DOKUZ EYLUL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

A MULTI AGENT SYSTEM FOR URBAN TRAFFIC CONTROL

by
Ahmet ŞAHAN

April, 2008
İZMİR

A MULTI AGENT SYSTEM FOR URBAN TRAFFIC CONTROL

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Computer Engineering Program

by
Ahmet ŞAHAN

April, 2008
İZMİR

THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"A MULTI AGENT SYSTEM FOR URBAN TRAFFIC CONTROL"** completed by **AHMET ŞAHAN** under supervision of **PROF. DR. TATYANA YAKHNO** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Doctor of Philosophy.

.....

Supervisor

.....	.....
Thesis Committee Member	Thesis Committee Member
.....	.....
Examining Committee Member	Examining Committee Member

Prof. Dr. Cahit HELVACI

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my advisor, Prof. Dr. Tatyana Yakhno, for guidance, and for providing me with excellent facilities to pursue my work. I would like to express my appreciation to the thesis committee members; Prof. Dr. Alp Kut and Prof. Dr., Erol Uyar for their valuable comments and insightful remarks. I am also grateful to Aselsan Inc. that I work for; my chief, Miren Izaskun Gallastegi and my manager, B. Tarık Oranç for their continued support.

I appreciate to the countless authors of all the free software that I have used during my research work - their tremendous efforts have significantly aided my work. Special thanks go to my friends, Oğuz, Olcay Akay brothers and Kemal Memiş. As a friend, as a colleague, and as a mentor, they have contributed to this thesis continuously.

Finally, I am also thankful to my mother, my father and my sister for providing a constant source of encouragement and support, and being there for me at all times.

Ahmet ŞAHAN

A MULTI AGENT SYSTEM FOR URBAN TRAFFIC CONTROL

ABSTRACT

Traffic signal control is a system for synchronizing the timing of any number of traffic signals in a target road domain. These systems automate the process of adjusting signals to optimize traffic flow by reducing stops, overall vehicle delay and thus maximizing throughput. The study presented in this thesis proposes a new intelligent traffic light control that is quickly adaptive to changing environment. The new controller focuses on urban intersections and road lanes that are incoming to this intersection. The system inputs are traffic volumes on road lanes and the outputs are continuously changing light periods for the traffic light units in target intersections. The proposed system is based on a hierarchical multi agent model and a fuzzy controller is executed through this agent hierarchy. In addition to this local traffic light control, a reasoning engine is also integrated into the system to evaluate neighbor intersection situations. The outputs of the reasoning engine are also used for updating traffic light periods. All these work has been implemented on software basis and the results are given according to some sample intersection scenarios. The obtained results show that the proposed dynamic signalization system outperforms the fixed time-plan based controllers and generate better vehicle flows through intersections.

Keywords: Multi Agents, Fuzzy Logic, Adaptive Control Systems, Traffic Lights

KENTSEL TRAFİK KONTROLÜ İÇİN ÇOKLU BİR ARACI SİSTEMİ

ÖZ

Trafik sinyalizasyonu, hedef bir yol ağında yer alan trafik ışıklarının araç trafiğini düzenleme amaçlı olarak eş zamanlamasını gerçekleştiren bir sistemdir. Sinyalizasyon sistemleri, trafik ışık sürelerinin otomatik olarak değişmesini sağlayarak trafik akışının araçların durma ve bekleme sürelerini, kavşaktan geçen araç sayısını azami olarak arttırmayı hedefler. Bu tezde sunulan çalışma ile değişen trafik koşullarına hızlı uyum sağlayan akıllı bir trafik kontrolü önerilmektedir. Yeni trafik kontrol birimi, kentsel kavşaklar ve bu kavşaklara giriş yapan yol şeritlerine odaklanmaktadır. Önerilen sistemin girdi değerleri, izlenen kavşakta yer alan yol şeritlerindeki araç yoğunluklarıdır, çıktı değerleri de sürekli değişim içinde olan trafik ışıkları gösterim süre değerleridir. Önerilen trafik kontrol sistemi, bir hiyerarşi içerisinde tanımlı ve farklı rollere sahip çoklu aracı modeline dayanmaktadır. Bu yapı boyunca işletilen bulanık mantık kontrol ünitesi ile trafik süreleri belirlenmektedir. Yerel kontrolü sağlayan bulanık mantık ünitesinin yanında diğer kavşakların durumunu da değerlendiren bölgesel etkileşim birimi de sisteme monte edilmiştir. Bu birimin ürettiği sonuçlar da yeni trafik sinyalizasyon sürelerinin hesaplamasına katılmıştır. Tüm bu tasarım çalışmaları Java yazılım platformunda uygulama ortamına aktarılmış ve sonuçları çeşitli seçilmiş kavşak senaryolar ışığında ifade edilmiştir. Elde edilen neticeler, önerilen dinamik sinyalizasyon sisteminin zaman eksenli sinyalizasyon sistemlerine göre kavşaklardan daha başarılı araç akışı sağladığı görülmüştür.

Anahtar sözcükler: Çoklu Aracılar, Bulanık Mantık, Adaptif Denetim Sistemleri, Trafik Işıkları

CONTENTS

Page

CHAPTER ONE
INTRODUCTION

1.1 Area of Research

As the number of the vehicles on roads and highways continues to increase and distribute non-uniformly, the demand and expectations from the transportation systems grow in the same level too. However, roads and highways are unlikely to expand much due to cost and land supply so intelligent systems such as advanced traffic light controls become significantly important to operating our existing roadway systems at maximum performance.

