An Evolutionary Immune Approach for University Course Timetabling

Y. Awad[†], A. Dawood[†] and A. Badr[††]

[†]Dept. of Business Information Systems, Arab Academy for Science & Technology & Maritime Transport, Cairo, Egypt
[††]Faculty of Computers and Information, Cairo University, Cairo, Egypt.

Abstract
The university course timetabling problem (UCTP) is a combinatorial NP-complete problem that has been subject to research since the early 1960's. Numerous solution techniques have been applied to the timetabling problem ever since. This paper aims at formulating an immune-inspired algorithm, namely the Clonal Selection Algorithm1 (CSA1) and testing its ability in solving the UCTP against the Genetic algorithm (GA). An Immune-Genetic algorithm (IGA) was also created, which combines the crossover operator borrowed from the genetic algorithm with immune-inspired concepts. Also, experimenting with the effects of changing the selection and re-selection schemes of the algorithms motivated the creation of a second version of CSA1, that is CSA2 and three more versions of IGA: IGA1, IGA2 and IGA3. All the devised algorithms were contrasted in their performances against the GA. Enhancements were applied to the mutation operator of the formulated algorithms by introducing a 'move factor'. As a means of improving the results attained by the algorithms, local search consisting of three variable neighborhoods was incorporated into each of them. The algorithms were tested over two problem instances, with varying complexities and the results demonstrate the effectiveness of the algorithms in solving the UCTP.

Keywords:
Clonal Selection Algorithms (CSA); Genetic algorithm (GA); Hybrid algorithm; University course timetabling problem (UCTP)

1. Introduction

The University Course Timetabling Problem (UCTP) is a problem that is constantly being tackled by educational institutes. Many consider the UCTP difficult to solve, partly because of the possibly large number of constraints associated with it. The fact remains that the UCTP is a problem of scarcity of resources. In other words, it is the problem of assigning events to timeslots, when teachers, rooms and other resources are scarce. The size of the timetabling problem Aitself; for instance, the number of events to be scheduled, the number of students and the number of courses per student is also a contributing factor to the complexity of the problem.

In spite of the fact that the timetabling problem was classified as NP-hard in a number of papers, including [9] and [6]. Reference [2], proves that the timetabling problem is NP-complete in five independent ways. A problem belonging to the NP-complete class has no method of being solved in polynomial time. Many solution techniques were applied to the timetabling problem in the past such as direct heuristics, network flow analysis, graph coloring, integer linear programming and others, as reported in [10]. As the timetabling problem became more complex and with the emergence of more restrictions to scheduling courses, obtaining a feasible or near feasible solution using only heuristics has become more difficult. Instead, more recently, heuristics are being used in combination with optimization techniques to better prune the search space. This method of solving timetabling problems is known as metaheuristics. A comprehensive survey of recent automated approaches used for university timetabling can be found at [4].

The work executed here aims at making a comparison between the performances of the algorithms under consideration when they are presented with post enrollment timetabling problems of different complexities. It must be stated that the genetic algorithm in addition to the local search and matching algorithm that are being used in this study are part of the work of the Metaheuristics Network (MN) [12], a European Commission project whose objective is to compare the effectiveness of metaheuristics in solving different combinatorial optimization problems; amongst which is the timetabling problem in hand. The genetic code can be found at [11].

Since the amount of literature available on timetabling using immune-based concepts is modest, the main contribution presented in this paper is embodied in the formulation of a clonal selection algorithm to solve the UCTP. And in order to evaluate its performance, it is compared to the genetic algorithm already formulated as part of the work of the MN. Also, a hybrid Immune-genetic algorithm was also formulated that integrates the crossover operator of the GA into the devised clonal selection algorithm, which was also compared to the GA. Lastly, several versions of the hybrid and immune-based algorithms were implemented with the only difference between them being in the selection and re-selection schemes to test the effect of changing the schemes on the performance of the algorithms.

Manuscript received February 5, 2011
Manuscript revised February 20, 2011

2. The Problem

The University Course timetabling problem attempted in this study is the same as the one being used by the MN. It is classified as a post-enrollment timetabling problem as it takes into consideration the timetabling problem after students are enrolled [5]. The problem consists of assigning timeslots to a number of events that students are enrolled in, provided that a feasible timetable is produced. A feasible timetable is one that does not violate any of the hard constraints. On the other hand, a "good" timetable is one that satisfies all hard constraints as well as a number of the soft constraints (or all if possible). Soft constraints are those that are set by the user to produce a timetable that is more suited to their preferences. In other words, violation of only soft constraints means that a valid solution was produced, but only with less quality, depending on the frequency of soft constraint violations. The aim of each algorithm in this paper is to produce a feasible solution to the timetabling problem in hand, while minimizing the number of soft constraint violations (SCVs).

