International Journal of Computer Science and Engineering (IJCSE)
ISSN(P): 2278-9960; ISSN(E): 2278-9979
Vol. 3, Issue 5, Sep 2014, 7-22

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

HYBRID METAHEURISTIC OF SIMULATED ANNEALING AND GENETIC ALGORITHM FOR SOLVING EXAMINATION TIMETABLING PROBLEM

OYELEYE C. AKINWALE, OLABIYISI S.OLATUNDE, OMIDIORA E. OLUSAYO & FAGBOLA, TEMITAYO

Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Ogbomoso, Nigeria

ABSTRACT

This paper introduces a hybrid metaheuristic of Simulated Annealing (SA) and Genetic Algorithm (GA) and demonstrates its superiority over the two hybridized algorithms in terms of their simulation time and software complexity measurement when used to solve a typical University Examination Timetabling Problem (ETP).Preparation of a timetable consists basically of allocating a number of events to a finite number of time periods (also called slots) in such a way that a certain set of constraints is satisfied. The developed model was used to schedule the first semester examination of Ladoke Akintola University of Technology, Ogbomoso Nigeria during the 2010/2011 session. A task involving 20,100 students, 652 courses, 52 examination venues for 17days excluding Saturdays and Sundays.The use of the implemented model resulted in significant time savings in the scheduling of the timetable, a shortening of the examination period and a well spread examination for the students. Also, none of the lecturers / examination invigilators was double booked or booked successively. It was clearly evident that the hybrid model outperformed Simulated Annealing and Genetic Algorithm in most of the evaluated parameters.

KEYWORDS: Hybrid Model, Simulated Annealing, Genetic Algorithm, Examination Timetabling Problem, Simulation Time and Software Complexity

INTRODUCTION

Preparation of a timetable consists basically of allocating a number of events to a finite number of time periods (also called slots) in such a way that a certain set of constraints is satisfied. Two types of constraints are usually considered, hard constrains, that have to be fulfilled under all circumstances, and soft constraints, that should be fulfilled if possible. In some cases, it is not possible to fully satisfy all the constraints, and the aim turns to be finding good solutions subject to certain quality criteria (e.g., minimizing the number of violated constraints, or alternatively satisfaction of hard constraints, while the number of violated soft constraints is minimized (Keshav *et al*, 2007). A practical timetable that does not violate hard constraints is called a feasible timetable. The second international timetabling competition (ITC2007) described hard and soft constraints (Di Gaspero et al, 2007).

Timetablings are combinatorial optimization problems, which consist of scheduling a set of courses within a given number of rooms and time periods. Solving a real world timetablingproblem manually often requires a significant amount of time, sometimes several days or even weeks (Abdennadher *et al*, 2000). The manual solution of the timetabling problem usually requires several days of work and the final solution may be unsatisfactory because it is a highly complex task to verify all constraints. For the above reasons, considerable attention has been devoted to automated timetabling. A large number of variants of the timetabling problem have been proposed in the literature, which differs from each other

based on the type of institution involved and the type of constraints imposed by the examination policy of the institution. Preparation of an academic examination timetable is a typical scheduling problem that appears to be a tedious job in every academic institute once or twice a year. The problem involves the arrangement of courses, students, teachers and rooms at a fixed number of time-slots, respecting certain restrictions. Wren defines the general problem of timetabling as follows: "*Timetabling* is the allocation, subject to constraints, of given resources to objects being placed in space time, in such a way as to satisfy as nearly as possible a set of desirable objectives" (Wren, 1996).

The inability of the classical method to handle the large number of real and integer variables involved in solving this class of problem and especially the number of constraints involved paved way for the adoption of non-classical techniques. Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithm (GA), Memetic Algorithm (MA) and Ant Colony System (ACS) are among the main algorithms for solving challenging problems of intelligent systems (Zahra, 2005). In this research, two of these techniques were carefully studied and subsequently hybridized. The two algorithms and the hybrid are in turn compared in terms of their software complexity and simulation time.

Genetic Algorithm (GA) is one of the most popular optimization solutions. It has been implemented in various applications such as scheduling. The operators of GA such as selection, crossover and mutation are applied to populations of chromosomes. Simulated Annealing *(SA*) is a random-search technique which exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) (Elmohamed *et al* 1998; Omidiora *et al*, 2009 and Oyeleye *et al*, 2012). In addition, the search for a minimum in a more general system forms the basis of an optimization technique for solving combinatorial based problems. It is generally regarded as a modified version of hill climbing algorithm. It has been proved that by carefully controlling the rate of cooling of the temperature, SA can find the global optimum. However, this requires infinite time. Fast annealing and very fast simulated reannealing (VFSR) or adaptive simulated annealing (ASA) are each in turn exponentially faster and overcome this problem. SA's major advantage over other methods is an ability to avoid becoming trapped in local minima. The algorithm employs a random search which not only accepts changes that decrease the objective function (assuming a minimization problem), but also some changes that increase it. An effective solution technique to ETP could be applied to other scheduling problems (Abramson, 1991).