Interest in Intelligent transportation systems comes from the problems caused by the traffic congestion worldwide and a synergy of new information technologies for simulation, real-time control and communications networks. Traffic congestion has been increasing world-wide as a result of increased motorization, urbanization, population growth and changes in population density. Congestion reduces efficiency of transportation infrastructure and increases travel time, air pollution and fuel consumption. A statistical survey shows that the fuel consumed by vehicles stopping and idling is approximately 40% of network wide vehicular fuel consumption.

In Today's world, most traffic signals are still of the pre-timed type with fixed splits and offsets that operate different timing plans based on time of day, congestion patterns or operator navigation. This type of signals is generally very good when it operates with progressive flow of traffic on an arterial street. However, pre-timed signals cannot respond to dynamically changing traffic flow. Although the traffic pattern changes, it still tries to execute the same active cycle length and split plan.

This type of operation often leads to the congestion if unusual traffic patterns occur or if there are major deviations in traffic flow. Furthermore, the timing plans in

use become obsolete unless they are checked regularly. Retiming also requires staffing that many organizations and administrations don't have.

The primary goal of an urban traffic control system must be regulating the vehicle flow patterns as fast and optimum as possible. In other words, it should have high efficiency to decrease vehicle delays by means of the management, in specific terms traffic lights are the main traffic control units.

1.2 Scope of Research

Intelligent systems such as adaptive traffic light controllers get specifically more attention to operate existing roadway systems at optimum performance. With the development of the computer technology and adapting it to the traffic engineering fields, fully automated models have been started to replace with manual operator assisted setting and optimization systems. These models generally combine the historical and current data with some intelligent techniques to estimate the optimal traffic light periods. These solutions aim to minimize vehicle delays and maximize total vehicle throughput while passing through an intersection or a group of them.

Since the beginning of eighties, some adaptive traffic light controller systems have already been proposed. These can be classified into two main groups according to their approach to the problem (Van Katwijk, Van Koningsbruggen, De Schutter, & Hellendoorn, 2005) and (Wiering, Van Veenen, Vreeken, & Koopman, 2004) expresses the same classification in terms of microscopic and macroscopic models.

- Vehicle-Oriented or Microscopic models focus on the control of the behavior of the individual vehicles.
- Measure-Oriented or Macroscopic (Traffic Light-based) models focus on the control of density of traffic.

Vehicle-Oriented models estimate the waiting periods of each vehicle that resides in incoming route queues to the junction and after the evaluation of these predicted

waiting times, it tries to minimize it by adjusting light periods. Applicability of these systems is very low because each vehicle should be controlled by a different module and some destination information might be required to estimate waiting times of the vehicles dynamically. Second type of the base models estimate density of the traffic and don't concentrate on waiting times of the vehicles individually. Thus it tries to eliminate the local congestion by predicting new light periods.

After the examination of the many proposed models and solutions to the traffic light control problem domain (Macroscopic models), the following results have been obtained:

- Some of them have been precisely defined but they couldn't be implemented or simulated in real-time.
- A group of them have no practical results
- Some of them have only been designed for single junctions or some strict topologies.
- Some of them have different approach to the problem (Travel time minimization centric, vehicle-oriented etc.)

After putting the general picture of the existing traffic control and signalization systems, it is seen that the traffic control strategies and policies are still hot topics and under research. Inspired from this point, the research presented with this thesis primarily addresses the problem of traffic flow management through intersections using traffic light control units.

The core objectives of the proposed model are the usage of intelligent techniques and providing generalization to be able to apply the system concepts to any target traffic network domain. The designed model focuses on traffic lights, junctions where the traffic lights are installed and intersections that consist of sub junction or junctions in a selected zone and tries to optimize the traffic light periods on local and global basis. The proposed solution looks have similarities with other Agent-based and fuzzy logic applied projects. However, the hierarchical multi-role agent

architecture, the independent and generic fuzzy logic implementation nature from the traffic network topologies differentiates our model than others.

1.3 Thesis Organization

The outline of the thesis is as follows.

In Chapter 2, initially the traffic systems overview is given. The taxonomy of the Traffic Control systems is defined in detail. And then, the previously designed and developed adaptive traffic control systems are described. A group of these systems is now already in market as commercial products.

Chapter 3 starts with the basic specification of the proposed controller system. The multi-role agent specifications, the agent hierarchy, distributed fuzzy controller flow and local and global congestion reasoning details are described here. This chapter is the core part of the thesis. Some extensions are referenced to Appendices.

In Chapter 4, the implementation details of the project are described. The implementation part consists of the software development and simulation details with tested scenarios of the proposed model. The specification of the software development environment is firstly given and then, the tools and utilities that are used at project development cycle are described in detail.

Chapter 5 defines the conclusion remarks according to the design, implementation and the test results of the thesis. Moreover, some future work possibilities are listed. After this chapter, the all reference list is given.

At the end of the thesis, an Appendices section is given. Some references related with the design issues and test scenarios with their details result graphs and tables are described in the sub sections of the Appendices part.

