doi:10.1017/S1471068425100306

# *A CASP-Based Solution for Traffic Signal Optimisation*[*]

ALICE TARZARIOL
*AICS, University of Klagenfurt, Klagenfurt, Austria*
(*e-mail:* alice.tarzariol@aau.at)

MARCO MARATEA
*DeMaCS, University of Calabria, Arcavacata, Italy*
(*e-mail:* marco@dibris.unige.it)

MAURO VALLATI
*School of Computing and Engineering, University of Huddersfield, Huddersfield, UK*
(*e-mail:* m.vallati@hud.ac.uk)

*submitted 28 July 2025; revised 28 July 2025; accepted 29 July 2025*

**Abstract**

In the context of urban traffic control, *traffic signal optimisation* is the problem of determining the optimal green length for each signal in a set of traffic signals. The literature has effectively tackled such a problem, mostly with automated planning techniques leveraging the PDDL + language and solvers. However, such language has limitations when it comes to specifying optimisation statements and computing optimal plans. In this paper, we provide an alternative solution to the traffic signal optimisation problem based on Constraint Answer Set Programming (CASP). We devise an encoding in a CASP language, which is then solved by means of *clingcon 3*, a system extending the well-known ASP solver *clingo*. We performed experiments on real historical data from the town of Huddersfield in the UK, comparing our approach to the PDDL+ model that obtained the best results for the considered benchmark. The results showed the potential of our approach for tackling the traffic signal optimisation problem and improving the solution quality of the PDDL + plans.

*KEYWORDS:* applications, constraints answer set programming, urban traffic control

[*] We thank the anonymous reviewers for their valuable feedback and suggestions that helped us extend the paper's contributions. We also thank Martin Gebser for his suggestions on the ASP encoding. Mauro Vallati was supported by a UKRI Future Leaders Fellowship [grant number MR/Z00005X/1]. Marco Maratea was supported by the European Union - NextGenerationEU and by Italian Ministry of Research (MUR) under PNRR project FAIR "Future AI Research," CUP H23C22000860006. This research was funded in part by the Austrian Science Fund (FWF) 10.55776/COE12. For open access purposes, the author has applied a CC BY public copyright license to any author accepted manuscript version arising from this submission.

## 1 Introduction

Urban traffic control aims at minimising average traffic delay in a given region or alleviating the extreme delays of traffic exiting due to major city events and passing through the target area. In this context, *traffic signal optimisation* is the problem of determining the optimal green length for each signal in a set of traffic signals, which may be dispersed around a region consisting of several spatially close traffic junctions. The problem is usually structured by grouping sets of green signals into stages: each signal in a stage shares the same green time, is situated in the same junction, and collectively lets traffic flow through the junction in a safe manner. This structuring leads to a more convenient representation to solve the problem of determining the optimal green length for each stage.

Typical practical approaches in this context consider fixed-time traffic-light phases; thus, with no information about the actual traffic. Model Predictive Controls (see, e.g., Papageorgiou 2013) are hardly used in routine operations, since they are computationally expensive, complicated when it comes to identify the right numerical parameters, and usually slow to converge. Traffic-reactive mechanisms are usually deployed in small sets of neighbouring junctions (between three and nine), and can take decisions in real-time on how to adapt stage duration based on sensor data (Taale *et al.* 1998). A major issue comes from the fact that reactive methods can not leverage on knowledge of incoming traffic that has not yet hit the controlled region, or on wider information about traffic in the area. To support this sort of reasoning, the traffic signal optimisation problem has been more recently tackled in the literature with automated planning techniques (Smith 2020; Vallati and Chrpa 2023) leveraging PDDL + language and solvers (Vallati *et al.* 2016; Antoniou *et al.* 2019; Percassi *et al.* 2023b), then extended to cope with a legacy traffic control infrastructure taking into account deployment constraints (Kouaiti *et al.* 2024). The automated planning solutions demonstrated interesting capabilities, and the generated strategies have been deployed in real-world trials in the Hull city region and in the Kirklees council area in the United Kingdom.

Despite the successful deployments, PDDL + has limitations when it comes to specifying optimisation statements and to computing optimal plans, which are particularly useful in practice to ensure that benefits materialise in the controlled region as soon as possible. At the state of the art, PDDL + planning engines focus on generating satisfying solutions, with no guarantees on the quality of the solution found. In practice, this can lead to traffic signal optimisation plans that show extremely long time horizons – potentially reducing the ability of plans to cope with identified issues. Considering such limitations, the nature of the problem and the additional constraints and domain size of the actual real-world infrastructure of the setting by Kouaiti *et al.* (2024), in this paper we present an alternative, novel solution to the traffic signal optimisation problem based on Constraint Answer Set Programming (CASP) (Mellarkod *et al*. 2008; Liu *et al.* 2012; Balduccini and Lierler 2017; Janhunen *et al.* 2017), which integrates ASP and Constraint Programming.

Our encoding can model the same solution space of the PDDL + models from Kouaiti *et al.* (2024), but limited up to a certain time horizon. After observing that the (pure) ASP representation struggles to scale with meaningful horizons for this problem, we extend it with CASP statements in order to tackle producing plans of higher horizons. In particular, we apply the system *clingcon 3* (Banbara *et al.* 2017), which is an implementation that extends the well-known ASP solver *clingo* with theory atoms and propagators for linear constraints. For the experiments, we compare and evaluate our approach with respect to FiRe, the PDDL + model that obtained the best performance among the proposed alternatives in (Kouaiti *et al.* 2024) and that, on average, showed better plans than the ones used in historical data. The benchmark is drawn from real data and we analysed two tasks: first, evaluating whether *clingcon* is capable of finding or improving solutions with certain restrictions; and second, evaluating the result of *clingcon* with optimisation statements. The results showed that our CASP encoding is a promising approach to tackle the traffic signal optimisation problem with limited horizons.

The paper is structured as follows: Section 2 introduces needed preliminaries about ASP and *clingcon 3* language. Then, Section 3 describes the traffic signal optimisation problem we address in this paper, while our CASP encoding is presented in Section 4. Further, Section 5 shows the results of an experimental analysis comparing our new approach to the best one in (Kouaiti *et al.* 2024), and proposes possible alternative objectives that exploit the optimisation capabilities of *clingcon*. The paper ends in Section 6 and 7 by discussing related literature and drawing final remarks, respectively.