2.1 Description of the University Course Timetabling Problem

The university course timetabling problem constitutes of events, students, and rooms. Students attend events that take place in rooms of certain features and capacities; required by the events. The timetable to be created has 45 timeslots split into 5 days, of 9 timeslots each. Assigning timeslots to events has to take place in compliance with the following hard constraints: 1) A student cannot attend more than one event simultaneously, 2) The room size must tolerate the number of attending students and it must satisfy all the features required by the event, 3) Only one event can take place in each room at any time. Furthermore, the following soft constraints define the quality of the timetable produced: 1) A student should not have a class in the last timeslot of the day, 2) A student should not have more than two consecutive classes, 3) A student should have more than a single class per day.

2.2 Problem Instances

The problem instances are created using a generator written by B.Paechter [7]. This generator is fed eight command line parameters, shown in table 1 that is acquired from [9], in addition to a random seed. The problem instances can belong to one of three complexity classes: small, medium or large. Fixing the value of the random seed as well as that of the parameters pertaining to a certain class will produce the exact same problem instance. However, changing the random seed while keeping the values for the parameters of a class constant would produce a different problem instance, but belonging to the same class. It is worth mentioning that every instance produced by the generator has a "perfect" solution. In other words, a solution that is without constraint violations. The scope of this paper is limited to testing only two problem instances: small1 and medium1 that are located at [11]. The problem instance files and their representation are further elaborated at [15].

table1. Parameter values for instance classes

		Instance Classes	
		Small	***Medium***
Parameters	No. of events	100	400
	No. of rooms	5	10
	No. of features	5	5
	Approximate features/ room	3	3
	Percentage of feature usage	70	80
	No. of students	80	200
	Max no. of events per student	20	20
	Max no. of students per event	20	50

2.3 Solution Representation and Room Assignment

The output format of a solution consists of two integer values for each event with the events ordered in the same manner as in the problem instance file. The first integer value is that of the timeslot for the event, and the second is the value of the room. The timeslot must be a number between or including 0 and 44, since there are 45 timeslots available for scheduling. While the room value should range between 0 and a value, depending on the number of rooms available in the problem instance being tackled.

It must be emphasized that the assignment of rooms to events, is implemented separately using a matching algorithm. After all events are assigned to timeslots, for each timeslot, a list of "possible rooms" that can be assigned to the events pertaining to that timeslot is produced. This list is executed with respect to the features and size of the events taking place in that timeslot. The matching algorithm then calls a network flow algorithm that establishes a maximum matching between the list of possible rooms and each event in the timeslot. If there still remains an event that is not assigned a room, the room that is currently occupied by the least number of events, as well as having the appropriate size and features is assigned to that event.

3. Local search and neighborhood structure

The local search (LS) addressed here consists of three neighborhoods N1, N2 and N3 that involve three different kinds of moves belonging to each neighborhood respectively. The first type of move, pertaining to N1, involves altering the timeslot for one event. The second type of move, belonging to N2, involves the swapping of timeslots between two different events. Finally the third type of move that is of neighborhood N3, permutes the

timeslots of three distinct events by either changing their timeslot as in the first type of move or swapping the timeslots between the three events as in the second type of move.

Search through each of the neighborhoods is initially assigned a probability. Neighborhoods N1 and N2 are assigned a probability of 1.0, meaning that both are explored. Since the search through N3 is very time consuming and ineffective [8], its probability is set to zero, meaning that only neighborhoods N1 and N2 are exploited. The number of moves allowed for the LS depends on the class of the problem being handled. The maximum number of moves is set to 200 for small problem instances and to 1000 for medium problem instances.

The local search used is a "first-improvement" local search. In other words, it examines the existing neighborhoods one by one until it reaches a solution that is better than the current solution which then becomes the new current solution. The local search algorithm attempts to handle the hard constraints of the solution in hand, then the soft constraints. It starts by generating a list of events that is randomly ordered, then moving through the list one event after the other. Each event within the list is first tested for hard constraint violations. If the event under consideration is involved in a hard constraint violation, it is subjected to a series of moves until all effort is expended to eliminate the violation. The moves that are tried on the event as an attempt to remove the constraint violation are initially from the neighborhood N1, then, neighborhood N2 is explored. After each move is implemented, the matching algorithm, (responsible for assigning rooms to each scheduled event) is applied to the timeslots affected by the move, and the resulting solution is delta-evaluated. If the move results in an improvement in the feasibility of the solution, then it is executed. Otherwise, another move is tried. Once the local search algorithm has processed all the events in the originally generated randomly ordered list of events as explained earlier, it then moves on to examine each event for soft constraint violations in the same way. More details about the local search algorithm and how it is applied can be found at [8].

4. The Algorithms

This paper conducts a comparison of the competency of seven different algorithms in solving the timetabling problem. The algorithms include the Genetic algorithm (GA), the Clonal Selection Algorithm1 (CSA1), the Clonal Selection Algorithm2 (CSA2), and a number of hybrid algorithms: the Immune-genetic algorithm (IGA), the Immune-genetic algorithm1 (IGA1), the Immune-genetic algorithm2 (IGA2) and the Immune-genetic algorithm3 (IGA3).

