RT
257
C.1

Incremental and Classical Genetic Algorithm

By

MOHAMAD M. AWAD

Submitted in partial fulfillment of the requirements

for the Degree of Master of Science

Thesis Advisor: **Dr. NASHAT MANSOUR**

Computer Science Program

LEBANESE AMERICAN UNIVERSITY

July 2001

LEBANESE AMERICAN UNIVERSITY

GRADUATE STUDIES

We hereby approve the thesis of

Mohamad M. Awad

Candidate for the *Master of Science* Degree*

Dr. Nashat Mansour: [redacted]
Associate Professor of Computer Science (Chair)

Dr. Ramzi Haraty: [redacted]
Assistant Professor of Computer Science

Dr. Khaled El Fakih: [redacted]
Assistant Professor of Computer Science

Date: 6/8/2001

*We also certify that written approval has been obtained for any proprietary material contained therein.

Incremental and Classical Genetic Algorithm

Abstract

By

MOHAMAD AWAD

A Classical Genetic Algorithm (CGA) is known to find an optimal or near optimal solution for complex and difficult problems. However, there are many cases where these problems are subject to frequent modifications each producing a new problem, if these new problems are large, it is costly to use a genetic algorithm to reoptimize these problems after each modification. In this thesis, we propose an Incremental Genetic Algorithm (IGA) to reduce the time needed to reoptimize large-scale modified problems. To validate the proposed approach, we consider three problems: optimal regression testing, general optimization, and exam scheduling. In addition, we develop a hybrid genetic algorithm (HGA) for the problem in order to improve the results of a classical genetic algorithm. The experimental results obtained by applying IGA to the three optimization problems, show that IGA requires a smaller number of generations and less time than that of CGA to converge to a solution. In addition, the quality of the solutions produced by IGA is similar or slightly better than that of the CGA.

KEY WORDS: artificial intelligence, incremental genetic algorithms, application of genetic algorithm, optimization algorithm, exam scheduling, general optimization, and regression testing.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Professor Nashat Mansour, for the help, encouragement, and support he gave me. I extend my gratitude to Professor Ramzi Haraty and Dr. Khaled El Fakih for being on my thesis committee.

Special thanks go to the Academic Computer Center staff for their cooperation. I greatly thank Dr. Khaled El Fakih for his dedication and the help he gave me to achieve what I achieved. I would like to thank all those who assisted me to accomplish this work, in one way or another.

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Genetic algorithms are based on the mechanics of natural evolution (Goldberg, 1989). They mimic natural populations reproduction and selection operations to achieve efficient and robust optimization. Through their artificial evolution, successive generations search for beneficial adaptations in order to solve a problem. Each generation consists of a population of chromosomes, also called individuals, and each chromosome represents a possible solution to the problem. The initial generation consists of randomly created individuals. Each individual acquires a fitness level, which is usually based on a cost function given by the problem under consideration.

The Darwinian principle of reproduction and survival of the fittest and the genetic operations of recombination (crossover) and mutation are used to create a new offspring population from the current population. The reproduction operation involves selecting, in proportion to fitness, a chromosome from the current population of chromosomes, and allowing it to survive by copying it into the new population. Then, two mates are randomly selected from this population, and crossover and mutation are carried out to create two new offspring chromosomes. Crossover involves swapping two randomly located sub-chromosomes (within the same boundaries) of the two mating chromosomes. Mutation is applied to randomly selected genes, where the values associated with such a gene is randomly changed to another value within an allowed range. The offspring population replaces the parent population, and the process is repeated for many generations with the aim of maximizing the fitness of the individuals.

Genetic algorithms are different from traditional methods in many ways, genetic algorithm works with a coding of the parameter set and not with the parameter themselves. Genetic algorithm searches from a population of points not a single point.

Here the term Classical Genetic Algorithm (CGA) is used to represent all types of Genetic Algorithms including Hybrid Genetic Algorithms, which uses random generated chromosomes as their initial population to optimize any problem.

Classical Genetic algorithms have been adapted for solving a variety of engineering, science, and operational research problems. Some examples of such applications can be found in (Davis, 1991), (Ebeling et al., 1996), (Baeck, 1997), (Haupt, 1997), (Fogel, 1998), (Banzhaf, 1999), (Sait et al., 1999), and (Pham, 2000). An outline of this CGA is given in Figure 1.

```
Random generation of initial population, size POP;
Evaluate fitness of individuals;
Repeat

    Rank individuals and allocate reproduction trials;

    Randomize
    Hillclimb

    For(I=1 to POP step 2) do
        Randomly select 2 parents from the list of reproduction trials
        Apply crossover and mutation;
    Endfor
    Evaluate fitness of offsprings;
Until(convergence criterion is satisfied)
Solution = Fittest
```

Figure 1. Classical Genetic Algorithm

In the above examples and in general, CGA finds one good solution, but clearly problem solving is more than that, it is (1) defining the problem, (2) representing the possible candidate solutions, (3) anticipating how the problem may change over time, and (4) searching for solutions that are robust to those changes

(Michalewicz and Fogel, 2000). Moreover, it is the action of re-solving the problem as it changes based on the most recent available information. Problem solving is a never-ending process.

In CGA evaluation, function appears as a landscape, which is the result of mapping alternative solutions to their corresponding functional values. Each possible solution corresponds to a point on that landscape. When the conditions of the problem change there are two alternatives: (1) the landscape changes due to a change in the evaluation function, or (2) the constraints on the feasible region of possible solutions change.

If large-scale problems are known to be subject to frequent modifications each producing a new problem, we present a new approach that reduces the cost of re-solving these large-scale problems by CGA due to frequent changes; our approach is based on an Incremental Genetic Algorithm (IGA).

The idea of Incremental Genetic Algorithm is if a change occurres to a certain problem due to changes to constraints, parameters, or the evaluation function. We save several chromosomes (individuals) from the 1st CGA run. These chromosomes are: one third to two third (depending on the problem) consists of best feasible chromosomes and their neighbor feasible chromosomes, one quarter or less of infeasible chromosomes. When a change occurs we do not start the GA with completely random initial population. Instead we start with the saved chromosomes and the rest are randomly generated.

We validated the IGA idea by applying it to three problems: optimal regression testing, general optimization, and exam scheduling. The experimental results show that IGA selects better solution and require shorter execution time (less number of generations) to converge than CGA. In addition, in some problems the performance of IGA overwhelms that of CGA if the size of the problem is large.

This paper is organized as follows. Section 2 describes the Incremental Genetic Algorithm. Section 3 gives example applications. Section 4 proposed Hybrid Genetic Algorithm for Exam Scheduling. Section 5 gives empirical results of HGA and CGA for Exam Scheduling. Section 6 gives empirical results of IGA and CGA. Section 7 contains the conclusion.

CHAPTER 2

Incremental Genetic Algorithm

When we have large scale or complex problems, which are subject to modifications to their constraints or to their objective functions, then reoptimizing these problems by creating major part of the initial population from chromosomes which are saved by running Classical GA on the same problem first time before modifications is called Incremental GA. These saved chromosomes which are used in the initial population of the IGA, consist of two parts best feasible and best infeasible chromosomes. Best feasible chromosomes are chromosomes, which have minimum cost (objective function value), and they satisfy all constraints. Best infeasible chromosomes are chromosomes with minimum cost and with minimum constraints violation.

2.1 Saving chromosomes in CGA

Classical GA maintains a population of chromosomes at any point in time, rather than just a single chromosome. If these chromosomes are saved on applying certain criteria, we can later use these saved chromosomes to provide us with the potential for diversity of approaches to large-scale, and complex problems solving. Moreover, modifying large-scale or complex problems can make one of these saved best infeasible chromosomes feasible, and perhaps another feasible solution in the saved population will still be feasible can increase. The criteria we use to save chromosomes in every generation are the following:

1st – Select best feasible and infeasible chromosomes and save in a list.

2nd – Ensure diversity

3rd – Ensure that we have enough saved feasible chromosomes (at least 50% of the size of the initial population).

