Memetic Comp. (2018) 10:29–42
https://doi.org/10.1007/s12293-017-0232-7

REGULAR RESEARCH PAPER

Guided genetic algorithm for the multidimensional knapsack problem

Abdellah Rezoug[1] · **Mohamed Bader-El-Den**[2] · **Dalila Boughaci**[3]

Received: 24 March 2016 / Accepted: 28 April 2017 / Published online: 10 May 2017

Abstract Genetic Algorithm (GA) has emerged as a powerful method for solving a wide range of combinatorial optimisation problems in many fields. This paper presents a hybrid heuristic approach named Guided Genetic Algorithm (GGA) for solving the Multidimensional Knapsack Problem (MKP). GGA is a two-step memetic algorithm composed of a data pre-analysis and a modified GA. The pre-analysis of the problem data is performed using an efficiency-based method to extract useful information. This prior knowledge is integrated as a guide in a GA at two stages: to generate the initial population and to evaluate the produced offspring by the fitness function. Extensive experimentation was carried out to examine GGA on the MKP. The main GGA parameters were tuned and a comparative study with other methods was conducted on well-known MKP data. The real impact of GGA was checked by a statistical analysis using ANOVA, *t*-test and Welch's *t*-test. The obtained results showed that the proposed approach largely improved standard GA and was highly competitive with other optimisation methods.

✉ Abdellah Rezoug
abdellah.rezoug@gmail.com

Mohamed Bader-El-Den
mohamed.bader@port.ac.uk

Dalila Boughaci
dalila_info@yahoo.fr

1 Department of Computer Science, University Mhamed Bougara of Boumerdes, Boumerdes, Algeria

2 School of Computing, University of Portsmouth, Portsmouth, United Kingdom

3 Department of Computer Science, University of Science and Technology Houari Boumediene, Algiers, Algeria

Keywords Genetic algorithm · Hybrid heuristic · Memetic algorithm · Multidimensional knapsack problem · Core concept · Guided genetic algorithm

1 Introduction

Genetic Algorithm (GA) was first introduced more than four decades ago and it is still widely used in several research applications. GA is mainly used as a stochastic method for solving combinatorial optimisation problems, especially NP-hard problems. In many real-world problems, exact methods fail to find good solutions in a reasonable time. GA has been applied successfully in many real applications as well as various traditional combinatorial problems [3,20,30]. Originally, GA was inspired by the biological evolution of living species. Starting with a randomly generated initial population of a set of individuals, GA aims to improve the quality of the successive generations by applying several genetic operators, e.g. crossover and mutation. It is known that GA is relatively simple to implement compared to several other methods, but, is it really able to provide the best solutions? In reality, GA is a stochastic process, so there is no guarantee of optimality, only a large number of generations and individuals can increase the confidence in the obtained solution [28].

Several variants of GA have been proposed during the past few decades, the main aim of many of these variations is to improve the performance of GA and accelerate its convergence in finding an optimal solution. The vast majority of these ideas are either articulated on changing the GA operators such as: crossover and mutation (e.g. one-point, two-point, cut and splice, three parents, uniform, flip bit, Boundary, non-uniform, uniform, etc.), or based on modifying the GA's evolutionary behaviour, such as: Hybrid GA [14,26], Parallel GA [29], Genetic Programming [4,5,22], etc. An extensive survey of the different variations of GA is available in [8]. The focus here is only on the methods that are related to the guided GA concept.

This paper presents a memetic algorithm—named Guided GA (GGA)— for the Multidimensional Knapsack Problem (MKP). GGA is a memetic algorithm that exploits prior knowledge about the problem data as an intensification strategy to drive the GA evolutionary process of optimisation toward promising areas of the solutions space.

GGA is inspired by two main concepts: the first is the Proximate Optimality concept which assumes that in most cases, the best solutions have a similar structure, in other words, part of the solution may appear in all the best individuals; the second is the *Core* Concept for the Multidimensional Knapsack Problem CCMKP [23,27] that provides a mathematical model for ordering the items in MKP based on a compromise between their weights (costs) and their values. GGA uses the output of the CCMKP model as an additional guide for the GA's evolutionary process. The CCMKP output is used at two stages of the evolutionary process the initialisation and evaluation (fitness function) stages.

The paper is structured as follows: Sect. 2 provides a definition of MKP and the *Core* concept for MKP (CCMKP). Section 3 gives an overview of the literature review related to the GGA. The proposed algorithm GGA is introduced in Sect. 4. The experimental setup and the parameters tuning are given in Sect. 5. Section 6 presents the conducted experiments and the obtained results. Conclusions and final remarks are drawn in Sect. 7.

2 The multidimensional knapsack problem

This section presents the MKP mathematical model adopted in this work and provides a quick overview on the *Core* concept for MKP [23].

2.1 Problem definition

The MKP is composed of n items and a knapsack with m different capacities c_i ($i \in \{1,\ldots,m\}$). Each item j ($j \in \{1,\ldots,n\}$) has a weight w_{ij} on each capacity i of the knapsack and a value p_j. The goal is to pack the items in the knapsack so as to maximise the overall value without exceeding the capacities of the knapsack. The MKP model can be represented by the following integer program:

$$\textbf{Maximise}: \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\textbf{Subject to}: \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad i \in \{1\ldots m\} \tag{2}$$

$$x_j \in \{0,1\} \qquad j \in \{1\ldots n\} \tag{3}$$

A feasible solution X for MKP represents the selected items to be packed. A decision variable x_j represent the item j and is binary where $x_j = 1$ means that item j is packed, and $x_j = 0$ means that item j is not packed in the knapsack. w_{ij} represents the weight of item j on dimension i.

2.2 The core concept for MKP

The *Core* concept for solving combinatorial and linear programming problems was specifically applied for the knapsack problems by Balas and Zemel [6] and later it was extended to the MKP by Puchinger et al. [23]. The CCMKP calculates an efficiency (score) for each variable (item) in the MKP, the efficiency reflects the expected added value of a variable to the final solution; a high efficiency indicates that the variable is likely to appear in the optimal solution, low efficiency indicates that the variable is unlikely to appear in the optimal or near-optimal solutions, while average efficiency indicate that there is uncertainty about the variable's added value. Therefore, after being sorted decreasingly according to CCMKP efficiency, the variables are divided into three sets. The variables with high efficiency are fixed to 1 whereas those with low efficiency are fixed to 0 and those with close efficiency represent the *Core*. Consequently, the *Core* concept allows reducing the original problem into only the *Core* problem.

The *Core* concept is based on an efficiency measure function. The aim is to assign an efficiency value to each variable, according to its significance in producing the optimal solution, in such a way to promote those having the high values and low weights. Several efficiency measures have been used as approximations of the efficiency function, for example, simple efficiency (e_j^{simple}) [12], scaled efficiency (e_j^{scaled}), Senju & Toyoda (e_j^{st}) [27] and general efficiency ($e_j^{general}$) [18] as shown in Eqs. 4, 5, 6 and 7–8 respectively.

$$e_j^{simple} = \frac{p_j}{\sum_{i=1}^{m} w_{ij}} \tag{4}$$

$$e_j^{scaled} = \frac{p_j}{\sum_{i=1}^{m} \frac{w_{ij}}{c_i}} \tag{5}$$

$$e_j^{st} = \frac{p_j}{\sum_{j=1}^{m} w_{ij}(\sum_{l=1}^{n} w_{il} - c_i)} \tag{6}$$

$$e_j^{general} = \frac{p_j}{\sum_{i=1}^{m} r_i w_{ij}} \tag{7}$$

$$r_i = \frac{\sum_{j=1}^{n} w_{ij} - c_i}{\sum_{j=1}^{n} w_{ij}} \tag{8}$$

Where e_j : efficiency of item j; p_j : value of item j; w_{ij} : weight of item j on dimension i; c_i : capacity of knapsack on dimension i and r_i : coefficient.

