ITC 3/48
Journal of Information Technology and Control
Vol. 48 / No. 3 / 2019
pp. 389-400
DOI 10.5755/j01.itc.48.3.21893

A New Genetic Algorithm with Agent-Based Crossover for the Generalized Assignment Problem

Received 2018/10/19 | Accepted after revision 2019/08/03

http://dx.doi.org/10.5755/j01.itc.48.3.21893

A New Genetic Algorithm with Agent-Based Crossover for the Generalized Assignment Problem

Murat Dörterler

Gazi University, Faculty of Technology, Department of Computer Engineering, Teknikokullar, 06560, Ankara, Turkey, Phone Number: +90 312 202 85 95, Fax Number: +90 312 202 89 47; e-mail: dorterler@gazi.edu.tr

Corresponding author: dorterler@gazi.edu.tr

Generalized assignment problem (GAP) considers finding minimum cost assignment of n tasks to m agents provided each task should be assigned to one agent only. In this study, a new Genetic Algorithm (GA) with some new methods has been proposed to solve GAPs. The agent-based crossover is based on the concept of dominant gene in genotype science and increases the fertility rate of the feasible solutions. The solutions are classified as infeasible, feasible and mature with reference to their conditions. The new local searches provide not only feasibility in high diversity but high profitability for the solutions. A solution is not given up through maturation-based replacement until it reaches its best. The computational results show that the agent-based crossover has much higher fertility rate than classical crossover. Finally, the proposed GA creates either optimal or near-optimal solutions.

KEYWORDS: Generalized Assignment Problem; Genetic Algorithm; Agent-Based Crossover.

1. Introduction

GAP is a well-known, NP-complete combinatorial optimization problem [23]. It is a type of one-to-many assignment problems that recognizes capacity limits [21]. GAP considers finding minimum cost assignment of n tasks to m agents provided each task should be assigned to one agent only. On the other hand, an agent may be assigned more than one task, subject to the agents' available capacity [22]. The GAP has several applications in real life, solved by exact and heuristic algorithms. Nevertheless, heuristic algorithms are more capable of solving large-scale GAPs than exact algorithms [5, 19].

Genetic Algorithm (GA) is one type of the heuristic

algorithms and there are a few GAP studies involving GA. Additional operators are used in these studies as standard GA is not capable of solving GAPs. These studies roughly proposed various local search methods to overcome the incapability. The aims of the local search methods are that either an infeasible solution is evolved into feasible one, or a feasible solution is evolved into one with its fittest value. In addition, classical crossover method is used in the studies even though the classical crossover method most probably causes to create infeasible solutions from feasible ones especially in difficult GAP instances. Moreover, a candidate solution can lose its place in population at the end of the selection and replacement methods without any improvement.

In this study, a new GA has been proposed by developing some new methods. The methods involve agent-based crossover, local search methods and maturity-based replacement. The local search methods aim to improve feasibility, efficiency and diversity of the solutions. Each candidate solution was evaluated and passed through some new improvement processes. The agent-based crossover is based on the concept of dominant gene in genotype science. When a solution in population reaches its best fitness value, it is judged as a mature solution and agent-based crossover is triggered. The offspring solutions are created through applying agent-based crossover process to a pair of parent. When the mature solution is thrown out of the population at the end of the agent-based crossover, offspring solution takes its place in population. The proposed methods provide diversity of candidate solution; thus, a near-optimal solution can be obtained in lesser execution time.

Section 2 presents GAP and the previous studies on GAP. Section 3 elaborates GA and the new method. Section 4 deals with the results of the computational studies. Section 5 is about the experimental results and analyses.

2. The Generalized Assignment Problem

The GAP can be formulated as an integer linear program beside its definition presented in the previous section:

$$\textit{Minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1}$$

$$\textit{Subject to:} \quad \sum_{i=1}^{m} x_{ij} = 1 \quad j = 1,\ldots,n, \tag{2}$$

$$\sum_{j=1}^{n} a_{ij}x_{ij} \le b_i \quad i = 1,\ldots,m, \tag{3}$$

$$x_{ij} \in \{0,1\}\ i=1,\ldots,m;\ j=1,\ldots,n, \tag{4}$$

where c_{ij} is the cost of assigning agent i to task j, and, if agent i is assigned to task j, x_{ij} = 1, else 0; a_{ij} is the amount of agent i's capacity consumed by task j in case it is assigned to i. Finally, b_i is the available capacity of agent i. The first constraint indicates that each task is assigned to only one agent; the second constraint is that the resource demand of the tasks, which are assigned to an agent, must not exceed the capacity of the agent.

Ross and Soland [22] have come up with GAP. Such real life problems as vehicle routing [10], grouping and loading for flexible manufacturing systems [17], assigning ships to overhaul [12], assigning jobs to computers in computer networks [2], land use allocation [6] relate to GAP. The chief applications and algorithms about GAP are presented in the survey study of Cattrysse and Van Wassenhove [4].

