A Multi-Agent Approach for Intelligent Traffic-Light Control

Visit Hirankitti, Jaturapith Krohkaew, and Chris Hogger, *Member, IAENG*

***Abstract*—In this paper we propose a multi-agent approach for traffic-light control. According to this approach, our system consists of agents and their world. In this context, the world consists of cars, road networks, traffic lights, etc. Each of these agents controls all traffic lights at one road junction by an observe-think-act cycle. That is, each agent repeatedly observes the current traffic condition surrounding its junction, and then uses this information to reason with condition-action rules to determine in what traffic condition how the agent can efficiently control the traffic flows at its junction, or collaborate with neighboring agents so that they can efficiently control the traffic flows, at their junctions, in such a way that would affect the traffic flows at its junction. This research demonstrates that a rather complicated problem of traffic-light control on a large road network can be solved elegantly by our rule-based multi-agent approach.**

***Index Terms*—Intelligent Transportation System, Multi-Agent System.**

I. Introduction

Traffic congestion is a crucial problem in a large city. It is normally caused by an improper control of traffic lights which is not corresponding to the current traffic condition surrounding the road junction. In this paper we propose a multi-agent approach for traffic light control which efficiently manages the traffic according to the current traffic condition. It aims to reduce each car's delayed time at each junction. This is achieved by an agent's observe-think-act cycle. That is, the agent continuously observes the current traffic conditions by collecting traffic data, and the data will then be used for reasoning with the traffic-light-control rules by the agent's inference engine to determine how a signal will be changed on each traffic light at each junction, so that the traffic can be managed efficiently.

The problem of intelligent traffic control has been studied in the area of intelligent transportation system for many years. We will refer to only a few that are related to our work. The first one is the method of Vehicle Actuated Signal Control [1]. This method controlled traffic lights by considering the number of cars waiting in the queue to be serviced by a traffic light. Sensors are placed at a short distance from the junction in order to detect cars and count the number of passing cars. When the current green light is going to turn red, but the sensor can detect that some cars have come in that range of distance, the duration of this green light is extended further. This scenario will repeat until no more cars have arrived in that range or the maximum duration for the green light has been reached.

Some approaches employed machine learning methods, such as reinforcement learning and genetics algorithm [2, 3], to learn traffic patterns of different time in a day and used them to control the traffic lights. This seems feasible when all the commuters behave normally. However, in real life it is hardly to be so. Other approaches used Fuzzy controllers [4, 5] to adjust only the duration of green light of each traffic light to match the current traffic condition, but not to change the signal patterns. Thus, this is not capable of controlling the traffic patterns.

Some works are based on a multi-agent approach. For example, [6] adopted case-based reasoning to control traffic lights. The agent observed traffic condition at a junction and used this information to match with candidate cases from its case-base, consequently it applied the solution of the selected case to control the traffic lights. Obviously this approach is quite similar to a rule-based approach we investigate in this paper. An agent proposed in [7] used some properties of the current states of all traffic patterns as the criterion to determine what will be the next traffic-light pattern.

For the rest of the paper, it is organized as follows. In the next section we first state our problem description. In section 3 the overall multi-agent system architecture is introduced. Section 4 describes our Logo-based traffic simulator and what traffic data the agents have to observe. In sections 5 and 6 the details of traffic control rules and the agent's inference engine are given respectively. An experiment with the system is reported in section 7. Finally we conclude our paper in section 8.

II. Problem Description

Initially we shall state our problem and its assumptions. That is, we assume all the roads under consideration are 2-way roads, each side of which has 3 lanes as depicted in Fig. 1; when each road meets others this creates a 'junction', which could be a 3-road junction, 4-road junction, etc. A traffic light is a device emitting light in green, red, and amber, to express different meanings, i.e. green means 'go ahead", red 'stop', and amber 'caution'. For convenience, in this paper we shall consider only the green and red signals.

We can calculate the number of traffic lights required for a junction from

Manuscript received April 28, 2007.