Among the leading paradigms for solving ETP are GA and SA, however, the two algorithms are less efficient because SA converges at an excessive time while GA utilizes excessive memory before returning result. Hence, there is a need to develop an improved algorithm that could both run at a lesser time and utilize less computing resources (Oyeleye, *et al*., 2012). In this work, a hybrid algorithm of SA and GA was developed such that it overcomes the weaknesses and combines the strength of these two existing algorithms to solve the a typical university ETP.

Timetabling Problem

This problem drew the attention of the researchers in the early 60's with the study of Gotlieb in 1962, who formulated a class-teacher timetabling problem by considering that each lecture contained one group of student, one teacher, and any number of time-slots which could be chosen freely. Schaerf, surveyed that most of the early techniques for automated timetabling were based on successive augmentation (Schaerf, 1999), where a partial timetable was filled in lecture by lecture until either all lectures were scheduled or no further lecture could be scheduled without violating constraints. In another survey, Abramson in 1991 reported the general techniques applied to the problem in the past, such as network flow analysis, random number generator, integer programming, and linear algorithm. In addition to these, worth

mentioning methods are exact method-based heuristic algorithm (De Werra, 1985), and graph coloring theory (Neufeld and Tartar, 1974). However, the classical techniques are not fully capable to handle the large number of integer and/ or real variables and constraints, involved in the huge discrete search space of the timetabling problem. These inadequacies of classical techniques have drawn the attention of the researchers towards the non-classical techniques. Worth mentioning non-classical techniques, that were / are being used to solve the problem, are genetic algorithms (Colorni et al., 1990, 1992; Abramson and Abela, 1992), neural network, simulated annealing, and tabu search algorithm.

However, compared to other non-classical methods, the widely used are the genetic/ evolutionary algorithms (GAs/ EAs). The reason might be their successful implementation in a wider range of applications. Once the objectives and constraints are defined, GAs appear to offer good solutions by evolving without a problem solving strategy (Al-Attar, 1994). In the GA, used by Abramson and Abela (1992) to school timetabling problem, a solution is likely to loose or duplicate a class under crossover operator. A repairing mechanism, in the form of a mutation operator, was used by them to fix up such lost or duplicated classes. Piola in 1994 applied three evolutive algorithms to school timetabling problem, and showed their capability to tackle highly constrained combinatorial problems, such as timetabling problem. A timetable is essentially a schedule which must suit a number of constraints. Constraints are almost universally employed by people dealing with timetabling problems (Burke *et al*, 1994).

Simulated Annealing and Genetic Algorithm

Schaffer and Eschelman, 1996 presented the comparison of Simulated annealing (SA) and genetic algorithms (GA) from two major perspectives. SA and GA are two stochastic methods currently in wide use for difficult optimization problems. Their theoretical backgrounds and empirical comparison are of utmost importance, especially to this work.

Theoretical Comparison

Theoretically, SA and GA are quite close relatives, and much of their difference is superficial. The two approaches are usually formulated in different ways using different terminologies. With SA, one usually considers solutions, their costs, and neighbours and moves; while with GA, one talks about individuals (or chromosomes), their fitness, and selection, crossover and mutation. This difference in terminology reflects the differences in emphasis, but also serves to obscure the similarities and the real differences between SA and GA. Basically, SA can be thought as GA where the population size is only one. The current solution is the only individual in the population. Since there is only one individual, there is no crossover, but only mutation.

This is in fact the key difference between SA and GA. While SA creates a new solution by modifying only one solution with a local move, GA also creates solutions by combining two different solutions. Whether this actually makes the algorithm better or worse, is not straightforward, but depends on the problem and the representation.It should be noted that both SA and GA share the fundamental assumption that good solutions are more probably found "near" already known good solutions than by randomly selecting from the whole solution space. If this were not the case with a particular problem or representation, they would perform no better than random sampling.What GA does differently here is that it treats combinations of two existing solutions as being "near", making the assumption that such combinations (children) meaningfully share the properties of their parents, so that a child of two good solutions is more probably good than a random solution. Again, if for a particular problem or representation this is not the case, then GA will not provide an

advantage over SA.This obviously depends on what the crossover operator is. If the crossover operator is poorly chosen in respect to the problem and its representation, then a recombination will effectively be a random solution. This kind of destructive crossover often results with combinatorial problems if a chromosome directly expresses a solution, and can sometimes be cured by choosing a different representation, where a chromosome is thought of as a "genotype" that only indirectly expresses a solution, a "phenotype". This approach, with two levels of solution representation, has traditionally been specific to GA (Schaffer and Eschelman, 1996).

Mühlenbein (1997) presents a theoretical analysis of genetic algorithms based on population genetics. He counters the popular notion that models that mimic natural phenomenon are superior to other models. The article argues that evolutionary algorithms can be inspired by nature, but do not necessarily have to copy a natural phenomenon. He addresses the behavior of transition operators and designs new genetic operators that are not necessarily related to events in nature, yet still perform well in practice.

