitness value; and examine which one comes with minimum generations, which one needs the maximum generations, and how it changes across different problem sizes.

In some problem instances the maximum fitness value accompanies the minimum number of generations, the minimum fitness value comes with the maximum number of generations, and the third run needs a number of generations between the maximum and the minimum generations; this is reported in Table 2 in the fields of white background. It cannot be blamed that CONFGA minimum fitness values come with the maximum number of generations, because in some tested cases the minimum fitness value comes with minimum number of generations as can be seen in the rows with a dark grey background in Table 2. In other tested cases, the maximum fitness values do not accompany the minimum number of generations, or the third run requires the maximum generations: this situation is recognised with the light grey background. The main conclusion of this analysis is that the randomness of the pool plays a crucial role in the performance of the CONFGA as it determines the convergence and the resultant packing solution.

To draw out the relationship between the problem size and the number of the required generations, the average number of generations is compared against the problem size for all the tested problems. Figure 2 depicts the relationship between the average number of generations and the number of VMs in the input that were tested using CONFGA. It can be seen from Figure 2 that the number of generations is not exponential. In fact, the maximum generations in all cases are less than a second order polynomial in regards to the chromosome length.

Finally, after analysing all the tested problem instances, the maximum number of generations is defined as: maxgen=N^2, where N is the length of the chromosome or, in other words, the number of VMs. It is obvious that the length of the chromosome is significantly affects the number of generations that CONFGA algorithm needs to produce its solutions, where $\text{maxgen} \propto (\text{problem space size})^2$.

Table 2: Relation between Number of Generations and Minimum Fitness Value in CONFGA

No-of-VMs	No-of-Generations			Fitness Values		
	G-3rd Run	G.-Max-FV	G-Min- FV	Average	Maximum	Minimum
20	10	15	10	473	474	471
30	20	15	20	626.6	636	608
40	50	50	100	186.6	196	168
50	90	65	150	368.3	465	235
60	75	145	175	1108	1248	906
70	225	200	90	1615.6	1652	1568
80	200	145	285	1985	2264	1115
90	295	300	220	949.3	1120	816
100	495	475	450	1404	1719	990
110	275	445	485	1993.3	2370	1560
120	690	870	685	1818.6	2167	1386
130	830	290	840	2164.3	4543	710
140	775	790	895	1334.6	1518	1100
150	1125	975	1145	2216	3240	1500
160	940	960	940	2834	4121	1937
170	1045	1130	1080	5049.3	5614	4144
180	1175	556	880	6990	7665	5880
190	1175	300	1115	6481.3	9184	3300
200	1465	1315	1185	3855.3	3968	3662
220	1410	1385	1115	6762	7686	6174
240	1890	1500	1905	8873	10298	5719
260	1995	1800	1910	13797	14889	12810
280	1920	1790	1985	18677.6	21390	14058
300	2400	2000	2440	15600	16872	14568
320	2360	2470	2215	21892	22048	21710
340	2885	2900	2490	18615.3	24580	13284

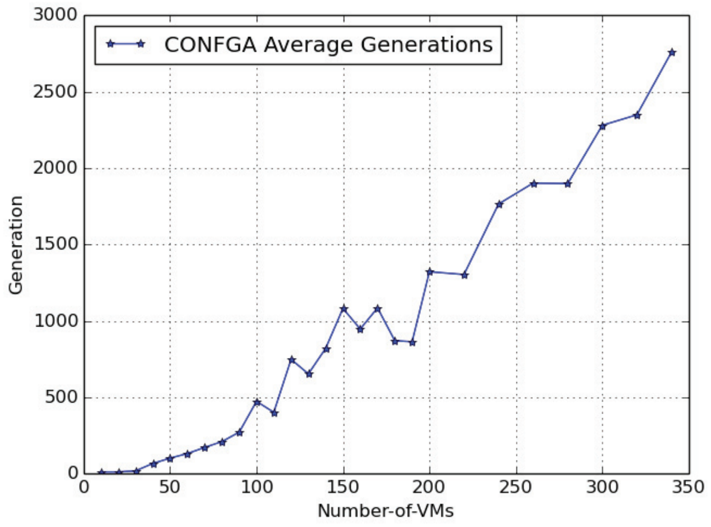

Figure 1: Relationship between the Average Number of Generations and the Number of VMs in CONFGA over Different Problem Instances

Analysing the Relationship between Number of Generations and Problem Size over COFFGA

This section discusses the relationship between the number of generations and the performance of COFFGA. This relationship is extremely important to consider as it helps to specify the termination condition for this algorithm. Defining the termination condition is required to prevent unnecessary running time.

