Evolution of Constraint Satisfaction Strategies in Examination Timetabling

Hugo Terashima-Marín
ITESM-Centro de Inteligencia Artificial
Sucursal de Correos J
Monterrey, N.L. C.P. 64849 México
terashim@campus.mty.itesm.mx

Peter Ross
Division of Informatics,
The University of Edinburgh
80 South Bridge,
Edinburgh EH1 1HN (UK)
peter@dai.ed.ac.uk

Manuel Valenzuela-Rendón
ITESM-Centro de Inteligencia Artificial
Sucursal de Correos J
Monterrey, N.L. C.P. 64849 México
mvalenzu@campus.mty.itesm.mx

Abstract

This paper describes an investigation of solving Examination Timetabling Problems (ETTPs) with Genetic Algorithms (GAs) using a *non-direct* chromosome representation based on evolving the configuration of Constraint Satisfaction methods. There are two aims. The first is to circumvent the problems posed by a *direct* chromosome representation for the ETTP that consists of an array of events in which each value represents the timeslot which the corresponding event is assigned to. The second is to show that the adaptation of particular features in both the instance of the problem to be solved and the strategies used to solve it provides encouraging results for real ETTPs. There is much scope for investigating such approaches further, not only for the ETTP, but also for other related scheduling problems.

1 INTRODUCTION

Recent investigations have shown that Genetic Algorithms (GAs) can be used for solving the Examination Timetabling Problem (ETTP) [AA91, PCLP94, PCNL96, BNW96, Erg96, CFM92, CRL94]. However, many of the investigations in the literature have failed to provide more general insights as to how good GAs are for solving this sort of problem since their main aim is usually the solution of a particular instance of a real timetabling task rather than a general study. Based on an extensive investigation carried out on this area by members of the group of Evolutionary Computation at the Department of AI at Edinburgh University, from both practical and academic points of view, the *direct* representation or encoding of the problem has been found to be restrictive even though it has been used to solve many real-world problems. The basic objective in the ETTP is to assign exams to a limited number of timeslots so that no student has to be in two places at once (often called a *clash* or *edge* constraint). Additional constraints such as seating capacity may also be considered. If the ETTP with only *edge* constraints is examined, it can be easily mapped into a Graph-Colouring Problem (GCP) by considering the exams as nodes, the timeslots as colours, and the constraints as edges between nodes. The goal on the GCP is to colour the nodes in the graph in such a way that no two adjacent nodes have the same colour.

A *direct* chromosome representation for the examination timetabling problem consists of an array of integers indexed by event in which each value represents the timeslot for the corresponding event, while in the *non-direct* representation used in the experiments presented in this paper, an array encodes instructions and parameters for guiding a search algorithm that builds the timetable. In other words, the chromosome now represents the way a timetable is constructed rather than the timetable itself.

There are two reasons for using such a *non-direct* encoding. One is to avoid the known limitations of the *direct* encoding, outlined below, the other is to explore new approaches that have been successful in related domains such as manufacturing, neural networks and scheduling [SE96, Kit90, TGMS94, Smi85] and graph-colouring [FF96, Dav91].

Difficulties with the *direct* representation can be illustrated by a graph-colouring example (a *pyramidal* graph) described by Ross et al. [RCH97] with the following features: there are C cliques in the problem each of size N. Cliques 1 and 2 overlap by $(N - M)$ nodes where $M < N$; cliques 2 and 3 overlap in M nodes; cliques 3 and 4 overlap in $(N - M)$ nodes; and so on, and finally cliques C and 1 overlap in M nodes. A problem with $C = 20$, $N = 4$ and $M = 1$ is shown in Figure 1 along with a suboptimal solution when 4 colours are used. The violated constraints are shown in bold. When two adjacent cliques overlap in $(N-1)$ nodes, i.e. 3 and 4, the colours of these nodes determine the colour for the single node between cliques 2 and 3 and between cliques 4 and 5, for example. Hence, for any optimal solution it is necessary that all of the *link* nodes have the same colour. However, a GA that uses a direct encoding and penalty-based fitness is easily misled, because solving any clique produces a significant drop in penalty. The need to co-ordinate the colouring of each clique so that all *link* nodes are the same colour does not become apparent to the GA, so to speak, until too late. The GA cannot backtrack out of this trouble. The point is illustrated in the example shown in Figure 1, observe that if both violated constraints are to be fixed, at least eight repairs are necessary to find an optimal solution. The research by Ross et al. [RCH97] also discusses a form of adaptive mutation that helps somewhat, but observes that the main problem is still the choice of representation.