Visit Hirankitti and Jaturapith Krohkaew are with the Intelligent Communication and Transportation Research Laboratory, Department of Computer Engineering, Faculty of Engineering, King Mongkut's Institute of Technology Ladkrabang, Chalongkrung Road, Ladkrabang, Bangkok, Thailand 10520 email: v_hirankitti@yahoo.com and kjatura@yahoo.com

Chris Hogger is with the Department of Computing, 180 Queen's Gate, South Kensington campus-Imperial College London, London, United Kingdom SW7 2AZ email: cjh@doc.ic.ac.uk

$$L = \sum_{i=1}^{n} \left[(n-1) - D_i \right] \; ; n \geq 3$$

where n is the number of roads met at a junction,
Di is the number of free-flow able to pass through the junction from each road i

For example, the number of traffic lights for a 4-road junction in Fig.2 is (3–1) + (3–1) + (3–1) + (3–1) = 8; they control the 8 traffic flows of the junction, and they are called **a, b, c, d, e, f, g,** and **h** respectively. Apparently, there are 12 possible patterns of the traffic lights for controlling the traffic flows, i.e. **{a,b},{a,c},{a,f},{b,d},{b,g},{c,d},{c,h},{d,e}, {e,f},{e,g},{f,h},** and **{g,h}**. For simplicity, we associate a number to each pattern as seen in Fig. 3. For a traffic light *l*, one can identify, from these 12 patterns, as many as 3 pairs for *l*, e.g. there are **{a,b},{a,c}**, and **{a,f}** for **a**, and we call either **b** or **c** or **f** as a buddy of **a**.

Figure 1. A 2-way road with 6 lanes.

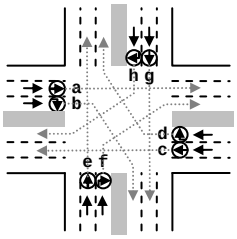


Figure 2. All traffic lights for a 4-road junction.

Precisely, the traffic-light control problem is how long to set green or red to each pattern and what sequence to arrange among the patterns, so that the average delayed time of each car at each traffic-light of the junction can be kept minimum.

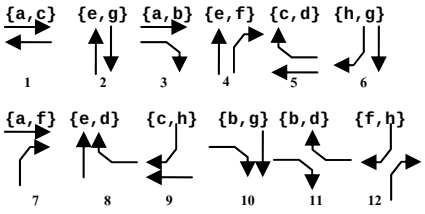


Figure 3. All the 12 possible traffic light patterns.

III. System Architecture

Our system consists of agents and their world. The world in this context is made of real cars on the roads, the road network, the traffic, the traffic lights, etc. Each agent is responsible for controlling all the traffic lights at one junction. It monitors and controls all the traffic lights at each junction by observing only the part of its world, a small part of the whole traffic world, in order to perceive the traffic information surrounding one junction, then reasoning with this information, and applying the traffic-control rules to control all the traffic lights; this activity will be repeated forever; see Fig. 4. The consequence of the rule application may result in a change in the current traffic-light pattern, or the need for the agent to collaborate with other agents so that they would control their traffic lights more efficiently.

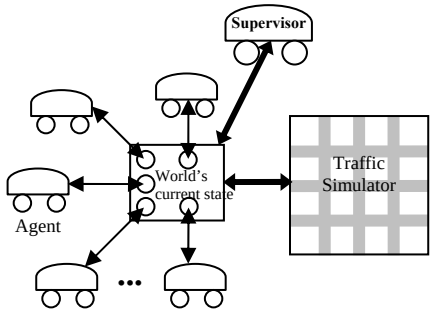


Figure 4. The multi-agent architecture for traffic-light control.

To work with a road network containing a large number of junctions, a community of such agents is required; these agents will perform their individual works while collaborating with others in order to solve complicated traffic problems. They are supervised by the supervisor agent who possesses traffic control rules for solving traffic problems at a larger scale. In Fig.4 although the supervisor agent cannot observe the world directly, it can reason with a part of the whole world's current state which is assembled from each small part observed by each agent.

A. The Traffic Simulator

It is not easy to do the real trial of traffic-light control, since it affects everyday commuters. To make life easier, the real agents' world is replaced by a virtual world, that is, a traffic simulator. In fact, this traffic simulator gives us many benefits; it is even better than the real traffic. Firstly, with the simulator we can simulate various traffic scenarios the way we wish to test our agent approach. Secondly, the simulator can easily provide necessary traffic data in real-time for analysis by the agents. This traffic data is observed continuously by the agents. Furthermore, the agents can control the traffic lights easily by just feeding instructions in terms of control parameters back to the simulator in order to set the duration of green or red lights for some traffic lights at some junctions.

B. The Agent

Each agent we adopted for the traffic-light control is a logical agent being composed of (Traffic) Observer, Knowledge Base, Inference Engine, and Communication Module.