CHAPTER TWO
PRELIMINARY WORK

2.1 Traffic Lights Control

Modern traffic lights can be grouped into three parts: pre-timed, semi-actuated, and fully actuated. Pre-timed lights ignore the current state of traffic and follow a pre-defined timing strategy. Semi-actuated lights are normally used at intersections between a minor road and a major road: the major road is given the right of way unless a car is sensed waiting at the minor road. Fully actuated signals detect the presence of cars at all directions. The function of the controller in this mechanism is to measure traffic flow on all incoming ways to an intersection and to make new period assignments in accordance with traffic demand. The classification details are given below (Pearson, 2001).

2.1.1 Pre-timed or Fixed Time Signal Controllers

Under pre-timed operation, the master controller sets signal phases and the cycle length based on predetermined rates. These rates are determined from historical data. Pre-timed signal control is appropriate for areas where traffic demand is very predictable (Pearson, 2001).

2.1.2 Progression Schemes

A progression scheme is a simple way of coordinating signals along an arterial, which is common in many urban areas. The signals can be set manually to run in a constant, synchronous manner. There are 3 different types of progression schemes (Pearson, 2001):

Simultaneous: Under simultaneous progression, all signals along the route operate with the same cycle length and display green at the same time. All traffic moves

at once and a short time later all traffic stops at the nearest intersection to allow cross street traffic to move. This type of progression is typically used in downtown areas where intersections are close together, 300 to 500 ft, and uniformly spaced (Pearson, 2001).

Alternate: For alternate progression, there is a common cycle length. However, each successive signal or group of signals shows opposite indications. This type of progression is associated with uniform spacing of intersections. Ideal spacing is in the range of 1000 to 2.000 feet (Pearson, 2001).

Limited or simple: Limited/simple progression schemes employ a common cycle length, though the relationship of the indications between intersections vary because spacing between intersections is not uniform, and therefore offsets at each intersection differ. This type of progression scheme is typically used where traffic flow is uniform throughout the day (Pearson, 2001).

Flexible: Flexible progression schemes are identical to simple progression schemes, except that the common cycle length can be changed to reflect changing traffic patterns. Similar to limited or simple progression schemes, flexible progression schemes use different offsets between intersections (Pearson, 2001).

2.1.3 Actuated Controllers

An actuated controller operates based on traffic demands as registered by the actuation of vehicle and/or pedestrian detectors. There are several types of actuated controllers, but their main feature is the ability to adjust the signal's pre-timed phase lengths in response to traffic flow. If there are no vehicles detected on an approach, the controller can skip that phase. The green time for each approach is a function of the traffic flow, and can be varied between minimum and maximum lengths depending on flows. Cycle lengths and phases are adjusted at intervals set by vehicle actuation of pavement loops (Pearson, 2001).

Semi-Actuated Control: A semi-actuated controller provides for traffic actuation of all phases except the main phase. A continuous green is maintained on the major street except when a demand is registered by the minor street detector. The right of way always returns to the major street when no vehicles are present on the minor street or a timing limit has been reached. Semi-actuated operation is best suited for locations with low volume minor street traffic. It may also be used to permit pedestrian crossings at mid street (Pearson, 2001).

Full Actuated Control: Under full actuated control, the function of the controller is to measure traffic flow on all approaches to an intersection and make assignments of the right of way in accordance with traffic demand. Full actuated control requires placement of detectors on all approaches to the intersection. The controller's ability to respond to traffic flow provides for maximum efficiency at individual locations. This type of control is appropriate for intersections where the demand proportions from each leg of the intersection are less predictable (Pearson, 2001).

2.1.4 Traffic Responsive

In traffic responsive mode, signals receive inputs that reflect current traffic conditions, and use this data to choose an appropriate timing plan from a library of different plans. An individual signal or a network of several signals may be traffic responsive. Capabilities include: (Pearson, 2001)

Vehicle Actuated: uses data from presence detectors and modifies the phase splits based on vehicle actuation and gaps. This procedure addresses current traffic and does not require traffic projections (Pearson, 2001).

Future traffic prediction: The control system uses the volume data from system detectors and projects future conditions (Pearson, 2001).

Pattern Matching: The volume and occupancy data from system detectors are smoothed and weighted and compared with profiles in memory. This enables identification of the stored profile most closely matching the existing traffic conditions. When a pattern is identified, appropriate parameters are placed into operation (Pearson, 2001).

2.1.5 Adaptive Controllers

Adaptive Traffic Light controllers are currently the most advanced and complex control systems available. They are similar to fully actuated controllers, but instead of matching current conditions to existing timing plans, the system uses a real-time computer to create continuously an optimal timing plan. No library of timing plans is needed, which works well for areas with high rates of growth, where libraries of timing plans would need to be updated frequently. However, the success of these systems against traditional models is still on debate. For a discussion of all these studies, see (Roozemond & Rogier, 2000).

2.2 Traffic Systems Terminology

Traffic signal operation can be described in terms of phase splits, cycle lengths and offsets. A phase split defines the activation periods of red and green lights. The cycle length is the total time required for a complete sequence of signal phases and is typically between 60 to 120 seconds for a four-legged intersection. The offset between successive traffic signals is the time difference between the start of the green phase at an upstream intersection as related to the start of the green phase at an adjacent downstream intersection.

Some notions associated with “traffic” are outlined as follows too:

- Number of Vehicles: The number of vehicles in the system as a whole will depend on matters such as the capacity of roads, times of day, and similar factors.
- Density of Traffic: This will be mainly a function of the number of vehicles and capacity of the roads.

