## Legend pattern calculation for dynamic traffic management using ILP

Verbakel, J.J.; van Meurs, Jeroen; van de Mortel-Fronczak, J.M.; Fokkink, Wan; Rooda, J.E.

***published in***
2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)
2023

***DOI (link to publisher)***
10.1109/CASE56687.2023.10260524

***document version***
Publisher's PDF, also known as Version of record

# Legend pattern calculation for dynamic traffic management using ILP

J.J. Verbakel[1], J. van Meurs[1], J.M. van de Mortel-Fronczak[1], W.J. Fokkink[2], J.E. Rooda[1]

2023 IEEE 19th International Conference on Automation Science and Engineering (CASE) | 979-8-3503-2069-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/CASE56687.2023.10260524

***Abstract*— To ensure the safety of highway traffic, traffic management systems are used. In the Netherlands, a traffic management system is used consisting of roadside units, which are controlled from traffic centers. The roadside units are capable of showing legends, such as speed limits or directional instructions. Operators are monitoring the road 24/7, and during roadworks or when an incident happens, they can request a measure, e.g., a lane closure. The system then automatically determines the required legend pattern, comprising of directional arrows to lead traffic away from the closed lanes and speed limits on the remaining open lanes. This system is managed by Rijkswaterstaat, which is part of the Dutch ministry of infrastructure and water management.**

**In this paper, integer linear programming is used to determine the legend pattern, based on operator input. The model adheres to the traffic rules used by Rijkswaterstaat. By using a model-based optimization approach, a solution is found which guarantees legend patterns follow the rules, while minimizing the impact on traffic. The model can be easily adapted to try out different road configurations or alternative rules. Furthermore, a graphical user interface is created for validation of the generated solutions, by the operators from Rijkswaterstaat. This model is also applicable to highways that are not equipped with physical signaling equipment. Therefore, it can be used to control two thirds of the Dutch highway network, where the current system is not implemented.**

## I. Introduction

Over the past decades, in many countries, highway usage has increased vastly. To ensure that this does not impair safety, traffic management systems (TMSs) are used. Two categories of TMSs are defined in [1]: infrastructure-based and infrastructure-free. Infrastructure-based TMSs make use of infrastructure, such as traffic lights and roadside units (RSUs) to monitor and control traffic. Infrastructure-free TMSs use only vehicle-to-vehicle communication.

Infrastructure-based TMSs often revolve around variable speed limits (VSLs) and ramp metering. For VSLs, the maximum is dynamically changed based on traffic intensity. An overview of different VSL strategies can be found in [2]. Infrastructure-based TMSs, as discussed in see [3], are used in various European countries, such as Spain and the Netherlands. In Barcelona there is an experimental highway setup, where VSLs can be tested, as was done in, e.g., [4]. For ramp metering, vehicles are queued on the on-ramps using traffic lights, to decrease occupancy on the main road, see e.g., [5].

This work is supported by Rijkswaterstaat, part of the Dutch Ministry of Infrastructure and Water Management.

[1] Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands j.j.verbakel@tue.nl

[2] Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, the Netherlands

In the Netherlands, around a third of the highways is equipped with an infrastructure-based TMS. This TMS is managed by Rijkswaterstaat, which is part of the ministry of infrastructure and water management. The TMS contains around 6000 RSUs which are controlled from 5 traffic centers. Each RSU has a gantry on which one or more matrix signal indicators (MSIs) are mounted. MSIs are used to show legends, such as speed limits and red crosses or directional arrows, which guide traffic around closed lanes safely, e.g., during roadworks or in case of an accident. These legends are initiated either by an operator through a measure request, or automatically, when an RSU detects slow moving traffic. An investigation by [6] shows that this system reduces the number of incidents, and decreases congestion. In [7], the benefits of a similar TMS on a US highway are shown.

When the operator requests a measure, resulting legends are computed automatically by the Motorway Traffic Management system (MTM). For example, when a lane closure is requested, red crosses are placed on that lane, and a pattern of directional arrows guiding the traffic away from that lane is added. In addition, a speed limit is shown on the remaining open lanes. To ensure safe and gradual deceleration of traffic, a series of successively stricter speed limits is shown upstream. Rijkswaterstaat has defined a set of rules that describe what legend pattern should be produced upon a measure request.

The calculation of legend patterns brings to mind solving a sudoku, where each square that is filled further constrains other squares. In the literature, sudokus are often solved using satisfiability, see e.g. [8]. For satisfiability, the problem is expressed as a logical equation over a set of variables, and the solution is either an assignment of variables that satisfies the equation or a proof that no such assignment exists. However, closing off the entire highway is always an allowed solution, even when less strict measures would also provide a safe situation and are therefore preferable. Therefore, some sort of optimization is required.