3 Related works

There are several methods related to the guided GA concept in literature, that have been applied to a wide range of applications. For solving the Course Timetabling Problem, Yang and Jat [33] used a memory denoted MEM to record useful information to guide the GA process and improve its performance. MEM is a list of limited size, in which a list of room and time slot pairs is recorded. This information is integrated into the crossover operator of the proposed guided GA. Other researchers used an external structure to guide GA such as [1]. Another approach for guiding the GA is through the use of approximate probabilistic models. In [9] GA is augmented with an approximate probabilistic model to guide the crossover and mutation operators. The probabilistic model is used to estimate the quality of candidate solutions generated by the traditional crossover and mutation operators. It also evaluates the quality of candidate solutions. This estimation enables the crossover and mutation operators to generate more promising solutions.

Specific characteristics of the addressed problem are used to guide the GA search process. The Process Discovery through a Genetic algorithm *ProDiGen* [31] is a GA that adopts three characteristics of the Process Discovery. The method calculates the precision, simplicity and completeness values of the treated model (i.e. log files of the information system process). These values are integrated into the expression of the GA fitness function to guide the optimisation process. A slowdown-guided GA for the Job Scheduling Problem is proposed in [15]. The proposed model is based on the estimation of the execution slowdown of the tasks which is used to guide the GA search process, the slowdown estimation is used to express the fitness function.

In other versions, a subset of genetic operators is guided. The proximate optimality principle assumes that good solutions have a similar structure. Based on this principle, the guided mutation proposed in [34] uses a probability model inspired by Estimation of Distribution Algorithms (EDA) mutation operator. The generated offspring by this operator is constructed using three components, the best solution found so far, a dynamic probabilistic model and a probability β. This allows conducting the searching process in promising areas. A guided crossover operator has been proposed in [24]. The crossover operator works by using guidance from all members of the GA population to select a direction for exploration. The first parent is selected by the selection operator. To select the second parent, a metric named $Mutual\_fitness$ is calculated for all the other chromosomes. The chromosome which has the maximum value is selected. One offspring is generated by crossing the parents in a point chosen randomly such that the offspring resulting is the best.

The guidance methods in these GA variants are specific to the addressed problems, they do not propose a formal way to extract the guidance information or are integrated to the optimisation process. Some approaches incorporate a partial guidance using genetic operators.

4 The guided genetic algorithm GGA

The algorithm in this paper is motivated by the observation that in many optimisation real-world problems, some prior information about the components/patterns that are likely to appear in the good solutions could be known. For example, in MKP, it is possible using linear relaxation or the optimal fractional solution [11], to predict some of the items that are likely or unlikely to appear in the good solutions. This study proposes a method for using such prior information as an additional guide for the GA evolutionary process for the MKP problem. By guide, we mean any structure external to GA, which maintains its original composition and is used to drive its search process. This can be through a subset of operators, in order to accelerate the search process and improve the speed of convergence. This section aims to describe the GGA components.

4.1 Chromosome design

The population is composed of a finite number of chromosomes. A chromosome represents a feasible solution to the problem (MKP). As mentioned before, the target in the MKP is to define the subset of items that maximises the total profit. The GGA chromosome consists in the set of items to be added to the knapsack. GGA uses the integer representation, where each gene presents an item ID. The items are coded as integer numbers. A chromosome is formed only by the number of items that it contains. This representation allows reducing the size of the processed data (Fig. 1).

4.2 Guiding information

The guiding information is based on the work by [23]. The items are sorted decreasingly according to their statistical efficiencies e_j based on the value and the weight (e_j^{simple}, e_j^{scaled}, e_j^{st} or $e_j^{general}$). In simple words, the items are sorted based on how likely each item is to appear in high performing individuals, the items at the top of this list are likely to be selected while the items at the bottom are unlikely to appear in good solutions. However, it is important to note here that this list is just an estimate and not a predefined part of the solution. It should also be noted that the greedy heuristic [27], as it is only based on the efficiency sorting, is not an effective solution for the strongly correlated problem instances of MKP [17].

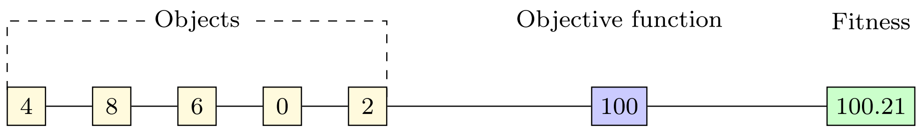

Fig. 1 An example of the chromosome design. The objects (items) packed in the knapsack are represented by their identifiers. The objective function value is calculated by summing the benefit of all the objects while the fitness is calculated according to the fitness function

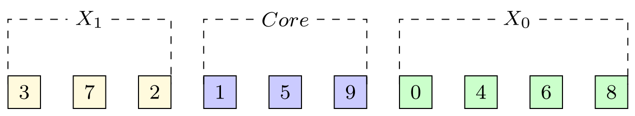

Fig. 2 An example of the guide construction. The objects (items) are sorted according to the efficiency e_j

The sorting operation allows favouring items that have a good compromise (i.e. efficiency) between the average value and the overall weight. The efficiency of an item is high if its value is high while its required global capacity is low. The sorted items are split into three sets (Fig. 2) where the value of each variable is assigned as follows:

- $X_1 : x_j = 1$ The variables have the best efficiency e_j. These variables are most likely to build the best solutions even the optimal solution.
- $Core : x_j =?$ The efficiency values e_j of these items are medium, therefore, it is difficult to predict with confidence whether or not some may appear in the optimal solutions.
- $X_0 : x_j = 0$ The variables have a very low efficiency e_j, in other words, the value is low or the weight is large or both.

The guide is represented by the variables of $X_1 \cup Core \cup X_0$. The sizes of X_1, $Core$ and X_0 are determined as follows: Construct a feasible solution by adding the items in order. The item that makes the solution infeasible represents the centre of $Core$. The size of each part of the guide depends on the size of $Core$. Setting the size of $Core$ defines the size of the other parts.

4.3 Initial population

The GGA algorithm uses a special initialisation process which allows the GA to make use of the prior information available about the items, and in the same time generates a diverse initial population to ensure exploration of the search space. A chromosome is generated from the items of X_1 completed by items generated randomly. In each chromosome, X_1 is integrated with a probability α. If α is set to zero this means that all the items in each individual are selected randomly, while $\alpha = 1$ means that each individual in the initial population contains all the items of X_1. This method allows having an initial population of good quality by integrating X_1 and ensures the diversification by adding the rest randomly.

4.4 Fitness evaluation

In GGA, the objective function is different than the fitness function $f(j)$. The first is the value of a solution relative to MKP problem. It is evaluated according to Eq. 1, while the fitness function is defined in a way to guide the search process of GGA. Different formulations of the fitness function are examined by introducing the efficiency e_j, X_1 and X_0.

A) The fitness function is calculated in the same way as the objective function (Eq. 9):

$$f(j) = \sum_{j=0}^{n} p_j x_j \tag{9}$$

B) The efficiency e_j is introduced in its evaluation according to Eq. 10. Every generation, the fitness value of each chromosome is calculated. The fitness formula allows giving more chance to the chromosome that has a high efficiency to be selected more than the others.

$$f(j) = \sum_{j=0}^{n} e_j p_j x_j \tag{10}$$

C) X_1 and X_0 are introduced in the fitness measuring; the first as a reward and the second as a penalty (Eq. 11). The aim is to reward (respectively penalise) each chromosome according to its similarity with X_1 (respectively X_0). Thus allows, at the same time, increasing the chance for the good chromosome to be selected and decreasing it for the bad one.

$$f(j) = \sum_{j=0}^{n} p_j x_j + reward - penalty \tag{11}$$

Where $reward = s_1 * p_z$, $penalty = s_0 * p_z$, s_1 and s_0 represent the similarity rate with X_1 and X_0 respectively, and p_z is a significant percentage of the average objective function of the generation (in the experiments $p_z = 0.1$ is used).