- Speed of Traffic: The speed of vehicles along a road has a direct bearing upon the rate of traffic flow along that road. The speed of individual vehicles along a section of road depends upon a number of factors: car types, weather, road condition, road obstacles, number of cars, the speed of other cars, and so on.

- Road Priority: Commonly, at any given intersection, one road will be more major than others.

- Road Capacity: This is a property of individual road segments, since only a finite number of vehicles can travel along a road at any given time. The maximum capacity of a segment is the number of vehicles, which can fit on the road in stationary “bumber-to-bumper” traffic.

2.3 Fuzzy Systems

(Brule, 1985) gives a general introduction to Fuzzy Logic. The fuzzy system is an alternative to traditional notions of set membership and logic that has its origins in ancient Greek philosophy, and applications at the leading edge of Artificial Intelligence. Yet, despite its long-standing origins, it is a relatively new field, and as such leaves much room for development.

The notion central to fuzzy systems is that truth values (in fuzzy logic) or membership values (in fuzzy sets) are indicated by a value on the range [0.0, 1.0], with 0.0 representing absolute Falseness and 1.0 representing absolute Truth. For example, let us take the statement (Brule, 1985):

"Jane is old."

If Jane's age was 75, we might assign the statement the truth value of 0.80. The statement could be translated into set terminology as follows:

"Jane is a member of the set of old people."

This statement would be rendered symbolically with fuzzy sets as:

$mOLD(Jane) = 0.80$

where m is the membership function, operating in this case on the fuzzy set of old people, which returns a value between 0.0 and 1.0.

At this juncture it is important to point out the distinction between fuzzy systems and probability. Both operate over the same numeric range, and at first glance both have similar values: 0.0 representing False (or non-membership), and 1.0 representing True (or membership). However, there is a distinction to be made between the two statements: The probabilistic approach yields the natural-language statement, "There is an 80% chance that Jane is old," while the fuzzy terminology corresponds to "Jane's degree of membership within the set of old people is 0.80." The semantic difference is significant: the first view supposes that Jane is or is not old (still caught in the Law of the Excluded Middle); it is just that we only have an 80% chance of knowing *which* set she is in. By contrast, fuzzy terminology supposes that Jane is "more or less" old, or some other term corresponding to the value of 0.80. Further distinctions arising out of the operations will be noted below (Brule, 1985).

The next step in establishing a complete system of fuzzy logic is to define the operations of EMPTY, EQUAL, COMPLEMENT (NOT), CONTAINMENT, UNION (OR), and INTERSECTION (AND). Before we can do this rigorously, we must state some formal definitions (Brule, 1985):

Definition 1: Let X be some set of objects, with elements noted as x. Thus,
X = {x}.

Definition 2: A fuzzy set A in X is characterized by a membership function
mA(x)
which maps each point in X onto the real interval [0.0, 1.0]. As mA(x) approaches 1.0, the "grade of membership" of x in A increases.

Definition 3: A is EMPTY iff for all x, mA(x) = 0.0.

Definition 4: A = B iff for all x: mA(x) = mB(x) [or, mA = mB].

Definition 5: mA' = 1 - mA.

Definition 6: A is CONTAINED in B iff mA <= mB.

Definition 7: C = A UNION B, where: mC(x) = MAX(mA(x),mB(x)).

Definition 8: C = A INTERSECTION B where: mC(x) =MIN(mA(x), mB(x)).

It is important to note the last two operations, UNION (OR) and INTERSECTION (AND), which represent the clearest point of departure from a probabilistic theory for sets to fuzzy sets. Operationally, the differences are as follows (Brule, 1985):

For independent events, the probabilistic operation for AND is multiplication, which (it can be argued) is counterintuitive for fuzzy systems. For example, let us presume that x = Bob, S is the fuzzy set of smart people, and T is the fuzzy set of tall people. Then, if mS(x) = 0.90 and uT(x) = 0.90, the probabilistic result would be (Brule, 1985):

*mS(x) * mT(x) = 0.81*

whereas the fuzzy result would be:

MIN(uS(x), uT(x)) = 0.90

The probabilistic calculation yields a result that is lower than either of the two initial values, which when viewed as "the chance of knowing" makes good sense. However, in fuzzy terms the two membership functions would read something like "Bob is very smart" and "Bob is very tall." If we presume for the sake of argument that "very" is a stronger term than "quite," and that we would correlate "quite" with the value 0.81, then the semantic difference becomes obvious. The probabilistic calculation would yield the statement (Brule, 1985):

If Bob is very smart, and Bob is very tall, then Bob is a quite tall, smart person.

The fuzzy calculation, however, would yield

If Bob is very smart, and Bob is very tall, then Bob is a very tall, smart person.

Another problem arises as we incorporate more factors into our equations (such as the fuzzy set of heavy people, etc.). We find that the ultimate result of a series of AND's approaches 0.0, even if all factors are initially high. Fuzzy theorists argue that this is wrong: that five factors of the value 0.90 (let us say, "very") AND'ed together, should yield a value of 0.90 (again, "very"), not 0.59 (perhaps equivalent to

"somewhat"). Similarly, the probabilistic version of A OR B is (A+B - A*B), which approaches 1.0 as additional factors are considered. Fuzzy theorists argue that a string of low membership grades should not produce a high membership grade; instead, the limit of the resulting membership grade should be the strongest membership value in the collection (Brule, 1985).