The GA, CSA1 and IGA can be considered the three main algorithms in comparison, as the remaining algorithms only differ slightly in the method of selection and re-selection. All algorithms use the same solution representation and neighborhood structure for the local search. They also are started off with a population of random solutions that is created by randomly assigning timeslots to events, then using the matching algorithm to assign the rooms. Local search is then applied to each of the initial random solutions to further improve them. The population size is fixed to ten individuals for all the algorithms under investigation. Each algorithm is given a time limit depending on the class of the problem being handled, during which it can perform for a number of generations. For the small1 instance, the time limit is set to 90 seconds, while for the medium1 instance; it is set to 900 seconds. The solution presented at the end of the time limit would be the best solution reached throughout all the generations that the algorithm has undergone. Additional parameters and their values that are set for each algorithm are shown in table 3.

Different terminology is used depending on whether the algorithm being referred to is the GA or an immune-based algorithm. For instance, fitness in the GA corresponds to affinity in the immune-based algorithms; an individual is referred to as an antibody, mutation is corresponds to hypermutation, and the population of individuals corresponds to the antibody repertoire respectively.

Since the algorithms require a technique in order to introduce constraints into their fitness/affinity functions, a static penalty function [15] is integrated into the fitness/affinity function that penalizes each occurrence of a constraint violation depending on its type (hard or soft). A single penalty is issued for each occurrence of a SCV in a solution, while each occurrence of a HCV is assigned a penalty that is amplified by multiplying it by a large constant. The fitness/affinity function used for the evaluation of candidate solutions is illustrated in (1):

$$f(s) = \frac{1}{1 + \sum_{i=1}^{n} w_i c_i(s)} \quad (1)$$

where,

s is the candidate solution; n is the number of types of constraints available in our case it is equal to two, since the two types of constraints available are soft and hard constraints; wi which represents the weight associated with constraint type i. The weight of a HCV is set to 1000,000, and a SCV is set to 1; ci (s) represents the number of occurrences of constraint violations of constraint type i for solution s.

4.1 The Genetic Algorithm

The genetic algorithm can be categorized as a global search heuristic that is based on the Darwinian theory of evolution. It is used to find solutions to optimization and search problems. While the GA is considered relatively strong in performing global search, its drawback lies in local search [13]. For this reason, the GA used here is supported by local search. The GA is presented with a population of randomly generated candidate solutions that are optimized by passing through the iterative process of selection, crossover and mutation.

4.1.1 Applying the Genetic Algorithm:

The algorithm starts by selecting two parents for reproduction from the population of individuals using tournament selection with a tournament size of five. Next, crossover takes place between the two parents with a crossover probability equal to 0.8. If crossover does occur, it is applied by randomly assigning timeslots to events from either the first or second parent, creating only a single child. Then, the rooms are handled using the matching algorithm. The next step is to apply mutation to the child solution. The mutation probability is set to 0.5. Mutation is represented in a random move of type 1 or 2, identical to the move types of the local search algorithm explained in section IV. Local Search is applied once again, but this time to the child solution. The replacement policy used here states that the child solution replaces the worst member of the original population regardless of the quality of the child solution. The elements of the population are sorted, and the best solution is selected.

The Genetic Algorithm

For i=1 to PopSize do
{
Generate a random initial solution
Apply Local Search to solution
}
End for
Sort solutions by fitness

While time limit not reached do
{
1. Select 2 parents from the population by tournament selection
2. Apply crossover to parents, with crossover probability μ
3. Apply mutation to child, with mutation probability α
4. Apply Local search to child solution
5. Child solution replaces worst member of the population
6. Sort population by fitness

A_{BEST} ← Best solution in population
}
End While

4.2 The Clonal Selection Algorithm

Clonal Selection Algorithms (CSAs) is a field of study that addresses problems by applying the principles of the immunological theory. Reference [1], defines a Clonal selection algorithm as an algorithm that is primarily focused on mimicking the Clonal selection principle (CSP) mechanisms.

The mechanisms of the CSP are responsible for selecting the antibodies necessary to combat foreign antigens. Simply speaking, once a match occurs between an antibody of the immune system and a foreign antigen, the antibody is capable of destroying that antigen. The CSP is realized when an antibody recognizes a foreign antigen, and it is activated, and then it proliferates creating clones of itself - an action known as 'Clonal Expansion'. Whilst proliferating, coping errors can occur in the antibody- a process known as 'Somatic Hypermutation' which can actually lead to an improvement (better match, known as 'affinity') in the recognition between the antibody and antigen. The resulting—higher affinity clones— can be inserted into the memory repertoire, so as to combat the same antigen if it re-attacks. A real life optimization problem can be solved by finding a suitable representation for the problem and an algorithm so as to resemble the optimization mechanisms of the CSP. This can be achieved through the use of Clonal Selection Algorithms. In this paper, this is realized by the formation of the CSA1 and CSA2 that attempt to solve the timetabling problem, and that resemble the Clonal Selection Algorithm namely the CLONALG [3].