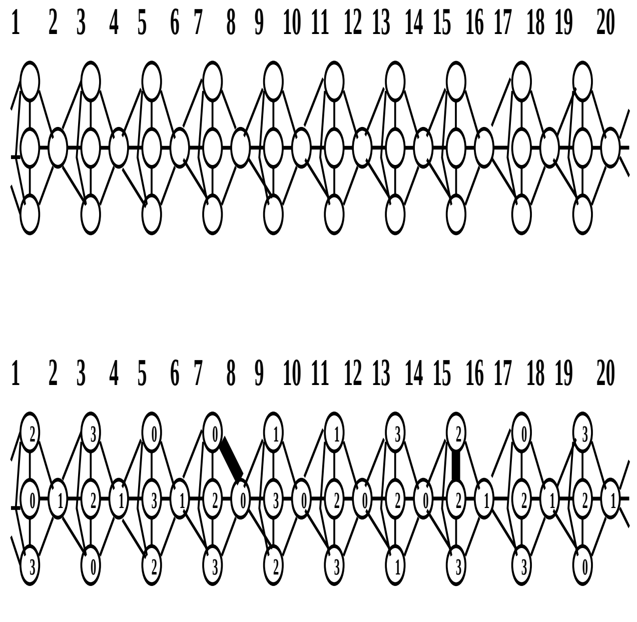

Figure 1: An incomplete solution for a pyramidal graph with C=20, N=4 and M=1.

Below, Section 2 presents some modifications to Brelaz's well-known DSATUR Algorithm to carry out this investigation, Section 3 describes the test problems and their features, and Section 4 contains experiments and results. Section 5 includes the definition of the process of evolving strategies and heuristics, related experiments and results and the comparison against results from the modified Brelaz Algorithm. Section 6 provides concluding discussion.

2 THE MODIFIED BRELAZ ALGORITHM

The Brelaz DSATUR algorithm [Bre79] is a graph-colouring algorithm which orders nodes according to the *colour degree* (also called *saturation degree*), that is, the number of different colours used among already-coloured adjacent nodes. The idea is that nodes of highest colour-degree should be coloured first since they have the least number of choices available. Ties are broken at random. Ross et al. [RCH97] show that the Brelaz method with backtracking in combination with other heuristics can be used to handle exam timetable problems with other kinds of constraints too, in particular *near-clash* constraints in which exams with an edge constraint and assigned to different slots should be spaced out rather than adjacent in the timetable, and *capacity* constraints which limit the number of available seats in any timeslot.

Certain heuristics can be used to handle the three kinds of constraints at the same time. For example, the following heuristics may be used for managing the *edge* constraint:

1. Order the events by *colour degree* and tie break by the number of arcs.
2. Order the events by *colour degree* and tie break by the number of instances of arcs.

Each student sitting two particular exams creates an instance of an arc between those two exams (nodes), and if at least one instance of an arc exists between two exams then only one arc (edge) is created between them.

After applying either of the two heuristics described above, and assuming that there are three slots per day, the near-clash constraint can be handled by any of the following heuristics which define the way in which the timeslot can be looked for:

1. By considering timeslots in the order: $1, 4, 7, ...3, 6, 8, ...2, 5, 8, ...$
2. By considering timeslots in the order: $1, 3, 4, 6, 7, 9, ...2, 5, 8, ...$
3. By considering timeslots in the order: $1, n, 3, n-2, ...2, 5, 8, ...$
4. By random selection.

For example, the first heuristic handles the near-clash constraint by looking for the first feasible slot starting on the first slot of each day. If this fails, then the algorithm looks into the third slot of each day, and finally if needed, it considers the middle slot of each day. If no suitable slot is found at the end of this process, then the algorithm backtracks. This tries to keep the most constrained events apart, which should result in fewer *near-clashes*. During this process for handling both the edge and the near-clash constraint types, the *capacity* constraint can also be applied. A slot is rejected if putting an exam there would violate the seating capacity.