The skeptical observer will note that the assignment of values to linguistic meanings (such as 0.90 to "very") and vice versa, is a most imprecise operation. Fuzzy systems, it should be noted, lay no claim to establishing a formal procedure for assignments at this level; in fact, the only argument for a particular assignment is its intuitive strength. What fuzzy logic does propose is to establish a formal method of operating on these values, once the primitives have been established. Fuzzy systems, including fuzzy logic and fuzzy set theory, provide a rich and meaningful addition to standard logic. The mathematics generated by these theories is consistent, and fuzzy logic may be a generalization of classic logic. The applications which may be generated from or adapted to fuzzy logic are wide-ranging, and provide the opportunity for modeling of conditions which are inherently imprecisely defined, despite the concerns of classical logicians. Many systems may be modeled, simulated, and even replicated with the help of fuzzy systems, not the least of which is human reasoning itself (Brule, 1985).

2.4 Agent Systems

An agent is basically a software that has its own life-cycle (autonomy). It is capable of making independent decisions and taking actions to satisfy internal goals based upon its perceived environment that is shared. (Tanenbaum & van Steen, 2002, p. 173-180)

(Nikraz, Caire, & Bahri, 2006) gives an introduction about agent systems and agent based programming. The term agent is very broad and has different meanings to different people. However, on close observation of the literature, it is sufficient to say that two usages of the term agent can be identified: the *weak* notion of agency and the *strong* notion of agency. The weak notion of agency constitutes the bare

minimum that most researchers agree on, while the stronger notion of agency is more controversial and a subject of active research.

The weak notion of agency denotes a software-based computer system with the following properties (Nikraz, Caire, & Bahri, 2006, p.6):

- *Autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- *Social ability*: agents interact with other agents (and possibly humans) via some kind of agent communication language.
- *Reactivity*: agents perceive their environment and respond in a timely fashion to changes occurring therein.
- *Pro-activeness*: in addition to acting in response to their environment, agents are able to exhibit goal-directed behavior by taking the initiative.

The strong notion of agency is an extension of the weaker notion, and advocates additional humanistic, mental properties such as belief, desire, and intention (BDI). Consistent with the weak notion of agency, it may be said that the software agents are application programs that communicate with each other in an expressive agent communication language. Though at first this definition may seem a little simplistic, it allows one to clearly identify what constitutes a multi-agent system, i.e. agents are just pieces of autonomous code, able to communicate with each other using an agent communication language. The view of agents assumed in the proposed methodology is based on this definition. Specifically, the methodology assumes the following definition for a software agent (Nikraz, Caire, & Bahri, 2006, p.6):

> *agents reside on a platform that, consistent with the presented vision, provides the agents with a proper mechanism to communicate by names, regardless of the complexity and nature of the underlying environment (i.e. operating systems, networks, etc)*.

A multi-agent system (MAS) is a system composed of several software agents, collectively capable of reaching goals that are difficult to achieve by an individual

agent or monolithic system (A monolithic architecture is where processing, data and the user interface all reside on the same system). Although MAS is still strictly a research topic, many graphic computer games today are developed using MAS algorithms and MAS frameworks. MAS is applicable in transportation, logistics, graphics, GIS systems as well as in many other fields. It is widely being advocated to be used in networking and mobile technologies, to achieve automatic and dynamic load balancing, high scalability, and self healing networks.

Agent-based software engineering is a relatively new field and can be thought of as an evolution of object-oriented programming. Though agent technology provides a means to effectively solve problems in certain application areas, where other techniques may be deemed lacking or cumbersome, there is a current lack of mature agent-based software development methodologies. This deficiency has been pointed out as one of the main barriers to the large-scale uptake of agent technology. Thus, the continued development and refinement of methodologies for the development of multi-agent systems is imperative, and consequently, an area of agent technology deserving significant attention (Nikraz, Caire, & Bahri, 2006, p.2).

2.5 Neural Networks

(Stergiou & Siganos, 1987) gives a detailed chapter about Neural Networks. An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurones. This is true of ANNs as well.

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyse. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Other advantages include (Stergiou & Siganos, 1987, Section 1):

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage (Stergiou & Siganos, 1987, Section 1).

An artificial neuron is a device with many inputs and one output. The neuron has two modes of operation; the training mode and the using mode. In the training mode, the neuron can be trained to fire (or not), for particular input patterns. In the using mode, when a taught input pattern is detected at the input, its associated output becomes the current output. If the input pattern does not belong in the taught list of input patterns, the firing rule is used to determine whether to fire or not (Stergiou & Siganos, 1987, Section 3).

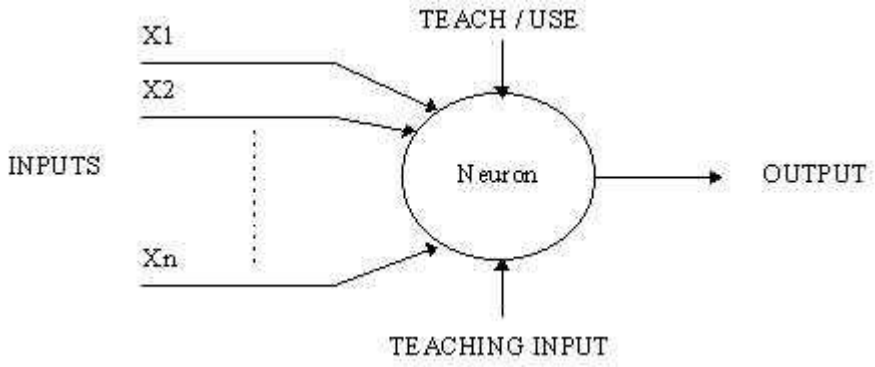


Figure 2.1 A simple neuron scheme.