In this paper, a formal model of the legend calculation process is defined using integer linear programming (ILP). The model was made in close cooperation with Rijkswaterstaat. It guarantees that the solution adheres to all traffic rules, even in complex road topologies. By using a model-based approach, the process can be simulated to validate the current rules or quickly try out new rules. Moreover, it is possible to explain the legend pattern that is produced by the algorithm, which is currently not possible.

This work is part of a larger project in which the entire TMS is modeled. In [9], a discrete-event model of an RSU was made, which is used for supervisor synthesis.

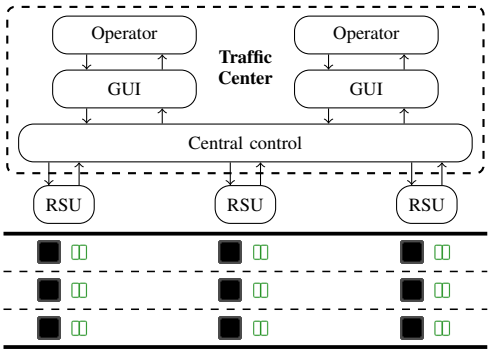


Fig. 1: Traffic management system structure.

The synthesis algorithm produces a correct-by-construction supervisor, from which controller code can be generated automatically. A method to efficiently create supervisors for many similar systems using a configurator is introduced in [10]. The configurator is used for RSUs, which comprise a few components in many different configurations.

This paper shows that formal methods can be used in real-life applications. The model presented here can be used to compute legend patterns for large traffic networks within a relatively short period of time. Furthermore, when the network is not yet equipped with MSIs, this method can also be used to generate legend patterns for in-car signaling. That way, the model can be employed on two thirds of the Dutch highway network which are not signaled at the moment. Rijkswaterstaat is considering to implement the approach put forward in this paper for the entire Dutch highway network in years to come.

There are numerous examples of (mixed) ILP applied to traffic control problems. Already in 1966, in [11] an ILP formulation of traffic signal synchronization is given. In [12], ILP is compared to a genetic algorithm and simulated annealing for an air traffic management problem and shown to be faster than the other two methods. In [13], an ILP mode is used to control autonomous vehicles crossing an intersection. In [14], a mixed ILP formulation is used to model traffic flow on a highway network, which is then used for model predictive control of a metered on-ramp. In [15], a mixed ILP model of a rail traffic management system is used to minimize system delays.

The remainder of this paper is structured as follows. In Section II, the main components and processes of a TMS are explained. In Section III, the model for the legend calculation problem is given. In Section IV, the implementation used in this work is introduced. In Section V, the results of a few requests are given, including some notes on how to interpret these results. In Section VI, the benefits and drawbacks of this method are discussed. In Section VII, concluding remarks and possible future research directions are given.

## II. System

A TMS is structured as given in Fig. 1. At the top are one or more operators, that monitor traffic and can request measures through a GUI. This GUI sends requests to the central system (CS), which computes the legend pattern. Next, the operator checks whether the legend pattern is correct. Last, each row of the legend pattern is sent to an RSU, which shows the legends on the MSIs. At the same time, the RSUs measure traffic speed and send a message to the CS if there is slow moving traffic. These messages are used to automatically place speed limits upstream from this traffic, without operator intervention. To alleviate the computational overhead of communication for the CS, a separate communication system is used, which handles the communication with RSUs.

### A. Components

The GUI provides a schematic overview of the road to the operator. The operator can request measures and see status information of MSIs, RSUs, and other network components. Each operator in a traffic center has their own GUI, which communicate concurrently with a single CS.

The CS is the brain of the system, it processes measure requests from the operators, and computes speed limits based on traffic data. When the operator requests a measure, the CS computes all legends, i.e., the legends of the measure itself, a lead-in pattern of directional arrows, to guide the traffic away from closed lanes, speed limits on the surrounding MSIs and end-of-restrictions after the measure. In addition, the speeds should not differ too much between adjacent MSIs to ensure smooth and safe traffic flows. When an RSU signals slow traffic, a similar process starts. However, these patterns are easier, since no lanes are closed, and because end-of-restrictions is not applied for these (often short-lived) measures. After computing the legend pattern, the CS sends the legends for each to the corresponding RSU.