Exact algorithms have been proposed to solve GAPs. Branch-and-price algorithm [25], cutting plane algorithm [1], branch-and-cut algorithm [18], branch-and-bound [11] have been the significant ones of them in the last two decades. The existing exact algorithms are not effective for more difficult large-scale problems [15].

The large-scale GAPs are currently handled through heuristic and metaheuristic methods to get the most near-optimal solutions. Osman [19] designed both tabu search and hybrid simulated annealing/tabu search methods for GAP. Moreover, tabu search was used by Diaz and Fernandez [7], dynamic tabu search was proposed by Higgins [13] and a hybrid tabu search/branch-and-bound was implemented by Woodcock and Wilson [29]. Differential evolution algorithm [27, 14, 26], bees algorithm [20], the neighbourhood search algorithm [32] and ejection chain approach [31] are other notable heuristic approaches applied to GAP.

The studies which apply GA to GAPs benefit from additional processes [5, 16, 28, 9]. As classical GA violates the capacity constraint (3), it creates infeasi-

ble solutions in high probability. Chu and Beasley [5] used GA involving classical crossover, binary tournament selection and single point mutation operators. Authors presented not only fitness function but also unfitness functions to measure infeasibility of a solution. They also proposed two local search methods, which were applied to each created solution at the end of the crossover method. The first one aim to turn into feasible solutions from infeasible ones. The second one aims to reduce the cost of the solution. The local searches are applied to the solutions for once and they do not fulfil their aims sufficiently.

Wilson [28] used classical crossover and tournament selection methods. He proposed a 2-phase local search method, sharing the similar aims with the local searches of Chu and Beasley [5]. The first phase improves the feasibility of the selected solution; the second phase seeks the lower cost value for the solution. GA process is interrupted if any feasible solution is obtained. This feasible solution is selected for the 2-phase local search. Otherwise, the best solution with the best fitness value is selected at the end of the GA process. The local search is only applied to the selected solution if the solution is not feasible. It is unlikely to get a near-optimal solution in case a feasible solution is obtained.

Feltl and Raidl [9] proposed several improvements for the study of Chu and Beasley [5]. They presented two alternative heuristic methods to increase proportion of feasible individuals in initial population. Selection and replacement schemas were suggested to eliminate infeasible individuals while creating an initial population. In addition, heuristic mutation solution was used to decrease the probability of turning feasible individuals into infeasible ones. They also presented a selection and replacement strategy to eliminate infeasible solutions aggressively. In their study, classical crossover method was used to generate an offspring where this operation causes infeasible solutions.

Dorterler et al. [8] proposed a new GA for a real problem seen as a special case of the GAP. The authors emphasised that classical crossover method violated the capacity constraint. Thus, Nucleotide Exchange operator was proposed instead of the classical crossover operator in the same study. A gene keeps multiple significant data, each of which is called nucleotide, as designated in the natural sciences. The operator creates offspring from a single parent by swapping the nucleotides between genes at the crossover stage. Even if experiential results are successful, the heuristic is not suitable for applying to classical GAP without some modifications.

3. The Proposed Heuristic

GA provides more near-optimal solutions with comparatively much lower computational complexity than deterministic methods. GA aims to get a number of solutions, which provide better results by iteration. GA is inspired by natural selection; thus, the same terminology is used. Solutions are named chromosomes and the set of solutions is named population. Chromosome is composed of meaningful data string; each one is called a gene. New solutions called offspring are created by means of crossover method in each iteration. Crossover creates offspring from the genes of its parent chromosomes selected by any selection method in population. Mutation process changes one or more genes of the offspring randomly in predefined probability, lest the GA process is trapped in local minima. The new generation for next iteration is composed by replacing a number of offspring with chromosomes selected in population. Fitness function measures profitability of the chromosomes in population. GA process is executed until the termination condition is occurred. The termination condition should be the one which gets either a number of iterations or a desired fitness value [33].

3.1. Representation

The structure of a solution should be coded according to the problem in question at the initial stage of a GA design. Chromosomes are depicted as a numerical vector. Bit array structures are used to code the solution. Nevertheless, codding the genes as real data provides considerable advantages [24]. The studies [5, 28, 9] depicted the chromosomes as an integer vector; $S=\{S_1,S_2,...,S_n\}$, if n number of jobs need to be done. S_j indicates the agent to which job j is assigned, for $j=1,...,n$.

3.2. Elaboration of the Proposed GA

The main steps of the proposed GA algorithm for GAP are as follows:

Step 1: Generate an initial population consisting of randomly created solutions,

Step 2: Evaluate the condition of each solution in the population.

Step 3: Improve each solution according to its condition

If it is infeasible: Convert the solution into a feasible one

Else if it is feasible: Improve fitness value of the solution

Else if it is mature: Create an offspring by agent based crossover. Replace the mature solution by the created offspring.

