2012 International Conference on Innovation and Information Management (ICIIM 2012)
IPCSIT vol. 36 (2012) © (2012) IACSIT Press, Singapore

An Artificial Immune Algorithm with Alternative Mutation Methods: Applied to the Student Project Assignment Problem

Mahmoud M. El-Sherbiny[1,a+] and Yasser M. Ibrahim[2,b]

[1]OR Dept., ISSR, Cairo Univ., Egypt
Currently, Dept. of Quantitative Analysis, College of BA, King Saud University, Saudi Arabia
[2]Social Science Computing Dept, Faculty of Economics and Political Science, Cairo Univ., Egypt Currently, Dept. of MIS, College of BA, King Saud University, Saudi Arabia

Abstract. This paper contrasts the performance of six different mutation methods of an *Artificial Immune Algorithm* for solving the *Student Project Assignment Problem* (*AIASPAP*). Factorial experiments are designed to investigate the influence of the essential factors of the algorithm, including the mutation methods, on the performance. The algorithm, with the best set of factor levels, is applied to four benchmark *Student Project Assignment Problems* (*SPAP*) of different sizes. The results are then compared with the results of a *Genetic Algorithm* (*GA*) proposed by Paul R, et al [1]. The results show that the non-uniform mutation that is based on the time as a non-uniform factor is the best among the suggested mutation methods. The results also show that the AIASPAP is highly competitive to, and even outperforms, the typical GA. In general, the artificial immune algorithm is shown to be a valuable alternative in solving the SPAP, in particular, and similar problems structure, in general.

Keywords: Artificial Immune Algorithm; Student Project Assignment; Genetic Algorithm.

1. Introduction

The *Student Project Assignment Problem* (*SPAP*) is considered to assign a set of students to a set of research projects based on the students' preferences. The students indicate their preferences in a student–project matrix. In the matrix, '1' indicates the student first choice, '2' the second choice, and so on up to a predefined maximum number of preferences allowed for a student. Allocating the projects according to the desire of all students becomes difficult, NP-hard in particular, when the number of students and projects becomes large with conflicts on the project choices.

Integer programming approach [2,8], Lagrangean/surrogate relaxation method in particular, has been used to solve the generalized assignment problem [5]. Genetic Algorithms (GAs) have been also applied to solve different forms of the assignment problem [1,6,7]. GAs have been so useful when no feasible solution to the problem is available. Another advantage of the GAs is that a number of different project allocations can be produced for any student–project matrix, which may facilitate the discussion on the merits of the various allocations.

Recently, Artificial Immune Algorithms (AIAs), inspired by the biological immune system, have been suggested as a search procedure for solving optimization problems [13]. For example, the aiNet system [4], and its application Opt-aiNet [3], are developed for function optimization. The objective function to optimize

+ Corresponding author
E-mail address: [a] *msherbiny@ksu.edu.sa*, [b] *ysabri@ksu.edu.sa*

is considered as an antigen invaded the system, while the candidate solutions as the antibodies. In an AIA, the antibodies gradually evolve, through mutation, in order to enhance their matching degree (fitness) with the antigen, hopefully to find a perfect match that solves the problem.

This paper aims at presenting six different mutation methods of an *Artificial Immune Algorithm* for solving the *Student Project Assignment Problem* (*AIASPAP*). Four benchmark SPAPs of different sizes are selected for testing the algorithm. Factorial experiments are firstly designed to investigate the influence of the essential factors of the algorithm, including the mutation methods, on the results. The performance of the mutation methods is contrasted and some conclusions about their applicability are drawn. The performance of the algorithm is then compared with the performance of a typical GA proposed by Paul R, et al. [1].

The rest of the paper is organized as follows. In Section 2, the student project assignment problem is described. In Section 3, the proposed *Artificial Immune Algorithm* for solving the *Student Project Assignment Problem* (*AIASPAP*) is illustrated. The factorial experiments are then presented in Section 4. Section 5 discusses the experimental results. Finally, conclusion is reported in Section 6.

