An Approach from AI to the Design of Routes in a Railway Interlocking*

E. Roanes-Lozano[†], L. M. Laita[‡]

[†] Dept. Algebra, Universidad Complutense de Madrid,
Edificio "La Almudena", c/ Rector Royo Villanova s/n, 28040-Madrid, Spain
[§] Dept. Artificial Intelligence, Universidad Politécnica de Madrid,
Campus de Montegancedo, Boadilla del Monte, 28660-Madrid, Spain

EXTENDED ABSTRACT

Abstract

An expert system devoted to the design of routes in a railway interlocking is presented in this article. It is able to extract knowledge from the given data (propose new routes and give information about them: number of sections, estimated time) and to verify the safety of proposed new route (interlocking). It takes advantage of the the powerful handling of lists provided by the computer algebra system Maple 6.

Once the topology of the network is introduced, the user can ask the expert system to exhaustively determine all routes departing from a given section and also all routes that are extension of a given one. In both cases it is possible to ask the system to return only those routes that end in a given goal section. The system orders the routes accordingly to their length (in number of sections). Moreover, if an approximated time is associated to each section (these times can be calculated taken into consideration their physical length, speed limitations...), the system can return the theoretical time corresponding to each possible route. Therefore it is possible to optimise the selection of the route w.r.t. different criteria.

Finally, the verification of the proposal, i.e. the compatibility of routes (the task of the interlocking system) is trivial with this data structure: it is enough to check that every pair of routes (of the routes to be authorised at the same time) don't share any common section.

*Partially supported by projects DGES PB96-0098-C04-03 and DGES PB96-0098-C04-01 (Spain).

[†]eroanes@eucmos.sim.ucm.es

[‡]laita@fi.upm.es

1 Introduction

There are two ways to study an interlocking by computer. In one of them the interlocking is interpreted as a logic problem of accessibility to sections. Before allowing a new situation of signals and turnouts, it must be presented to the system to be authorised, and the system has to decide in real time upon its security.

The most sophisticated approaches that we know work this way and are independent from the topology of the station. But, as far as we know, up to now they are only theoretical models (Montigel, 1994; Hansen, 1996, Roanes & Laita 1996a; Roanes & Laita, 1998). This approach has the disadvantage that the complexity of the problem (related to its exhaustive nature) makes that only small or medium size installations can be treated. The advantage comes also from its exhaustive nature: they are absolutely safe.

In real installations, because of the problem mentioned above and, possibly, also because of tradition (this is the way mechanical interlockings were designed), routes are established first and compatibility is checked during the design stage (Siemens, 1988). Checking the compatibility of routes was usually done manually. The interlocking allows only the simultaneous authorisation of routes previously declared as compatible. For instance in 1993 the Spanish railway company (Renfe) developed, in cooperation with other companies (Villamandos, 1993), a computer based tool to automatically check routes compatibility.

2 The Approach

2.1 Search for routes

The later approach is the one developed in this article.

The topology of the station is given as a list of lists of three elements (one for each turnout). These three elements are (in this order) section in the point side, section in direct route, section in diverted route.

The routes are stored as lists of sections.

Two auxiliary procedures have been implemented: `last(l)`, `lput(el,l)` and `selec_goal(l,g)`. Procedure `last(l)` returns the last element of list `l`. Procedure `lput(el,l)` returns a list formed by including element `el` at the end of list `l`. Procedure `selec_goal(l,g)` returns a list, constructed by selecting, from the list of lists `l`, all the elements (lists) which last element is `g`.

Let us describe the main procedures:

- `nextsect(i_l,t_l)`
  Input: list of routes, list of turnouts.
  Algorithm: For each route in `i_l` it creates a list of sections accessible from the last section of the route (sections that are not already in the route, to avoid repetitive forward and backwards movements), according

to the information provided by `t_l`. The search is exhaustive.
Output: A list of lists with the 1-section possible extensions.

