Solving Timetabling Problems Using Genetic Algorithms

MILENA KAROVA
Department of Computer Science
Studentska 1 Technical University Varna
Email: mkarova@ieee.bg

Abstract

This paper describes techniques that can be applied to a different scheduling and timetabling problems. The problems are characterized as constraints satisfaction problems. The solution methodology uses genetic algorithms to minimize the total penalty for constraint violation. An encoding, genetic operators and fitness evaluation are implemented.

To solve this problem a genetic algorithm maintains a population of chromosomes each of which represents a possible solution (timetable). In every generation, a new population of chromosomes is created using bits and pieces of the fittest of the old generation. The main tasks of applying genetic algorithm to solve a problem are encoding the solution as chromosome, developing a fitness evaluation function and choosing genetic operators and run parameters. The genetic algorithm includes the following functions: *initialize, evaluate, select, crossover, mutate, create new population.*

Our genetic algorithm proposes solution which consists of number of tuples , one for each class. The timetabling constraints are classified: unary constraints, binary constraints and k-nary constraints.
The fitness function is a linear combination of a cost function and a penalty function. The goal is all constraints to be satisfied. .We use constraint propagation approach. There are experimental results.
Keywords: genetic algorithms, timetabling, fitness function, chromosome, genetic operators, constraints.

Introduction

The general Timetabling Problem can be stated as a set E of events, a set S of timeslots, and a set C of constraints between the events. The intention is to arrange all the events into to slots in such away that no constraints are violated.

Although the Timetabling problem could be perceived as very simple problem, the truth is that intrinsically it contains features which characterize it as a member of the class of most difficult problems known as NP-complete. The problem has many practical applications: the scheduling of examinations for students; the scheduling of classes, teachers and rooms in a university; the scheduling of biasness meetings or congress conferences.

Several approaches have been proposed in the literature for solving Timetabling problem and they include: Constraint Satisfaction Problem (CSP), Simulated Annealing (SA), Genetic Algorithms (Gas) and many other heuristic methods.

The process in a GA is based on the combination of three main genetic operators: Selection, Crossover and Mutation. Selection refers to the way an individual is chosen for mating with another individual in the population; Crossover indicates the procedure to extract important features from the mating parents in order to hopefully produce better fitted offspring; and Mutation emulates the process in natural evolution for randomly changing specific characteristics in the chromosome representing the individual.

Applying GA to solve an optimization problem involves the following tasks:

- Encode (represent) solution to the problem to a chromosome (by a bit string)
- Formulate fitness evaluation function
- Select or create the appropriate genetic operators (crossover, mutation, selection e etc.)
- Select run parameters (population size, crossover rate, mutation rate, generation gap, convergence criteria etc.).

Encoding in GA

There are three approaches to encode scheduling problem [3]

- Direct encoding. This is simple strict forward approach. This method directly represents a solution by chromosome. Constraints violation is penalized by decreasing the chromosome fitness. A penalty function is included in the fitness evaluation function.
- Indirect encoding. Direct encoding is not always possible or effective. When the solution space is highly constrained, we will

0-7803-8422-9/04/$20.00 c2004 IEEE

need domain specific operators to restrict the search to within the legal solution space.

An alternate scheme is to encode only the raw information of the solution instead of the whole solution. A solution builder or decoder equipped with constraint information is used to process the raw information and build a legal solution.

- Structured encoding. Use separate chromosomes to encode different levels of structure.

Chromosome structure

A solution to a Timetabling Problem consists of a number of tuples, one for each class. Each tuple consists of several values which are the instantiation of variables for that class. If we order the tuples according to the variable class each class can be identified by its position in the solution and *class* variable can be dropped. Foe example, if there are 5 classes and 2 constraint variables *time-slot* and *room*, the chromosome will be:

$t_1\ t_2\ t_3\ t_4\ t_5\ r_1\ r_2\ r_3\ r_4\ r_5$, where t_i and r_i are the time-slot and room for class i respectively.

With this structure, a chromosome can be specified by the following parameters:

- Number of classes
- Number of variables other than classes
- Number of values each variable can take.

Suppose there are 30 time-slots a week and there are 16 rooms foe use, the above chromosome can be specified as:

- Number of classes=5
- Number of variables other *class*=2 (*time-slot and room*)
- Number of values time-slot can take = 30
- Number of values room can take =16.

With these parameters, we can dynamically define the chromosome as a bit string of length 36. GA uses this mechanism to define the chromosome dynamically for different timetabling problems.

Constraint representation

For constraints, pure data representation lists all variables values that violate or satisfy the constraints. Some measures have been taken to simplify and reduce the number of data structures

1. Eliminate inequality.

Three of the four variables in a Timetabling problem *class, room* and *teacher* are nominal. They take unrelated discrete values. They will not involve inequality. We can treat the variable time as nominal. For example, the constraint *before Wednesday* can be expressed as *equal to Monday or equal to Tuesday.*

2. Unify reward and penalty, soft and hard constraints.

We used a single field to represent reward and penalty. Positive values represent rewards and negative values represent penalties. It also made use a single field to distinguish hard and soft constraints. Rewards/penalties for soft constraints are values +1/-1. The hard constraints must always satisfy.