Step 4: Repeat Steps 2 and 3 until satisfactory solution has been reached.

3.2.1. Initialisation of the Population

In this step, a certain number of solutions are created randomly as initial population. While each agent is assigned to a job randomly, the agent's capacities are not taken into account at initial phase. Hence, initial solutions are more likely to be infeasible due to violating the capacity constraint within (3). Successive steps are processed to improve feasibility, profitability of the solutions in guide of Algorithm 1.

3.2.2. Evaluation of the Solutions

Two indicators are needed to evaluate a GAP solution. The first one is cost, which is the aim of the problem. The other one is feasibility. In this study, the cost is defined as fitness value and feasibility is defined as provisional value. Provisional value (P_k) of solution k is defined as a boolean value and determined by

$$A_i = b_i - \sum_{j=1}^{n} a_{ij} x_{ij} \mid S_j = i \tag{5}$$

$$P_k = \begin{cases} 1, & \text{if } \forall A_i \geq 0 \mid i \in I \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Only if the solution is feasible is fitness value of the solution computed. Thus, the execution time decreases. Nevertheless, the worst-case computational complexity of this phase is $O(mn)$. The evaluation of a solution is implemented by Algorithm 1.

Algorithm 1
Evaluation of the solutions

```
1: for j=1 to n do
        // evolve into feasible if the solution infeasible
2:      if P_k=0 then
3:          search for an agent i* | a_i*j ≤ A_i*
4:          if such i* exists, select one of them randomly
5:          else select i* randomly;
6:          if i = i* then continue;
7:          rand ← [0,1];
8:          if (r_i*j < r_ij and rand > CSR) or (A_i < 0 and rand > CSR) then
9:              S_j ← i*;
10:             A_i ← A_i + a_ij; A_i* ← A_i* - a_i*j;
11:         end if
        // decrease the cost if the solution is feasible
12:     else
13:         select j* randomly;
14:         if S_j = S_j* then continue; | S_j = i; S_j* = i*;
15:         f'_k = f_k - c_ij + c_ij*;
16:         A'_i ← A_i + a_ij - a_ij*;
17:         A'_i* ← A_i* + a_ij* - a_ij;
18:         if f'_k < f_k and P'_k = 1 | A'_i ≥ 0, A'_i* ≥ 0 then
19:             swap ← S_j; S_j ← S_j*; S_j* ← swap;
20:             A_i ← A'_i; A_i* ← A'_i*;
21:             f_k ← f'_k;
22:         end if;
23:     end if;
24: end for;
```

3.2.3. Improvement of Solutions

Each solution in the population is evaluated according to its condition. A solution can be in three different conditions. These are infeasible, feasible, and mature. After the conditions of the solutions are detected in evaluation step, each solution is improved in terms of its condition.

Making a solution feasible: It is highly probable that most of the solutions in initial population are infeasible. Moreover, offspring created by agent-based crossover may be infeasible. The proposed local search in this study ensures that an infeasible solution is evolved into a feasible one. Moreover, it provides the diversity of the feasible solutions to ensure a near-optimal solution.

Shift neighborhood type local search, which is performed by changing the assigned agent of one job, is adopted for these aims [31]. In this study, the candidate agent for the job should have available capacity at the time (A_i) if possible. Moreover, the local search is dependent on two different alternative conditions, for the performance of using single condition is subject to change according to resources of the GAP in question. It is benefited from Condition Selection Ratio (CSR) indicating which condition is considered over what percentage. One of the conditions is fulfilled as the capacity of the agent assigned to the job in question is overloaded. The other condition is based on relative cost-resource value (r_{ij}) from [9]. The value is calculated by the formula (7):

$$r_{ij} = c_{ij} * \frac{a_{ij}}{b_i} . \quad (7)$$

The second condition is fulfilled if r_{i^*j} of the candidate agent is lower than r_{ij} of the agent assigned to the job.

Making a solution mature: The maturing a solution means minimising the fitness value of a solution as much as possible. In this approach, it is fixed that which agent executes how many jobs for a feasible solution. This situation does not change in this step. Hence, each feasible solution has its own ultimate cost value. It is judged that the solution reaches to the value when the maturation condition occurs for the solution.

Swap neighborhood type local search, which is performed by exchanging the assigned agents of two jobs, is adopted for maturation [30]. In this study, a second job is selected randomly together with the next one in iteration process. The local search depends on two conditions. Not only should fitness value (f_k') be lesser, but also feasibility (P_k') should be maintained for the subjected solution at the end of the exchange.

The maturation condition depends on the number of iterations in which a better fitness value is not obtained for the solution. The value is called Maturation Value (MV) and it affects not only the performance positively but also the execution time adversely. Hence, the MV increases from a predefined base value to predefined the top value gradually. The mechanism is defined in Algorithm 3 in Section 4.