Empirical Comparisons

As advocated by JukkaKohonen in 1999, execution time should be considered first when dealing with the empirical comparisons of optimization algorithms for combinatorial problems. It should be a key element of any such comparison. One of the most important factor considered before choosing the winner during the second international timetabling competition (ITC-2007) was the computation time (McCollum, 2009). This can be viewed at http://www.cs.qub.ac.uk/itc2007/winner/bestexamtrack.htm.

After all, if there were no limits on execution time, one could always perform a complete search, and get the best possible solution. Most stochastic algorithms can do the same, given unlimited time. In practice, there are always some limits on the execution time.

Also, a key property of stochastic algorithms such as SA and GA is that given more time, they usually provide better solutions, at least up to some limit. If, in an empirical comparison, algorithm A is allowed to use more time than algorithm B, their solution qualities are no longer comparable, since there is no indication on how good solutions algorithm A would have produced given the same time as algorithm B. It is, of course, possible that for short time limits, one algorithm outperforms, while for longer time limits, the other one does.

In 1996, Manikas and Cain compared SA and GA for a circuit partitioning problem. They very carefully analyze the statistical confidence of the results, when comparing approximately 20 trials with each algorithm. However, there is no mention of the execution time used. Still, they concluded that "the genetic algorithm was shown to produce solutions equal to or better than simulated annealing". This conclusion may be true, but its relevance is in doubt because of the missing information (Manikas and Cain, 1996).

In 1996 also, Mann and Smith compared SA and GA for a traffic routing problem. They reported the execution times, but the comparison mainly focuses on solution cost. However, they do not report what happens if the algorithms are given the same amount of time(JukkaKohonen, 1999).

GA was proposed by Holland as an algorithm for probabilistic search, learning, and optimization, and is based in part on the mechanism of biological evolution and Darwin’s theory of evolution. This algorithm is a powerful search tool, particularly when applied to solve combinatorial optimization problems of this nature. However, the implementation of an

efficient GA often faces two major problems, on one side, the premature convergence to local optima and on the other the requirements for the GA search of long times in order to reach an optimal or a good suboptimal solution. In order to prevent the premature convergence, the coupling of GA and one point search algorithm (local search algorithm), such as Simulated Annealing to form hybrid GA can be advantageous (Kanoh and Nakamura, 2000). SA repeatedly generates succeeding solutions using the local search procedure. Some of them are accepted and some will be rejected, according to a predefined acceptance rule. The acceptance rule is motivated by an analogy with annealing processes in metallurgy. On the other hand, GA repeatedly propagates generations. Genetic algorithms (Liepens and Hilliard, 1989) emulate the evolutionary behavior of biological systems. They generate a sequence of populations of candidate solutions to the underlying optimization problem by using a set of genetically inspired stochastic solution transition operators to transform each population of candidate solutions into a descendent population. The three most popular transition operators are reproduction, cross-over, and mutation (Davis, 1991). Davis and Principe (1991) and Rudolph (1994) attempt to use homogeneous finite Markov chain techniques to prove convergence of genetic algorithms (Cerf, 1998), but are unable to develop a theory comparable in scope to that of simulated annealing.

One criticism of simulated annealing is the slow speed at which it sometimes converges. Delport (1998) combines simulated annealing with evolutionary algorithms to improve performance in terms of speed and quality of solution. He improves this hybrid system of simulated annealing and evolutionary selection by improving the cooling schedule based on fast recognition of the thermal equilibrium in terms of selection intensity. This technique results in much faster convergence of the algorithm.

Sullivan and Jacobson (2000) links genetic algorithms with simulated annealing using generalized hill climbing algorithms. They first link genetic algorithms to ordinal hill climbing algorithms, which can then be used, through its formulation within the generalized hill climbing algorithm framework, to form a bridge with simulated annealing.

Software Complexity Measures

Software complexity measures can be used to predict critical information about reliability and maintainability of software systems from automatic analysis of the source code (Olabiyisi *et al*, 2005 and 2007). Complexity measures also provide continuous feedback during a software project to help control the development process. During testing and maintenance, they provide detailed information about software modules to help pinpoint areas of potential instability.

There are a number of ways to quantify complexity in a program. The best known metrics which provide such features are Halstead's volume and McCabe's cyclomatic number. These metrics have been extensively validated and compared (Aggarwal *et al*, 2002, Olabiyisi *et al*, 2005 and 2007). This research focuses on the former and Lines of Codes (LOC) in evaluating the three implemented algorithms..

MATERIALS AND METHODS

The two standard algorithms (i.e simulated annealing and Genetic algorithm) and the developed hybrid were implemented using MATLAB development kit on an Intel® Dual core CPU with 2.20GHz speed, 2.91GB Random Acccess Memory (Accessible) and 146GB hard disk drive with windows 7 ultimate edition.

Problem Representation

Examination timetabling is a specific case of the more general timetabling problem.

In the case of examination timetabling, a set of exams

$E = \{e1, \ldots, en\}$ to be scheduled within a certain number of periods

$P = \{p1, \ldots, pm\}$

subject to a variety of hard and soft constraints (see Table 2 below).

(Piola, 1994 and Burke *et al*, 1996; Oyeleye *et al*, 2012).