4.2.1 Implementing the Clonal Selection Algorithm1:

For the CSA1, the affinity/fitness function along with the problem resemble the antigens, and the solutions resemble antibodies. The CSA1 is started off with antibodies that are randomly generated and stored in the antibody repertoire P. The first step of the CSA1 is the selection of a number of the best antibodies (ns) that are to undergo the process of cloning. The repertoire of selected antibodies is symbolized by Ps. Then, the repertoire of cloned antibodies C, is formed by cloning each of the selected antibodies in Ps a number of times equal to PopSize multiplied by β, which is a variable, as in (2). The cloning of the antibodies here is not affinity proportionate. This means that all antibodies are cloned the same number of times and not depending on their affinity.

$$\text{No. of clones of each } ns = \beta .PopSize \qquad (2)$$

Next, Hypermutation is applied to each clone in *C*, including the parent clones (i.e. the antibodies from which the clones were formed). The Hypermutation implemented here indicates that the amount of mutation applied to a clone must be inversely proportional to its affinity. In

other words the greater the affinity of the clone the less the mutation that is to be applied to it. To achieve this, first, the mutation rate is calculated, which is inversely proportional to the affinity of each clone, as shown in (3).

Mutation rate (α) = exp [(-1*pd) (1/penalty)]. (3)

,where *pd* corresponds to the decay factor; and (1/*penalty*) corresponds to the affinity of a clone. Then, the number of random moves is computed, according to (4).

No_of_Random_Moves = *mf* * α (4)

,where mf corresponds to the Move Factor. A random move represents a single move of either type one or two (see section IV). The higher the mutation rate, (the lower the affinity) the higher the number of random moves implemented on the clone, as shown in (3) and (4). In other words, the move factor is a variable that when multiplied by the mutation rate controls the number of random moves applied to an antibody; the higher its value, the more the mutation that will be applied. Local search is then applied to each clone, and they are sorted according to their affinity. The reselection process executed here relies on choosing the best clone(s) (nr) from C (after Hypermutation and LS), and applying local search once again to this re-selection. The nr clone(s) after local search replace the worst member(s) of the original repertoire of antibodies P. In order for metadynamics to be realized, after ng generations, a number of clones from P are replaced with new random antibodies (d), in order to enhance the exploration capabilities of the algorithm. For these newly introduced antibodies to be fit opponents to the rest of the antibodies in P, they are subjected to LS, with a magnified number of moves that is equivalent to the maximum moves allowed multiplied by the current generation number. Finally, P is sorted, and the antibody with the highest affinity is selected as the best antibody PBEST. The CSA1 is a simple clonal selection

The Immune-genetic Algorithm (IGA)

For i =1 to PopSize do
{
P Generate a random antibody
Apply local search to antibody
}
End for
Sort antibody repertoire by affinity
While time limit not reached do
{
1. Ps Selection: Select ns antibodies with the highest affinity from the antibody repertoire
2. C Cloning: Clone each of the ns selected antibodies, with number of clones = (β.PopSize)
3. Hypermutation: Mutate each clone inversely proportional to its affinity:

(a) Calculate Mutation rate α= exp [(-1*pd) (1/penalty)]
(b) Calculate number of Random Moves

4. Apply Local Search to each clone in C
5. Sort the clones according to affinity
6. Re-selection: Select the best 2 clones from C as parents
7. Apply crossover to parents, with crossover probability μ
8. Apply Local Search to child
9. Child replaces the worst antibody of the antibody repertoire P
10. Sort antibody repertoire by affinity
11. [Case: After ng generations]Metadynamics: Replace worst d antibodies in the antibody repertoire P with d random antibodies
12. Sort antibody repertoire P by affinity

P_{BEST} Best antibody in repertoire
}
End While

The Clonal Selection Algorithm1 (CSA1)

For i=1 to PopSize do
{
P Generate a random antibody
Apply local search to antibody
}
End for
Sort antibody repertoire by affinity

While time limit not reached do
{
1. P_s← Selection: Select *ns* antibodies with the highest affinity from the antibody repertoire
2. *C*← Cloning: Clone each of the *ns* selected antibodies, with number of clones = (β.*PopSize*)
3. Hyper mutation: Mutate each clone in *C* inversely proportional to its affinity

(a) Calculate Mutation rate α= exp [(-1*pd) (1/penalty)]
(b) Calculate number of Random Moves

4. Apply Local Search to each clone in *C*
5. Sort the clones according to affinity
6. Reselection: Select the best *nr* clones from *C*
7. Replace the *nr* clones with the worst *nr* antibodies of the antibody repertoire *P*
8. Sort antibody repertoire by affinity
9. *[Case: After ng generations]* Metadynamics: Replace worst *d* antibodies in the antibody repertoire *P* with *d* random antibodies
10. Sort antibody repertoire *P* by affinity

P_{BEST} Best antibody in repertoire
}
End While

addition to devising the mf that controls that amount of mutation applied to each of the antibodies.