3.2.4. Creating and Replacing Offspring

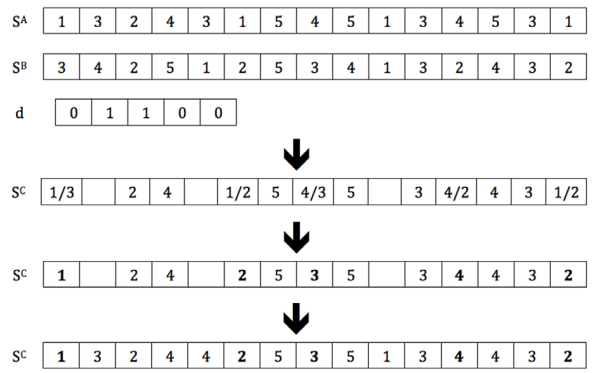
This step covers selection, creation, and replacement processes of GA. The trials showed that there is a low possibility in operating classical crossover to get a better feasible solution, for the capacity constraint is more likely to make it infeasible. Therefore, agent-based crossover method is designed and performed in this study to increase the possibility. When a solution is judged as a mature one, it should be replaced with an offspring. Two parent chromosomes are selected by tournament selection method and agent-based crossover method is applied to the selected parents. The parents are selected as feasible solutions in population, if they exist.

Multi point ($1+\lfloor n/100 \rfloor$) mutation is applied to created offspring in terms of mutation rate. The mutation process is performed by assigning randomly selected agents to randomly selected jobs. Maturation-based replacement is proposed for replacement at the end of this step. The offspring solution replaces the mature one in the population. Hence, it is aimed that no solutions are thrown out of the population until it reaches its best fitness value.

The agent-based crossover method: The agent-based crossover is inspired from the concepts of dominant and recessive genes in genotype science. The dominant feature describes which certain phenotype to pass from parent to offspring. The agents of the parents are classified as dominant or recessive for the jobs due to the restriction based on agent capacity (3). The key point in this operation is the question of what jobs performed by which agents are passed on the offspring from which parent.

In Figure 1, the agent-based crossover is depicted. S^A and S^B are selected parents, S^C is the offspring for $n=15$ and $m=5$. $d_i = 0$ indicates the jobs performed by the agent i in solution S^A, these jobs are passed on to S^C.

Figure 1
Illustration of Agent-based Crossover



Else if $d_i = 1$ indicates the jobs performed by the agent i in solution S^B, these jobs are passed on to S^C. The agent-based crossover is described in Algorithm 2.

Algorithm 2
The Agent-based Crossover

```
1: for i=0 to m do
2:    d_i ← 0 or 1 randomly;
3:    A_i ← b_i;
3: end for;
4: for j=1 to n do
        // if a collision occured
5:      if d_i=0 ∧ d_i*=1 | S_j^A=i , S_j^B=i* then
6:              if c_ij < c_i*j then S_j^C ← i else S_j^C ← i*
8:              end if;
9:      else if d_i=0 then S_j^C ← i ;
10:     else if d_i*=1 then S_j^C ← i*;
11:     end if
12:     A_i'' ← A_i'' - a_i''j | S_j^C = i'' ;
13: end for
        //check unassigned jobs
14: for j=1 to n do
15:     if S_j^C = null then
16:         search for an agent i | a_i'j ≤ A_i'
17:         if such i' exists then select one with minimum c_i'j
18:         else select i' randomly;
19:                 S_j^C ← i';
20                  A_i' ← A_i' - a_i'j ;
21:             end if
22:     end if
23: end for
```

The agents of the parents are identified as dominant or not; this domination is based on the jobs performed by each agent. It is possible that some of the jobs of the offspring are assigned to either no agent or two agents after passing dominant features. If a collision occurs in assigning two agents to one job, the job is assigned to the agent, who has a lower cost for the job. On the other hand, the other agent is released and its current available capacity increases. If there is an unassigned job, any agent, having current available capacity for the job, is assigned the job. If current available capacities of the agents are still not enough for the job, a random agent is assigned to the job. The computational complexity of the agent-based crossover is *O(n)*.

4 Computational Results

Chu and Baeasley's test problem instances from OR library [3] were utilized in the present study to test and compare the proposed algorithm. These instances were used for the computational experiments in other previous studies. The problem instances are divided into two groups as small sized and large sized. The large sized instances are generated as both agent/job combination m=5, 10, 20 and n=100, 200 and four different resources types from A to E [5, 30]. The proposed algorithm is applied only to large sized instances; because, the small sized ones are not

challenging enough for contemporary computers.

In this computational study, the conditions are arranged according to the study by Chu and Beasely [5] as possible. The population size is always 100. The termination condition depends on Sterility Index (SI) indicating the number of offspring were created without getting any better fitness values. When SI is reached to 500000, GA is terminated. Maturation value (MV) varies between 30 and 50 and is updated according to Algorithm 3 for iterations. CSR value is determined as 0.1, 0.9, 0.1, 0.9 and 0.9 for the instance from A to E, respectively. The mutation probability is defined as 0.001.