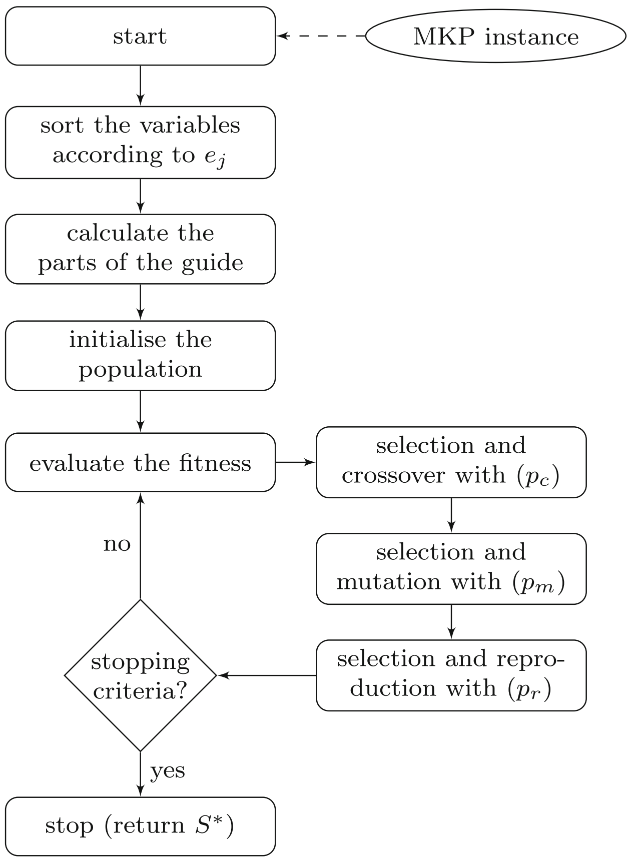
start
MKP instance
sort the variables according to e_j
calculate the parts of the guide
initialise the population
evaluate the fitness
selection and crossover with (p_c)
selection and mutation with (p_m)
selection and reproduction with (p_r)
stopping criteria?
no
yes
stop (return S^*)

Fig. 3 Flowchart of the GGA optimisation process

D) The fitness uses the similarity of the chromosome with X_1 as follows:

$$f(j) = (1 + s_1) \sum_{j=0}^{n} p_j x_j \tag{12}$$

4.5 Genetic operators

GGA uses standard genetic crossover and mutation operators. Tournament selection of size 5 is used as the selection method, the random single point method is applied with a probability p_c. For mutation, the mutation by random multiple point bit flip is applied with the probability p_m. And finally, a reproduction operator copies a subset of individuals with the probability p_r such as $p_c + p_m + p_r = 1$. The pseudo-code of the GGA is illustrated in Fig. 3.

5 Experimental setup and parameters tuning

This section explains the experimental setup and presents the parameter tuning analysis for the GGA algorithm. It is important to note that the concept of guide can be applied to any problem if an effective method for sorting the problem variables exists. For an experimental purpose, and because the chosen sorting method concerns MKP, it is natural to use data from this problem. The test platform is a Toshiba laptop with 4GB RAM capacity and an Intel Core (TM) i5-4200 M 2.5 Ghz CPU. The Java language is used to implement the approach. As for the test data, the well-known *Chu&Beasley* benchmarks from the OR-Library[1] are used.

Table 1 Parameters of the GGA used to perform the experiments

Parameter	Description	Value
ps	Population size	500
ng	Number of generations	500
p_c	Crossover probability	0.2
p_m	Mutation probability	0.7
p_r	Reproduction probability	0.1
α	X_1 integration rate on the initial population	0.9
st	Selection tour	10
nmp	Number of mutation points	3
nbk	Number of best chromosomes kept	5
$nrun$	Number of runs for each instance	30

Due to the lack of standard methods, trial-and-error is the most common used method for parameter tuning in most heuristic-based optimisation algorithms. We conducted several experiments in order to analyse the performance of the GGA algorithm under different parameter values. For this task the subset of instances *OR5×100-0.25* is used. All parameter values are set as shown in Table 1; only the parameter to be measured is changed.

The optimum values are known for almost all the instances composing the *Chu&Beasley* dataset. In this work, the Distance From the Optimum $D.F.O$ and the Average Distance From the Optimum $A.D.F.O$ are measured according to the best-known solutions.

Figure 4a, displays how α may affect the GGA $D.F.O$ by changing its value. Based on the value of α, the $D.F.O$ ranged from 0.0 to 0.9.

Better results were achieved when the value of α was equal to 0.9. This indicates that using more items of X_1 in the initial population individuals improves significantly the quality of the obtained solutions. However, the integration of the whole group (i.e. $\alpha = 1$) reduces the diversity of individuals.

The Average Distance From the Optimum $A.D.F.O$ of the GGA application on the *OR5×100-0.25* instances with different values of ng is given in Fig. 4b. The results indicated that when ng was high $A.D.F.O$ decreased and in some cases GGA found the optimal solution. In the next experiments it is considered that $ng = 500$.

The impact of the population size parameter ps on the GGA performance is given in Fig. 4c. The results show that

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/.

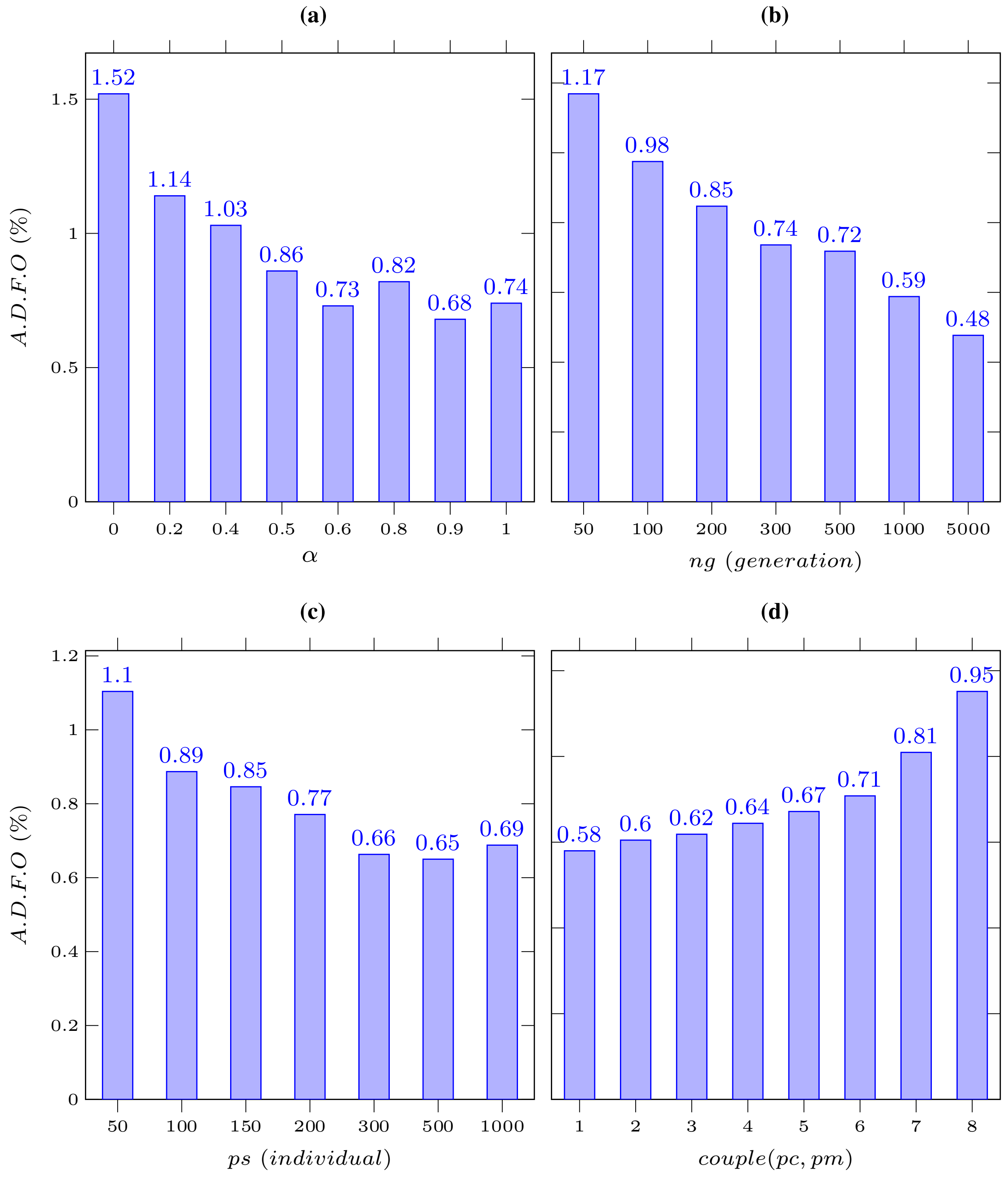

Fig. 4 GGA tuning of four parameters by calculating the average distance from the optimum $A.D.F.O$ of 30 runs on the *OR5×100-0.25* instances. α, number of generation ng, population size ps and the couple crossover/mutation probability (pc, pm) are illustrated in **a**, **b**, **c** and **d** respectively. (pc, pm) value is in {(0.1, 0.8), (0.2, 0.7), (0.3, 0.6), (0.4, 0.5), (0.5, 0.4), (0.6,0.3), (0.7, 0.2), (0.8, 0.1)}

large number of individuals (ps between 100 and 1000) discovered better solutions with $A.D.F.O$ close to the optimum. $ps > 1000$ gave a low value because a large population requires more time to reach high-quality solutions.