An RSU processes vehicle speeds, measured through detector loops in the road surface. When slow traffic is detected, a speed recommendation is sent to the CS. If the RSU receives legends from the CS, it also checks that a (locally) consistent legend pattern is shown, e.g., a directional arrow should not point towards a closed lane. If the local pattern is consistent, it is shown on the MSIs, otherwise the CS is notified. As such, the CS always knows the currently shown legend on the MSIs.

### B. Process of requesting a measure

The process of requesting a measure consists of several steps. The operator requests a functional measure, such as closing a lane from an RSU to another RSU. In addition to the measure request, some options can be set, such as reduced speeds alongside crosses (50 instead of 70) and the use of red rings around speeds, emphasizing that it is a maximum speed. This request is sent to the CS, which computes the legend pattern, and returns the found solution. Next, the operator checks if it is correct and makes changes if it is not. Then, the current state of the RSUs is checked, and the measure is merged with already active legend patterns. This is again sent to the operator for verification. Last, if the operator is satisfied with the solution, it is sent to the RSUs, which show the measure if it is consistent.

### C. Rules

The resulting legend patterns must adhere to certain rules (called 'business rules' by Rijkswaterstaat), e.g., a cross must be preceded by an arrow or another cross. After careful study of the documentation of the current system ([16], not public), 43 rules were formulated. These rules were verified in close cooperation with experts from Rijkswaterstaat. The rules can be broadly separated in three types: directional rules, speed rules and other rules.

Directional rules pertain to placing crosses and directional arrows. They ensure traffic frees up the closed lanes, without encountering unexpected crosses. For multiple closed lanes, this is done by creating a 'wedge', i.e., a pattern of crosses and arrows, closing of the lanes one by one, as traffic gets closer to the incident.

Speed rules prevent traffic from suddenly encountering large decreases in traffic speed, by leading in lower speeds incrementally. Furthermore, speed rules ensure a consistent speed limit along the road laterally, without large differences between adjacent carriageways.

There are some rules that do not fit the above types. Such as adding end-of-restrictions legends after the measure, and rules regarding rush-hour lanes.

### D. Current implementation

The MTM is a sequential, rule-based system, meaning that legend patterns are calculated in a fixed order. First, all RSUs that are involved in the functional measure are listed, e.g., the RSUs along the closed lane. Next, legends are assigned to the MSIs, RSU by RSU, assuming all MSIs are blank. If there are any relations to secondary RSUs, these RSUs are added to the list. Once the list is empty, end-of-restrictions is applied to downstream RSUs. Afterwards, legend restrictions, such as maximum speed limits, are applied. Sometimes, in case of special arrow patterns, a second pass is required.

After computing the legend pattern for a blank road, the operator checks the result. Then, the measure is combined with the legend pattern that is currently shown on the road; and red rings are applied if necessary. Last, some safety checks are done, e.g., there is no arrow pointing towards a cross. If there are severe problems, the measure request is ended without placing the request. Otherwise, the measure is returned with possible warnings about minor problems, such as an end-of-restrictions legend directly followed by a non-blank MSI. The operator can then decide to effectuate the measure and to send it to the RSUs.

Since the traffic rules are hard-coded, it is difficult to change them. In addition, there is no direct linkage between the textual rule and the implemented code. In certain cases, combining the measure with an existing pattern fails, which is not uncommon for complex topologies. The implementation is tightly interwoven with the communication system, which makes it hard to simulate a measure or see what results would be obtained on a new configuration.

To improve on these issues, a model-based approach is proposed in this paper. By translating the business rules to a mathematical language, there is no ambiguity on their meaning. Moreover, the use of an optimization technique ensures that the best solution is obtained. Also, simulations can be used to quickly try out new rules or configurations.

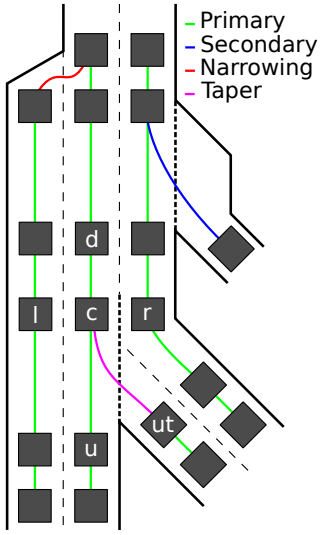


Fig. 2: Relative positioning of MSIs.

## III. Model

The model for the legend calculation problem is given in this section. First, ILP is discussed. To concisely represent constraints, a specific notation is introduced. Next, the goal function and different types of constraints that comprise the model are discussed using illustrative examples. Last, some notes on the size and scalability of the model are given. The model presented in this section is based on [17].