Algorithm 3
Computation of Maturation Value

$MV_{base} \leftarrow 30$;
$MV_{interval} \leftarrow 20$;
$MV \leftarrow MV_{base}$;
//Update in each iteration
if $M - M_{base} < SI / (500000 / MV_{interval})$ ***then***
$MV \leftarrow M_{base} + SI / (500000 / MV_{interval})$
end if

The proposed algorithm was coded in Java programming language and run on Java JRE v. 1.8.0_25 for Windows x64 installed on a machine which has an Intel(R) Core(TM) i5 3.1 GHz CPU and a 6 GB of RAM. Minimum and maximum memory allocations are specified as 256 MB and 4096 MB, respectively, for the execution of the code.

Table 1 indicates the comparison of fertility between the proposed agent-based crossover and classical single point crossover. This comparison was conducted by applying the proposed algorithm to the 24 large sized GAP instances. After the agent-based crossover was performed, the classical crossover was applied to the same parent solutions. The percentage of the feasible offspring generated by the proposed crossover methods was much higher than the classical crossover method in 27 instances out of 30.

Table 2 indicates the solution values of the proposed GA applied to the large-sized GAP instances. The GA was performed 10 times for each instance. The obtained best costs are presented separately. Moreover, the averages of the execution times and the average number of iterations at the best solution obtained are presented. Both values are reasonable.

Table 3 shows the quality of solutions and a comparison of the results. The percentage of deviation (σ) between best solution value (S_i) and their known best-cost value (S_o) (8) is used to measure the quality of solutions:

$$\sigma = (S_i - S_o) / S_o * 100\% . \quad (8)$$

The average percentage of gap of the run's best solution is shown in Table 3 (σ_a), the standard deviation of the best solutions of trials (*stddev*) and the percentage gap of the best run's solution (σ_b) are presented as well.

The results of Type A instances are not included in Table 3, for both GAs find out the optimum cost for all six easy instances. Even if the agent-based method provides superiority over classical crossover method, as in Table 2, the percentage of the deviation values is partly better than those of Chu and Beasley's GA. One of the reasons for this would be that the trials for Type D were terminated when IS had reached 3×10^6. Nevertheless, the differences between the compared values are easily tolerable. Moreover, the standard deviation values of the proposed GA are better for almost all instances.

In terms of computational complexity, the evaluation of solutions and agent-based crossover have complexities of $O(mn)$ and $O(n)$, respectively. Thus, the heuristic has a complexity of $O(mn)$. The complexity of the Chu and Beasley's GA is $O(m^2n)$ [5]. On the other hand, the computational times are not compared because all the studies have different codes in different programming languages, different CPU times on different hardware, and different stopping criteria. In addition, larger sized GAP instances were used in a few previous heuristic studies. The answers to the objective questions would not change with such a large-sized problem. Additionally, testing with this type instances takes larger execution times. It is anticipated, in the present study, that this type of larger sized instances are more suitable for competition type papers.

Table 1
Percentages of feasible offspring created by Classical and Agent-based Crossover Methods

Prob. Type	Size (m/n)	Percentage of feasible offspring created by		Total number of created offspring
		Classical	Agent-based	
A	5/100	99.977	**99.998**	500155
	5/200	82.631	**99.996**	500080
	10/100	61.912	**99.994**	500210
	10/200	88.683	**99.995**	500254
	20/100	93.716	**99.996**	500392
	20/200	95.677	**99.994**	500843
B	5/100	10.161	**17.893**	1237979
	5/200	5.453	**11.875**	847653
	10/100	73.034	**97.788**	506485
	10/200	4.326	**17.333**	534634
	20/100	74.291	**99.985**	501314
	20/200	74.929	**99.999**	502193
C	5/100	16.716	**20.804**	533032
	5/200	7.991	**10.153**	540689
	10/100	6.784	**31.675**	896716
	10/200	3.660	**14.186**	820793
	20/100	9.881	**67.208**	938666
	20/200	4.418	**49.901**	1606478
D	5/100	3.813	**8.003**	1579449
	5/200	4.201	**8.479**	739419
	10/100	**3.229**	3.073	1248862
	10/200	**2.101**	1.604	1262487
	20/100	2.920	**10.579**	955999
	20/200	**2.014**	1.584	1084807
E	5/100	21.927	**73.621**	135829
	5/200	0.235	**0,627**	1226390
	10/100	0.474	**2.073**	584783
	10/200	0,173	**0.374**	1163414
	20/100	0.217	**5.620**	1223224
	20/200	0.265	**0.373**	729617

Table 2

Computational Results for Large Sized GAP Instances. o: integer optimum reached. b: best overall solution value