Simulated Annealing Pseudocode

Start with the system in a known configuration, at known energy *E*

T=temperature =hot; frozen=false;

While (! frozen) {

repeat {

Perturb system slightly (e.g., moves a particle)

Compute *E*, change in energy due to perturbation

If ($\Delta E < 0$)

Then accept this perturbation, this is the new system config

Else accept maybe, with probability = $\exp(-\Delta E/KT)$

} until (the system is in thermal equilibrium at this *T*)

If (ΔE still decreasing over the last few temperatures)

Then $T=0.9T$//cool the temperature; do more perturbations

Else frozen=true

}

return (final configuration as low-energy solution)

Pseudocode for Genetic Algorithm

Step 1 Generate initial population.

Step 2 Evaluate population.

Step 3 Apply Crossover to create offspring.

Step 4 Apply Mutation to offspring.

Step 5 Select parents and offspring to form the new population for the next generation.

Step 6 If termination condition is met finish, otherwise go to Step 2.

In general, a GA has five basic components:

- A genetic representation of potential solutions to the problem.
- A way to create a population (an initial set of potential solutions).
- An evaluation function rating solutions in terms of their fitness.
- Genetic operators that alter the genetic composition of offspring (crossover, mutation, selection, etc.).
- Parameter values that genetic algorithm uses (population size, probabilities of applying genetic operators, etc.) (Gen, Cheng, and Lin, 2008; Oyeleye *et al*, 2012).

Pseudocode for the Developed Hybrid SAGA Algorithm

The developed SAGA hybrid algorithm for solving ETP is as follows:

- Set the Initial Temperature T and the Cooling rate to a pre-determined value.
- Initialize the chromosome length N, number of generation and Population size.
- Generate timeslot sequence and classroom capacity sequence for *N* chromosomes.
- Find the best fit room capacity value for each and every chromosome using the objective function and also find the maximum classroom-capacity value (*best*) among *N* number of classrooms.
- Select 2 chromosomes from Number ofchromosomes (Perform Selection).
- Crossover the selected chromosomes with the probability as 0.9 and Mutate the new chromosomes with the probability to get new chromosomes. (Perform Crossover and Mutation).
- Find the appropriate Timeslot values for newly generated chromosomes using the objective function.
- Choose the *N* best chromosomes which have the maximum room-capacity values from the newly generated and also from old chromosomes.
- Find the optimal room-capacity value (*best*) among the *N* best chromosomes.
- If *best* chromosome is not changed over a period of time then find a new chromosome.
- Accept the new chromosome as *best* with probability as exp-($\Delta E/KT$), even though current position is worse. Here ΔE is the difference between the initial temperature T and the temperature of the best chromosome value. K is Boltzmann's constant and it is 1.381×10^{-23}.
- Reduce *T by the cooling rate* (Cooling Schedule).
- If the maximum number of iterations is reached Or optimal value is obtained Or $T \leq 0$ then Terminate and Produce Result Else Goto step 3.

Steps 1, 11 and 12 and 13 are the cooling schedule of simulated annealing used as the starting and ending conditions of the hybrid algorithm. They made it possible for the algorithm to converge without wasting the system resources of simulation time. Other steps are used to perform the operations of genetic algorithm.

The hybrid is a representation of genetic algorithm operations fully embedded withing the cooling schedule of simulated annealing.

Data Used for the Work

The following are the set of data used to automatically generate the examination timetable:

- Available venues and their corresponding capacity
- Special examination venue (if any) and capacity
- List of subjects (exams) to be written
- The list of all registered students per exam or course
- The list of available invigilators
- Maximum examination period (no of exam days or weeks)
- Duration of each examination (maximum number of hours)

The Summary of Data Used

Table 1

No_of_courses	652
No_of_venues	52
Total No of students	20,100
Total sitting capacity per time	6872
Exam Venue Capacity	
500 and above	5
200 to 499	1
50 to 199	33
1 to 49	13 (Oyeleye *et al*, 2012)

Complexity of the Three Algorithms

In order to evaluate the coded algorithms, software complexity metrics were adopted. Software complexity metrics used were Halstead software complexity measure and Lines of Code (LOC). Halstead measure calculates program volume, program effort, program level and intelligence content of the program. The formulae for measuring these metrics are as presented in Table 3. These measures are valid under the assumption that the program is "pure," i.e., free of so-called "poor programming practices." (Olabiyisi *et al,* 2005 and 2007; Oyeleye *et al*., 2012).

RESULTS AND DISCUSSIONS

After the implementation of the three algorithms, Table 4 shows the measured parameters and their various values which are in turn used to calculate the software complexity of the algorithms. It should be noted that n1 is the number of distinct operators found in the program, n2 is the number of distinct operands, N1 is the total number of operators, N2 is the total number of operands, N is the addition of N1 and N2 and n is the addition of n1 and n2.

Table 5 shows that the three algorithms produced feasible solutions, because none violated the constraints considered in this work. Table 2 contains the summary of the hard and soft constraints considered.