## 2 Preliminaries

*Answer Set Programming (ASP)* (Gelfond and Lifschitz 1991; Niemelä 1999; Baral 2003; Brewka *et al.* 2011) is a declarative programming paradigm that applies non-monotonic reasoning and relies on the stable model semantics. In the following, we describe a fragment of the ASP syntax, focusing on the constructs appearing in our encoding. An ASP programme $P$ is a set of *rules* $r$ of the form: $\texttt{h:- b}_1\texttt{,...,b}_n\texttt{.}$, where $\texttt{h}$ is an atom and each $\texttt{b}_i$, for $1 \leq i \leq n$, is a literal, a comparison, or an aggregate. An *atom* is an expression of the form $\texttt{p(t}_1\texttt{,...,t}_m\texttt{)}$, where $\texttt{p}$ is a predicate and $\texttt{t}_i$, for $1 \leq i \leq m$, are terms. A *literal* $\texttt{c}$ is either an atom $\texttt{a}_i$ or its negation $\texttt{not a}_i$. A comparison is equal to $\texttt{t}_1 \circ \texttt{t}_2$ where $\texttt{t}_1$ and $\texttt{t}_2$ are terms and $\circ \in \{<=, <, =, >, >=\}$. An aggregate is of the form $\texttt{t=\#count}\{\texttt{t}_1\texttt{:c}_1, \cdots \texttt{,t}_n\texttt{:c}_n\}$ or $\texttt{t=\#sum}\{\texttt{v}_1\texttt{,t}_1\texttt{:c}_1, \cdots \texttt{,v}_n\texttt{,t}_n\texttt{:c}_n\}$ where $\texttt{t}$ and each $\texttt{v}_i$ are integers, $\texttt{t}_i$ are terms and $\texttt{c}_i$ are literals, for $1 \leq i \leq n$. The programme $P$ can also contain *choice rules* $r$ of the form: $\{\texttt{a}_1\texttt{:c}_1\texttt{;}\cdots\texttt{;a}_n\texttt{:c}_n\}\texttt{=s:- b.}$, where $\texttt{b}$ is an atom, $\texttt{s}$ is a positive integer and, for $1 \leq i \leq n$, $\texttt{a}_i$ is an atom and $\texttt{c}_i$ is a literal. A rule $r$ is called a *fact* when $n = 0$, and a *constraint* if $\texttt{h}$ is not present. The left side of the symbol `:-` in a rule is called *head*, while the right side is called *body*. The semantics of an ASP programme $P$ is given in terms of the answer sets of its *ground instantiation* $P_{grd}$, computed by replacing each (first-order) rule $r \in P$ with ground rules obtained by substituting the variables in $r$ by constants occurring in $P$ and evaluating arithmetical expressions. An *answer-set* is a collection of (true) ground atoms such that all rules of $P_{grd}$ are satisfied and allow for deriving each of the ground atoms in the head of some rule whose body is satisfied. We refer to Calimeri *et al.* (2020) for more details on the ASP syntax and semantics.

*Clingcon.* Among the various CASP dialects, we focus in this paper on the one of *clingcon 3* (Banbara *et al.* 2017), which is the CASP solver we used in the experiments.

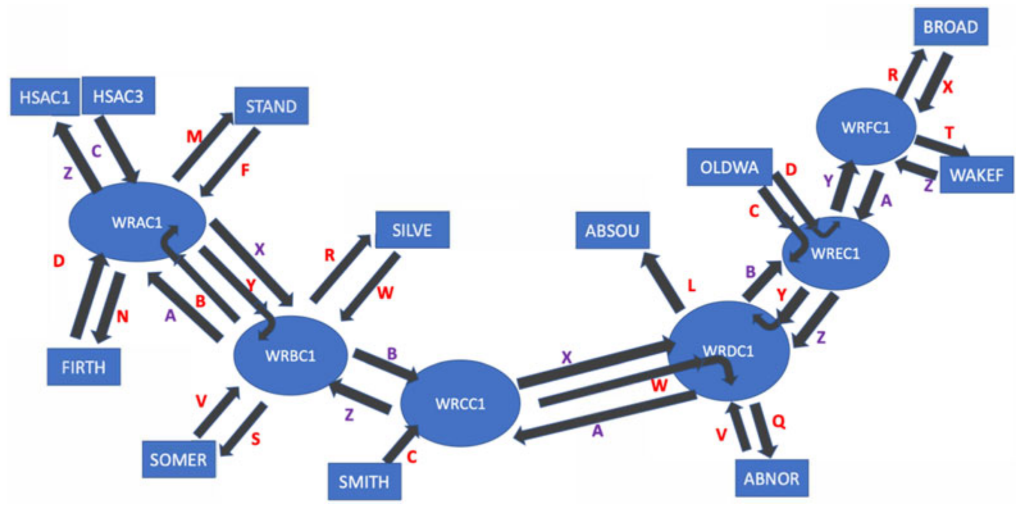


Fig 1. A diagram of the considered corridor in terms of junctions (circles), links, and boundaries (rectangles). For readability, the map is not correctly scaled.

It extends the fifth generation of the ASP system *clingo* by introducing theory atoms in its language and augmenting the solver with propagators for linear constraints, allowing it to deal with integer domains of considerable size. Among its language extensions, *clingcon 3* defines theory atoms as expressions such as `&dom`{`l..u`}=`a`, where `l` and `u` are integers and `a` is a constraint variable, or expressions as `&sum`{`t1;t2;...;tn`}=`t0`, where each `ti` for `i`∈ [`0`..`n`] is a constraint variable or an integer. After grounding, the constraint expressions of the former type are transformed into domain restrictions, requiring that the value associated to `a` is an integer included in [`l`..`u`], while the latter become linear constraint atoms requiring $\sum_{i\in[1..n]}$`ti`=`t0`. Lastly, expressions as `&maximize`{`a`} and `&minimize`{`a`} where `a` is a constraint variable become directives that define as optimal any answer set with the maximal or minimal possible assignment for `a`, respectively.

## 3 Scenario and problem description

This section describes the elements and constraints considered in the traffic signal optimisation problem, in particular using the same formalism and scenario applied in Kouaiti *et al.* 2024. It is worth reminding that this formalism is based on a mesoscopic representation of traffic (Ferrara *et al.* 2018), in which the number of vehicles in road links are considered rather than representing individual vehicles. This is a standard approach to reduce complexity and ensure the solvability of the problems.

The general goal of traffic signal optimisation is to minimise the average traffic delay for a region of interest. In some cases, it can be more specific and focus on alleviating the extreme delays of traffic exiting major city events and passing through the region, or dealing with accidents or unusual events. In this work, we focus on a major corridor in the Kirklees council area within West Yorkshire, United Kingdom, which is approximately 1.3 km long. The corridor, shown in Figure 1 in a simplified form, allows traffic from the Huddersfield ring road (that sits on the left of the corridor) to reach the M62 and M1 highways (right-hand side), and vice versa. Further, it also serves people joining or leaving events hosted at the nearby John Smith's Stadium. The corridor includes 6 *junctions* and considers 34 road *links*.

Each junction contains traffic signals, the status of which determines whether vehicles on certain (incoming) roads are allowed to proceed through to other (outgoing) ones. In the problem representation, the concept of traffic signals in a junction is abstracted, and instead we use the concept of the junction's *stage*, that is, set of traffic movements that can be active at the same time on the junction, hence controlling the flows of vehicles between connected links. A *cycle* is a complete sequence of all stages, defining their order. In our scenario, stages are always interleaved by *intergreen* times, that is, times where no traffic movement occurs in the junction. This occurs when all the junction's traffic signals are on red to allow pedestrians to cross and vehicles still transiting in the junction to leave the area before the next stage begins. Constraints on the problem include the legal and practical restrictions on the minimum and maximum duration of each stage, and the minimum and maximum length of the overall traffic signal cycle. The order of stages in a cycle can not be modified.

We can then formalise the problem's objective as optimising the length of traffic signal stages for each junction in the controlled urban region, to minimise the average traffic delay. In the focus area, the SCOOT (Taale *et al.* 1998) system is in operation for performing traffic signal optimisation. SCOOT is a traffic reactive control mechanism used widely around the world, and is aimed at handling cycle-to-cycle changes in traffic demand. In response to changes in traffic flows, SCOOT would gradually adapt the traffic signal timings of a set of managed neighbouring junctions. The adapting process is gradual, in the range of 4–8 s difference per cycle, and naturally discretised: the minimum granularity is 1 s. In performing its task, SCOOT is dependent on its own local data sensors, usually inductive loops embedded in the road surface, and stores sensed data and operational information in a dedicated database.