In the next two sections, we will discuss two methods, the first one is called Crowding, which we use to ensure diversity, and the second one is called LFDC, which is used to save more feasible chromosomes. At the end, we will explain the method that we used to decide the percentage of the saved feasible and infeasible in the initial population of IGA. Figure 2 gives a summary of IGA outline.

```
Phase I :
        During CGA run and in each generation
        Find best so far infeasible chromosome
        Apply Crowding
        If no similar chromosome can be found in the saved in feasible chromosomes list
        Save best so far infeasible chromosome
        Endif
        Find best so far feasible chromosome
        Apply Crowding
        If no similar chromosome can be found in the saved feasible chromosomes list
        Save best so far feasible chromosome
        Else
        /*case where problems represent their parameters as binary encoded string*/
        /*compute modulus 4 of the convergence counter*/
           If ( best so far feasible chromosome has not changed after (convergencecounter mod 4 = 0))
           Compute LFDC /*Local Fitness Distance Correlation*/
             If(LFDC < 0 and LFDC >= -1)
             Apply Crowding
             Save selected local feasible chromosomes
             EndIf
           EndIf
        /*case where problems represent parameters as non binary encoded string*/
        Find feasible chromosome which is close in value to the best so far feasible chromosome
        Apply Crowding
           If no similar chromosome can be found in the saved feasible chromosomes list
           Save selected feasible chromosome
           EndIf
        EndIf
        Sort

Phase II:
        Run IGA starting with initial population which is composed of
        > 50 % best feasible and selected individuals according to LFDC or other method
         <=25 % best infeasible
         <=25 % randomly generated
```
Figure 2. Incremental Genetic Algorithm

2.2 Crowding method

To avoid duplicates and to ensure diversity, a method that is called *Crowding* (De Jong 1975) determines the chromosomes to maintain by applying a measure between a new chromosome and the old ones; this measure is called Hamming distance. Hamming distance is the number of bits by which two chromosomes differs. For this reason, a new function is added which checks for duplicates by comparing the value of the objective function of the new chromosome (to be saved) and the previously saved chromosomes (if Hamming distance between two chromosomes is different than 0 it is accepted). This will help in the existence of diversity between individuals.

2.3 LFDC (Local Fitness Distance Correlation) method

During the first run of the CGA the population may converge to a single point (no change in the value of the objective function). In order to ensure that we have enough saved feasible chromosomes for use in the creation of the initial population of IGA, LFDC (Local Fitness Distance Correlation) method is used in optimal regression testing and general optimization problems, another method is used with exam scheduling problem, which we will explain at the end of this section. LFDC is introduced and advocated in (Kallel & Schoenauer 1996) selects part of the local (current generation) feasible chromosomes, which are strongly related to the best so far feasible chromosome. LFDC is used if after certain number of generations (here the number equal one quarter the convergence counter) there is no change in the value of the best so far feasible chromosome. To compute LFDC we carry out the following steps:

1st – Compute f_i, which is the fitness value of the local feasible chromosome i, and where the fitness is the inverse of the objective function value.

2nd – Compute d_i which is the Hamming distance from the local feasible chromosome i to the best so far chromosome.

3rd – Compute the average of both the fitness and Hamming distance of the local feasible chromosomes we call them av_f and av_d.

Next step is that a sample of the local feasible chromosomes are selected if they satisfy the following two conditions:

A- The fitness of the local feasible chromosome i should be greater than av_f.

B- The Hamming distance of the local feasible chromosome i should be less than av_d.

We use the selected n local feasible chromosomes to compute the covariance as in Equation 1, where $\bar{f}$ and $\bar{d}$ are the average fitness and the average Hamming distance of the selected n sample. Last step is to compute LFDC as in Equation 2,

$$\frac{1}{n}\sum_{i=0}^{n-1}(f_i-\overline{f})(d_i-\overline{d}) \qquad (1)$$

σ_f, σ_d are respectively the standard deviations of the fitness and distance for the n local feasible chromosomes. For minimization problems for instance, LFDC values

$$LFDC = \frac{Equation(1)}{\sigma_f \sigma_d} \qquad (2)$$

less than 0 and greater or equal to –1 in case of minimization indicate that we can save this sample of n local feasible chromosomes, while values greater or equal to 1 means ambiguous in case of minimization and in this case we reject this sample of n local feasible chromosomes. The underlying idea of this operation is that it guides the initial population in IGA towards the best structure.

However, this cannot be used with a complex problem such as exam scheduling, this is because we are not working with binary representation of the problem. For this reason in every generation, we save the feasible chromosome which has an objective function value close to the value of the best so far feasible value.

2.4 Creating initial population in IGA

After the convergence toward the global optimum in CGA, all saved best, feasible and best infeasible chromosomes are sorted. The reason for sorting these individuals by ascending order in case of minimization or descending order in case of maximization is to select the needed individuals from the best population in the list.

The last step is to create the initial population in IGA when the problem is modified. Initialization is recognized to be crucial issue in evolutionary algorithms (EAs) in general, and in Genetic Algorithm in particular, all EA practitioners have experienced that a bad initialization can, in the best case modify the online performance (i.e. increase the time-to solution) (Kallel and Schoenauer 1997).

So in order to avoid bad initialization and decrease the time to find an optimum solution in IGA we have to decide on the size of the initial population of IGA, and to decide on the number of best feasible and best infeasible chromosomes to use from the saved chromosomes, and the number of randomly generated chromosomes. To decide these numbers, an approach that uses a procedure that works on varying the parts, which form the initial population in the IGA, and by running IGA several times on different problems. These variations and changes are the following:

a- Modifying the values of the variables or constraints

b- Changing the size of the initial population

c- Changing the percentage of the saved feasible chromosomes in the initial population d- Changing the percentage of saved infeasible chromosomes in the initial population

e- Changing the percentage of the random generated chromosomes in the initial population

CHAPTER 3

Example Applications

In order to evaluate the performance of the IGA we run it on three different problems

The first problem is optimal regression testing, the second problem general optimization, and the third problem is exam scheduling. These problems vary in the degree of difficulty from simple to very complex the following subsections explain each problem.

3.1 Optimal regression testing

Regression testing is the re – execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. Regression testing is a significant component of maintenance. Hence reducing the cost of regression testing is very important for making software maintenance a less expensive activity. We use the Genetic Algorithm (GA) for regression testing developed by (Mansour and El-Fakih, 1999).

In order for the regression testing to reduce the high cost of repeating the whole set of test cases used in the initial development of the program, it is unreliable to choose a random subset of test cases. Therefore, it is important to select a subset of Test cases that have minimal cardinality, and which accomplish the goal of regression testing. This problem is referred to as optimal regression testing.

In regression testing we assume that the module under consideration is represented by a control flow graph with M program segments, where a program segment represents either a control statement or a contiguous sequence of non-control statements. In addition, it is assumed that the set of N test cases used in the initial development of the module has been saved, and that a table of the test cases and the program segments they cover can be determined. Further, the control flow graph enables us to derive information about the reachability of program segments from other segments.

Given that program segment k has been modified, the optimal retesting problem consists of finding values for $(X_1, X_2, \ldots, X_N)$ that minimize the cost function

$$Z = X_1 + X_2 + \ldots\ldots + X_N \tag{3}$$

subject to the constraints

$$\sum_{j=1}^{N} a_{ij} X_j \geq b_i \; ; \; i = 1, \ldots, M \tag{4}$$

Where $X_j = 1$ (or 0) indicates the inclusion (or exclusion) of test case j in the selected subset of retests. The matrix $[a_{ij}]$ is derived directly from the test-segment coverage table, i.e., $a_{ij} = 1$ if segment i covered by test case j; $b_i = 1$ (or 0) indicates whether segment i needs to (or need not) be covered by the subset of retests due to the modification of segment k, where the values b_i are derived from the segment reachability information.

3.2 General optimization

Many of these complex problems require huge amount of computational time and resources, sometime it is not possible to obtain a solution using an exact algorithm such as Branch and Bound in a reasonable time due to it is complexity.

