Application of a parallel computational approach in the design methodology for the Course timetabling problem

Soria-Alcaraz Jorge A. · Carpio Martín · Puga Héctor · Sotelo-Figueroa Marco A.

the date of receipt and acceptance should be inserted later

Abstract The process of gathering enough experimental statistical data over a set of instances of the Course timetabling problem (CTTP) could take a lot of time to an interested researcher. There exist several parallel computing models capable to accelerating the execution process of metaheuristic algorithms. This paper explores the idea to use a parallel model in a metaheuristic algorithm over the Course Timetabling Problem in order to reduce the time that a investigator needs to collect enough data to make a proper conclusion. our parallel approach uses the Methodology of design model for CTTP. The methodology of design is a strategy applied before the execution of an algorithm for timetabling problem. This strategy has recently emerged and aims to generalize and provide a context-independent layer to different versions of the Course timetabling problem. Finally a well-know set of instances was tested with a parallel GA and a sequential GA in order to determine the advantages of the proposed approach for CTTP.

Keywords Methodology of design · Parallel computing · Genetic Algorithm · Cellular Genetic Algorithm

1 Introduction

The timetabling problem is one of the most difficult, common and diverse problems inside an university. This problem tries to assign several activities into *timeslots* to make a *timetabling*. The main objective of this problem is to obtain a timetabling with the minimum conflicts between assigned activities. [1]

There exist several university timetabling problems as described by Adriaen et. al [2] for example the *Faculty timetabling* tries to assign teachers to

León Institute of Technology, División de Estudios de Posgrado e Investigación, León Guanajuato, México E-mail: soajorgea@gmail.com,jmcarpio61@hotmail.com

subjects *Class-teacher timetabling* assigns subjects to a fixed group of students. *Classroom assignment* ensures that every pair teacher-subject have a classroom. *Examination timetabling* assigns events like final exams to a set of individual student and *Course timetabling* assigns subjects to individual students minimizing the conflicts (usually time-conflicts) between the assigned events. This paper is focused into the last timetabling problem type.

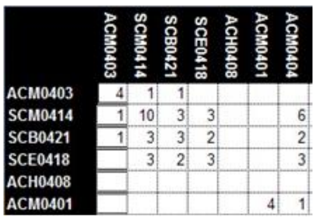
Like most timetabling problems, the *Course timetabling* is NP-Complete [3] [4]. The reason for this is attributed by the literature to a combinatorial explosion of possible events assigned into time slots, as well as the constraints that each university uses in its own course timetabling creation.

The course timetabling problem (CTTP) is also considered by Rodriguez y Quezada[5], like a "offline" problem meaning that the resolution of the course timetabling is not needed in real time i.e the final user of a solver for the CTTP can wait a reasonable time (several days or even a week) in order to get a "good" solution. However in the academic or experimental field the researcher needs to perform a high number of test with his/her proposed algorithm in order to accumulate enough statistical data to prove the quality of the proposed approach.

That is why the researcher of CTTP have an added challenge: in addition to create an algorithm that is at least comparable with the current state of the art, such algorithm needs to be reasonably fast in order to perform enough experiments and gather statistical evidence to finally make a conclusion. This paper explores the possibility by means of parallel metaheuristics and parallel computing to create fast algorithms for the CTTP with good performance over a well-know instances of the CTTP.

It is an undisputed fact that the CTTP problem usually differs greatly from one university to another. So the researcher has the risk that once he tuned his algorithm to a specific college and moving to testing in another university his approach could do not replicate desired characteristics or ,in the worst of cases, cannot be possible to obtain a solution applicable to reality. In this sense a new approach has emerged ,The Methodology of Design, this approach give us a generic methodology to resolve a widely set of CTTP problems by means of the application of a context-independent layer. [1] In this paper we use this Methodology of Design in order to build a parallel algorithm capable to: A) being applied to the ITC2002, ITC2007 instances and B) have a higher speed than its sequential counterpart making faster the process of experimentation and gathering data.