Simulation Time

The time utilized by an algorithm to run until the result is produced is usually called execution time or simulation time. Table 5 and Figure 1 show the measured values of the simulation time of the three considered algorithms. The simulation time of GA, SA and the developed SAGA Hybrid are 19.73, 56.16 and 17.67 seconds respectively to return a feasible examination timetable. This is a clear evidence that SA utilized more time than GA, while the developed SAGA hybrid algorithm used the least time.

Program Size

The program size is the amount of disk space occupied and it is usually measured in bits, bytes, kilobytes (Kb), Megabytes, Gigabyte, Terabytes, etc depending on the actual size under consideration. Table 5 and Figure 2 show that the program sizes of GA, SA and SAGA Hybrid algorithms are 20Kb, 16.5Kb and 6.5kb respectively. These measured values clearly show that GA code utilized more disk space than SA while the developed SAGA hybrid occupies the least disk space.

Lines of Code

The lines of code (LOC) is the number of lines of the executable codes in a program. Table 5 show the number of lines of the implemented algorithms. The LOC of GA, SA and the developed SAGA hybrid algorithm are 500, 256 and 194 respectively. These values show that GA code has more number of executable lines of code while the developed SAGA hybrid algorithm has the least. This is an indication that the implementation time and effort required by GA was more than the other two and the developed SAGA Hybrid has the least.

Program Volume

The program volume (V) is the value that signifies the volume of the computer memory being utilized during the execution of the implemented algorithms. The program volume for Genetic Algorithm, Simulated annealing and the developed SAGA hybrid are 2108.07, 2088.00 and 1785.99 respectively. These values are shown in Table 5 and Figure 3. It shows that the SAGA Hybrid occupies lesser memory space in terms of volume than the other two, while GA occupies more memory space.

Program Effort

This is widely known as the number of discriminations made in the preparation of a program, it specifies the extent to which personnel involved in software production are effectively engaged. It could also be referred to as the quantitative measure of the effort involved in the implementation of an algorithm.

The measured values for the three considered algorithms are presented in Table 5 and Figure 4. The program effort of GA, SA and the developed SAGA hybrid are 33729.12, 17013.33 and 11230.38 respectively. This is an indication that the program effort of GA is the highest, followed by SA, while the developed SAGA hybrid has the least.

Program Level / Difficulty of Understanding the Program

This program level (L) otherwise called difficulty of understanding a program. As presented in Table 5 and Figure

5, GA, SA, and the developed SAGA hybrid algorithm has 0.06, 0.12 and 0.16 respectively as their values for the difficulty of understanding the program. The result revealed that SA is more difficult to understand than GA while the developed SAGA hybrid is the most difficult to understand of the three implemented algorithms.

Intelligent Content of the Program

The Intelligent Content of the Program is the quantitative representation of how logically reasonable and experienced the program writer is. Table 5 and Figure 6 show the intelligent content of the program for GA, SA, and the developed SAGA hybrid algorithm to be 131.75, 256.25, and 284.03 respectively. These values indicate that the developed SAGA hybrid algorithm is the most logically reasonable closely followed by SA while GA is the least logically reasonable.

Peculiarities of the Developed SAGA Hybrid Algorithm

Some of the peculiar features of the SAGA Hybrid algorithm were studied and evaluated by varying the values of its major components, such as the initial temperature, rate of cooling, number of generations and population sizes. The effect of the variations were consequently observed on the Simulation Time, Number of courses clashed and number of lecturers double booked. Since the three algorithms returned feasible solutions therefore emphasis was further laid on the simulation time.

Table 6 shows that none of the scheduled courses clashed and none of the lecturers on invigilation was double booked. It was also observed that reducing the number of generations reduces the simulation time. Therefore, the more the number of generation the more the simulation time.

Number of Generations and Initial Temperature

Table 6 shows the effect of number of generations and Initial temperature on simulation time. The SAGA hybrid model provided feasible solution with minimum Number of Generations and reasonable Initial Temperature. Table 6 shows that variation in simulation time of various initial temperature and number of generation 10, 100, 1000 and 10000.

The table further shows that at number of generation of 100 and an initial temperature of 100 a reasonable simulation time of 2.6052 seconds was obtained. It was therefore easily observed that the increase in number of generation has a considerable effect on simulation time than increase in initial temperature. So, the more the number of generation the more the simulation time. The variation of the initial temperature does not have a noticeable effect on the simulation time as does the variation in number of generation. See Table 6.

Simulation Time – Initial Temperature and Cooling Rate at Constant Number of Generation and Population Size

Table 7 below shows the simulation time of various cooling rates at different initial temperatures at constant number of generations of 100 and number of population 10000. A thorough study of the table shows that the minimum simulation time was obtained at an initial temperature of 220 and cooling rate of 0.001 as highlighted in table 7.

In summary, both initial temperature and cooling rate of the Simulated Annealing components have impact on the simulation time of the developed SAGA hybrid algorithm. This is an indication that by carefully selecting an initial temperature and carefully selecting a suitable cooling rate, the algorithm tends to perform better in returning quickly a feasible solution.