3 SET OF TEST PROBLEMS

The experiments described in this paper were carried out using a test suite containing a collection of real-life ETTPs from various universities and which were slightly modified and adjusted in their features to suit the particular needs of this investigation. This set, herein called the Toronto Set, has been collected by Mike Carter and is available from `ftp://ie.utoronto.ca/mwc/testprob/`. Some results on these problems have been published, for example in the work by Carter et al. [CLL96]. Table 1 summarises the main features on these problems. The column *Max Ex Size* indicates the size of the largest exam, that is, the size of that exam with most students registered to sit in it. The column headed *Edges* gives the number of edges in the graph of the problem, where each exam is a node, each edge represents the fact that those two nodes cannot be in the same timeslot, and edges are not duplicated. The column headed *Slots* indicates the number of timeslots available for that problem and *Seats* represents the seating capacity for any timeslot. Note that certain information for some problems is not available so that, for some experiments the missing detail was invented based on other parameters.

4 INITIAL EXPERIMENTS AND RESULTS

A set of experiments were conducted for obtaining the overall performance of the modified Brelaz algorithm with the Toronto set. Several assumptions were made to complete the information on the problems in the real set. The capacity (number of seats per timeslot) for those problems lacking this figure was established by estimating the information based on experience and other parameters such as the size of large exams, information in similar problems, etc. The number of slots was determined as follows: keep the same number for those problems for which the number of slots and seating capacity exist in the original definition (CARF92, UTAS91 and CARS91 in Table 1) and use the number of slots produced by the modified Brelaz Algorithm using heuristic 2 for handling the clash constraint and heuristic 1 for handling the near-clash constraint. The same combination also helped to establish the number of slots used in problems TRES92 and KFUS93 since those numbers in the original definition were found to be inappropriate. This combination had produced the best results in preliminary experiments for problems with only edge and near-clash constraints. Results on the experimentation for the completed set are shown in Table 2. The columns numbered 1 through 4 represent the four different heuristics integrated in the modified Brelaz algorithm for handling the near-clash constraint, each corresponding to those explained in the previous section. Moreover, the Brelaz algorithm was run with heuristic 2 for the clash constraints, and with capacity constraints in force as well.

Results show that all heuristics, except number 4, produce the best result for at least one problem. This

Table 1: Carter's real-life Exam Timetable Problems (Toronto Set).

Problem	Exams	Students	Max Ex Size	Edges	Slots	Seats
HECS92	81	2823	634	1363	18	
STAF83	139	611	237	1381		
YORF83	181	941	175	4706		
UTES92	184	2750	482	1430		
EARF83	190	1125	232	4793		
TRES92	261	4360	407	6131	35	655
LSEF91	381	2726	382	4531		
KFUS93	461	5349	1280	5893	20	1955
RYES93	486	11483	943	8872		
CARF92	543	18419	1566	20305	40	2000
UTAS92	622	21266	1314	24249	38	2800
CARS91	682	16926	1385	29814	51	1550

Table 2: *Brelaz* algorithm on Carter's real-life exam timetable problems with *edge*, *near-clash* and *capacity* constraints using various heuristics.

Problem	Sls	Max ExSz	Seats	**Heuristics** 1	2	3	4
HECS92	21	634	1250	0/318/0	**0/302/0**	0/322/0	0/1112/0
STAF83	15	237	600	**0/1338/0**	0/1348/0	0/1450/0	0/2418/0
YORF83	21	175	500	0/790/0	0/865/0	**0/783/0**	0/1171/0
UTES92	12	482	1250	**0/816/0**	0/870/0	0/1593/0	0/2643/0
EARF83	24	232	700	**0/880/0**	0/933/0	0/946/0	0/1448/0
TRES92	27	407	655	**0/613/0**	0/645/0	0/716/0	0/1239/0
LSEF91	21	382	900	0/421/0	0/428/0	**0/302/0**	0/1309/0
KFUS93	24	1280	1955	**0/951/0**	0/957/0	0/996/0	0/2484/0
RYES93	27	943	2500	0/1471/0	**0/1045/0**	0/1451/0	0/4676/0
CARF92	40	1566	2000	0/428/0	**0/383/0**	0/427/0	0/2441/0
UTAS92	38	1314	2800	**0/952/0**	0/1104/0	0/1032/0	0/2984/0
CARS91	51	1385	1550	0/342/0	**0/230/0**	0/356/0	0/2217/0

indicates that features both in the problem and in the particular heuristic interact, as one might reasonably expect.