Exploiting the architecture proposed by Bhatnagar *et al.* (2023), we can extract information from the SCOOT infrastructure and simulate historical data and generated solutions. Such architecture allows the use of external tools to perform traffic signal optimisation, and to inject the generated strategies to be deployed in the region. To allow deployment on heritage traffic control infrastructure (Kouaiti *et al.* 2024), the traffic signal optimisation problem needs to be redefined by considering the following additional constraints: (i) the length of the stages can not be modified arbitrarily; instead, for each junction, the *configuration* of cycles (i.e., the specification of the length of every stage in the cycle) can only be selected from a predefined set, and (ii) the cycles considered for the junctions in the controlled region should have approximately the same duration – to avoid synchronisation issues and ensure that green waves are preserved.

The problem formalisation applied in (Kouaiti *et al.* 2024) abstracts the vehicle capacity of all the road links by considering instead the numbers of "passenger car units" (PCU), which is the standard unit for measuring traffic flows, corresponding to the typical passenger car. Moreover, it represents with the *turnrate* the average traffic flows between links in number of PCU's per second, that is, the number of vehicles flowing through a particular junction at a certain time of day, when the corresponding traffic signal stage is green. Lastly, to minimise the average traffic delay, the problem formalisation uses the concept of *counter* to measure the number of vehicles that navigated through the link over a considered period of time, and was introduced in Percassi *et al.* (2023a) to support the heuristic reasoning of the planning engine. Increasing the number of vehicles that navigate the region in a given period of time is a proxy for minimising average delay, as it aims at increasing the throughput of the network, hence reducing time wasted queuing. In this work, we follow the same concept.

## 4 CASP modelling

This section introduces our representation of the traffic signal optimisation problem by means of ASP and *clingcon 3* language. We start by illustrating the facts contained in each problem instance in Subsection 4.1, and then we introduce our encoding. To improve the readability, we split it into two parts, first describing the rules where exclusively ASP is used, followed by the part where *clingcon* expressions appear. The Subsection 4.2 models the decision points of the solver, that is, the time where a configuration can be selected for each junction, and the status of the junctions at each time, that is, which stage or intergreen time is active. On the other hand, Subsection 4.3 uses *clingcon* expressions to model the occupancy, that is, the number of PCU present at each moment, and the counter of each link, that is, the number of PCU entering it. Differently from the PDDL+ models, where the heuristics of the planner are used to return a promising sequence of configurations to reach a target value for every counter, with our ASP model we can define an optimisation statement aiming to maximise the values of counters within a given horizon.

### *4.1 Problem instances*

Problem instances represent the initial setting and status of the corridor to be managed, containing a set of facts defining the following atoms: `controllable(J)` identifies every junction `J` for which the solver can change its stages' green light duration, selecting from the set of available configurations, defined through the atoms `available_conf(J,C)`. In our encoding, a cycle is represented as a sequence of phases, where a phase is either a traffic-light stage or an intergreen time. Specifically, for each configuration `C` the set of atoms `phase_limit(P,C,D)` defines the duration `D` of the phase `P` (where `P` is either a stage or an intergreen). The atoms of the form `status(J,P)` specify the phases `P` occurring in a junction `J`; the order of phases in a cycle is defined through the atoms `next(P1,P2)`, where the phase `P1` is followed by `P2`, and `end(P)` identifies the final intergreen time. Listing 1 contains a simple example of ASP atoms characterising the junction cycle in Figure 2.

The atoms of the form `link(J1,ID,J2)` represent (directed) road links connecting the junction `J1` to `J2` and are identified by `ID` to avoid ambiguities for the same connections. Thus, we can represent the top left links in Figure 1 as, for example, `link(wrac1,z,hsac1)` and `link(hsac1,c,wrac1)`. For the sake of compactness, we use the unary predicate `link(L)`, where `L` is equal to `link(J1,ID,J2)`. Moreover, the atoms `precedes(J,L)` and `follows(J,L)` identify the junction `J` preceding and following

```
available_conf(j1,j1_c1).            available_conf(j1,j1_c2).
next(stage(j1,1),inter(j1,1)).       next(stage(j1,2),inter(j1,2)).
next(inter(j1,1),stage(j1,2)).       next(inter(j1,2),stage(j1,2)).
status(j1,stage(j1,1)).        status(j1,inter(j1,1)).
status(j1,stage(j1,2)).        status(j1,inter(j1,2)).      end(inter(j1,2)).
phase_limit(stage(j1,1),j1_c1,12).   phase_limit(inter(j1,1),j1_c1,2).
phase_limit(stage(j1,2),j1_c1,7).    phase_limit(inter(j1,2),j1_c1,4).
phase_limit(stage(j1,1),j1_c2,8).    phase_limit(inter(j1,1),j1_c2,2).
phase_limit(stage(j1,2),j1_c2,11).   phase_limit(inter(j1,2),j1_c2,4).
```

Listing 1. ASP facts describing the cycle in Figure 2.

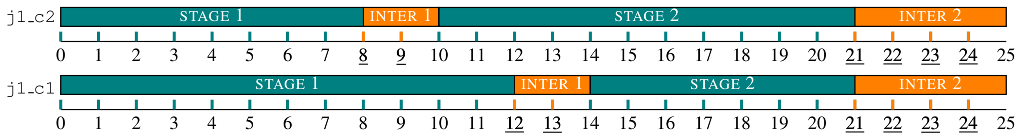


Fig 2. Example of cycles of 25 seconds with two stages for two configurations, `j1_c1` and `j1_c2`.

each link `L`, respectively. Every link `L` has a maximum capacity and initial occupancy, contained in the second term of `capacity(L,C)` and `initial_occ(L,O)`, respectively. Capacity defines the maximum amount of PCU that can be in the link at the same time, while occupancy indicates the actual amount of PCU in the link in the beginning of the simulation. If the atom capacity is not defined for a link, then we assume that there is no limit for it. Moreover, the atoms `turnrate(S,L1,L2,U)` contain the PCU `U` that transits from a link `L1` to link `L2` during the stage `S` per unit of time (i.e., second). The capacity, occupancy and turn rate numbers have a precision of five decimal digits and are normalised by multiplying each number by $10^5$. The initial status of the corridor (i.e., time 0) is described by (i) the active phase in each junction, entailed by the second term of the atoms `active_p(0,P)` (with just one active stage `P` per junction), (ii) the amount of time since `P` is active, entailed by the third term of the atoms `active_t(0,J,T)` (note that `J` is the junction where `P` is active), and (iii) the active configuration for each junction, defined through the third term of `active_c(0,J,A)`. Lastly, the atoms `initial_count(L,D)` contain the links `L` for which we want to maximise the flow of vehicles, where `D` is the initial value that we assign to the counter (by default, it is equal to zero).