The paper is organized as follows. Section 2 presents a brief explication of the Methodology of Design, the parallel build-up for the course timetabling problem, the solution approach and its justification. Section 3 contains the experimental set-up, results, their analysis and discussion. Finally Section 4 include some conclusions and future work.



	ACM0403	SCM0414	SCB0421	SCE0418	ACH0408	ACM0401	ACM0404
ACM0403	4	1	1				
SCM0414	1	10	3	3			6
SCB0421	1	3	3	2			2
SCE0418		3	2	3			3
ACH0408							
ACM0401						4	1

Fig. 1 MMA Matrix

2 Solution Approach

2.1 Problem Definition

A clear and concise definition of the CTTP is given by Conant-Pablos [6]: A set of events(courses or subjects) $E = e_1, e_2, \ldots, e_n$ is the basic element of a CTTP. Also there are a set of periods of time $T = t_1, t_2, \ldots, t_s$, a set of places (classrooms) $P = p_1, p_2, \ldots, p_m$, and a set of agents (students registered in the courses) $A = a_1, a_2, \ldots, a_o$. Each member $e \in E$ is a unique event that requires the assignment of a period of time $t \in T$, a place $p \in P$ and a set of students $S \subseteq A$, so that an assignment is a quadruple(e, t, p, S). A timetabling solution is a complete set of n assignment, one for each event, which satisfies the set of hard constraints defined usually by each university of college. This problem is documented to be at least as a NP-complete problem [3] [4].

2.2 Methodology of Design for the Course Timetabling Problem

In the literature it can be seen that there is a problem with the diversity of course timetabling instances due different university policies. This situation directly impacts in the reproducibility and comparison of course timetabling algorithms[7]. The state of art indicates some strategies to avoid this problem. For example, a more formal problem formulation [7] as well as the construction of benchmark instances [8]. These schemes are useful for a deeper understanding of the university timetabling complexity, but the portability and the reproducibility of a timetabling solver in another educational institution is still in discussion[1]. In this sense, we use a context-independent layer for the course timetabling resolution process. This new layer integrates timetabling constraints into three basic structures *MMA matrix, LPH list and LPA list.*

MMA matrix: This matrix contains the number of students in conflict between subjects i.e. the number of conflicts if two subjects are assigned in the same timeslots. An example of this matrix can be seen in the Figure 1.

LPH list :This structure have in its rows the subjects offered. In its columns have the offered timeslots, So this list give us information about the allowed timeslots per subject. one example of this list can be seen on 1.

Table 1 LPH list

	Day 1	Day 2
e_1	$< t_3 >$	$< t_2 >$
e_2	$< t_2 >$	$< t_2$ or $t_1 >$

LPA list :This list shows in its rows each event and the classrooms available to be assigned to each event without conflict.

Table 2 LPA list

event	*Classrooms*
e_1	$< p_4, p_{l1}, p_{c2} >$
e_2	$< p_{lab}, p_{c2} >$
e_3	$< p_6, p_{b2}, p_{b3}, p_{b4} >$
e_4	$< p_{lab}, p_{l2} >$
$\vdots$	$\vdots$
e_{530}	$< p_{d7} >$

Once we obtain these structures by means of the natural/original inputs of our CTTP problem, we ensures *by design* the non-existence of violations by the selection of any values shown in LPH and LPA. Our problem now is to deal with students conflicts only. We work with these conflicts by means of the next minimization function:

$$min(FA) = \sum_{i=1}^{k} FA_{V_i} \tag{1}$$

$$FA_{V_j} = \sum_{s=1}^{(M_{V_j})-1} \sum_{l=1}^{M_{V_i}-s} (A_{j,s} \wedge A_{j,s+l}) \tag{2}$$

Where: FA= Student conflicts of current timetabling. V_i= Student conflicts from "Vector" i of the current Timetabling. $A_{j,s} \wedge A_{j,s+l}$= students that simultaneously demand subjects s and $s+1$ inside the "Vector" j. A means a student that demands subject s in a timetabling j.

