TRAIN TIMETABLE GENERATION USING GENETIC ALGORITHMS

C. J. Hinde, M. S. Withall, I. W. Phillips
Department of Computer Science, Loughborough University, Loughborough, U.K.

T. W. Jackson
Department of Information Science, Loughborough University, Loughborough, U.K.

S. Brown, R. Watson
RWA Rail, RWA-Rail, Epinal Way, Loughborough, Leicestershire, U.K.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

Keywords: Railway timetabling, Evolutionary systems, Multiobjective optimisation.

Abstract: The scheduling of railway trains has been a research problem for many years. Many of the choices required are not known *a priori* and require exploration of the problem to determine them. A modular Genetic system was designedmake the evaluation function and preparation of the timetable tractable. The Genetic system consists of a Genome, split into Chromosomes so the extra choices that become known throughout the evolution can be added to the Chromosomes. A weighted fitness function and a multiobjective non-dominated fitness function were tried, and then partial objective ranking was added. The system has tackled a mixture of problems has produced promising results.

1 INTRODUCTION

Producing railway timetables has long been a difficult problem and along with many scheduling and timetabling problems is NP Complete (Ullman, 1975). There are many aspects to the problem and typically they have been solved as optimised subproblems. Actual rail timetables take of the order of 18 months and many railway planners.

Train timetabling, which determines leaving and arrival times along with other factors, has some interesting characteristics. The constraints may determine that some trains will leave a station at a fixed time. The duration of journeys between Tiplocs (timing point locations — places on the rail network such as stations and junctions) will be specified by a document known as the "Rules of the Plan", (ROTP). These are exactly what they state and determine many aspects of the emerging plan. For example it will take 11 minutes for a high speed train to move from Loughborough to Leicester.

There are decisions to make about train movements, such as which platform to arrive at where there are many possible platforms. There are also decisions that are not known at the start of the planning process and arise as a result of exploring the space of possible plans. An example of this is a train leaving a station at a set time and catching up with an earlier slower train, this is known as a "headway clash". There is now a choice whether to delay the start of the faster train so it does not interfere with the slower train, or delay the train en-route.

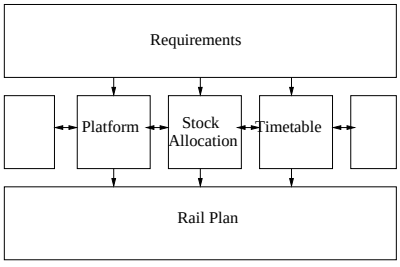

Figure 1: The basic timetabling system. This also shows that the subproblems are connected and cannot be optimised separately. Platform allocation cannot be performed until the train arrival and departure times are known.

J. Hinde C., S. Withall M., W. Phillips I., W. Jackson T., Brown S. and Watson R..
TRAIN TIMETABLE GENERATION USING GENETIC ALGORITHMS.
DOI: 10.5220/0003074801700175
In *Proceedings of the International Conference on Evolutionary Computation* (ICEC-2010), pages 170-175
ISBN: 978-989-8425-31-7
Copyright © 2010 SCITEPRESS (Science and Technology Publications, Lda.)

2 BASIC SYSTEM

The problem of railway plan generation includes several subproblems. Typically these have been solved in order with feedback taking place if a subsequent subproblem cannot be solved. This is due to the size of the problem and not to any intrinsic property of it. Figure 1 illustrates this, the three subproblems denoted are not exhaustive.

2.1 Subproblems

There are several subsystems. In order to test the interdependency mechanisms and to allow a single Genome to control the details of any plan produced. The three illustrated involve:

- Timetable delivers the times of arrival and departure, which parts of the track are to be used and which platforms each train on each journey at each station shall use. The tracks and platforms used are determined by the relevant Chromosome.
- Stock Allocation delivers the particular arrangement of rolling stock that make up a particular train.
- Platform subsystem checks that there are no platform clashes and that each platform used is able to service the trains that arrive to it and depart from it. The platform subsystem takes some of its input from the Timetable subsystem.