### *4.2 ASP encoding*

Our ASP representation simulates the status of the corridor at every second, from time 0 up to the `horizon`. The unique choice rule in our encoding selects the active configuration for each junction. The decision points can be precomputed since changing configuration during a cycle is not allowed and the length of different configurations must coincide, as specified in Section 3. Listing 2 contains the rules to determine the decision points and configuration for every junction. The rule in line 1 computes in `cycle(J,D)` the duration `D` of a cycle for a junction `J`. The atoms `prev_status(P,P1)` enumerate, for each phase `P`, all its previous phases `P1` up to the beginning of the cycle. The rule in line 6 calculates

```
 1 cycle(J,D) :- available_conf(J,C), D=#sum{T,P: phase_limit(P,C,T)}.
 2
 3 prev_status(S,S1) :- next(S1,S), not endcycle(S1).
 4 prev_status(S,S2) :- prev_status(S,S1), next(S2,S1), not end(S2).
 5
 6 sub(J,T+M):- active_conf(0,J,A), status(J,S), active_t(0,J,T), active_p(0,S),
 7              M=#sum{L1,S1: prev_status(S,S1), phase_limit(S1,A,L1)}.
 8 step(J,1,D-M) :- controllable(J), cycle(J,D), sub(J,M), D-M<=horizon.
 9 step(J,C,T+D) :- step(J,C-1,T), cycle(J,D), T+D<=horizon, C>1.
10
11 {conf(J,C,T,A): available_conf(J,A) } = 1 :- step(J,C,T).
12 conf(J,0,0,A) :- active_c(0,J,A).
13
14 change(J,C,T,A) :- conf(J,C,T,A), 0<C, not conf(J,C-1,_,A).
15 :- conf(J,0,0,A), count_c(J,I), C=1..k-I-1, not conf(J,C,T1,A), step(J,C,T1).
16 :- change(J,C,T,A), I=1..k-1, not conf(J,C+I,T1,A), step(J,C+I,T1).
```

Listing 2. Encoding part 1 - Define decision points and set configuration.

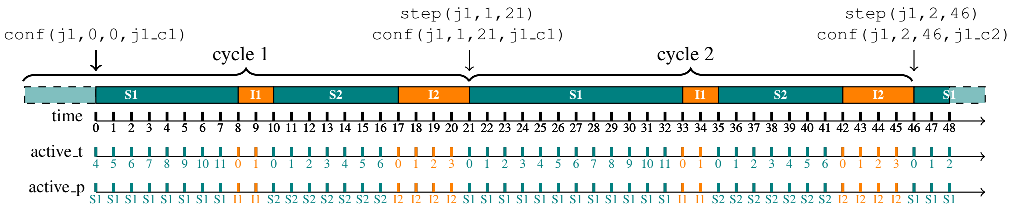


Fig 3. Example of simulation of junction `j1` from Figure 2, with `horizon` = `48`, `active_p(0,stage(j1,1))`, `active_t(0,j1,4)` and `active_c(0,j1,j1_c1))`. S and I are shorthand representations of stages and intergreen times, respectively.

with the atoms `sub(J,S)` the seconds `S` since the current cycle in `J` was active, up to the point where the simulation has begun. The rules in lines 8 and 9 use these atoms to derive `step(J,C,T)`, which identifies for each controllable junction `J` the time point `T` in which the `C`-th cycle ends; in other words, `T` is the `C`-th time point where the configuration of `J` can be changed. Lastly, line 11 contains the choice rule for `conf(J,C,T,A)` that selects the active configuration `A` for each atom `step(J,C,T)`, while its value at time 0 is defined in line 12. Figure 3 illustrates a possible example of the predicates `conf` and `step`. Our encoding can model the solution space up to a specific horizon of the three models presented in Kouaiti *et al.* (2024), namely CbC, FiRe, and VaRe. Their difference regards the restrictions on forcing to keep the same configuration for at least a certain number of cycles. We define a constant `k` representing this restriction. The rule in line 14 detects with `change(J,C,T,A)` the time `T` (i.e., end of `C`-th cycle) when the configuration for `J` is changed to `A`; these atoms are then used in the constraints in lines 15 and 16 to rule out unwanted solutions (the term `I` in `count_c(J,I)` is the number of `J`'s completed cycles at time 0 since its configuration was changed).

Once the atoms of the predicate `conf` are defined, the rules in Listing 3 are applied to compute the active predicates for the interval of time points ranging from 1 to `horizon`, entailed by `time` in line 17. The rule in line 18 defines the auxiliary atoms `range(S,A,B,E)`, computing for each configuration `A` the terms `B` and `E`, which represent the starting and ending time (relative to the cycle) for each phase `S`. For instance, for

```
17 time(1..horizon).
18 range(S,A,M,M+L-1):- available_conf(J,A), phase_limit(S,A,L),
19                      M=#sum{L1,S1: prev_status(S,S1), phase_limit(S1,A,L1)}.
20 active_p(T,S) :- conf(J,0,0,A), sub(J,M), range(S,A,B,E), M<E, B<=M, T=1..E-M.
21 active_p(T,S) :- conf(J,0,0,A), sub(J,M), range(S,A,B,E), M<B, T=B-M..E-M.
22 active_p(T,S) :- conf(J,C,T1,A), C>0, range(S,A,B,E), T=T1+B..T1+E, time(T).
23
24 active_t(X,J,X+M-B):- conf(J,0,0,A),sub(J,M),range(S,A,B,E),M<E,B<=M,X=1..E-M.
25 active_t(X,J,X+M-B):- conf(J,0,0,A),sub(J,M),range(S,A,B,E),M<B,X=B-M..E-M.
26 active_t(T+X,J,X-B):- conf(J,C,T,A),C>0,range(S,A,B,E),X=B..E,time(X).
27
28 active_c(1..horizon,J,A):- conf(J,0,0,A), not step(J,1,_).
29 active_c(1..T,J,A)      :- conf(J,0,0,A), step(J,1,T).
30 active_c(T..T1-1,J,A)   :- conf(J,C,T,A), step(J,C+1,T1), C>0.
31 active_c(T..horizon,J,A):- conf(J,C,T,A), not step(J,C+1,_), C>0, horizon>=T.
```

Listing 3. Encoding part 2 - Define `active` predicates from time 1 to `horizon`.

the configuration `j1_c1` in Figure 2, we get `range(stage(j1,1),j1_c1,0,11)`, `range(inter(j1,1),j1_c1,12,13)`, `range(stage(j1,2),j1_c1,14,20)` and `range(inter(j1,2),j1_c1,21,24)`. Then, lines 20-22 define at each time `T` the atoms `active_p(T,P)`, containing in `P` the active phase for each junction. The rules in lines 24-26 define the atoms `active_t(T,J,TS)`, containing in `TS` the amount of time since `P` (phase of `J`) is active, at time `T`, and lastly, lines 28-31 define with the atoms `active_c(T,J,A)` the active configuration `A` for junction `J` at time `T`. Lines 20-21, 24-25, and 28-29 compute the values for the time points included in the first cycle, while the other lines compute the values for the subsequent ones. The last two timelines in Figure 3 represent the pointwise values for `active_t` and `active_p`; while the atoms of `active_c` have the configuration `j1_c1` active for every time point ranging between 1 a 45, and `j1_c2` from 46 to 48.