Now we can talk about the most important element in the design methodology: the concept of vector. This vector is a binary representation of an event.[9][1] We can construct them as seen on table 3 where each v_i is a vector who represents event e_i.

The vectors can be easily added and subtracted allowing them to form *sets*. the symbols used for these sets of vectors are $V_A, V_B, \ldots$ and so on. One characteristic is that the number of vectors sets is related with the number of timeslots offered by the current timetabling. The main idea about vectors is to have a space where we can work with events without assigned them yet to a fixed timeslot. This independent layer of context generalizes in some way the solution process of the CTTP problem.

Our problem now is to construct a fixed number of vectors sets (usually the cardinality of timeslots set) in order to obtain zero conflict on MMA, LPH and LPA. It is precisely for the vector sets construction that we build a parallel metaheuristic algorithm, but once we have it, if other CTTP problem can be expressed by means of the Methodology of design then we expect work with it without any modification in the algorithm.

Table 3 Vector Construction

Events	e_1	e_2	$\dots$	e_{a-1}	e_a
v_1	1	0	$\dots$	0	0
v_2	0	1	$\dots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$	$\vdots$
v_{a-1}	0	0	$\dots$	1	0
v_a	0	0	$\dots$	0	1

2.3 Parallel Computing and Cellular Genetic Algorithms

The main objective of parallel computing is to execute code concurrently on different processors i.e in the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem for example: To be run using multiple CPUs, To solve a problem broken into discrete parts that can be executed concurrently and instructions from an algorithm executed simultaneously on different CPUs [10].

Also we use a genetic algorithm. A genetic algorithm (GA) is a search heuristic that mimics the process of natural evolution.[11] This heuristic is routinely used to generate useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover [11].

With these two concepts we built a Cellular Genetic Algorithm (cGA). the cellular genetic algorithm was initially designed for working in massive parallel machines composed of many processors executing simultaneously the same instructions on different data.[12]. The first cGA model know was proposed by Robertson in 1987 [13] implemented on a CM1 computer. It was a model were all steps of GA algorithm(selection, replacement, recombination and mutation) were executed in parallel.[12]. This approach has shown great execution speed as well as better fitness performance against sequential or canonical GA.

Many researchers still think about the equivalence between cGA and massive parallel machines. Today with technologies like Java Threads or CUDA cores, we do not need a massive parallel computer in order to build and execute a cGA.

The cGA model simulates the natural evolution from the point of view of the individual. The essential idea of this model is to provide the population of a special structure defined as a connected graph, where each vertex is a common GA individual or *Cell* that is only allowed to communicate with its nearest neighbours. Particularly, individuals are conceptually in a toroidal mesh and are only allowed to recombine with close individuals.[12] An example of this type of interaction can be seen on figure 2.

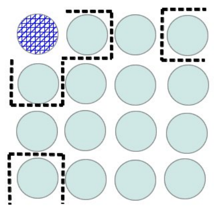


Fig. 2 Simple toroidal population and interaction

The neighborhood of a specific individual on the cellular grid is overlapped by its neighbors. This ensures that good traits and characteristics can travel throughout the grid. In cGAs the reproductive cycle is execute inside the neighborhood of each individual and, generally, consists in selecting among its neighbors with a certain criterion (Tournament selection or Roulette wheel) a parent with witch the cell can apply any recombination operators and finally update its own genetic material.

The mutation is simply performed by selecting randomly one cell and minimum change its genetic material. This reproduction cycle can be execute in parallel, executing each cell in a different java thread or CUDA core. A pseudo-code of canonical cGA proposed by Alba Et.al [12] can be seen on algorithm 1.