The Effect of Population Size and Number of Generation on Simulation Time at Constant Initial Temperature and Cooling Rate

The developed SAGA hybrid algorithm was subjected to various population sizes and different Number of generation at constant Initial temperature of 220 and cooling rate of 0.001. The Simulation time of the algorithm was observed to vary as presented in Table 8. It was observed that 10000 number of population with 10 number of generation produced the minimum simulation time.

This cannot be taken as the best parameter since the more the number of generation the better will be the quality of the final result. Table 8 further shows that the rate at which the simulation time increases with number of generation is higher than the rate it increases with number of population. This shows that the more the number of generations the more the simulation time. Hence, by carefully controlling the Number of Generation also leads to a greater reduction in the time the algorithm takes before returning feasible results. It can therefore be deduced here, that increase in the Number of Population has a negligible effect on the Simulation Time compared to the highly noticeable effect of increase in the Number of Generation on simulation time.

CONCLUSIONS

The three considered algorithms produced feasible university examination timetable, but the developed SAGA hybrid algorithm used the least computing resources of time and space. It utilized the least simulation time, program size, lines of code, program volume, program effort, and the highest intelligent content of the program.

Conclusively, the results generated from the analysis indicates a very high consumption of computing resources by genetic algorithm while simulated annealing results show that though the consumption of computing resources is reduced yet the two algorithms still consume considerable computing resources compared to their hybrid counterpart.

Table 2: Summary of Constraints Considered

Label	Definition
HC1	The number of exams a student will write at a time
HC2	Number of classes a teacher should be at a time
HC3	Number of examination in the schedule
HC4	The Type and Capacity of the room where a class is to be scheduled
HC5	Number of timeslot at which an examination of a course is to be scheduled
SC1	Total number of free time-slots between two examinations (or events) of students
SC2	Total number of consecutive classes of a teacher

HC – Hard Constraints
SC – Soft Constraints

Table 3: Formulae for Measuring the Complexity Metrics of the Three Algorithms

Complexity Metrics	Formulae
Volume (V)	N* $\log_2$n
Effort (E)	V/L
Program Level (L)	(2*n2) / (n1*N2)
Intelligent Content of the program (I)	L*V

Table 4: Data Obtained for Measuring the Complexity of the Three Algorithms

	GA	SA	SAGA HYBRID
No of Distinct operators (n1)	14	10	13
No of Distinct operands (n2)	42	54	92
Total Number of operators (N1)	267	260	177
Total Number of operands (N2)	96	88	89
N i.e. (N1+N2)	363	348	266
n i.e. (n1+n2)	56	64	105

Table 5: Data Obtained During and After the Execution of the Three Algorithms

Parameters	GA	SA	SAGA Hybrid
Simulation Time (seconds)	19.73	56.16	17.67
Number of Courses Clashing	0	0	0
Number of Lecturers Double Booked	0	0	0
Program Size (KB)	20	16.5	6.5
Lines of code	500	256	194
Program Volume (V)	2108.07	2088.00	1785.99
Program Effort (E)	33729.12	17013.33	11230.38
Difficulty of Understanding the Program	0.06	0.12	0.16
Intelligent Content of the Program	131.75	256.25	284.03

Table 6: The Effect of Initial Temperature and Number of Generations on Simulation Time

Initial Temperature	Simulation Time of Different Generation				No of Courses Clashed	No of Lecturers Double Booked
	10	100	1000	10000	0	0
10	0.4856	3.7752	27.8462	270.531	0	0
100	0.3744	2.6052	27.9398	271.567	0	0
1000	0.4056	3.6192	25.241	283.048	0	0
10000	0.3744	3.0108	30.6542	360.9239	0	0
100000	0.3588	3.5412	29.7494	309.428	0	0
1000000	0.4056	3.4320	28.0022	281.847	0	0

Table 7: The Simulation Time of Some Initial Temperature and Various Cooling Rates at Constant Number of Generations (100) and Number of Population (10000)

Initial Temperature	Simulation Time of Different Cooling Rates (Seconds)			
	0.1	0.01	0.001	0.0001
10	3.4320	2.4648	3.3696	3.4164
20	3.9624	3.0576	3.2760	3.1200
30	3.3852	3.4008	2.7456	3.0108
40	2.6676	2.8080	3.8064	2.9172
50	3.5256	3.6660	3.2916	2.7924
60	3.1356	3.1356	2.8548	2.8392
70	2.9640	2.9640	3.1356	2.6988
80	2.7768	2.8860	2.8548	3.9000
90	3.0420	2.6364	2.9328	3.1356
100	2.8236	3.0888	3.7284	3.7596
150	3.7596	2.8704	2.6052	3.4944
180	3.1044	3.1512	3.6972	2.8548
200	2.6052	2.6208	2.9484	3.8688

Table 7: Contd.,				
210	3.5256	2.8080	3.5568	3.2136
220	2.9328	3.5568	**2.3712**	2.7456
230	3.7284	3.5100	3.3540	2.4804
250	3.4230	3.3852	2.4648	3.5100
300	3.7284	3.6036	2.7768	2.9484
350	4.3368	4.2432	3.2604	4.2900
400	3.9000	4.0716	2.5272	3.6036
450	3.5568	3.8688	3.5724	3.6504
500	3.5568	4.2276	3.9156	3.8844