5 EVOLVING COMBINATIONS OF CSP METHODS AND KNOWLEDGE-BASED CONDITIONS

Previous studies have shown that some heuristics work better for solving certain instances (see [CLL96] and [MJPL92] for example). However, it is not clear which approach is the most beneficial for solving a given set of problems, or even for solving a particular problem. Furthermore, for some problems a strategy might work well, but occasionally some other approach might work better. This behaviour was also observed in results produced by the modified Brelaz algorithm for handling the edge, near-clash, and capacity constraints in Table 2, where no single strategy was the winner for all problems. Some problem features may determine which particular heuristic is appropriate for solving it. For example, in some problems it may be necessary to pay attention to the packing question first because there are many large exams; spreading them out rather than trying to find a packing that best uses available capacity could mean that in due course there is nowhere left to place yet another big exam. For instance, suppose there are 10 timeslots, each of capacity 100, and there are 18 exams with 50 candidates each and one with 51. The problem is solved only by packing the 18 exams into 9 slots.

In order to produce solutions that depend on the particular characteristics of the problems being solved, a GA was set to evolve combinations of these heuristics to find the right one to solve the particular instance. The basic goal of this representation is to characterise the space of algorithms which are variants of the methods explained above. These methods normally are going to choose an examination E and place it in a timeslot S. The question is always related to how to choose and where to place. The idea with the evolution process is to find a mix of choices. In what follows, in particular, the aim is to evolve something that uses heuristic A to select, and heuristic B to place, until a certain condition C first holds. Thereafter, the process continues using heuristic D to choose a exam, and heuristic E to place it in a slot.

Three different strategies, Brelaz (BR), Backtracking (BT), and Forward Checking (FC), were used in this GA. In the BT method, when the current node/exam is assigned a value, a consistency check is carried out against the past variables. If a consistency check fails, a new value is tried. If no suitable value is found to be consistent, then the algorithm backtracks to reconsider the previous node/exam, and so on. In the FC algorithm (also known as *look ahead* algorithm), when the process instantiates the current variable, it looks ahead at unassigned nodes/exams and removes any values for them that are inconsistent with the current instantiation. If this causes the elimination of all possible values for a future variable, then a new value is sought for the current node/exam. If a value for the current node/exam cannot be found, the process simply backtracks.

The representation used in the GA consists of a 10-position array of characters in which each position means the following:

- *Flag* is a set of bits that determines which particular constraint is handled purely by the penalty function (0) or by the application of a given heuristic which tries to handle that constraint (1). One bit relates to the edge constraint type, another to the near-clash constraint type and a third one to the capacity constraint type. After some preliminary experiments, it was found to be beneficial to set all bits to 1 so that no constraint type are handled by penalty methods.
- *Strategy 1.* This strategy will initiate the construction of the timetable (**0: BT**, **1: FC** and **2: BR**).
- *Variable Ordering* used for Strategy 1.
- *Value Ordering* used for Strategy 1.
- *Condition.* This condition establishes when the construction process of the timetable will change from strategy 1 to strategy 2.
- *Strategy 2.* This strategy will continue with the construction of the timetable if *Condition* establishes the change in strategies (**0: BT**, **1: FC** and **2: BR**).
- *Variable Ordering* used for Strategy 2.
- *Value Ordering* used for Strategy 2.
- *Number*(α). This is an integer between 0 and the size of the problem. This number is a parameter used by a choice of *Condition* above. Its purpose will be explained below.

Various heuristics related to variable and value ordering were defined for each of the three strategies incorporated into the GA. Since BT and FC work in similar fashion, the heuristics for them are the same.