Algorithm 1 Pseudo-code of a canonical cGA

```
1: $procEvolve(cga)$
2: $GenerateInitialPopulation(cga.pop)$
3: $Evaluation(cga.pop)$
4: while $!StopCondition()$ do
5:     for $individual \leftarrow 1$ to $cga.popsize$ do
6:         $neighbors \leftarrow CalculateNeighborhood(cga, position(indicidual))$;
7:         $parents \leftarrow Selection(neighbors)$;
8:         $offspring \leftarrow Recombination(cga.Pc, parents)$;
9:         $offspring \leftarrow Mutation(PM)$;
10:        $Replacement(position(individual), auxiliaryPop, offspring)$;
11:    end for
12:    $cga.pop \leftarrow auxiliaryPop$;
13: end while
14: return $Best(cga.pop)$
```

Table 4 Cell Codification

	e_1	e_2	$\dots$	e_l
$cell_1$	V_B	V_D	$\dots$	V_B
$cell_2$	V_A	V_B	$\dots$	V_C
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$cell_n$	V_D	V_A	$\dots$	V_C

From algorithm 1 we seen that the cGA algorithm not differ greatly from a sequential GA. The differences of cGA approach are simply the Grid/Toroidal interaction and its parallelism.

2.4 Combining methodology of Design with cGA for the CTTP problem

As seen in previously sections the definition of the cGA have similar operators and parameters as a Sequential GA(sGA). In this section we define the codification, operators and parameters used as well as several details of our grid configuration.

We use the Methodology of Design approach shown in section 2.2 in order to have a CTTP solver to test over different CTTP type instances. We have for each instance 3 list MMA,LPH and LPA. The construction of each of these lists is beyond the scope and purpose of this article. These list ensures by design that every 3-tuple (e, t, p) will be an allowed selection so it can be applied to reality. The main optimization exercise is to minimize the conflict of students $S \subseteq A$ by means of the permutation of the events/vectors into timeslots/Vector-sets and the MMA matrix.

The Codification of each Cell or individual is an array of Integer values each integer represent an ID of a Vector set and its size is equal that the cardinality of the event set. The number of Vector sets is defined by the cardinality of the desire timeslots set. So basically from table 4 we can read for the Cell 1: *event* 1 is assigned into $Vector - set$ B, *event* 2 is assigned into $Vector - set$ D... and so on.

The operators used in each cell are simple. For Selection we use Roulette wheel so every neighbour may have the chance to reproduce with the selected cell but better solutions have more probability to do it. Given the integer representation the recombination operator is single point crossover, in each generation a random crossover point is selected so the genetic material of parents is interchanged from it. The Mutation operator is done at the final of each generation, where we randomly selects a cell and a single integer to change. Also we implement a form of Elitism: in each execution the best cell from the grid will not be modify it all i.e the best cell can interact and update genetic information to its neighbors but no one neighbor can change its own genetic material.

The Parameters used are: for stop criteria we use a fixed number of functions points (a function point means a single decoding of the information of the cell and its execution in the fitness function), for recombination and mutation we use a percentage fixed by the user.

For the grid configuration we divide our population (cells) into several islands. At the beginning of each iteration, all the islands send the cells of their first/last column/row to their neihgbor islands. After receiving the individuals from the neighbors, a sequential cGA is executed in each island/subpopulation (figure). This approach has been documented by Alba et al [12] with good results. Finally the neighborhood model used by each cell and island is the NEWS model (North, West, East, South) similar to figure 3. Each island is executed in a JAVA Thread and synchronously waits for all the other islands ends a generation to interchange cells to continue.