4.2.2 The Clonal Selection Algorithm 2:
The CSA2 operates in the same way as CSA1 except for step six. In the reselection process, instead of selecting only the best clone(s) (nr) from C for replacement, the CSA2 additionally selects a random clone(s) (nrr) from C to replace the worst members of P. In other words, both the best and random antibodies are chosen for replacement. This added feature gives the algorithm the opportunity of exploiting current solutions (by reselecting the best), as well as the ability to further explore (by reselecting randomly).

4.3 The hybrid Immune-Genetic Algorithm

The Immune-genetic algorithm (with its variations) can be considered as a combination of both the Clonal Selection Algorithm and the Genetic algorithm. It operates in the same manner as the CSA1, but, in addition to introducing crossover between clones produced by the cloning process. Several versions of the Immune-genetic algorithm were implemented that vary in both the method of selection and re-selection. The objective of experimenting with a number of versions of the Immune-genetic algorithm is to discover the effect of changing the former mentioned factors on the performance of the algorithm.

4.3.1 The IGA:
The Immune-genetic algorithm (IGA) starts, and functions in the same way as CSA1 for a series of steps; more specifically until step number five. At step six, the best two clones from C are reselected to become the parents of crossover. Then the crossover operator is put into action, in the same way as in the genetic algorithm, and with the crossover probability set to 0.8 as well. The child produced by the crossover undergoes local search, then, it replaces the worst member of the original antibody repertoire P. Finally, metadynamics is applied in a similar way as CSA1.

4.3.2 The IGA1:
The Immune-genetic algorithm1 (IGA1) differs from IGA in the method of selection. Instead of selecting the absolute best ns antibodies in step one, tournament selection is used.

4.3.3 The IGA2:
The Immune-genetic algorithm2 (IGA2) differs from the IGA in two ways. Firstly, in the use of tournament selection in step number one instead of selection of absolute best ns antibodies. Secondly, in the re-selection process, in step six, the IGA2 can either re-select the absolute best two clones from C to undergo crossover, or, select two random clones from C to become the parents for crossover. The decision of which re-selection method the IGA2 uses is based on a probability of 0.5. In other words, at every generation, there is an equal chance for random re-selection as there is for the re-selection of the absolute best clones for crossover.

4.3.4 The IGA3:
The difference between the Immune-genetic algorithm3 (IGA3), and IGA is only in the re-selection method. It utilizes the same re-selection method as IGA2. But, the IGA3 uses the absolute best method for selection (not tournament selection). Table 2 summarizes the differences in operation between the versions of the IGA.

table 2. Differences in selection and re-selection between versions of IGA

Algorithm	Method of selection	Method of re-selection
IGA	Absolute best	Absolute best
IGA1	Tournament selection	Absolute best
IGA2	Tournament selection	Absolute best or random
IGA3	Absolute best	Absolute best or random

table 3. Algorithm parameters and values

Symbol	Description	Parameter presence in Algorithms						
		1	*2*	*3*	*4*	*5*	*6*	*7*
ns	No. of antibodies/sol. selected for cloning; =3 for sml. & med.	×	✓	✓	✓	✓	✓	✓
p	Problem type, determines the MaxSteps for LS. For sml p=1 MaxSteps= 200; for med. p=2 MaxSteps =1000.	✓	✓	✓	✓	✓	✓	✓
nrr	No. of rnd child antibodies to replace worst members in the original pop.; =1 for sml &med.	×	×	✓	×	×	×	×
b	A variable representing β used in the calculation of no. of clones for each sol.; =0.2 sml & med.	×	✓	✓	✓	✓	✓	✓
pd	Decay factor for calculation of α; =30 for sml & med.	×	✓	✓	✓	✓	✓	✓
nr	No. of best child antibodies to replace worst members of pop.; =1 sml & med.	×	✓	✓	×	×	×	×
mf	Move factor sets the no. of mutation moves; =5 for sml & med.	×	✓	✓	✓	✓	✓	✓
d	No. of rnd solutions introduced into pop. ; =2 for sml & med.	×	✓	✓	✓	✓	✓	✓
ng	No. of generations after which rnd antibodies are introduced; =20 for sml. & med.	×	✓	✓	✓	✓	✓	✓

The numbers in the table subheading stand for the algorithms accordingly: 1:GA, 2:CSA1, 3:CSA2, 4:IGA, 5:IGA1, 6:IGA2, 7:IGA3

5. Implementation

All algorithms were written in C++, and compiled using the GNU compiler gcc version 4.4.1, on a Core 2 Duo 2.20 GHz processor. A number of different parameters were created for each of the algorithms; these are described in table 3. Table 3 shows the parameter symbol, a description of it, its value for the small and medium instances and whether or not it is present in each of the algorithms.