Prob Type	Size		Best solution in each of the 10 trials										Best Overall solution	Avg. Best Sol'n time (s)	Avg. Best Sol'n iter.
	m	n													
A	5	100	o	o	o	o	o	o	o	o	o	o	1698	0.06	187
		200	o	o	o	o	o	o	o	o	o	o	3235	0.16	280
	10	100	o	o	o	o	o	o	o	o	o	o	1360	0.14	501
		200	o	o	o	o	o	o	o	o	o	o	2623	0.30	552
	20	100	o	o	o	o	o	o	o	o	o	o	1158	0.19	650
		200	o	o	o	o	o	o	o	o	o	o	2339	0.48	867
B	5	100	1846	1848	1849	1848	1846	1848	o	1849	1846	o	1843	115	361061
		200	3567	3567	b	3564	3564	3566	3564	3567	3565	b	3561	725.7	122042
	10	100	o	o	o	o	o	o	o	o	o	o	1407	10.27	33802
		200	b	2857	2855	2856	2856	2858	2856	2858	2859	b	2853	753.23	1251610
	20	100	1168	1169	1169	o	1169	1168	1168	1169	1169	1169	1166	0.64	2110
		200	2344	2347	2345	b	2344	2345	2349	2345	2346	b	2343	2.02	3361
C	5	100	1935	1933	1932	o	1936	1935	1935	o	1933	1937	1931	241.24	256203
		200	3472	3472	3471	3469	3473	3473	3473	3474	3475	b	3470	491.14	816716
	10	100	1408	1409	1408	1407	b	1409	1407	1407	1408	1408	1404	199.04	627121
		200	2841	2834	2840	2836	2838	2838	2840	2839	2835	b	2831	1017.29	1646017
	20	100	1258	1257	1257	1255	1258	1257	1258	b	1255	1257	1251	221.19	691173
		200	2432	2428	2430	2431	b	2429	2432	b	2431	2430	2427	942.70	1429466
D	5	100	6472	6485	6478	6488	6485	b	6484	6487	6484	6483	6461	355.85	1034714
		200	13015	13010	13008	13022	13011	13013	13020	13015	b	13020	12991	2528.52	3453532
	10	100	b	6544	6558	6555	6546	6542	6556	6560	6551	6542	6536	293.34	896739
		200	b	12833	12836	12765	12756	12754	12788	12774	12788	12747	12740	1100.43	1769034
	20	100	6454	6440	b	6445	6441	6446	6439	6436	6441	6447	6419	205.93	641491
		200	12776	12765	12770	12764	12753	12747	12749	b	12759	12755	12733	1005.80	1569777
E	5	100	13141	13148	13092	13100	b	13151	13054	13114	13154	13033	12984	187.34	582696
		200	26320	26205	26429	26459	26280	26422	26475	26180	26424	b	25825	520.23	828994
	10	100	12142	12143	12179	12177	12164	12108	b	12518	12469	12171	12071	242.82	767136
		200	25088	25186	24740	24933	24948	25023	25071	25027	24970	b	24724	835.66	1242852
	20	100	b	8971	8984	9015	8986	8935	8982	9022	8969	9002	8906	139.24	455248
		200	23998	b	24109	25504	24094	24039	24055	24085	24062	24170	23784	1171.04	1969467

Table 3

Comparison of Chu and Beasley's GA and proposed GA. o: integer optimum reached

Prob Type	Size		Best Known*	Chu and Beasley's GA**			Proposed GA		
	m	n		σ_a	stddev	σ_b	σ_a	stddev	σ_b
B	5	100	1843	0.35	0.31	o	0.20	0.11	o
		200	3552	0.30	0.15	0.03	0.35	0.06	0.25
	10	100	1407	0.07	0.07	o	o	0.00	o
		200	2827	0.34	0.13	0.14	1.03	0.07	0.92
	20	100	1166	0.07	0.06	o	0.21	0.08	o
		200	2339	0.10	0.05	0.04	0.26	0.08	0.17
C	5	100	1931	0.38	0.21	o	0.15	0.10	o
		200	3456	0.23	0.11	0.06	0.47	0.05	0.38
	10	100	1402	0.29	0.26	0.07	0.39	0.10	0.14
		200	2806	0.48	0.13	0.29	1.11	0.11	0.89
	20	100	1243	0.51	0.26	0.08	1.07	0.17	0.64
		200	2391	0.62	0.19	0.25	1.62	0.07	1.51
D	5	100	6353	0.66	0.21	0.31	2.01	0.12	1.70
		200	12742	0.66	0.15	0.42	2.12	0.07	1.95
	10	100	6348	1.25	0.38	0.49	3.17	0.12	2.96
		200	12432	1.57	0.15	1.36	2.78	0.26	2.48
	20	100	6190	1.91	0.36	1.28	4.05	0.14	3.70
		200	12241	2.35	2.30	1.72	4.22	0.10	4.02
E	5	100	12681	n/a	n/a	n/a	3.28	0.45	2.38
		200	24930	n/a	n/a	n/a	5.50	0.79	3.59
	10	100	11577	n/a	n/a	n/a	4.89	0.30	4.26
		200	23307	n/a	n/a	n/a	7.13	0.62	6.07
	20	100	8443	n/a	n/a	n/a	6.32	0.41	5.48
		200	22379	n/a	n/a	n/a	7.42	0.46	6.27

* The best values are reported in [30].
** The values are figured out from [5].

