International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 10 (2018) pp. 143-153

Received: 7 Dec. 2017; Acceptance: 29 April, 2018; Published: 28 May 2018

Distribution of fiscal coupons via Genetic Algorithms and Greedy Randomized Adaptive Search Procedure

Kadri Sylejmani[1], Qëndresë Hyseni[1], Sule Yildirim[2], Agon Qurdina[1], Lekë Mula[1] and Bujar Krasniqi[1*]

[1] Faculty of Electrical and Computer Engineering, University of Prishtina,
Bregu i Diellit p.n., Prishtinë 10000, Kosovo
{kadri.sylejmani, bujar.krasniqi}@uni-pr.edu, {qendrese.hyseni, agon.qurdina, leke.mula}@studentet.uni-pr.edu

[2] Department of Information Security and Communication Technology, Norwegian University of Science and Technology,
Teknologivegen 22, 2802 Gjøvik, Norway,
sule.yildirim@ntnu.no

***Abstract*: When customers buy goods or services from business entities they are usually given a receipt that is known with the name fiscal or tax coupon, which, among the others, contains details about the value of the transaction. In some countries, the fiscal coupons can be collected during a certain period of time and, at the end of the collection period, they can be handed over to the tax authorities in exchange for a reward, whose price depends on the number of collected coupons and the sum of their values. From the optimization perspective, this incentive becomes interesting when, both the number of coupons and the sum of their value is large. Hence, in this paper, we model this problem in mathematical terms and devise a test set that can be used for benchmarking purposes. Furthermore, we solve this problem by means of two metaheuristics, namely Genetic Algorithms and Greedy Randomized Adaptive Search Procedure. Finally, we evaluate the proposed algorithms by comparing their results against the relaxed versions of the proposed problem. The computational experiments indicate that both approaches are competitive, as they can be used to solve realistic problems in a matter of few seconds by utilizing standard personal computers.**

***Keywords*:** Distribution of fiscal coupons, Mathematical Modelling, Genetic Algorithms, Greedy Randomized Adaptive Search Procedure**.**

I. Introduction

The tax authorities of many countries try to find alternative ways to enforce business entities (e.g. shops, restaurants, travel agencies, etc.) to fully declare the profit they gain from their business activities, so that they have to pay taxes accordingly. The tax authorities from several countries, like for example Republic of Kosovo [1] or State of Minnesota in USA [2], utilize the strategy of encouraging the customers to collect the fiscal coupons when they do any kind of transaction with business entities. The collected coupons [1] can be enveloped and submitted to the tax authorities in exchange of a reward that depends on the number and the total value of the fiscal coupons enclosed. In general, depending on the actual rules put in place by specific tax authorities, there can be different types of envelopes that can be submitted. Obviously, envelopes with more coupons and with higher total values, have higher rewards.

In more formal terms, in the case of the Distribution of Fiscal Coupons Problem (DFCP), each person has N number of coupons (see Figure 1) collected for a period of time (e.g. a three-month period). At the end of the collection period, the coupons will be distributed into a T number of envelopes by the person who possesses the coupons. The person has to make a decision related to which coupon is placed in which envelope. Each coupon has a value and consequently it should be placed in the envelope ultimately where the sum of values of the coupons in the envelope will lead to an overall higher reward. Each coupon can only be placed into a single envelope. The number of coupons in each envelope cannot be less than a minimum, whereas also the sum of all coupons cannot fall under minimum value. The achievable reward from each envelope type is predefined based on the number and values of the coupons placed inside it.

For illustrative purposes, in Tables 1, 2 and 3, we present a

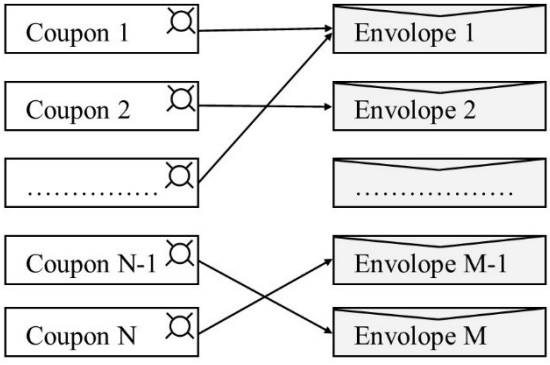

Figure 1. The schematic view of the fiscal coupons problem

* Corresponding author

sample scenario of a fiscal coupons distribution problem. Table 1 depicts the basic problem details, such as number of coupons, number of types of envelopes, the values of individual coupons and the total values of coupons. In Table 2, we present the details about types of envelopes, in terms of applicable constraints (i.e. minimum number of coupons and value) and achievable reward. Whilst, in Table 3, we present a possible solution (i.e. distribution of fiscal coupons into envelopes) to this actual problem, where it can be seen that the total reward is 5.5 €.

No. of coupons	No. of types of envelopes	Coupon values (€)	Sum of all coupons (€)
15	3	7, 4, 9, 15, 2, 4, 6, 3, 10, 5, 7, 8, 1, 12, 7	100

Table 1. Basic details of fiscal coupons problem

Envelope type	Minimum coupons	Minimum value (€)	Reward (€)
1	2	20	1
2	4	25	1.5
3	6	30	2

Table 2. Types of envelopes

No. of envelope	Type of envelope	Reward (€)	List of coupons
1	1	1	5, 15
2	1	1	8, 12
3	2	1.5	7, 7, 7, 9
4	3	2	1, 2, 3, 4, 4, 6, 10
	Total reward:	**5.5**	

Table 3. Sample solution

The significance of the work in this paper can be underlined by outlining its main contributions, which are: (i) introduction of a new optimization problem for the scientific community by presenting a mathematical formulation, as well as a test set that can be used for bench-marking purposes, (ii) development of two metaheuristic based solutions to the newly introduced problem, where one of them is based on Genetic Algorithms, whereas the other one is based on Greedy Randomized Adaptive Search Procedure, and (iii) presentation of the systematic computation results that compare the performance of the proposed algorithms against solutions of the relaxed version of the envisioned problem, which can be used as benchmark results for future solutions.