The (Traffic) Observer continuously observes the traffic condition at a road junction under its responsibility while passing the observed traffic data, in terms of facts, to the working memory of its Inference Engine. Given the observed facts in its working memory, Inference Engine retrieves condition-action rules from Knowledge Base and tries to match the observed facts with the rules' condition parts. If any of these succeed, the actions parts of the eligible rules will be fired; this produces two possible consequences: firstly, Inference Engine takes the actions as being stated in the fired

rules, and this results in some control parameters being passed to the simulator to control some traffic lights, or the agent communicates some facts using the Communication Module to another agent for a co-operative control; alternatively, some facts are inferred as the result of the firing of the rules and these facts will be asserted to the working memory of Inference Engine and ready to be matched with the rules in the next round of the rule application cycle. After that the same process as explained from the beginning of this paragraph will repeat again. In short, the agent performs an endless observe-think-act cycle to continuously observe the traffic data and control the traffic lights.

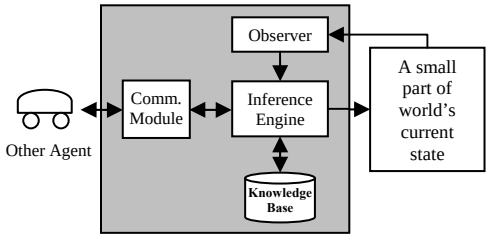


Figure 5. The single agent architecture for traffic-light control.

The agent's Knowledge Base does not only contain rules for traffic-light control, but also facts necessary for the control purpose, for example, those describing the agent's local topology of the road network, the names of the traffic flows and traffic lights under its responsibility, and the names of neighboring agents who collaborate with this agent.

C. The Community of Agents

The big picture of the traffic-light control proposed here is a multi-agent system where individual agents take charge in observing traffic condition only at the junctions assigned to them while using this information to control only the traffic lights of those junctions; to do so, in some situation these agents may require collaboration with other nearby agents to exchange extra traffic information, which cannot be observed by themselves, in order to enhance the performance of their traffic control.

IV. Traffic Simulation And Observation

The traffic simulator employed in this research is developed in NetLogo [9]. The traffic world is hence simulated by a NetLogo world, i.e. the cars are simulated by turtles, the roads by patches, and sensors by observers, and it fits well with the assumption we made earlier in section 2. With the ability of massive concurrent computation the NetLogo can provide, this traffic simulator suits our research purpose very well.

A. Traffic Data

The traffic simulator observes the traffic data continuously and instantly reports it to the world's current state, a small part of which is currently observed by every agent, see Fig. 5. This data consists of:

- The current active pattern (being set to green) of the traffic lights.
- The current value (in seconds) of the count-down counter of the current active pattern.
- Queue length of each traffic flow to be serviced by a traffic light. It is calculated from the discrepancy between the number of the cars getting into and the number of the cars leaving the queue of each lane. These two figures are measured by the two sensors placing at each end of a queue, see Fig. 6. To ensure the accuracy of these figures, we assume that when any car getting into a queue it cannot change the lane until it has left the queue.
- Downstream space availability, measured in the number of cars able to fill in this space. This figure measures the space availability at the downstream of the traffic flow. (The downstream indicates the direction on another side of the junction where the cars leaving the traffic light will move to.)
- Incoming rate indicating the number of the cars getting into the queue per second.
- Service rate indicating the number of the cars leaving the queue per second.

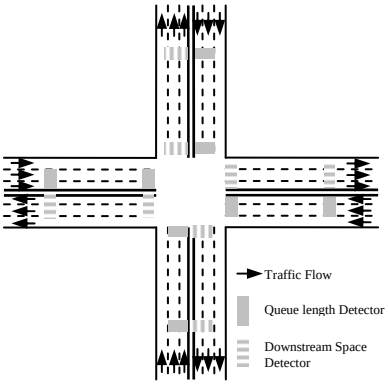


Figure 6. All the traffic sensors.

B. Traffic Evaluation

We can easily evaluate the performance of the overall traffic under control, since NetLogo allows every agent, i.e. turtle, patch, and observer, in the NetLogo world to perform introspection, in that, such an agent is aware of its own state and can perform book-keeping of this information within itself. With this ability we can program a car to record its delayed time caused by being in a queue, and we can later use this information collected from every car to find out the average delayed time of all cars during the traffic simulation in real-time whenever required.

V. Knowledge Base

Our logical agent is developed in Prolog. Now we describe its knowledge base which contains facts and rules for the traffic-light control purpose.

A. Facts for Traffic-Light Control

For each agent to be able to control all the traffic lights at each junction and also collaborate with other neighboring agents, it needs to have knowledge about road topology, paths (road segments), traffic flows, traffic lights, neighboring

agents, and their relationships.