The option BR treats the ordering differently so others heuristics were included. The heuristics for each strategy were chosen from the following list:

- **Variable Ordering**. The ordering of nodes for these strategies is generated at start and it depends on the basis of some particular features of the problem. The choices are the following:

  0 By Number of Instances.
  1 By Number of Arcs.
  2 By Number of Students.
  3 By Number of Instances + Arcs + Students.
  4 Pure Random.
  5 By Number of Instances on the nodes using the available colours.
  6 By Number of Arcs on the nodes using the non-available colours.
  7 By Number of Students on the nodes using the available colours.
  8 Random (pre-established). At start this random ordering between each pair of nodes is established so that when the same situation is encountered a consistent decision is made.

- **Value Ordering**. Once that a node has been selected and a set of possible values (timeslots) has been created, a heuristic should order these values to establish how the particular strategy will pick the values. Heuristics for doing this process are the following:

  0 By considering slots in the order: $1, 4, 7, ...3, 6, 8, ...2, 5, 8, ...$
  1 By considering slots in the order: $1, 3, 4, 6, 7, 9, ...2, 5, 8, ...$
  2 By considering slots in the order: $1, n, 3, n-2, ...2, 5, 8, ...$
  3 By considering slots in the order: $1, 2, 3, ..., n$.
  4 By increasingly ordering the timeslots by the number of incoming Instances from the nodes in adjacent timeslots.
  5 By increasingly ordering the timeslots by the number of incoming Arcs from the nodes in adjacent timeslots.
  6 By increasingly ordering the timeslots by the number of incoming Students from the nodes in adjacent timeslots.
  7 By increasingly ordering the timeslots by the sum of incoming Instances, Arcs, and Students from the nodes in adjacent timeslots.
  8 By increasingly ordering the timeslots by the number of nodes using each of them.
  9 Random.

BT and FC use the following heuristics for variable ordering: **0, 1, 2, 3, 4**; and these heuristics for value ordering: **0, 1, 3, 4, 5, 6, 7, 8, 9**.

BR uses the following heuristics for variable ordering **0, 1, 2, 5, 6, 7, 8**, and these heuristics for value ordering: **0, 1, 2, 4, 5, 6, 7, 8, 9**.

Several conditions have been considered and tested in the GA. They include:

0 Limiting Backtracking (This is always ON). If the number of backtracks steps is greater than the prespecified number when using strategy 1, then the construction process changes automatically to use strategy 2. If the backtracks steps are exceeded when using strategy 2, then an incomplete solution is produced.

1 Limiting TIME (This is always ON). If the number of seconds is greater than the prespecified number when using strategy 1, then the construction process changes automatically to use strategy 2. If the number of seconds is exceeded when using strategy 2, then an incomplete solution is produced.

2 Schedule the LARGE (20 percent of total seat capacity) events with Strategy 1 according to the selected variable and value ordering, then change to Strategy 2.

3 Schedule the first α events from the chosen variable ordering using Strategy 1 (α is obtained after decoding the last two positions of the chromosome), then schedule the remaining ones with Strategy 2. This is a way for keeping the random variable ordering (called *controlled randomness*) in such a way that it can be used again when building timetables from similar chromosomes in future generations.

The partial timetable generated by strategy 1 which has followed a certain variable ordering to produce it, may call for adjustments that have to be sorted out properly to have a smooth transition to strategy 2. The partial permutation of variables used by strategy 1 to generate the partial timetable has to be eliminated from the permutation of variables to be used by strategy 2, when this is either BT or FC. The variables in that order will guide the construction of the rest of the timetable. For the Brelaz case, the remaining variables will be dynamically ordered according to the colour-degree.

The random variable ordering for the three strategies is handled in a way intended to keep a *good* previous random ordering, that is, an ordering that has been used to construct an acceptable timetable. When this option is selected, the best random ordering so far is used again with a probability of 1/3. A new random ordering is generated otherwise.

Experiments with this new representation were carried out using a simulated parallel GA with elitism, 5 populations of 25 individuals each using migration every 10 generations, modified tournament selection of size 5, and two-point crossover. The process runs for 100 generations producing a maximum of 625 evaluations of the fitness function (125 initially, plus 500 afterwards).