The remainder of this paper is structured as is in the following. Section 2 presents a literature review of the related problems and their respective solution approaches. Next, in Section 3, we present the mathematical modelling of the DFCP problem as an Integer Linear Programming Problem. In Section 4, we present the proposed approaches for solving the DFCP problem, while in Section 5, we show computation results of the proposed approaches against a data set of 10 instances. Finally, in Section 6, we conclude the paper and present our view for the subsequent future work.

II. State of the art

In the classical Knapsack Problem (KP) there is a set of items and a container (knap-sack) that has to be used for carrying a subset of items. Each item is characterized with two properties, namely value and weight, whereas the container has a single property, which is the maximum weight it can carry. The goal is to place a subset of items into the container such that the total value of the placed items is maximized subject to the capacity of the container. The Multiple Knapsack Problem (MKP) extends the KP problem by allowing multiple containers of the same capacity [3], whereas the Distributed Multiple Knapsack Problem (DMKP) supports containers of varying capacities, which can be modelled as a general Distributed Constraint Optimization Problem (DCOP) [4] [5] [6] [7]. The DCOP problem was solved by using memory-bounded and asynchronous algorithms that are based on the search strategy of ADOPT, which is a technique that alternates between best-first search and depth-first branch-and-bound search [3]. Another approach for DCOP problem is a distributed algorithm called optimal asynchronous partial overlay, which is a partial centralization technique known as cooperative mediation. The basic idea of this search technique is to focus the search process within areas of a sub problems that overlap, and then increase the size of the search horizon as the smaller sub problems gets solved [5]. Another extension of MKP is the generalized quadratic multiple knapsack problem (GQMKP) [8] [9], which introduces three new features: (i) *quadratic component* that allows extra profit in case two items are selected jointly, (ii) *setups* that enable grouping items into classes, where each class is characterized with a fixed cost and capacity, and (iii) consideration of *knapsack preferences* for the items. The GQMKP was recently solved by using a memetic approach [8] that combines a back-bone crossover operator with a simulated annealing algorithm that utilizes a neighborhood exploration mechanism consisting of multiple operators.

Problem	Objective	Capacity constraints		No. of. Containers	No. of features per item	No. of features per container	Preference of items for containers	Relation between items	All items to be selected
		Upper limit	Lower limit						
KP	Max	Yes	No	1	2	1	No	No	No
MKP	Max	Yes	Yes	>1	2	1	No	No	No
DMKP	Max	Yes	Yes	>1	2	1	No	No	No
GQMKP	Max	Yes	Yes	>1	3	1	Yes	Yes	No
BPP	Min	Yes	No	>1	1	1	No	No	Yes
RCP	Min	Yes	No	>1	1	3	No	No	Yes
DFCP	Max	No	Yes	>1	1	3	No	No	No

Table 4. Comparison of features of various related problems

Another related problem is the Bin Packing Problem (BPP), where a set of items need to be placed into a set of containers (bins). Each item has a weight property, whereas each container has a maximum capacity property. The goal is to place each item into a container such that the number of containers used is minimized. In comparison to KP problem, where only a subset of items can be picked, in the BPP problem all items have to be picked up [10]. Further, the two-dimensional bin packing problem is used in situations where a set of rectangle objects have to be accommodated into larger standardized rectangles with the aim of minimizing the waste (i.e. leftover from the standardized rectangles when all rectangles are placed). Lodi et al. [11] did a survey of different modellings for the two-dimensional BPP and described a plethora of approaches utilized in the literature.

An additional related problem is the Rack Configuration Problem (RCP), where there is a set of items (electronic cards) that need to be placed (connected) into a set of containers (racks). Each item has a single property (i.e. power it requires), whereas each container has three properties, namely maximal power it can supply, number of connectors and the price. The goal is to plug in all the electronic cards into a set of racks with the smallest cost possible [12]. This problem has been tackled by using constraint satisfaction problem techniques, where the issue of reducing the symmetry in RCP is considered. Further, the primal and dual model of RCP are integrated in order to produce better results [12].

In Table 4, we compare the newly proposed DFCP problem against the above presented related problems, by outlining different characteristics of the individual problems, such as: type of objective function, capacity constraints (i.e. upper/lower limit), the number of containers, the number of features per item/container, and whether all items need to be selected. By analyzing the details given in the table, one can conclude that DFCP problem is closely related to RCP problem, in terms of number of containers and features per item/container, but differs in terms of the objective function, capacity constraints and in the aspect whether all items need to be selected.

In [13], we presented a steady-state genetic algorithm for solving the DFCP problem, where the search space is explored by using a combination of two mutation operators, namely swap (that interchanges two coupons belonging to distinct envelopes) and shift (that shits a given coupon from one envelope to another one). More details of this approach are given in the section of Solution Approaches. To the best of authors' knowledge, except our work presented in [9], there is no any problem in the literature that models or solves the problem of the distribution of fiscal coupons, hence in the following section, we present a mathematical modelling of this problem, along with the proposed solutions.

III. Mathematical modeling

The mathematical modelling for the problem of the optimal distribution of the fiscal coupons is formulated as an Integer Linear Programming (ILP) model that has a range of parameters and a couple of decision variables, as specified below:

Parameters:

N – Number of coupons
v_i – Value of coupon i, $\forall i=1, \ldots, N$
T – Number of types of envelopes
C_k – Minimum number of coupons in envelope of type k, $\forall k=1, \ldots, T$
S_k – Minimum sum of all coupons in envelope of type k, $\forall k=1, \ldots, T$
R_k – Achievable reward from envelope of type k, $\forall k=1, \ldots, T$

Decision variables:

M – Number of envelopes
x_{jk} -equals 1, if envelope j is of type k, otherwise it is 0, $\forall j=1, \ldots, M$, $\forall k=1, \ldots, T$
y_{ij} -equals 1, if coupon i is placed in envelope j, otherwise it is 0, $\forall i=1, \ldots, N$, $\forall j=1, \ldots, M$,