COFFGA was tested over the 26 problem instances and the results of three runs are listed in Table 3. The results are organised in a way that specify maximum, minimum and average fitness values over the three runs. It also shows the number of required generations in the three runs and identifies the generations that related to the minimum, maximum and the third run. By analysing the data in Table 3, it is evident that there is no significant difference in the fitness values over the three runs of the same problem instance. Correspondingly, the number of required servers from the resultant packing solutions for these fitness values is identical.

By analysing the number of generations of minimum, maximum and third run fitness values, it has been concluded that in most tested cases the maximum fitness value goes together with the minimum number of generations; the minimum fitness value accompanies the maximum number of generations; and the third run needs a number of generations between the maximum and the minimum generations. The main conclusion behind this observation is the randomness of the pool affects the performance of COFFGA, but the fitness function of COFFGA is strong enough to be the driver of the convergence. The randomness of the initial generated pool affects the number of required generations, but it does not seriously affect the resultant minimum fitness value. Thus the number of servers that result from the COFFGA packing solutions is rarely varied over all the problem instances.

Given that the relationship between the input size and the number of generations is a vital issue to be identified over the tested COFFGA, the average number of generations is compared against the problem size for 28 problem instances in Figure 3. The relationship between the input size and the number of generations is analysed as linear with exemption of sudden decline in the number of generations. It is obvious from Figure 3 that the average number of generations increases linearly with the size of the problem, but it suffers from unexpected fluctuation in some problems instances. There are two reasons behind this sudden fluctuation: the first one is that COFFGA's performance depends mainly on the values and the combination of vectors in the input data, thus it sometimes converged easily when the combination of the VMs requirements in the input data is not very complex to organize; the second reason is that COFFGA is really fast because of the effectiveness of its fitness function, and therefore it sometimes finds the minimum fitness value with very short time. On the other hand, it cannot be ignored that the size of input clearly affects the performance of COFFGA, as the average number of generations increases with the input size for all problem instances.

Finally, after analysing all the tested problem instances, the maximum number of generations is calculated as: maxgen= min(10n, 3000), where n is the length of the chromosome or the number of VMs. As the number of generations does not exceed 2,000 in all problem instances, the number of required generations in COFFGA mainly depends on the values and the combination of vectors in the input data from one side, and the size of the input data from the other side.

Table 3: Relationship between Number of Generations and Minimum Fitness Value in COFFGA

No-of-VMs	No-of-Generations			Fitness Values		
	G-3rd Run	G.-Max-FV	G-Min- FV	Average	Maximum	Minimum
10	4	4	4	98	98	98
20	5	5	5	246	246	246
30	6	6	6	333.5	357	357
40	19	21	17	76	76	76
50	15	17	25	688.2	760	760
60	33	30	45	400	450	433
70	107	97	107	102	120	72
80	120	85	110	154	96	84
90	135	145	180	195.7	201	184
100	215	120	175	381	396	369
110	195	215	250	150	165	140
120	300	325	500	824	825	715
130	600	450	530	510	440	620
140	655	580	850	414.75	460	407
150	850	780	910	1046.75	1007	996
160	480	375	965	2184	1495	1391
170	575	615	1000	2842	3010	2786
180	800	485	1005	2268	2660	2380
190	1100	850	1160	1417.5	1320	810
200	1400	885	1480	1608	1936	1744
220	1500	870	1745	3026.7	2261	2646
240	1850	1380	1955	5377	5510	4940
260	1115	1360	1940	9769.2	9765	7220
280	1460	995	1800	9152	9812	9416
300	1160	1060	1695	10352	9288	14328
320	1205	1160	1200	10975	12500	13200
340	1440	1200	1520	10231.5	14013	13017

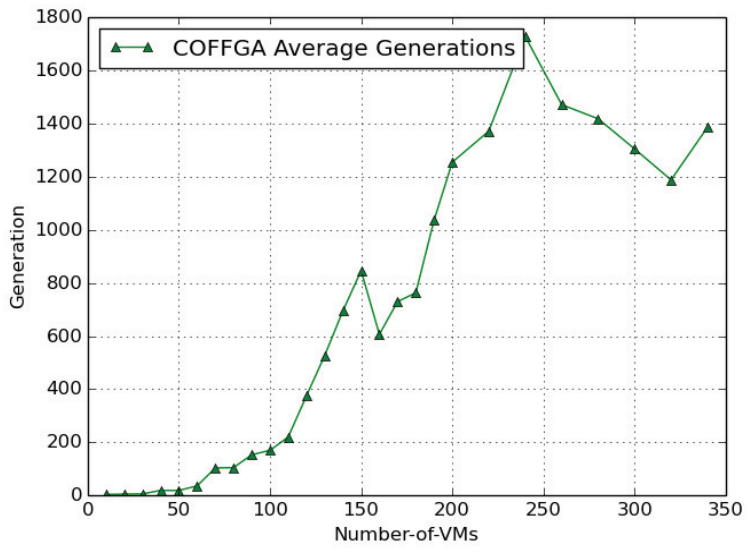

Figure 3: The relationship between the average number of generations and the number of the VMs in the input tested using COFFGA