2. The Student Project Assignment Problem

Let $S = \{1, 2,\ldots, s\}$ be a set of students, and $P = \{1, 2,\ldots, p\}$ be a set of projects, where ($p \geq s$). For $i \in S$, $j \in P$, we define c_{ij} as the preference given by student i to being assigned project j. A student-project preference matrix ($s \times p$) is formed to contain the students' preferences in the following sense. The value of '1' indicates the first choice of the student, the value of '2' indicates the second choice, and so on up to a specific number indicating the maximum number of projects a specific student might select. Additionally, some priority weights, w_i, could be assigned to each student so as to give some students a better chance of being allocated their higher preference projects. The mathematical programming formulation of SPAP can be illustrated as follows:

$$Min \quad \sum_{i=1}^{S}\sum_{j=1}^{P} w_i c_{ij} x_{ij}. \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{P} x_{ij} = 1 \quad \forall\, i \in S. \tag{2}$$

$$\sum_{i=1}^{S} x_{ij} \leq 1 \quad \forall\, j \in P. \tag{3}$$

$$x_{ij} = \begin{cases} 1 & if\ student\ i\ is\ assigned\ project\ j \\ 0 & Otherwise \end{cases}.$$

The first constraint (Eq.2) ensures that each student is assigned only one project. The second constraint (Eq.2) specifies that each project is assigned at most once as the number of projects might exceed the number of students.

2.1 The Proposed AIASPAP

In this paper, a typical immune algorithm structure is utilized. The algorithm preserves the essential principles of natural immune systems including the cloning, mutation, and clonal selection. The pseudocode, in

Fig.1 describes the main steps of the algorithm proposed. It includes the essential methods of the algorithm, highlighted in bold.

```
1. g = 1                                  //First Generation
2. Pg.CreateNewAntibodies(PopSize)        //Create new PopSize antibodies and evaluate them
3. For each antibody Ai in Pg             //Repeat for all antibodies in the population
   4. Repeat till Step 8 for NoC times    //NoC is a parameter representing the number of clones
      5. M = Ai.Clone( )                  //Clone Ai
      6. M.Mutate( )                      //Mutate the clone
      7. M.Evaluate ( )                   //Evaluate the mutated clone
```

8. TP.Add(M) //Add the mutated clone to a temporary population TP
9. BM = TP.GetBest( ) //Get the mutated clone with the best fitness in TP
10. If BM.fitness better than Ai.fitness then Ai=B //If best mutated clone better than its parent: BM replaces *Ai*
11. TP.Clear() //Delete all antibodies from TP
12. Next Antibody //Return to step 3 to get another antibody to mutate and clone
13. Pg.**Affinity**() //Calculate the affinity between each 2 antibodies in Pg
14. Pg+1.Add(Pg.**Select**()) //Based on the affinity, select antibodies from Pg for Pg+1
15. Pg+1.**CreateNewAntibodies**(PopSize – Pg+1.Size) //Add new antibodies to Pg+1 to replace eliminated ones
16. g = g + 1 //Next generation
17. If g > IterationsNumber Then Stop Else Go to Step 3 //Stopping criteria based on the number of iterations

Fig.1: Pseudocode of an Artificial Immune Algorithm (Essential Methods Highlighted in Bold)

It has to be stated that the implementation of the main methods of an AIA is often different for each problem handled. That is, the representation and hence the creation of the solutions, the mutation, and the affinity calculation should be tailored and implemented to fit each specific case at hand. For the SPAP, the problem of interest in this research, we develop some particular methods to fit the case. The following subsections describe these methods.

2.2 Population Initialization

The AIASPAP starts with a creation of an initial population containing a set of feasible solutions (antibodies). The solutions are created randomly and then evaluated. Each solution follows the structure presented in Fig.2 The solution is represented as an s-dimensional vector of integers. That is, each solution contains s genes, each of which contains an integer number between 1 and p. The integer number identifies the project $j \in \{1,2,\ldots,p\}$ that has been assigned to the student denoted by the vector index $i \in \{1,2,\ldots,s\}$. This representation ensures that each student is assigned only one project, i.e., the first constraint (Eq. 2) is satisfied and hence the computational effort is kept minimal regarding this constraint.

Student *i*	1	2	3	…	*s*
Project *j*	5	3	4	…	1

Fig.2: An Antibody Representation of SPAP

In order to guarantee the validity of the second constraint (Eq.3), i.e., each project has to be assigned at most once, the repetition of any project number either in the initiation of the new solutions or during the mutation process has to be avoided. Regarding the solution initiation, different algorithms have been already developed for creating a list of randomly-ordered unique numbers in a specific range. The mutation methods used in this research are also designed to avoid the repetition of the projects (see below). Therefore, the algorithm also guarantees the validity of the second constraint.