Objective function:

$$Maximize \sum_{j=1}^{M} \sum_{k=1}^{T} R_k x_{jk} \quad (1)$$

Constraints:

$$\sum_{i=1}^{N} y_{ij} x_{jk} \geq C_k, \forall j = 1, \ldots, M, \forall k = 1, \ldots, T \quad (2)$$

$$\sum_{i=1}^{N} v_i y_{ij} x_{jk} \geq S_k, \forall j = 1, \ldots, M, \forall k = 1, \ldots, T \quad (3)$$

$$\sum_{i=1}^{N} y_{ij} \leq 1, \forall j = 1, \ldots, M \quad (4)$$

$$\sum_{j=1}^{M} x_{jk} = 1, \forall k = 1, \ldots, T \quad (5)$$

In the mathematical formulation presented above, Equation (1) denotes the objective function of the problem at hand, which is maximizing the total reward, by determining which combination of envelopes yields to the highest possible profit. Constraints (2) and (3) ensure the validity of envelopes in terms of the requirement for the minimum number of coupons and the minimum sum of their values, respectively. Constraint (4) guarantees that each single coupon is inserted into at most one envelope, whilst Constraint (5) makes sure that each single envelope can belong to only one particular type of envelope.

IV. Solution approaches

In this section, we present the two approaches designed for solving the DFCP problem, where the first one is based on Genetic Algorithms (GA), whereas the second one is based on Grady Randomized Adaptive Search Procedure (GRASP).

A. Genetic algorithms

In this implementation, we use the Steady State approach of Genetic Algorithms, which was popularized by Whitley & Kauth [14]. Its main idea, compared to the traditional generational approach, is to update the population in a slight manner rather than all at one time. The algorithm iteratively breeds a new child or two, assesses their fitness, and then restores them directly into the population itself, slaying off some preexisting individuals to make room for them. The Steady-State Genetic Algorithm has two essential features. First, it uses half the memory of a standard genetic algorithm,

because there is only one population at a time. Second, it is more exploitative compared to a generational approach [15].

1) Algorithm particularities

The particular details of the Steady-State Genetic Algorithm implemented here, can be summarized as in the following:

Representation of a given candidate solution is made as a list of lists, where the size of the main list corresponds to the number of assigned envelopes *M*, whereas each single member of the main list is also a list that corresponds to the number of coupons n_i placed inside a given envelope *i*. A sample representation of a given solution is: $S = \{E_1, E_2, \dots, E_i, \dots, E_M\}$, where $E_i = \{C_1, C_2, \dots, C_{n_i}\}$.

Initialization of a given candidate solution begins by reading all the coupons values from the given problem instance, and then, randomly distributing them into a random number of envelopes, by considering the hard constraints about the minimum number / sum of coupons. The number of generated initial solutions is equal to the *population size* (*ps*) parameter.

Mutation mechanism of the algorithm consists of two operators, namely *swap* and *shift*, where the earlier swaps two coupons belonging to distinctive envelopes, while the later shifts a coupon from a given envelope to some other envelope, of the same candidate solution. In order to apply a number of swaps between different coupons of a given individual, the *swap* operator iterates through a loop for a number of iterations (as specified by *sw* parameter). During the course of a single iteration, initially, two distinct envelopes are selected randomly, and then, for swapping purposes, one random coupon is selected from each of these envelopes. The shift operator is also executed for several iterations, as specified by *sh* parameter. During the evolution of a given iteration, initially, two distinct envelopes are selected randomly, and then one random coupon is selected from the first envelope, and gets shifted to the second one.

Algorithm 1 Steady State Genetic Algorithm

Require: coupons *C* (*N*, v_i, *T*, C_j, S_j, R_j), where *i*=1…*N* and *j*=1…*T*; population size *ps*; maximum generations *mg*; tournament size *ts*; swap mutate intensity *sw*; shift mutate intensity *sh*; operator alternation frequency *af*.

```
P = {}; Best = ∅; Worst = ∅;
for ps times do
  C_r = Random Individual (C)
  AssessFitness(C_r)
  if Worst == ∅ or Fitness(C_r) < Fitness(Worst) then
    Worst = C_r
  P = P ∪ C_r
for each generation until mg do
  P_w = SelectWithReplacement(P),
  P_b = TournamentSelection(P, ts),
  C_b = Select best from P_b
  C_b = Mutate(C_b, sw, sh, af)
  if Fitness(C_b) < Fitness(Worst) then
    Worst = C_b
  SelectForDeath(P_w)
  P = P - P_w
  P = P ∪ C_b
  for each individual P_i ∈ P do
    AssessFitness(P_i)
    if Best == ∅ or Fitness(P_i) > Fitness(Best) then
      Best = P_i
return Best
```

Evaluation of a given candidate solution is done using Equation (1), which, as described before, maximizes the total reward, by determining which combination of envelopes yields to the highest possible profit. Each member of the population has a certain number of envelopes of different types, and the sum of these envelope values denotes the fitness of the member.

Selection of the parent that will take part in breading the next population is completed by using the Tournament Selection algorithm. This algorithm is a simple and an effective one, as it returns the fittest individual of some ts individuals picked at random from the population [9].

Population update strategy mechanism replaces the worst fit member of the current population (i.e. it replaces the current worst solution from the population with the best picked from tournament size individuals).

2) Pseudocode of the algorithm

In abstract terms, as shown in Algorithm 1, the envisioned GA approach has 6 parameters, which can be used for fine tuning its performance for different problem complexities and sizes. Besides the default genetic algorithm parameters, such as population size (*ps*), maximum generations (*mg*) and tournament size (*ts*), the particular implementation at hand, uses three so called "intensity" parameters, namely swap mutate (*sw*) and shift mutate intensity (*sh*), for specifying the number of times a certain operator (i.e. swap or shift) will be applied when called upon. In addition, the algorithm uses a special parameter called the alternation frequency (*af*) to change the mutation operator from swapping to shifting and vice versa every *af* number of generations.

At the very start of the algorithm, a population *P* of *n* individuals is created by using the procedure for creating the initial solution explained above. Next, in the repetitive phase of the algorithm, at each iteration, the following steps are undertaken: (1) evaluation of all individuals, (2) selection of the parents based on tournament selection and mutation over the operators (i.e. swap and shift) used in the running iteration, and (3) formation of the new population by replacing the individual with the worst fitness, with the mutated new individual if the fitness of the second is better. The algorithm terminates when the maximum number of foreseen generations is achieved.