Table 8: Simulation Time – Population Size and Number of Generation at Constant Initial Temperature (220) and Cooling Rate (0.001)

Number of Population	Simulation Time of Different Number of Generation (Seconds)				
	10	100	150	200	210
10000	0.3588	2.4336	4.1808	4.6160	4.9452
15000	0.3588	2.5584	4.5708	7.1136	6.5988
20000	0.3744	2.9328	4.6020	6.8796	6.5676
25000	0.4680	2.5272	4.5552	6.4272	6.6612
26000	0.3588	3.1512	4.0716	7.3944	5.9436
27000	0.4368	3.4320	3.6348	6.8952	7.2072
28000	0.4836	3.4320	4.7580	5.4288	7.0356
30000	0.3744	3.0420	4.1496	6.2400	5.7876
35000	0.4524	2.6988	3.7128	5.6004	5.7408
40000	0.4680	2.8704	5.2104	6.3960	6.5520
45000	0.4368	2.5428	4.8828	6.0216	6.8796
50000	0.3744	2.6208	4.4304	6.1464	6.5052

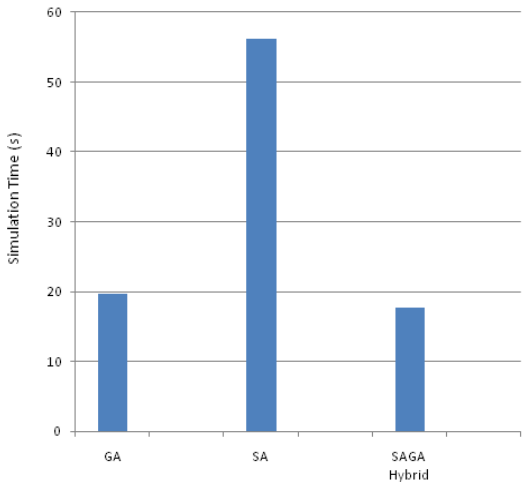

Figure 1: The Simulation Time of the Three Algorithms

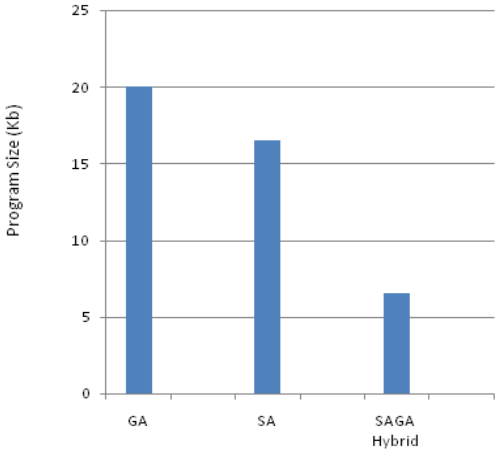

Figure 2: The Program Size of the Three Algorithms

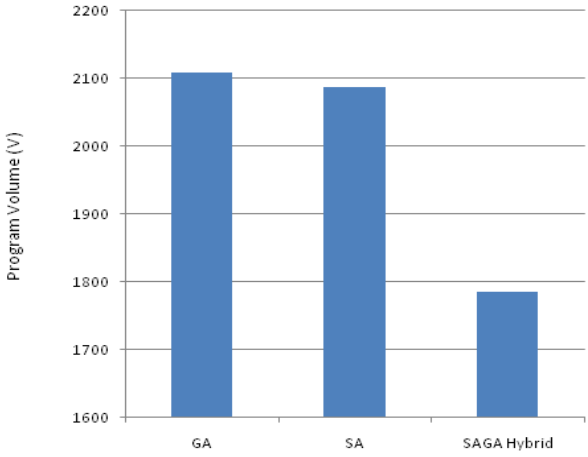

Figure 3: The Program Volume of the Three Algorithms

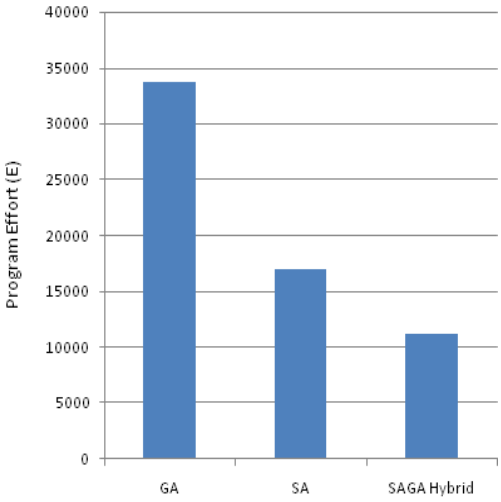

Figure 4: Program Effort of the Three Algorithms

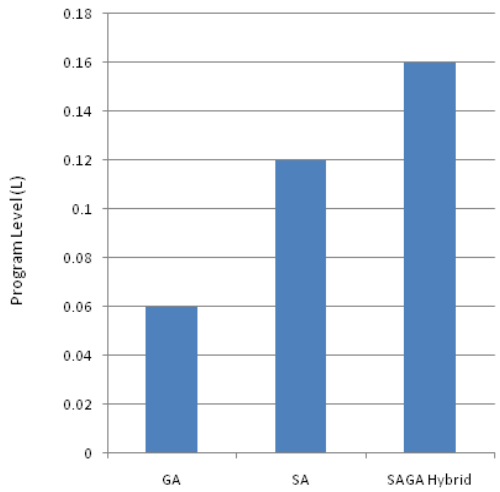

Figure 5: Program Level / Difficulty of Understanding the Programs

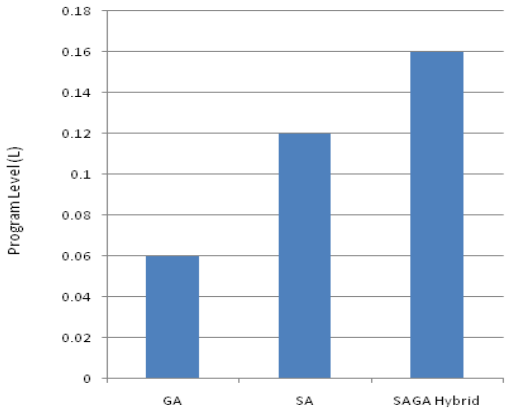

Figure 6: The Program Level (i.e. Difficulty of Understanding the Program) of the Three Algorithms

REFERENCES

1. Abdennadher S. and Marte M (2000).: *University course timetabling using constraint handling rules*, Journal of Applied Artificial Intelligence, 14(4): 311–326.
2. Abramson, David (1991): "Constructing school timetables using simulated annealing": sequentialand parallel algorithms. Management Science, 37(1) pp. 98–113.
3. Abramson, D. and Abela, J. (1992): "A parallel genetic algorithm for solving the school timetabling problem". In Proceedings of 15 Australian Computer Science Conference, Hobart, pp. 1–11
4. Al- Attar, A (1994), "A hybrid GA-heuristic search strategy", A white paper, AI Expert, USA
5. Andrea Schaerf, (1999), "A survey of automated timetabling, Artificial Intelligence Review", Kluwer Academic Publishers, 13(2):87- 127.
6. Burke, E.K, Elliman, D.G. and Weare, R.F. (1994), *"A University Timetabling System based on Graph Colouring and Constraint Manipulation"* Journal of Research on Computing in Education, 27(1): 1-18