For a road network shown in Fig. 7, it contains four junctions: A, B, C, and D, and the traffic at A is managed by agentA, the traffic at B is managed by agentB, and so on. Two junctions are connected by a directed path, which contains two traffic flows: f1 and f2. Each traffic flow is controlled by a traffic light, and it also has a downstream flow which is the flow that the cars will go to when the light turns green.

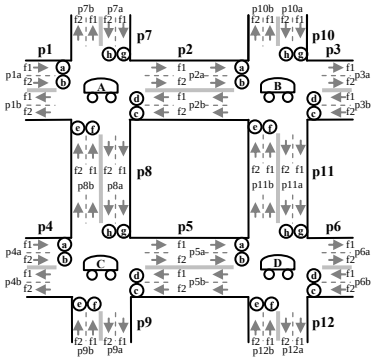


Figure 7. illustrates agents, junctions, paths, traffic flows, and traffic lights.

For example, the agentA's Knowledge Base contains the following facts:

manages(myself,junctionA) manages(agentB,junctionB)
manages(agentC,junctionC)
…

connects(p2a,junctionA,junctionB) connects(p2b,junctionB,junctionA)
connects(p8a,junctionA,junctionC) connects(p8b,junctionC,junctionA)
…

flows_of_path(p2a,p2a:f1,p2a:f2) flows_of_path(p2b,p2b:f1,p2b:f2)
flows_of_path(p8a,p8a:f1,p8a:f2) flows_of_path(p8b,p8b:f1,p8b:f2)
…

controls(a, p1a:f1) controls(b, p1a:f2) controls(c, p2b:f2) controls(d, p2b:f1)
controls(e, p8b:f2) controls(f, p8b:f1) controls(g, p7a:f1) controls(h, p7a:f2)

downstream_of(p2a:f1,p1a:f1) downstream_of(p8a:f2,p1a:f2)
downstream_of(p1b:f2,p2b:f2) downstream_of(p7b:f1,p2b:f1)
downstream_of(p7b:f2,p8b:f2) downstream_of(p2a:f2,p8b:f1)
downstream_of(p8a:f1,p7a:f1) downstream_of(p1b:f1,p7a:f2)

B. Rules for Traffic-Light Control

The rules possessed by each individual agent for traffic-light control are in the form

Condition → Action,

where **Condition** is conjunctive normal form of atomic sentences and **Action** is a conjunction of atomic sentences; the whole implication is universally quantified.

Condition states the condition that what agent can observe from the world's current state can be satisfied and that requires a certain **Action** response from the agent. In [8] we have identified 13 rules for controlling the traffic-lights in different traffic conditions as follows.

Rule1: **if** current_state(P) and state_counter(N) and N ≤ 0
then pattern_duration(T) and change_state(P,T)

Rule2.1: **if** current_state(P) and flow_of_state(P,F1,F2) and queue_length(F1,Q) and Q = 0 and flow_is_on&bflow_is_off (F1,Fb)
then find_max_delayed_bflow(F2,F1,F2b) and turn_off_flow&on_bflow(F1,F2b) and controls(LF1,F1) and controls(LF2b,F2b) and turn_off_light(LF1) and turn_on_light (LF2b)

Rule2.2: **if** current_state(P) and flow_of_state(P,F1,F2) and queue_length(F2,Q) and Q = 0 and flow_is_on&bflow_is_off (F2,Fb)
then find_max_delayed_bflow(F1,F2,F1b) and turn_off_flow&on_bflow(F2,F1b) and controls(LF2,F2) and controls(LF1b,F1b) and turn_off_light (L2) and turn_on_light (LF1b)

Rule2.3: **if** current_state(P) and flow_of_state(P,F1,F2) and flow_is_on&bflow_is_off(F1,Fb) and downstream_of(Fd,F1) and downstream_maxsize(Fd,M) and downstream_size(Fd,D) and M - D ≤ 1
then find_max_delayed_bflow(F2,F1,F2b) and turn_off_flow&on_bflow(F1,F2b) and controls(LF1,F1) and controls(LF2b,F2b) and turn_off_light (LF1) and turn_on_light (LF2b)

Rule2.4: **if** current_state(P) and flow_of_state(P,F1,F2) and flow_is_on&bflow_is_off(F2,Fb) and downstream_of(Fd,F2) and downstream_maxsize(Fd,M) and downstream_size(Fd,D) and M - D ≤ 1
then find_max_delayed_bflow(F1,F2,F1b) and turn_off_flow&on_bflow(F2,F1b) and and controls(LF2,F2) and controls(LF1b,F1b) and turn_off_light (L2) and turn_on_light (LF1b)