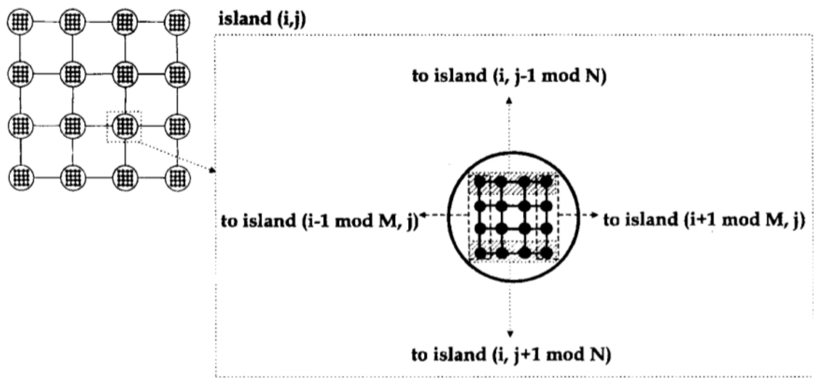


Fig. 3 Grid configuration model used. From Alba et. al

3 Experiments and Results

Once we have cGA proposed algorithm we can do several experiments in order to find the speed-up or performance difference between our cGA and sGA. In this section we will explain each experiment as well as the characteristics of the used benchmark.

3.1 **Instances Used**

We chose a set of well-known instances for the TTP such as ITC2002, ITC2007 from PATAT. The timetabling problem instances ITC 2002 has been designed by Ben Paechter for the Metaheuristics Network.[15] It is a reduction of a typical university course timetabling problem. It consists of a set of events to be scheduled in 45 timeslots (5 days of 9 hours each), a set of rooms in which events can take place, a set of students who attend the events, and a set of features satisfied by rooms and required by events. Each student attends a number of events and each room has a size.

The ITC2007 instances has been uses as benchmark for the the second International Timetabling Competition contest sponsored by PATAT and WATT. These instances are similar to the ITC2002 ones but, these instances have 3 more constraints: a subset of valid timeslots for subject, an ordering between subjects and a preferred set of timeslots per subject.

3.2 **Experimental design**

According to Alba et.al [12] the recommendations used to compare parallel-sequential algorithm we will use two algorithms: Our parallel proposed approach (cGA) defined in section 2.4 and its sequential counterpart(sGA). For the cGA we implement 16 isles divided into a 4x4 grid, each island have a inner grid of 16 Cells in a 2D array of 4x4 cells similar to figure 3 making 256 individuals/cells in total with an elitism of 16 cells per generation(one cell per island). For the sGA we can say that it is a normal GA without any special modifications. Its recombination, selection and mutation operators are the same that the proposed cGA. Its Elitism is implemented by selecting the best 16 individuals from a population of 256 elements the same of our cGA.

We use a the *weak speedup* metric proposed by Alba et.al [12] because it compares the parallel algorithm developed by a researcher against his own sequential version. For the experiment we execute 100 independent test in each instance from ITC2002 and ITC2007 with cGA and sGA. Each algorithm executes exactly 1000 functions points before stop, the results are shown on table 5.

Our tables 5 and 6 shown the results for our test over ITC2002 and ITC2007 instances. Our fitness function evaluates the number of conflicts (student conflicts) on the timetabling built by the algorithms, so a less value of fitness means a better solution for the CTTP.We need to say that these evaluations was made just only considering hard constraints for ITC2002 and ITC2007. The column $Best\_fit$ shows the best fitness reached by the Algorithm (sGA or cGA) in our experiments. Avg_fit shows the average fitness from our algorithms over 100 independent test. std_devf shows the standard deviation value for the fitness obtained from our series of test. *Avg.time* shows the average time in seconds needed for the algorithm to finish one single test. *std_devt* shows the standard deviation value of the time used. Finally the *speedup* column shows the weak speed-up metric proposed by Alba et.al [12].

3.3 Analysis of results

As it can be seen on table 5 we achieve a better speed in the cGA against the sGA, this is not a surprise because we utilize JAVA Threads so each island