### A. Integer linear programming

To model the legend pattern calculation, Binary ILP is used. It is an optimization problem where all variables are binary, i.e. either zero or one, and the objective function and constraints are linear. In general, a Binary ILP problem is formulated as:

$$\min_{\mathbf{p}\in\{0,1\}^n} \mathbf{c}^\top \cdot \mathbf{p}$$
$$\text{s.t. } A \cdot \mathbf{p} \leq \mathbf{b}$$

Here $\mathbf{p}$ is a vector of design variables, vector $\mathbf{c}$ is the cost for each design variable, and matrix $A$ and vector $\mathbf{b}$ define the constraints. In this work, variables represent either showing (1) or not showing (0) a specific legend. $A$ and $\mathbf{b}$ are never explicitly used, but both are very sparse, which allows for efficient computations, even on large systems. The cost denotes the impact on traffic and the constraints encode the business rules.

### B. Notation

Every business rule, and subsequently each ILP constraint, only refers to legends from nearby MSIs. Therefore, relative positioning is used, as shown in Fig. 2. All positions are

TABLE I: Legend variables.

| Name | Legend | letter | restrictivity |
|---|---|---|---|
| Cross |  | $x$ | 210 |
| Right arrow |  | $r$ | 209 |
| Left arrow |  | $l$ | 208 |
| Speed 50 |  | $e$ | 16 |
| Speed 70 |  | $g$ | 13 |
| Speed 80 |  | $h$ | 10 |
| Speed 90 |  | $i$ | 7 |
| Speed 100 |  | $j$ | 4 |
| Speeds above 100 |  | $k$ | 3 |
| Green arrow |  | $y$ | 2 |
| End-of-restrictions |  | $z$ | 1 |
| Blank |  | | |

determined from the viewpoint of an MSI $c$. The MSIs directly upstream and downstream from $c$ are called $u$ and $d$, respectively. Similarly, the neighboring MSIs are $l$ and $r$. Further MSIs on the row are not given a variable. The MSIs on a row are grouped into one or more traffic streams, comprising adjacent MSIs with a similar speed. For every secondary relation, there is an upstream and a downstream position.

Every legend that can be shown on an MSI has a binary variable, denoted by a letter, as given in Table I. These variables are indexed by the relative position of the MSI, so $x_c$ is used to denote a cross legend on this MSI, and $i_r$ denotes a speed 90 legend on the right-hand MSI. There is a special variable $x_c^{rhl}$ for a cross on a rush-hour lane, because a closed rush-hour lane should show a cross, without all the surrounding legends. Arrows and speed legends can also be shown with flashers on the corners of the MSI, this is denoted by variable $a$. Similarly, speed legends can have a red ring around them, which is denoted by $b$.

Often, a constraint refers a set of legends, in ILP this is modeled as a sum of the legend variables. To compactly represent such a set of legends, square brackets are used. For example, a cross or arrow legend at MSI $c$, normally denoted $x_c + r_c + l_c$, is represented as $[xrl]_c$. The special variables are never included between brackets, for clarity.

### C. Goal

The solution of the ILP problem should minimize the impact on traffic flow. To this end, each legend is assigned a restrictivity, representing its impact on traffic. Not showing any legend, i.e., blank, has a restrictivity of 0, all subsequent legends in order of their impact get a restrictivity of one higher. These values are shown in Table I. Note that there are gaps of 3 between speed values, because legends with variation flashers or a red ring have their restrictivity increased by one, resp. two. For example, speed 90 has restrictivity 7 normally, restrictivity 8 with flashers, and restrictivity 9 with a red ring. The gap between left arrow and speed 50 is there to prevent showing directional arrows instead of speed 50, which shortens the lead-in pattern. The goal function is the sum of the restrictivity of all shown legends.

### D. Constraints

Each of the business rules is translated into one or more constraints. If an MSI $c$ has all the relations that are mentioned in the constraint, then the constraint is added. This is done for every MSI in the model. Below, several different types of constraints that are used in the model are given.

Most constraints are a combination of legends of two MSIs. As an example, consider the rule: *A cross legend must be preceded by a cross or a directional arrow legend.* This results in the following constraint:

$$x_d - [xrl]_c \leq 0.$$

If there is a cross on the MSI directly downstream ($x_d = 1$), this MSI must show either a cross or an arrow to satisfy the constraint. If the downstream MSI shows anything other than a cross, the constraint is always satisfied. Note that $x_c^{rhl}$ is not allowed here, to ensure a proper lead-in of the crosses.

