9th International Conference on Prestigious Applications of Intelligent Systems – PAIS@ECAI2020
Santiago de Compostela, Spain

ASP-based Signal Plan Adjustments for Traffic Flow Optimization

Thomas Eiter[1] and **Andreas Falkner**[2] and **Patrik Schneider**[1,2] and **Peter Schüller**[1]

Abstract. Worldwide, many cities spend considerable effort to reduce traffic and specifically to avoid traffic congestions. Adaptive traffic control systems serve this purpose by dynamically adjusting traffic signals for optimizing the traffic flow on intersections. Systems such as SCOOT are based on an "intelligent" combination of different traffic optimization strategies. However, they miss the possibility (i) to add and change on-demand rules to implement new optimization strategies, and (ii) to simulate the outcome of new strategies on-the-fly which is similar to the capabilities of microscopic traffic simulation tools such as SUMO. In order to overcome the above limitations, we present a novel approach for calculating signal phase plans (SPPs) used for optimizations in traffic control systems. Our approach is based on Answer Set Programming (ASP) and combines ASP encodings of an abstract mesoscopic flow model and a strategy for generating possible SPPs. Experimental results shows that traffic simulation can be well approximated and that the generated SPPs improve the traffic flow effectively.

1 Introduction

Many large cities worldwide spend considerable effort to reduce the overall traffic, for instance, by limiting parking space or introducing congestion pricing. Despite these efforts, traffic congestions are still a large problem with an estimated cost of $87 billion for the U.S. economy in 2018 as estimated by the World Economic Forum [9]. Urban Traffic Management (UTM) solutions are one of the technology-based strategies to reduce traffic congestions. One cornerstone of UTM solutions are (near) real-time, adaptive traffic control systems such as SCOOT [5], which are used to dynamically control traffic signals to optimize the traffic flow on intersections. However, the most successful systems such as SCOOT were developed in the 1980s and are based on an "intelligent" combination of different traffic optimization strategies, but miss the possibility (a) to add and change on-demand rules to implement new optimization strategies, and (b) to replicate microscopic traffic simulation tools such as SUMO [17], which allow to simulate the outcome of new strategies on-the-fly.

Our main challenge is to support reasoning on the level of microscopic traffic simulations, in order to predict the behaviour of traffic flows for extracting quality criteria, and to find better or even optimal signal plans with respect to these quality criteria. As the solution space is huge, the goal is to find suitable optimization strategies which, ideally, can also be applied online.

Standard traffic simulations tools such as SUMO allow for microscopic simulation runs, but they do not provide any native support for reasoning over simulation models. Moreover, running millions of simulations (even with automated support) is not a basis for feasible solutions, if new signal plans have to be calculated online to react on changing traffic patterns. A common way to overcome this are macroscopic traffic flow models, in which one abstracts over the individuals and uses aggregated values such as the number of vehicles on a traffic segment. However, the inherent loss of information since the road network and signal plans are not fully represented, makes such models imprecise approximations of reality, whose quality may be insufficient for targeted applications such as adaptive green wave coordination.

In this paper, we consider a middle-ground and develop a mesoscopic flow model which abstracts not only individual but also time and the road network to reduce complexity. On the other hand, it keeps useful details of the network structure to achieve a good quality of the abstraction such that the results approximate microscopic flow models. Briefly summarized, our main contributions are as follows:

- We present a new *mesoscopic* model for traffic flow, which features (a) nodes of different types (source, link, intersection, and sink nodes), (b) distribution from incoming to outgoing edges, via individual rates, and (c) denial constraints on (pairs of) edges regarding possible status (in particular, signal phases) to ensure safety (Section 3).
- We develop an encoding of the flow model in Answer Set Programming (ASP), which allows us to generate the abstract runs of concrete traffic flows in the answer sets of the ASP encoding; in this way, it is possible to reason at a high level about traffic flows (Section 4).
- We address the problem of signal phase plan (SPP) configuration, for which we propose different optimizations strategies. We describe an ASP encoding of one such strategy, which takes advantage of the modeling power of ASP (Section 4).
- Experimental results show that the approach is promising, as initial SPPs could be significantly improved, and in most cases even optimal SPPs could be found (Section 5).

Our results are encouraging and show the potential of an ASP based approach for traffic flow studies, which allows for flexible modeling and problem solving.

2 Preliminaries

2.1 Answer Set Programming

For problem solving and optimization, we adopt the declarative Answer Set Programming (ASP) paradigm [7, 6, 24]. ASP is based on non-monotonic logic programs, whose core rules are of the form

$$a \leftarrow b_1, \ldots, b_n, not\ c_1, \ldots, not\ c_m$$

where a and all b_i, c_j are atoms in a first-order predicate language and "not" is negation-as-failure (NAF); a may be missing, which then yields a *constraint*. Intuitively, a is inferred if all b_i and no c_j are

[1] Vienna University of Technology, Vienna, Austria
[2] Siemens AG Österreich, Vienna, Austria

inferred. The *stable models* (aka *answer sets*) of a program are special Herbrand models that satisfy a fixpoint condition [12].

In ASP, problems are encoded to logic programs P such that the answer sets I of P represent solutions; using a "guess and check" (aka generate-and-test) methodology. A program P consists of [8]:

1. non-deterministic rules with unstratified negation (or disjunction in the heads) to generate candidate solutions;
2. constraints that check whether a candidate solution is suitable; and
3. auxiliary rules used in the first and second part.

Optimal solutions may be determined using special directives, e.g., $\#maximize\ V$ selects the solution where V has the largest value [11]. Further notions and examples will be given in Section 4.3.

2.2 Traffic Flow Models

Roughly, *road traffic* can be described using the concept of *traffic flow*, which is measured by predefined variables such as speed and density of vehicles on a predefined location over a predefined time span [13]. Traffic flow itself is often described by the qualitative states of *free flow*, *traffic jam*, and a state of *synchronized flow* [14]. Predefined variables can be used to describe local (resp., global) aspects, which we call *local* (resp., *global*) *variables*. Local variables are measured on specific road segments and time intervals of length s_d or time points t and may include (see [13] for more variables)

- *Traffic flow count*: $q_S = \frac{n}{s_d}$, where n is the number of vehicles that pass a cross-section S during the (span of) a time interval s_d;
- *Average speed*: $s_X(t) = o$, where o is the average speed of all vehicles on a segment X at a specific time point t.

Global variables describe the whole network and are extracted over a longer time span and serve as a base for overall performance measurement; e.g., average speed of all vehicles for a simulation run.

A more powerful approach to gain insight into traffic control and optimization are traffic flow models. A common classification is based on the dimension of representation (vehicle or flow based) and the granularity of behaviour rules (microscopic vs. macroscopic) [13].

Macroscopic Flow Models. A basic macroscopic model for traffic dynamics uses queuing theory, where the number n of vehicles in a queue is observed [13]. The latter results as the difference between the inflow to a bottleneck node (q), e.g., an intersection, and its capacity, which determines the outflow of the queue (denoted as C). The time-dependent flow model can be written as: $dn = q(t)dt - C(t)dt$, where dn is the change in vehicle number and dt a time span over t.

