Evolutionary Algorithms in Lecture Timetabling

Panagiotis Adamidis, Panagiotis Arapakis
Dept. of Informatics
Technological Educational Institution of Thessaloniki
Thessaloniki, Greece 541 01
adamidis@it.teithe.gr

Abstract- Several approaches have been used to solve the timetabling problem. Recently, Evolutionary Algorithms (EAs) have been applied with very good results to various types of timetabling problems. This paper presents the results of our work on a real course timetabling problem using EAs. We compare two versions of the EA with different representation schemes. We also compare the performance of three configurations, with or without mutation and different recombination and mutation operators. The results show that EAs can always produce a feasible solution without violation of any hard constraint.

1 Introduction

Evolutionary Algorithms (EAs) have been successfully applied to various types of timetabling problems, such as school (Abramson and Abela 1992, Colorni and Dorigo 1991), rail (van Wezel and Kok 1994), course (Corne et al 1994, Ross et al 1994) and examination (Burke et al 1994, Burke et al 1995, Corne et al 1994) timetables.

The problem of creating the proper timetable includes the distribution of events to a weekly period. The timetable must satisfy a number of constraints that concern the capacity and suitability of the rooms, the availability and suitability of the instructors, the relation with other courses, etc.

In this paper we are concerned with University Course Timetabling. Usually each department is responsible for its own timetable. Student course choices sometimes are finalised after the courses have started. Given this, the best the timetabler can do is to minimise the number of potential conflicts and then let students choose their courses on the basis of the timetable produced. Here, we describe the methodology we used to automate the procedure of using EAs to create the weekly timetable of the department of Informatics at the Technological Educational Institution of Thessaloniki. Until now, the procedure of creating a suitable timetable was assigned to a faculty member.

The methodology used is not bound to the peculiarities and constraints of the specific department, and it can easily be extended to include any weekly lecture timetable.

2 Evolutionary Algorithms in Timetabling problems

The basic element of a lecture timetabling problem is a set of events $E = \{e_1, e_2,...,e_n\}$. There are also a set of times $T = \{t_1, t_2,...,t_s\}$, a set of places $P = \{p_1, p_2,...,p_m\}$, and a set of 'agents' $A = \{a_1, a_2,...,a_t\}$. Each member of E is a unique event requiring assignment of a time, a place, and an agent (lecturer, tutor, technician etc.). An *assignment* is a four-tuple (e,t,p,a) such as $e \in E$, $t \in T$, $p \in P$, and $a \in A$, with the interpretation "event e starts at time t in place p involving agents a". A lecture timetable is simply a collection of n assignments, one for each event. The problem is to find a timetable that satisfies, or minimally violates a collection of constraints (Ross et al 1994).

There are several kinds of constraints, the most common being an 'edge constraint' between two events, which states that a given pair of events must not overlap in time. There are several other kinds of constraints such as unary, capacity, and agent constraints (Corne et al 1994, Ross et al 1994).

EAs are stochastic search techniques that explore combinatorial search spaces using simulated evolution. The effectiveness of EAs is limited by their ability to balance the need for a diverse set of sampling points, with the desire to quickly focus the search on potential solutions. Exploration is achieved through the recombination of data structures (which represent possible solutions). The data structures constitute the population of the EA, which is evolved towards the optimum, concentrating the search in areas of higher fitness (exploitation). Recombination and mutation perturb the individuals of the population providing general heuristics for exploration.

In applying an EA to a problem, central considerations are the choice of a chromosome, the design of the fitness function and the genetic operators used to evolve the population.

There are essentially two different types of representation that have been used in evolutionary timetabling. The direct representation encodes the actual timetable, where an individual is a vector of symbols *nxe*, where e is the number of events and n is the number of genes for each event (Abramson and Abela 1992, Burke et al 1994, Burke et al 1995, Corne et al 1994, Ross et al 1994). The implicit representation, on the other hand, encodes a set of instructions as to how the timetable should be build (Paechter et al 1994, Paechter et al 1995). The fitness function and the genetic operators used are described in the sequel.