5 Conclusions

This study has proposed a GA with new methods for GAP. Classical crossover method is given up and agent-based crossover is presented to increase the possibility of creating feasible solutions. The solutions are classified as infeasible, feasible, and mature. The fitness value is not computed for infeasible ones to shorten the execution time. Moreover, a solution is not thrown up from the population until it is matured. Moreover, it is ensured that all the infeasible solutions are turned into feasible ones by the proposed local search methods in the feasible solution diversity as much as possible.

The computational results show that the agent-based crossover is more effective in terms of creating feasible solutions. Further, the proposed GA is able to create optimal and near-optimal solutions for GAP. The proposed GA obtains the best solution in both the execution time and number of iterations. Additionally, the lower standard deviation values indicate the consistency of the algorithm. Prospective studies could use the agent-based crossover and maturity-based replacement methods with different local searches than the ones in the previous studies, and should develop new methods.

Acknowledgement

The author thanks Dr. Ömer Faruk Cantekin, member of the Academic Writing Centre at Gazi University, for his support to checking the manuscript in terms of vocabulary and grammar.

References

1. Avella, P., Boccia, M., Vasilyev, I. A Computational Study of Exact Knapsack Separation for the Generalized Assignment Problem. Computational Optimization and Applications, 2010, 45(3), 543-555. https://doi.org/10.1007/s10589-008-9183-8
2. Balachandran, V. An Integer Generalized Transportation Model for Optimal Job Assignment in Computer Networks. Operations Research, 1976, 24(4), 742-759. https://doi.org/10.1287/opre.24.4.742
3. Beasley, J. E. OR-Library. Imperial College of Science Technology and Medicine, London, U.K., http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html. Accessed on July 19, 2019.
4. Cattrysse, D. G., Van Wassenhove, L. N. A Survey of Algorithms for the Generalized Assignment Problem. European Journal of Operational Research, 1992, 60(3), 260-272. https://doi.org/10.1016/0377-2217(92)90077-M
5. Chu, P. C., Beasley, J. E. A Genetic Algorithm for the Generalised Assignment Problem. Computers & Operations Research, 1997, 24(1), 17-23. https://doi.org/10.1016/S0305-0548(96)00032-9
6. Cromley, R. G., Hanink, D. M. Coupling Land Use Allocation Models with Raster GIS. Journal of Geographical Systems, 1999, 1(2), 137-153. https://doi.org/10.1007/s101090050009
7. Díaz, J. A., Fernández, E. A Tabu Search Heuristic for the Generalized Assignment Problem. European Journal of Operational Research, 2001, 132(1), 22-38. https://doi.org/10.1016/S0377-2217(00)00108-9
8. Dörterler, M., Bay, Ö. F., Akcayol, M. A. A Modified Genetic Algorithm for a Special Case of the Generalized Assignment Problem. Turkish Journal of Electrical Engineering & Computer Sciences, 2017, 25, 794-805. https://doi.org/10.3906/elk-1504-250
9. Feltl, H., Raidl, G. R. An Improved Hybrid Genetic Algorithm for the Generalized Assignment Problem. Proceedings of the 2004 ACM Symposium on Applied Computing, (SAC'04), Nicosia, Cyprus, March 14-17, 2004, 990-995. https://doi.org/10.1145/967900.968102
10. Fisher, M. L., Jaikumar, R. A Generalized Assignment Heuristic for Vehicle Routing. Networks, 1981, 11(2), 109-124. https://doi.org/10.1002/net.3230110205
11. Fisher, M. L., Jaikumar, R., Van Wassenhove, L. N. A Multiplier Adjustment Method for the Generalized Assignment Problem. Management Science, 1986, 32(9), 1095-1103. https://doi.org/10.1287/mnsc.32.9.1095
12. Gross, D., Pinkus, C. E. Optimal Allocation of Ships to Yards for Regular Overhauls, Technical Memorandum 63095. Institute of Management Science Engineering, George Washington University, 1972, Washington, USA.
13. Higgins, A. J. A Dynamic Tabu Search for Large-Scale Generalised Assignment Problems. Computers & Operations Research, 2001, 28(10), 1039-1048. https://doi.org/10.1016/S0305-0548(00)00024-1

14. Jiang, Z.-Z., Xia, C., Chen, X., Meng, X., He, Q. A Discrete Differential Evolution Algorithm for the Multi-Objective Generalized Assignment Problem. Journal of Computational and Theoretical Nanoscience, 2013, 10(12), 2819-2825. https://doi.org/10.1166/jctn.2013.3284