Microscopic Flow Models / Traffic Simulations. A microscopic model is based on the properties and behaviours of individual vehicles, which may emerge from an internal agent model that reacts on external stimuli such as other vehicles. A central agent model is the car-following model, which describes the interactions with preceding vehicles, where either a general car-following, safety-distance, or psycho-physical car-following model can be applied [25].

Microscopic traffic simulations are one of the standard tools for UTM practitioners to conduct offline studies and experiments in order to validate novel traffic control techniques. Several tools supporting microscopic traffic simulations are available, with SUMO [17], MITSIM [4], VISSIM (http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/), and AIMSUN (https://www.aimsun.com/) among the most prominent. Our work is based on SUMO, as it is available under an open source licence and customizable using PYTHON scripts.

Mesoscopic Flow Models. Mesoscopic flow models were developed to fill the gap between macroscopic and microscopic models, where different levels of aggregations are introduced, but also the behaviour of individual vehicles or traffic control systems are represented. Headway distribution, cluster, and gas-kinetic models are applied to represent the mesoscopic flow [15].

3 Symbolic Mesoscopic Flow Model

Our model covers the elements of a microscopic and a macroscopic flow model such as (1) the static traffic network and intersection topologies, (2) dynamic states such as vehicle counts and signal phase states, as well as (3) the transformation in the dynamic states. It is accordingly composed of a *static component* defining the road network and local topologies, and a *dynamic component* defining traffic flow, which is based on (a) a time model using a timeline, data items of aggregations, and streams, (b) traffic flow generation for different node types, to capture the number of vehicles present at different time points; and (c) signal phase plans, to capture the generation of signal phases at different time points.

Example 1 Figure 1a shows a road network in SUMO with two intersections that connect three roads (one horizontal and two vertical) with two incoming and two outgoing lanes for each road. Figure 1b is the extracted mesoscopic flow model of Figure 1a including nodes for intersections, links, sources, and sinks.

3.1 Static Component

The static traffic model describes the structure of the road network including the roads, lanes, intersections, intersection topologies, and traffic control installations such as traffic lights (TLs). Traffic control installations have one or more TL signal groups attached and are managed by a TL controller, which assigns *red* (for stop) and *green* (for go) states to signal phases. We assume for our setting that we have a global TL controller that manages all intersections. The signal phases for each signal group are encoded in a "default" signal plan, where the green/red split of a full phase length is defined. Usually, each pair of adjacent incoming lanes has one signal group assigned.

Let N be a road network of several intersections. Its static component is represented by a directed, labeled graph

$$G_N = (V, E, l_V, l_E, R_V, R_C), \quad \text{where}$$

- the nodes V represent real entities (e.g., intersections) or virtual (e.g., boundary to external environment) "crossing points". Every node $v \in V$ has a fixed type $tp \in \{src, snk, ins, lnk\}$ assigned, denoted as $v : tp$ where src means source node, snk sink node, ins intersection node, and lnk link node.
- the edges E represent entire lanes or segments of lanes, and define the (possible) traffic flow in the network by connecting the nodes.
- l_V and l_E are a node and edge labelings and provide contextual information. The function $l_V : V \to \mathbb{N}^+$ assigns the *bottleneck capacity* to each node, thus defining the highest possible throughput of vehicles at a single time step, while the function $l_E : E \to \mathbb{N}^+$ assigns the *maximum load capacity* to each edge that is the maximum capacity of cars that can simultaneously be located on an edge.
- relation $R_V \subseteq V \times E \times \mathbb{R}^+$ captures the traffic flow distribution:
  $R_V = \{(v, e, w) \mid v \in V, v : ins,\ e = (v, v') \in E, 0 < w \leq 1\}$.
  If w is not defined, there is no traffic outflow to e, if $w = 1$ all traffic flows to e. For example, intersection v_1 with an incoming lane $e_1 = (s_1, v_1)$ and two outgoing lanes $e_2 = (v_1, s_2)$ and $e_3 = (v_1, s_3)$, then an even split of traffic flow is represented as $R_V = \{(v_1, e_2, 0.5), (v_1, e_3, 0.5)\}$.

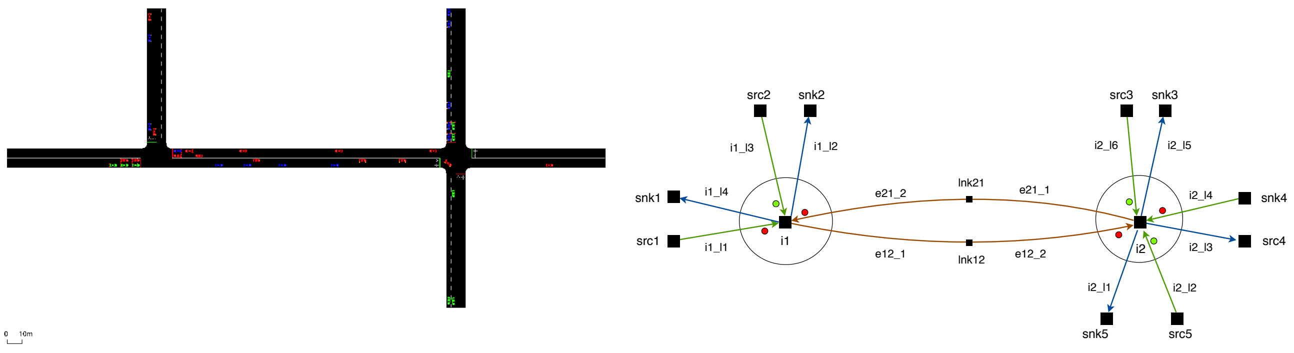

Figure 1: (a) Rendering of two intersections in SUMO and (b) corresponding mesoscopic flow model

- the relation $R_C \subseteq E \times E$ captures implicit denial constraints on an intersection for safety, to ensure that orthogonal crossing lanes are not in a green phase at the same time point.

Segmentation. The static component is extracted from a given traffic simulation model (such as one generated by SUMO) by segmentation. A simple segmentation algorithm has the following steps: (1) uniformly segmentate the road network, i.e., divide whole lanes into individual segments of similar (ideally same) length; (2) reduce it to equi-distant edges and nodes representing G_N; (3) generate source/sink nodes for the boundary of the network. Step (2) reduces the complexity of the simulation model as follows:

1. reduce each intersection to a single node,
2. generate the relation R_V from the intersection topology, using (offline) traffic distribution statistics to estimate the weights, and
3. merge adjacent incoming (resp. outgoing) lanes to a single incoming (resp. outgoing) edge.

The lane splitting in Step 1) allows us to assign (a) the same capacity and (b) the current traffic flow count for a uniform time interval s_d to each edge. The uniform segmentation is central to our approach, as it makes the propagation of traffic through the network in the dynamic component much simpler. Segmentation is a preprocessing step, and creates a graph as depicted in Figure 1b.

3.2 Dynamic Component

The intuition of the dynamic component of our model is illustrated in Figure 2. It extends the static component with temporal information related to single *simulation steps* also called *time points*. The temporal information consists of two kinds of data items which might change in each simulation step: (a) the *number of vehicles* (NrV), which is the count of vehicles on a specific edge, and (b) the *signal phase states* (SPS), which is the TL status of specific incoming lanes on (intersection) nodes at a specific time point. Note that the data includes "real" observed data extracted from the simulation model, but also values derived from the extracted data.