3 Problem Presentation

In order to handle the problem we had to consider all the factors, constraints and peculiarities of the specific department, before the coding of the population.

The main difficulties are:

- The large number of courses held in laboratories
- The small number of available lab places
- The different hour limit (upper and lower) of each agent
- Most of the labs need two agents
- Some courses are taught twice a week

The desirable problem solution (weekly timetable) must satisfy the following constraints:

1. Each course must be scheduled once. Courses that are taught twice a week have different codes for each teaching (*couple teachings*)
2. All courses must be present in the final timetable
3. The two agents scheduled for each lab must be different
4. Couple teachings must have the same agents
5. Couple teachings must be scheduled on different days
6. There is an upper limit for the hours that can be assigned to each agent
7. There is a lower hour limit for some agents
8. The daily timetable for each agent must be continuous
9. For each day, courses must be scheduled between some time limits, i.e. no courses can be scheduled after 10:00 pm or before 08:00 a.m.
10. The daily timetable for each semester must be continuous
11. No overlaps on lectures of the same semester
12. Each agent can be assigned only one lecture at the same time
13. Each place can host only one course per time slot
14. Each course must have a suitable agent
15. Courses must be assigned to suitable places

When a constraint is violated a penalty is imposed on the individual. The fitness of the individual depends on the penalties imposed on him. Constraints are distributed in different categories of importance. Even though the final solution must satisfy all the constraints, the constraints No 1, 2, 9, 12, and 13 are considered harder, and the penalty associated with them is larger. In order to find whether a constraint is violated, we need to check the timetable thoroughly, and make hundreds of comparisons.

Some constraints are connected in some way. For example, when constraint 1 is violated, constraint 2 is violated as well, since for each course that appears twice in the timetable, there is a lack of another course. Also, because of the very large number of lab courses, it is very difficult to satisfy constraint 10 and at the same time minimise total duration of courses per day (Constraint 9). This is mainly the reason for the frequent violation of constraint 10.

The problem becomes harder as the number of courses increases, while the number of agents, available places and time slots remains the same. The question is not if the algorithm is capable of finding the optimum solution, but what constraints will be violated by the final solution. A solution is usually acceptable if the violated constraint(s) is not crucial.

4 Implementation

Each chromosome/individual represents an entire weekly timetable. Each individual in the initial population is assigned random values within some range.

The population evolves to the final solution using genetic operators (recombination and mutation). An individual may be created using both recombination and mutation, or one of them, or no operator at all. We also use elitism (best individual survives always), and adaptive operator probabilities. Operator probabilities are adjusted after the creation and evaluation of each generation, depending on the convergence of the population.

The algorithm terminates when a number of generations is completed, or the global optimum is reached (i.e. no constraints are violated), or when a best individual is not improved for a certain number of generations. After termination, the best individual is recorded for future reference.

4.1 Coding of the population

We used integer-valued encoding of each gene, with values within some range. We preferred integer-valued from string encoding because we should avoid creating a gene with parts from two different parents. If we allow this to happen then it is very likely to produce genes with allelic values that are out of range. Integer-valued encoding is a natural way to overcome this problem.

We tried two different “direct” representation schemes. In both schemes, each individual represents a weekly timetable. An individual is an array *nxe*, where *e* is the number of events and *n* is the number of genes for each event.

In the first representation, the *n* alleles in each event represent the course, place, agent 1, agent 2, day, starting time, duration, and semester. This means that *n*, the number of lines of the array (genes) is 8. There are 180 different events to be scheduled (columns of the array).

Assignment of initial values for each event is random, but within some limits for each one. For example the value for day must be between 1 and 5. In this way we eliminate out of limit values for each assignment. Figure 1 gives a schematic representation of an array representing an individual (timetable). For example, interpretation of event 1 is: course with code 4 to be taught to the 5th semester, at place 2, by the two agents with codes 1 and 6, every Monday (code 1), from 8:00 to 10:00. When the code for

agent 2 is equal to 0, then only one agent is assigned to this event.