Evaluating the Solutions. Each solution is evaluated to determine its fitness. According to the immune systems, the evaluation is a measurement of the affinity between the solution (the antibody) and the problem (the antigen invaded the system). For the SPAP, the value of the objective function (Eq.1 above) is used as a measurement of the affinity of the solutions with only a single modification. Recalling that the SPAP is a minimization problem, the students' preferences cij are squared to amplify the evaluation, hence reduces the fitness, as the projects allocated get lower preferences (i.e. higher cij value). In case a student i has not included project j in her list of preferences, the corresponding cij in the preference matrix is assigned a suitably large penalty value. This is actually to reduce the chance of project j being assigned to student i by the algorithm through more amplification to the affinity between the solution and the problem.

The Normalized Fitness (NF) for each solution is also calculated as in Eq.3. The NF is bringing the evaluations of the solutions in each generation to a common scale, [0,1] in particular.

$$\mathrm{NF} = \frac{\mathrm{LowestFitness} - \mathrm{Fitness}}{\mathrm{LowestFitness} - \mathrm{HighestFitness}}. \quad (4)$$

The LowestFitness and HighestFitness are the lowest and the highest fitness of all the solutions in the current generation, respectively. The Fitness is the value of the evaluation function of the solution for which the normalized fitness is calculated.

2.3 Cloning and Mutation

Each antibody is cloned (copied) a number of times, determined by the Number of Clones (NoC) Parameter. The clones are then mutated to get a new set of antibodies that are different from their parent. In addition to the typical uniform swap mutation (see below), five different mutation methods are developed and tested. They all based on the typical swap mutation, but with a non-uniform number of swaps. That is, the Number of Swaps (NS) applied in the five mutation methods changes based on some particular factors including the progression of the search and the fitness of the antibodies.

In a typical swap mutation, two variables of the solution to mutate are randomly selected and then swapped. In a classical uniform swap mutation, the NS applied is always equal to a parameter representing the Maximum Number of Swaps (MNS). For example, if the MNS is five, then all the clones throughout the search are mutated through five consecutive random swaps. The uniform swap mutation is tested against the other five mutations used in this research. As shown in the results of the experiments below, the uniform swap technique might be useful only at the first stages of the search process for exploring the search space. However, it is not really effective for the fine tuning required at the last stages of the search. Accordingly, we test five different non-uniform mutations.

In the case of the non-uniform mutations, the NS is represented as a function of two parameters. In addition to the non-uniform factor parameter, a degree of non-uniformity parameter, u, is included. All the functions are designed to be directly related with u.

The first method is the Fitness mutation. In this mutation, the NS is based on the affinity (fitness) of the solution as a non-uniform factor. As the project assignment is a minimization problem, the NS function is designed to be directly related with the Normalized Fitness (NF) of the solution. That is, solutions with normalized fitness closer to one, i.e. relatively bad solutions, will be subject to more swaps. This actually gives the chance for low affinity solutions to mutate more in order to increase their chance of improving their affinities. The NS function of this mutation is calculated as in Eq.4. The normalized fitness function is illustrated in Eq.3 above.

$$\mathrm{NS} = \mathrm{MNS}^{(1-(1-\mathrm{NF})^{u})}. \quad (5)$$

The second method is the Time mutation, in which the NS function is designed to be inversely related with the ratio (T) of the current iteration number and the total number of iterations. That is, the more the search goes, the less the NS is. This is really intuitive as in contrast to the first stages of the search where a real exploration of the search space through significant changes in the solutions are required, at the last stages of the search, fine tuning with little changes of the supposed-to-be near-optimal solutions is more reasonable. The NS function of the Time mutation is represented in Eq.5.

$$\mathrm{NS} = \mathrm{MNS}^{(1-\mathrm{T}^{u})}. \quad (6)$$

The third method is the Time and Fitness mutation, in which the NS is based on both the ratio T and the NF of the solution. It basically uses the average of these two factors to decide the NS. The function is designed to be directly related with the fitness but inversely related with the time. The average of Time and Fitness (TF) is calculated as in Eq.6. The NS for this mutation is represented in Eq.7.