Time Model. The underlying time model is *interval-based* and captures the *interval length* s_d (duration) of a single simulation step delimited by the *finishing time* t_i of the simulation step. However, the data model used is *point-based* and captures only t_i, thus expressing that a *data item* is valid between $t_{(i-1)}$ and t_i, where the difference $s_d = t_i - t_{i-1}$ is the duration. Using a point-based model does not lead to a loss of information, given that we have a uniform segmentation of the timeline that can be captured by time points.

For capturing a full traffic simulation run, we introduce the *timeline* $\mathbb{T} = [0, l] \subset \mathbb{N}$, which is a *closed* interval of length l (also referred to as *simulation length* t_l). A simulation *stream* is a pair $D = (\mathbb{T}, f)$ of a timeline $\mathbb{T}$ and a function $f : \mathbb{T} \rightarrow 2^O$ that assigns to each element (*time point*) $t \in \mathbb{T}$ a set $f(t) \subseteq O$ of *data items*; O is the set of tuples of form $\langle o, x \rangle$, where $o \in \mathbb{R}$ is a real number and $x \in E$.

Data items are one form of abstraction in our model, where NrV or signal phases (represented by numbers) are aggregated over the duration s_d. As said, we have two streams:
(1) the stream $D_{NrV} = (\mathbb{T}, f_1)$ of vehicles counts, where each $f_1(i)$ consists of edge data items. This way, we link NrV data (generated only by nodes) to edges.
(2) the stream D_{SPS}=$(\mathbb{T}, f_2)$ of signal phases, where f_2 is defined accordingly with the difference that the observed values o are binary encodings of the signal states: *red* is 0, *green* is 1; f_2 consists of data items $\langle o, e \rangle$, where $e = (v', v)$ is an incoming edge to an intersection node v. Thus, SPS data items are only assigned to incoming lanes of an intersection, which captures the intuition of a signal phase plan.

For convenience, we use for f_i the notation $x@t$ if $x \in f_i(t)$, e.g. $\langle o_1, e_1 \rangle @t_1$, and $\langle o_2, e_2 \rangle @t_2$.

Traffic Flow Calculation. The traffic flow (TF) calculation brings dynamicity into our model, by capturing the vehicle flow. It generates NrV data items based on the mentioned nodes, and assigns them at the next time point to edges in the network. Sink nodes play a special role and accumulate the incoming data items, but do not generate data items themselves.

A TF calculation is performed at each time point t_i and adds for each edge $e_j \in E$ in our network the new data items for stream D_{NrV}. It is based either on the incoming edges for intersection and link nodes, or on a predefined generation function for source nodes. The result of each node's calculation are new data items that are propagated to the outgoing edges, updating their state. The state of an edge is captured by the set of data items attached to it on a specific timepoint t_i. Figure 2 illustrates three calculation steps for four time points represented by the dotted arrows. Simplest is the calculation of link nodes, as one incoming NrV data item generates a single outgoing data item. Intersection nodes are more difficult to handle, as incoming data items must be split into more outgoing data items according to the TF distribution to the outgoing edges.

Source nodes. A source node $v : src$ generates for each time point NrV data items according to a given function as follows, where $e = (v, v')$ is an updated outgoing edge:

$$f_{src}(v, e, t_j) = \langle min(l_E(e), f_D(t_{j+1})), e \rangle @t_{j+1},$$

where the value generation function $f_D(t_{j+1})$ is a stochastic function that creates a new value for time point t_{j+1} based on a predefined distribution model. Its purpose is the replication of demand modelling functionality of traffic simulations, with for instance a constant

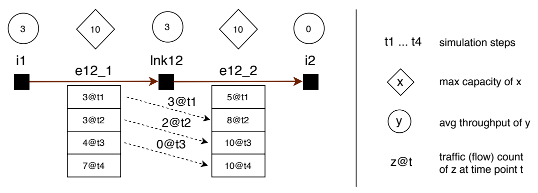

Figure 2: Example of three calculation steps for time points $t1$ to $t4$; data items with $o = 3$ are generated in $i1$ and propagated via $lnk12$ to $i2$, where they are collected up to the load capacity limit of 10 due to bottleneck capacity of 0.

generation of the same NrV data item for each t_i.

Link nodes. A link node $v : lnk$ propagates for each simulation step a NrV data item from a source edge $e_1 = (v_1, v)$ to a target edge $e_2 = (v, v_2)$ as follows:

$$f_{lnk}(v, e_1, t_j) = \langle min(l_V(v), l_E(e_2), f_{lnk'}(e_1, t_j)), e_2\rangle \,@t_{j+1},$$

where function $f_{lnk'}(e, t)$ returns the value o in $\langle o, e\rangle@t$ taken from stream D_{NrV}. Note that for link nodes, only one incoming and one outgoing edge is allowed (otherwise segmentation creates an intersection node).

Intersection nodes. An intersection node $v : ins$ aggregates the different NrV data items from incoming edges, and distributes them to the outgoing edges according to the predefined ratios of relation R_V. For each outgoing edge $e = (v, v')$ of intersection v, we have

$$f_{ins}(v, e, t_j) = \langle min(l_V(v), l_E(e), f_{ins'}(v, e, t_j)), e\rangle \,@t_{j+1},$$

where $w_{v,e}$ is given by $R_V(v, e, w_{v,e})$ and $f_{ins'}$ is as follows:

$$f_{ins'}(v, e, t_j) = \sum_{e_i=(v_i,v)\in E} \langle o_i, e_i\rangle \,@t_j \cdot w_{v,e} \cdot f_{sps}(v, e_i, S_v, t_j)\ .$$

In $f_{ins'}$, we slightly abuse the notation as we do not sum up and multiply the tuples $\langle o_i, e_i\rangle$ but only their values o_i. The function f_{sps} returns the signal state tuple (with $o'_i = 0$ or $o'_i = 1$) for e_i at t_j, thus ignoring incoming lanes which are on state 0; details of f_{sps} will be given in the next section. The relation R_V is important, as it defines the "weights" of the outgoing values that are summed up; it can also be used to close lanes by setting the weights to 0.

Sink nodes. A sink node $v : snk$ does not generate values on its own, but collects all NrV data items of incoming edges at each time point. The aim is to record the number of vehicles that have left the network. The sink generates result tuples

$$f_{snk}(v, t_j) = \langle f_{snk'}(v, t_j), v\rangle \,@t_{j+1}, \quad \text{where}$$
$$f_{snk'}(v, t_j) = \sum_{e_i=(v_i,v)\in E} \langle o_i, e_i\rangle \,@t_j.$$

Sink nodes will play a central role later, as the impact of an improved signal plan can be determined using the accumulated sink nodes.

Signal Phase Generation. Streams of signal phase states (SPS) are similar to TF streams, but only consider SPS data items, i.e., from $O_{SPS} = \{\langle o, e\rangle \mid o \in \{0, 1\}, e = (v', v) \in E \text{ and } (v : ins) \in V\}$.