	Events					
	1	2	3	4	...	Max
Course	4	30	5	17	...	20
Semester	5	4	1	2	...	3
Place	2	2	3	10	...	11
Agent 1	1	14	4	20	...	1
Agent 2	6	5	0	0	...	0
Day	1	4	5	4	...	3
Starting Time	8	10	12	14	...	12
Expiration Time	10	13	14	14	...	14

Figure 1: First representation scheme of a timetable.

In an effort to make the algorithm faster and more effective, we modified the initial representation. Even though the modifications mainly concern the representation, they influence the genetic operators and the fitness function as well.

The main modification is the placement of courses to specific places in the array. First event (first column) corresponds to the course with code 1, second event (second column) to course with code 2, and so on. The main advantage of this change is that there is no violation of constraints 1 and 2. Each course is scheduled once, and all courses are present in the timetable. So, there are no penalties due to the violation of the first two constraints. It also has an obvious positive impact to the recombination operator, which always produces individuals with all courses arranged in the preset order. The speed of the algorithm is also improved.

Since each course is taught at a specific semester, the semester value of each event is also pre-determined. So, the characteristics, course and semester, are redundant and there is no need to include them in the array. Thus, the second representation scheme reduces *n*, the lines of the array (assignments for each event), to 6. This is expected to reduce the computational time of the algorithm. It is also expected to improve the performance, since it reduces the search space. Figure 2 gives a schematic representation of an array representing an individual (timetable), using the second representation scheme. "Max" is also equal to 180.

Another modification concerns the initialisation of the population. Agent and place assignments are suitable for each course, even though they remain random. That is, for each event, agents and place are chosen randomly from a specific set suitable for that course. In this way it is certified that all courses are assigned suitable agents and places. This is not affected by the recombination operator. Thus the constraints 14 and 15 are never violated. This is also expected to improve the computational time of the evaluation of the fitness of the individuals, since it will not check for the violation of these two constraints.

	Events					
	1	2	3	4	...	Max
Place	2	2	3	10	...	11
Agent 1	1	14	4	20	...	1
Agent 2	6	5	0	0	...	0
Day	1	4	5	4	...	3
Starting Time	8	10	12	14	...	12
Expiration Time	10	13	14	14	...	14

Figure 2: Second representation scheme of a timetable.

4.2 Coding of the population

The genetic operators used are recombination and mutation. Two different recombination operators were constructed and used in two different versions of the EA.

4.2.1 Recombination operator

Recombination operator is responsible for inheriting the characteristics of the parents to the children. We use multiple-point recombination. The number of cutting points is *int(e/c)*, where *e* is the number of events and *c* is a constant with a value of 20. We have chosen c to be 20, expecting it to give a balanced number of cutting points for the problem at hand--neither too many, nor too few. Initial results with smaller (2 and 10) and bigger (30 and 50) values of c confirmed that value 20 was the best choice.

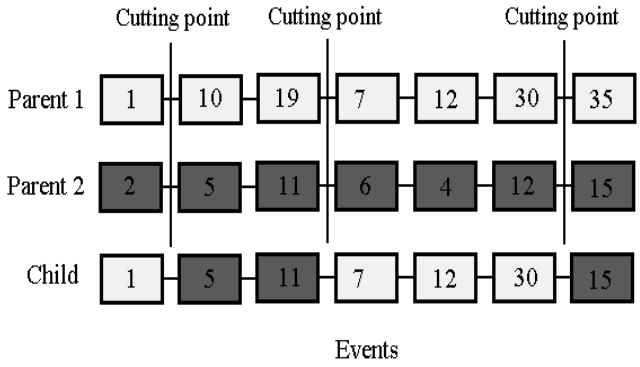

Figure 3: First recombination operator. Events are not divided.

As already mentioned, we developed two different recombination operators. The first one allows only recombination of different events (fig. 3). This means that during initialisation each and every event must be correctly initialised. Otherwise, only mutation can change and correct an assignment to the events that violate one or more constraints.