The performance of GGA was tested using eight couple values of crossover and mutation probabilities (p_c, p_m). The *OR5×100_0.25* instances were executed 30 times and the $A.D.F.O$ results are shown in Fig. 4d. The $A.D.F.O$ significantly improved with increase of the crossover rate and decrease of the mutation rate (Fig. 4d). The first couple gave the best $A.D.F.O$.

Different functions to measure the efficiency of items are presented in Sect. 2.2. They have been compared as methods for performing the pre-analysis in GGA. For that, the *OR5×100-0.25*; *OR5×250-0.25* and *OR5×500-0.25* have been used. The results are illustrated by Fig. 5. From the

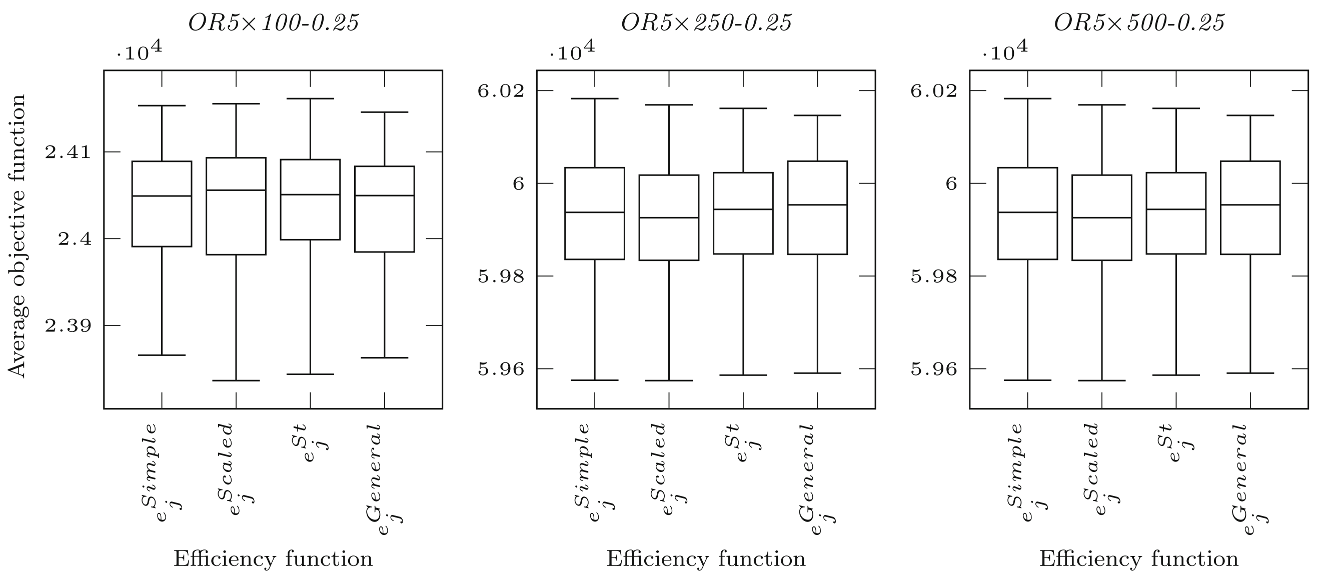

Fig. 5 The average objective function values range obtained by GGA with different values of efficiency measurement functions and using the dataset: *OR5×100-0.25*, *OR5×250-0.25* and *OR5×500-0.25*

Table 2 Comparison of the $D.F.O$ obtained by GGA with different expression of the $fitnessfunction$, using the *OR5×100-0.25* dataset

Instance	Eq. 9	Eq. 10	Eq. 11	Eq. 12
1	1,180	**0,682**	1,318	1,017
2	1,237	**0,757**	1,111	0,908
3	0,843	**0,382**	0,781	0,487
4	1,254	**0,663**	1,488	0,967
5	0,951	**0,583**	0,955	0,768
6	1,061	**0,523**	1,084	0,666
7	1,710	**0,856**	2,111	1,485
8	1,414	**0,570**	1,249	0,949
9	1,086	**0,620**	0,909	0,813
10	1,230	**0,563**	1,112	0,745

The Bold represents the best obtained D.F.O

charts, no significant difference between the functions was observed.

Four formulations of the fitness function are presented in Sect. 4.4. To decide which one gives better results, a comparison has been carried out by applying GGA on the *OR5×100-0.25* instances, Table 2 summaries the results. It is shown that GGA with Eq. 10 as fitness function gives the best $D.F.O$ for all the instances.

6 Experimental results

In order to provide a comprehensive analysis of the proposed GGA algorithm, this section provides two sets of experiments. The first set evaluates the gain in terms of performance from adding the guiding component to the standard GA.

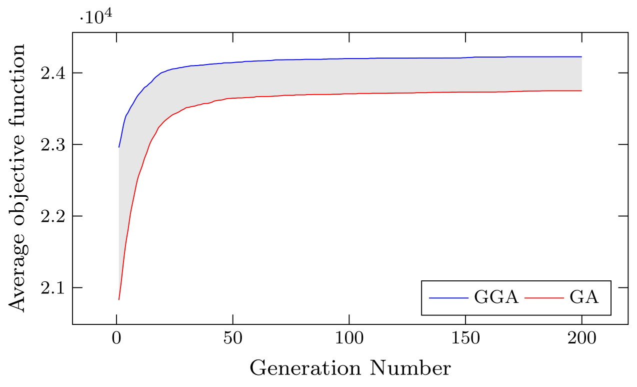

Fig. 6 Comparing the convergence of GGA with GA using the *OR5×100-0.25_1* instance

The results are supported with a statistical analysis. This is achieved by comparing the GGA with GA using the same evolutionary parameters (Sect. 6.1). The second set aims to compare the proposed GGA with the state-of-art results reported in the literature (Sect. 6.2).