Results after running 5 trials for each real problem are shown in Table 3. The first column presents the features of each of the problems in the real set. The column headed *Brelaz Best* refers to results coming from the modified Brelaz algorithm and shown previously in Table 2. Recall that this table presents the results for the modified Brelaz algorithm with the four heuristics to handle the value ordering (near-clash) explained in section 2. The final column shows the results of the GA with the new representation. The first of the columns provides the result of the best of the 5 trials carried out, in form of *edge/near-clash/capacity* which indicates the number of violated constraints of each type. The second column indicates the particular combination of strategies and heuristics which achieved the best result for each problem. The last column shows the rule that was used to change strategies and the number of events scheduled with strategy 1. WL (With-Large) indicates that the change of strategies occurred by effect of condition 2 and Wα (With-α) implies that condition 3 was employed for making the change.

Results are very encouraging since the new approach beats all previous outcomes for the tested problems. This is particularly observed in the number of violated near-clash instances in each problem. For both the Brelaz and the GA, no violations of the edge and capacity constraints are found in the solution. It is also clear that no single strategy is the one that succeeds for all problems. Although the Brelaz heuristic appears in most of the combinations, it is also true that this is combined with the FC and the BT in some cases, or even with the BR strategy itself with different variable and value orderings. It is worth noting also that for problems EARF83 and TRES92 the best result is produced with the FC strategy or a combination of this one with the BT strategy. There exist some cases in which a single strategy is capable of generating the complete timetable at once, for example for EARF83, LSEF91, RYES93, CARF92 and CARS91.

Table 3: Evolution of CSP Strategies against Best solution of modified Brelaz on Carter's real-life exam timetable problems. Figures in columns labeled *Brelaz Best*, *GA Avg.*, and *GA Best* indicate the number of violated constraints of each type (*edge/near-clash/rooms over capacity*). Column *Best Strategy* shows the strategy or strategies used to solve the problem with its (their) respective variable and value orderings. Next, the condition that established the change of strategies and the number of events scheduled with strategy one are presented if a combination of two strategies was used.

Problem	Sls	Seats	Brelaz Best	GA Avg.	GA Best	Best Strategy	
HECS92	21	1250	0/302/0	0/190/0	0/154/0	BR(7,1)-BT(0,1)	WL-24
STAF83	15	600	0/1338/0	0/932/0	0/821/0	BR(8,2)-BT(3,0)	Wα-127
YORF83	21	500	0/783/0	0/764/0	0/708/0	BR(0,2)-FC(2,1)	Wα-119
UTES92	12	1250	0/816/0	0/632/0	0/594/0	BR(2,0)-BT(1,1)	Wα-16
EARF83	24	700	0/880/0	0/723/0	0/723/0	FC(4,0)	
TRES92	27	655	0/613/0	0/599/0	0/586/0	FC(4,1)-BT(3,0)	WL-25
LSEF91	21	900	0/302/0	0/247/0	0/221/0	BR(8,0)	
KFUS93	24	1955	0/951/0	0/231/0	0/223/0	BR(1,0)-FC(3,0)	Wα-97
RYES93	27	2500	0/1045/0	0/754/0	0/671/0	BR(8,1)	
CARF92	40	2000	0/383/0	0/285/0	0/285/0	BR(2,0)	
UTAS92	38	2800	0/952/0	0/936/0	0/902/0	BR(0,0)-BR(6,2)	Wα-262
CARS91	51	1550	0/230/0	0/170/0	0/130/0	BR(8,0)	

There is an assortment of variable and value orderings employed by the strategies. For the variable ordering for example, all possible options (except option 5) appear in at least one solution. In the value ordering however, options 0, 1, and 2 prevail in the solutions.

Table 3 also shows results comparing the Best solution by the modified Brelaz for each of the problems against the average on the 5 trials obtained by the GA approach. Again, from this point of view, this approach outperforms the Brelaz in all problems confirming its benefit when tackling this kind of ETTPs.

Results suggest that the evolution of combinations of strategies proves its utility for solving a variety of real problems. It seems that exploiting the problem-specific features by means of choosing a set of strategies and heuristics which best adapt to that, a better overall performance can be achieved.