Table 5 Results experiment ITC2002

Instance		Best_fit	Avg_fit	Std_devf	Avg.time	std_devt	Speedup
ITC2002-1	sGA	281	324.3	18.8	12.59	2.96	
	cGA	149	188	14.9	2.35	0.09	5.35
ITC2002-2	sGA	262	305.5	18.2	7.97	2.50	
	cGA	146	179	14.0	2.97	0.15	2.68
ITC2002-3	sGA	296	330.4	15.3	10.96	2.58	
	cGA	165	202.7	13.0	2.95	0.17	3.71
ITC2002-4	sGA	429	485.5	23.9	8.30	2.25	
	cGA	255	304	22.9	3.01	0.10	2.75
ITC2002-5	sGA	424	480.18	26.57	6.82	2.09	
	cGA	246	293.3	21.4	2.78	0.13	2.45
ITC2002-6	sGA	427	481.8	26.56	7.09	2.24	
	cGA	247	294.8	21.06	2.65	0.15	2.67
ITC2002-7	sGA	419	503.4	30.55	7.21	2.23	
	cGA	238	287.9	24.9	2.63	0.16	2.74
ITC2002-8	sGA	311	371.94	25.14	8.25	2.51	
	cGA	168	210.2	18.65	3.01	0.19	2.74
ITC2002-9	sGA	299	346.8	20.54	12.99	2.82	
	cGA	186	207.9	16.56	3.2	0.20	4.05
ITC2002-10	sGA	285	335.15	21.24	9.67	2.74	
	cGA	176	201.9	14.2	3.0	0.19	3.22
ITC2002-11	sGA	304	350.4	19.4	12.12	2.43	
	cGA	174	208.2	14.56	3.12	0.32	3.88
ITC2002-12	sGA	268	312.1	19.15	10.03	3.12	
	cGA	148	188.5	14.25	3.40	0.52	2.95
ITC2002-13	sGA	343	396.44	24.26	9.51	1.42	
	cGA	186	231.6	19.54	2.30	0.32	4.13
ITC2002-14	sGA	455	520.15	28.17	7.79	0.877	
	cGA	251	313.3	23.20	3.21	0.19	2.42
ITC2002-15	sGA	362	449.5	28.65	8.06	0.8	
	cGA	209	261.4	22.44	3.10	0.22	2.6
ITC2002-16	sGA	304	369.67	20.44	10.14	1.69	
	cGA	191	223.6	18.22	3.70	0.36	2.74
ITC2002-17	sGA	396	468.9	29.22	7.33	2.46	
	cGA	231	289.4	22.19	3.18	0.05	2.30
ITC2002-18	sGA	250	307.14	20.3	12.19	2.55	
	cGA	148	181.5	14.98	3.59	0.06	3.39
ITC2002-19	sGA	408	497.1	28.95	9.45	2.97	
	cGA	224	297.9	23.0	3.60	0.13	2.28
ITC2002-20	sGA	380	449.14	26.35	8.35	2.47	
	cGA	224	266.0	21.54	3.18	0.16	2.62

or sub population is working in a parallel way. However we obtained not only a better speed but a good performance as well, as seen on the results table, our cGA presents a lower fitness value and a lower standard deviation value for both the time and the objective function. This result can be explained because the elitism used in the cellular model. This approach conserves several cell/individual with different genetic value, then in the phase of interchange this genetic material travels over the grid. In the sGA cells preserved by the elitism have similar genetic material. An snapshot of the Celular Grid as well as its performance graph can be seen on fig 4 and 5.