$$TF = \frac{NF+(1-T)}{2}. \quad (7)$$

$$NS = MNS^{(1-(1-TF)^u)}. \quad (8)$$

In the fourth and fifth mutations, we include a random factor (Rand). The NS in these two mutations is based on the non-uniform factors fitness and time, respectively, but with some randomization included. The random factor takes values between zero and one. The functions are designed to behave the same way as the Fitness mutation and Time mutation when Rand is small (exactly equal to 1/MNS). The closer the Rand to one is, the more the functions to behave uniformly. These two mutations are suggested to allow clones to be mutated differently, that is, some clones based on their non-uniform factor, others more, and others less. This is suggested to allow the search to escape from local optima by frequently increasing the NS or fine tune near optimal solutions when the non-uniform factor is misleading. The NS functions of these two mutations are represented in Eq.8 and Eq.9, respectively.

$$NS = MNS \times Rand^{(1-NF)^u}. \quad (9)$$

$$NS = MNS \times Rand^{(T^u)}. \quad (10)$$

2.4 Selection and the Affinity Function

In the AIAs, the selection of the antibodies to survive from one generation to the next one depends on some measurement of the affinity (similarity) among all the antibodies of the current generation. The calculations of the affinity between each two antibodies are applied to prevent similar solutions, especially with high evaluation, from being copied to the next generation and hence leading the search process. This is technically applied to reduce the chance of a premature convergence to local optima.

The typical affinity techniques that depend on the difference between the fitness of each two solutions cannot really be useful in the case of the SPAP. It is not really difficult to prove that in the SPAP far different solutions might have close or even the same fitness. Accordingly, a new affinity technique is developed to measure the similarity between every two antibodies in a population. The suggested technique basically counts the number of similar variables in the two solutions. The affinity function of two antibodies Aj and Ak is represented as in Eq.10.

$$AF(A_j, A_k) = \sum_{i=1}^{s} y_i \quad (11)$$

$$\text{where } y_i = \begin{cases} 1 & \text{If variable i in } A_j = \text{variable i in } A_k \\ 0 & \text{Otherwise} \end{cases}$$

This affinity function is tailored to the SPAP. The basic idea is that the more the number of similar variables in the two solutions is, the higher the similarity between them is. Based on a specific parameter, Remove Threshold (RT), the algorithm eliminates those solutions that have more similar variables than this parameter.

After applying the elimination of the similar solutions from a population, new solutions are randomly created to replace the eliminated ones and restore the population back to its original size. Some immune algorithms use a diversity parameter to add a number of new solutions based on the current population size. However, we have decided to fix the number of solutions during the search for comparison reasons as shown in the results section below.

3. Factorial Experiments

The effectiveness of an artificial immune algorithm greatly depends on the correct choice of the factors of the algorithm including the parameters and operators. This section investigates the effect of the different

factors of the AIASPAP on the results. In addition to the Mutation Methods (MM), three basic factors of the algorithm are considered: the Population Size (PopSize), the Number of Clones (NoC), and the Remove Threshold (RT). The number of alternatives to consider in the experiments for these factors are: six mutation methods (described above), three different population sizes, three different number of clones, and three remove thresholds. These factors, along with their different levels, are summarized in Table 1.

Table 1: Different Factors of the Algorithm and Their Different Levels

Factor	Symbol	Levels
Mutation methods	A	Uniform mutation, Fitness mutation, Time mutation, Time and Fitness mutation, Fitness with a randomized factor mutation, Time with a randomized factor mutation.
Population size	B	30, 50, and 80.
Number of Clones	C	10, 20, and 30.
Remove Threshold	D	At least 98% of the variables are similar to remove. At least 50% of the variables are similar to remove. At least 10% of the variables are similar to remove.

To evaluate the impact of each factor on the algorithm performance, four problems with different sizes are selected. The sizes of the problems are (s×p): 100×100, 200×200, 300×300, and 400×400. These problems are considered benchmarks for the SPAP. In addition, these problems are addressed by Paul R. et al. [1] and solved using a GA, hence, contrasting our results with some other searching technique, namely the GA, is applicable (see Section 5 below).

3.1 Factor Design

A full factorial design, which tests all the possible combinations of the factors, is the widely-used method especially when the number of combinations (instances) of the factors is relatively small [9,10]. In our case, the total number of instances of the four factors considered is 6×3×3×3=162 instances (see Table 1 above). Considering that each experiment, instance in this context, is run for 10 times and we have four problems to consider, the total number of runs required for a full factorial design is 162×10×4 = 6480 runs. Accordingly, a cost-efficient approach, popularized by Taguchi in the mid-1980s, is applied [12,15]. The Taguchi method uses an orthogonal array to reduce the total number of runs required. The appropriate orthogonal array for our experiments is found to be L18 (including 18 instances) instead of 162. The orthogonal array is illustrated in Table 2. The factorial experiements are designed and analysed using MINITAB 14.1 statistical software.