6. Results

The performance of the algorithms in this paper is measured by conducting a series of tests: For each class of the problem handled (small and medium), firstly, a comparison of the fitness/affinity of the algorithms against generations is performed. Secondly, the algorithms are contrasted in terms of the number of soft constraint violations (SCVs). Thirdly, a comparison of the best solution attained by each algorithm and the time at which this solution was attained is also presented. All runs for all algorithms for both the small1 and medium1 problem instances produced feasible solutions. Therefore, an assessment of the number of invalid solutions produced was unnecessary.

For the comparison of fitness/affinity against generations, the average fitness/affinity for 10 runs for each algorithm were plotted against 6000 generations for the small1 instance and 3000 generations for medium1 instance. The result of the comparison between the hybrid algorithms (IGA, IGA1, IGA2 and IGA3) for the small1 instance is shown in Fig.1. Fig.2 shows the results when the best-performing hybrid algorithm (IGA1) was compared to the remaining algorithms. Fig.3 shows the comparison between the hybrid algorithms for the medium1 instance. The best performing hybrid algorithm (IGA) for the medium1 instance is compared to the remaining algorithms in Fig.4.

As for the evaluation of the number of soft constraint violations, 100 runs of the small1 problem instance were implemented for each algorithm, with a time limit of 90 seconds for each run. In addition to executing 50 runs for the medium1 instance with a time limit of 900 seconds for each run. The results of this comparison are shown in box plots of Fig.5 and Fig.6 respectively. In the box plots the median value for the number of SCV is represented by a dash; and each box represents the ranges between the upper and lower quartile. The whiskers emerging from the box signify the maximum and minimum values produced by the algorithm.

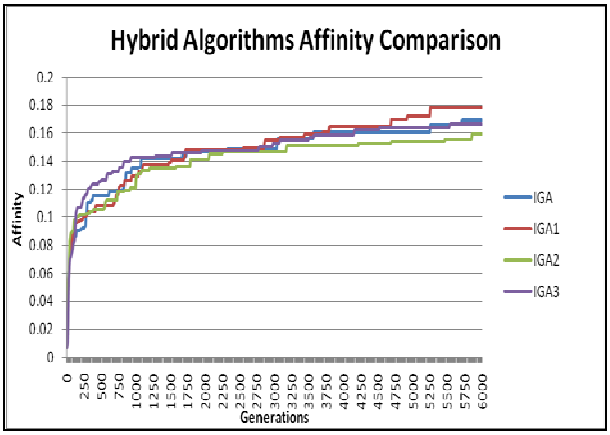

Figure1. A comparison of the affinity of hybrid algorithms for the small1 instance

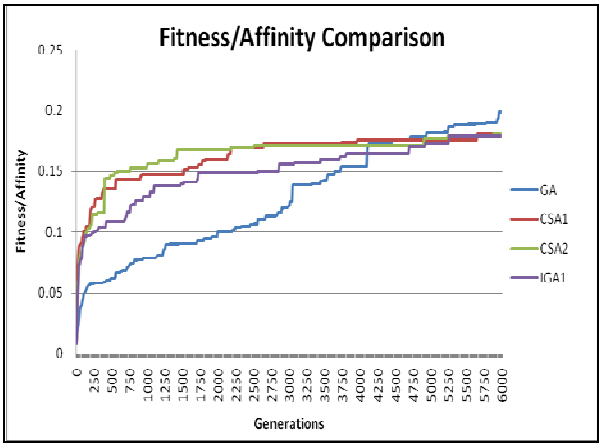

Figure2. A comparison of the fitness/affinity between GA, CSA1, CSA2 and IGA1 for the small1 instance.

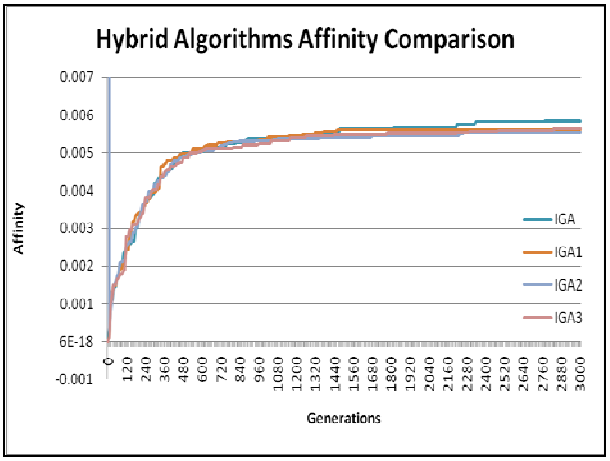

Figure3. A comparison of the affinity of hybrid algorithms for the medium1 instance.

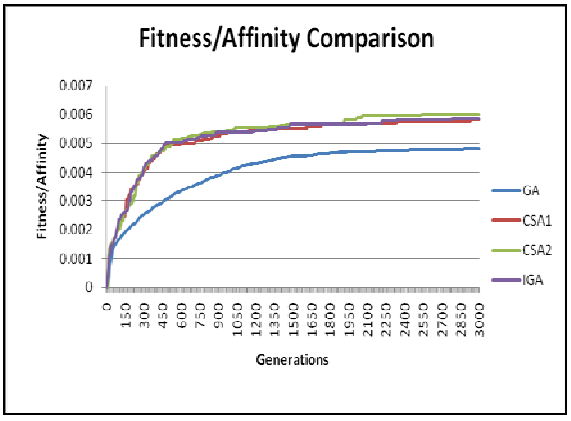

Figure4. A comparison of the fitness/affinity between GA, CSA1, CSA2 and IGA for the medium1 instance.