The GA algorithm is developed by using the C# programming language through the developing environment of MS Visual Studio 2015.

B. Greedy Randomized Adaptive Search Procedure

Another approach used for solving the DFCP problem is Greedy Randomized Adaptive Search Procedure (GRASP), which is a single-state metaheuristics algorithm built on concepts of constructing a feasible solution and then applying a local search heuristic. This algorithm was introduced by Feo & Resende [16] and, in overall, it is quite simple [15], and it can be characterized with two main steps, explicitly: (i) create a feasible solution by constructing from among the highest

value (fitness) components, and (ii) do some hill climbing on feasible solution and choose the best one.

1) Algorithm particularities

The particular GRASP approach (Algorithm 2) that we implemented in this case can be configured by using four distinct parameters, such as: max iterations (*mi*), max local search iterations (*mls*) with no improvement, percentage of components (*pc*) chosen randomly, and envelope filling *mode*. While the first two parameters are common GRASP parameters, the *pc* parameter specifies the percentage of elements (components) to be considered when choosing them randomly from the whole group of the coupons. Further, the envelope filling mode parameter determines one of the three possible orders of using envelope types (as described below).

Besides the representation, which is the same as in the case of GA approach, the other two main characteristics of the GRASP approach (see Algorithm 2), proposed in this paper are the solution construction procedure and the local search mechanism. The **solution construction procedure** (lines 3 to 13 in Algorithm 2) initializes a solution by filling up the envelopes in a sequential order until no more coupons are left. Based on the order of consideration of the envelope types, with respect to the reward value, three different modes of solution construction are devised: (i) Consider envelopes with higher reward first, (ii) Consider envelopes with smaller reward first and (iii) Chose an envelope type at random. The order of inserting the coupons into envelopes is done based on a heuristic function, which is defined as the ratio between the *coupon value* and the *minimal number of coupons needed to fill a given envelope type*. When a coupon needs to be inserted into a given envelope, all the left coupons are sorted based on this heuristic value, and afterwards, the best *pc* percent of them are selected as candidates for getting into the current envelop. In the next step, one of the coupons from this set is selected at random (with uniform probability) for insertion into the current envelope. A given envelope is considered to be complete when it satisfies its minimum type constraints, and, when that is the case, a new envelope starts filling up. This process is repeated until all coupons are inserted in one of the envelopes.

The **local search mechanism** (lines 14 to 20 in algorithm 2) uses a *swap* operator, which swaps two groups of coupons belonging to two distinct envelopes, where the selection of individual envelopes and the size of the groups is made at random. During the local search phase, the swap operator is applied repeatedly, until the foreseen number of iterations without improvement (as specified by *mls* parameter) is exceeded.

The GRASP algorithm is developed by using Ruby on Rails programming language over RubyMine IDE.

Algorithm 2 Greedy Randomized Adaptive Search Procedure

Require: coupons C (N, v_i, T, C_j, S_j, R_j), where i=1…N and j=1…T; max iterations *mi*; max local search iterations with no improvement *mls*; percentage of components chosen randomly *pc*; envelope filling *mode*.

1: *Best* = ∅;
2: **repeat**
3: S = ∅;
4: **repeat**
5: E = ∅;
6: **repeat**
9: C' = Select the best *pc*% coupons in C not yet inserted
10: $E = E \cup$ coupon chosen uniformly at random from C' based on *mode*
11: **until** E is valid **or** no more coupons are available
12: $S = S \cup E$
13: **until** S is a complete solution
14: **for** *mls* times **do**
15: R = Apply the swap operator in a copy of S
16: **if** *Quality(R)* > *Quality(S)* **then**
17: $S = R$
18: **if** *Best* == ∅ **or** *Quality(S)* > *Quality(Best)* **then**
19: *Best* = S
20: **until** *mi* is reached
21: **return** *Best*

#	Type of envelope	Reward (€)	List of coupons	Sum of envelope
1	3	2	3,2,7,1,15,5	33
2	2	1.5	6,7,8,5	26
3	1	1	9,7,4	20
4	1	1	10,12	22
		Total reward: 5.5		

Table 5. A sample solution for the simple scenario of 15 coupons returned by GA approach

#	Type of envelope	Reward (€)	List of coupons	Sum of envelope
1	2	1.5	10, 12, 3, 1	26
2	2	1.5	7, 15, 4, 2	28
3	2	1.5	7, 9, 6, 4	26
4	1	1	7, 8, 5	20
		Total reward: 5.5		

Table 6. A sample solution for the simple scenario of 15 coupons returned by GRASP approach

C. Description of the solution for the simple scenario

In order to clarify the applicability of the presented approaches (i.e. GA and GRASP), in Tables 5 and 6, respectively, we present the solutions returned by them for the simple scenario of 15 coupons that was presented in the introduction section. As it can be seen in the respective tables, both approaches return equally optimal solutions, having the quality (i.e. total reward) of 5.5, which is also equal to the manual solution that was presented in the introduction section. Furthermore, both approaches produce solutions with four envelopes, though the difference lies in the type of used envelopes, where the GA approach uses two envelopes of type 1, one envelope of type 2 and one envelope of type 3, whereas the GRASP approach uses one envelope of type 1 and three envelopes of type 2. Nevertheless, both approaches have been executed 10 times, therefore the particular solutions returned by them varies from one execution to the other one, but the quality of the solutions has been always the same (i.e. 5.5). Hence, this result should only be seen from a didactic perspective that serve only for the purpose of explanation of the structure of a given solution, while an extensive experimental study, as presented in the next section, is meant

to express the level of effectivity and efficiency of the proposed approaches.

V. Computational experiments

In this section, we initially present a test set of 10 instances that are used for conducting the evaluations of the solution presented in this paper. Further, we show the computational results for tuning the parameter values of the proposed approach. After that, we compare the obtained results against the upper bound values that are within the reach, when relaxing individual hard constraints of the problem at hand.