Another example is the rule: *Within a traffic stream at most one speed may be shown.* This rule results in multiple constraints, one for each speed legend ($e$ through $k$).

$$\sum_{\tau \in T} e_\tau - n \cdot ([xrle]_c + x_c^{rhl}) \leq 0$$

$$\vdots$$

$$\sum_{\tau \in T} k_\tau - n \cdot ([xrlk]_c + x_c^{rhl}) \leq 0$$

Above, $T$ stands for all MSIs in the traffic stream of $c$, and $n$ the number of MSIs in $T$. If any MSI in the traffic stream shows a speed, $c$ must show the same speed or directional legend.

Some rules depend on measure options or logical conditions, such as 'the rush-hour lane is closed'. To efficiently represent these rules, indicator constraints are used. Indicator constraints are of the form:

$$y = f \rightarrow \mathbf{a}^\top \cdot \mathbf{p} \leq b.$$

If expression $y$ is equal to $f \in \{0, 1\}$, then the linear constraint must be satisfied, otherwise it can be violated. Note that the linear constraint can also be an equality. For example, the rule *If the rush-hour lane is closed, a cross must be shown* translates into:

$$\texttt{rhl\_closed}_c = 1 \rightarrow x_c + x_c^{rhl} = 1.$$

Note that it is also allowed to show a normal cross ($x_c$) on a rush-hour lane.

The constraints are chosen in such a way that crosses are always allowed, thus the ILP problem is guaranteed to be feasible.

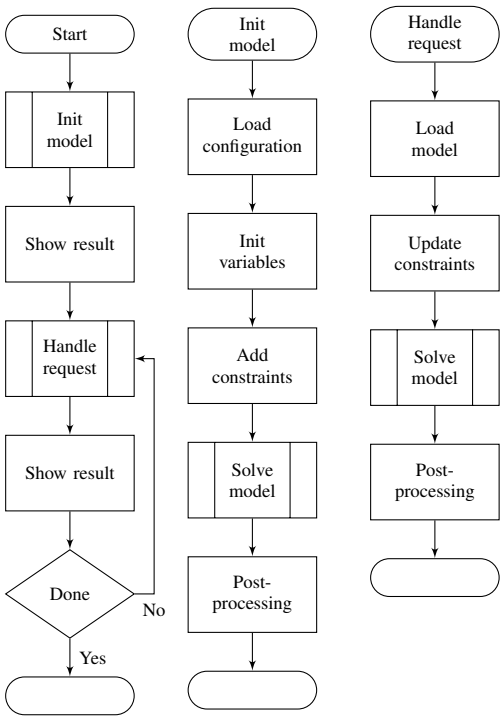


Fig. 3: Schematic structure of the program. Left: Main routine, mid: initialize model, right: handle request.

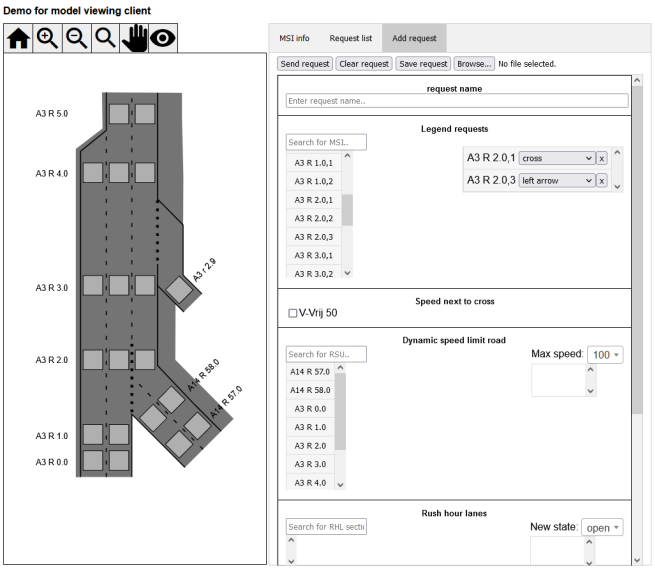


Fig. 4: The user interface for manual validation. The current road is shown on the left, requests can be made on the right.

### *E. Size and structure*

The size of the model obviously depends on the configuration. However, since each constraint is added locally, the complexity of the model remains low. For every MSI, variables and constraints are added based on the topology. 13 variables are added that represent the legends, and up to 21 additional variables that are shared with other MSIs in the same RSU, carriageway, traffic stream or stretch of rush-hour lane. In addition, up to 125 constraints are added for an MSI, if it were to have every relation, which cannot occur in real life. Typically, there are around 20 variables and 25 constraints per MSI.