## *4.3 Encoding with CASP atoms*

To decide a promising configuration assignment, the number of PCU transiting in the corridor during the considered interval must be taken into account. To compute this information, we need to evaluate at each time point the unit of vehicles transiting in each link, considering the active stage of the adjacent junctions and the number of cars in each link (no cars can transit from one link, if its occupancy is negative, and no cars can transit in one link, if it has already reached its capacity). Because of the size of the grounding when considering a horizon greater than 100 seconds, it was necessary to extend the ASP encoding with CASP language (in particular, *clingcon*). In the following, we directly describe this reformulation using *clingcon* expressions (described in Section 2). The theory atoms consider the predicates `occ(T,L)` and `counter(T,L)`, associated respectively to an integer number representing the occupancy and the number of cars reaching the link `L` at time `T`. The theory atoms `&dom{Lo..Up}=occ(T,L)` defines the minimal and maximal number that can be assigned to each `occ(T,L)`, where `Lo` and `Up` are the value zero and the corresponding link's capacity (or a default value if the latter is not specified) plus an approximation error.

```
32 full(T,L,0) :- &sum{occ(T,L)}>=C, time(T), capacity(L,C).
33 full(T,L,1) :- time(T), link(L), not full(T,L,0).
34 empty(T,L,0) :- &sum{occ(T,L)}<=0, time(T), link(L).
35 empty(T,L,1) :- time(T), link(L), not empty(T,L,0).
36
37 in_ord(L,L1,N) :- turnrate(_,L1,L,_), N=#count{L2: turnrate(_,L2,L,_),L2<=L1}.
38 out_ord(L,L1,N):- turnrate(_,L,L1,_), N=#count{L2: turnrate(_,L,L2,_),L2<=L1}.
39 last_in(N,L)    :- link(L), N=#count{L1: in_ord(L,L1,M)}.
40 last_out(N,L)   :- link(L), N=#count{L1: out_ord(L,L1,M)}.
41
42 delta(T,0,L,0) :- time(T), link(L).
43 delta(T,N,L,D+R*E*F) :- time(T), precedes(J,L), status(J,S), active(T-1,S),
44                         delta(T,N-1,L,D), in_ord(L,L1,N), turnrate_z(S,L1,L,R),
45                         empty(T-1,L1,E), full(T-1,L,F).
46 delta(T,M+N,L,D-R*E*F) :- time(T), follows(J,L), status(J,S), active(T-1,S),
47                           delta(T,M+N-1,L,D), last_in(M,L), out_ord(L,L1,N),
48                           turnrate_z(S,L,L1,R), empty(T-1,L,E), full(T-1,L1,F).
49 &sum{D}=occ(0,L) :- initial_occ(L,D).
50 &sum{occ(T-1,L);D}=occ(T,L) :- delta(T,I+O,L,D), last_out(O,L), last_in(I,L).
51 &sum{D}=counter(0,L) :- initial_count(L,D).
52 &sum{counter(T-1,L);D}=counter(T,L) :- delta(T,O,L,D), last_in(O,L).
53 :- initial_count(L,_), &sum{counter(horizon,L) } < bound.
54 &maximize{counter(horizon,L) : initial_count(L,_)}.
```

Listing 4. Encoding part 3 - Theory atoms for occupancy and counter.

Listing 4 contains the rules used to define the theory atoms and their auxiliary predicates. The rules in lines 32 and 33 define the auxiliary atoms `full(T,L,B)` where `B` is equal to 1 if `L` has not reached its capacity at time `T` and 0 otherwise. Similarly, lines 34 and 35 define the auxiliary atoms `empty(T,L,B)` where `B` is 1 if `L` is not empty at time `T` and 0 otherwise. The rule in line 37 defines `in_ord(L,L1,N)` that contains, for each link `L`, its incoming link `L1`, where `N` is its rank according to the lexicographic order; similarly, in line 39, the atoms `out_ord(L,L1,N)` contains the same information, but for the outgoing links of `L`. Lines 39 and 40 define, respectively, `last_in(N,L)` and `last_out(N,L)`, two auxiliary predicates containing the number `N` of incoming and outgoing links for each `L`. All these atoms are used to compute, for each link `L` and time `T`, a set of `N` atoms `delta(T,N,L,D)`, incrementally computing the value that must be summed to the theory atoms with `occ(T-1,L)` to obtain `occ(T,L)`. Each delta starts from zero (line 42), then, following the lexicographic order of its incoming links (thanks to the predicate `in_ord`), it incrementally adds to the previous delta the turn rates of the current active phase, only if the incoming link is not empty and `L` is not full (rule in line 43). Note that we use the predicate `turnrate_z`, which is not included in the listing but generalises `turnrate` by adding the case when the turn rate is zero (i.e., not defined by the original `turnrate`). Lastly, line 46 incrementally diminishes the value of delta, considering the turn rates of each outgoing link, similarly to the rule above. Lines 49 define the theory atoms with `occ` for each link at time 0, initialising them with their initial occupancy; subsequently, line 50 computes each link's occupancy at time `T` by adding the value of the link's last delta to its occupancy at time `T-1`. Lines 51 and 52 define the theory atoms for `counter`, which is computed similarly to `occ`, but without considering the outgoing links (thus, it sums the value of delta obtained up to the last incoming link). Similarly to the goal restrictions of the PDDL + models, the constraint in line 53 removes solutions with a value `counter` lower than `bound` at the time corresponding to `horizon`. Additionally, we add the optimisation objective in line 54, that is, maximising the number of cars transiting in the links contained in `initial_count`.

## 5 Experiments

This section describes the experiments we conducted to evaluate the performance of our approach and discusses possible applications. In Subsection 5.1, we detail the characteristics of the problem instances introduced in (Kouaiti et al., 2024) considered for our experiments. Then, Subsection 5.2 details the experiments we run, comparing the performance of our suggested approach with the PDDL + version. Lastly, in Subsection 5.3, we highlight the optimisation capabilities of *clingcon*, by suggesting possible alternative objectives and a way to combine PDDL + with our model to guarantee that a solution is always found.

### *5.1 Benchmark*

The benchmark contains six situations in two distinct days on the corridor discussed in Section 3, and shown in Figure 1: the 26th, which is a Wednesday, and the 30th, a Sunday, both in January 2022. Each day was examined at three different time slots: the morning peak hour at 8:30 am, noon at 12:30 pm, and the evening peak hour at 4:30 pm. This provides variability in terms of traffic volumes, directions, and conditions. Further, an additional situation is considered, involving exceptional traffic circumstances, that is, a concert held at John Smith's Stadium on Tuesday the 20th of June 2023, which attracted an approximate audience of $30,000$ people. The time considered is 4:00 pm, which is before the start of the concert. This is interesting because there is a clash between commuters leaving the town and spectators arriving at the concert, creating two opposed traffic demands. For each junction, six different cycle configurations are available, generated according to historical data. Two different sets of cycle configurations are considered, according to the historical data used for their extrapolation, for a total of 14 scenarios $(7 \times 2)$.

Five instances are produced for every scenario by consistently increasing the number of links in the corridor considered in the goal, starting from `link(wrac1,y,wrbc1)`, till considering `link(wrec1,y,wrfc1)`. For each scenario, we identify the instances as $p_i$ for $i \in [1..5]$ where $i$ represents the number of goals. The different numbers of goals correspond to different requirements and behaviours in the region. Focusing on one or two links in the goal can lead to flushing vehicles out of them as soon as possible, potentially congesting nearby ones. When larger chunks of the corridor are considered, there is the need to ensure traffic remains smoothly moving in the whole area. Overall, we consider 70 problem instances.

### *5.2 Experiment setting and results*

We run our experiments on an AMD EPYC 9354 (4) @ 3.2 GHz machine with 32 GB of memory under Linux (Ubuntu 22.04.5 LTS), using the system *clingcon* (v5.2.1) with the *libclingo* v5.8.0. To run the PDDL + solver, *enhsp* (Scala *et al.* 2020), we use *java* *openjdk* (v21.0.6). The instances, ASP and *clingcon* encoding, as well as the experiment setup can be found at this link https://github.com/altarzariol/traf_sign_casp.

Our *clingcon* encoding models the same solution space (up to a certain horizon) of the PDDL + models from Kouaiti *et al.* (2024), in particular, in the experiments we focus on FiRe (i.e, setting the limit $k = 4$ for every junction) since it is the model obtaining the best performance among the proposed PDDL + alternatives and that, on average, showed better plans than the ones used in historical data. In the paper, the authors set a threshold of 350 vehicles in every goal, and evaluate the plans found by cutting their horizons up to 900 s (i.e., 15 min). Limiting the plan's horizon for the evaluation was necessary in order to consider only simulations consistent with the real-world; indeed, when considering horizons greater than 15 minutes, the results diverge from the simulations because of the shifting of underlying turn rates (Bhatnagar *et al.* 2022b). Therefore, for our *clingcon* encoding, we set the constant `horizon` up to 900. Moreover, while state-of-the-art PDDL+ approaches for traffic signal optimisation can generate solutions quickly, in many cases, strategies are generated in advance, and validated and tested before being deployed in the controlled region – effectively enabling the use of approaches that could generate higher quality solutions more slowly. For this reason, in the following experiments, we set a timeout of 10 minutes for every run.

