Creating Synthetic Test Data for Rail Design Tools – The Case of Linear Scheme Plans

Marek T. Jezinski[1], Markus Roggenbach[1], Monika Seisenberger[1], Victor Cai[2], and Fabio Caraffini[1]

[1] Swansea University, Swansea, Wales, UK
{m.t.jezinski, m.roggenbach, m.seisenberger, fabio.caraffini}@swansea.ac.uk
[2] Siemens Mobility Limited (UK), Chippenham, UK
victor.cai@siemens.com

Abstract. Many application domains resolve to the use of synthetic test data motivated by privacy concerns and safety reasons, but also, on the positive side, due to cost and quality considerations. Genetic Algorithms have long been studied for graph optimisation, however, to the best of our knowledge, not for scheme plan generation. We present a new approach that automatically constructs scheme plans from a set of tiles. This transforms the scheme plan generation problem into a combinatorial optimisation process. The manual design of scheme plans is laborious, costly, and in itself an error-prone process. Thus, there is a demand in the rail industry for synthetic scheme plans. All constructions are given. The runtimes achieved by our tool are presented.

Keywords: testing · genetic algorithms · scheme plans · decision tables

1 Introduction

The rail industry increasingly uses software tools, for example, to design scheme plans. However, how can such tools be tested, qualified, or certified? The manual design of rail artefacts such as scheme plans is laborious, costly, and in itself error-prone. Thus, there is a demand in the rail industry for synthetic test data.

We present a technique and a tool for generating artificial scheme plans. To this end, we use genetic algorithms (GAs) with controlled fault injection. Which faults ought to be injected is steered by decision tables. This allows for a systematic case analysis for different design rules. The generated scheme plans can then be supplied to rail design tools such as the Siemens' data checker [1], the OVADO tool [2], or Luteberget's Junction tool suite [3], and one can evaluate whether the rail design tool finds injected faults or reports faults that are not there.

Fault analysis through decision tables is a manual process that requires rail engineers to determine which scenarios to consider. In contrast, the following steps, namely test case generation, test execution, and test evaluation, can be automated. Thus, when decision tables are given, our approach leads to a fully automatic quality evaluation of check tools for scheme plans.

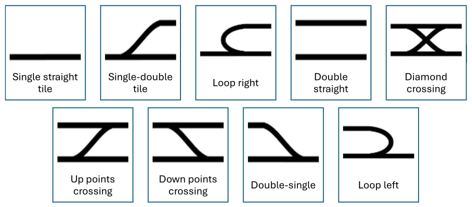
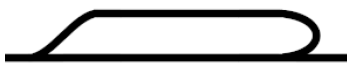
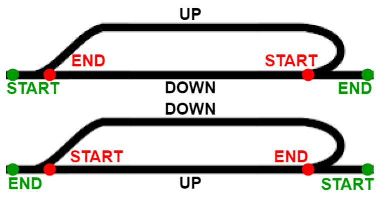


Fig. 1: Set of tiles available for generating linear, abstract scheme plans.



(a) Abstract scheme plan.



(b) Concrete scheme plans. Labels provide track start and end points with a direction label.

Fig. 2: An abstract scheme plan and two of its corresponding concrete examples; further variation comes from differing distances.

2 Scheme Plans and Design Rules

In the context of railways, a scheme plan is a graphical representation of a railway, depicting artefacts such as tracks, points, crossings, and line-side equipment, including balises (electronic beacons placed between the rails).

Following several discussions about testing coverage with industry representatives, we depicted elements of the track topology as tiles; see Fig. 1 for the tiles we have identified. A scheme plan is *linear* if it can be produced as a sequence of these tiles. For such a sequence, the following *adjacency rule* must hold for all pairs of consecutive tiles: the number of tracks at the right boundary of a given tile must be the same as the number of tracks at the left boundary of a subsequent tile. We call such a scheme plan *abstract*.