In order to determine the relative importance of each factor, two criteria are used in the literature: (1) the Signal-to-Noise ratio (S/N) [11], and (2) the Relative Percentage Deviation (RPD) [12,15]. In the case of S/N ratio, the term ‘signal’ denotes the desirable value (response variable) and ‘noise’ denotes the undesirable value (standard deviation). Therefore, the aim is to maximize the S/N ratio. In the case of a minimization objective, the S/N ratio is calculated as in Eq.11 [14].

$$S/N \text{ ratio} = -10\log_{10}(objectivefunction)^2. \tag{12}$$

The second criterion, RPD, is used when the scale of the values of the objective functions are different from one problem to another and hence cannot be used directly in the analysis. the RPD is then calculated for each instance. The RPD is calculated as in Eq.12.

$$\text{RPD} = \frac{\text{Alg}_{\text{sol}} - \text{Min}_{\text{sol}}}{\text{Min}_{\text{sol}}} \times 100. \tag{13}$$

where Algsol is the solution for a specific run, and Minsol is the best solution obtained out of a set of runs of a particular instance. After converting the objective values to RPDs, The average of the RPD of the runs of each instance is calculated (recall that each instance is run for 10 times).

Table 2: The Orthogonal Array L18 of the AIASPAP Four Factors

Instance Number	A	B	C	D	Instance Number	A	B	C	D	Instance Number	A	B	C	D
1	1	1	1	1	7	3	1	2	1	13	5	1	2	3
2	1	2	2	2	8	3	2	3	2	14	5	2	3	1
3	1	3	3	3	9	3	3	1	3	15	5	3	1	2
4	2	1	1	2	10	4	1	3	3	16	6	1	3	2
5	2	2	2	3	11	4	2	1	1	17	6	2	1	3
6	2	3	3	1	12	4	3	2	2	18	6	3	2	1

3.2 Analysis of the Factorial Experiments

According to the orthogonal array of the factorial experiments (Table 2 above), 18 instances are designed. Fig.3 and Fig. 4 plot the results of all the experiments using the RPD and the S/N ratio criteria, respectively. Obviously, the results based on the two criteria comply with each other, though inversely related.

Both figures show that the best mutation function is the Time mutation (level 3), while, as expected, the worst performance is for the uniform mutation (level 1). The Fitness mutation (2) and the Time and Fitness mutation (4) have similar performance and come second to the Time mutation. The Time with a randomized factor (6) comes after with a little better performance than the Fitness with a randomized factor mutation (5).

These results state the following. First, the randomized factor added to the Fitness and Time mutations has not enhanced the performance as anticipated. The mutations with a randomized factor behave as if we merge the uniform mutation with the non-uniform mutation. The idea of randomized factors might actually show much success with some other types of problems with a large number of local optima or with more complex search spaces; more investigation is required (see Conclusion and Further Research below). Second, regarding the Fitness mutation, it has not been as successful as the Time mutation. This is because it depends on the normalized fitness that is relative to the solutions at each generation. That is, a solution with a high NF relative to the others might actually need much more intensive mutation than the Fitness mutation allows. Similarly, in the Time and Fitness mutation, the NF misleads the algorithm again. Finally, the Time mutation outperforms the others as it avoids all the drawbacks described and depends on a consistent non-uniform factor, that is, the time.

Regarding the population size, the experiments show an insignificant impact of the different sizes attempted. This has actually been a typical result in some other studies of meta-heuristic search techniques including the GAs, except for relatively very small population sizes.

In contrast, the results show that the more the number of clones is, the better the performance is. Indeed, more clones mean more different mutations for the same solution and hence a much better chance for improvement.

In addition, the results show that a tougher remove threshold, with respect to the degree of resemblance among the solutions, is better. That is, the RT that restricts the elimination to the very similar solutions, such as, at least 98% resemblance, leads to a better performance. This is actually because a loose RT would definitely lead to the elimination of far different solutions, which in turn reduces the population size dramatically. When that severe elimination is recovered through initiating new randomly created solutions, the search would inevitably become not so effective.