To validate the simulation obtained with our *clingcon* encoding, we compare the status of the corridor and decision points with respect to FiRe, using the `pps` simulation tool https://github.com/hstairs/pps: this approach demonstrated to support simulations that are close to real-world traffic evolution within 15 minutes time windows (Bhatnagar *et al.* 2022a). We evaluate two tasks: in the former, we compare our encoding to the PDDL + model FiRe on a similar setting, that is, the decision version of the problem; while, in the latter, we compare the result of FiRe with the optimisation version of our *clingcon* encoding. For both tasks, we run *clingcon* with the flag `--config=crafty` since this is the option that obtained the best performance in our experiments.

*Task 1: decision problem with bound.* The horizon of the plans returned by FiRe with goals of 350 vehicles is higher than 15 minutes. Thus, to provide a fair ground to compare the two approaches, we set the constant `bound` for each instance in the benchmark by considering the minimal counter observed in the plans of FiRe at $10, .., 15$ minutes (namely, $600, .., 900$ seconds) and setting the constants `bound` and `horizon` accordingly. Figure 4 shows the aggregated improvement of the values `counter` in the corresponding horizon, considering only the instances for which *clingcon* managed to find a plan within a timeout of 10 minutes. In the axis "Horizon," we report the value `horizon` used by *clingcon* and the limit used for the evaluation of the plan found by FiRe, while in parentheses we specify the number of considered instances (i.e, those for which *clingcon* found a solution equal or better than the given bound within the timeout). The results indicate that, despite the struggle with the grounding size, *clingcon* can be used to improve PDDL+ solutions. Moreover, although for 43 instances with an horizon of 900 seconds we get a timeout, the solutions found for the remaining 27 instances obtained a considerable improvement.

*Task 2: optimisation problem without bound.* Subsequently, we evaluate our encoding by running it with the optimisation statement in line 54 of Listing 4. We set a time limit

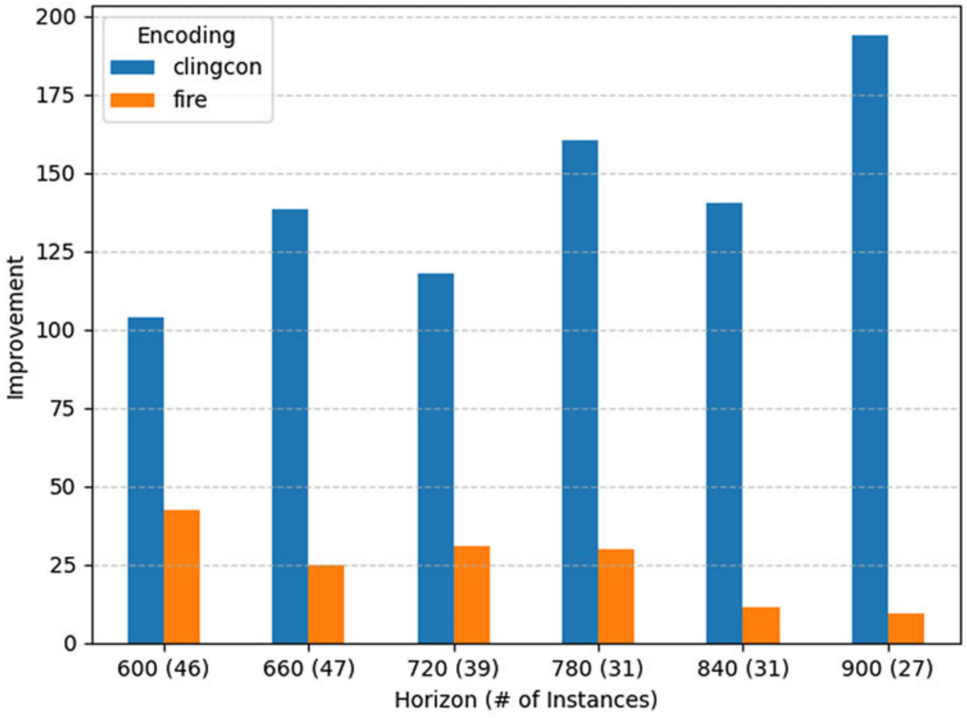


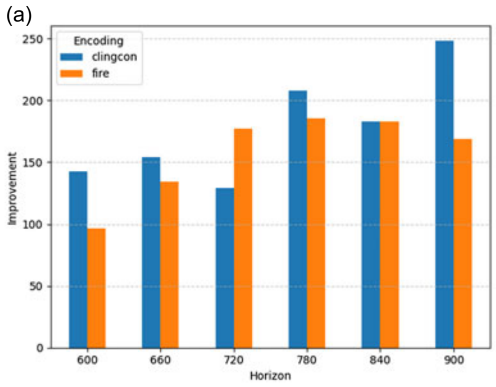
Fig 4. Task 1 - decision version with bound.

of 10 minutes, no request on the bound (thus, `bound` = `0` and `horizon` = `900`), and the results considered are derived from the best solution obtained within the time limit.

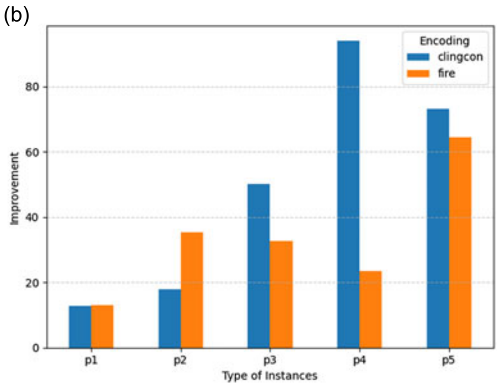
Figure 5a shows the aggregated improvements observed for the 70 instances of the benchmark, by projecting the plan of the two approaches at different time points. Although not every *clingcon*'s plan improves the quality of the solutions returned by FiRe, the overall improvement from its successful runs overcomes the one of FiRe. The only exception can be seen at 720 seconds. One possible explanation for this observation is that, in that moment or slightly before, most of FiRe solutions maximise the flow in the corridor, possibly penalising neighbouring links, and subsequently leaving the corridor with low occupancy, thus requiring more time to increase the counter consistently. Figure 5b, on the other hand, projects the aggregated result at horizon 900, dividing it by type of instances. Here we can observe that the majority of *clingcon* improvements derive from instances with three, four or five goals. One reason for this observation is that, by focusing on one or two links, it is easier to congest the link below, and then get worse overall results. Lastly, in both tasks, we observe that *clingcon* struggles with returning a solution when many decision points occur in the simulated horizon.

### *5.3 Alternative optimisations and combination*

Thanks to its optimisation capabilities, *clingcon* can express alternative optimisation goals, making the suggested model modular and capable of adapting to different targets.



Aggregated results projecting horizon.



Aggregated results at horizon 900.

Fig 5. Task 2: optimisation version without bound.