The previous neuron doesn't do anything that conventional computers don't do already. A more sophisticated neuron (Figure 2) is the McCulloch and Pitts model (MCP). The difference from the previous model is that the inputs are 'weighted', the effect that each input has at decision making is dependent on the weight of the particular input. The weight of an input is a number which when multiplied with the input gives the weighted input. These weighted inputs are then added together and if they exceed a pre-set threshold value, the neuron fires. In any other case the neuron does not fire (Stergiou & Siganos, 1987, Section 3).

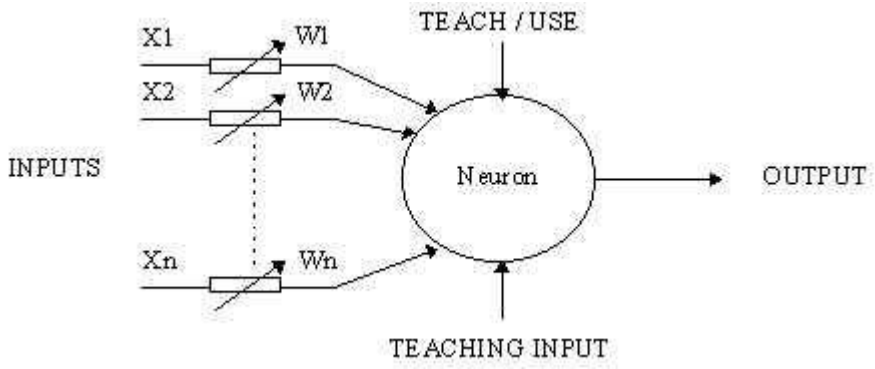


Figure 2.2 An MCP neuron.

In mathematical terms, the neuron fires if and only if;

$X_1W_1 + X_2W_2 + X_3W_3 + ... > T$

The addition of input weights and of the threshold makes this neuron a very flexible and powerful one. The MCP neuron has the ability to adapt to a particular

situation by changing its weights and/or threshold. Various algorithms exist that cause the neuron to 'adapt'; the most used ones are the Delta rule and the back error propagation. The former is used in feed-forward networks and the latter in feedback networks (Stergiou & Siganos, 1987, Section 3).

2.5.3 Feed-forward networks

Feed-forward ANNs (Figure 2.3) allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organisation is also referred to as bottom-up or top-down.

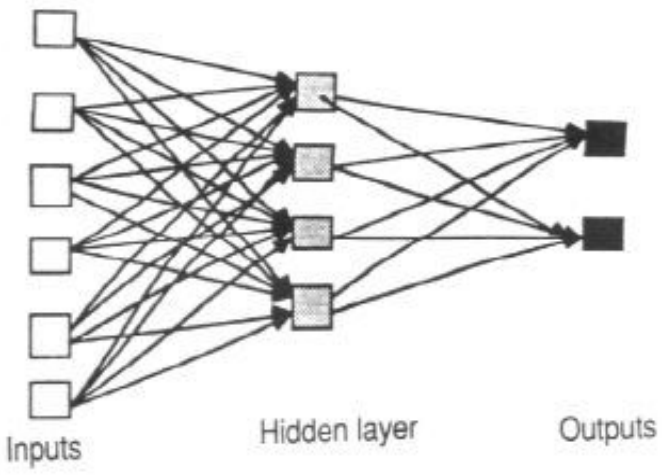


Figure 2.3 The feed forward network.

The commonest type of artificial neural network consists of three groups, or layers, of units: a layer of "**input**" units is connected to a layer of "**hidden**" units, which is connected to a layer of **"output**" units (See Figure 2.3). The activity of the input units represents the raw information that is fed into the network. The activity of each hidden unit is determined by the activities of the input units and the weights on the connections between the input and the hidden units. The behavior of the output units depends on the activity of the hidden units and the weights between the hidden and output units (Stergiou & Siganos, 1987, Section 4).

This simple type of network is interesting because the hidden units are free to construct their own representations of the input. The weights between the input and hidden units determine when each hidden unit is active, and so by modifying these weights, a hidden unit can choose what it represents. We also distinguish single-layer and multi-layer architectures. The single-layer organization, in which all units are connected to one another, constitutes the most general case and is of more potential computational power than hierarchically structured multi-layer organizations. In multi-layer networks, units are often numbered by layer, instead of following a global numbering. The behavior of an ANN (Artificial Neural Network) depends on both the weights and the input-output function (transfer function) that is specified for the units. This function typically falls into one of three categories (Stergiou & Siganos, 1987, Section 4):

- linear (or ramp)
- threshold
- sigmoid

For **linear units**, the output activity is proportional to the total weighted output. For **threshold units**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value. For **sigmoid units**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations. In order to make a neural network that performs some specific task, we must choose how the units are connected to one another (see figure 2.3), and we must set the weights on the connections appropriately. The connections determine whether it is possible for one unit to influence another. The weights specify the strength of the influence (Stergiou & Siganos, 1987, Section 4).