Rule2.5: **if** current_state(P) and flow_of_state(P,F1,F2) and flow_is_off&bflow_is_on(F1,_) and flow_is_off&bflow_is_on(F2,_)
then set_state_counter(0)

Rule2.6: **if** current_state(P) and flow_of_state(P,F1,_) and flow_is_off&bflow_is_on(F1,Fb) and downstream_of(Fd,F1) and downstream_maxsize(Fd,M) and downstream_size(Fd,D) and M - D > 10 and queue_length(F1,Q) and Q ≠ 0
then turn_on_flow&off_bflow(F1,Fb) and controls(LF1,F1) and controls(LFb,Fb) and turn_off_light(LFb) and turn_on_light(LF1)

Rule2.7: **if** current_state(P) and flow_of_state(P,_,F2) and flow_is_off&bflow_is_on(F2,Fb) and downstream_of(Fd,F2) and downstream_maxsize(Fd,M) and downstream_size(Fd,D) and M - D > 10 and queue_length(F2,Q) and Q ≠ 0
then turn_on_flow&off_bflow(F2,Fb) and controls(LF2,F2) and controls(LFb,Fb) and turn_off_light(LFb) and turn_on_light(LF2)

Rule3.1: **if** current_state(P) and state_counter(N) and 0 < N ≤ 5 and flow_of_state(P,F1,_) and queue_length(F1,Q) and boundary_queue_length(F1,Qb) and Q > 0.90 * Qb and queue_service_rate(F1,S) and incoming_rate(F1,I) and I > S
then to_extend_time_for_flow(F)

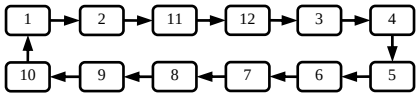
Rule3.2: **if** current_state(P) and state_counter(N) and 0 < N ≤ 5 and flow_of_state(P,_,F2) and queue_length(F2,Q) and boundary_queue_length(F2,Qb) and Q > 0.90 * Qb and queue_service_rate(F2,S) and incoming_rate(F2,I) and I > S
then to_extend_time_for_flow(F)

Rule3.3: **if** current_state(P) and flow_of_state(P,F1,F2) and state_counter(N) and max_greentime(Tmax) and N < Tmax and to_extend_time_for_flow(F1)
then state_greentime(Tg) and greentime_extension(Tx) and time_ext4counter(Tg,Tx,Tmax,Tadd) and Text is N + Tadd and set_state_counter(Text) and Tgx is Tg + Tadd and set_state_greentime(Tgx)

Rule3.4: **if** current_state(P) and flow_of_state(P,F1,F2) and state_counter(N) and max_greentime(Tmax) and N < Tmax and to_extend_time_for_flow(F2)
then state_greentime(Tg) and greentime_extension(Tx) and time_ext4counter(Tg,Tx,Tmax,Tadd) and Text is N + Tadd and set_state_counter(Text) and Tgx is Tg + Tadd and set_state_greentime(Tgx)

Rule4: **if** current_state(P) and state_greentime(Tg) and max_greentime(Tmax) and Tg > Tmax
then pattern_duration(T) and change_state(P,T)

Rule 1 states that if the current pattern of the traffic lights, that being set green, has its counter value reduced to 0, then the current pattern needs to change to the next pattern. The change cycle of the traffic pattern is:



Rule 2.1 or **2.2** states that if one of the two flows of the current pattern has no cars waiting in its queue, then its current green light has to be set to red; and while the green time still remains for this pattern, we will find all the buddies (except the current buddy that forms this pattern) for the light that remains green and set the longest waiting one to be green as well.

Rule 2.3 or **2.4** states that if the downstream space of one of the two flows of the current pattern is full (no car can move

into), then its current green light has to set to red; and while the green time still remains for this pattern, we will find all the buddies (except the current one that forms this pattern) for the light that remains green and set the longest waiting one to be green too.

Rule 2.5 states when both of its flows were set to red (due to Rule 2.1, 2.2, 2.3, or 2.4), it needs to change to the next pattern.

Rule 2.6 and **2.7** produce the reverse effects of 2.3 and 2.4 respectively, i.e. if one of the lights of the current pattern was turned red, but now there is a plenty of space at its downstream, then it has to be set back to green again.