These problems are characterized by the diversity of values, which can be assigned to the variables and the constraints. These values can be zero one or minus one, and it can be greater than one for the constraints.

In order to simulate large-scale problems with different values of parameters and constraints, CGA is modified to accept these problems. We created Random values, which consist of –1, 0 and 1 for parameters, and constraints b_i values are 0 and 1. Another large-scale problem is created consists of -1,0,1 and constraints b_i varies between 0 and 20.

These types of complex problems are created to demonstrate the efficiency of IGA compared with CGA. Random generated tables of data, which represent more than binary data, such that the objective function is similar to the following Equation:

$$\text{Min O.F.} = \sum_{i=1}^{N} (1)^{i-1} X_i \qquad (5)$$

Subject to constraint

$$\sum_{j=1}^{N} a_{ij} X_j \geq b_i \, ; \; i = 1, \ldots, M \qquad (6)$$

Where b_i varies between 0 and 20, or it could be only 0 and 1.

3.3 Exam Scheduling

The general scheduling problem is among the hardest combinatorial problems because it belongs to the class of NP-complete problems, which means that no deterministic algorithm is known yet for solving the problem in polynomial time. Scheduling of final exams for large numbers of courses and students in universities, such as the Lebanese American University (L.A.U.), is done manually by the Registrar's Office, a large number of complaints are made by students about conflicts or unfairness in the schedule. Conflicts occur when simultaneous exams are scheduled for the same student, and unfairness to a student refers to consecutive exams or more than two exams on the same day. A good exam schedule at L.A.U. would aim for minimizing conflicts and the two unfairness factors based on user-assigned weights to these three factors and subject to some constraints. We use the Genetic Algorithm developed for exam scheduling (Tarhini and Mansour, 1998). However, this GA for exam scheduling proved inefficient to be run for CGA and IGA comparison because of the high number of conflicts this GA produces. For this reason a new hybrid GA is developed which consists of the following specifications and structures taken from the above GA, and more features are added to reduce conflicts (see chapter 4 for more detail).

Given that A exams are to be taken by students over B days, where E exam sessions can be done per day, the exam scheduling problem consists of assigning A exams to Π $(=B*E)$ exam sessions, within specified classrooms. The objective is to minimize the conflict and the unfairness factors, which are:

i) The number of students with simultaneous exams, *ii*) The number of students with consecutive exams, and

iii) The number of students with two or more exams on the same day.

we assume the following conditions and constraints that apply at L.A.U.:

a-The user should be provided with the flexibility of assigning weights to the three conflicts and unfairness factors.

b- The number of exam periods, Π, is predefined.

c- A limited predefined number of classrooms, ψ, are available for exams.

d- Room capacity is taken into consideration in assigning exams to rooms. In addition, more than one exam/section can be assigned to the same room at the same time if they fit.

e- The total number of exams is not being greater than $\Pi * \psi$.

f- The user (The Registrar's Office) has the option of forcing (an) exam(s) to be fixed to a specified day/period/room before scheduling.

g- Scheduling of a user-defined group of exams to the same period should be allowed.

h- The last session of one day is considered consecutive to the first session of the next day.

Scheduling problems can be represented by graphs. Let $G(V, E)$ be a graph in which: vertex $v_i \in V$ represents an exam to be scheduled; vertex weight w_i represents the number of students taking exam v_i; edge $e \in E$ joining two vertices v_i and v_j represents the existence of students taking both exams v_i and v_j; weight of edge e, w_{ij}, represents the number of students taking both exams v_i and v_j. The vertex weight is used to match a room's capacity.

The exam scheduling problem can be expressed as a modified graph coloring problem, where we color the vertices of a graph using a specified maximum number of colors (exam periods), Π, such that the objective function O.F. (Equation 7 below) is minimized and the constraints (listed above) are met. A solution to the exam scheduling problem is henceforth denoted as the configuration C. Note that each color corresponds to an exam period and all vertices having the same color represent the exams that can be assigned to the same period.

Let $c(v)$ be the color of vertex v, and $\xi = \{ c_1, c_2, \ldots c_\pi \}$ be the set of ordered, available colors; that is, $|\xi| = \Pi =$ maximum number of available colors, and $(c_i - c_{i-1}) = 1$ for $i=2,\ldots, \Pi$. The objective function, O.F., is given in terms of the following factors:

(i) S_{SE}, the total number of students taking conflicting simultaneous exams $= \sum w_{ij}$ with $c(i) = c(j)$.

(ii) S_{CE}, the total number of students taking consecutive exams $= \sum w_{ij}$ with $|c(i) - c(j)| = 1$.

(iii) S_{ME}, the total number of students taking two or more exams per day $= \sum w_{ij}$ with

$c(i)$ and $c(j)$ referring to exam periods on the same day.

$$\text{O.F.} = \alpha * S_{SE} + \varphi * S_{CE} + \sigma * S_{ME} \tag{7}$$

Where α, φ, and σ are user-defined weights; the following table (Table 1) summarizes the symbols used in the objective function and the constraints.

Table 1. Symbols used in the objective function

Symbol	***Meaning***
Π	The maximum number of available exam periods (maximum number of colors).
ψ	The maximum number of available classrooms.
ξ	The set of available rooms (available colors); $\|\xi\| = \Pi$
c_j	An available color in ξ (i.e exam period).
$c(i)$	The period to which exam i is assigned (The color of vertex i).
C	The system configuration, i.e the exams schedule.
S_{SE}	The total number of students taking conflicting simultaneous exams.
S_{CE}	The total number of students taking conflicting consecutive exams.
S_{ME}	The total number of students taking two or more exams per day.
α	A weighting factor related to the importance of S_{SE} in O.F.
φ	A weighting factor related to the importance of S_{CE} in the O.F.
σ	A weighting factor related to the importance of S_{ME} in the O.F.
w_i	The number of students taking exam i.
w_{ij}	The number of students participating in both exams i and j.

CHAPTER 4

New Hybrid Genetic Algorithm for Exam Scheduling

Checking the previous work on Classical Genetic Algorithm for final exam scheduling for L.A.U., we found that is not good enough in optimizing exam scheduling for L.A.U. this is due to the high number of simultaneous and consecutive conflicts, which is not fair for students who are seating for exams. By investigating this Genetic Algorithms for Exam Scheduling, the fact came out clear, the high conflicts are caused by the reproduction process (crossover) that may alter a good solution to a bad solution (violate constraints & increase conflicts). Two parents are selected by a certain selection process (Roulette wheel selection) are crossed at random to produce new offspring.

In order to improve the CGA for final exam scheduling in L.A.U. several improvements are added. In summary, the following lists the differences between our HGA and other previously created CGA for final exam scheduling in L.A.U.:

A- In HGA we added room constraint satisfaction into the objective function.

B- HGA works on minimizing the cost, which is associated with the number of conflicts (specifically simultaneous and consecutive conflicts) using a local search mechanism called *Hill-Climbing*.

C- *A* new process is added in HGA called *feasibilize*, works on minimizing room constraint violation.

D- CGA uses one mutation to change one gene, while HGA uses two types of mutation operation, light and heavy.

E- A procedure *regrouping* is added in HGA, which works on placing same course with different sections together in the same period while reducing conflicts and avoiding constraints violation, no such process was found in any of the studied CGA for final exam scheduling in L.A.U.

Figure 3 shows the outline of the Hybrid Genetic Algorithm for Exam Scheduling.

```
Random generation of initial population, size POP;
Evaluate fitness of individuals;
Repeat
    Check room violation add penalty
    Save_best_sofar();
    Rank individuals and allocate reproduction trials;
    for i = 1 to POP step 2
    Randomly select two parents from list of reproduction trials;
    Apply crossover and mutation;
    Endfor
    Light mutation
    Heavy mutation
    Feasibilize half courses
    Regroup same courses in the same periods
    Hill-Climbing
    Evaluate fitness of offsprings;
until convergence;
```

Figure 3. Outline of the Hybrid Genetic Algorithm for Exam Scheduling

We start by explaining each of the above improvements in detail.