There are three types Timetabling constraints:

- Unary constraints that bind a variable one or more possible values. Unary constraints can specify pre-assignments and/or preferences.
- Binary constraints that specify the difference between two variables. Binary constraints can specify sequence of two classes.
- K-nary constraints are all equal constraints that specify a number of variables must take the same values. K-nary all equal constraints can specify concurrency of a number of classes and time conflicts to avoid.

Structure of Genetic Algorithm

The general structure a genetic algorithm is:

Set Initial parameters (population size, number of generation, etc)
Initialize population
Evaluate every individual
While (not termination condition reached) {
    Form new individuals (the number depends on the reproduction strategy)
        Generate new individuals from previous population (*Selection* process)
        Recombine and Mutate (Crossover and Mutation)
        Evaluate new individuals
    Delete individuals in the population to make room for new individuals and so create the new population
    }
Report results

The GA maintains a population of potential solutions (chromosomes, strings or individuals). Each chromosome is evaluated using a specific fitness function that is constructed according to the specific problem. Then the chromosomes are selected according to fitness to start forming the next population (Selection operator). Two parents then recombine to form one or two offsprings, depending on the selection operator, by swapping corresponding parts of the parents (Crossover operator). Mutation is then used to randomly alter a chromosome position with very low probability. This modification in a chromosome injects variability to the population, trying to avoid a problem known as premature convergence which means that the entire population, after a number of generations has the same non-optimal value.

In order to solve a particular problem using Gas some things have to be taken into account when considering specific characteristics of the domain. Some of the things to bear in mind are [4]:

- Determine the chromosome to generate the initial population.
- Determine the fitness function that will evaluate each individual in the population.
- Select the specific genetic operators used to generate the new population
- Choose accordingly the different values of several parameters (population size, chromosome length, probabilities and so on).

Fitness function

In the penalty function approach, fitness evaluation function is a linear combination of accost function and a penalty function. A cost function measures the performance of a legal solution. In Timetabling problem, it estimates the degree of soft constraint violation. Penalty function measures the degree of hard constraint violation. In our constraint structure, the number of satisfied or violated equalities will be a good measure of distance from feasibility.

The overall constraint to a Timetabling problem is a conjunction of individual constraints overall reward or penalty to a chromosome is therefore a simple sum on the unit distance from feasibility calculated for each individual constraint.

In order to further differentiate chromosomes with little fitness difference, we applied a square function on the overall reward or penalty. We kept the sign of the overall reward or penalty unchanged to differentiate between reward and penalty.

The evaluation function as follows:

```
Function Evaluate
{
    For each individual constraint, do
    {
        check the nuber of equalities satisfied, n;
        calculate reward or penalty R as (n*constraint)
        cumulate R to total reward T (T+=R)
        }
    calculate chromosome fitness as (sign of T*T²);
}
```

If we represent a particular problem that having a chromosome with length equal to the number of *events* and each position with a value between 1 and t that is the number of *time-slots* available, for example [7, 9, 1, 3, 2] means that event 1 is schedule in slot 7, event 2 in slot 9 and so on. A specific weight is given to each as a constraint type in order to evaluate each chromosome using the fitness function. The fitness function used is [1]:

$$f(c,d) = \frac{1}{1+(\sum P_i m_i)} \qquad (1)$$

where c is a chromosome, d is the student data, m_i is the number of instances of a constraint violation of type i and P_i are the weights for each type of violation. The maximum value for this function is 1 when no constraint violation is detected and it gradually decreases as more constraints are violated.

Conclusion

These results immediately induce new directions of research for finding new approaches on representation, in which instead of looking for a particular solution for a specific problem, GAs are used for searching for a good algorithm that solve the problem.

We therefore choose a iterative approach. The GA is built with robust genetic operators and run parameters. Each schedule will be set to run for affixed number of generations. The best schedule found at that point will be reported for user review. If part of the schedule seems acceptable while others still violate many constraints, users can confirm the acceptable part. Users can re-adjust the constraint priorities (allowing less important constraints to be violated and leaving room or time-slots for more important constraints to be satisfied) or users can do nothing and let the system continue to search for some more generations. The above search, review and adjust process can be repeated until a satisfactory schedule is found.

References

1. Abramson D., Abela, A Parallel genetic algorithm for Solving the school Timetabling Problem., Royal Melbourne Institute of technology, 1991
2. Burke E., Ross P., Practice and Theory of Automated Timetabling. Lectures Notes In Computer Science Springer, Berlin 1996
3. Michalewicz Z., Janikow C., Handling constraints in genetic algorithms. In Proceeding of the 4th International Conference in Gas. Morgan Kauffman, 1991.
4. Michalewicz Z, Genetic Algorithms+ Data Structures = Evolution Programs, Springer Verlag, 1992.
5. Syswerda G. Uniform crossover in genetic algorithms. Proceeding of Third International Conference of Genetic algorithms
6. Corne D., Ogden J., Evolutionary Optimisation of Methodist Preaching Timetables, Selected papers from the Second International Conference on Practice and Theory of Automated Timetabling II, p.142-155, August 20-22 1997.
7. Goldberg D., Web courses, http://www.engr.uiuc.edu/OCEE, 2000