Rule 3.1 or **3.2** states if the duration of green time of the current pattern is ending, but the rate of the cars getting into the queue of one of its flows is greater than the service rate, i.e. the rate of the cars leaving the queue, while the current queue length is greater than 90% of the boundary (maximum) queue length, then a small extra green time is given to this pattern.

Rule 3.3 and **3.4** ensure that the extra green time to be added to the green time of a pattern (as the results of 3.1 and 3.2 respectively) will not result in a total green time exceeding the maximum green time.

Rule 4 overrides all the rules, i.e. the duration for any pattern be given green lights in total has to be not more than the maximum green time. When it reaches that, it must be changed to the next pattern.

These are some rules relevant for intelligent control of the traffic lights by individual agents.

C. Rules for Agent Collaboration

In addition, rules for agent communication and collaboration are also necessary. In this traffic-light control problem, each agent can observe the traffic condition only surrounding itself and can control only the traffic lights at its junction. In many circumstances they need to collaborate with other neighboring agents to extend their knowledge (or observation) to be able to achieve a better traffic-light control.

An example is illustrated by Fig.8, agentA can detect that the downstream at flows p2a:f1 and p2a:f2 are full whilst agentB cannot know that. This will cause a problem for agentA who wants to release the cars into path p2a to avoid traffic congestion at its junction. To collaborate with agentB, it therefore reports this observation to agentB, so that agentB can act upon this quickly by releasing flow p2a:f1 and/or p2a:f2. Here we can express Rule 5 and 6 for this agent collaboration:

Rule5.1: **if** downstream_size(Fd,D) and downstream_maxsize(Fd,M) and D ≥ M and flows_of_path(P,Fd,_) and connects(P,J,NextJ) and manages(myself,J) and manages(Agent,NextJ) **then** tell(Agent,full_end_of_flow(P,Fd,null))

Rule5.2: **if** downstream_size(Fd,D) and downstream_maxsize(Fd,M) and D ≥ M and flows_of_path(P,_,Fd) and connects(P,J,NextJ) and manages(myself,J) and manages(Agent,NextJ) **then** tell(Agent,full_end_of_flow(P,null,Fd))

Rule5.3: **if** downstream_size(Fd,D) and downstream_maxsize(Fd,M) and D < M and flows_of_path(P,Fd,_) and connects(P,J,NextJ) and manages(myself,J) and manages(Agent,NextJ) **then** tell(Agent,not_full_end_of_flow(P,Fd,null))

Rule5.4: **if** downstream_size(Fd,D) and downstream_maxsize(Fd,M) and D < M and flows_of_path(P,_,Fd) and connects(P,J,NextJ) and manages(myself,J) and manages(Agent,NextJ) **then** tell(Agent,not_full_end_of_flow(P,null,Fd))

With **Rule 5.1** and **5.2** when the agent detects that a downstream at a flow is full, it alerts the neighboring agent. **Rule 5.3** and **5.4** capture the inverses of **5.1** and **5.2** respectively.

For every agent to act in response to the message sent by **Rule 5**, it requires the following rules:

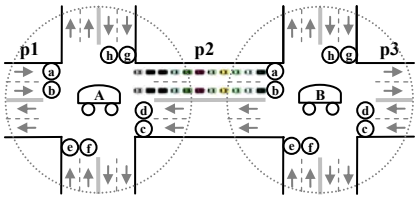


Figure 8. Efficient traffic-light control requires collaboration between agents.

Rule6.1: **if** current_state(P) and state_counter(N) and (full_end_of_flow(_,F1,null) or full_end_of_flow(_,null,F1)) and queue_length(F1,Q1) and boundary_queue_length(F1,Qb1) and Q1 ≥ Qb1 and downstream_of(Fd1,F1) and downstream_maxsize(Fd1,M1) and downstream_size(Fd1,D1) and M1 – D1 > 40 and flows_of_state(Next_P,F1,F2) and (not_full_end_of_flow(_,F2,null) or not_full_end_of_flow(_,null,F2)) **then** store_previous_state(P,N) and pattern_duration(T) and set_current_state(Next_P,T)