## IV. Implementation

The program that is used to create models and handle requests is discussed in this section. First, the structure of this program is introduced. Then, the interface (GUI) used for manual validation is given.

### *A. Program structure*

The structure of the program is schematically represented in Fig. 3. The main routine is shown on the left-hand side. First, the ILP model is initialized from the road configuration. Next, the initialized model is shown to the user. Then, the routine enters a loop in which it handles new requests and updates the visualization. This loop is repeated until the routine is stopped.

The initialization subroutine is shown in the middle of Fig. 3. This routine is needed when the configuration changes, which typically happens once every few weeks. It starts with loading the road configuration from a configuration file. Decision variables are created based on the topology and constraints are added for each MSI. Next, the model is solved, this is needed if a rush-hour lane is present, because rush-hour MSIs should show red crosses initially. If no rush-hour lane is present, all MSIs should be blank. Based on the solution, the shown legend is determined for each MSI. In addition, the current model is saved for later use.

Whenever a request is made, the routine as shown on the right-hand side of Fig. 3 is used. First, the current model is loaded. Then, the constraints are updated, depending on the request. If a new measure is requested, a constraint is added for each affected MSI, which prevents the MSI from showing a less restrictive legend than the requested legend, and the constraints for options are updated with the new values, if options change. If an existing measure is removed, the constraints that were added for that measure are removed, and the constraints for options are reset to their original value. Next, the model with the updated constraints and options is solved. Last, post-processing is applied, as was done for the initialization.

### *B. User interface*

To simplify validating the model manually, we have created a GUI. The GUI allows the operator to request measures and see the generated legend pattern, without having to check all the ILP variables manually. A screenshot of the GUI is shown in Fig. 4, on the left-hand side is the current road, on which requests can be made. On the right-hand side is the request menu, where different measure requests can be made; the information of every MSI in the model can be viewed; and the currently active measures can be viewed and removed. When a request is added or removed, it is sent to the main program, which handles all computations. The

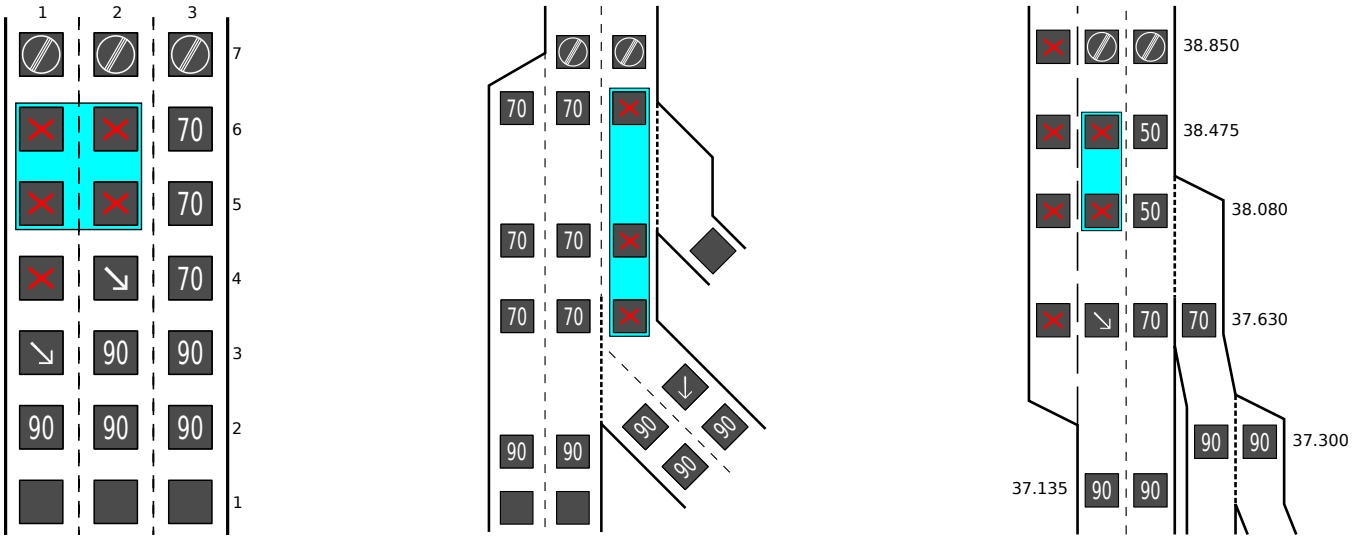


Fig. 5: Three requests and their solutions. The requests are given in the cyan colored area. For the right-hand image, an open lane speed of 50 is requested and the rush-hour lane is closed.

new pattern is determined, a visualization is generated and the image is returned to the GUI, along with the list of active measures.