The other operator allows the cutting points to be within an event (fig. 4). So, this operator can recombine assignments from different events, changing the initial event assignments. In this way the operator is able to correct the

events that violate one or more constraints, but also destroy correct events with false assignments.

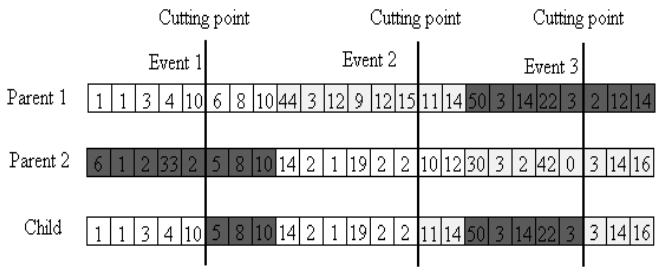

Figure 4: Second recombination operator. Events may be divided.

4.2.2 Mutation operator

Generally, mutation plays a dual role in EAs. It provides and maintains diversity in a population and it can also work as a search operator in its own right. Moreover, the mutation operator used here has the advantage of avoiding to create unfeasible solutions. It replaces the value of a gene with some allelic value within the proper range for the specific assignment affected. This value is suitable for the specific event as well. That is, the replacement value is not totally random but in the direction of improving each event with proper values. This also helps in creating feasible timetables. Table 1 shows appropriate range of values for different assignments. In the second representation scheme, course is not affected by the mutation operator.

For the agent(s) assignment, mutation randomly selects an agent from the subset of agents that are suitable for the specific course. In case that agent 2 was also assigned, then the course refers to a lab place. Expiration time changes according to the value of starting time.

Before deciding on the final form of the mutation, we tried totally random mutation where each affected gene is replaced by a random value within the range of possible values for the affected assignment. The performance of this scheme was poor and was abandoned in favour of "directed" mutation as described earlier.

Table 1: Range of values for different mutation assignments.

Assignment	Range of Values
Course	1 - e
Place	Suitable for specific course
Day	1 - 5 (five weekdays)
Starting time	8 - 17
Agent(s)	Suitable for specific course

The genes of an event that may be affected by mutation are: course, place, day, agent(s) and starting time. The semester and expiration time depend on the course and starting time respectively.

The number of genes that will be mutated is a random number in the space [1,max], with max given from $(nxe)/c_2$, where e, the number of events is 180, n, the number of genes is 8 in the first representation scheme, and 6 in the second, and c_2 is a constant number. The constant 50 was chosen in order to limit the maximum number of possible mutations to relatively small values. We have only initial experimental results that propose the use of value 50 against two other constant values (20 and 100).

The cells of the array that will be affected by mutation are also determined at random.

4.3 The Evaluation Function

The evaluation function is responsible for calculating the fitness of an individual, that is, the quality of a timetable. When a constraint is violated a penalty is imposed on the individual. The fitness of the individual is inversely proportional to the number of constraints violated in a timetable. Each instance of a violated constraint is weighted according to how important or not it is to satisfy it, that is, to a weighted sum of the penalties imposed on the individual.

Let C be the set of constraints in the problem, P_i a penalty associated with constraint i, and $u_i(t)$=1, if timetable t violates constraint i, and 0 otherwise. The fitness function is:

$$f(t) = 1/(1+\sum_{i \in C} P_i u_i(t)) \quad (1)$$

Assuming that all the penalties are positive, this function is 1 if and only if all the constraints hold, otherwise it is less than one. Further discussion of this kind of fitness function appears in [6, 10].

4.4 Penalty Settings

Penalties must be set with care. The relative penalty values are usually chosen to reflect intuitive judgement of the relative importance of satisfying different kinds of constraint. They also lead the solution towards satisfying the constraints that the user considers most important. Search concentrates on a region of the search space that is low in violations of the more penalised constraint. For example, if we want to eliminate place overlapping, we have to increase the penalty for this violation.