Each subproblem has its own requirements either from the specified requirements of the plan or because of the output from another subproblem further specifies the problem. For example, it is not possible to evaluate whether a platform clash exists until the arrival and departure times of the various trains have been determined. All the journeys required are known at the outset so it is possible to determine a stock allocation before timetabling, however it is not possible to determine a good stock allocation without knowledge of the train sequences and times. The modules are connected in 2 main ways, they depend on one another for some of their parameters; typically a timetable will be formed, followed by allocation of Rolling Stock and then a Crew will be allocated to the train. However, a change in the Timetable will affect which Rolling Stock are available, and also which Crews are available.

Early work on our timetabling problem tackled platform allocation in a major station using real arrival times and a specified platform resource (Clarke et al., 2009). Glasgow Central has nearly 1000 trains a day and the system resolved the allocation problem in under 30 seconds on a fairly modest PC.

This work was encouraging and similar exploratory work on timetabling after attending a course on rail timetabling was also encouraging. This resulted in modules that could address single problems such as the timetabling and platform allocation separately but not together. The attraction of being able to build the plan segments using modules which could be separately included, omitted, extended or modified was powerful. At this stage it was decided to separate the modules so that this was possible.

Having produced a timetabling module that did not encompass any Artificial Intelligence but which just loaded the trains into a schedule, plus a platform allocation module that applied a Genetic Algorithm to the problem, the next step was to develop an overall architecture which would support the whole problem. The idea is that the problem specific modules would have little or no intelligence but would be parameterised by the requirements file and a parameters file generated by the AI module. The requirements file would specify such things as “Train 1F01 leaves Sheffield at 10.00”, which would be part of the fixed requirements. The parameters would specify that “Train 1F01 leaves Sheffield from platform 1”; and would not be part of the fixed requirements but would be decided by the AI module.

At this point an optimisation framework exists, based currently on a GA that mixes fixed requirements with choices based on a Genome so the plan generation software responsible for allocating times, platforms and rolling stock would become determined. All that would then be required would be to feedback a utility measure so the AI module can improve its decisions, Figure 2.

There are several difficulties with this, while many of the decisions about the timetable are fixed and so may not be changed by the AI system, there are decisions that are known *a priori* such as which platform a train is to arrive on. It is relatively straightforward to determine which decisions are required at this level. However, it is possible that a train may leave at a time that results in it catching up an earlier slower train and as such violates the restrictions that determine how far apart trains on the same line must be; this is known as Headway. A problem arises where a train that can leave at any time, tries to leave at a time that results in a headway clash. There are two choices, either change the time the following train leaves, or change the time the followed train leaves. So decisions are required from the AI system that are not known *a priori*. The modules therefore have to feed back a list of options for the AI module to resolve, see Figure 2.

A simple example of this feedback is shown in Figure 3, which is the initial choices file, with Fig-

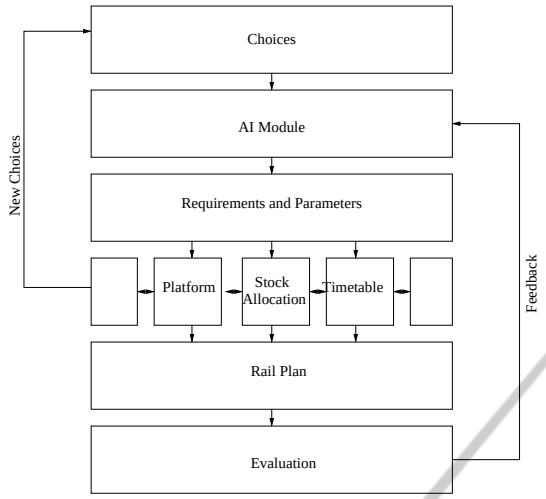

Figure 2: The modules feed back new choices required to resolve problems.

```
8 /* number of Genes */
10 /* number of choices for this Gene */
startingtime_p('4B01',
540
545
:
625
).
:
2
track_p('2C03',warob_jn,down,llanbart,
ufl
sl
).
```