6.1 Analysis of the GGA performance

A comparison between GA and GGA was conducted to measure the contribution of the data pre-analysis information on the convergence of GA. GGA and GA both were executed 30 times, each run given 200 generations on the *OR5×100-0.25_1*. The obtained objective function value of each generation was recorded. The average objective function of both approaches is compared in Fig. 6. The curves indicated two search steps: diversification (0-35) and intensification (36-200) in both experiments, GGA outperformed

Table 3 The GGA compared to GA in term of $A.D.F.O$, Best $D.F.O$, Worst $D.F.O$ and Average execution $Time$. The first and the last instance of each group is used

Instance	GGA				GA			
	Average D.F.O	Best D.F.O	Worst D.F.O	Time (s)	Average D.F.O	Best D.F.O	Worst D.F.O	Time (s)
5 × 100								
0.25_01	**0.56**	**0**	**1.13**	3	2.46	1.45	4.71	3
0.25_10	**0.58**	**0**	**1.18**	3	1.55	0.55	2.71	2
0.50_01	1.03	0.73	**1.6**	5	**0.74**	**0.12**	1.73	5
0.50_10	**0.44**	**0.22**	**0.77**	5	0.91	0.43	1.79	4
0.75_01	0.44	**0**	0.78	7	**0.35**	0.12	**0.58**	6
0.75_10	0.57	0.21	**0.74**	7	**0.34**	**0.09**	0.77	6
5 × 250								
0.25_01	**0.83**	**0.42**	**1.25**	7	3.45	2.68	5.32	5
0.25_10	**0.8**	**0.25**	**1.51**	7	4.33	2.74	5.63	4
0.50_01	**0.53**	**0.31**	**0.73**	13	1.21	0.71	1.86	11
0.50_10	**0.59**	**0.38**	**0.93**	12	0.97	0.48	1.47	10
0.75_01	**0.4**	**0.18**	**0.62**	82	0.53	0.29	0.78	17
0.75_10	**0.39**	**0.24**	**0.59**	19	0.44	0.24	0.85	17
5x × 500								
0.25_01	**0.87**	**0.52**	**1.32**	15	4.21	3.46	5.26	10
0.25_10	**0.91**	**0.58**	**1.57**	16	3.82	3.14	4.74	11
0.50_01	**0.42**	**0.23**	**0.81**	29	1.58	1.23	1.94	23
0.50_10	**0.5**	**0.29**	**0.73**	29	1.31	0.9	1.74	24
0.75_01	**0.25**	**0.14**	**0.41**	47	0.64	0.44	0.86	41
0.75_10	**0.3**	**0.14**	**0.55**	51	0.59	0.42	0.9	40
10 × 100								
0.25_01	**0.81**	**0.2**	**1.88**	3	1.77	1.31	2.63	3
0.25_10	**1.02**	**0**	**1.82**	3	2.82	1.29	4.63	3
0.50_01	**0.67**	**0.24**	**1.21**	5	1.2	0.51	2.09	5
0.50_10	**0.47**	**0.27**	**0.65**	6	1.87	0.7	2.88	5
0.75_01	**0.31**	**0.23**	**0.44**	8	0.51	0.25	1.4	7
0.75_10	**0.09**	**0**	**0.32**	8	0.61	0.3	0.94	7
10 × 250								
0.25_01	**0.92**	**0.39**	**1.56**	9	3.76	2.16	4.93	6
0.25_10	**0.88**	**0.61**	**1.59**	9	3.65	2.78	4.41	7
0.50_01	**0.56**	**0.26**	**0.84**	15	1.29	0.93	1.9	12
0.50_10	**0.49**	**0.23**	**0.95**	16	1.7	0.85	2.78	13
0.75_01	**0.28**	**0.14**	**0.52**	23	0.54	0.29	0.95	19
0.75_10	**0.29**	**0.08**	**0.6**	23	0.69	0.36	1.08	19
10 × 500								
0.25_01	**1.01**	**0.58**	**1.62**	19	2.78	2.17	4.05	14
0.25_10	**0.88**	**0.57**	**1.24**	19	3.46	2.38	4.62	13
0.50_01	**0.46**	**0.28**	**0.84**	35	1.45	1.04	2.15	28
0.50_10	**0.54**	**0.32**	**0.8**	35	1.49	1.07	2.14	27
0.75_01	**0.26**	**0.12**	**0.41**	53	0.8	0.63	1.11	44
0.75_10	**0.26**	**0.13**	**0.59**	53	0.68	0.52	0.82	45
30 × 100								
0.25_01	**1.58**	**0.79**	**2.73**	3	2.15	1.08	3.73	4
0.25_10	**1.28**	**0.18**	**2.63**	4	2.15	1.6	4.7	3
0.50_01	**1.18**	**0.54**	**1.79**	7	1.76	0.99	2.13	6

Table 3 continued

Instance	GGA				GA			
	Average D.F.O	Best D.F.O	Worst D.F.O	Time (s)	Average D.F.O	Best D.F.O	Worst D.F.O	Time (s)
0.50_10	**0.66**	**0.42**	**1.17**	7	2.5	1.61	3.19	5
0.75_01	**0.48**	**0.15**	1.23	10	0.9	0.46	**0.97**	8
0.75_10	**0.35**	**0.11**	**0.6**	10	0.54	0.34	0.93	9
30 × 250								
0.25_01	**2.09**	**1.83**	**3.13**	9	2.88	1.86	3.93	9
0.25_10	**2.68**	**1.51**	**3.29**	8	3.61	2.6	4.79	8
0.50_01	**1.17**	0.74	**1.58**	18	1.23	**0.61**	2.41	18
0.50_10	**1.2**	**1.04**	**1.41**	18	1.63	1.35	2.12	17
0.75_01	**0.38**	**0.23**	**0.64**	30	0.9	0.68	0.99	25
0.75_10	**0.58**	**0.41**	**0.77**	30	0.77	0.5	1.29	25
30 × 500								
0.25_01	**3.39**	**2.18**	**4.69**	100	4.83	3.48	5.49	164
0.25_10	3.9	**2.9**	6.29	99	**3.58**	3.37	**3.89**	16
0.50_01	2.2	2.05	2.42	35	**1.45**	**1.36**	**1.72**	33
0.50_10	**1.51**	**1.25**	**1.77**	41	1.63	1.48	2.39	35
0.75_01	**0.41**	**0.35**	**0.5**	67	0.68	0.41	1.01	61
0.75_10	**0.35**	**0.26**	**0.49**	77	0.6	0.47	1	59

The Bold represents the best obtained results (Average D.F.O, Best D.F.O, Worst D.F.O)

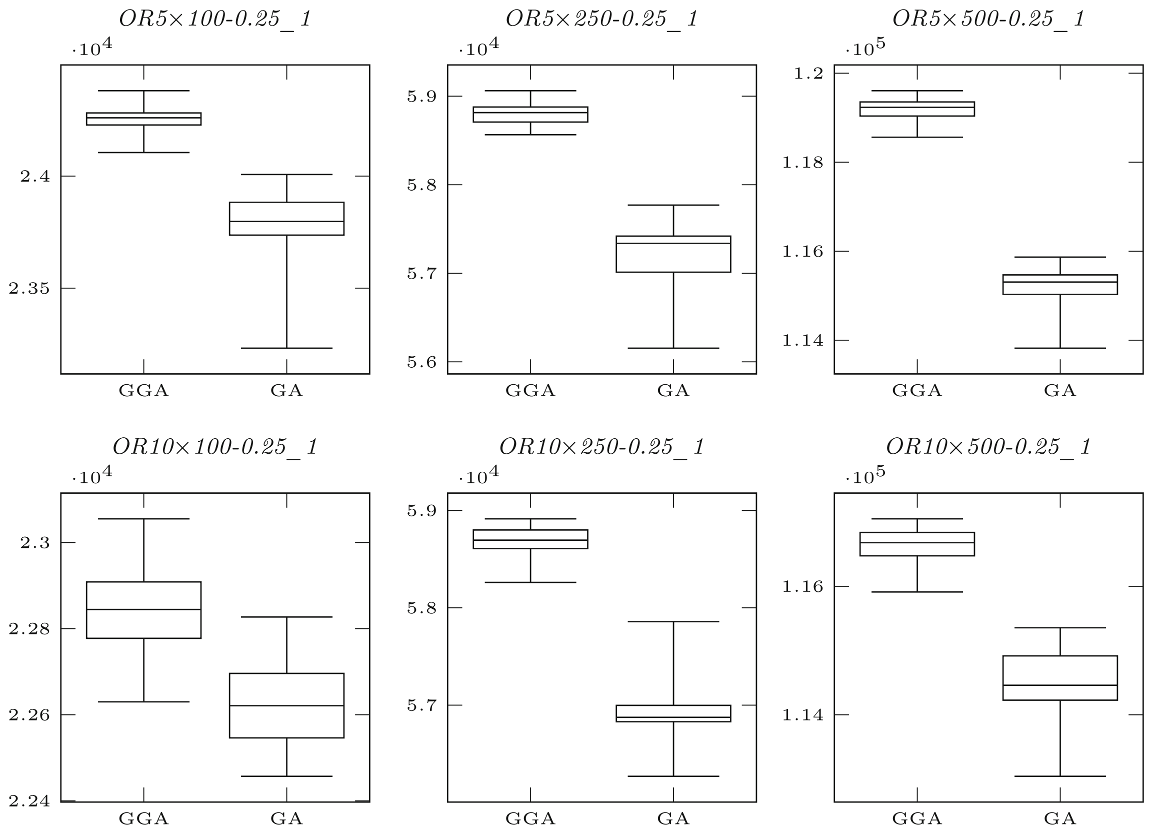

Fig. 7 The objective function rang obtained by GGA and GA within 30 times of run

Table 4 Comparison of GGA to GA, SAHS-SLS, SGHS, HS and greedy algorithms in terms of $A.D.F.O$ and average $time$