Table 2, presents the penalties used in our implementation for the two representation schemes. These are the final values used after extensive experimentation with a lot of penalty settings.

The algorithm takes care of assigning appropriate values to starting and expiration times of all the events. Thus, no penalty has been set for the violation of the ninth constraint. As can be seen, the second representation scheme does not penalise some constraint violations (1, 2, 14, 15). This is because under this representation these constraints are never violated. The fact that these constraints are not checked,

reduces the computational time of the fitness function and speeds up the EA.

Table 2: Penalty settings for the two representations

Constraints	First scheme	Second scheme
Course scheduled more than once (No. 1)	2	–
Missing course (No. 2)	2	–
Agent 1 is the same as agent 2 (No. 3)	0.3	0.3
Different agents in "*couple teachings*" (No. 4)	0.2	0.2
Couple teachings on the same day (No. 5)	0.2	0.2
Each hour that exceeds an agent's hour limit (No. 6,7)	0.05	0.05
An empty hour in the daily schedule of an agent (No. 8)	0.01	0.01
The time of an event is out of limits (No. 9)	–	–
An empty hour in the daily schedule of a semester (No. 10)	0.1	0.1
Two events of the same semester overlap in time (No. 11)	0.03	0.03
More than one event assigned to an agent at the same time (No. 12)	0.5	0.5
For each place overlap (No. 13)	0.2	0.2
Agent not appropriate for the course (No. 14)	0.2	–
Place not appropriate for the event (No. 15)	0.2	–

5 Experiments

We tested three configurations of the algorithm for each representation scheme. In the first configuration, mutation is the only evolutionary operator used (configuration *Mutate*). The other two use the two different recombination operators, with adaptive probabilities of recombination and mutation. One of them uses the first recombination operator (configuration *Adapt_1*), and the other the second (configuration *Adapt_2*), as described earlier in section 4.2.1.

Table 3: Operators probabilities for configurations Adapt_1 and Adapt_2.

Operator	Probability
No operator	5%
Recombination	55%
Mutation	15%
Both operators	25%

Initial probabilities of applying recombination and mutation are presented in Table 3. These values are dynamically altered during the evolution of the population as described in the sequel. We keep statistics of the population. When the population converges, the probability of recombination is reduced by 5%. On the other hand, when the population diverges, the mutation probability is reduced by 5%. Upper and lower limits of the two operator probabilities are 60%, and 10% respectively. The other two probabilities (no operator, both operators) remain constant.

All configurations use elitism (survival of the fittest individual).

The selection mechanism used is tournament selection which was found to be the best choice in [8], with tournament size equal to 5 .

In all cases we used two different population sizes (50 and 100 individuals). Twenty trials were run for each experiment. We allow each EA trial to evolve for 5000 generations, unless another termination criterion is satisfied.

Tables 4 and 5 present the results from the experiments. Table 4 presents the results from the first representation scheme, and Table 5 the results from the second. All trials of the first representation scheme terminate when the generation limit (5000) is reached. This is not the case for the second representation scheme where the improvement of the population is the termination criterion mostly used.

As can clearly be seen in Table 4, using the first representation scheme, the fitness of the individuals, is far from being optimum (i.e. 1). This is mainly due to the violation of the first two constraints (absence or repetition of courses), which are highly penalised (Table 2).

Table 4: Comparative results of the first representation scheme (Fitness values).

Population size: 50 individuals			
	Adapt_1	Adapt_2	Mutate
Average	0.00656	0.00678	0.01410
Best	0.00761	0.00750	0.01578
Worst	0.00577	0.00629	0.01224

Population size: 100 individuals			
	Adapt_1	Adapt_2	Mutate
Average	0.00681	0.00692	0.014399
Best	0.00775	0.00775	0.01480
Worst	0.00599	0.00590	0.01389

It can also be seen that the performance of configuration "*Mutate",* is better than the other two. The main reason for the better performance of "*Mutate",* is the "directed" nature of the mutation operator. It alters the affected gene in the direction of satisfying some constraint(s) in the event that this gene belongs to. This constraint is usually one of those with high penalties (first two). The recombination operator

is not very helpful in satisfying these constraints. The best solution obtained can be applied with only few modifications.