2.5.4 The Back-Propagation Network

In order to train a neural network to perform some task, we must adjust the weights of each unit in such a way that the error between the desired output and the actual output is reduced. This process requires that the neural network compute the error derivative of the weights (**EW**). In other words, it must calculate how the error changes as each weight is increased or decreased slightly. The back propagation algorithm is the most widely used method for determining the **EW**. The back-propagation algorithm is easiest to understand if all the units in the network are linear. The algorithm computes each **EW** by first computing the **EA**, the rate at which the error changes as the activity level of a unit is changed. For output units, the **EA** is simply the difference between the actual and the desired output. To compute the **EA** for a hidden unit in the layer just before the output layer, we first identify all the weights between that hidden unit and the output units to which it is connected. We then multiply those weights by the **EA**s of those output units and add the products. This sum equals the **EA** for the chosen hidden unit. After calculating all the **EA**s in the hidden layer just before the output layer, we can compute in like fashion the **EA**s for other layers, moving from layer to layer in a direction opposite to the way activities propagate through the network. This is what gives back propagation its name. Once the **EA** has been computed for a unit, it is straight forward to compute the **EW** for each incoming connection of the unit. The **EW** is the product of the EA and the activity through the incoming connection. Note that for non-linear units, the back-propagation algorithm includes an extra step. Before back-propagating, the **EA** must be converted into the **EI**, the rate at which the error changes as the total input received by a unit is changed (Stergiou & Siganos, 1987, Section 4).

2.5.5 Learning Process

Every neural network possesses knowledge which is contained in the values of the connections weights. Modifying the knowledge stored in the network as a function of experience implies a learning rule for changing the values of the weights. Information is stored in the weight matrix W of a neural network. Learning is the

determination of the weights. Following the way learning is performed, we can distinguish two major categories of neural networks (Stergiou & Siganos, 1987, Section 5):

Fixed networks in which the weights cannot be changed, i.e. dW/dt=0. In such networks, the weights are fixed a priori according to the problem to solve.

Adaptive networks which are able to change their weights, i.e. dW/dt not= 0.
All learning methods used for adaptive neural networks can be classified into two major categories:

Supervised learning which incorporates an external teacher, so that each output unit is told what its desired response to input signals ought to be. During the learning process global information may be required. Paradigms of supervised learning include error-correction learning, reinforcement learning and stochastic learning. An important issue concerning supervised learning is the problem of error convergence, i.e. the minimization of error between the desired and computed unit values. The aim is to determine a set of weights which minimizes the error. One well-known method, which is common to many learning paradigms, is the least mean square (LMS) convergence.

Unsupervised learning uses no external teacher and is based upon only local information. It is also referred to as self-organization, in the sense that it self-organizes data presented to the network and detects their emergent collective properties. A neural network learns off-line if the learning phase and the operation phase are distinct. A neural network learns on-line if it learns and operates at the same time. Usually, supervised learning is performed off-line, whereas unsupervised learning is performed on-line (Stergiou & Siganos, 1987, Section 5).

2.6 Other System Solutions

In the following paragraphs, the known traffic control systems are described shortly and the primitive properties of these systems from the positive and negative points of view are emphasized.

UTCS (Urban Traffic Control System): (Pearson, 2001) gives a short description about a centralized traffic control system that controls all intersections in a system with fixed or variable timing plans, developed by the Federal Highway Administration (FHWA). UTCS generates fixed timing schedules off-line based on average traffic conditions for a specified time of day.

SCATS (Sydney Coordinated Traffic Adaptive System): is a dynamic control system with a decentralized architecture (Pearson, 2001). It updates the intersection cycle length using the detectors. The basic traffic data used is the "degree of saturation", defined as the ratio of the effectively used green time to the total available green time. Basic limitation of this approach is that no major changes are permitted. Each light must have the same length cycle to remain in sequence with every other light. Therefore, SCATS can not make taking the advantage of major shifts in traffic patterns.

SCOOT (Split Cycle and Offset Optimization Technique) is a centralized traffic computerized control system developed at the Transportation Road Research Laboratory in the U.K. It is an enhancement over first generation UTCS systems and provides real-time adaptive control. SCOOT uses system detectors to measure traffic flow profiles in real time, and along with predetermined travel times and the degree of saturation (the ratio of flow-to-capacity), predicts queues at intersections. Adjustments of cycle length, phase splits and offsets are made in small steps to operate at a preset degree of saturation (usually 90%). Tests have shown that SCOOT is most effective when demand approaches, but is less than, capacity, where demand is unpredictable, and when distances between intersections are short. It knows how many cars are about to arrive at each light it controls, and thus figures

out which lights should get priority. SCOOT's objective is to minimize the sum of the average queues in an area. At fixed intervals, it modifies the splits, offsets and cycle times of each signal. SCOOT is slightly more sophisticated design than SCATS. However it has still the same limitation with SCATS.

Expert System: An expert system uses a set of given rules to decide upon the next action. (Findler & Stapp, 1992) proposes that a network of roads is connected by traffic light-based expert systems. These expert systems can communicate each other for synchronization and the system could optimize rules and learn new rules.

UTOPIA/SPOT: This PC based system uses a Rolling Horizon Optimization technique. It is economic to implement in a small town with as little as three or four intersections and is scalable into a large city system. The system uses an industrial grade single board PC card which can be installed in a wide range of existing traffic signal controllers. The card takes control of the unit and communicates with neighboring control units upstream and downstream of its location.