Dataset	GGA		GA		SAHS-SLS		SGHS		HS		$greedy$
	A.D.F.O	Time	A.D.F.O	Time	A.D.F.O	time	A.D.F.O	Time	A.D.F.O	Time	A.D.F.O
5×100											
0.25	0.62	3	2.17	2.2	**0.60**	18.9	0.78	16.3	5.36	9.5	29.25
0.50	0.67	5.1	0.86	4.0	**0.57**	24.8	0.78	22.3	5.15	12.4	13.01
0.75	0.34	6.9	0.42	6.0	**0.32**	29.1	0.47	27.6	3.09	16.6	6.40
Average	0.54	5	1.15	4.1	**0.49**	24.2	0.68	22.1	4.53	12.8	16.22
5×250											
0.25	**0.80**	7.6	4.03	4.8	2.14	31.5	2.62	26.6	8.41	22.4	23.81
0.50	**0.56**	12.6	1.15	10.4	1.48	68.1	2.27	57.2	6.21	36.3	9.10
0.75	**0.33**	18.9	0.58	16.8	1.22	99.3	1.77	80.5	3.78	62.3	4.07
Average	**0.56**	13.0	1.92	10.7	1.61	66.3	2.22	54.7	6.13	40.3	12.33
5×500											
0.25	**0.84**	15.6	4.27	10.5	1.85	146.3	3.75	109.1	9.57	66.7	20.70
0.50	**0.53**	30.8	1.45	24.0	1.31	248.4	3.60	177.3	7.25	116.4	9.62
0.75	**0.26**	47.7	0.65	42.3	1.33	340.0	2.69	269.6	4.67	187.6	3.39
Average	**0.54**	31.4	2.12	25.6	1.50	244.9	3.34	185.3	7.16	123.6	11.24
10×100											
0.25	1.23	3.0	2.40	3.0	**0.90**	29.0	1.26	27.8	5.31	16.5	21.69
0.50	**0.61**	5.2	1.53	5.0	0.64	46.6	0.97	33.6	4.90	20.6	10.20
0.75	**0.35**	7.9	0.53	6.8	0.36	48.8	0.64	43.5	3.22	27.3	4.91
Average	0.73	5.4	1.49	4.9	**0.63**	41.5	0.96	35.0	4.48	21.5	12.26
10×250											
0.25	**0.98**	8.9	3.56	6.2	1.49	93.3	2.33	74.1	7.17	44.7	15.94
0.50	**0.58**	15.0	1.35	12.6	0.91	160.2	2.06	116.3	6.01	76.6	7.91
0.75	**0.33**	23.0	0.66	19.2	0.67	261.5	1.59	172.8	3.85	116.6	3.57
Average	**0.63**	15.6	1.86	12.7	1.03	171.7	1.99	121.1	5.68	79.3	9.14
10×500											
0.25	**0.91**	18.5	3.61	13.2	2.11	157.1	4.16	118.5	9.95	78.0	15.39
0.50	**0.50**	32.6	1.44	27.4	1.61	267.0	4.23	207.4	7.95	143.5	6.10
0.75	**0.31**	50.3	0.71	45.4	1.45	384.4	4.11	297.3	5.04	229.3	2.61
Average	**0.57**	33.8	1.92	28.7	1.72	269.5	4.16	207.7	7.65	150.3	8.03
30×100											
0.25	1.66	3.6	2.27	3.0	**1.23**	77.0	0.93	68.5	2.79	49.7	18.04
0.50	1.10	6.0	1.72	5.8	**0.68**	64.9	0.96	52.5	3.90	36.5	8.83
0.75	0.49	9.6	0.78	8.9	**0.33**	87.1	0.52	67.9	3.22	46.8	5.97
Average	1.08	6.4	1.59	5.9	**0.75**	76.3	0.80	63.0	3.30	44.3	10.94
30×250											
0.25	2.03	9.2	3.20	8.6	**1.93**	155.9	2.47	91.9	7.59	61.8	11.53
0.50	1.17	17.1	1.46	17.3	**0.84**	155.8	2.49	103.0	7.00	71.6	5.55
0.75	**0.48**	27.7	0.73	28.3	0.66	231.7	1.77	150.7	4.28	108.3	3.33
Average	1.23	18.0	1.80	18.1	**1.14**	181.1	2.25	115.2	6.29	80.6	6.80
30×500											
0.25	4.07	99.1	3.50	18.8	**2.24**	173.3	5.63	127.1	9.71	89.0	44.18
0.50	1.90	40.4	1.45	36.1	**1.19**	264.2	4.82	206.3	8.39	137.1	6.21
0.75	**0.43**	63.4	0.69	58.5	0.87	344.0	6.31	285.1	6.44	243.3	2.30
Average	2.14	67.6	1.88	37.8	**1.43**	260.5	5.58	206.2	8.18	156.4	17.56

The Bold indicates the best A.D.F.O

Table 5 Statistics of *ANOVA* test used to check the GGA performance on GA using *OR5×100-0.25_1*

Algorithm	Count	Sum	Average	Variance	SD	Best	Worst
GGA	32	20.53	0.64	0.12	0.34	0.30	0.98
GA	32	82.88	2.59	0.68	0.82	1.77	3.41

Table 6 The *ANOVA* test results of the GGA performance on GA using *OR5×100-0.25_1*

Source of variation	SS	df	MS	F	P value	F crit
Between groups	60.75	1	60.75	152.75	2.23×10^{-18}	3.996
Within groups	24.66	62	0.40			
Total	85.40	63				

Table 7 Statistics of the *ANOVA* test used to check the GGA performance on GA using *Chu&Beasley* instances

Algorithm	Count	Sum	Average	Variance	SD	Best	Worst
GGA	27	24.08	0.89	0.63	0.80	1.69	0.10
GA	27	47.19	1.75	1.43	1.20	2.94	0.55

standard GA throughout the evolutionary process. GGA kept a large gap on GA and maintained it throughout the process.

An extended investigation is done using the first and the last instances of each class of the *Chu&Beasley* benchmarks (i.e. $m \times n - \alpha$_01, $m \times n - \alpha$_10, for example: *5×100-0.25_01, 5×100-0.25_10*). Table 3 shows a comparison between the performance of GGA against GA. The table shows the Average $D.F.O$, Best $D.F.O$, Worst $D.F.O$ and the average processing $Time$ for each algorithm. The results show that GGA outperformed GA on many instances. Also, GGA with high α value performed better than GGA with small α value, the lower and upper whiskers show the worst and best results achieved from 30 independent run times. Figure 7 compares GGA to GA in terms A.D.F.O range. Both methods were applied 30 times on the first instance of each class. Many values were recorded. Figure 7 shows lower, upper and median values obtained by GGA and GA.

6.1.1 Performance on large benchmarks

GGA is evaluated on the *Chu&Beasley* dataset and compared to: Harmony Search (HS), Self-adaptive Global best Harmony Search (SGHS), Self-Adaptive Hybrid Harmony Search-Stochastic Local Search (SAHS-SLS) [25], GA and greedy sorting (i.e. the feasible solution of adding the items in the sorting order as long as the constraints are verified). Table 4 shows the $A.D.F.O$ and the execution $time$ for all the above algorithms on 270 instances.

Closer inspection of the results shows that for 60% of the data, GGA was able to give better solutions than almost all the other approaches. In addition, the gap improvement from the greedy method after the application of GGA varied from 2% up to 40%. Furthermore, on many instances, GGA results were very close to the optimum. In terms of time, the approach was faster than the other approaches except the GA.