The difference in the performance of the two *"Adaptive"* configurations is negligible, with "*Adapt_1"* being slightly better than *"Adapt_2"*. The difference is small because of the high penalties imposed on the violation of the first two constraints. This keeps the fitness function to very low values not allowing the two configurations to show how theycan handle the other constraints.

Table 5 shows that the performance of the second representation scheme is near the optimum. Hard constraints are never violated due to the representation scheme. The best timetable obtained violates only some minor constraints. It can also be seen that the performance of configuration "*Mutate",* is now comparable with the other two. The weighted sum of the penalties for the best solutions is 3.22, 3.59, and 3.51 for "*Adapt_1", "Adapt_2",* and *"Mutate"* respectively.

Table 5: Comparative results of the second representation scheme.

Population size: 50 individuals						
	Adapt_1		Adapt_2		Mutate	
	Fitness	Gener	Fitness	Gener	Fitness	Gener
Avg	0.19747	3604.1	0.16438	3431.3	0.19290	3289.7
Max	023202	5000	0.20964	5000	0.22371	4868
Min	0.14409	2222	0.12376	1770	0.14599	1933

Population size: 100 individuals						
	Adapt_1		Adapt_2		Mutate	
	Fitness	Gener	Fitness	Gener	Fitness	Gener
Avg.	0.20986	3031	0.17002	3534.4	0.19355	3459.8
Max	0.23697	5000	0.21787	5000	0.22173	4888
Min	0.16260	1858	0.14706	2296	0.16779	2264

Comparing the performance of the two representation methods (figure 5), it is obvious that the second representation scheme has a much better performance. It can also be seen that the second representation needs less generations to reach the best solution. This is due to the larger search space of the first representation scheme, and also because of the better initialisation of the population.

In all cases, we had better timetables using a population of 100 individuals. This was expected since larger populations offer more diverse genetic material, improving exploration and exploitation of the search space. Better performance of larger populations is at the expense of the computational time (more time needed to compute the fitness function of 100 individuals), and the convergence speed of the algorithm (more generations needed).

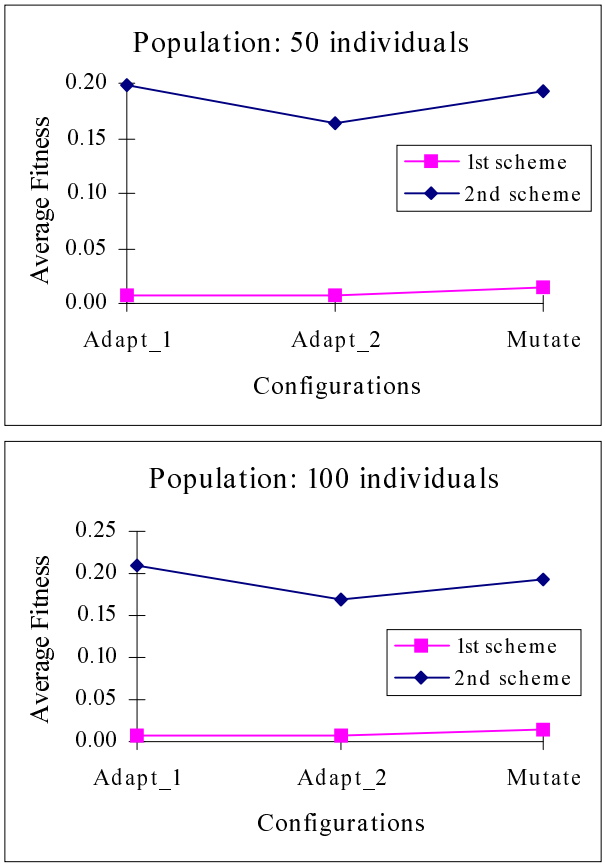

Figure 5: Performance comparison of the two representation schemes for two population sizes.