Rule6.2: **if** current_state(P) and state_counter(N) and (full_end_of_flow(_,F1,null) or full_end_of_flow(_,null,F1)) and queue_length(F1,Q1) and boundary_queue_length(F1,Qb1) and Q1 ≥ Qb1 and downstream_of(Fd1,F1) and downstream_maxsize(Fd1,M1) and downstream_size(Fd1,D1) and M1 – D1 > 40 and (full_end_of_flow(_,F2,null) or full_end_of_flow(_,null,F2)) and F1≠F2 and queue_length(F2,Q2) and boundary_queue_length(F2,Qb2) and Q2 ≥ Qb2 and downstream_of(Fd2,F2) and downstream_maxsize(Fd2,M2) and downstream_size(Fd2,D2) and M2 – D2 > 40 and flows_of_state(Next_P,F1,F2) **then** store_previous_state(P,N) and pattern_duration(T) and set_current_state(Next_P,T)

Rule 6.1 handles the case where one flow is full whilst **Rule 6.2** handles the case where two flows are full altogether. In either case, the agent will determine which traffic-light pattern to set in order to release the cars from the full flow(s) while ensuring that the downstream(s) can accommodate the flow(s). The new pattern is set as an interrupt to the current pattern which will be restored later once **Rule 1** is fired.

Another example is the circumstance supposing p2a:f1 and/or p2a:f2 is empty (see Fig. 7 and compare it with the previous circumstance). In this case, agentB has to report this to agentA, so that agentA can release the cars to utilize this empty space.

Rule7.1: **if** queue_length(F,0) and flows_of_path(P,F,_) and connects(P,J,NextJ) and manages(Agent,J) and manages(myself,NextJ) **then** tell(Agent,empty_flow(P,F,null))

Rule7.2: **if** queue_length(F,0) and flows_of_path(P,_,F) and connects(P,J,NextJ) and manages(Agent,J) and manages(myself,NextJ) **then** tell(Agent,empty_flow(P,null,F))

Rule7.3: **if** queue_length(F,Q) and Q ≠ 0 and flows_of_path(P,F,_) and connects(P,J,NextJ) and manages(Agent,J) and manages(myself,NextJ) **then** tell(Agent,not_empty_flow(P,F,null))

Rule7.4: **if** queue_length(F,Q) and Q ≠ 0 and flows_of_path(P,_,F) and connects(P,J,NextJ) and manages(Agent,J) and manages(myself,NextJ) **then** tell(Agent,not_empty_flow(P,null,F))

For every agent to act in response to the message sent by **Rule 7**, it requires the following rules:

Rule8.1: **if** current_state(P) and state_counter(N) and (empty_flow(_,Fd,null) or empty_flow(_,null,Fd)) and downstream_of(Fd,F1) and queue_length(F1,Q) and boundary_queue_length(F1,Qb) and Q ≥ 0.9 * Qb and

flows_of_state(Next_P,F1,F2) and
(not_empty_flow(_,F2,null) or not_empty_flow (_,null,F2))
then store_previous_state(P,N) and pattern_duration(T) and
set_current_state(Next_P,T)

Rule8.2: **if** current_state(P) and state_counter(N) and
(empty_flow(_,Fd1,null) or empty_flow(_,null,Fd1)) and
downstream_of(Fd1,F1) and queue_length(F1,Q1) and
boundary_queue_length(F1,Qb1) and Q1 ≥ 0.9 * Qb1 and
(empty_flow(_,Fd2,null) or empty_flow(_,null,Fd2)) and Fd1≠Fd2 and
downstream_of(Fd2,F2) and queue_length(F2,Q2) and
boundary_queue_length(F2,Qb2) and Q2 ≥ 0.9 * Qb2 and
flows_of_state(Next_P,F1,F2)
then store_previous_state(P,N) and pattern_duration(T) and
set_current_state(Next_P,T)

Rule 8.1 handles the case where one flow is empty whilst **Rule 8.2** handles the case where two flows are empty altogether. In either case, the agent will choose a traffic-light pattern whose flow(s) having full queue(s) to fill the empty downstream(s).

D. Rules for the Supervisor Agent

Every agent is required to report some of its traffic observation to the supervisor agent who can access all the facts in the whole world's current state (see Fig. 4.). Because of this the supervisor agent can identify and solve a traffic problem at the global level. It employs the same architecture as the agent at each junction, but possesses a larger knowledge base. One instance of the rules for a global traffic control in its Knowledge Base is

if all_current_full_flows(Flow_List) and possible_loops(Flow_List,Loops)
then break(Loops)

This rule detects a traffic deadlock by gathering all the current full flows, as those reported by the agents at the junctions, and trying to find all possible loops from them, if there exist some loops, the agent will try to break them right away by requesting some actions to be taken by some agents at the junctions. Further details of the supervisor agent's role will be explored elsewhere.