A. Test set

In order to test the algorithm for various scenarios of the distribution of fiscal coupons, we have set up a test set that consist of 10 different instances, where the values of individual coupons are generated randomly. Table 2 shows the characteristics of individual instances, which includes instance name, number of coupons and the total value of all coupons. The instance name, in addition to problem abbreviation DFCP, also encompasses the number of coupons and the total value present in a particular instance, e.g. Instance DFCP_2h_3k contains 200 (2hekta - 2h) coupons with a total value of 3000 (3kilo - 3k) currency units.

In practice, the value of a fiscal coupon ranges from very small amounts (e.g. a chewing gum might cost less than a euro) to large amounts (e.g. a technological appliance might cost several, dozens, hundreds or even thousands of euros). However, during a certain period of time (e.g. a month or a year quartile), the number of large value transactions (i.e. fiscal coupons) made by a person is usually much lower than the number of transactions with small values. Hence, in order to make the test instances more realistic, 30% of coupons are set to have larger values, which range from several up to dozens of currency units (e.g. euros).

Furthermore, based on the constraints enforced in practical situations, such as in the case of Tax Authorities of the Republic of Kosovo [1], three envelope types are defined throughout all test instances. In general, an envelope type is described with three properties, namely the minimum number of coupons, the minimum sum of the coupons and the foreseen reward. In particular, the types of envelopes utilized in the test set are described in the following:

Type1= {30, 250, 10},
Type2= {40, 500, 15} and
Type3= {50, 800, 20}.

B. Upper bound limits

In addition, in Table 7, we present the maximal reward that can be achieved per instance if individual problem constraints are relaxed (i.e. either the constraint for the sum or number of coupons in the envelope is not enforced). In case the constraint for the sum of coupons is relaxed (i.e. it is not taken into account), the maximal reward that can be achieved, in all instances, is when the envelopes are all of Type3 (i.e. the number of coupons is 50). On the other hand, when the constraint for the minimum number of coupons is relaxed, the best scenario, in all instances, is when all the envelopes are of Type1 (i.e. the minimum sum of coupons is 250). If the constraint for the minimum sum of coupons is relaxed then the formula for calculation of upper bound values is *UB = [No. coupons] / [Min. no. of coupons per envelope type]*[Reward per envelope type],* otherwise, if the constraint for the minimum number of coupons is relaxed the envisioned formula is *UB = [Total value] / [Min. sum of coupons per envelope type]*[Reward per envelope type]*. In the case of relaxation of the minimum sum of coupon constraint, a sample calculation of the upper bound value for instance DFCP_2h_2k (the sixth column in Table 7) is *UB*=200/50*20=80. Comparing the values in the sixth and the seventh column of Table 7, one can notice that the scenario of having envelopes of Type1 (i.e. the sum of coupons is 250) while relaxing the constraint for the minimum number of coupons, is the best scenario for all instances in the test set. Hence, in the following section, we use these values as Upper Bound (UB) limits (i.e. benchmark values) for evaluating the results that are obtained by the introduced solution in this paper.

C. Parameter settings

In order to fine tune the values of the parameters of the GA and GRSAP approaches, a systematic experimentation is performed by using the complete test set. Initially, based on some preliminary experimentation, for each parameter, a range, consisting of several best performing values, is selected. Then, for each selected value, the algorithm is executed for each test instance 10 times. As a result, for each single parameter, the value that on average produces better results than the other considered values, is adapted for the final round

Instance name	Instance details		Envelope details					
			Number of coupons			Sum of coupons		
	Number of coupons	Total value	30	40	50	250	500	800
DFCP_2h_2k	200	2000	60	75	80	**80**	60	50
DFCP_2h_3k	200	3000	60	75	80	**120**	90	80
DFCP_5h_5k	500	5000	160	180	200	**200**	150	120
DFCP_5h_6k	500	6000	160	180	200	**240**	180	150
DFCP_1k_10k	1000	10000	330	375	400	**400**	300	250
DFCP_1k_11k	1000	11000	330	375	400	**440**	330	280
DFCP_2k_20k	2000	20000	660	750	800	**800**	600	500
DFCP_2k_22k	2000	22000	660	750	800	**880**	660	550
DFCP_5k_50k	5000	50000	1660	1875	2000	**2000**	1500	1250
DFCP_5k_55k	5000	55000	1660	1875	2000	**2200**	1650	1380

Table 7. Test set details and maximal reward when relaxing individual constraints

of the experimentation that is done with the aim of evaluating the performance of the proposed algorithm. In this section, we average results over the complete data set, by using the average result of a given instance, that runs for 10 times when experimenting with a specified value of a given parameter.

For all six parameters of the GA approach, during the preliminary experimentation, five best performing values are selected. As depicted in Figure 2, the best performing values for the *maximum generations* and *population size* parameters are 10000 and 5000, respectively. Further increasing the values of these two parameters only costs longer computation