An abstract scheme plan can be turned into a *concrete* version by adding *lengths* (measured in metres) and *track directions* (DOWN and UP); see Fig. 2. The lengths are the distances from a reference point called *datum*, which in our case is to the left of the entire scheme plan. Track direction DOWN means that as the track progresses, the distances increase; on the contrary, UP denotes the distances decreasing, according to our industrial partner's conventions. One difference between the two concrete linear scheme plans in Fig. 2 (b) is that – imposed by the choices of UP and DOWN – start and end point of a line of track change position.

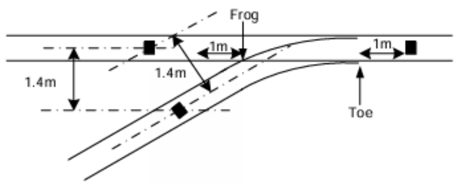
C1: Balises	F	F	T	T	T	T	T	T	T
C2: Direction	–	–	–	UP	UP	UP	DOWN	DOWN	DOWN
C3: Distance	–	–	–	$>$	$=$	$<$	$>$	$=$	$<$
C4: Points	F	T	F	T	T	T	T	T	T
A1: Correct	X	X	X	X	X		X	X	
A2: Incorrect						X			X

Table 1: Decision table reflecting the BG-03 rule.

2.1 Design Rules

In the railway domain, the placement of track equipment must meet strict layout requirements to ensure the safe operation of the signalling system. They can be vendor-specific, product-specific, or determined by local standard bodies.

Cai [4] studied and analysed a collection of 300+ design rules from our industrial partner Siemens. A subset of these is related to balise groups (BG). Cai identified five recurring patterns, some of which are local, some of which concern BGs separated by other infrastructure at larger distances. The rule **BG-03** is an example of a so-called local distance constraint. It specifies: *Spacing smaller than one metre between a single balise and a point toe constitutes a fault.*



3 Testing with Decision Tables

Decision tables allow one to describe and analyse complex logical relationships. Some claim that presenting decision procedures in tabular form goes back at least to ancient Babylon [5]. In computer science, an early documented use was in 1957 for programming [6]. They are at the core of Decision Table Based Testing, an established black-box testing technique [7].

Decision tables consist of a condition and an action part; see Table 1 for an example. The condition part analyses the logical relationships between input parameters while the action part expresses the expected outcome. The entry "-" stands for a don't care value, i.e., the decision is not influenced by the value of this parameter.

Our systematic case analysis for the design rule BG-03 led to the following conditions characterising families of concrete linear scheme plans:

C1: "Are there any balises existing in the scheme plan?"; this question can be answered with "T" (true) or "F" (false).

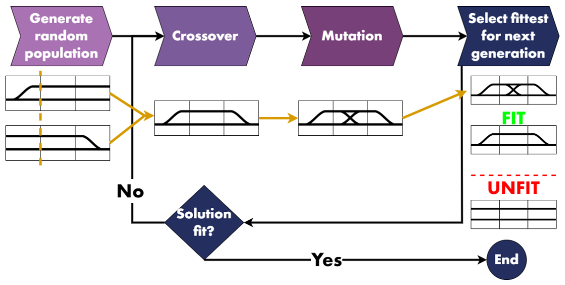

Generate random population
Crossover
Mutation
Select fittest for next generation
FIT
UNFIT
No
Solution fit?
Yes
End

Fig. 3: Diagram depicting Genetic Algorithm utilised for scheme plan generation; for simplicity, showing only one of the two children in the crossover step.

C2: "What is the direction of track on which the balise is placed?"; this question can be answered by one of the values "UP" or "DOWN".

C3: "How does the distance between balise and point toe relate to the 1 metre distance?"; this relation can be one of ">, "=", and "<"

C4: "Does the scheme plan include a point?"; this question can be answered with "T" (true) or "F" (false).

Depending on the answers to the questions C1 to C4, we know if a concrete scheme plan is adhering to BG-03 rule. For instance, if there is a balise in the scheme plan, the balise under discussion is on a track with direction DOWN, the balise is placed at a distance $<$ 1m to a point node (and, naturally, there is a point in the scheme plan), then the scheme plan is failing to obey the BG-03 design rule.