The first improvement is shown in the objective function where a penalty is added every time a violation to the room size is found. This penalty represents the sum of the number of rooms in every period which have their capacity violated (room constraint).

This final sum in each chromosome is multiplied by a weight $\gamma = 200$ as shown in Equation 8 and this penalty is added to the objective function as shown in Equation 9.

$$\text{Penalty} = \gamma * (\Sigma \rho_{k\ 1<=k<=\Pi}) \tag{8}$$

$$\text{O.F.} = \alpha * S_{SE} + \varphi * S_{CE} + \sigma * S_{ME} + \text{Penalty} \tag{9}$$

Where Π represents number of periods, and ρ_k is the sum of all room violations in period (k). The introduction of this new penalty aims to reduce room constraints violation.

Finding room violation is implemented by using a simple procedure, keeping records of the number of students taking exams in each period, and in each room. When any of the GA operations changes the room or the period or both of them, these records are updated

The second improvement is to hybridize Genetic Algorithm for final exam scheduling by adding a local search mechanism named Hill-Climbing that finds an optimal solution in shorter time than the other GA operators (less iterations needed). Hill-Climbing uses a technique of calculating the penalty incurred by scheduling a gene (course) in a particular period, given that the other genes are fixed. In more clear words, Hill-Climbing tries to re-assign a course (including all sections) in a way that will reduce the cost and does not cause any room constraint violation. This has the advantage that when attempting to reschedule each individual gene, the improvement can be found in a fraction of the time that it would take to perform a full evaluation.

The addition of Hill-Climbing to the normal genetic operators inevitably has some computational expenses, but this can be justified by the reduction in the search space that must be explored in order to find the optimum solution.

The third improvement is by adding procedure called *feasibilize.* This procedure works on half of the infeasible population by selecting chromosomes with highest cost (objective function), and it forces these chromosomes to satisfy room constraint. In addition, Hill-Climbing will work on the other half of the infeasible chromosomes to make them feasible while trying to reduce conflicts and room violation, by moving courses to different periods and rooms.

Moreover, two types of mutations are created light, and heavy mutations. Where light mutation operates on small portion of genes while heavy mutation works on altering large number of genes. In more details, the light mutation works on one gene by changing the period, while the heavy mutation works in a different way by finding the period that has the maximum assigned courses, and the period with the small assigned courses. Then, it tries to find a balance between these periods by moving those courses in the period with maximum courses to the period with small assigned courses, but on condition that the conflicts number is decreased and constraints are respected.

The reason for having two mutations is to create a balance between the number of genes (courses) assigned to each period, which will cause in addition to the help of the crossover operator to overcome the fast convergence to local optimal solution which is caused by the local search mechanism.

However, during the reproduction and the enforcing feasibility processes, same courses with different sections are misplaced in different periods. In order to place them in the same period to satisfy one of the conditions stated above, a new process called *regroup* rearranges these courses to be in the same period or to another period in a way to place them together and to reduce conflicts, if necessary, a new room is assigned without violating room constraints.

In general, the addition of these improvements make HGA runs in $O(Gen*Pop*Ex^2)$ time where Gen is the number of generations, Pop is the number of chromosomes and Ex is the number of exams.

In order to build confidence about the high quality of the output, and in addition to the procedure which keeps records of the room violation, another procedure *evaluate* justifies the number of conflicts after obtaining the final solution.

The genetic operators rates employed in HGA are 0.75 and 0.01 for crossover and mutation.

When the value of the objective function does not change for a specific number of generations the run should be stopped. For this reason a counter is set which increments by one each time we find no change or improvement in the value of the objective function and we say that it is converged when we have the counter equal to 20.

CHAPTER 5

Empirical Comparison of HGA and CGA for Exam Scheduling

In order to compare the results obtained by running HGA, several data are collected from previous experiments, which were carried to optimize L.A.U exam scheduling using classical genetic algorithm results (Mikati, 1999).

The data covers three semesters for spring 1994-1995, fall 1996-1997, fall 1998-1999. The following version is chosen see Table 2, which represents one of the different parameters values used in the previously developed CGA. From the run of both HGA and CGA, the following results are obtained which are summarized in Table 3. These results are shown in charts (Figures 4, 5, and 6) to make the task of comparison easier.

Table 2. Weights values used in the objective function

α	ϕ	σ
100	10	0.2

Comparing the results, we can see the size of improvement. Although, the number of two or more exams increased (multiple exams), we can see the improvement in the first two conflicts, which are very important in creating a fair exam schedule. This improvement again is obvious if we compare the difference between the cost of HGA and CGA see Figure 7, the cost here represents the objective function which consists of three type of conflicts, the total number of each conflict is multiplied by a weight factor. In addition, higher weights values are given to the first two types of conflicts, that is why we see that the cost in HGA is lower than that in CGA because number of simultaneous conflicts are eliminated in HGA, and consecutive conflicts are significantly lower than that in CGA.

Table 3. Experimental results of running CGA and HGA

Spring 94-95	Number of Periods = 32	S_{SE}	S_{CE}	S_{ME}	# rooms used	Violation of room capacity
	CGA	43	968	692	21	0
	HGA	0	194	1022	21	0
Fall 96-97	Number of Periods = 32	S_{SE}	S_{CE}	S_{ME}	# rooms used	Violation of room capacity
	CGA	9	987	726	21	0
	HGA	0	137	1083	21	0
Fall 98-99	Number of Periods = 32	S_{SE}	S_{CE}	S_{ME}	# rooms used	Violation of room capacity
	CGA	12	700	529	21	0
	HGA	0	91	559	21	0

However, the advantage of the HGA is hindered by the time requirement compared with any of the Genetic Algorithms for Exam Scheduling implemented for L.A.U. However, we run HGA using a PC 1GHz in CPU speed and 128 Mbytes of memory and it required less than an hour.