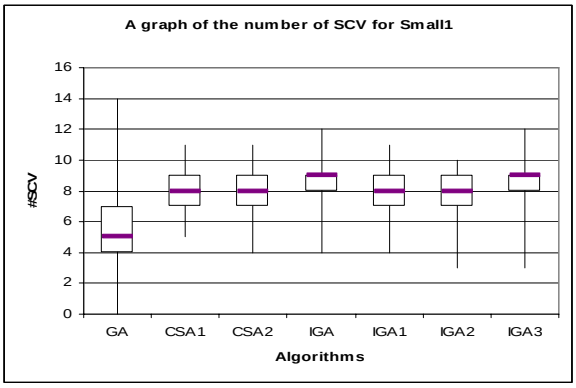

Figure5. A boxplot of the number of SCVs for all trials on the small1 instance

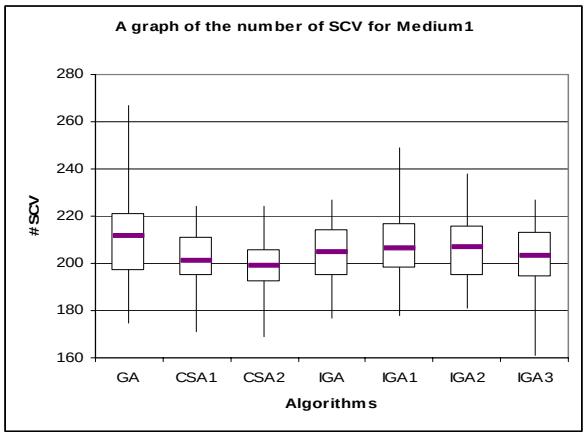

Figure6. A boxplot of the number of SCV for all trials on the medium1 instance.

Finally, table 4 depicts the best solution reached by each algorithm with regards to the least number of SCVs, and the time at which it was reached for 100 runs of the small1 problem instance, with a time limit of 90 seconds; and 50 runs of the medium1 problem instance, with a time limit of 900 seconds.

Table1. The minimum number of SCV and the time of their attainment by algorithms

Algorithm	Small1		Medium1	
	#SCV	*Time(sec)*	*#SCV*	*Time(sec)*
GA	0	22.937432	175	334.992936
CSA1	5	11.708732	171	861.853863
CSA2	4	69.624351	169	797.905867
IGA	4	20.849303	177	884.907303
IGA1	4	31.525970	178	788.213260
IGA2	3	84.401275	181	859.245699
IGA3	3	69.976373	161	871.746480

7. Discussion

With regard to the test of fitness/affinity of the algorithms against generations, for the small1 instance, the hybrid algorithms' performances were closely similar (see Fig.1). In spite of the fact that IGA3 surpassed the remaining hybrid algorithms towards the beginning, the IGA1 outperformed the remaining algorithms in the end. When IGA1 was compared to CSA1, CSA2 and the GA in Fig.2, it was clear that both of the Clonal Selection Algorithms (CSA1 and CSA2) yielded better affinity results than the hybrid IGA1 at the start. However the performance of both the CSA1 and CSA2 stabilized a little over quarter the way through and they were later joined by IGA1. As for the GA, throughout the 6000 generations, its affinity has continued to grow until it surpassed that of the remaining algorithms. On the other hand, towards the first generations, the GA's performance can be regarded as the worst (see Fig.2). The various hybrid algorithms were even more similar in affinity for the medium1 instance than for the small1 instance (see Fig.3). The IGA can be regarded as the best performing algorithm amongst them. When a more comprehensive comparison was conducted between the IGA, the CSA1, the CSA2 and the GA (see Fig.4); it was noted that the GA was the poorest performing algorithm in terms of fitness. All the other algorithms (CSA1, CSA2 and IGA) produced noticeably better fitnesses/affinities than the GA; especially CSA2. This observation is opposite to what was expected given the GA's good performance for the small1 instance.

With regards to the test of the number of soft constraint violations for the small1 instance; the box plot of Fig. 5 shows that the GA as having the lowest median, upper and

lower quartile value amongst all algorithms, its only drawback being its high value for the maximum number of SCV. The numbers of SCVs for the remaining algorithms were in close resemblance to each other. The IGA2 can be considered as the second best in terms of number of SCVs after the GA. As for the medium1 instance, the boxplot of Fig.6 shows the GA as one of the least effective algorithms in terms of the number of SCVs having the highest median, maximum and upper quartile values for number of SCVs. The CSA2 is regarded as the best performing algorithm having the lowest values for the number of SCVs amongst all algorithms.