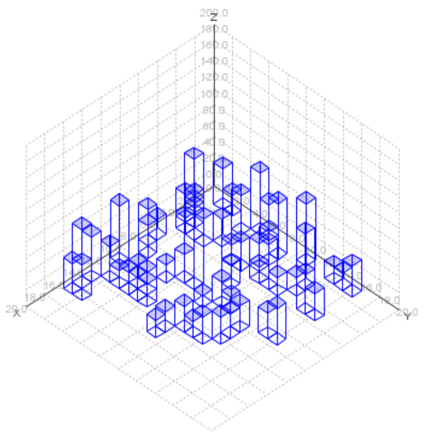


Fig. 4 Snapshot of cGA grid after a test

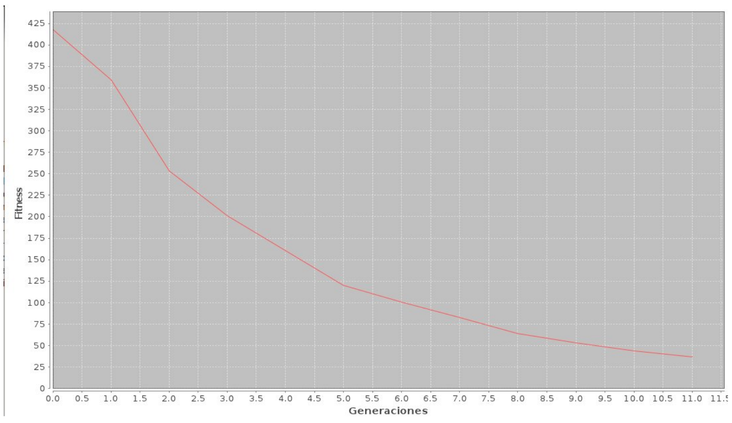


Fig. 5 Performance of cGA over a test

Table 6 Results experiment ITC2007

Instance		Best_fit	Avg_fit	Std_devf	Avg.time	std_devt	Speedup
ITC2007-1	sGA	1253	1362.4	55.0	11.1	3.0	
	cGA	881	975.3	45.96	3.92	1.05	2.83
ITC2007-2	sGA	1251	1388.5	56.87	11.21	3.06	
	cGA	892	999.0	42.78	4.57	0.37	2.45
ITC2007-3	sGA	434	556.8	57.80	4.79	1.11	
	cGA	215	286.75	27.0	2.66	0.28	1.80
ITC2007-4	sGA	527	619.2	44.0	4.71	1.08	
	cGA	275	342.8	30.55	2.56	0.29	1.83
ITC2007-5	sGA	698	805.3	35.70	9.73	3.04	
	cGA	479	564.4	32.28	4.68	0.42	2.07
ITC2007-6	sGA	723	794.81	36.72	10.73	3.41	
	cGA	480	552.18	33.15	3.8	0.39	2.82
ITC2007-7	sGA	265	334.9	28.96	4.5	1.27	
	cGA	154	187.8	15.5	2.46	0.39	1.82
ITC2007-8	sGA	287	375	36.14	5.75	0.93	
	cGA	147	196.3	18.72	2.5	0.42	2.3
ITC2007-9	sGA	1190	1403.1	69.77	10.52	3.26	
	cGA	860	979.23	52.67	4.29	0.70	2.45
ITC2007-10	sGA	1256	1400.3	53.58	11.05	3.41	
	cGA	899	1011.6	47.38	3.83	0.45	2.88
ITC2007-11	sGA	488	612.84	58.64	4.55	0.97	
	cGA	235	319.21	32.20	2.68	0.14	1.69
ITC2007-12	sGA	433	588.11	65.15	4.83	1.24	
	cGA	249	317.74	28.55	1.56	0.16	3.09
ITC2007-13	sGA	751	843.75	34.24	8.26	2.66	
	cGA	521	590.17	31.19	2.75	0.20	3.00
ITC2007-14	sGA	724	813.11	36.41	8.20	2.62	
	cGA	512	572.86	26.77	2.28	0.08	3.59
ITC2007-15	sGA	231	300.96	30.23	4.67	1.04	
	cGA	131	154.0	12.20	1.43	0.06	3.26
ITC2007-16	sGA	237	326.85	34.47	4.59	0.96	
	cGA	113	147.21	16.82	1.41	0.06	3.255
ITC2007-17	sGA	286	432.15	69.81	3.23	0.21	
	cGA	46	160.15	35.05	1.10	0.05	2.93
ITC2007-18	sGA	865	1000.59	56.14	4.47	0.95	
	cGA	512	635.17	38.87	1.47	0.04	3.04
ITC2007-19	sGA	686	814.96	58.10	7.14	2.24	
	cGA	426	482.5	32.52	1.88	0.05	3.79
ITC2007-20	sGA	738	879.07	59.34	10.53	2.86	
	cGA	430	497.52	38.96	2.38	0.05	4.42
ITC2007-21	sGA	761	847.17	28.99	12.97	4.34	
	cGA	516	614.27	26.82	3.03	0.07	4.28
ITC2007-22	sGA	1400	1556.15	59.93	16.87	4.98	
	cGA	1084	1185.74	75.48	3.84	0.08	4.39
ITC2007-23	sGA	2595	2882.6	114.88	7.93	2.42	
	cGA	1902	2137.85	88.12	2.36	0.07	3.36
ITC2007-24	sGA	757	886.48	52.31	8.78	2.61	
	cGA	453	527.9	35.12	2.41	0.06	3.64