## V. Experiment

In this section, first some different requests and solutions are given. Next, it is discussed how the model can be validated automatically. Last, some insight is given into how the solutions can be interpreted, if the results are undesired.

### A. Measure requests

Three requests and their solutions are shown in Fig. 5. On the left-hand side, a straight road with no on- or off-ramps is depicted. The model has 21 MSIs, 410 variables and 507 constraints, initialization takes around 0.1 s. The operator requests crosses in the cyan colored area, no options are selected, which results in 4 constraints. The solution as given in the figure is found in less than 0.1 s. As expected, 70 is shown on all rows that contain a cross legend, including the cross generated by the lead-in pattern. Furthermore, in front of the arrow legends are speed 90 legends, and end-of-restrictions legends are shown after the measure.

In the center of Fig. 5 a cross request is shown for more complex situation. The model has 20 MSIs, 434 variables and 480 constraints, initialization takes less than 0.1 s. Note that this model has more variables, due to the complex topology, but fewer constraints due to the smaller number of pairs of neighboring MSIs. The request adds 3 constraints and computations take less than 0.1 s. Note that speeds are led in on secondary roads as well.

On the right-hand side of Fig. 5, a small section of a real highway is given. The road shown here is the right half of the Dutch highway A27, just past Knooppunt Gorinchem, a cloverleaf interchange. The entire interchange and surrounding roads are modeled. The resulting model contains 213 MSIs, spread out over 96 RSUs. It has 4769 variables and 5176 constraints, initialization takes around 1.4 s. Two crosses with an open lane speed of 50 are requested next to the rush-hour lane at the kilometer marks given in the figure. This results in 2 new constraints for the crosses and 2 adapted constraints for the open lane speeds. The solution is found in 0.5 s. Note that speed 70 is shown on the MSIs at 37.630 km, since the open lane speed does not apply to rush-hour lane crosses.

### B. Automatic validation

The model presented here was validated by having operators from Rijkswaterstaat request measures in the GUI and verify the correctness of the solution. Besides requesting measures and validating the result in the GUI, the model can also be used for automatic validation. This is done by writing scenarios, i.e., a sequence of request additions and removals, with the expected final result. Then, for each scenario, the model is initialized, the requests are executed in order, and the result is compared to the expected result. This allows the operator to validate many situations, without having to enter everything manually, as is currently done.

### C. Interpreting measures

Some insight in the solution can be obtained by looking at the values of constraints. If the left-hand and the right-hand sides of a constraint are equal, it is active. Active constraints indicate which business rules are relevant. For example, looking at MSI 1 on row 4 on the right-hand side of Fig. 5, the constraint $x_d - [xrl]_c \leq 0$, that was mentioned above is active, since both $x_d = 1$ and $x_c = 1$. Furthermore, $r_c + [xrl]_r \leq 1$ is active, denoting an arrow to the right may not point to another arrow or a cross, and $l_c = 0$ is active, denoting the arrow may not point off the side of the road. These rules caused a cross to be shown, since other legends are not allowed.

If a certain legend pattern is considered wrong, the rules can be analyzed by adding constraints that force the desired legend variables to equal 1. If this problem is feasible, the desired solution is more restrictive than the 'wrong' solution, which suggests a rule is missing. Otherwise, an irreducible inconsistent system can be generated, which is the smallest set of constraints that is still infeasible. This set contains the rules that (together) cause the wrong legend pattern. Note that both these methods require a good understanding of ILP and the business rules.

## VI. Discussion

By using a model-based approach, several benefits are achieved, which are discussed here.

First of all, a model-based approach enables simulation. Therefore, when the configuration changes, e.g., by roadworks, the new configuration can be simulated, to find any situations that cause undesired legend patterns. This way, errors are detected earlier in the design process, reducing the project delay and cost incurred, compared to finding the mistake once the system is realized.

In addition, if legislation causes the business rules to change, the implications of this change are quickly visualized in the model. With automatic validation, the situations affected by these rule changes can be identified and verified.

When a certain request gives an undesired legend pattern, the underlying rules that caused this can be evaluated. However, specific knowledge about the model is required for this. As such, it is expected that this method cannot be used in day-to-day practice by operators. If undesired situations are recorded, they can be analyzed by an expert afterwards, to improve the system.

Besides possible improvements to the current system, the model-based approach does not require physical MSIs to be present. This allows the model to be used on roads without RSUs, which currently comprise about two-thirds of the Dutch highway network. The model output can be sent directly to service providers, who can use the information for in-car signaling.