Signal phase states, shortly signal states, for traffic lights are defined in signal phase plans, where we assume that each intersection has one plan assigned that covers all the signal groups and lanes of the intersection. In Figure 3 we give an example of a signal plan as used in SUMO. The length of a full cycle is the sum of `durations` (60 in our example). Each signal group state is shown in `state`, where "`r`", resp., "`G`", represents red, resp., green phases for the duration. Note that in this encoding, each position is an index and assigns the state to a signal group, i.e., the first position defines the state of the signal group for first lane. The plan also shows *implicit* constraints, e.g., that

```
<tlLogic id="I1" type="static" programID="0" offset="0">
  <phase duration="30" state="rrrrGGGGGGGG"/>
  <phase duration="30" state="GGGGGrrrrrrr"/>
</tlLogic>
<tlLogic id="I2" type="static" programID="0" offset="0">
  <phase duration="30" state="GGGGGGrrrrrrGGGGGGrrrrrr"/>
  <phase duration="30" state="rrrrrrGGGGGGrrrrrrGGGGGG"/>
</tlLogic>
```

Figure 3: Example of a signal plan as used in SUMO

the groups at position 1-4 and 5-12 should not be green at the same time. From the SUMO encoding shown in Figure 3, we can conclude that a formal signal plan has to include the following components: a fixed cycle length, a timeline covering the phase time for each signal group, where red/green states have alternating time slots, and a list with pairs of denial (lane crossing) constraints.

More formally, a signal plan s_v can be represented by an $n \times m$ matrix S, where the columns n represent time points and the rows m are lanes. As defined above, a state $s_{i,j}$ can be either 0 (red phase) or 1 (green phase). The matrix S_v for m incoming lanes of an intersection v over the simulation length n is thus:

$$\begin{array}{c|cccc}
 & t_1 & t_2 & \dots & t_n \\
\hline
l_1 & s_{1,1} & s_{2,1} & \dots & s_{n,1} \\
\dots & & & & \\
l_m & s_{1,m} & s_{2,m} & \dots & s_{n,m}
\end{array}$$

The signal plan s_v associated with an intersection node v can also be represented by a function that associates with v and any incoming edge $e = (v', v)$ a signal state at a specific time point:

$$f_{sps}(v, S_v, e, t_j) = \langle pos(S_v, t_{j+1}, e), v, e\rangle \,@t_{j+1},$$

where the function $pos(S_v, t, e))$ extracts from S_v the state (0 or 1), depending on the position of time point t and edge e in S_v.

3.3 Frame Axiom and Evaluation Order

The above definitions cover the case when we have a "free" movement of vehicles. They do not cover the case when we have no or fewer movements (in NrV) due to a red signal phase or low bottleneck capacity defined in l_V. Ignoring these, we face a loss of a NrV in the next time step; hence we have to generate also NrV data items for non-movements. For this purpose, we introduce two functions f_{fr1} and f_{fr2}, which preserve NrV data items that are not moved. The function f_{fr1} is defined for an outgoing node $e(v', v)$ of an intersection v and NrV data item $\langle o_N, e\rangle \,@t_k$ as follows:

$$f_{fr1}(e, v, o_N, t_k) = \begin{cases} \langle min(0, o_N - l_V(v)), e\rangle \,@t_{k+1}, \\ \qquad\qquad \text{for } o_S = 1 \text{ of } \langle o_S, v\rangle \,@t_{k+1}; \\ \langle o_N, e\rangle \,@t_{k+1}, \\ \qquad\qquad \text{for } o_S = 0 \text{ of } \langle o_S, v\rangle \,@t_{k+1}. \end{cases}$$

In the first case, the signal phase is on green, so we subtract the maximal NrV for movements, i.e., the bottleneck capacity, from the NrV at t_k to calculate the remaining NrV at t_{k+1}. In the second case the signal phase is on red, hence we have no movement and the existing NrV on edge e at t_k is still on edge e at t_k. The function f_{fr2} is defined along the same line, for an outgoing node $e(v', v)$ with an intersection node v and an NrV data item $\langle o_N, e\rangle \,@t_k$:

$$f_{fr2}(e, v, o_N, t_k) = \langle min(0, o_N - l_V(v)), e\rangle \,@t_{k+1}.$$

The above functions are akin to frame axioms in logic-based AI planning [18]: they cover the cases where the bottleneck capacity of l_v bounds the maximum flow. If the actual flow is smaller as up-flow edges at the next time point are already saturated, NrV data items get lost. This loss is an effect of the abstraction in our mesoscopic model, where a saturation does not spill back to down-flow edges. Adding the latter, which is observable in microscopic models, is challenging, in particular in case of cyclic traffic flows.

Evaluating the functions f_{src}, f_{lnk}, f_{ins}, f_{fr1} and f_{fr2} over all nodes in V and and time points in $\mathbb{T}$ limited by the simulation length t_l, naturally induces a stream $D_{NrV} = \{\langle o_1, e_1\rangle \,@t_1, ..., \langle o_j, e_k\rangle \,@t_l\}$. Similarly, evaluating f_{sps} over all intersections in V and time points up to t_l, induces a stream $D_{SPS} = \{\langle o_1, e_1\rangle \,@t_1, \ldots, \langle o_i, e_k\rangle \,@t_l\}$.

The only requirement in the evaluation order of the above functions is the temporal order, where all functions of time point t_i are evaluated, before the functions of t_{i+1} are started.

4 ASP-based Problem Solving and Optimization

The mesoscopic flow model from above describes the quality of a network with a fixed signal phase plan (SPP), where the quality is calculated over a pre-defined number of time steps through summarizing the sink nodes at the last time point. This abstract flow model however, does not include any strategy to improve such plans, which should lead to our main goal of optimizing the traffic flow by adapting SPPs. As the next step, we will provide the encoding for optimizing traffic flows based on different configurations of SPPs.

4.1 Generic Problem Solving Strategies

As the aim of this work is the (re-)configuration of SPPs, we introduce three candidate plan generation strategy that can be applied to (a) generate entirely new or (b) improve existing SPPs.

We call a full set of assigned states in matrix S_v for v an *SPP configuration*, where all states are either 0 or 1. The different solving strategies aim at finding "good" candidates of SPP configurations based on S_v under the constraints R_C from the static component. We discuss here some strategies for generating SPP configurations.

***S1*: Random generation**. The first strategy *S1a* is the simplest way to generate SPP configurations. A single *random* assignment for an intersection v is generated by the following steps:

1. start with an empty matrix S_v, and fill it randomly with 0 and 1;
2. repair two states that violate the (crossing lanes) constraints of R_C by setting one of them to 0.

Step 2) is needed to rule out "unsuitable" configurations, where vehicles move simultaneously on crossing lanes. Moreover, *S1a* generates fragmented configurations, which are not usable in practice. An improvement of *S1a* is strategy *S1b*, which ensures that larger state blocks of length l_n are generated randomly; this leads to fewer and longer signal phases. Like *S1a*, it needs the repair step which could reintroduce more fragmentation.