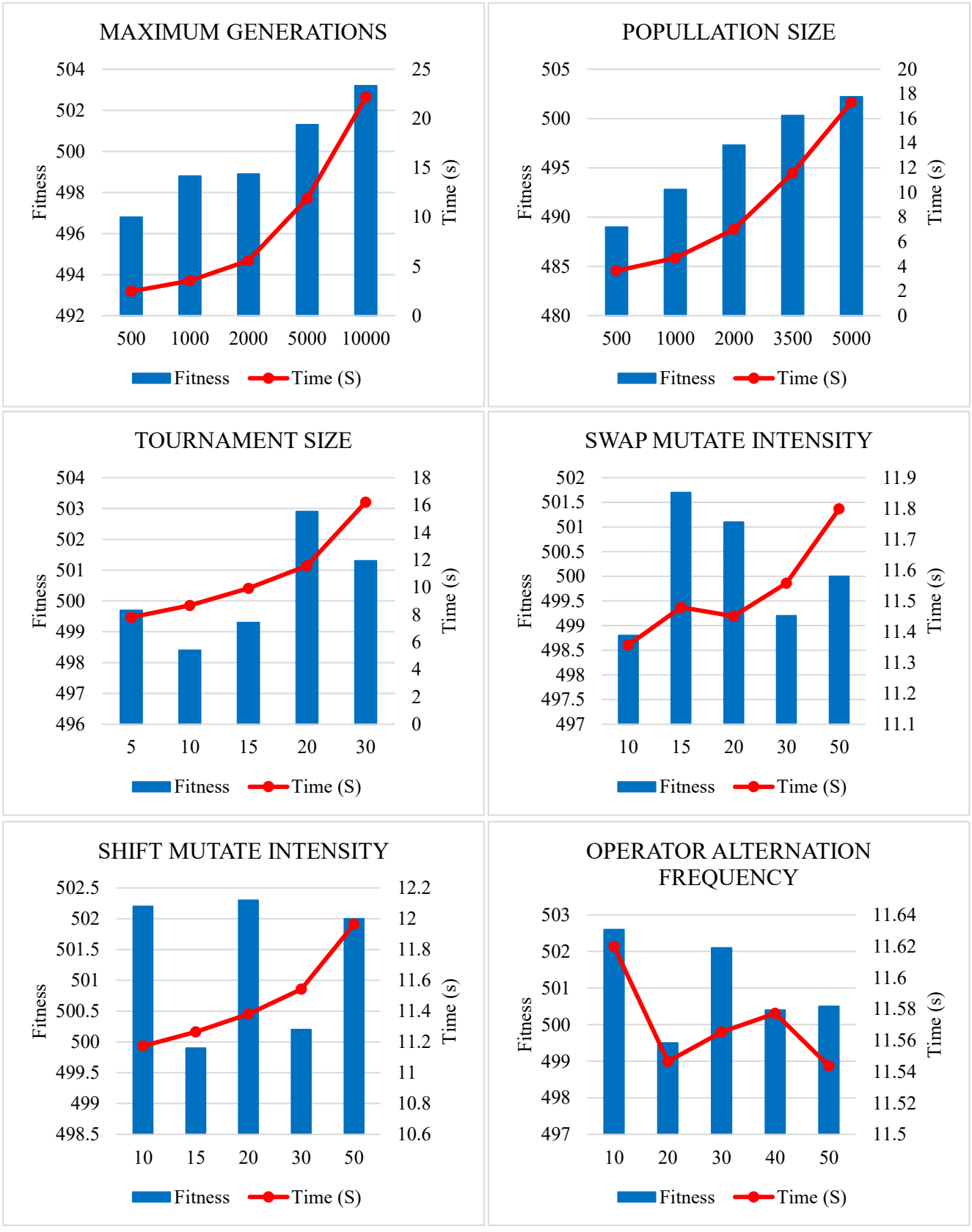

Figure 2. GA parameter settings

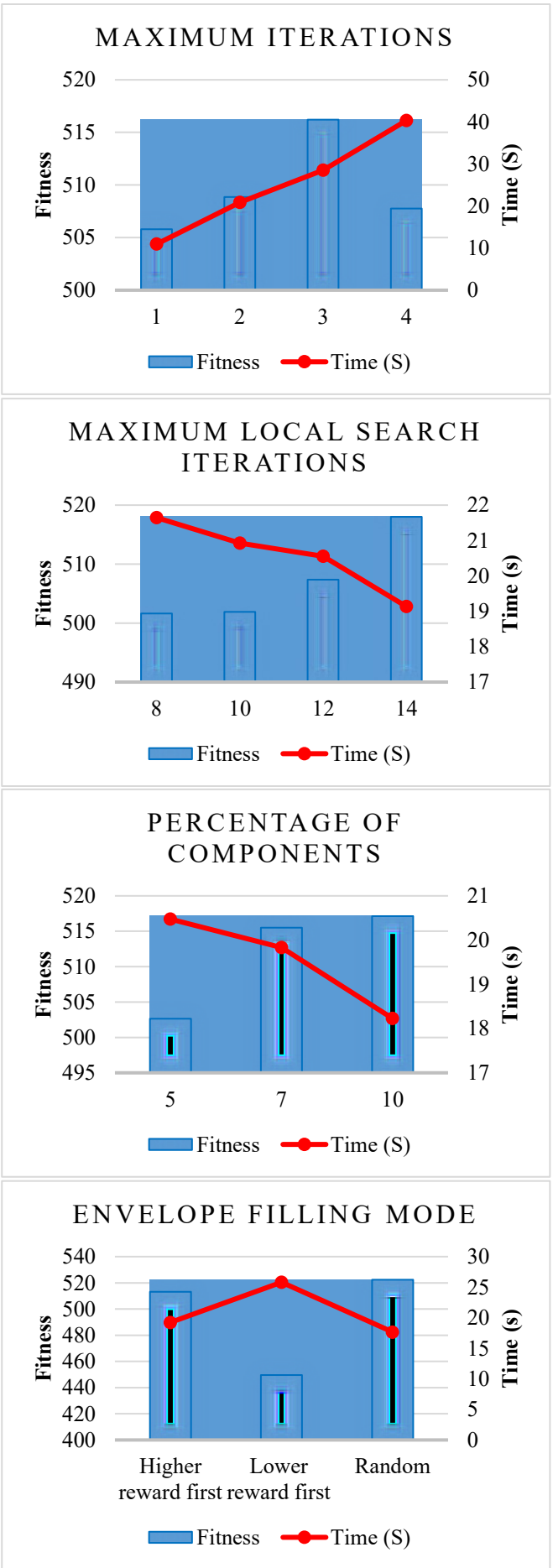

Figure 3. GRASP parameter settings

time, while no further improvement can be reached. The best value for the *tournament size* parameter is 20, while for the *swap* and *mutation intensities*, the best values are 15 and 20 respectively. In regard to the alternation of the operators, based on the experiments with *operator alternation frequency* (*oaf*) parameter, whose best value is 10, it is obvious that it is better to alternate (between shifting and swapping) more frequently. In terms of the computation time, the higher the value of a given GA parameter is, the slower the algorithm becomes, except for the parameter of *operator alternation frequency*, whose values seem not to make any noticeable impact in this regard, as the maximal difference, from the fastest scenario (*oaf*=50) to the slowest one (*oaf*=10) is only 0.12 seconds.