Table 8 The *ANOVA* test results of the GGA performance on GA using *Chu&Beasley* instances

Source of variation	SS	df	MS	F	P value	F crit
Between groups	9.89	1	9.89	9.58	0.003	4.03
Within groups	53.67	52	1.03			
Total	63.56	53				

6.1.2 Statistical analysis

Simple statistical analysis was used to investigate whether or not the guidance has a real effect on the GA, hence, ANalysis Of the Variance (*ANOVA*) pairwise comparison was conducted. It has been supposed that the null hypothesis H_0 is: “GGA has not a significant improvement on GA”. The first comparison includes only one instance (Tables 5 and 6) while the second includes all the instances (Tables 7 and 8). The first comparison indicated a $F = 152.72$ largely greater than $F\ crit = 3.99$, and a $P\text{-}value = 2.23 \times 10^{-18}$ largely lower than $\alpha = 0.05$. The second comparison including all the instances showed a $F = 9.58$ greater than $F\ crit = 4.03$, and a $P\text{-}value = 0.003$ lower than $\alpha = 0.05$. Both results confirm that GA is significantly improved by adding the guidance to its search process. Consequently, the null hypothesis can be rejected.

The statistical pairwise comparison of GGA with greedy, GA, SAHS-SLS, SGHS and HS approaches is reported in Table 9. The comparison is performed with t-test, *ANOVA* and $Welch's$ t-test (also known as t-test with two-sample assuming unequal variances) and using the same data. The obtained t-test values were less than $P\text{-}value = 0.05$ except with SAHS-SLS. Also, *ANOVA* comparison results indicated a $P\text{-}value$ less than $\alpha = 0.05$ and F largely greater than F $crit$ except when compared with SAHS-SLS. $Welch's$ t-test obtained a negative $t\text{-}Stat$ and a $P\text{-}value$ less than $\alpha = 0.05$

Table 9 Summary of t-test, $ANOVA$ and Welch's t-test of GGA performance compared to greedy, GA, SAHS-SLS, SGHS and HS on the *Chu&Beasley* instances

		GGA vs. greedy	GGA vs. GA	GGA vs. SAHS-SLS	GGA vs. SGHS	GGA vs. HS
t-test	*P* value	2.3×10^{-6}	0.002	0.093	3.7×10^{-5}	2.7×10^{-13}
ANOVA	*P* value	5.1×10^{-7}	0.003	0.19	4.6×10^{-5}	9.1×10^{-16}
	F	32.85	9.58	1.80	19.75	130.08
	F crit	4.03	4.03	4.03	4.03	4.03
Welch's	t Stat	−5.54	−2.89	−1.36	−4.50	−11.02
t-test	P(T<=t) one-tail	4.6×10^{-6}	0.003	0.091	3.2×10^{-5}	1×10^{-12}
	t Critical one-tail	1.71	1.68	1.68	1.69	1.69
	P(T<=t) two-tail	9.3×10^{-6}	0.006	0.182	6.4×10^{-5}	2×10^{-12}
	t Critical two-tail	2.06	2.02	2.01	2.03	2.04

Table 10 Comparison of the results obtained by GGA to some constructive and improvement heuristics

			Constructive							Improvement		
n	m	α	GGA	GA	PECH	MAG	VZ	PIR	SCE	CB	NR(P)	MCF
5	100	0.25	**0.62***	2.17	7.3	13.6	10.3	1.6	3.5	0.99	0.94	1.09
		0.50	**0.66**	0.86	3.4	6.7	6.9	0.77	2.6	0.45	**0.44***	0.57
		0.75	**0.34**	0.42	2.02	5.1	5.6	0.48	1.1	0.32	**0.22***	0.38
	250	0.25	0.79	4.03	7.1	6.6	5.8	**0.53**	4.3	**0.23***	0.46	0.41
		0.50	0.56	1.15	3.2	5.2	4.4	**0.24**	3.3	**0.12***	0.17	0.22
		0.75	0.33	0.58	1.8	3.5	3.5	**0.16**	1.5	**0.08***	0.1	0.14
	500	0.25	0.83	4.27	6.4	4.9	4.1	**0.22**	4.6	1.56	**0.15***	0.21
		0.50	0.53	1.45	3.4	2.9	2.5	**0.08**	3.6	0.79	**0.06***	0.1
		0.75	0.26	0.65	1.7	2.3	2.41	**0.06**	1.8	0.48	**0.03***	0.06
10	100	0.25	**1.23**	2.40	8.2	15.8	15.5	3.4	6.8	**0.09***	2.05	1.87
		0.50	**0.61**	1.53	3.7	10.4	10.7	1.8	5.1	**0.04***	0.81	0.95
		0.75	**0.35**	0.53	1.8	6.1	5.67	1.1	2.4	**0.03***	0.44	0.53
	250	0.25	**0.98**	3.56	5.8	11.7	10.5	1.1	6.9	**0.51***	0.88	0.79
		0.50	0.58	1.35	2.5	6.8	5.9	**0.57**	5.4	**0.25***	0.39	0.41
		0.75	**0.32**	0.66	1.5	4.4	3.7	0.33	2.8	**0.15***	0.19	0.24
	500	0.25	0.9	3.61	5.1	8.8	7.9	**0.52**	6.8	**0.24***	0.34	0.44
		0.50	0.5	1.44	2.4	5.7	4.1	**0.22**	5.8	**0.11***	0.14	0.2
		0.75	0.31	0.71	1.2	3.6	2.9	**0.14**	3.4	**0.07***	0.1	0.13
30	100	0.25	**1.65***	2.27	6.8	17.3	17.2	9.1	8.6	2.91	2.24	3.61
		0.50	**1.09***	1.72	3.2	11.8	10.1	3.51	6.6	1.34	1.32	1.6
		0.75	**0.49***	0.78	1.9	6.58	5.9	2.03	3.6	0.83	0.8	0.97
	250	0.25	**2.03**	3.20	4.8	13.5	12.4	3.7	8.3	**1.19***	1.27	1.75
		0.50	**1.16**	1.46	2.1	8.6	7.1	1.5	6.9	**0.53***	0.75	0.79
		0.75	**0.48**	0.73	1.2	4.4	3.9	0.84	3.8	**0.31***	0.38	0.43
	500	0.25	4.07	3.50	3.7	9.8	9.6	**1.89**	8.6	**0.61***	0.89	1.05
		0.50	1.9	1.45	1.7	7.1	5.7	**0.73**	7.4	**0.26***	0.36	0.44
		0.75	0.43	0.69	0.9	3.7	3.5	**0.48**	4	**0.17***	0.23	0.27

The Bold represents the best A.D.F.O per category and the star indicates the overall best A.D.F.O

except for SAHS-SLS. From all this statistical analysis it may be concluded that the null hypothesis H_0 is rejected. Therefore, GGA performs significantly better than the other approaches and is comparative to the SAHS-SLS.

6.2 Comparison with the literature

Heuristics could be classified into two groups: *constructive* heuristics that aim to construct a solution for the treated prob-

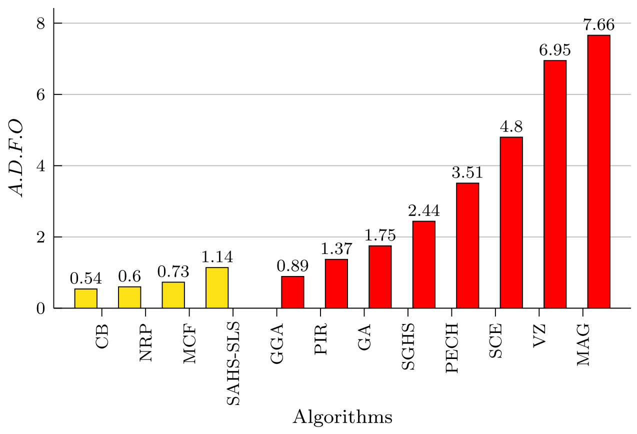

Fig. 8 Comparing the average distance from the optimum $A.D.F.O$ obtained by GGA using the *Chu&Beasley* instances with nine other algorithms from the literature

lem and *improvement* heuristics which improve a given initial solution. GGA is compared with a set of *constructive* and *improvement* heuristics. A brief description of each heuristic is given bellow.