***S2*: SPP Shortening/lengthening**. In strategy *S2*, we assume that existing SPPs are already in place, but temporal adjustments are needed due to changes in traffic patterns. The most generic SPP used in *S2* is an uniform distribution of signal phases over the incoming lanes. We start with a filled matrix S_v called *base SPP* that captures the current plan for intersection v and proceed as follows, where $c = [c_n, c_p]$ is a given adjustment parameter:

1. extract from S_v for each lane l_j in v the segmentation (by red phases) of green signal phases, denoted as $GP_{l_j} = \{gp_1, \ldots, gp_k\}$;
2. guess for each gp_i two values c_i^b and c_i^e from c, to change gp_i at its beginning by c_i^b and at its end by c_i^e, where a negative value means shortening and a positive value enlarging;
3. check if the guess satisfies R_C; if so, apply c_i^b and c_i^e on gp_i.

Note that this strategy needs a reasonable range for interval c, as it affects the number of configurations twice (at gp_i's begin and end).

***S3*: Templates with SPP Shortening/lengthening**. Strategy *S2* does not perform well, if the underlying SPP is inefficient. A way to overcome this is to extend *S2* with SPP templates of varying structure. Given a set $ST = \{st_1, ..., st_k\}$ of templates, we proceed as follows: choose a template st_i from ST, and adjust the length of st_i according to the simulation length l, and then proceed as described in *S2*.

4.2 Problem Size and Optimization

The problem size $size(S)$ of a strategy S is the count of all possible configuration candidates generated by S. The parameters are the SPP matrix $M = n \times m$, the state space of $|\{0, 1\}| = 2$, and further ones depending on the strategy. For strategies *S1-S3*, we obtain:

- $size(S1a) = 2^{(n \cdot m)}$, e.g., for two lanes over 10 time steps there are $2^{(10 \cdot 2)} = 1,048,576$ candidates;
- $size(S1b) = 2^{((n/l_n) \cdot m)}$, where l_n is the phase length. E.g., the above example with $l_n = 2$ has $2^{(5 \cdot 2)} = 1,024$ candidates;
- $size(S2) = 2^{(2 \cdot |c| \cdot m)}$, where $c = [c_n, c_p]$ is the adjustment parameter for signal phases. E.g., $c = [-1, 1]$ with $|c| = 3$ and two lanes yields $2^{(2 \cdot 3 \cdot 2)} = 4,096$ candidates;
- $size(S3) = l_{st} \cdot 2^{(2 \cdot |c| \cdot m)}$, where $|c|$ is as above and $l_{st} = |ST|$ is the number of templates. In our example, we have $8,192$ candidates.

As easily seen, strategy $S1a$ is infeasible for real-world networks, due to a rapid growth of candidates that is exponential in n and m.

Optimizations. Even with the most economic strategy *S2*, enumerating all configuration candidates to find the optimal solution is infeasible. Quality indicators can thus be used to (a) guide the search to find solutions faster using soft constraints, and (b) restrict the search space using hard constraints. As remarked in Section 3, the sinks in the mesoscopic flow model keep the NrV data items for all time points; the *quality indicator* Q_{NrV} for traffic flow is thus defined as:

$$Q_{NrV} = \textstyle\sum_{i=0, v_i\,:\,snk}^{l} \langle o_i, v_i\rangle \,@t_i.$$

Another intuitive indicator, obtained from the SPP matrix $M = n \times m$, is the number of time points with SP status 1 (i.e., green); it measures the length of all green SPs (whose maximization may increase the traffic flow). The *quality indicator* Q_{SPS} for a matrix S is defined as:

$$Q_{SPS} = \textstyle\sum_{i=0}^{n} \sum_{j=0}^{m} s_{i,j}.$$

The two indicators can be combined using optimization statements and strong constraints (for cut-off) as follows: (a) maximize Q_{NrV}, then maximize Q_{SPS}, (b) maximize Q_{SPS}, then maximize Q_{NrV}, (c) maximize Q_{NrV} and cut-off Q_{SPS}, and (d) maximize Q_{SPS} and cut-off Q_{NrV}. In cases (a) and (b) weights can be used to prioritize one of the indicators, and in cases (c) and (d) a reasonable cut-off value for the constraints must be estimated.

4.3 ASP Encoding of Strategy S2

We next illustrate the ASP encoding for the signal plan optimization problem. The encoding is divided into two programs P_I, which contains the data encoding, and P_G, which contains the guess-and-check problem encoding.

Data Encoding (P_I). The program P_I must be adapted for each traffic simulation and includes the static components of the flow model, the materialization of all generated NrV data item over time, and the materialization of matrix M holding the initial SPP over time.

We start with the ASP encoding of the (traffic) network graph $G_N = (V, E, l_V, l_E, R_V, R_C)$, where V is represented by predicates `node_src` (for source), sink, link, and intersection nodes, where the predicate `edge` defines the edges of E and predicate `link` constructs the graph based on edges and nodes.

As an example, we give an encoding for five nodes $n_1, ..., n_5$ and four edges, which are linearly connected:

$$\texttt{node\_src}(n_1).\ \texttt{node\_ins}(n_2).\ \texttt{node\_lnk}(n_3).\ \texttt{node\_snk}(n_4).$$
$$\texttt{node\_snk}(n_5).\ \texttt{link}(n_1, n_2, we).\ \texttt{link}(n_2, n_3, we).$$
$$\texttt{link}(n_3, n_4, we).\ \texttt{link}(n_2, n_5, we).$$

The label functions l_V and l_E are simplified given that the bottleneck and load capacities are uniform; e.g., we set $l_V() = 16$ and $l_E() = 10$. This can be represented using constants

$$\#\texttt{const}\ maxflow = 10. \quad \text{and} \quad \#\texttt{const}\ maxcap = 16.$$

The relation R_V represents the traffic distribution of the intersections in the traffic network. We use the predicate $\texttt{link\_ratio}(n_1, n_2, r)$, where $n1$ is an intersection node, n_2 forms with n_1 an outgoing edge $e = (n1, n2)$, and r is the percentiles of vehicles moving out on e. E.g., uniform distribution on the outgoing lanes of n_2 is defined by

$$\texttt{link\_ratio}(n_2, n_3, 50).\ \texttt{link\_ratio}(n_2, n_5, 50).$$

The relation R_C is encoded as $\texttt{conflict\_tl}(J, X, Y)$. It states that two edges (J, X) and (J, Y) are not allowed to have green phases simultaneously. Continuing the example, we we declare that (n_2, n_3) and (n_2, n_5) are conflicting:

$$\texttt{conflict\_tl}(n_1, e_2, n_3).$$
$$\texttt{conflict\_tl}(J, X, Y) \leftarrow \texttt{conflict\_tl}(J, Y, X).$$

The rule here is convenient to ensure that R_C is symmetric.

Besides static facts, we define in program P_I also the temporal horizon by $\texttt{time}(0..t_l)$, where $0..t_l$ is stands for all time points between 0 and the simulation length t_l. For instance, $\texttt{time}(0\ ..\ 15)$ sets the simulation length to 15.

