SOLVING TIMETABLELING PROBLEMS
USING GENETIC ALGORITHMS BASED
ON MINIMIZING CONFLICT HEURISTICS

Hitoshi Kanoh
Institute of Information Sciences and Electronics, University of Tsukuba
Tsukuba, Ibaraki, 305-8573, Japan
Email: kanoh@is.tsukuba.ac.jp,
web page: http://www.kslab.is.tsukuba.ac.jp/

M. Kondo, M. Sugimoto
College of Information Sciences,
Third Cluster of Colleges,
University of Tsukuba
Tsukuba, Ibaraki, 305-8573, Japan

Abstract. This paper describes techniques that can be applied to large-scale real-life employee timetabling problems. The problems are characterized as constraint satisfaction problems. The solution methodology uses genetic algorithms to minimize the total penalty for constraint violation. An encoding for search space reduction, genetic operation for a quick accumulation of partial solutions, and a penalty in fitness evaluation are implemented. An experiment involving 99 employees, 138 jobs and 27 working time slots showed that this timetable solution method is superior to the usual genetic algorithms and man-made schedules.

Key words: employee, timetabling, scheduling, genetic algorithm, constraint, virus, infection.

1 INTRODUCTION

Scheduling problems [1] that belong to the class of NP-hard problems are quite complex in nature and very difficult to solve using conventional optimization techniques, such as tabu search [2], simulated annealing [3] and genetic algorithms (GAs) [1, 4]. In this paper, we investigate real-life employee allocation problems. The problems are generally characterized as constraint satisfaction problems.

An employee-timetabling problem consists of dealing with a number of different factors, such as the number of employees, jobs and working time slots, and various constraints on the employees and jobs. The basic objective
in solving the problem is to allocate employees to working time slots while minimizing constraint violation. Some of the constraints used in our experiments were that employees should work more than three hours continuously and employees should be allocated at the desired time. The solution methodology we propose is based on minimizing the total penalty for constraint violation. Many versions of GAs have been applied successfully to real-life situations. The general approach in applying GAs to a problem is to use a GA to evolve an appropriate permutation and then to use a heuristic method to construct a subsequent solution according to the permutation [1]. This approach, however, needs heuristics that is dependent on individual problems. In this paper, we develop general techniques concerning genetic operation [5], fitness evaluation, and encoding that can be applied to a variety of large-scale real-life employee timetabling problems.

2 PROBLEMS

2.1 Formulation

The problems are defined using four terms: Problem = \{X, D, C, W\}, where X = \{E, T, J\}, E = \{e_1, \ldots, e_p\} is a set of employees, T = \{s_1, \ldots, s_q\} is a set of working time slots, J = \{j_1, \ldots, j_r\} is a set of jobs, D = \{p_1, \ldots, p_m\} is a set of priorities for allocation, C = \{c_1, \ldots, c_n\} is a set of constraints, and W = \{w_1, \ldots, w_n\} is a set of weights for the constraints, that is, penalties. In addition, each employee e has attributes such as name, position, qualification, the total time of contract a month, period of a paid vacation left, and preferred work hours.

2.2 Constraints

The fitness of a schedule is measured by the degree to which the schedule meets the constraints. The constraints to which a schedule has to adhere in order to be feasible are called definite constraints. We ensure that the definite constraints are not violated during the scheduling process, i.e., a mapping from chromosomes (genotype) in a coding space to schedules (phenotype) in a solution space. A different set of constraints, hard or soft constraints, incur a penalty that is incorporated into the evaluation function. Twenty five constraints are applied to the problems. The main constraints are shown below. Weights for hard constraints are \(90/N_{\text{max}}\), and the weights for soft constraints are shown in parentheses, where \(N_{\text{max}}\) is the maximum number of violations for the constraint.

(1) \textbf{Definite constraints}

- no employee could work more than eight hours a day.
- no employee could work during vacation.
(2) **Hard constraints**
- no time slot should be empty.
- employees should work more than two hours continuously.

(3) **Soft constraints**
- employees should be allocated at the desired time (5).
- two employees who have the same qualification should not work at the same time (10).

### 2.3 Strategy

There are numerous extensions to GAs that have been developed to help improve performance. We proposed the following strategies concerning encoding, genetic operation [5], and fitness evaluation.

1. We used a general heuristics to simplify the encoding. That is to say, employees working on the same day were allocated similar time slots. This can effect a reduction in the space to be searched.

2. Partial solutions are considered to be viruses, and a population of viruses is created in addition to a population of individuals. As a result of infection (a separate operation), the genes of a virus are substituted for those of an individual. This is aimed at a quick accumulation of partial solutions by using infection to improve the rate of search.

3. The fitness of an individual producing, by crossover, offspring with low fitness is decreased. This means that a penalty is imposed on parents producing bad offspring.

### 3 METHOD

#### 3.1 Encoding

We use a priority-based representation for encoding, that is, a gene on a chromosome represents the priority to allocate the next working time slot to an employee. We call it c-priority (connection priority) in the following.