Indeed, one may want to increase the occupancy of a link as much as possible – a proxy for reducing vehicles' speed on that link. Another goal can be to reduce the occupancy of a link as much as possible, to optimise the flow of traffic from a specific entry point/direction. In both cases, the most effective way to indirectly express the optimisation statement is to maximise/minimise the values computed with the predicates `delta`, which represent the increment/decrement in the occupancy of a link at every time. This allows for reducing the size of the values considered, which can quickly become intractable when considering the difference between occupancy and capacity. To do that, we can simply add the following *clingcon* expressions to define a new theory atom `increments(T,L)` that represents the total PCU that entered in/exited from the link `L` identified by the input atoms `max_occupancy(L)` (if we consider maximisation) at time `T`:

```
&sum{0}=increments(0,L):- max_occupancy(L).
&sum{increments(T-1,L);D}=increments(T,L):-
delta(T,I+O,L,D), last_delta_out(O,L), last_delta_in(I,L),
max_occupancy(L).
```

Then, we maximise the occupancy of the target links with the following optimisation statement:

```
&maximize{increments(horizon,L) : max_occupancy(L)}.
```

If, on the other hand, we aim to minimise the occupancy of target links, we can simply add the same rules but instead of `max_occupancy(L)`, we represent the target with `min_occupancy(L)` and then, instead of `&maximize` we write an optimisation statement with `&minimize`.

It is also possible to encode goals where counters on exit links have to be minimised – to represent cases where we want to reduce traffic pressure on a nearby region by slowing down vehicles in the controlled one. Similarly to the previous case, to achieve this, we can simply replace the directive `&maximize` with `&minimize` in line 54 of Listing 4.

Lastly, more complicated goals can be used to represent traffic accident management, where traffic before the accident has to be slowed down (minimise counter value), while traffic on that link and subsequent ones has to be flushed away as soon as possible (maximise corresponding counters). To model this, we can identify the links before and after the accident, for example, with `slow_traffic(L)` and `flush_traffic(L)`, and write the following optimisation statements:

```
&maximize{counter(horizon,L) : flush_traffic(L)}.
&minimize{counter(horizon,L) : slow_traffic(L)}.
```

As long as all these optimisation statements do not aim to pursue contrasting objectives, for example, by specifying the same link for `max_occupancy(L)` and `min_occupancy(L)`, we can include more than one optimisation target in the encoding and even specify the priority level, similarly to weak constraints for ASP.

*Combining PDDL + and CASP.* The *clingcon* encoding can also be used to improve the quality of solutions returned by the PDDL + approach, hence combining the strengths of the approaches. From the solution found by the PDDL + planner, the values of `counter` at a given horizon of each target link can be extracted. This information can be encoded in atoms of the form `pddl_solution(L,C)`, where `L` is a target link for the optimisation and `C` is the value of `counter` obtained by the PDDL + plan. Then, by including in our encoding the following constraint, we can force *clingcon* to return a solution that is strictly better than the PDDL + one:

```
:- &sum{counter(horizon,L) : pddl_solution(L,_)} <= S,
   S=#sum{B,L: pddl_solution(L,B)}, pddl_solution(_,_).
```

We implemented an automated pipeline to exploit the synergies of the two approaches and tested it on the same benchmark and settings used for our experiments. We considered horizons of $10, \ldots, 15$ minutes. By combining PDDL + with ASP, we managed to improve the quality of almost half of the solutions. The automated pipeline and the results of the experiments can be found at this link https://github.com/altarzariol/traf_sign_casp.

## 6 Related work

A large number of planning and scheduling-based approaches have been developed for traffic signal optimisation. Gulić *et al.* (2016) proposed a system that integrates an AI planning engine with the Sumo simulator (López *et al.* 2018) via an "Intelligent Autonomic System" module. Their pddl2.1 model utilises relative density descriptors (e.g., "low," "medium") to represent traffic concentration on road links, abstracting away from individual vehicle counts. This approach enables scalability to regions with thousands of vehicles. The work by Pozanco *et al.* (2021) builds upon this approach, introducing also the ability for continuous learning and knowledge model evolution for improved network adaptation. The preliminary work by Ivankovic *et al.* 2022 performs traffic signal optimisation by leveraging on planning techniques that reason with global state constraints (Haslum et al. 2018), which can provide valuable insights into the broader impact of light changes.

On a different line of work, (Vallati *et al.* 2016 and McCluskey and Vallati 2017) exploits PDDL + for encoding a flow model of vehicles through traffic-light controlled junctions. Those initial works have then been extended in (Percassi *et al.* 2023b and Kouaiti *et al.* 2024), where the proposed approaches have been extensively validated with historical data from urban regions in Manchester and Huddersfield, from the northern part of the United Kingdom. These most recent works take into account the constraints of the existing infrastructure and are hence suitable for deployment. Finally, Percassi and Vallati (2025) assesses the suitability of LLMs to generate valid and effective traffic signal configurations from scratch, and Percassi and Vallati (2024) demonstrates how to perform a what-if analysis on the basis of the engineered knowledge models.

From a different perspective, the SurTrac system leverages a decentralised scheduling technique for urban traffic signal control (Xie *et al.* 2012; Hu and Smith 2019; Smith 2020). Each intersection acts as an autonomous scheduling agent, collaborating with neighbouring intersections to predict future traffic demand and minimise expected vehicle wait times at their respective signals. This distributed approach exhibits good potential for scalability due to its localised decision-making, but may exhibit reduced flexibility in achieving specific system-wide goals compared to centralised methods.

Instead, there are far fewer approaches to problems related to traffic signal optimisation that use ASP. Eiter *et al.* (2020) introduces an approach to optimise the coupling of traffic movements at junctions, according to expected traffic demand in the area, and to simulate using a mesoscopic-level representation. The experiments considered a realistic area with two junctions and are compared to the microscopic traffic simulator SUMO (López *et al.* 2018). In the wider area of traffic control, Cardellini *et al.* (2024) deals with the problem of dynamic traffic distributions in urban areas. As a part of a framework defined for solving such problem, they employed ASP for the computation of the best possible routes for all the vehicles in the network, starting from a set of candidate routes for all vehicles within the framework. On other directions in traffic research, ASP has been employed by Beck *et al.* (2012a,b) to manage the inconsistency in traffic regulations in Smart Cities, and by Vaseqi and Delgrande (2013) as a component in a situation awareness system for maritime traffic control.

## 7 Conclusion

In this paper, we presented a novel approach to the traffic signal optimisation problem leveraging CASP. By encoding the problem within a bounded time horizon, our method addresses a key limitation of existing PDDL + solutions, which do not properly support the specification of optimisation criteria. Our empirical evaluation, conducted on real-world historical traffic data for a range of traffic conditions, highlighted the capabilities of the proposed approach and its benefits over the PDDL + state of the art, as well as the potential of combining the approaches. Future work will focus on further improving the solving phase by tackling the limitations of one-shot search by adapting multi-shot techniques, and by defining and implementing domain heuristics, possibly extending the approach to larger urban regions. We are also interested in exploring the use of ASP-based approaches for identifying suitable traffic signal cycle configurations to be used according to the expected traffic conditions to be dealt with.

## References

Antoniou, G., Batsakis, S., Davies, J., Duke, A., Mccluskey, T. L., Peytchev, E., Tachmazidis, I. and Vallati, M. 2019. Enabling the use of a planning agent for urban traffic management via enriched and integrated urban data. *Transportation Research Part C: Emerging Technologies* 98, 284–297.

Balduccini, M. and Lierler, Y. 2017. Constraint answer set solver EZCSP and why integration schemas matter. *Theory and Practice of Logic Programming* 17, 4, 462–515.