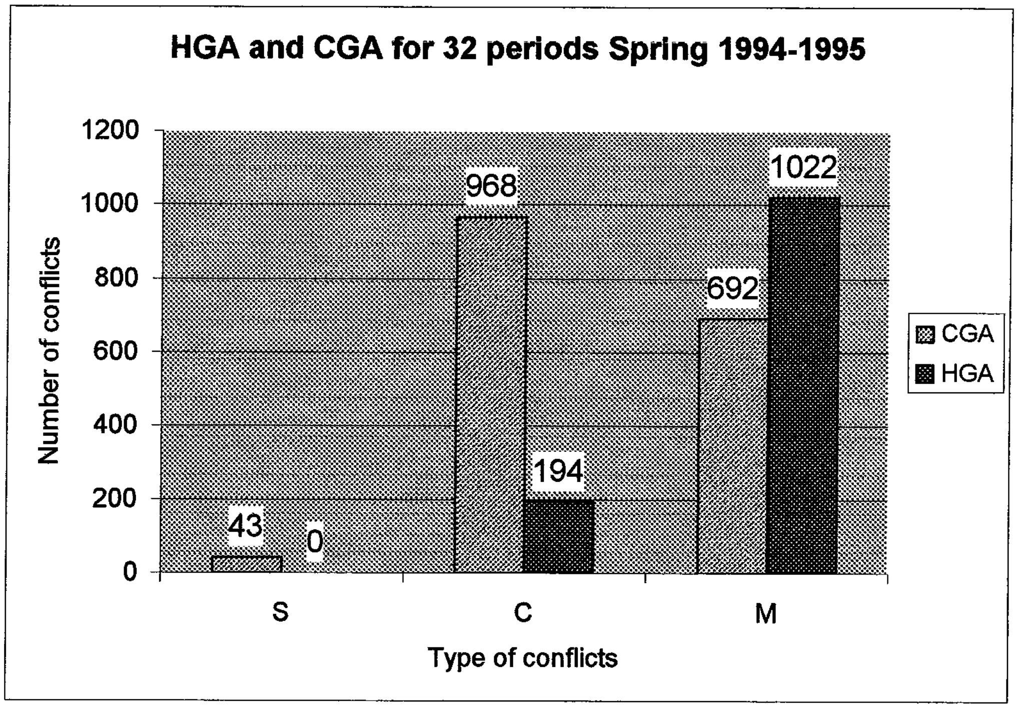

Figure 4. HGA and IGA performance for Spring 1994-1995

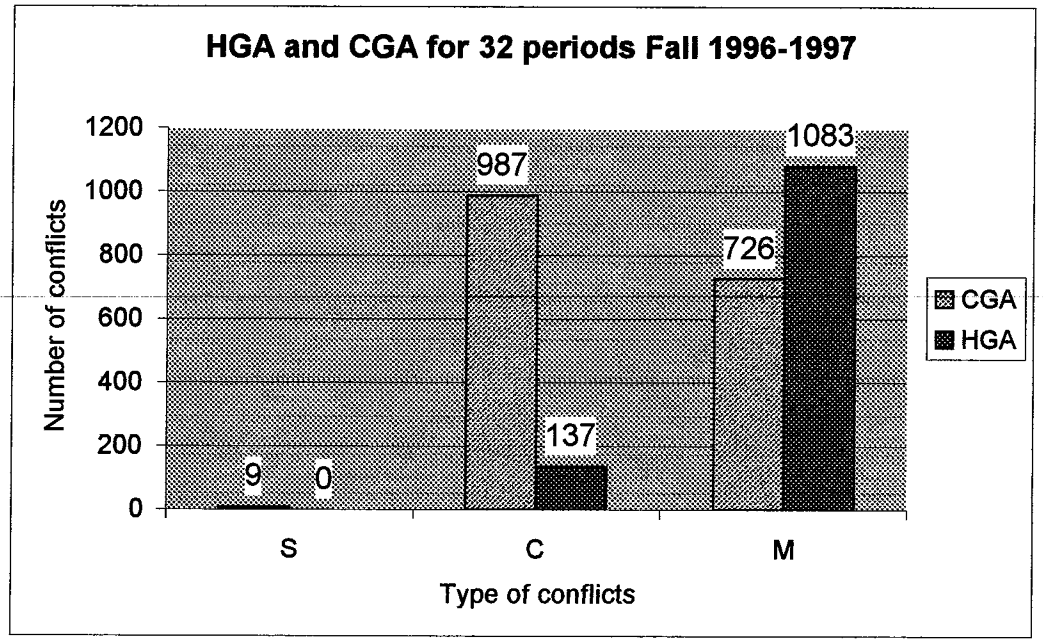

Figure 5. HGA and CGA performance for Fall 1996-1997

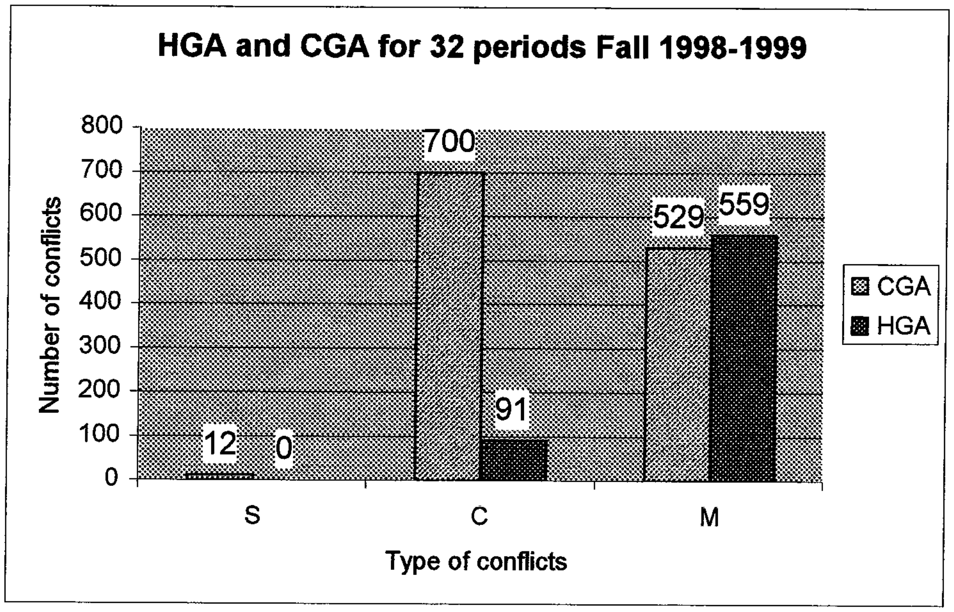

Figure 6. HGA and CGA performance for Fall 1998-1999

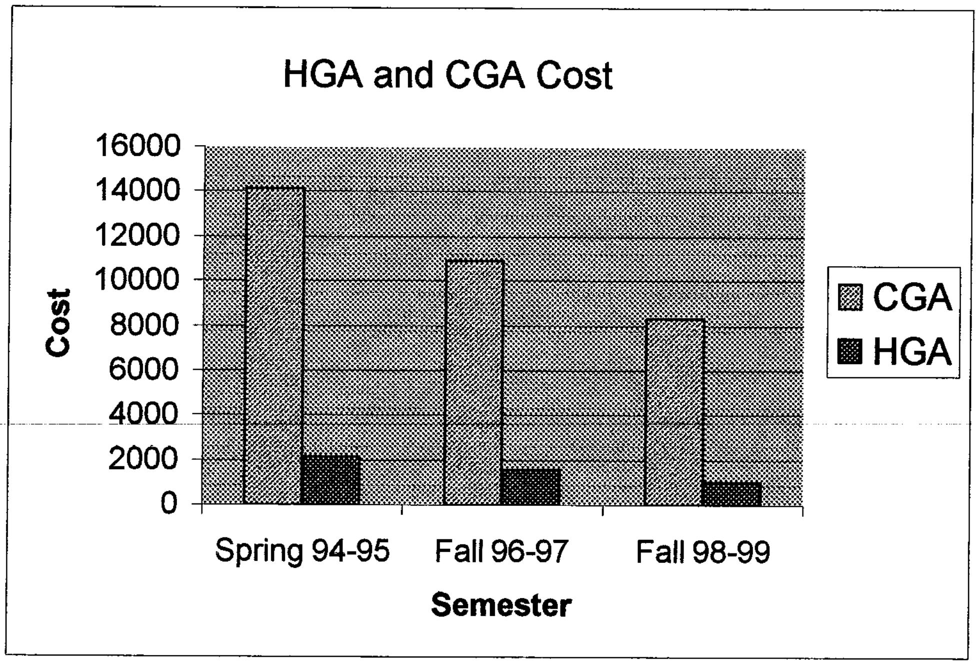

Figure 7. A comparison between the cost of HGA and CGA

CHAPTER 6

Empirical Comparison of IGA and CGA

6.1 Experimental setup for IGA and CGA

In order to evaluate the performance of the Incremental Genetic Algorithm (IGA), we run the Classical Genetic Algorithm (CGA) for Optimal regression testing, general problem optimization, and exam scheduling and we compared them with the results obtained from running IGA on each of the above problems.

In all what follows, the population size is 100 except for exam scheduling the population size varies between 40 and 100 depending on the saved best feasible solutions.

In order to demonstrate that IGA is better than CGA, each problem is modified several times by changing the values of the variables and/or constraints.

For example, in the case of exam scheduling optimization, the modification represents the number of courses, which were found later that no final exam is needed. These final exams can be courses, which were not given during the semester, but were, printed in the courses schedule. The verification is done by checking the final exam schedule that was setup manually by the registrar office. On the other hand, the modifications can represent eliminating periods or adding periods.

6.2 CGA and IGA comparison for optimal regression testing

The modification in regression testing indicates whether certain segments need (or need not) be covered by the subset of retests due to the modification of these segments. Alternatively, it can mean that a test is included (1) or excluded (0) in the test segment.