6 DISCUSSION

Utilisation of non-direct representations for solving ETTPs, and possibly other similar problems, seems to be the right direction when using GAs. There are so many ways, however, in which the construction of a timetable can be carried out since it depends on the particular representation and the method for interpreting the given chromosome. Direct encodings for very large problems require long chromosomes, and lead to the kinds of failure to co-ordinate different parts of a solution that were outlined above.

A modified version of a Brelaz algorithm was considered and the approach showed acceptable performance. However, the algorithm had to be customised for the problem, so another approach based on the evolution of strategies and heuristics was proposed. It consists of evolving lists of CSP strategies (Brelaz, Forward Checking, and Backtracking), heuristics (variable and value orderings) and conditions for establishing the change from one strategy to another when constructing the timetable. This manner of building the timetable clearly showed its usefulness since it obtained the best results for each one of the problems in the Toronto real-world benchmark set. Issues about the time for delivering solutions with this method are still a matter of further research but, in general, this has always been a concern in the field of GAs.

Results regarding the number of violations on the edge, near-clash, and capacity types of constraints for problems in the Toronto set have been improved in relation to the modified Brelaz algorithm, which had previously produced the best solutions for these problems. However, it is important to go a little bit further and consider the real utility of the combinations of strategies approach since one argument against such technique might be the time for delivering solutions.

Considering all possible combinations of strategies, heuristics, and conditions, there are a total of 93636[1]. This number does not take into account the different *flags* in the first position of the chromosome (recall that it was decided that all constraints would be handled by the heuristics), nor the number α which is used by condition 3 when appearing in the chromosome. This number in particular, would increase the number of possible combinations by $93636 \cdot 0.25 \cdot size\_of\_problem$, since there is a probability of 0.25 of having condition 3 in the chromosome and $size\_of\_problem$ different values for changing from strategy 1 to strategy 2. For example, for problem HECS92 there are around 1.8 million possible combinations.

For comparison purposes 36992 of these combinations were evaluated, by considering all options for the three kinds of constraints. The random value ordering option for each strategy and the conditions 0 and 1 were not varied in this brute-force search. Figure 2 shows results from all those combinations for problem TRES92. The plot presents the spectrum of combinations in the x-axis and their penalty earned in the y-axis. Penalty values in the plot are restricted up to certain values (recall that penalties are computed according to the number of violated constraints of each type times the penalty earned by each, 100 for a clash, 1 for a near-clash, and 20 for a seat violation). For example, for problem TRES92 the plot shows only combinations which earn no more than a penalty of 2000, so that combinations with a higher value do not appear in it (there are many such cases). Note that the best solution delivered by the GA for each tested problem, is very close to the minimum found by the enormously more costly brute-force – see Table 3 (TRES92(0/586/0)). And this has been accomplished by running the GA for 625 evaluations, which is just a small percentage of the total number of possible evaluations (one for each combination).

Finally, it is worth pointing out that a university tends to have similar patterns of exam constraints and sizes from year to year. It may be that a combination of heuristics that was found to be good for one year will also be good for that university for the next year too. This would be worth investigating.

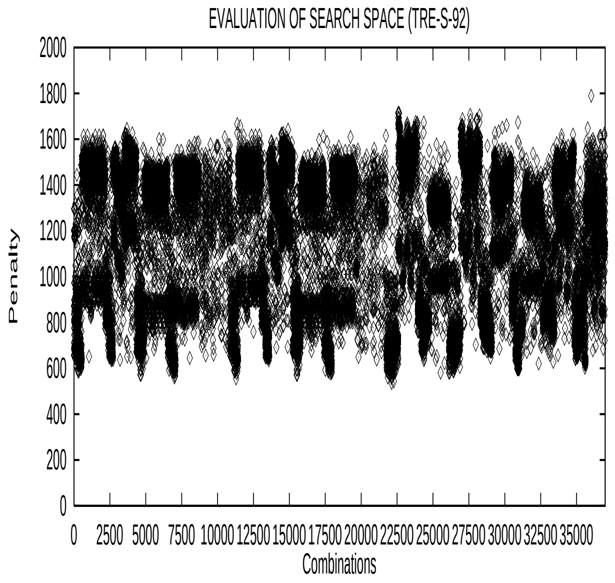

Figure 2: Fitness Landscapes for problem TRES92.