Another important temporal predicate is $\texttt{tl\_plan\_red}(X, Y, T_1, T_2)$, which materializes all red phases up to t_l. The tuple (X, Y) is an incoming edge to an intersection Y, and $[T_1, T_2]$ is a (maximal) interval of red phases. In our example, for intersection n_2 and its incoming edge (n_1, n_2) red phase intervals of length 5 may be defined by

$$\texttt{tl\_plan\_red}(n_1, n_2, 1, 5).\ \ \texttt{tl\_plan\_red}(n_1, n_2, 11, 15).$$

thus implying that between 6 and 10 the signal phase is green.

Problem Encoding (P_G). The program P_G is generic and changes only if parameters must be adjusted. We describe the “guess and check” encoding for SPP configurations by strategy *S2*, along with the encoding of the dynamic part of the mesoscopic flow model.

We start with a non-deterministic rule that generates all possible SPP configurations, which according to *S2* are the intervals $c = [c_n, c_p]$ for shortening/lengthening a red signal phase on an incoming lane. The following example illustrates this for $c = [-1, 1]$, where an incoming lane is given as tuple (X, Y):

$$\{\texttt{change\_red}(X, Y, \text{-}1..1, \text{-}1..1)\} = 1 \leftarrow \texttt{tl\_plan\_red}(X, Y, \_, \_).$$

Note that c is used twice in the rule since the beginning and the end of a SP can be shortened/lengthened. The rule head is an ASP-specific encoding for non-determinism, stating that we generate for an edge (X, Y) stored by $\texttt{tl\_plan\_red}(X, Y, \_, \_)$ (say e.g. (n_2, n_3) in our example) all possible combinations (c_1, c_2) from $\{-1, 0, 1\} \times \{-1, 0, 1\}$. As some of the above generated SPP configurations are invalid, we introduce rules that enforce hard (lane crossing) constraints defined in R_C. Incoming lanes are defined as (X, J) and (Y, J) for intersections J, resulting in the following constraint:

$$\leftarrow \texttt{conflict\_tl}(J, X, Y), \texttt{time}(T),$$
$$\quad\quad not\ \texttt{tl\_calc\_red}(X, J, T),\ not\ \texttt{tl\_calc\_red}(Y, J, T).$$

In the above constraints, the predicate $\texttt{tl\_calc\_red(X, J, T)}$ is used to materialize the effect of $\texttt{change\_red}$ on the SPP as follows:

$$\texttt{t\_calc\_red}(X, J, T) \leftarrow \texttt{tl\_plan\_red}(X, J, T_1, T_2), \texttt{time}(T),$$
$$\quad\quad \texttt{change\_red}(X, J, V_1, V_2), T \geq (T_1 + V_1), T \leq (T_2 + V_2).$$

Next, we describe the main rules for the traffic flow calculation. We use a term $on(x, y)$ expressing that data item x is at node y. First, the predicate $\texttt{move}(X, Z, D, T)$ is used to generate the next move of data times taking the SPP into account, where X is a data item, Z a newly assigned node, D the flow direction, and T the current time point:

$$\texttt{move}(X, Z, D, T) \leftarrow \texttt{pos}(on(X, Z), D, T{-}1), \texttt{link}(Y, Z, D),$$
$$\quad\quad not\ \texttt{tl\_calc\_red}(Y, Z, T), time(T).$$

Second, the predicate $\texttt{pos}(on(X, Y), D, T)$ represents the state after the data item D was moved (first rule) or stays in place (second rule), where $on(X, Y)$ links the data item to the new or same position:

$$\texttt{pos}(on(X, Y), D, T) \leftarrow \texttt{move}(X, Y, D, T),\ T \leq steps.$$
$$\texttt{pos}(on(X, Z), D, T) \leftarrow \texttt{pos}(on(X, Z), D, T{-}1),$$
$$\quad\quad not\ \texttt{moved}(X, D, T),\ T \leq steps.$$

Third, the predicate $\texttt{load}(on(X, Z), V, T)$ encodes the flow model functions f_{lnk}, f_{ins}, and f_{fr1}, The defining rules use *external functions* as provided by more advanced ASP solvers. Such functions (prefixed with @) allow one to use PYTHON code, which we exploit to implement the more complex functions of the flow model. Each flow model function is represented by a single rule with $\texttt{load}$ in the head, where the function $@fIns$ implements f_{ins}:

$$\texttt{load}(on(X, Z), @fIns(V, Y, Z, T, maxflow, maxcap), T)$$
$$\leftarrow \texttt{moved}(X, D, T), \texttt{load}(on(X, Y), V, T{-}1),$$
$$\texttt{link}(Y, Z, D),\ \texttt{pos}(on(X, Y), D, T{-}1),$$
$$\texttt{link\_ratio}(Y, Z, D, \_), T \leq steps.$$

Due to space constraints, we omit the rules for f_{lnk} and f_{fr1} (replace $@fIns$ with the respective implementation).

Optimizations. Optimizations based on the indicators Q_{NrV} and Q_{SPS} can be realized straight in ASP using aggregation, maximization directives and cut-offs via hard constraints. The predicates $\texttt{sum\_traffic}$ and $\texttt{sum\_tl\_red}$ are used for aggregation as follows:

$$\texttt{sum\_traffic}(M) \leftarrow M = \#sum \left\{ \begin{matrix} V, X : \texttt{load\_end}(X, V, T), \\ T{==}steps \end{matrix} \right\}.$$

Note that $\texttt{load\_end}$ is represented by a rule for tracking all data items that have left the road network. The aggregation rule for $\texttt{sum\_tl\_red}$ is similar, where the maximization and cut-off for the above aggregations are defined as follows:

$$\#maximize\ M : \texttt{sum\_traffic}(M).$$
$$\leftarrow \texttt{sum\_tl\_red}(M), M \geq threshold\_red.$$

The first statements effects ordered enumeration of the answer sets by increasing value of M, while the hard constraint drops all answer sets where M is below threshold $threshold\_red.$ The rules for maximizing $\texttt{sum\_tl\_red}$ and restricting $\texttt{sum\_traffic}$ are very similar.

5 Experiments and Results

We conducted two experiments with the aim of evaluating (a) the quality of our ASP-encoded flow model and (b) the suitability of strategy *S2* for improving existing signal plans. We highlight that we frequently found the best configuration of a signal plan shortening/lengthening (for a fixed interval), which could be applied to optimize the traffic flow based on specific traffic demand.

CLINGO 5.3.0 was used to evaluate the ASP programs, which is a state-of-the art ASP system [11]. The experimental setup, ASP encodings, logs, results (including further ones) are available at the webpage http://www.kr.tuwien.ac.at/research/projects/loctrafflog/pais2020.

5.1 Experimental Setup

The models for the experiments must be based on and compared to a microscopic traffic simulation such as SUMO (https://sumo.dlr.de/index.

html), which is the de-facto standard open source tool. We extracted from SUMO the configuration files that describe the road network, the traffic light programs, as well as traffic demand modelling, and generated the ASP data encoding using custom PYTHON scripts.