The idea of testing from such a decision table is to have one test case for each of its columns. The conditions provide values or a range of the input parameters, the action determines the expected outcome.

Though the answer to question C2 does not affect the expected output, it is useful to include this criterion: the system under test might carry out different calculations depending on track direction. This is, e.g., the case in Siemens' data checker. Condition C2 illustrates that decision tables for testing also ought to consider how the system under test could be implemented.

4 Generating Abstract Scheme Plans with GAs

Building on the work of Harrison [8], we implemented a tool in Python[3] that automatically generates abstract scheme plans of length k inspired by the GA framework [9]. This is a popular randomised search heuristic inspired by Darwin's natural selection, iteratively applying operators to evolve candidate solutions whose quality is evaluated by an objective function – commonly referred to as the fitness function – based on the optimisation problem to be solved. We selected

[3] We refrained from using standard GA libraries to achieve better performance.

this algorithm for its simplicity, versatility, ease of use without needing training materials, and low overhead. We interpret the standard GA notation as follows:

Gene: The gene pool consists of the tiles presented in Fig. 1.
Chromosome: A chromosome is an abstract linear scheme plan.
Population: A population is a set of abstract linear scheme plans.
Fitness: A fitness value, a real number $\in [0, 1]$, allows the evaluation of newly generated solutions tailored to meet the requirements.

Our tool works as shown in Fig. 3. First, a *random population* consisting of n scheme plans is generated. 'Parents' are selected using multiple tournament selections of size s, forming a mating pool of m solutions. Random pairs from this pool are selected 'without replacement' for crossover with a probability p_c set up by the user.

The *crossover* operation takes two linear abstract scheme plans of the same length $k \geq 2$ as input, say a sequence of tiles $p_1 = \langle s_0, \ldots, s_{k-1}\rangle$ and $p_2 = \langle t_0, \ldots, t_{k-1}\rangle$. It chooses a random value $0 \leq i < k - 1$ and produces the two new linear abstract scheme plans $c_1 = \langle s_0, \ldots, s_i, t_{i+1}, \ldots t_{k-1}\rangle$ and $c_2 = \langle t_0, \ldots, t_i, s_{i+1}, \ldots s_{k-1}\rangle$. c_1 and c_2 might be ill-formed according to the adjacency rule. Consequently, there is a post-processing repair function to ensure feasible solutions. This process checks if the tile at index $i + 1$ adheres to the adjacency rule. If not, it randomly replaces this tile with one that does.

Following this, c_1 and c_2 are mutated. The *mutation* operation takes a scheme plan $p = \langle s_0, \ldots, s_{k-1}\rangle$ as input, and has a probability p_m (set by the user) for each of its positions $0 \leq i \leq k - 1$, to change by randomly selecting a fitting tile t (a tile t is fitting if adjacency rule applies to both, s_{i-1} and t, if $i > 0$, as well as t and s_{i+1}, if $i < k - 1$), and returns a new scheme plan $m = \langle s_0, \ldots, s_{i-1}, t, s_{i+1}, \ldots s_{k-1}\rangle$. Such tile always exists, as our tiles in Fig. 1 cover all combinations of one and two tracks on the left and right boundaries. There is no need for repair at other positions, as we can presume p_1 and p_2 to be well-formed.

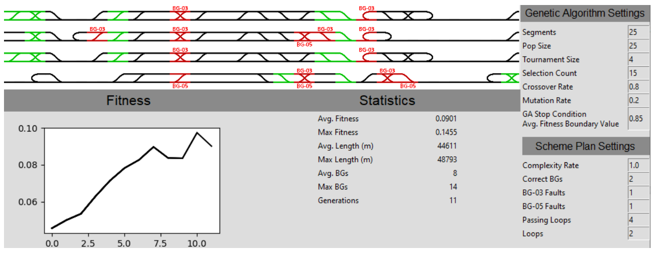
The newly generated solutions replace their parents in the population for the next iteration of the algorithm. Our tool terminates if the chosen *threshold* is reached.