On the other hand, the *maximum iterations* parameter of the GRASP approach has been tested against four different values, and, as it can be seen in Figure 3, the best scenario (in terms of fitness and time) is when the value of this parameter is 3. Further, the experiments with the parameter of *maximum local search* iterations show that the more iterations without improvement runs the GRASP algorithm, the better the quality of the solutions becomes. This is also the case of the parameter of *percentage of components*, where its best (tested) value is 10. In terms of *mode of envelope filling*, the experiments show that the best way to fill the envelope is the random one. In regard to the computation time, in general the higher the value of the parameter the better the computation time, expect normally for the *maximum iterations* parameter. Also, the random way of envelope filling, in average, makes the GRASP approach run faster than any of the two other modes.

A. Comparison of results against upper bounds

In Table 8, we present the averaged results for individual instances over ten executions in each of the two algorithms (i.e. GA and GRASP). Further, the results are compared against the upper bound values that were described in the previous section. In general, it can be noticed that the GA approach produces better results than the GRASP approach for smaller instances with up to 1,000 coupons and a sum of up to 11,000 currency units. Whilst, for larger instances, the GRASP approach shows to perform better than the GA approach, where the difference, in favor of GRASP approach, goes up to more than 6.61% for instance DFCP_2k_22k, except for instance DFCP_5k_50k, where the difference, in favour of GA approach, is 2.84%. When the results are averaged over the whole test set, the gap of GA and GRASP from upper bound values is 29.16% and 32.01%, respectively. This gap should be considered as relative, since the upper bound values do not represent actual solutions to the problem, but only the solutions to the relaxed version of it. With regard to comparisons of the two approaches presented in this paper, the experimental results show that the GA approach, in overall, has an advantage of 3.74% in comparison to the GRASP approach.

Table 9 shows the details concerning the best returned results over all executions of both algorithms. As in the case of the averaged results, also in the case of the best results, the GA approach performs better than the GRASP approach for smaller instances, whereas for larger instances the GRASP approach is better, which drives up to almost 13% for instance

DFCP_5k_55k. Further, in average, the best results that can be achieved by the two approaches, in comparison to upper bound values, have the gap of 24.99% and 25.15%,

Instance name	Upper bound (UB)	GA	GRASP	GA vs. UB (%)	GRASP vs. UB (%)	GRASP vs. GA (%)
DFCP_2h_2k	80	61.82	53.0	22.73	33.75	14.26
DFCP_2h_3k	120	71.26	70.0	40.62	41.67	1.77
DFCP_5h_5k	200	159.57	146.0	20.21	27.00	8.50
DFCP_5h_6k	240	172.10	152.0	28.29	36.67	11.68
DFCP_1k_10k	400	297.20	289.5	25.70	27.63	2.59
DFCP_1k_11k	440	317.77	300.0	27.78	31.82	5.59
DFCP_2k_20k	800	574.25	582.0	28.22	27.25	-1.35
DFCP_2k_22k	880	613.43	625.0	30.29	28.98	-1.89
DFCP_5k_50k	2000	1348.26	1310.0	32.59	34.50	2.84
DFCP_5k_55k	2200	1427.17	1521.5	35.13	30.84	-6.61
			Avg.	29.16	32.01	3.74

Table 8. Results of GA and GRASP versus upper bound limits (averaged over ten runs)

Instance name	Upper Bound (UB)	GA	GRASP	GA vs. UB (%)	GRASP vs. UB (%)	GRASP vs. GA (%)
DFCP_2h_2k	80	**65**	60	18.75	25.00	7.69
DFCP_2h_3k	120	**75**	**75**	37.50	37.50	0.00
DFCP_5h_5k	200	**170**	160	15.00	20.00	5.88
DFCP_5h_6k	240	**185**	165	22.92	31.25	10.81
DFCP_1k_10k	400	**330**	**330**	17.50	17.50	0.00
DFCP_1k_11k	440	**345**	330	21.59	25.00	4.35
DFCP_2k_20k	800	600	**660**	25.00	17.50	-10.00
DFCP_2k_22k	880	645	**660**	26.70	25.00	-2.33
DFCP_5k_50k	2000	1365	**1435**	31.75	28.25	-5.13
DFCP_5k_55k	2200	1470	**1660**	33.18	24.55	-12.93
			Avg.	24.99	25.15	-0.16

Table 9. Results of GA and GRASP versus upper bound limits (best run scenario)

respectively. This shows that for the best case scenario the GRASP approach slightly outperforms the GA approach for an average margin of 0.16%. Nonetheless, the GA approach is also quite competitive to the GRASP approach, since its results are either better or equal in 6 (out of 10) instances.

In Table 10, we show the average computation times (derived from 10 executions) for both approaches. The results show that the GA and GRASP approaches need about 27.79 and 13.21 seconds, respectively, to solve the DFCP problem. These results indicate that, in overall, the GRASP approach is about 2.1 times faster than the GA approach for the envisioned test set. Nonetheless, it is evident that for smaller instances, the GA approach takes a considerably longer computation time than GRASP approach, which, in the worst case, can be up to 67 times slower.

Instance name	GA (S)	GRASP (S)	GRASP/GA
DFCP_2h_2k	21.44	0.32	67.09
DFCP_2h_3k	21.01	0.32	65.89
DFCP_5h_5k	21.53	1.01	21.24
DFCP_5h_6k	22.06	1.01	21.85
DFCP_1k_10k	26.37	2.86	9.21
DFCP_1k_11k	24.41	2.90	8.42
DFCP_2k_20k	30.21	9.16	3.30
DFCP_2k_22k	32.46	8.01	4.05
DFCP_5k_50k	39.95	59.57	0.67
DFCP_5k_55k	38.43	46.90	0.82
Avg.	27.79	13.21	2.10

Table 10. Computation time of GA and GRASP

The worst case execution scenario of GA and GRASP, always remains under a computation time of less than 40 and 60 seconds, respectively. This shows that both algorithms can be used in practice, where generating good quality solutions would enable the user to gain more revenue from the process of coupon collection that is applied in tens of countries around the globe (e.g. Republic of Kosovo [1]).