- Constructive heuristics:

- PECH (Primal Effective Capacity Heuristic) [2] is a simple greedy heuristic which incorporates a strategy based on the *effective capacity* for selecting and adding the items to the knapsack.
- MAG [19] and VZ [32] are algorithms based on the Lagrange multipliers approach.
- PIR [21] is a heuristic based on a dual surrogate relaxation of the MKP supported with a branch and bound method.
- SCE (Shuffled Complex Evolution) [7] is an evolutionary heuristic that iteratively redistributes the population individuals into smaller structures (or complexes) according to their fitness.

- Improvement heuristics:

- CB [10] is a GA augmented with a feasibility and constraint operator which utilises problem-specific knowledge and repair operators which locally improves the offspring.
- NR (P) (New Reduction (Pirkul)) [16], operates a lagrangian dual relaxation on MKP, and proposes a dynamic estimation of the *Core* size relative to the problem difficulty. The *Core* is then solved with a greedy heuristic combined with a local improvement phase.
- MCF (Modified Choice Function - Late Acceptance Strategy) [13] is a hyper-heuristic based on heuristics selection function named *Modified Choice Function*.

The comparison illustrated in Table 10 and Fig. 8 revealed that GGA was competitive with both *constructive* and *improvement* methods and has managed to outperform both group of methods on a few instances.

7 Conclusion

The purpose of the current study was to introduce a memetic algorithm named Guided Genetic Algorithm (GGA) for solving the Multidimensional Knapsack Problem (MKP). GGA analyses the problem data based on a greedy algorithm. Useful information about the items of the MKP are extracted and integrated in the initialisation and evaluation operators of a GA. To validate the approach, several experiments were conducted on well-known MKP test data. The research has shown that adding the guidance has significantly improved the performance of the GA and accelerated its speed of convergence. These experiments confirmed that GGA has a real impact on GA performance. One of the more significant findings of this study is that prior-knowledge about the data of a combinatorial optimisation problem could be significantly helpful to accelerate its solving. The future work intends to extend and evaluate GGA on other combinatorial optimisation problems.

References

1. Acan A, Tekol Y (2003) Chromosome reuse in genetic algorithms. In: Genetic and evolutionary computation conference, Springer, pp 695–705
2. Akçay Y, Li H, Xu SH (2007) Greedy algorithm for the general multidimensional knapsack problem. Ann Oper Res 150(1):17
3. Bader-El-Den M, Gaber M (2012) Garf: towards self-optimised random forests. In: International conference on neural information processing, Springer, pp 506–515
4. Bader-El-Den M, Poli R (2007) Generating sat local-search heuristics using a gp hyper-heuristic framework. In: International conference on artificial evolution (Evolution Artificielle), Springer, pp 37–49
5. Bader-El-Den M, Poli R, Fatima S (2009) Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. Memet Comput 1(3):205–219
6. Balas E, Zemel E (1980) An algorithm for large zero-one knapsack problems. Oper Res 28(5):1130–1154
7. Baroni MDV, Varejão FM (2015) A shuffled complex evolution algorithm for the multidimensional knapsack problem. In: Iberoamerican congress on pattern recognition, Springer, pp 768–775
8. Castro F, Gelbukh A, González M (2013) Advances in Soft Computing and Its Applications. In: 12th Mexican international conference, MICAI 2013, Mexico City, Mexico, November 24-30, 2013, proceedings. No. pt. 2 in lecture notes in computer science, Springer Berlin Heidelberg, https://books.google.com.om/books?id=WgC6BQAAQBAJ
9. Chen SH, Chang PC, Cheng T, Zhang Q (2012) A self-guided genetic algorithm for permutation flowshop scheduling problems. Comput Oper Res 39(7):1450–1457

10. Chu PC, Beasley JE (1998) A genetic algorithm for the multidimensional knapsack problem. J Heuristics 4(1):63–86
11. Dantzig GB (1957) Discrete-variable extremum problems. Oper Res 5(2):266–288
12. Dobson G (1982) Worst-case analysis of greedy heuristics for integer programming with nonnegative data. Math Oper Res 7(4):515–531
13. Drake JH, Özcan E, Burke EK (2015) Modified choice function heuristic selection for the multidimensional knapsack problem. In: Genetic and evolutionary computing, Springer, pp 225–234
14. Fatima S, Bader-El-Den M (2010) Co-evolutionary hyper-heuristic method for auction based scheduling. In: Evolutionary computation (CEC), 2010 IEEE congress on, IEEE, pp 1–8
15. Gabaldon E, Lerida JL, Guirado F, Planes J (2014) Slowdown-guided genetic algorithm for job scheduling in federated environments. In: International conference on nature of computation and communication, Springer, pp 181–190
16. Hill RR, Cho YK, Moore JT (2012) Problem reduction heuristic for the 0–1 multidimensional knapsack problem. Comput Oper Res 39(1):19–26
17. Huston S, Puchinger J, Stuckey P (2008) The core concept for 0/1 integer programming. In: Proceedings of the fourteenth symposium on Computing: the Australasian theory-Volume 77, Australian Computer Society, Inc., pp 39–47
18. Kellerer H, Pferschy U, Pisinger D (2004) Introduction to NP-completeness of knapsack problems. Springer, Berlin, Heidelberg, pp 483–493
19. Magazine M, Oguz O (1984) A heuristic algorithm for the multidimensional zero-one knapsack problem. Eur J Oper Res 16(3):319–326
20. Perry T, Bader-El-Den M, Cooper S (2015) Imbalanced classification using genetically optimized cost sensitive classifiers. In: Evolutionary computation (CEC), 2015 IEEE congress on, IEEE, pp 680–687
21. Pirkul H (1987) A heuristic solution procedure for the multiconstraint zero? one knapsack problem. Naval Res Logist 34(2):161–172
22. Poli R, Koza J (2014) Genetic programming. In: Search methodologies, Springer, pp 143–185
23. Puchinger J, Raidl GR, Pferschy U (2006) The core concept for the multidimensional knapsack problem. Springer, Berlin
24. Rasheed K (1999) Guided crossover: a new operator for genetic algorithm based optimization. In: Evolutionary computation, 1999. CEC 99. Proceedings of the 1999 congress on, IEEE, vol 2, pp 1535–1541
25. Rezoug A, Boughaci D (2016) A self-adaptive harmony search combined with a stochastic local search for the 0–1 multidimensional knapsack problem. Int J Bio Inspired Comput 8(4):234–239
26. Rezoug A, Boughaci D, Bader-El-Den M (2015) Memetic algorithm for solving the 0-1 multidimensional knapsack problem. In: Portuguese conference on artificial intelligence, Springer, pp 298–304
27. Senju S, Toyoda Y (1968) An approach to linear programming with 0-1 variables. Management Science pp B196–B207
28. Snášel V, Dvorskỳ J, Ochodková E, Krömer P, Platoš J, Abraham A (2010) Evolving quasigroups by genetic algorithms. In: Proceedings of DATESO, Citeseer, pp 108–117
29. Sudholt D (2015) Parallel evolutionary algorithms. In: Springer handbook of computational intelligence, Springer, pp 929–959
30. Tang KS, Man KF, Kwong S, He Q (1996) Genetic algorithms and their applications. IEEE Signal Process Mag 13(6):22–37
31. Vázquez-Barreiros B, Mucientes M, Lama M (2014) A genetic algorithm for process discovery guided by completeness, precision and simplicity. In: International conference on business process management, Springer, pp 118–133
32. Volgenant A, Zoon J (1990) An improved heuristic for multidimensional 0–1 knapsack problems. J Oper Res Soc 41(10):963–970
33. Yang S, Jat SN (2011) Genetic algorithms with guided and local search strategies for university course timetabling. IEEE Trans Syst Man Cybern Part C (Appl Rev) 41(1):93–106
34. Zhang Q, Sun J, Tsang E (2005) An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Trans Evolut Comput 9(2):192–200