Figure 3: A fragment of the initial choices file containing the patterns to be fed to the timetabler based on the values of the Genes.

ure 4 shows a typical fragment of a parameter file aimed at the timetabling module. The timetabling module is written in prolog and the control programs including the Genetic system are written in C. whereas Figure 5 shows a fragment of the choices file after a few generations.

```
startingtime_p('4B01',600).
dwelltime_p('1F01',staffrayner,2).
platform_p('1F01',down,dfl,watergalley,1).
:
track_p('2C02',llanbart_n_jn,up,llanbart,sl).
:
```

Figure 4: A fragment of the initial parameters file containing the actual values fed to the timetabler based on the values of the Genes.

```
14 /* extended from 8 Genes to 14 */
:
2
delaystartingtime_p('2C01',
0
3.5
).
:
```

Figure 5: A fragment of the final choices file containing the patterns to be fed to the timetabler based on the values of the Genes. Notice that there are now 14 Genes available to control the timetabler.

```
startingtime_p('4B01',600).
:
delaystartingtime_p('2C01',0).
:
```

Figure 6: A fragment of the final parameter file with the choices resolved.

Figure 4 shows the parameter file after 1 generation, it contains no added Genes but the values are in fact reasonable. Each Genome will have its own parameter file and will return a fitness vector appropriately based on the performance of the plan.

After a few generations the system has found several possible delays required due to headway clashes. Some of these headway clashes are resolvable through other means. The potential delay to '1F03' is due to the freight train '4B01' leaving at an inappropriate time and would not normally be considered by a human timetabler; however the system has tried to resolve this difficulty and come to the conclusion that it was not necessary as '4B01' could be scheduled to leave at a different time. Although this aspect of the problem was known by the human timetablers it may occur in any other timetabling problem and so should be considered.

Figure 6 is the final parameter file with the choices resolved. The main "troublemaker" is the time of departure of '4B01', which is a freight train. Unhelpful departure times have led the timetabler to request possible delays to several trains, however the only delay actually necessary if the freight train leaves at 600 (10.00AM) is a delay to '2C02'.

3 IMPLEMENTATION DETAILS

The system has been implemented in a variety of languages; whichever language seemed appropriate and which made implementation as fast and easy as possible consistent with obtaining a correct working system. The main AI system, a Chromosome based Ge-

netic Algorithm, was written in 'C', because one of the main tasks is to control the execution of the subprograms and other services. The Timetable and Platform subsystems are written in prolog.

4 EVOLUTIONARY SYSTEMS

The evolutionary system is responsible for making the choices in the timetabling system. A series of strategies were used in the experiments delivering various benefits. The structure and characteristics of the problem have influenced the design of the evolutionary system as the Chromosomes need to be extensible, and to influence different parts of the problem while being part of a whole solution.

4.1 Genome Structure

Each Genome has as many Chromosomes as there are subproblems. Each Chromosome may be any length depending on the subproblem addressed, as each subproblem may require more decisions to be made the separation into Chromosomes allows this to happen without destroying the interpretation of that Chromosome's Genes. The choices file determines how each Gene value is to be interpreted and determines the structure of the eventual parameters file that controls the connected subsystem.

4.1.1 Reproduction

Crossover is implemented using Uniform Crossover (Syswerda, 1989) and parent selection is by tournament. As Chromosomes may be different lengths, Uniform Crossover is performed up to the length of the shortest Chromosome followed by copying up to the length of the longest Chromosome. The choices file for that Chromosome has a maximum limit and the remainder of the Chromosome is selected randomly. An alternative strategy would be to terminate the child Chromosome, however this would result in no extensions to the Chromosome. Parents who are fitter than offspring are kept to their length and survive to the next generation. From a population of N parents delivering N offspring the new offspring are evaluated and the fittest of the 2N contenders are kept for the next generation.