During the experiments on the optimal regression testing, we varied the number of variants and constraints. We present the results of running CGA and IGA using tables of M program segments and N test cases.

The goal here is to prove that IGA works better than CGA (means less number of generations). When we have 0-1 ILP problem to solve, it is known that there are several software packages (such as lp_solve), which can help in solving them, but on condition that the size of the problem is small (it is not robust when the problem is large). However, if we ask the following question, what about if the problem size is very huge? In addition, what if it represents a critical area in software development and we need fast results for optimal regression testing. For this reason a comparison between CGA and IGA covers the medium size of 0-1 ILP problem to a large one (from 1000x1000 to 10000x10000). Table 4. gives the results for a problem size of 1000x1000. We notice that in the first case where we made 16 random changes to b_i value, CGA has the same number of generations as IGA, but there is significant diffrence in time, this is due to the fact that CGA takes more time in Hill-Climbing procedure than IGA takes to arrive to the feasible solution with which IGA has started.

Table 4. Results of CGA and IGA with MxN = 1000x1000

	CGA				IGA	
Optimal Regression	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
1000x1000 16 changes	10	19	257	10	18	161
1000x1000 32 changes	17	18	300	10	18	156

In The next experiment, we increased the size of the program segments and the test cases. However, we kept the size of the population unchanged. The results see Table 5. shows again no significant improvement in the number of generations in finding an optimal solution using IGA.

Table 5. Results of CGA and IGA with MxN = 2000x2000

	CGA				IGA	
Optimal Regression	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
2000x2000 16 changes	17	20	1339	10	20	1050
2000x2000 64 changes	10	21	927	10	20	726

Table 6. shows the results of doubling the size, and the modifications to the constarints. Again, no significant improvement until the size is increased to 6000x6000 and later to 10000x10000 see Table 7. the results in this table show that as the problem get more complex (the size is very large) IGA outperform CGA significantly.

Table 6. Results of CGA and IGA with MxN = 4000x4000

	CGA			IGA		
Optimal Regression	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
4000x4000 128 changes	20	16	1339	20	16	1050
4000x4000 256 changes	23	16	1450	20	16	1320

Table 7. Results of CGA and IGA with MxN = 6000x6000 and 10000x10000

	CGA			IGA		
Optimal Regression	**Number of Generations**	**Objective Function**	**Execution Time in hrs**	**Number of Generations**	**Objective Function**	**Execution Time in hrs**
6000x6000 256 changes	26	45	1 hrs	16	45	0.65 hrs
6000x6000 512 changes	35	44	1.31 hrs	10	45	0.6 hrs
10000x10000 512 changes	29	49	2.9 hrs	10	50	1.25 hrs
10000x10000 768 changes	23	49	2.1 hrs	10	49	1.2 hrs

6.3 IGA and CGA comparison for general optimization

In general optimization two different problems are chosen. In the first problem the variables can be $\{-1, 0, 1\}$, and the constraints ranges between 0 and 20. In the second problem, the variables can be $\{-1, 0, 1\}$, and the constraints can be $\{0,1\}$.

The results in Table 8 shows that IGA performs better. In addition, we change the constraints b_i to different values, and we make some modifications, Table 9 shows another good performance of IGA.

Table 8. Results of IGA and CGA for variables {-1,0,1} and constraints 0..20

	CGA				IGA	
General optimization	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
1000x1000 16 changes	37	195	39	10	194	12
1000x1000 32 changes	29	199	35	20	189	22

In general optimization problem, there are clear evidences that even with small size problems but more complex, IGA shows improvement with comparison to CGA. The reason is that it takes more generations to find an optimal solution for this type of complex problems using CGA. However, IGA starts with a solution that is closer to the optimal solution.

Table 9. Results of IGA and CGA for variables {-1,0,1} and constraints {0,1}

	CGA			**IGA**		
General optimization	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
1000x1000 16 changes	36	100	38	15	99	17
1000x1000 32 changes	43	111	110	32	111	83

Increasing the size of the complex problem is another way to demonstrate the good performance of IGA. In Table 10 and 11 another two results of different problems, here we see that the number of generations increases for CGA, while it is trying to find an optimal solution in the search space.

Table 10. Results of IGA and CGA for variables {-1,0,1} and constraints {0,1}

	CGA			**IGA**		
General optimization	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
4000x4000 128 changes	23	163	956	10	151	435
4000x4000 192 changes	56	150	2163	10	151	450

Table 11. Results of IGA and CGA for variables {-1,0,1} and constraints 0..20

	CGA				**IGA**	
General optimization	**Number of Generations**	**Objective Function**	**Execution Time in secs**	**Number of Generations**	**Objective Function**	**Execution Time in secs**
4000x4000 128 changes	31	449	1274	10	440	463
4000x4000 192 changes	31	459	1273	10	440	472

6.4 IGA and CGA comparison for Exam Scheduling

Table 12 shows the semesters, number of exams, number of rooms, and the number of periods…etc, we used these numbers to do the experiments. In addition, we varied the number of generations to improve the quality of output, this variation depends on the number of the saved chromosomes from the previous run of the Hybrid Genetic Algorithm (HGA) which is the basis of both IGA and CGA. Moreover, the condition states that the initial population of the IGA consists of at least 50 % or more of the saved feasible chromosomes. For this reason, the population number varies between 40 and 100, with each chromosome represents a complete schedule (courses, the period, and the room to which the courses are assigned).

Further, different results are obtained depending on the choice of the values of the coefficients of the objective function. The different versions we used are shown in Table 13.

Table 12. Attributes of the exam scheduling problem

Attribute	*Spring 94-95*	*Fall 95-96*	*Fall 98-99*
Number of enrollments	9550	12275	12406
Number of enrollments manual	9550	9735	10836
Number of exams	336	426	477
Number of exams manual	336	357	359
Number of exam periods	20-40	20-40	20-40
Number of sessions within a day	4	4	4
Number of available classrooms	21	21	21

First step in the experiments is to run each problem before the modifications and to save best feasible and infeasible solutions. Next step is to vary the number of periods from 20 (5 days) to 40 (10 days). The reason for changing the number of periods is to show further differences between CGA and IGA.

Table 13. Different Weights values used in the objective function

Version number	α	ϕ	σ	γ
V1	100	5	0.2	200
V2	100	1	1	200

In the first experiment, the number of periods is 32, and the comparisons, which are carried out between IGA and CGA cover three semesters for different years. After the modifications, the results show that IGA requires fewer number of generations, this is due to the fact that saving large number of best feasible solutions narrowed the search in the search space where subspaces of feasible solutions are disjoint from the infeasible solutions. Moreover, Table 14 shows that IGA performs better than HGA in decreasing simultaneous, consecutive, and multiple conflicts.

Another experiment, we try to vary the complexity of the problem by decreasing and increasing the number of days of exams. In addition, we modified the constraint that forces the courses with different sections to have their exams in the same day; we carried out this modification to Spring 1994-1995 semester. The results of comparisons between IGA and CGA are summarized in Table 15. Moreover, we modified the number of courses (eliminating some courses), this means decreasing the number of conflicts. This type of modification is carried out to Fall 1996-1997 and Fall 1998-1999 semesters, the results of comparisons are summarized in Table 16 and

Table 17. Furthermore, these results are displayed as charts see Figures 8,9,10, and 11 to make them easily understood.