Our *fitness functions* have a number of parameters. These include:

- the length $k \in \mathbb{N}$ of the abstract scheme plan and
- a rate $r \in \mathbb{Q}$ that determines the average expected complexity of its tiles.

The *fitness function* uses a table of values, which for each tile says how many points a tile includes and how many tracks a tile 'creates'. Here, we present an excerpt of this table for tiles from Fig. 1:

type	no. of new points *nPoints*	no. of new tracks *nTracks*
single-straight	0	0
diamond-crossing	4	2



Avg. Fitness	0.0901
Max Fitness	0.1455
Avg. Length (m)	44611
Max Length (m)	48793
Avg. BGs	8
Max BGs	14
Generations	11

Segments	25
Pop Size	25
Tournament Size	4
Selection Count	15
Crossover Rate	0.8
Mutation Rate	0.2
GA Stop Condition Avg. Fitness Boundary Value	0.85

Complexity Rate	1.0
Correct BGs	2
BG-03 Faults	1
BG-05 Faults	1
Passing Loops	4
Loops	2

Fig. 4: Generation of abstract, linear scheme plans for BG-03 rule. The fittest solutions are at the top. Snapshot of our tool during the converging process.

With the help of the complexity rate r, we define the ideal complexity of a plan consisting of k tiles to be

$$idealComplexity(k) = r * k.$$

Given a linear abstract scheme plan T, we define its complexity to be

$$complexity(T) = \sum_{i=0}^{k-1} nPoints(T_i) + nTracks(T_i).$$

Using *complexity* as a penalty term, we define:

$$coreFitness(T) = \frac{1}{1 + 0.05(idealComplexity(k) - complexity(T))}$$

Our experiments show that, e.g., with the empirically identified parameters of $r = 2.5$, $k = 25$, $s = 4$, $m = 15$, $p_c = 0.8$, $p_m = 0.2$, and $n = 25$, our GA implementation reaches a satisfactory threshold of average fitness equal to 0.85 in less than 10 iterations. This is a very fast process.

4.1 Creating Linear, Abstract Scheme Plans for BG-03

Testing for BG-03 utilising Table 1 requires the generation of nine test cases, one for each table column, except the first.

We define a colour scheme for the tiles from Fig. 1. A tile is *black*, if it does not have a balise, it is *green* if it has a correctly placed balise around a point, and *red* if it has a balise around a point placed incorrectly. For example, a single straight tile is black. In contrast, a single-double tile can take on all three colours.

Given a linear abstract scheme plan T based on coloured tiles, we can now determine the following natural numbers:

– $cBG(T)$ how many balise groups in it are correctly placed (i.e., how many green tiles it includes),
– $fBG(T)$ how many balise groups in it are wrongly placed (i.e., how many red tiles it includes), and

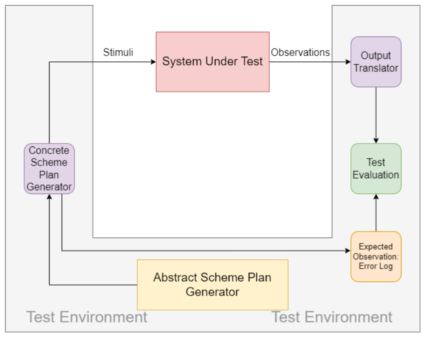


Fig. 5: Automated Test Environment

- $pL(T)$ how many passing loops it includes.

Consequently, we define a number of parameters for the ideal values:

- $icBG \in \mathbb{N}$, the ideal number of correct BGs,
- $ifBG \in \mathbb{N}$, the ideal number of BG-03 faults, and
- $ipL \in \mathbb{N}$, the ideal number of passing loops in the scheme plan, respectively.