4.2 Execution

Each member of the Genome population has a subdirectory containing a run time execution image of the relevant subproblem. The main control program writes the parameter file based on the choices file and the value of the Gene to the relevant directory and initiates the appropriate sub program. There is an order in which these subprograms need to be executed, for example the Platform subprogram does not have enough information until the Timetable has been formed. The control program is aware of this ordering and will not initiate a subprogram until all its predecessors have completed and provided the requisite additional parameters needed.

After all the subprograms have executed in the correct sequence, passing parameters between themselves as appropriate, the new required choices are collected together and added to the choices file. This is not quite straightforward as the choices may have already been added, and furthermore there may be additional options to an earlier choice. For example, there may be a headway clash between two trains, requiring the faster train to be delayed by 3 minutes. Once all decisions have been made this delay may be unnecessary so two options, 0 and 3 minutes, are given to the Genetic system. After the 3 minute delay is imposed, the faster train may catch up the slower further on; requiring a further delay, say 2 minutes. So the options would now be 0, 3 and 5 minutes.

5 FITNESS ASSESSMENT

These experiments are based on a tutorial example, used in the training of real train planners and designed to illustrate all major aspects of railway planning. The example problem has several passenger trains and finally a freight train that can leave at any time but does not stop until it reaches its destination. All that is required is that one Gene has sufficient options that it can represent all the potential departure times of the freight train. The major concern in this project is scalability and so times are relevant. As the eventual users of the system will not be experts in tuning evolutionary systems guaranteed convergence is also important. The experiments were run on a Macintosh MacPro dual 3.2GHz quad core machine running OSX 10.6. The software was written in a mixture of C and SWIPL Prolog. The software was also organised to be executed concurrently, as described above. All experiments were conducted with 7 different random number seeds and the results shown are the medians of those runs. The overall measures for the strategies are averages of the medians. All experiments were terminated after generation 1000, starting at generation 0, as having not converged, hence the values of 1001 generations on some entries.

The fitness strategies tested were as follows:

- A weighted sum of the fitness values.
- A fitness vector using non-dominated selection.
- A layered ranked fitness vector using non-dominated selection.

5.1 Weighted Sum

In order to establish a base line for the tests a simple weighted sum of the fitness values was used. As expected the system worked reasonably well, converging in a few seconds.

Table 1: The results of some experiments on population size with a weighted sum fitness vector.

Population	Time Seconds	Generations	Fitness Tests
5	77	489	2445
10	112	558	5580
20	38	135	2700
30	18	45	1350
40	12	23	920
50	13	21	1050
60	52	71	4260
70	15	17	1190
80	19	19	1520
90	16	15	1350
100	15	13	1300
110	24	19	2090
120	19	13	1560
Average	32.9	111	2101
Average > 40	21.6	23.5	1790

5.1.1 Problems and Analysis

The system as described executes reasonably quickly but is directed towards a weighted sum of fitness values that does not capture the requirements adequately. The major problem is that with a weighted fitness function there is a point where not delaying a train results in a headway clash; and if the sum of delays is sufficient to match the weighted headway clash we exchange a late train for a severely damaged one. Typically signalling would just delay the train to avoid a crash, but it is unwise to plan for a headway clash.

5.2 Multi Objective Non-dominated Fitness

Deb's (Deb et al., 2002) Non-Dominated Genetic algorithm potentially avoids the problem of weighting the undesirability of a headway clash against delayed trains. The fitness values used in these experiments were based on Train delays, Headway Clashes and Platform Clashes. The selection was based on non-dominated fitness vectors, duplicate solutions and finally, to resolve potential ambiguities, the number of other individuals in the population that were dominated by an individual. The fittest individuals would be non dominated, non duplicates that dominated other individuals. One individual in a set of duplicates was marked as a nonduplicate.

One benefit of this is that a set of pareto optimal solutions are generated, see (Deb et al., 2005).

Table 2: The results of some experiments on population size with a multi-objective non-dominated fitness vector.