6 Discussion - Conclusions

It is obvious that hard-coding the fundamental constraints into the representation scheme increases the probability that an appropriate solution can be found. This representation has no side-effects on the generality of our methodology and reduces the size of the search space.

The results clearly indicate that the configuration that uses both mutation and recombination of events (*Adapt_1*) has the best performance. The worst performance of the other adaptive configuration (*Adapt_2*) is due to the destructive attribute of the recombination operator used in this case.

The mutation operator used is very helpful. This is obvious from the very good performance of the configuration that uses only mutation (*Mutate*). Using this operator, the search can be guided towards parts of the solution space with feasible timetables.

Using a larger population size (100 individuals) improves the performance of the algorithm, in all cases. On the other hand, larger populations need more computational resources, which do not always compensate for their use. Initialisation of the population with feasible solutions can also improve the performance and speed up the algorithm.

Despite the numerous constraints imposed, and the generalisation of the problem formulation, the best weekly timetable produced, using the second representation scheme, can be used even with no modifications at all. The only penalty points in the final solutions are from overlaps between lectures/labs of the same semester. These overlaps cannot be avoided because of the very small number of available lab places compared to the courses that must be held in lab places.

References

1. D. Abramson, J. Abela, A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In *IJCAI Workshop on Parallel Processing in AI*, (1992) Sydney, Australia,
2. E.K. Burke, D.G. Elliman, and R.F. Weare, A Genetic Algorithm Based University Timetabling System. In East-West Conference on Computer Technologies in Education, (1994) pp. 35-40, Ukraine.
3. E.K. Burke, D.G. Elliman, and R.F. Weare, Specialised Recombinative Operators for Timetabling Problems. In Proceedings of the AISB Workshop on Evolutionary Computing, pp.75-85, (1995) Univ. of Shefield, UK.
4. A. Colorni, M. Dorigo, and V. Maniezzo, Genetic Algorithms and Highly Constrained Problems: The Timetable Case, Proceedings of the First International Workshop on Parallel Problem Solving from Nature, pp 55-59, (1991) Dortmund, Germany.
5. D. Corne, P. Ross, H.-L. Fang, Fast Practical Evolutionary Timetabling. In Proceedings of the AISB Workshop on Evolutionary Computation, (1994) University of Leeds, UK
6. B. Paechter, A. Cumming, and H. Luchian, The Use of Local Search Suggestion Lists for Improving the Solution of Timetable Problems with Evolutionary Algorithms. In Proceedings of the AISB Workshop on Evolutionary Computing, (1995) Sheffield, England.
7. B. Paechter, H. Luchian, A. Cumming, M. Petriuc, Two Solutions to the General Timetable Problem Using Evolutionary Methods. In *Proceedings of the First IEEE Conf. on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, vol I, pp.300-305, (1994) Orlando, Florida.
8. P. Ross, D. Corne, and H.-L Fang, Successful Lecture Timetabling with Evolutionary Algorithms", Working Paper, Dept of A.I., University of Edinburg, UK, 1994.
9. P. Ross, D. Corne, and H.-L Fang, Improving Evolutionary Timetabling with Delta Evaluation and Directed Mutation. Presented at the *Proceedings of the Third Conf. on Parallel Problem Solving from Nature - PPSN III* (Y. Davidor, H-P. Schwefel, and R. Manner, Eds.), (1994) pp.556-565, Israel.
10. A.E. Smith, D.M. Tate, Genetic Optimisation using a penalty function. Presented at the *Proceedings of the Fifth International Conference on Genetic Algorithms* (S. Forrest, Ed.), (1993) pp. 499-503, San Mateo: Morgan Kaufmann
11. M.C. Van Wezel, J.N. Kok, Genetic Improvement of Railway Timetables. Presented at the *Proceedings of the Third Conference on Parallel Problem Solving from Nature - PPSN III*(Y. Davidor, H-P. Schwefel, and R. Manner, Eds.), (1994) pp.556-565, Jerusalem, Israel.