15. Liu, L., Mu, H., Song, Y., Luo, H., Li, X., Wu, F. The Equilibrium Generalized Assignment Problem and Genetic Algorithm. Applied Mathematics and Computation, 2012, 218(11), 6526-6535. https://doi.org/10.1016/j.amc.2011.12.025

16. Liu, Y. Y., Wang, S. A Scalable Parallel Genetic Algorithm for the Generalized Assignment Problem. Parallel Computing, 2015, 46, 98-119. https://doi.org/10.1016/j.parco.2014.04.008

17. Mazzola, J. B., Neebe, A. W., Dunn, C. V. R. Production Planning of a Flexible Manufacturing System in a Material Requirements Planning Environment. International Journal of Flexible Manufacturing Systems, 1989, 1(2), 115-142. https://doi.org/10.1007/BF00223019

18. Nauss, R. M. Solving the Generalized Assignment Problem: An Optimizing and Heuristic Approach. Informs Journal on Computing, 2003, 15(3), 249-266. https://doi.org/10.1287/ijoc.15.3.249.16075

19. Osman, I. H. Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches. OR Spektrum, 1995, 17(4), 211-225. https://doi.org/10.1007/BF01720977

20. Özbakır, L., Baykasoğlu, A., Tapkan, P. Bees Algorithm for Generalized Assignment Problem. Applied Mathematics and Computation, 2010, 215(11), 3782-3795. https://doi.org/10.1016/j.amc.2009.11.018

21. Pentico, D. W. Assignment Problems: A Golden Anniversary Survey. European Journal of Operational Research, 2007, 176(2), 774-793. https://doi.org/10.1016/j.ejor.2005.09.014

22. Ross, G. T., Soland, R. M. A Branch and Bound Algorithm for the Generalized Assignment Problem. Mathematical Programming, 1975, 8(1), 91-103. https://doi.org/10.1007/BF01580430

23. Sahni, S., Gonzalez, T. P-Complete Approximation Problems. Journal of the ACM, 1976, 23(3), 555-565. https://doi.org/10.1145/321958.321975

24. Salomon, R. The Influence of Different Coding Schemes on the Computational Complexity of Genetic Algorithms in Function Optimization. Proceedings of 4th International Conference on Parallel Problem Solving from Nature, (PPSN IV), Berlin, Germany, September 22-26, 1996, 227-235. https://doi.org/10.1007/3-540-61723-X_987

25. Savelsbergh, M. A Branch-and-Price Algorithm for the Generalized Assignment Problem. Operations Research, 1997, 45(6), 831-841. https://doi.org/10.1287/opre.45.6.831

26. Sethanan, K., Pitakaso, R. Improved Differential Evolution Algorithms for Solving Generalized Assignment Problem. Expert Systems with Applications, 2016, 45, 450-459. https://doi.org/10.1016/j.eswa.2015.10.009

27. Tasgetiren, M. F., Suganthan, P. N., Chua, T. J., Al-Hajri, A. Differential Evolution Algorithms for the Generalized Assignment Problem. Proceedings of IEEE Congress on Evolutionary Computation, (IEEE CEC 2009), Trondheim, Norway, May 18-21, 2009, 2606-2613. https://doi.org/10.1109/CEC.2009.4983269

28. Wilson, J. M. A Genetic Algorithm for the Generalised Assignment Problem. The Journal of the Operational Research Society, 1997, 48(8), 804-809. https://doi.org/10.1057/palgrave.jors.2600431

29. Woodcock, A. J., Wilson, J. M. A Hybrid Tabu Search/Branch & Bound Approach to Solving the Generalized Assignment Problem. European Journal of Operational Research, 2010, 207(2), 566-578. https://doi.org/10.1016/j.ejor.2010.05.007

30. Yagiura, M., Ibaraki, T., Glover, F. A Path Relinking Approach for the Generalized Assignment Problem. Proceedings of International Symposium on Scheduling, (ISS2002), Hamamatsu, Japan, June 4-6, 2002, 105-108.

31. Yagiura, M., Ibaraki, T., Glover, F. An Ejection Chain Approach for the Generalized Assignment Problem. Informs Journal on Computing, 2004, 16(2), 133-151. https://doi.org/10.1287/ijoc.1030.0036

32. Yagiura, M., Iwasaki, S., Ibaraki, T., Glover, F. A Very Large-Scale Neighborhood Search Algorithm for the Multi-Resource Generalized Assignment Problem. Discrete Optimization, 2004, 1(1), 87-98. https://doi.org/10.1016/j.disopt.2004.03.005

33. Zolfaghari, S., Liang, M. A New Genetic Algorithm for the Machine/Part Grouping Problem Involving Processing Times and Lot Sizes. Computers & Industrial Engineering, 2003, 45(4), 713-731. https://doi.org/10.1016/j.cie.2003.09.003