VI. Inference Engine

Each of the agents controls all the traffic lights at each junction by employing the observe-think-act cycle, which can be expressed abstractly as

Infinite-loop read facts from the working memory
match those facts with all rules
fire all rules that applicable

This observe-think-act cycle is implemented as a simple rule interpreter in Prolog. Thus, the agent's inference engine is a meta-program below.

```
prove(true).
prove(A or B) :- !, (prove(A) ; prove(B)).
prove(A and B) :- !, prove(A), prove(B).
prove(A) :- working_memory(A),!.
prove(A) :- built-in(A),!,A.
one_step_rule_interpreter(R_type) :-
        findall(R,rule(R_type,R),Rule_list),
        try_all_rules_and_fire_them(Rule_list).
try_all_rules_and_fire_them([]).
try_all_rules_and_fire_them([R|RL]) :- [Condition,'->',Action] = R,
                        (prove(Condition), prove(Action) ; true ),
                try_all_rules_and_fire_them(RL).
rule_interpreter :-
        one_step_rule_interpreter(_),
        rule_interpreter.
```

The consequence of a rule being fired is the action part of the rule will be executed; this results in the agent either sending this action as the control instructions to the simulator for it to change states of some traffic lights, asserting some facts to the agent's working memory, or sending a message to a neighboring agent.

VII. Experimentation With 4 Junctions

We test our system with a 4-junction road network as shown in Fig. 7. and compare the performance of the traffic-light control by collaborative agents with that of the traffic-light control by individual agents without collaboration (**Rule 5**, **6**, **7** and **8** are not used). In Fig. 9 we can see that the average delayed time (in seconds) of each car at each traffic-light of the four junctions for the former case is better than the latter when the simulator feeds the cars in 1 peak, 2 peaks, 3 peaks, and 4 peaks of different directions.

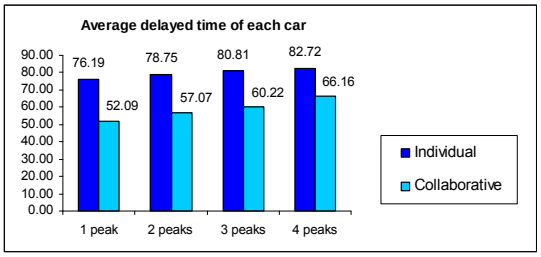


Figure 9. A performance comparison between multi-agent control and individual-agent control of traffic lights.

VIII. Conclusion

We have proposed a multi-agent approach, based on a rule-based system, for traffic light control which efficiently manages the traffic according to the current traffic condition. It employs a uniform inference of observe-think-act cycle as well as communication and collaboration among agents.

References

[1] H. Taale, "Comparing Methods to Optimise Vehicle Actuated Signal Control," in *IEE conference publication Road Transport Information and Control*, 2002, 486, pp. 114-119.

[2] S. Mikami and Y. Kakazu, "Genetic reinforcement learning for cooperative traffic signal control," in *Proc. IEEE World Congress Computational Intelligence*, 1994.

[3] L. Ying, M. Shoufeng, L. Wu and W. Huanchen, "Microscopic urban traffic simulation with multi-agent system," in *Proc. of the Joint Conference of the Fourth International Conference on Information, Communications and Signal Processing, and the Fourth Pacific Rim Conference on Multimedia*, 2003, pp. 1835 - 1839.

[4] G. Nakamiti and F. Gomide, "Fuzzy sets in distributed traffic control," in *Proc. 5th IEEE Int. Conf. Fuzzy Systems*, 1996, pp. 1617–1623.

[5] S. Chiu and S. Chand, "Self-organizing traffic control via fuzzy logic," in *Proc. 32nd IEEE Conf. Decision Control*, 1993, pp. 1987–1902.

[6] W. Xiao Xiong, Y. Shu Shen and Z. Xue Feng, "Architecture of Multi-agent System for Traffic Signal Control," in *Proc. of 8th IEEE Conf. Control, Automation, Robotics and Vision*, 2004, vol.3, pp. 2199–2204.

[7] F. D. Enrique, S. Eswaran and M. Dietrich, "Intelligent agents in decentralized traffic control," in *Proc. of IEEE Conf. Intelligent Transportation Systems*, 2001, pp. 705–709.

[8] V. Hirankitti and J. Krohkaew, "An Agent Approach for Intelligent Traffic-Light Control," in *Proc. of Asian Modelling Symposium AMS2007*, 2007, pp. 496-501.

[9] Uri Wilensky., *NetLogo User Manual version 3.0.2*, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 2005. Available at: http://ccl.northwestern.edu/netlogo/.