Figure 1 shows chromosomes in the usual and proposed method. The first k numbers on both chromosomes are the initial values of c-priorities and the subsequent numbers are the c-priorities to allocate the next time slots to the employees on the timetable. The usual method is to directly encode \( \{X, D\} \) to the chromosome. It has as many sets of time slots as the number of jobs. In contrast, only one set of time slots is in the proposed chromosome. The c-priorities \( \{p_{k+1}, \ldots, p_{k+q}\} \) corresponding to the time slots \( \{s_1, \ldots, s_q\} \) on the proposed chromosome are used in all jobs \( \{j_1, \ldots, j_r\} \) repeatedly.
Figure 2 shows an example of encoding for the problem involved three employees, three time slots, and two jobs. The shaded cells on the timetable are working hours for the corresponding employees.

\[
\begin{array}{cccccc}
\text{X : Locus} & \text{D : Gene} \\
\begin{array}{cccccc}
\theta_1 & \ldots & \theta_3 & j_1 & \ldots & j_2 \\
S_1 & \ldots & S_3 & S_1 & \ldots & S_2 \\
p_1 & \ldots & p_3 & p_k+1 & \ldots & p_k+q \\
\end{array}
\end{array}
\]

(1) Usual method

\[
\begin{array}{cccccc}
\text{Locus} & \text{Gene} \\
\begin{array}{cccc}
\theta_1 & \ldots & \theta_3 & S_1 & \ldots & S_3 \\
p_1 & \ldots & p_k & p_k+1 & \ldots & p_k+q \\
\end{array}
\end{array}
\]

(2) Proposed method

Figura 1 – Chromosome.

<table>
<thead>
<tr>
<th>Tom</th>
<th>Jon</th>
<th>Merry</th>
<th>10-11</th>
<th>11-12</th>
<th>12-13</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

(1) Chromosome

<table>
<thead>
<tr>
<th>Employee</th>
<th>Initial value</th>
<th>Job 1</th>
<th>Job 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>2</td>
<td>10-11</td>
<td>11-12</td>
</tr>
<tr>
<td>Jon</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Merry</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>

(2) Timetable

Figura 2 – Example of encoding (shaded cells are working hours for corresponding employees).

3.2 Fitness evaluation

We use a dynamic penalty technique for the fitness evaluation. That is to say, the penalty ratio between the hard and soft constraints is adjusted according to the iteration number of the GA [1].

\[
\text{fitness} = W_{\text{hard}} \times (1 + a t) + W_{\text{soft}}
\]

where \(W_{\text{hard}} (W_{\text{soft}})\) is the sum of the weights of hard (soft) constraints that are not satisfied, \(t\) is a generation number, and \(a\) is a constant.

3.3 Algorithm

The minimal generation gap model was used for alternation of the generations. In this model, two individuals are replaced by crossover to produce each new generation and thus, two fitness evaluations are required.
between generations. In addition, let viruses be given and mutation not be
used in our method.

procedure main( ) {
    generate a population of individuals;
    input a population of viruses;
    evaluate the fitness of the individuals;
    repeat until the upper limit of generation{
        select two individuals as parents at random;
        single point crossover to produce offspring;
        re-evaluation( ); /* strategy (3) */
        select the elite from the parents and offspring and put it back in the
        population;
        select the individual from remaining parents and offspring by
        roulette wheel and put it back in the population;
        infection( );
    }
}

t

procedure infection( ) {
    select a virus at random from the population of viruses;
    select an individual at random from the population of individuals;
    substitute the genes of the virus for those of the individual;
    if ( the fitness of the individual decreases ) this infection is canceled;
}

4 EVALUATION

To evaluate the performance of the proposed methods, we applied them to
two real-life problems. Problem 1 is an employee timetabling problem at a
store in Tsukuba city involved seven employees, three jobs and 1612
working time slots; problem 2 is a student time tabling problem at a campus
festival in our university involved 99 employees, 138 jobs and 27 working
time slots. We assigned GA1 as the GA used in the first strategy (see section
3.3), and GA2 as the GA used in the second and third strategies. We applied
GA1 to problem 1 and GA2 to problem 2.

Figure 3 shows the experimental results. Except for the three strategies, the
usual GA is the same as the algorithm shown in section 3.3. The proposed
GAs give lower total penalties compared with man-made schedules, while
the usual GAs fall into evolutionary retardation at higher penalties.
5 CONCLUSION

In this paper, we solved employee timetabling problems using partial solutions, offspring’s fitness, and simplified encoding. Experiments using real-life problems showed that this approach is an effective solution method. This method can be applied to many scheduling problems. In our forthcoming study, we will combine this new method with a local search method and make a detailed comparison with other solutions.

![Figure 3 - Experimental results for real-life problems (Penalty = W_{hard} + W_{soft}).](image)

6 REFERENCES