7. Burke, E.K. and Ross, P., (1996), "*Practice and Theory of Automated Timetabling"*, volume 1153 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg.

8. De Werra, D. (1985):"*An introduction to Timetabling*", European Journal of Operations Research, 19, 151-162.

9. Elmohamed S., Coddington P., Fox G., (1998),"*A Comparison of Annealing Techniques for Academic Course Scheduling"*, Practice and Theory of Automated Timetabling II, Springer- Verlag, Vol. 1408, pp. 92-112.

10. Keshav P. Dahal, Kay Chen Tan, Peter I. Cowling, (2007), "Evolutionary Scheduling", Springer, Berlin Heidelberg Germany, Vol. 49.

11. Neufeld, G.A. and Tartar, J., (1974), "Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem", Communications of the ACM, 17:8, pp. 450-453.

12. Olabiyisi, S.O., Akanmu, T.A., Oyeleye, C.A., Sobayo, O.D. and Adelana, J.B, (2007), "Complexity Analysis of A New Edge-Adaptive Zooming Algorithm For Digital Images", Journal of Research in Physical Sciences, Vol. 3, Number 2 pg 67-71.

13. Olabiyisi, S.O., Omidiora, E.O. and Oyeleye, C.A., (2005), "Asymptotic Time Complexity Analysis For Two-Dimensional Object Inspection Using String matching", Science Focus, Nigeria. Vol. 10, Number 3 pg 83-87.

14. Omidiora, E.O., Olabiyisi, S.O., Arulogun, O.T., Oyeleye, C.A. and Adegbola, A.A., (2009), "A Prototype of An Access Control System For a Computer Laboratory Scheduling", AICTTRA 2009 Proceedings, Obafemi Awolowo University, Ile-Ife, Nigeria pg 114-120.

15. Oyeleye, C. Akinwale**,** Olabiyisi, S. Olatunde, Omidiora, E. Olusayo and Oladosu, J. Babalola. (2012). Performance evaluation of Simulated Annealing and genetic algorithm in solving examination timetabling problem.Scientific Research and Essays.Vol. 7(13).pp1727-1733.

16. Piola, R. (1994): "Evolutionary solutions to a highly constrained combinatorial problem", Proceedings of IEEE Conference on Evolutionary Computation (First World Congress on Computational Intelligence), Orlando, Florida 1:446–45.

17. Wren, A, (1996),"*Scheduling, timetabling and rostering - A special relationship?*. In *Practice and Theory of Automated Timetabling"*, volume 1153 of *Lecture*

18. *Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, pp. 46-75.

19. Zahra NajiAzimi (2005), **"**Hybrid heuristics for Examination Timetabling problem**",** Applied Mathematics and Computation, 163(15): 705-733.