	$\#v$	t	s_d	t_l	$\#r_r$	$\#r_s$	$\#r_t$
d_1	42	115	6	19	0	6.605	56.14
d_2	126	250	6	42	10	4.407	74.49
d_3	210	360	6	60	19	3.595	92.43

Table 1: Scenarios statistics

Benchmark Data. The benchmark data itself is based on the example shown in Figure 1b, which includes two intersection connecting three roads with two lanes on each road. The *base* signal plan is shown in Figure 3; it equally splits the traffic between two crossing roads. Based on the network, we created three simulation scenarios, where we defined traffic demand models capturing light (d_1), medium (d_2), and heavy traffic (d_3).The statistics on the scenarios is summarized in Table 1, where $\#v$ is the overall vehicle count, t the SUMO simulation length, s_d the interval length, t_l the flow model length, $\#r_r$ the NrV running, $\#r_s$ the mean speed (in m/s), and $\#r_t$ the mean travelling time (in s). Note that $\#r_s$ is the main indicator, which was selected to compare the results.

Calibration. SUMO was also used in the calibration process, whereby we created the mentioned equi-distant segmentation of the road network. An important parameter for the calibration is s_d, which defines how many simulation steps are in a time point, e.g., if $s_d = 2$, two time points are four simulation steps. The interval length influences the estimate for the bottleneck capacity of l_V and load capacity of l_E. The calibration also includes a validation step, where the NrV throughput of the mesoscopic flow model is compared to SUMO simulation runs, to guarantee that the mesoscopic model approximates the microscopic model with a deviation of $< 10\%$.

5.2 Results

We conducted our experiments on a Mac OS X 10.14.4 system with an Intel Core i7 2.9GHz, 8GB of RAM, and a 250GB SSD. The best results of three restarts were taken for each experiment. The given traffic demand model and the deterministic flow model encoding did not yield any deviation; however the evaluation with CLINGO using solving options such as parallel solving with competition-based search can lead to different optima (of similar quality) in runs [10].

Experiment 1. The first experiment serves to calculate the optima based on strategy *S2* by (1) generating all possible candidates that satisfy the hard constraints R_C, (2) simulating each candidate using SUMO, and (3) searching all simulation results and choosing the candidate with highest mean speed $\#r_s$. The results are shown in Table 2, where the columns are as before, c is the interval of SP changes, $dev(\%)$ is the deviation of $\#r_s$ regarding the base plan, and m is the count of generated candidates. Note that all the presented results are the best possible improvements of the base signal plan in regard to the constraints R_C.

Experiment 2. The second experiment serves to run the combined strategy *S2* and the evaluation of the mesoscopic flow model to calculate the quality indicators Q_{NrV} and Q_{SPS}. Based on the indicators, we evaluated the optimization methods of (a) maximizing Q_{NrV} with cut-off Q_{SPS}, and (b) maximizing Q_{NrV} and Q_{SPS}. The results are shown in Table 3 where the columns are as in Table 2 except that $dev(\%)$ is the deviation of $\#r_s$ regarding Experiment 1.

c		$\#r_r$	$\#r_s$	$\#r_t$	$dev(\%)$	$\#m$
d_1	$[-1, 1]$	0	7.026	56.00	6.37	682
	$[-2, 2]$	1	7.134	56.00	8.01	3 521
	$[-3, 3]$	0	7.188	54.93	8.83	13 782
	$[-4, 4]$	1	7.227	52.10	9.42	66 737
d_2	$[-1, 1]$	0	5.346	71.52	21.30	641
	$[-2, 2]$	0	5.557	70.26	26.09	1 128
	$[-3, 3]$	0	5.938	67.60	34.73	10 003
	$[-4, 4]$	0	5.962	66.02	35.28	36 015
d_3	$[-1, 1]$	0	4.697	83.50	30.65	900
	$[-2, 2]$	0	4.810	88.40	33.81	4 763
	$[-3, 3]$	0	5.169	82.16	43.79	10 210
	$[-4, 4]$	0	5.501	77.70	53.01	68 527

Table 2: Results of Experiment 1

c		(a) $\#r_s$	$\#r_t$	$dev(\%)$	(b) $\#r_s$	$\#r_t$	$dev(\%)$
d_1	$[-1, 1]$	6.405	54.93	-8.83	6.548	56.83	-6.80
	$[-2, 2]$	6.539	53.82	-8.35	6.548	56.83	-8.22
	$[-3, 3]$	6.991	53.88	-2.75	6.466	54.95	-10.05
	$[-4, 4]$	6.751	56.38	-6.59	6.466	54.95	-10.53
d_2	$[-1, 1]$	4.544	80.23	-15.01	4.860	83.92	-9.09
	$[-2, 2]$	5.027	68.76	-9.55	5.019	70.97	-9.68
	$[-3, 3]$	5.714	68.16	-3.76	5.372	73.94	-9.54
	$[-4, 4]$	5.962	66.02	0.00	5.962	66.02	0.00
d_3	$[-1, 1]$	4.697	83.5	0.00	4.697	83.5	0.00
	$[-2, 2]$	4.780	86.67	-0.63	4.653	92.22	-3.28
	$[-3, 3]$	4.689	87.39	-9.29	4.625	93.18	-10.52
	$[-4, 4]$	4.843	81.34	-11.95	4.349	91.59	-20.94

Table 3: Results of Experiment 2

Discussion Experiment 1 indicates that calculating optima based on strategy *S2* improves an already suitable base plan by 6.37% to 53.01%. It is particularly satisfactory that the signal plan adjustments are most effective with the heavy traffic demand model d_3. However, the exhaustive search with millions of possible candidates needs an evaluation time starting from below $2\,min$ with d_1 and $[-1, 1]$ changes, ending at an upper limit of $180\,min$ with d_3 and $[-4, 4]$.

If one aims at computing optimal candidates online, combining strategy *S2* with the mesoscopic flow model is crucial, as it allows for the tight integration of the quality evaluation (of a candidate) using the mesoscopic flow model and to optimize the candidate generation iteratively. In Experiment 2, we evaluated the performance of this integrated approach based on the methods (a) maximize Q_{NrV}, cut-off Q_{SPS} and (b) maximize Q_{NrV}, maximize Q_{SPS}. The findings are encouraging, as we could find in two cases the same optimum as in Experiment 1; this is surprising, as the flow model is less accurate than a simulation run in SUMO. Otherwise we only measured a deviation between 0.63% and 11.95% (with an outlier at 15.01%). The use of CLINGO 5.3.0's "enumerate models" functionality was of importance for these experiments; but after a timeout of $5\,min$ we stopped the evaluation and selected the best model as the result [10].

6 Related Work and Conclusion

The potential of AI techniques for traffic management was recognized in [21], stating that for instance Artificial Neural Networks, Genetic Algorithms, or Fuzzy Logic Models could be applied. In the field of ASP, we are not aware of other work, with the exception of [3] where the authors applied ASP to detect inconsistencies in traffic regulations in a static context. In AI-based planning, Vallati et al.'s work [19, 26, 20] is foremost – they extended the PDDL+ planning engine UPMurphi [22] with a traffic-control heuristic using a token-based approach and a PDDL+ encoding that is (as for now) closest to our approach. Linking the PDDL+ encoding with Constraint ASP as in [1] could be interesting. However, we decided to choose a different encoding based on a mesoscopic flow model and three solving and optimization strategies tailored for native and efficient evaluation with native ASP engines.