With these, we define a function

$$deviation(T) = |icBG - cBG(T)| + |ifBG - fBG(T)| + |ipL - pL(T)|$$

Finally, we set the fitness function for testing for BG-03 to

$$BG03Fitness(T) = \frac{1}{1 + deviation(T)} * coreFitness(T).$$

Our experiments show that with the choice of $r = 2.5$, $k = 25$, $p_c = 0.8$, $p_m = 0.2$, and a population $n = 25$, our GA implementation reaches a threshold of average fitness equal to 0.85 in less than 100 iterations. p_c and p_m have been chosen based on the suggested rates available in [10], while the remaining values are an effect of experimentation. Fig. 4 shows our running tool with these parameters.

By choosing suitable parameters, we can guarantee the generation of scheme plans for all combinations of C1, C3, and C4. To guarantee property C2 concerning directions, we pass a parameter to the concrete scheme plan generation.

5 Generating Concrete Scheme Plans

Each abstract scheme plan corresponds to a set of concrete instantiations. This is the case, as we are adding lengths using random number generation; furthermore, we are adding directions according to the parameters passed. For instance, the length of a tile is given by a randomly chosen number between 800 and 1200 metres. The point in the single-double tile is placed at $\frac{1}{4}$ of the tile length. This data is stored in several text files, the structures of which are defined by Siemens' proprietary "RailDNA" schemas.

On the conceptual side, the generation of concrete scheme plans is more or less straightforward. However, it is a challenge to adhere to the specified input format of the system under test – as usual when it comes to testing. Notably, about 40% of our code base is related to concrete scheme plan generation.

6 Test Environment

Overall, we have established an automated test environment (see Fig. 5) for, in our case, Siemens' Data Checker – however, open also to other tools as mentioned in the Introduction. Using GAs and decision tables, we produce abstract scheme plans. These are then turned into concrete realisations, encoded in the bespoke input format of Data Checker, which is our system under test (SUT). Running Data Checker results in an error report. This actual result can then be compared with the expected result produced by the Concrete Scheme Plan Generator in the form of an error log.

For test evaluation, we say that a test is passed iff the SUT's reported errors are an exact match with the generated log file. This means that the SUT should report all errors as mentioned in the error log, and no errors beyond those mentioned in the error log. With our test environment, we could demonstrate that Siemens' data checker conforms to its specification w.r.t. several design rules, BG-03 as an example of a local one, as well as others spanning over several tiles. Therefore, our testing environment can be reused for other design rules from the aforementioned collection of 300+ rules (see: Section 2.1).

7 Related Work

Many application domains resolve to the use of synthetic test data. An example of this is Behjati et al. [11]. They were motivated by privacy concerns and safety reasons. Technically, they deal with test data (static and dynamic) in the form of table entries. Their SUT is an electronic national registry in Norway. In contrast, we are dealing with graphs. In the context of GAs, the generation of (finite) graphs is a long-investigated problem with standard solutions [9], [12]. Our tile graph encoding transforms the generation problem into a combinatorial optimisation process [13]. Although the rail industry is in need of automatically generated artificial scheme plans, to the best of our knowledge, we are the first to address this challenge.

8 Performance

At the heart of our approach is an combinatorial optimisation process realised as a Genetic Algorithm (GA). Our approach scales, as the following run-time data for the decision table for BG-03, see Table 1, demonstrates:

Scheme Plan Length	10	25	100	200	300	400
Runtime Abstract (in s)	0.10	0.80	2.60	8.60	34.40	115.10
Runtime Concrete (in s)	0.02	0.05	0.23	0.50	0.84	1.04

Runtimes are measured in isolation, i.e., either only for abstract scheme plan generation or only for concrete scheme plan generation. The overall runtime is dominated by the GA algorithm producing an abstract scheme plan. The time for concretising such an abstract scheme plan is negligible.

9 Conclusion

As a step toward tool certification and qualification in the rail industry [14], we have presented and implemented a new, automated testing approach. Our software considers railway tools as systems under test. These railway tools shall decide if scheme plans adhere to given design rules. Our approach establishes the correctness of such tools.