Results

The maximal number of evolution generations depends on the problem size. It is considered as a main factor to affect the maximal number of generations: the maximum number of generations for CONFGA is calculated as maxgen=N^2(see Figure 2) and the COFFGA maximum generations is identified as maxgen= min(10N, 3000) (see Figure3). In this case, the termination criterion is only related to the problem size, not to the randomness of solutions in each run. It has been used to verify the effectiveness of COFFGA and CONFGA. An important fact that the experiments show is that CONFGA needs more function evaluations than COFFGA in all problem instances (see Tables 2 and 3).

Finally, Table 4 compares the total complexity of the COFFGA and CONFGA, not just for single chromosome, but for the whole population. The main conclusion from the previous analysis is that both of number of VMs (i.e. the length of the chromosome), and the size of the pool have dramatic effects on the overall GA complexity. Even COFFGA has higher complexity than CONFGA, but it easily converges, which makes COFFGA faster than CONFGA.

Table 4: Comparison between CONFGA and COFFGA Complexities for a Population of Chromosomes

Developed GA	Fitness Function	Selection	Population Encoding	Crossover	Max Generation
COFFGA	O(P(N log N+Ts^2))	O(P)	O(P(N))	O(PN)	O(N)
CONFGA	O(P(N+Ts^2))	O(P)	O(P(N))	O(PN)	O(N^2)

CONCLUSIONS

The complexities of the new algorithms (CONFGA and COFFGA) are analysed individually, with the aim of identifying the parameters that affect their performance.

The complexities of CONFGA and COFFGA are conducted by analysing their key aspects in terms of the computational complexity. However the analysis indicates that the complexities of both algorithms depend mainly on: the number of VMs (or the length of the chromosome N), as it directly affects the fitness evaluation of the chromosome; the termination condition, as it impacts on running time and the final solution of the developed algorithms; and also the size of the pool, as it has a dramatic effect on the overall complexity. COFFGA has a fitness function with a higher complexity O(P(N log N+Ts^2)) than CONFGA's fitness function, which is O(P(N+Ts^2)), but COFFGA converges faster than CONFGA. In terms of the relationship between number of generations and problem size for the two algorithms, the maximum for COFFGA is identified as (10N), while CONFGA's maximum number of generations equals(N^2).

COFFGA and CONFGA have different time complexities which depend on the length of the chromosome, the size of the pool, and the termination condition. Although that COFFGA is characterised with a higher complexity than CONFGA, it is still faster than CONFGA because COFFGA easily converges with a fewer number of generations.

REFERENCES

[1] H. Hallawi, J. Mehnen, H. He,“ Multi-capacity combinatorial ordering GA in application to cloud resources allocation and efficient virtual machines consolidation, Future Generation Computer Systems, vol. 69, April 2017, pp. 1–10.

[2] A. Yousefipour, A. M. Rahmani, M. Jahanshahi, Energy and cost-aware virtual machine consolidation in cloud computing, April 2018, Software Practice and Experience 48(4),DOI: 10.1002/spe.2585

[3] K. K.Vivek Jain, Pushpneel Verma, An Analysis on Virtual Machine Migration Issues and Challenges in Cloud Computing, International Journal of Computer Applications , October 2018, ,182(22):25-30,DOI: 10.5120/ijca2018918016

[4] P. Samimia, Y. Teimourib , and M. Mukhtara, “ A combinatorial double auction resource allocation model in cloud computing”, Elsevier, Information Sciences, 2014, available online on: www.else vier.com/locate/ins

[5] H. Hallawi, J. Mehnen, H. He, “A comparison of resource allocation process in grid and cloud technologies”, Journal of Physics: Conference Series, Volume 1032, conference 1 , pp. 1–13, 2018.

[6] ftp://ftp.aic.nrl.navy.mil/pub/galist/src/ga/libga100.tar.Z

[7] L. Corcoran and R. Wainwright, LibGA: a User-Friendly Workbench for Order-Based Genetic Algorithm Research, Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice, pp. 111-117, 1993, DOI10.1145/162754.162828.

[8] L. Corcoran, and R. Wainwright, A parallel island model genetic algorithm for the multiprocessor scheduling problem, SAC, pp, 483-487. ACM, 1994.