Finally, concerning the test of the most fit solution produced and the time of their attainment; for the small1 instance, the GA managed to reach the best results accomplishing no SCVs at nearly quarter of the time allowed. On the other hand, the remaining algorithms produced a minimum number of SCVs ranging between 3 and 5, at a time that is between 12 and 84 seconds. As for the medium1 instance, the solution with the smallest number of SCVs was generated by the IGA3 in about 872 seconds.

8. Conclusion

It is rather challenging to select a single algorithm as the 'best' at solving the timetabling problem in hand. The only apparent observation is the poor performance of the GA for the medium1 instance and its good performance for the small1 instance. This emphasizes that an algorithm can behave differently across different classes of the problem. Also, the fact that the performance of the hybrid algorithms (IGA, IGA1, IGA2 and IGA3) compared to the Clonal Selection Algorithms (CSA1 and CSA2) was indefinable as each of the aforementioned algorithms performed differently across classes of instances in most tests. An additional observation is that for the small1 instance, all algorithms converged much earlier than the GA, producing much better results at the beginning.

With regards to the effect of changing the selection and re-selection methods on the performance of the algorithms; introducing some randomness in the re-selection process such as in CSA2 rather than CSA1 has resulted in the CSA2 performing better than the CSA1 for almost all tests. On the other hand the effect of incorporating randomness on the performance of the hybrid algorithms was indeterminable, as their performances were comparable to each other in most tests.

It must be noted that the algorithms implemented here may have produced entirely different results if certain factors were changed, such as the local search neighborhood structure. Also, the values assigned to the parameters and probabilities used by various algorithms were not ideal; they were merely assigned after conducting brief experiments about their effect on the results.

Based on the above conclusions, it is clear that the contributed algorithms (CSA1, CSA2, IGA, IGA1, IGA2 and IGA3) are capable of producing good solutions, sometimes even better than that of the GA. However, the effect of changing the selection and re-selection methods on the performance of the algorithms was difficult to determine.

Acknowledgement

We would like to thank all who supported this work technically and morally.

References

[1] J. Brownlee, "Clonal Selection Algorithms," CIS Lab., Centre for Information Technology Research, Swinburne Univ. of Tech., Melbourne, Technical Report 070209A, Feb 2007.

[2] T. B Cooper and J. H. Kingston, "The complexity of timetable construction problems," in 1995 Proc. of PATAT, LNCS, vol.1153, pp.283–295,Springer-Verlag,1996.

[3] L. N DeCastro, "Learning and Optimization Using the Clonal Selection Principle," IEEE Trans. Evol. Comput., Special Issue on Artificial Immune Systems, vol. 6, pp. 239-251, June 2002.

[4] R. Lewis, A survey of metaheuristic-based techniques for university timetabling problems, OR Spectrum, vol. 30(1) pp.167-190, 2008.

[5] Z. Lü, J. K. Hao, "Adaptive Tabu Search for Course Timetabling," Eur. Journal Oper.Res., vol.200(1), pp.235–244, Jan.2010.

[6] M. R. Malim, A. T. Khader and A. Mustafa, "An Immune-Based Approach to University Course Timetabling: Negative Selection Algorithm," in the Proc. of the 2nd IMT-GT Regional Conf. on Mathematics, Statistics and Applications, Univ. of Sains Malaysia, Penang, pp. 13-15, 2006.

[7] B.Paechter, School of Computing, Napier University, http://www.dcs.napier.ac.uk/~benp

[8] O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, K. Socha,and B.Paechter, "A local search for the timetabling problem," in 2002 Proc. of PATAT, LNCS, vol 2740, pp.124–127, Springer-Verlag, 2003.

[9] O. Rossi-Doria, M. Samples, M. Birattari, M. Chiarandini, J. Knowles, Manfrin, M. Mastrolilli, L. Paquete, B. Paechter, and T. Stützle, "A comparison of the performance of different metaheuristics on the timetabling problem," in 2002 Proc. of PATAT, LNCS, vol. 2740, pp. 329-351, Springer-Verlag, 2003.

[10] A. Schaerf, A survey of automated timetabling, Artificial Intelligence Review, 13(2), pp.87-127, 1999.

[11] Supplementary information on [9], Available: http://iridia.ulb.ac.be/supp/IridiaSupp2002-001/index.html

[12] The Metaheuristics Network, Available: http://www.metaheuristics.net

[13] L. Wang and L. Jiao, "The immune genetic algorithm and its convergence," in Proc. of 4th Int. Signal Processing, vol.2, pp. 1347 – 1350, Oct. 1998.
[14] O. Yeniay, Penalty function methods for constrained optimization with genetic algorithms, Mathematical and Computational Applications, vol. 10(1), pp.45–56, 2005.
[15] International Timetabling Competition (ITC), available at: http://www.idsia.ch/Files/ttcomp2002/IC_Problem/node7.html, accessed February 2010.