Table 14. Experimental results for 32 periods for IGA and CGA

		S_{SE}	S_{CE}	S_{ME}	Number of rooms used	Violation of room capacity	Objective function	Number of generations	Time in hours
Spring 94-95	HGA-V1	0	258	987	21	0	1487	66	0.82
	IGA-V1	0	196	1115	21	0	1203	34	0.44
	HGA-V2	0	344	810	21	0	1154	84	1
	IGA-V2	0	345	802	21	0	1147	62	0.75
Fall 96-97	HGA-V1	0	92	828	21	0	625	101	1.18
	IGA-V1	0	82	697	21	0	549	59	0.7
	HGA-V2	0	268	600	21	0	868	43	0.62
	IGA-V2	0	224	494	21	0	718	32	0.46
Fall 98-99	HGA-V1	0	66	377	21	0	405.4	69	1.1
	IGA-V1	0	57	334	21	0	351.8	39	0.62
	HGA-V2	0	41	197	21	0	238	59	1
	IGA-V2	0	45	185	21	0	230	33	0.53

Table 15. Experimental results for different periods for Spring 94-95

HGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	14	2	0	0	0
S_{CE}	1230	879	572	231	95
S_{ME}	2148	1680	1060	647	422
Room violation	0	0	0	0	0
Objective function	4778	2759	1632	878	517
Generations	74	60	89	72	77
Time in hours	1.313	0.64	1.47	0.75	1.1
IGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	13	2	0	0	0
S_{CE}	1175	804	455	180	91
S_{ME}	2116	1477	1109	527	423
Room violation	0	0	0	0	0
Objective function	4591	2481	1564	707	514
Generations	43	40	40	63	34
Time in hours	0.67	0.43	0.69	0.6	0.55

Table 16. Experimental results for different periods for Fall 96-97

HGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	8	0	0	0	0
S_{CE}	904	571	367	104	34
S_{ME}	1645	1170	758	360	247
Room violation	0	0	0	0	0
Objective function	3349	1941	1125	464	281
Generations	70	63	65	75	92
Time in hours	0.9	0.64	0.8	0.88	1.56
IGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	5	0	0	0	0
S_{CE}	899	525	281	87	39
S_{ME}	1764	1129	782	344	231
Room violation	0	0	0	0	0
Objective function	3163	1654	1063	431	270
Generations	55	44	52	40	54
Time in hours	0.4	0.5	0.52	0.55	0.72

Table 17. Experimental results for different periods for Fall 98-99

HGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	4	0	0	0	0
S_{CE}	431	263	163	40	32
S_{ME}	757	534	376	176	99
Room violation	0	0	0	0	0
Objective function	1588	797	539	216	131
Generations	57	29	32	30	56
Time in hours	0.5	0.25	0.30	0.3	0.53
IGA	Periods = 20	Periods = 24	Periods = 28	Periods = 36	Periods = 40
S_{SE}	3	0	0	0	0
S_{CE}	410	238	135	38	11
S_{ME}	831	506	335	176	108
Room violation	0	0	0	0	0
Objective function	1541	744	460	214	118
Generations	33	21	22	22	31
Time in hours	0.28	0.2	0.20	0.25	0.28