[9] D. Wilcox, A. McNabb, K. Seppi, Solving Virtual Machine Packing with a Reordering Grouping Genetic Algorithm, 2011, Congress on Evolutionary Computation (CEC), pp. 362 - 369, 2011. IEEE ISBN:978-1-4244-7834-7, DOI: 10.1109/CEC.2011.5949641,

[10] J. Xu and J. Fortes, Multi-objective Virtual Machine Placement in Virtualized Data Center Environments, International Conference on Green Computing and Communications IEEE, Hangzhou, China, IEEE Computer Society, pp. 179-188, 2010, DOI 10.1109/GreenCom-CPSCom.2010.137.

[11] H. Ravani, H. Bheda, V, Patel, Genetic Algorithm Based Resource Scheduling Technique in Cloud Computing, International Journal of Advance Research in Computer Science and Management Studies, vol.1, pp.168-174, 2013. available online at: www.ijarcsms.com.

[12] C. Joseph, K Chandrasekaran, and R. Cyriac, Improving the Effecieny of Genetic Algorithm Approach to Virtual Machine Allocation, 5th International Conference on Computer and Communication Technology (ICCCT), IEEE, 2014, 978-1-4799-6758-2

[13] S. Aggarwal, R. Garg, and P. Goswami, A Review Paper on Different Encoding Schemes used in Genetic Algorithms, International Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 1, January 2014 ISSN: 2277 128X

[14] L. Dubies, Optimizing Resource allocation while handling SLA violations in Cloud Computing Platform, Proceedings of the IEEE 27th International Symposium on Parallel & Distributed Processing (IPDPS), pp 79-87, DOI: 10.1109/IPDPS.2013.67.

[15] S. Genaud, and J. Gossa, "Cost-wait Trade-offs in Client-side Resource Provisioning with Elastic Clouds", in proceeding of 2011 IEEE 4th International Conference on Cloud Computing, published by IEEE Computer Society, DOI 10.1109.

[16] S.Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, and K. Talwar, Validating Heuristic for Virtual Machines Consolidation, TechReport, Microsoft, 2011, Microsoft Research Silicon Valley, Mountain View, CA 94043

[17] A. Suphalakshmi and S. M, "An intelligent, energy conserving load balancing algorithm for the cloud environment using ant's stigmergic behavior", International Journal of Communications and Engineering, vol. 04, no. 03, March 2012.

[18] E. Browna, R. Sumichrastb, Impact of the Replacement Heuristic in a Grouping Genetic Algorithm, Computers & Operations Research, vol 30, 2003, pp:1575–1593.

[19] M. Quiroz-Castellanosa, L. Cruz-Reyesa, Jose T.Jimenezb , S. Claudia Gómez, H. Huacujaa , and A. Alvimc, A Grouping Genetic Algorithm with Controlled Gene Transmission for the Bin Packing Problem, Computers & Operations Research, vol:55, pp: 52-64, 2015.

[20] H. Ravani, H. Bheda, V, Patel, Genetic Algorithm Based Resource Scheduling Technique in Cloud Computing, International Journal of Advance Research in Computer Science and Management Studies, vol.1, pp.168-174, 2013. available online at: www.ijarcsms.com.

[21] R. Klein, A. Scholl, and C. Jurgens, Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. Computers and Operations Research, 1997, vol. 24, no. 7, pp. 627–645.

[22] W. Leinberger., G. Karypis, and V. Kumar, Multi-capacity bin packing algorithms with applications to job scheduling under multiple constraints, Proceedings International Conference on Parallel Processing 1999, IEEE, pp. 404 - 412, 0-7695-0350-0, DOI: 10.1109/ICPP.1999.797428.

[23] R. Masson, T. Vidal, J. Michallet, P. Penna, V. Petrucci, A. Subramanian, H. Dubedout, An Iterated Local Search Heuristic for Multi-Capacity Bin Packing and Machine Reassignment Problems. Expert Systems with Applications. vol 40, pp. 5266 - 5275, 2013. DOI: 10.1016/j.eswa.2013.03.037

[24] Rina Panigrahy , Kunal Talwar , Lincoln Uyeda , and Udi Wieder, Heuristics for Vector Bin Packing, Microsoft Research Silicon Valley

[25] Gerhard J. Woeginger, There is no asymptotic PTAS for two-dimensional vector packing, Information Processing Letters 64 293-297, 1997.

[26] F. C. R. Spieksma, A Branch-and-Bound Algorithm for the Two-Dimensional Vector Packing Problem, Computers Ops Res. vol. 21, no. 1, pp. 19-25, 1994.

[27] Jens Gottlieb, Permutation-Based Evolutionary Algorithms for Multidimensional Knapsack Problems, SAC'O0 March 19-21 Como, Italy, pp 408-414

[28] Hongmei Hea, Ondrej Sýkoraa , Ana Salageana , Erkki Mäkinen Parallelisation of genetic algorithms for the 2 page crossing number problem, Journal of Parallel and Distributed Computing, vol 67, pp. 229 – 241, 2007.