Population	Time Seconds	Generations	Fitness Tests
5	160	1001	5005
10	206	1001	10010
20	5	18	360
30	8	21	630
40	486	1001	40040
50	9	15	750
…	…	…	…
Average	76	244	5128
Average > 40	15	15	1328

5.2.1 Problems and Analysis

This technique worked as far as described above. However, for a station with several platforms, each having equal access to the network, while there may be only one distinct solution to the platform allocation the numbering of the platforms does not affect the solution so given N platforms there are N! equivalent solutions. The first way in which this manifests itself is that one solution is found very quickly which has no delays but several headway clashes, there are many that are equivalent but not duplicates and this quickly exhausts the population.

A solution that delays a train and eliminates the headway clash would emerge as another pareto optimal solution, however the headway clashes are not always eliminated in 1 generation and this solution gets lost. Table 3 shows some of the times and generation at which a solution appeared.

The situation that arises is that a headway clash is detected requiring an X minutes delay to be applied. If this is applied it can mean that although that headway clash is eliminated, the following train is still travelling faster than the leading train and catches it later on. Another headway clash is generated which leaves the potentially optimal solution dominated by the solutions with no delays. A much larger population size in our test scenario allows the solution to

Table 3: Showing how the solution, once in the system, gets lost in subsequent generations. The solution appears at the nominated generation but then disappears.

Time Seconds	Generations
25	205
28	221
38	305
39	310
...	...

emerge, but smaller populations become locked into a local minimum, see Table 2.

So the non-dominated multi-objective fitness function has some instabilities but once the population size is sufficient to avoid those it is much faster. The instabilities in the system are undesirable and so this solution does not meet the needs of the problem.

5.3 Ranked Multi Objective Non-dominated Fitness

This solution was the fastest and gave the most satisfactory set of solutions by ranking the fitness functions, also reflecting the actual requirements of the solution. We care about delays, but not at the expense of a headway clash. Clashes are very important and so we search for solutions where the optimum is based on the important characteristics, and within those the less important characteristics. Our tests placed "headway clashes" and "platform clashes" at rank 1 and "train delays" at rank 2.

Table 4: The results of some experiments on population size using a ranked multi-objective non-dominated fitness vector.

Population	Time Seconds	Generations	Fitness Tests
10	10	64	640
20	8	30	600
30	10	26	780
40	8	17	680
50	14	23	1150
...	...	...	...
Average	17.0	43.7	1341
Average > 40	19.1	19.6	1670

5.3.1 Problems and Analysis

This system is fast, the set of solutions are pareto optimal and the instabilities are eliminated, see Table 4.

6 CONCLUSIONS

The paper has described a Genetic System for timetabling railway trains. The problem is such that the Genome was split into Chromosomes to allow more Genes to be added for each subproblem. The system runs the subproblems in parallel on as many processors as are available and runs quickly. The results are the optimum timetables for the problems and constraints given and the requirements are designed to cover most of the issues that occur in real timetabling. The architecture is extensible and can support various different problem breakdowns as appropriate. The system has moved from a simple weighted sum of the fitness functions, through a non-dominated fitness vector, to a non-dominated system that uses ranks fitness functions. This extension to Deb's system performs better than others on the problem of railway timetabling and reflects the objectives of the train planning problem more accurately.

SCITEPRESS
SCIENCE AND TECHNOLOGY PUBLICATIONS

REFERENCES

Clarke, M., Hinde, C., Withall, M., Jackson, T., Phillips, I., Brown, S., and Watson, R. (2009). Allocating railway platforms using a genetic algorithm. In *Proceedings of AI-2009, The Twenty-ninth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 421–434. SGAI.

Deb, K., Mohan, M., and Mishra, S. (2005). Evaluating the domination based multiobjective evolutionary algorithm for a quick computation of pareto-optimal solutions. *IEEE Transactions on Evolutionary Computation*, 13(4):501–525.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):181–197.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In Schaffer, J., editor, *Proceedings of Third International Conference on Genetic Algorithms*, pages 2–9, San Francisco, CA, USA. Morgan Kaufmann.

Ullman, J. (1975). NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393.