Banbara, M., Kaufmann, B., Ostrowski, M. and Schaub, T. 2017. Clingcon : The next generation. *Theory and Practice of Logic Programming* 17, 4, 408–461.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

Beck, H., Eiter, T. and Krennwallner, T. 2012b. Inconsistency management for traffic regulations: Formalization and complexity results. In *Proc. of JELIA 2012b*, Springer, Berlin, Heidelberg, 80–93.

Beck, H., Eiter, T. and Krennwallner, T. 2012a. Inconsistency management for traffic regulations. In *Proc. of Semantic Cities, Papers from the 2012 AAAI Workshop 2012a*, Vol. WS-12-13 of AAAI Technical Report, AAAI Press, 2–8.

Bhatnagar, S., Guo, R., Mccabe, K., Mccluskey, T. L., Percassi, F. and Vallati, M. 2023. Automated planning for generating and simulating traffic signal strategies. In *Proc. of IJCAI 2023*, 7119–7122. ijcai.org.

Bhatnagar, S., Guo, R., Mccabe, K., Mccluskey, T. L., Scala, E. and Vallati, M. 2022a. Leveraging artificial intelligence for simulating traffic signal strategies. In *Proc. of ITSC 2022a*, IEEE, 607–612.

Bhatnagar, S., Mund, S., Scala, E., Mccabe, K., Mccluskey, T. L. and Vallati, M. 2022b. On-the-fly knowledge acquisition for automated planning applications: Challenges and lessons learnt. In *Proceedings of the International Conference on Agents and Artificial Intelligence, ICAART 2022b*, SciTePress, 387–397.

Brewka, G., Eiter, T. and Truszczynski, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54, 12, 92–103.

Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F. and Schaub, T. 2020. ASP-Core-2 input language format. *Theory and Practice of Logic Programming* 20, 2, 294–309.

Cardellini, M., Dodaro, C., Maratea, M. and Vallati, M. 2024. Optimising dynamic traffic distribution for urban networks with answer set programming. *Theory and Practice of Logic Programming* 24, 4, 825–843.

Eiter, T., Falkner, A. A., Schneider, P. and Schüller, P. 2020. Asp-based signal plan adjustments for traffic flow optimization. In *ECAI 2020 - 24th European Conference on Artificial Intelligence 2020*, IOS Press, 3026–3033.

Ferrara, A., Sacone, S., Siri, S., Ferrara, A., Sacone, S. and Siri, S. 2018. Microscopic and mesoscopic traffic models. *Freeway Traffic Modelling and Control* 1, 113–143.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9, 3-4, 365–386.

Gulić, M., Olivares, R. and Borrajo, D. 2016. Using automated planning for traffic signals control. *PROMET-Traffic & Transportation* 28, 4, 383–391.

Haslum, P., Ivankovic, F., Ramírez, M., Gordon, D., Thiébaux, S., Shivashankar, V. and Nau, D. S. 2018. Extending classical planning with state constraints: Heuristics and search for optimal planning. *Journal of Artificial Intelligence Research* 62, 373–431.

Hu, H. and Smith, S. F. 2019. Using Bi-Directional information exchange to improve decentralized schedule-driven traffic control. In *Proc. of ICAPS* 2019, AAAI Press, 200–208.

Ivankovic, F., Vallati, M., Chrpa, L. and Roveri, M. 2022. Urban traffic control via planning with global state constraints (extended abstract). In *Proc. of SoCS 2022*, AAAI Press, 291–293.

Janhunen, T., Kaminski, R., Ostrowski, M., Schellhorn, S., Wanko, P. and Schaub, T. 2017. Clingo goes linear constraints over reals and integers. *Theory and Practice of Logic Programming* 17, 872–888.

Kouaiti, A. E., Percassi, F., Saetti, A., Mccluskey, T. L. and Vallati, M. 2024. PDDL+ models for deployable yet effective traffic signal optimisation. In *Proc. of ICAPS 2024*, AAAI Press, 168–177.

Liu, G., Janhunen, T. and Niemelä, I. 2012. Answer set programming via mixed integer programming. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012*, AAAI Press, 32–42.

López, P.Á., Behrisch, M., Bieker-WALZ, L., Erdmann, J., Flötteröd, Y., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P. and Wiebner, E. 2018. Microscopic traffic simulation using SUMO. In *Proc. of ITSC 2018*, IEEE, 2575–2582.

Mccluskey, T. and Vallati, M. 2017. Embedding automated planning within urban traffic management operations. In *Proc. of ICAPS 2017*, AAAI Press, 391–399.

Mellarkod, V. S., Gelfond, M. and Zhang, Y. 2008. Integrating answer set programming and constraint logic programming. *Annals of Mathematics and Artificial Intelligence* 53, 251–287.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 25, 3-4, 241–273.

Papageorgiou, M. 2013. *Concise Encyclopedia of Traffic and Transportation Systems*. Vol. 6. Elsevier.

Percassi, F., Bhatnagar, S., Guo, R., Mccabe, K., Mccluskey, T. L. and Vallati, M. 2023a. An efficient heuristic for AI-based urban traffic control. In *8th International Conference on Models and Technologies for Intelligent Transportation Systems, MT-ITS 2023a*, IEEE, 1–6.

Percassi, F., Scala, E. and Vallati, M. 2023b. A practical approach to discretised pddl+ problems by translation to numeric planning. *Journal of Artificial Intelligence Research* 76, 115–162.

Percassi, F. and Vallati, M. 2024. Leveraging ai planning in a what-if analysis framework for assessing traffic signal strategies. In *2024 IEEE 27th International Conference on Intelligent Transportation Systems (ITSC) 2024*, IEEE, 1330–1335.

Percassi, F. and Vallati, M. 2025. Automated planning for urban traffic control with LLM-generated configurations. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference, FLAIRS 2025*, Vol. 38, LibraryPress@UF. URL: https://journals.flvc.org/FLAIRS.

Pozanco, A., Fernández, S. and Borrajo, D. 2021. On-line modelling and planning for urban traffic control. *Expert Systems* 38. URL: https://onlinelibrary.wiley.com/doi/epdf/10.1111/exsy.12693.

Scala, E., Haslum, P., Thiébaux, S. and Ramirez, M. 2020. Subgoaling techniques for satisficing and optimal numeric planning. *Journal of Artificial Intelligence Research* 68, 691–752.

Smith, S. F. 2020. Smart infrastructure for future urban mobility. *AI Magazine* 41, 1, 5–18.

Taale, H., Fransen, W. and Dibbits, J. 1998. The second assessment of the SCOOT system in Nijmegen. In *IEEE Road Transport Information and Control 1998*, IET, 21–23.

Vallati, M. and Chrpa, L. 2023. In defence of good old-fashioned artificial intelligence approaches in intelligent transportation systems. In *26th IEEE International Conference on Intelligent Transportation Systems, ITSC 2023*, 4913–4918.

Vallati, M., Magazzeni, D., De Schutter, B., Chrpa, L. and Mccluskey, T. 2016. Efficient macroscopic urban traffic models for reducing congestion: A PDDL+ planning approach. In *Proc. of AAAI 2016*, AAAI Press, 3188–3194.

Vaseqi, Z. and Delgrande, J. P. 2013. An application of answer set programming for situational analysis in a maritime traffic domain. In *Proc. of AI 2013*, Vol. 7884 of Lecture Notes in Computer Science, Springer, 315–322.

Xie, X.-F., Smith, S. and Barlow, G. 2012. Schedule-driven coordination for real-time traffic network control. In *Proc. of ICAPS 2012*, AAAI Press, 323–331.