Based on the discussion above, the optimal setting for the AIASPAP is as follows: the best mutation is the Time mutation (level 3), the best NoC is 30 (3), and the best RT is at least 98% resemblance to remove (3). Regarding the PopSize, no significant difference between the three levels has been shown; the PopSize is then selected and reported for each case separately.

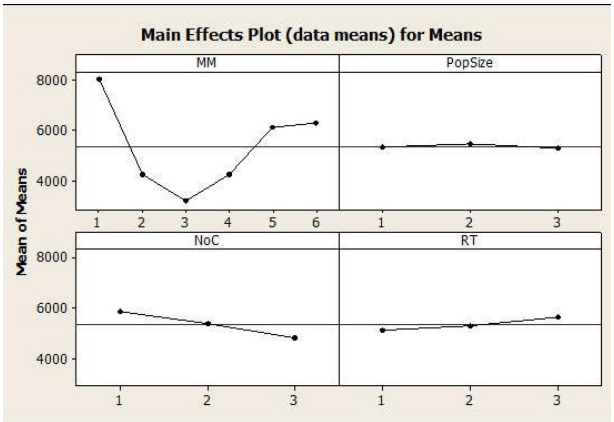

Fig.3: Mean Effects Plot (Fitted Mean) for RPD of Factor Levels of the AIASPAP

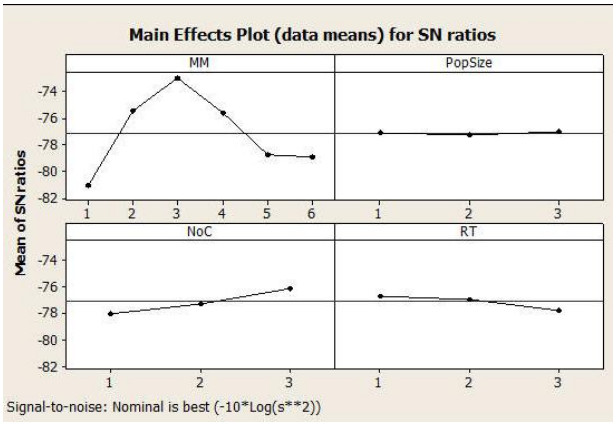

Fig. 4: Mean Effects Plot (Fitted Mean) for S/N Ratio of Factor Levels of the AIASPAP

4. AIASPAP vs. GA

Based on the results of the factorial experiments, the four benchmark problems are solved. The results of the AIASPAP are compared with the results of a typical GA, developed by Paul R, et al. [1], for solving the SPAP. Table 3 summarizes the results.

Four criteria are selected for the comparison: the average result of the 10 runs, the number of projects that are assigned to more than one student, the number of projects that are assigned to students not including them in their list of preferences, and finally the running time. In the experiments, the number of iterations for the AIASPAP is 3000 generations and the PopSize is 30. With respect to the number of evaluations occurred in a run, this setting is found to be equivalent to 60,000 iterations required by the GA to reach to its best solution.

As illustrated in Table 3, the results show that the AIASPAP outperform the GA with respect to three criteria. First, the quality of the solutions obtained by the AIASPAP is much better than those obtained by the typical GA in the four problems. The average results of the 10 runs of each problem clearly show that and are reported in column 1, for the GA, and in column 6, for the AIASPAP, in the table.

Second, with respect to the number of repeated projects, the AIASPAP also outperforms the GA. The AIASPAP is actually designed to avoid repeating the projects (see Section 3.1 above). Column 3 and column 8 in the table contrast the results of both algorithms.

Third, regarding the number of the projects assigned to students not including them in their preference list, column 4 and column 9, the AIASPAP shows much better performance with less number of the non selected projects.

With respect to the running time, the simplicity of the GA allows it to outperform the AIASPAP. However, the essential aspect of the assignment problem is to satisfy as many students as you can with the least number of conflicts. And this is exactly what the AIASPAP achieves.

Table 3: The Results of the GA vs. the Results of the AIASPAP

Problem Size (s×p)	**GA**					**AIASPAP**				
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
	Average Result	**Running time**	**# RP**	**# NSP**	**% NSP**	**Average Result**	**Running time**	**# RP**	**# NSP**	**% NSP**
100×100	4195	00:33.6	0	17	17%	1627	01:15.5	0	4	4%
200×200	10643	01:08.8	2	50	25%	2425	02:21.2	0	7	3.5%
300×300	17932	02:04.2	7	77.8	26%	7000	03:31.2	0	33	11%
400×400	27053	03:05.8	12	124.6	31%	8693	04:34.1	0	36.4	9.1%

RP: number of Repeated Projects (assigned to more than one student), #NSP: number of Not Selected Projects, and %NSP: the percentage of Not Selected Projects to the total number of projects.