This work is sparked by the lack of symbolic AI techniques in the field of traffic control systems (TCS) used for traffic optimizations, which allows one (a) to add and change on-demand rules to implement new optimizations strategies, and (b) to replicate microscopic traffic simulation tools for on-the-fly runs to test a strategy. We also introduced a novel ASP-based approach for calculating signal phase plans (SPPs) as used in TCS, including a mesoscopic traffic flow model to calculate the quality of new SPPs, and three strategies for generating new SPPs. We moreover provided an ASP encoding of the traffic flow model and of one strategy (called signal plan adjustments), which we then used to conduct experiments on a benchmark road network. The results indicate (a) the power of signal plan adjustments as effective means to improve traffic flow under heavy network load, and (b) that the mesoscopic flow model makes online evaluation realistic without losing too much accuracy regarding a microscopic simulation.

Future work. Our ongoing and future research is directed towards increasing the expressiveness of the mesoscopic flow model and to an in-depth study of the template-based strategy. In a broader scope, future work should lead to a more systematic investigation by taking other knowledge representation (KR) formalisms into account.

A model extension would be desirable that propagates the NrV data items to down-flow edges, if a level of saturation is reached on an edge. This would allow us to simulate more complex traffic patterns and lead a to more expressive flow model. Furthermore, the strategy *S3* introduces templates in combination with signal plan adjustments. This is currently not feasible, as only the original plans for a network are available; learning new templates would allow us to have "custom" templates for each network. To underline the feasibility of our approach, we aim to conduct a larger experimental evaluation using more complex traffic networks with more diverse flows.

Finally, the proposed modelling and encoding could be transferred to other KR formalisms, which could lead to a more systematic investigation of the traffic flow optimization problem in a KR setting. Event Calculus [16] and Situation Calculus [23] could be suitable candidates, in particular Hybrid Temporal Situation Calculus is an appealing approach, which was already applied to Signal Plan Adjustments [2] using temporal functional fluents.

Acknowledgements. This work has been supported by the Austrian Research Promotion Agency under grant number 861263 (DynaCon).

References

[1] Marcello Balduccini, Daniele Magazzeni, and Marco Maratea, 'PDDL+ planning via constraint answer set programming', *CoRR*, **abs/1609.00030**, (2016).

[2] Vitaliy Batusov, Giuseppe De Giacomo, and Mikhail Soutchanski, 'Hybrid temporal situation calculus', in *Proc. 32nd Canadian Conference on Artificial Intelligence, (Canadian AI 2019)*, volume 11489 of *LNCS*, pp. 173–185. Springer, (2019).

[3] Harald Beck, Thomas Eiter, and Thomas Krennwallner, 'Inconsistency management for traffic regulations: Formalization and complexity results', in *Proc. 13th European Conf. Logics in Artificial Intelligence (JELIA 2012), Toulouse, France, Sept. 26-28, 2012*, pp. 80–93, (2012).

[4] Moshe Ben-Akiva, Haris Koutsopoulos, Tomer Toledo, Qi Yang, Charisma Choudhury, Constantinos Antoniou, and Ramachandran Balakrishna, 'Traffic simulation with mitsimlab', *Fundamentals of Traffic Simulation*, **145**, 233, (06 2010).

[5] R.D. Bretherton, 'Scoot urban traffic control system—philosophy and evaluation1', *IFAC Proceedings Volumes*, **23**(2), 237–239, (1990).

[6] Gerd Brewka, Thomas Eiter, and Miroslaw Truszczyński, eds. *AI Magazine: Special Issue on Answer Set Programming*. AAAI Press, 2016. Volume 37, number 3. Editorial pp. 5-6.

[7] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski, 'Answer set programming at a glance', *Commun. ACM*, **54**(12), 92–103, (2011).

[8] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner, 'Answer set programming: A primer', in *Reasoning Web, 5th International Summer School 2009, Tutorial Lectures*, number 5689 in LNCS, 40–110, Springer, (2009).

[9] Sean Fleming. Future of mobility - Traffic congestion cost the US economy nearly $87 billion in 2018. https://www.weforum.org/agenda/2019/03/traffic-congestion-cost-the-us-economy-nearly-87-billion-in-2018/ accessed Nov 24, 2019.

[10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo, 2010.

[11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider, 'Potassco: The potsdam answer set solving collection', *AI Commun.*, **24**(2), 107–124, (2011).

[12] M. Gelfond and V. Lifschitz, 'The stable model semantics for logic programming.', in *Proc. of ICLP/SLP 1988*, volume 88, pp. 1070–1080, (1988).

[13] Serge Hoogendoorn and V. Knoop, 'Traffic flow theory and modelling, in the transport system and transport policy', *The Transport System and Transport Policy*, 125–159, (01 2013).

[14] Boris S. Kerner, 'Congested traffic flow: Observations and theory', *Transportation Research Record*, **1678**(1), 160–167, (1999).

[15] Femke Kessels, 'Mesoscopic models', in *Introduction to Traffic Flow Theory Through a Genealogy of Models*, chapter 6, 99–106, Springer International Publishing, (2019).

[16] Robert Kowalski and Marek Sergot, 'A logic-based calculus of events', in *Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence Applications*, 23–55, Springer, Berlin, Heidelberg, (1989).

[17] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner, 'Microscopic traffic simulation using sumo', in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, (2018).

[18] John McCarthy, 'Generality in artificial intelligence', *Commun. ACM*, **30**(12), 1029–1035, (1987).

[19] Thomas Leo McCluskey and Mauro Vallati, 'The simplyfai project: Using AI planning in urban traffic management or if at first the representation does not work, try, try and try and again', in *Proc. 34th Workshop of the UK Planning and Scheduling Special Interest Group, PlanSIG 2016*, (2016).

[20] Thomas Leo McCluskey, Mauro Vallati, and Santiago Franco, 'Automated planning for urban traffic management', in *Proc. 26th International Joint Conference on Artificial Intelligence, IJCAI 2017*, pp. 5238–5240, (2017).

[21] J. C. Miles and A. J. Walker, 'The potential application of artificial intelligence in transport', *IEE Proceedings - Intelligent Transport Systems*, **153**(3), 183–198, (Sep. 2006).

[22] Giuseppe Penna, Daniele Magazzeni, Fabio Mercorio, and Intrigila Benedetto, 'Upmurphi: A tool for universal planning on pddl+ problems', in *ICAPS 2009 - Proceedings of the 19th International Conference on Automated Planning and Scheduling*, (01 2009).

[23] Raymond Reiter, *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*, MIT Press, 2001.

[24] Torsten Schaub and Stefan Woltran (eds.), eds. *KI–Zeitschrift: Special Issue on Answer Set Programming*, 2018. Volume 32, number 2-3. Editorial pp. 105-108.

[25] Martin Treiber and Arne Kesting, *Traffic Flow Dynamics: Data, Models and Simulation*, Springer, 2013.

[26] Mauro Vallati, Daniele Magazzeni, Bart De Schutter, Lukás Chrpa, and Thomas Leo McCluskey, 'Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach', in *Proc. 30th AAAI Conference on Artificial Intelligence*, pp. 3188–3194, (2016).