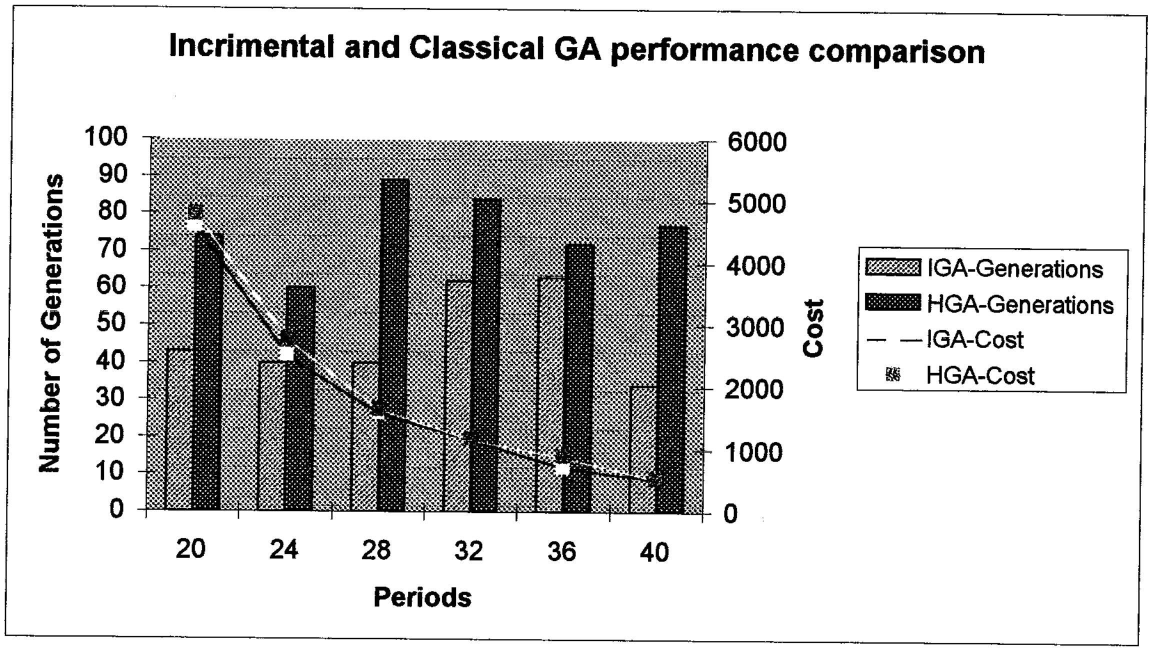

Figure 8. Spring 1994-1995 experimental results graph

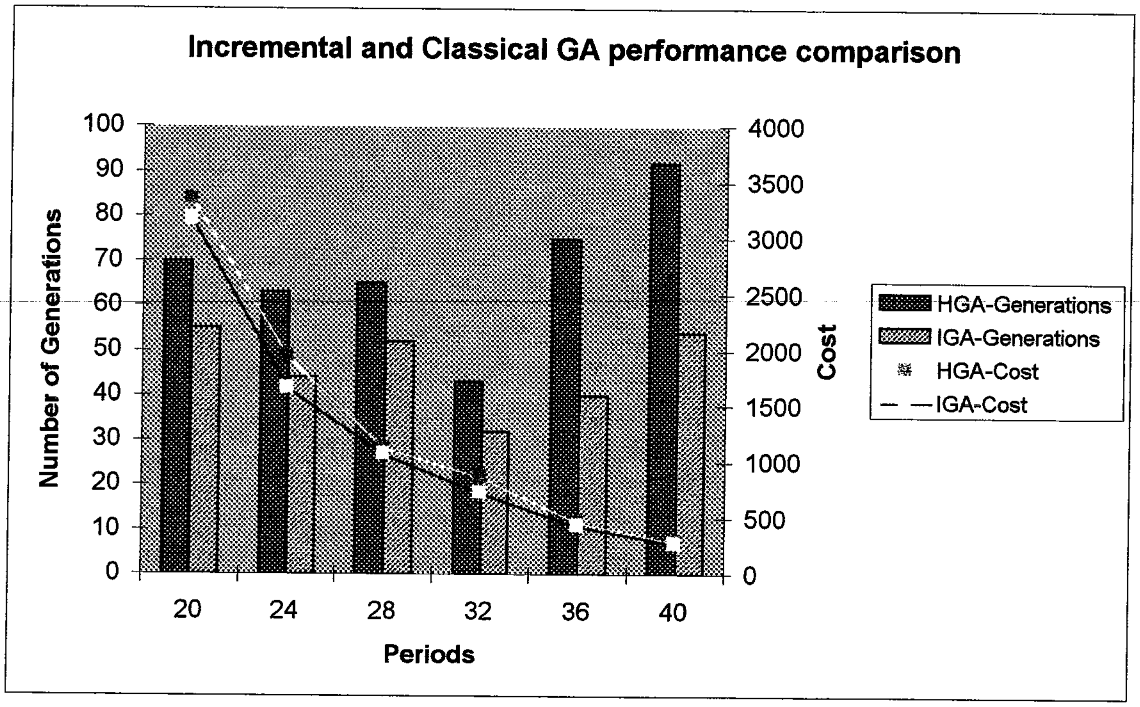

Figure 9. Fall 1996-1997 experimental results graph

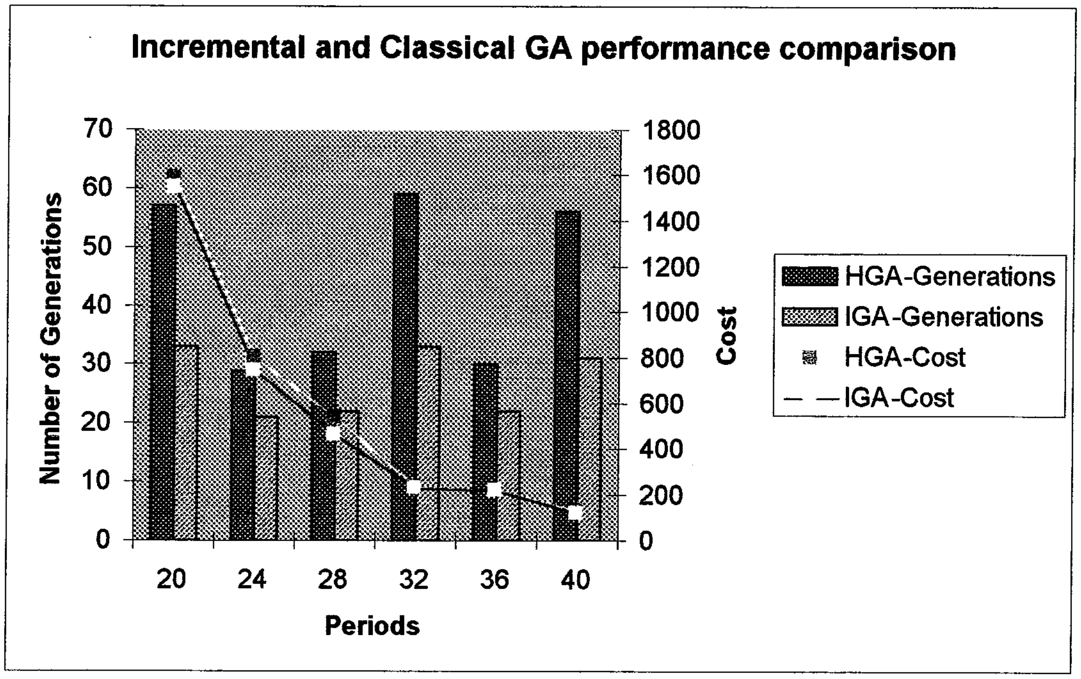

Figure 10. Fall 1998-1999 experimental results graph

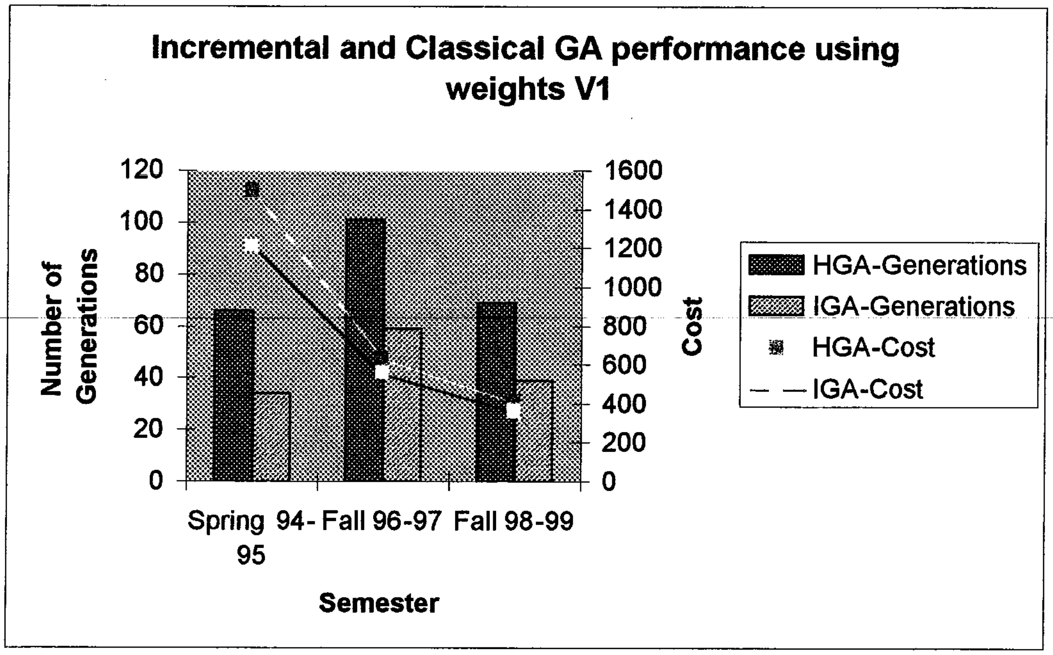

Figure 11. Experimental results for 32 periods for three different semesters

CHAPTER 7

Conclusion

Experiments on three different problems have demonstrated that IGA, which creates part of its initial population from the feasible and infeasible chromosomes we save during the run of CGA on these problems first time, performs better than CGA, which starts its initial population from random generated chromosomes. The performance is clear when we see that IGA requires significantly less number of generations to obtain an optimal solution than CGA. In addition, the value of the optimal solution we obtain by IGA, is similar or sometime slightly better than the one we obtain by running CGA.

Moreover, a new Hybrid GA for exam scheduling is created to be able to obtain better results, and to be able to compare IGA and CGA more efficiently. This Hybrid GA demonstrates to be better in reducing the number of simultaneous, consecutive, and multiple conflicts than any of the previously created GA for exam scheduling in L.A.U. In addition, we can control the number of conflicts we want to reduce in HGA using weights factor in the objective function better than we can do in any of the previous GA for exam scheduling in L.A.U.

Bibliography

Baeck, T. (Ed.) (1997) *Proc. International Conference on Genetic Algorithms*, San Mateo, Ca.: Morgan Kaufmann.

Banzhaf W. (Ed.) (1999) *Proc. of the Generic and Evolutionary Computations Conference*, San Francisco,Ca.: Morgan Kaufman.

Davis, L. (Ed.) (1991) *Handbook of Genetic Algorithms*, New York : Van Nostrand Reinhold.

De Jong, K. A. (1975) Analysis of the Behavior of a Class of Genetic Adaptive Systems. Ph.D. Thesis, Dept. of Computer and Communication Sciences, Ann Arbor: University of Michigan.

Ebeling, W. , Rechenberg, I., Schwefel, H.P., and Voigt, H.M. (Eds.) (1996) Parallel Problem Solving from Nature, Lecture Notes in Computer Science 1141, Berlin: Springer-Verlag.

Fogel, D. (Ed.) (1998) *Proc. IEEE Int. Conference on Evolutionary Computation*, Piscataway, NJ :IEEE Press.

Goldberg D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, Addison-Wesley, Reading.

Haupt, R. , Haupt , S. (1997) *Practical Genetic Algorithms,* New York: John Wiley & Sons Inc.

Kallel L. and Schoenauer M. (1997) Alternative Random Initialization in Genetic Algorithms. *Proc. of 7th International Conference on Genetic Algorithms, East Lansing, Mi:* Morgan Kaufman.

Mansour, N. and El-Fakih, K. (1999) Simulated Annealing and Genetic Algorithms for Optimal Regression Testing, *Journal of Software Maintenance: Research & Paractice,* John Wiley & Sons.

Michalewicz, Z., Fogel, D. (2000) *How to solve it: Modern Heuristics,* Berlin: Springer.

Mikati, Z. (1999) Exam Scheduling Algorithms, MSc. Project., L.A.U.

Pham, D., Karaboga, D., (2000) *Intelligent Optimisation Techniques,* London: Springer.

Sait, S. and Youssef, H., (1999*) Iterative Computer Algorithms with Applications in Engineering,* Los Alamitos: IEEE computer Society.

Schoenauer, M. and Kallel, L. (1996) Fitness Distance Correlation For variable length representation. *Technial Report 363,* CMAP Ecole Polytechnique.

Tarhini, A. and Mansour, N. (1998) Natural Optimization Algorithms for Exam Scheduling. *Proc. of the IASTED International Conference on Computer Systems and Applications,* ACTA Press.