## VII. Conclusion

In this paper, model-based design is applied to a real-life traffic management system. The system is modeled using ILP, where business rules are translated into constraints and the goal is to minimize the impact on traffic. An implementation with a GUI is created to validate the model. The implementation determines solutions at a competitive speed, compared to the current system. When the rules or configuration are changed, the new situation can be validated much faster than in the current system.

Rijkswaterstaat is considering to use the methods presented here for managing the Dutch highways in the future. First, the model will be extended to form a 'digital twin' of the current MTM system. This will allow operators to verify the functional behavior and gain confidence in model-based methods. Next, the model will be applied to roads that are not covered by MTM, to improve the safety of unsignaled roads and make the road user familiar with in-car signaling. Last, when existing roads are going to be renovated years from now, the current physical MTM system can gradually be replaced by a model-based in-car system.


## Acknowledgements

The authors thank Marco Schreuder and Han Vogel from Rijkswaterstaat, and Pascal Etman from Department of Mechanical Engineering, Eindhoven University of Technology for their support and suggestions.


## References


[1] A. M. De Souza, C. A. R. L. Brennand, R. S. Yokoyama, E. A. Donato, E. R. M. Madeira, and L. A. Villas, "Traffic management systems: A classification, review, challenges, and future perspectives," *International Journal of Distributed Sensor Networks*, vol. 13, apr 2017.

[2] B. Khondaker and L. Kattan, "Variable speed limit: A microscopic analysis in a connected vehicle environment," *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 146–159, 2015.

[3] N. Nezamuddin, N. Jiang, T. Zhang, S. T. Waller, and D. Sun, "Traffic operations and safety benefits of active traffic strategies on TxDOT freeways," tech. rep., Center for Transportation Research UT Austin, october 2011.

[4] F. Soriguera, I. Martínez, M. Sala, and M. Menéndez, "Effects of low speed limits on freeway traffic flow," *Transportation Research Part C: Emerging Technologies*, vol. 77, pp. 257–274, 2017.

[5] A. Kotsialos, M. Papageorgiou, and F. Middelham, "Optimal coordinated ramp metering with advanced motorway optimal control," *Transportation Research Record*, vol. 1748, no. 1, pp. 55–65, 2001.

[6] F. Middelham, "State of practice in dynamic traffic management in the Netherlands," *IFAC Proceedings Volumes*, vol. 36, no. 14, pp. 293–298, 2003.

[7] N. Dutta, R. A. Boateng, and M. D. Fontaine, "Safety and operational effects of the interstate 66 active traffic management system," *Journal of Transportation Engineering, Part A: Systems*, vol. 145, no. 3, p. 04018089, 2019.

[8] I. Lynce and J. Ouaknine, "Sudoku as a sat problem.," in *AI&M*, 2006.

[9] J. J. Verbakel, M. E. W. Vos de Wael, J. M. van de Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "Supervisory control of roadside units," in *16th Workshop on Discrete Event Systems (WODES)*, pp. 79–86, IFAC, 2022.

[10] J. J. Verbakel, M. E. W. Vos de Wael, J. M. van de Mortel-Fronczak, W. J. Fokkink, and J. E. Rooda, "A configurator for supervisory controllers of roadside systems," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 784–791, IEEE, 2021.

[11] J. D. C. Little, "The synchronization of traffic signals by mixed-integer linear programming," *Operations Research*, vol. 14, no. 4, pp. 568–594, 1966.

[12] J. Rios and J. Lohn, "A comparison of optimization approaches for nationwide traffic flow management," in *AIAA Guidance, Navigation, and Control Conference*, p. 6010, 2009.

[13] H. Ghaffarian, M. Fathy, and M. Soryani, "Vehicular ad hoc networks enabled traffic controller for removing traffic lights in isolated intersections based on integer linear programming," *IET Intelligent Transport Systems*, vol. 6, no. 2, pp. 115–123, 2012.

[14] M. Hajiahmadi, B. De Schutter, and H. Hellendoorn, "Control of traffic networks using the link transmission model and mixed integer linear programming," in *Proceedings of the 1st European Symposium on Quantitative Methods in Transportation Systems*, 2012.

[15] B. Kersbergen, T. van den Boom, and B. De Schutter, "Distributed model predictive control for railway traffic management," *Transportation Research Part C: Emerging Technologies*, vol. 68, pp. 462–489, 2016.

[16] M. A. Schreuder, *TOP 4.6.1 - FEP Functional Overview*, oct 2015.

[17] J. van Meurs, "Calculating legend patterns using integer linear programming," Master's thesis, Eindhoven University of Technology, 2022.