4 Conclusions and future work

This paper has presented a comparison between a parallel computing model for a genetic algorithm and a sequential genetic algorithm for the context of CTTP problem. The cGA proposed aims to help the researcher of CTTP to obtain good solutions in a relative short lapse of time against sequential Genetic Algorithm. The cGA proposed utilizes a toroidal grid configuration that ensures the interchange of good genetic material over the whole population. The proposed approach utilizes the Methodology of Design model to generalize the process of CTTP solution by means of generic structures, this model ensures that if any other type of CTTP can be produce the same structures the proposed algorithm will be capable to work over it with similar performance. This paper has explored the use of parallel computing for a well known set of instances for the CTTP. The cGA has show a better performance against a sequential GA in a better time allowing the researcher to accelerate the process of gathering statistical data.

For future work we propose to apply this approach to Unitime.org instances. Also we Propose to apply this model over not only Java Threads but CUDA Nvidia cores in order to perform a faster experimentation.

Acknowledgement

Authors thanks the support received from the Consejo Nacional de Ciencia y Tecnologia (CONACYT) México.

References

1. J.A. Soria-Alcaraz, Diseño de horarios con respecto al alumno mediante tcnicas de cmputo evolutivo. Master's thesis, Instituto Tecnologico de Len (2010)
2. M. Adriaen, P. Causmaecker, Proccedings PATAT **1**, 330 (2006)
3. T.B. Cooper, J.H. Kingston, The compexity of timetable construction problems. Ph.D. thesis, The University of Sydney (1995)
4. R.J. Willemen, School timetable constructrion: Algorithms and complexity. Ph.D. thesis, Institutefor Programming research and Algorithms (2002)
5. R. Martinez, Q. Aguilera, COMCEV **1**, 169 (2005)
6. S.E. Conant-Pablos, D.J. Magaa-Lozano, H. Terashima-Marin, MICAI Mexican international conference on artificial intelligence **1**, 408 (2009)
7. A. Schaerf, L.D. Gaspero, Practice and Theory of Automated Timetabling, PATAT, Springer-Verlag Berlin Heidelberg, LNCS 3867 **1**, 40 (2007)
8. R. Lewis, Metaheuristics for university course timetabling. Ph.D. thesis, University of Notthingham. (August 2006)
9. J.A. Soria-Alcaraz, M. Carpio, H. Puga, Dcima Primera Reunin de Otoo de Potencia, Electrnica y Computacin del IEEE, XI ROPEC, Morelia **1**, 24 (2009)
10. B. Barney. Introduction to parallel computing. URL https://computing.llnl.gov/tutorials/parallel_comp/
11. J. Holland, The University of Michigan Press, Ann Harbor (1975)
12. E. Alba, B. Dorronsoro, Cellular Genetic Algorithms **1**, Springer Science+Business Media, LLC (2008)
13. G. Robertson, in *In Proc. of the Second International Conference on Genetic Algorithms(ICGA)* (1987), pp. 140–147

14. E. Alba, B. Dorronsoro, Cellular Genetic Algorithms Springer Science+Business Media, LLC (2008)
15. B. Paechter. The course timetable problem (2002). URL http://www.metaheuristics.net/index.php?main=4&sub=44