The test objective is to verify that the considered checking tool treats all design rules in a correct manner. To this end, we provide one test suite per design rule, that is, for each design rule we analyse - with the help of a decision table - which scheme plan 'types' to use as a challenge for the checker. We used our tool to produce 100+ scheme plans that we manually checked to verify that Siemens' Data Checker is working correctly. In all cases, the scheme plans were as expected, i.e., our tool produced synthetic test data as required by the decision table. All tests performed were passed, i.e., Siemens' data checker correctly classified the scheme plans to be either correct or incorrect.

While the testing process itself is fully automated, characterising suitable inputs relies on human ingenuity: the quality of our testing approach is as good as the analysis of the design rules, which results in decision tables.

Future Work. An alternative use of our tool would be to utilise it for random testing [15]. Furthermore, one could argue that design rules should rather be tested in combination, i.e., decision tables for single rules should be somehow 'merged'. Finally, scheme plans 'in the wild' take the shape of long stretches of linear implementations, combined at junctions and stations.

We can rightfully claim that this paper solves the case of linear scheme plans, leaving the case of junctions and stations for future work. Experiments are underway to demonstrate that our tiling approach can be extended to 'two-dimensional' scheme plans. To create scheme plans of varying lengths, we will explore replacing the current GA with other methods like Genetic Programming.

References

[1] M. Banerjee, V. Cai, S. Lakhsmanappa, et al. "A Tool-Chain for the Verification of Geographic Scheme Data". In: *RSSRail 2023, LNCS.* Vol. 14198. 2023.

[2] R. Abo and L. Voisin. "Formal implementation of data validation for railway safety-related systems with OVADO". In: *SEFM'13, LNCS*. Vol. 8368. 2014.

[3] B. Luteberget. "Automated Reasoning for Planning Railway Infrastructure". PhD thesis. 2019.

[4] Victor Cai. "Show Me How It's Wrong: Counterexample Visualisation in Static Railway Verification". MA thesis. Swansea University, Dec. 2023.

[5] LogicGem. *Decision Tables - LogicGem.* `https://logicgem.com/decision-tables/`. [Accessed: 2025-06-06].

[6] Jan Vanthienen. "The History of Modeling Decisions using Tables (Part 1)". In: *Business Rules Journal* 13.2 (Feb. 2012). `https://www.brcommunity.com/articles.php?id=b637`. [Accessed: 2025-06-06].

[7] Xin Feng, David Lorge Parnas, T.H. Tse, et al. " A Comparison of Tabular Expression-Based Testing Strategies ". In: *IEEE Transactions on Software Engineering* 37.05 (Sept. 2011). URL: `https://doi.ieeecomputersociety.org/10.1109/TSE.2011.78`.

[8] Thomas Harrison. "Exploring the Feasibility of Using Genetic Algorithms for Generating Railway Map Test Data". Bachelor's Thesis. Swansea University, Apr. 2024.

[9] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1st. USA: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN: 0201157675.

[10] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. 2nd ed. Springer, 2015, p. 100.

[11] Razieh Behjati, Erik Arisholm, Margrethe Bedregal, et al. "Synthetic Test Data Generation Using Recurrent Neural Networks: A Position Paper". In: *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. 2019, pp. 22–27. DOI: `10.1109/RAISE.2019.00012`.

[12] Qiaoyan Yang and Qinghong Zeng. "Application of Genetic Algorithms in Graph Theory and Optimization". In: *2016 3rd International Conference on Materials Engineering, Manufacturing Technology and Control*. Atlantis Press. 2016, pp. 24–29.

[13] Bushra Alhijawi and Arafat Awajan. "Genetic algorithms: Theory, genetic operators, solutions, and applications". In: *Evolutionary Intelligence* 17.3 (2024), pp. 1245–1256.

[14] Jean-Louis Boulanger. "Tool qualification". In: *CENELEC 50128 and IEC 62279 Standards* (2015), pp. 287–308.

[15] Joe W Duran and Simeon C Ntafos. "An evaluation of random testing". In: *IEEE transactions on Software Engineering* 4 (2009), pp. 438–444.