Acknowledgements

The authors acknowledge the support received from CONACyT, ITESM, and the Division of Informatics of the University of Edinburgh. The investigation was partially funded by the CONACyT research project

[1]This number of combinations is obtained as follows: $2 \cdot 5 \cdot 9 \cdot 4 \cdot 2 \cdot 5 \cdot 9 + 2 \cdot 5 \cdot 9 \cdot 4 \cdot 1 \cdot 7 \cdot 9 + 1 \cdot 7 \cdot 9 \cdot 4 \cdot 2 \cdot 5 \cdot 9 + 1 \cdot 7 \cdot 9 \cdot 4 \cdot 1 \cdot 7 \cdot 9 = 32400 + 22680 + 22680 + 15876 = 93636$.

3849P-A9607, and Hugo Terashima-Marín was supported by the CONACyT grant 66696.

References

[AA91] D. Abramson and J. Abela. A parallel genetic algorithm for solving the school timetabling problem. Technical report, C.S.I.R.O., April 1991.

[BNW96] E. K. Burke, J.P. Newall, and R.F. Weare. A memetic algorithm for university exam timetabling. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling, LNCS 1153*. Springer Verlag, 1996.

[Bre79] D. Brelaz. New methods to colour the vertices of a graph. *Communications of the ACM*, 22, 1979.

[CFM92] D. Corne, H. L. Fang, and C. Mellish. Solving the modular scheduling problem with genetic algorithms. In *Proceeding of the 6th Int. Conference: Industrial and Engineering Applications of AI*, Edinburgh, Scotland, 1992.

[CLL96] M. W. Carter, G. Laporte, and S. Y. Lee. Examination timetabling: Algorithmic strategies and applications. *Journal of Operations Research Society*, 47:373–383, 1996.

[CRL94] D. Corne, P. Ross, and H.L. Lang. Fast practical evolutionary timetabling. In *AISB workshop on Evolutionary Computation*. Springer Verlag, 1994.

[Dav91] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrtand Reinhold, New York, 1991.

[Erg96] Ayhan Ergul. GA-based examination scheduling experience at middle east technical university. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling, LNCS 1153*. Springer Verlag, 1996.

[FF96] C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.

[Kit90] H. Kitano. Designing neural networks using gas with graph generation system. *Complex Systems*, 4:461–476, 1990.

[MJPL92] S. Minton, M. D. Johnston, A. Phillips, and P. Laird. Minimizing conflicts: A heuristic repair method for csp and scheduling problems. *Artificial Intellgence*, 58:161–205, 1992.

[PCLP94] B. Paechter, A. Cumming, H. Luchian, and M. Petriuc. Two solutions to the general timetable problem using evolutionary methods. In *The Proceedings of the IEEE Conference of Evolutionary Computation*, 1994.

[PCNL96] P. Paechter, A. Cumming, M. G. Norman, and H. Luchian. Extension to a memetic timetabling system. In E. Burke and P. Ross, editors, *Practice and Theory of Automated Timetabling, LNCS 1153*. Springer Verlag, 1996.

[RCH97] P. Ross, D. Corne, and E. Hart. Some observations about GA-based exam timetabling. In *Proceedings of the Second Conference on the Practice and Theory of Automated Timetabling*, Toronto, Canada, 1997.

[SE96] J.D. Schaffer and L.J. Eshelman. Combinatorial optimization by genetic algorithms: The value of the genotype/phenotype distinction. In V. Rayward-Smith, I.H. Osman, C. R. Reeves, and G.D. Smith, editors, *Modern Heuristic Search Methods*. John Wiley and Sons, 1996.

[Smi85] D. Smith. Bin packing with adaptive search. In J. Grafenstette, editor, *Proceedings of the International Conference on Genetic Algorithms*, 1985.

[TGMS94] G. Trzewik, E. Gudes, A. Meisels, and G. Solotorevksy. Traps: Time dependent resource allocation language. In *Proceedings of the ECAI Workshop on Practical CSPs*, pages 65–72, 1994.