5. Summary

This paper has proposed and tested six different mutation methods developed for an Artificial Immune Algorithm, applied to solve the Student Project Assignment Problem (AIASPAP). In order to study the performance of the mutation methods, factorial experiments are designed. The results show that the time is the most consistent factor to consider for the non-uniform mutations. The results also show a significant impact of the remove threshold and the number of clones but an insignificant impact of the population size on the algorithm performance.

The overall performance of the AIASPAP, in comparison with a typical GA, is proved to be very competitive. The AIASPAP proposed outperforms the typical GA with respect to the problem solutions, the number of projects that are assigned to more than one student, the number of projects that are assigned to students not including them in their list of preferences. However, the simplicity of the GA let it outperform the AIASPAP with respect to the running time.

Our further research focuses on investigating the performance of the mutation methods with some other types of problems. For example, we currently investigate the applicability of the non-uniform mutations with randomized factors to some other problems and attempt to improve their performance through controlling the randomization factor.

6. Acknowledgment

This paper is supported by the Research Center at the College of Business Administration and the Deanship of Scientific Research at King Saud University, Riyadh, Saudi Arabia.

7. References

[1] P. R. Harper, V. de Senna, I.T. Vieira, and A.K. Shahani: A Genetic Algorithm for the Project Assignment Problem. Computers & Operations Research, 32 (2005) 1255-1265.

[2] D.G. Catrysse and L.N. Van Wassenhove: A Survey of Algorithms for the Generalized Assignment Problem. European Journal of Operational Research, 60 (1992) 260–72.

[3] L.N. De Castro and J. Timmis: An Artificial Immune Network for Multimodal Function Optimization. In Proc. of IEEE Congress on Evolutionary Computation, 1 (2002) 699-674.

[4] J. Timmis, T. Knight, L.N. De Castro, and E. Hart: An Overview of Artificial Immune Systems. In: R. Paton, et al., eds. Computation in Cells and Tissues: Perspectives and Tools Thought. Natural Computation Series, Springer-Verlag, (2004) 51-86.

[5] M.G. Narciso and L.A.N. Lorena: Lagrangean/Surrogate Relaxation for Generalized Assignment Problems. European Journal of Operational Research, 114 (1999) 165–77.

[6] P.C. Chu and J.E. Beasley: A Genetic Algorithm for the Generalized Assignment Problem. Computers and Operations Research 24 (1997)17-23.

[7] J.M. Wilson: A Genetic Algorithm for the Generalized Assignment Problem. Journal of the Operational Research Society 48 (1997) 804-809.

[8] G.T. Ross and R.M. Soland: A Branch and Bound Algorithm for the Generalized Assignment Problem. Mathematical Programming 8 (1975) 91–103.

[9] R. M. Al-Aomarm and A. Al-Okaily: A GA-Based Parameter Design for Single Machine Turning Process with High-Volume Production. Computers and Industrial Engineering, 50 (2006) 317–337.

[10] F. Jabbarizadeh, M. Zandieh, and D. Talebi: Hybrid Flexible Flow Shops With Sequence-Dependent Setup Times and Machine Availability Constraints. Computers & Industrial Engineering, 57 (2009) 949-957.

[11] M.S. Phadke: Quality Engineering Using Robust Design. Prentice-Hall, NJ (1989).

[12] G. Taguchi: Introduction to Quality Engineering. White Plains, Asian Productivity Organization/UNIPUB (1986).

[13] J.T. Tsai, W.H. Ho, T.K. Liu, and J.H. Chou: Improved Immune Algorithm for Global Numerical Optimization and Job Shop Scheduling Problems. Applied Mathematics and Computation, 194 (2007) 406–424.

[14] B. Naderi, M. Zandieh, A.K.G. Balagh, and V. Roshanaei: An Improved Simulated Annealing for Hybrid Flow Shops with Sequence-Dependent Setup and Transportation Times to Minimize Total Completion Time and Total Tardiness. Expert Systems with Applications, 36 (2009) 9625–9633.

[15] G.S. Peace: Taguchi Methods: A Hands-On Approach. Addison-Wesley Publishing Company (1993).