- `nextlist(i_l,t_l)`
  Input: list of routes, list of turnouts.
  Algorithm: For each route in `i_l` it creates the possible routes extension of that one by adding the sections calculated by `nextsect`.
  Output: A list of lists with the possible routes extended one section.
  Remark 1: Let us observe that, if applied recursively to a route, `nextlist` returns the routes with 1 section more than the given one, 2 sections more, 3 sections more... that are extensions of the given one. The last time it returns the empty list (what means that there are no routes with that number of sections, what always finally happens as repeating a section is not allowed).
  Remark 2: The complexity of this procedure is the highest one (the growth of the number of routes is exponential with base 2 in the worse case). An upper boundary is 2 to the maximum of the lengths of routes. The worse case is clearly reachable for a binary-tree-like network (what is a common shape e.g. of marshalling yards).

- `tuti_ite(i_l,t_l)`
  Input: list of routes, list of turnouts.
  Algorithm: Applies `nextlist` recursively to the given route until the empty list is obtained. This routes (lists) are grouped in lists (according to the number of sections) and this lists of lists are grouped in a list.
  Output: A list of lists of lists (routes) with the possible extension routes.

- `tuti_term(i_l,t_l,ob_)`
  Input: list of routes, list of turnouts, goal section.
  Algorithm: Similar to `tuti_ite`, at each step only the routes that end in the goal section are left.
  Output: As `tuti_ite`, but all routes end in the goal section.

There is an auxiliary procedure `clasi(iti_)` that presents on the screen the given list of routes according to the number of sections in it.

2.2 Time Calculi

More data have to be given now: the list of sections and a list of times (corresponding to the sections in the other list, respectively).

There is an auxiliary procedure: `cal_ti_aux(l_s,t_s,l_)` that substitutes the i-th element of list `l_s` by the i-th element of list `t_s` (for i ranging along the whole list `l_s`) in list `l_`.

The main procedure is `cal_ti`.

- `cal_ti(l_s,t_s,l_)`
  Input: list of sections, list of times, list of lists of routes.
  Algorithm: Uses `cal_ti_aux` to substitute sections by corresponding times

and then adds the times for the sections in each route.
Output: A list of lists with the times corresponding to the given list of lists of routes.

2.3 Study of Compatibility of routes

The procedure is named `compatible`.

- `compatible(it_,itau_)`
  Input: list of sections, list of lists of routes already declared compatible.
  Algorithm: Converts `itau_` into a set (to avoid repetitions) and checks if the resulting set is disjoint with `it_` (converted to set).
  Output: "Authorised" or "Non-Authorised".

2.4 Remarks

Maple 6 was chosen because the use of lists is very convenient and powerful. For instance the code commented above occupies altogether a couple of pages. Nevertheless this package could also be implemented in a classic language (like C). This should be done for speed reasons if a real application was to be finally developed.

3 Future Development

This work is part of a global environment still under development. It was initiated with the development of computer based topology independent railway interlockings (Roanes & Laita 1996a; Roanes & Laita, 1998). These were not related to establishing routes but directly giving turnouts and signals positions. The work presented here is based in the previous study of routes.

The third part of the work would be the development of GUI. It should be able to allow the user to draw with the mouse the station at the initial stage (locating turnouts and signals) and to introduce the position of trains, turnouts and signals (also with the mouse) during the use stage. This GUI should offer to work either using routes or directly giving turnouts and signals positions and would call the corresponding inference engine accordingly.

It would be better to translate this model and that in (Roanes & Laita 1996a) to C for speed reasons. Then it should be decided weather to use the Gröbner bases based model (Roanes & Laita 1996a) or model (Roanes & Laita, 1998) for the case of directly giving turnouts and signals positions.

4 Conclusions

Computer algebra systems provides an environment that allows to develop quickly and easily complicated AI tools. We think the one presented here is a very interesting application to transportaton technology.