VI. Conclusion and Future Work

In this paper, we introduced a new problem for modelling the optimal distribution of fiscal coupons and devised a MILP mathematical formulation. Further, we presented two metaheuristic approaches based on GA and GRASP algorithms, which are able to solve the formulated problem at hand in matter of few seconds by using standard computing devices. In addition, a newly introduced test was used for benchmarking purposes, where it was shown that both approaches produce competitive results. On average, the GA approach is better than GRASP approach for around 3.74%, although, with regard to best returned results, the GRASP approach returns slightly better results than the GA approach for 0.16%, if the results are averaged over the complete test set.

For additional comparison, as part of future work, we plan to develop exact methods from the field of dynamic programming and investigate hybridization of the presented approaches, as well as utilization of constraint satisfaction problem (CSP) techniques within the existing metaheuristics for the envisioned problem.

Acknowledgment

We thank the former student of the University of Prishtina Sejfi Hoxha for developing a tool for generation of test cases for the Problem of Distribution of Fiscal Coupons.

References

[1] Tax Administration of Kosovo, "Tax Administration of Kosovo," 2015. [Online]. Available: http://www.atk-ks.org/.

[2] Minnesota Department of Revenue, "Sales Tax Fact Sheet 167," 2015. [Online]. Available: http://www.revenue.state.mn.us/businesses.

[3] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing,* vol. 35, no. 3, pp. 713-728, 2005.

[4] W. Yeoh, A. Felner and S. Koenig, "BnB-ADOPT: An asynchronous branch-and-bound DCOP algorithm," in *roceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, 2008.

[5] R. Mailler and V. Lesser, "Solving distributed constraint optimization problems using cooperative mediation," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, 2004.

[6] T. Le, T. Cao Son, E. Pontelli and W. Yeoh, "Solving distributed constraint optimization problems using logic programming," *Theory and Practice of Logic Programming,* vol. 17, no. 4, pp. 634-683, 2017.

[7] S. Yang, Q. Liu and J. Wang, "IEEE Transactions on Automatic Control," *IEEE Transactions on Automatic Control,* vol. 62, no. 7, pp. 3461-3467, 2017.

[8] Y. Chen and J.-K. Hao, "Memetic search for the generalized quadratic multiple knapsack problem," *IEEE Transactions on Evolutionary Computation,* vol. 20, no. 6, pp. 908-923, 2016.

[9] M. Avci and S. Topaloglu, "A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem," *Computers & Operations Research,* vol. 83, pp. 54-65, 2017.

[10] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *SFCS'08. 23rd Annual Symposium on. IEEE*, 1982.

[11] A. Lodi, S. Martello and M. Monaci, "Two-dimensional packing problems: A survey," *European Journal of Operational Research,* vol. 141, no. 2, pp. 241-252, 2002.

[12] Z. Kızıltan and B. Hnich, "Symmetry breaking in a rack configuration problem," in *IJCAI-2001 Workshop on Modelling and Solving Problems with Constraints*, 2001.

[13] Q. Hyseni, S. Yildirim Yayilgan, B. Krasniqi and K. Sylejmani, "Solving the Problem Of Distribution Of Fiscal Coupons By Using A Steady State Genetic Algorithm," in *8th International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA2017)*, Marrakech, Morocco, 2017.

[14] D. Whitley and J. Kauth, "GENITOR: A different genetic algorithm," in *1988 Rocky Mountain Conference on Artificial Intelligence*.

[15] S. Luke, Essentials of Metaheuristics, Second ed., Lulu, 2013.

[16] T. Feo and M. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization,* vol. 6, no. 2, pp. 109-133, 1995.

Author Biographies

Kadri Sylejmani was born in May 1977**.** He did his PhD thesis at Vienna University of Technology, Faculty of Informatics. His main areas of research interest include algorithm design and development for hard optimization problems. He works as professor assistant at Faculty of Electrical and Computer Engineering in University of Prishtina, where he lectures courses in various subjects within the ICT domain, such as programming languages, algorithms and data structures, nature inspired algorithms, e-commerce and legal, ethical and social issues in ICT.

Qëndresë Hyseni was born in May 1992. She received her Bachelor degree in Telecommunications Engineering from University of Prishtina, Kosovo, in 2013. Currently she is doing research for her Master thesis in Computer Engineering, again from University of Prishtina, Kosovo. Her research covers mainly metaheuristics and optimization algorithms.

Sule Yaldirim Sule Yildirim was born July 1971. She is associate professor at the Norwegian University of Science and Technology (NTNU) at the Department of Information Security and Communication Technology. Her main fields of research interests are artificial intelligence, application of machine learning in various fields, signal and image processing and biometrics. She has participated in projects funded by EU Horizon 2020, Eurostars and Erasmus+ programs, the Research Council of Norway, the Regional Research Council of Norway and the Ministry of Foreign Affairs, Norway. She belongs to the Norwegian Information Laboratory, Center for Cyber Information Security and the Norwegian Biometrics Laboratory. She has been supervising students at different academic levels over 20 years now and has been publishing more than 80 journal and conference papers in her fields of research. She also actively takes part as PC in conferences and acts as reviewer in several journals

Agon Qurdina was born in December 1992. He received his Bachelor degree in Computer Engineering at University of Prishtina, Kosovo, in 2014. Currently he is doing research for his Master thesis in Computer Engineering, again at University of Prishtina, Kosovo. The research is based on neural networks' application in specific domains.

Lekë Mula was born in July 1993. He received his Bachelor degree in Computer Engineering from University of Prishtina, Kosovo, in 2015. He is currently working on finishing his master degree thesis in the same university. His research covers many recommendation algorithms on improving users search relevance in different areas.

Bujar Krasniqi was born in October 1981. He received his PhD degree in Electrical Engineering from Vienna University of Technology, Austria in 2011. During the academic year 2012-2013 he has been engaged as guest lecturer, and after that he has become Professor in Faculty of Electrical and Computer Engineering, University of Prishtina, Kosovo. His main research interests are wireless communication, green communications, optimization, algorithms etc.