Each traffic light controller becomes a node in a local area network, with TCP/IP capability. Data is exchanged with neighboring intersections every 3 seconds and optimization is constantly performed over a rolling horizon 2 minute time frame. Public transport and emergency vehicle priority is supported, without sacrificing adaptive performance. Public transport priority operates on a selective priority basis – i.e. only vehicles running behind schedule receive priority at signalized intersections.

The UTOPIA / SPOT system has delivered increases of up to 35% in public transport speeds and 30% in private vehicle traffic speeds, when compared with fixed time signal systems. UTOPIA/SPOT is one of the popular adaptive control systems. These systems are powerful when the number of detectors installed at intersections is satisfactorily high and the parameter configurations are very reliable.

Fuzzy Logic Model: Fuzzy Logic offers a formal way of handling terms like "more", "less", "longer" etc., so rules like "if there is more traffic from south to north then the lights should stay green longer" can be reasoned with. (Taale & others, 1998) tried simulated a fuzzy logic traffic system. It measures the traffic the same way SCATS does, it is capable of determining how many cars pass through a given green light. It then applies this data to a set of 40 rules and adjusts the timings of the lights to correspond to large trends in traffic movement. This proposed design appears to be effective in simulations. A side effect of this research is the crucial point of consistency among all the lights. When the changes in timings are applied to only three lights and left other lights in standard cycle, the number of stops a car faced increased dramatically.

Reinforcement Learning Model: In simplest terms, reinforcement learning is used to learn agent control by letting the agent (can be traffic light or a vehicle) interact with its environment and learn from the obtained feedback (reward signals). Using a trial-and-error process, a reinforcement learning agent is able to learn a policy that optimizes the cumulative reward intake of the agent over time. For traffic light control it has first been studied by (Thorpe, 1997). He used a traffic light-based value function. A neural network is used for this value function that predicts the waiting time for all cars standing at the junction. (Thorpe, 1997) trained only one single traffic light controller and tested it by instantiating on a grid of 4x4 traffic lights. The system outperformed both fixed and rule-based controllers. The disadvantage of this model is that the difficulty to compute total trip waiting times for all road users standing at the traffic node, since this quantity has a large variance. To eliminate this problem, (Shen & Norrie, 2001) uses a bit different approach: A predictor is made for each car to estimate the waiting time of the car alone when the light is green or red. Then a voting scheme adds all predicted waiting times of cars for different traffic node decisions that will be used to minimize the overall waiting time.

(Cools, Gershenson & D'Hooge, 2005) points out an application self organizing logic to the control of traffic lights in a realistic simulation. The given concept is simple: counting the vehicles and making decisions to switch to green or red signal

periods. The solution is green wave centric and it tries to optimize the vehicle densities caused by standard green waves. The results are very impressive. However, because of its design principles only focused to straightforward intersection network, it can not be ported to isolated intersections or to the complex ones.

(Lee, J., Lee, K., Seong, Kim & Lee-Kwang, 1995) describes the use of fuzzy logic in controlling multiple junctions. Controllers at target point collect information at previous and next junctions, thus to provide green wave functionalities. The model outperforms a fixed light controller, and is best at both light and heavy traffic. The controller could easily handle changes in traffic flow. However it is strictly dependent on intersection topologies and requires some specific parameter adjustments.

(Liu, 2007) argues that Fuzzy Controller is much likely a traffic police who makes quick decisions by using interrelated qualitative knowledge. Also, it tells that Fuzzy Logic is more appropriate for single intersection management because of insufficient coverage of large and complex traffic networks.

(Eagan, Lamstein, Mappus, 2003) presents Intelligent Agent architecture for the traffic light control. Intelligent Traffic signalling agents and Road Segment agents try to perform their own tasks, and try to achieve local optimality. One or more coordinator agents can communicate with the group of these agents for global performance. All agents act upon their BDI (Belief, Desire, and Intention) properties. In Intelligent Agent terminology, Beliefs represent the informational state of the agent, Desires (or goals) represent the motivational state of the agent, Intentions represent the deliberative state of the agent. Although the model has a well-defined architecture, no practical results are presented.

The research studies show that the most traffic controllers are still operator assisted; pre-timed control systems or type of switching a group of plans based on a date/time information or vehicle densities. As it is stated before, those types of the systems can not manage dynamic changes in traffic flow. In order to handle all traffic

patterns, many adaptive traffic controller systems have already been proposed. However, there exist problems in these adaptive systems: Some of them are designed for only single intersection; some of them are strictly designed for a specific intersection arterial or limited network topologies and some of them have different approaches to the problem such as travel time minimization centric or some dependencies (i.e. reliable communication, lots of parameter adjustments etc.).

CHAPTER THREE
ARCHITECTURAL MODEL

The solution model defined here puts a new adaptive traffic control framework that tries to eliminate the deficiencies of the existing signalization control systems. In this thesis, we concentrate on Measure-Oriented models (traffic-light based macroscopic model) and to make global decisions some connectivity relations are defined between intersections. The proposed solution is mainly based on Multi-Agent paradigm.

3.1 Main Model

Our system is composed of several agent roles and they communicate with each other in the form of message communication or changes in their shared environment. There are five types of agents defined here: Road Agent, Light Agent, Junction Agent, Intersection Agent and Area Agent. There is a hierarchy between these agents shown as follows:

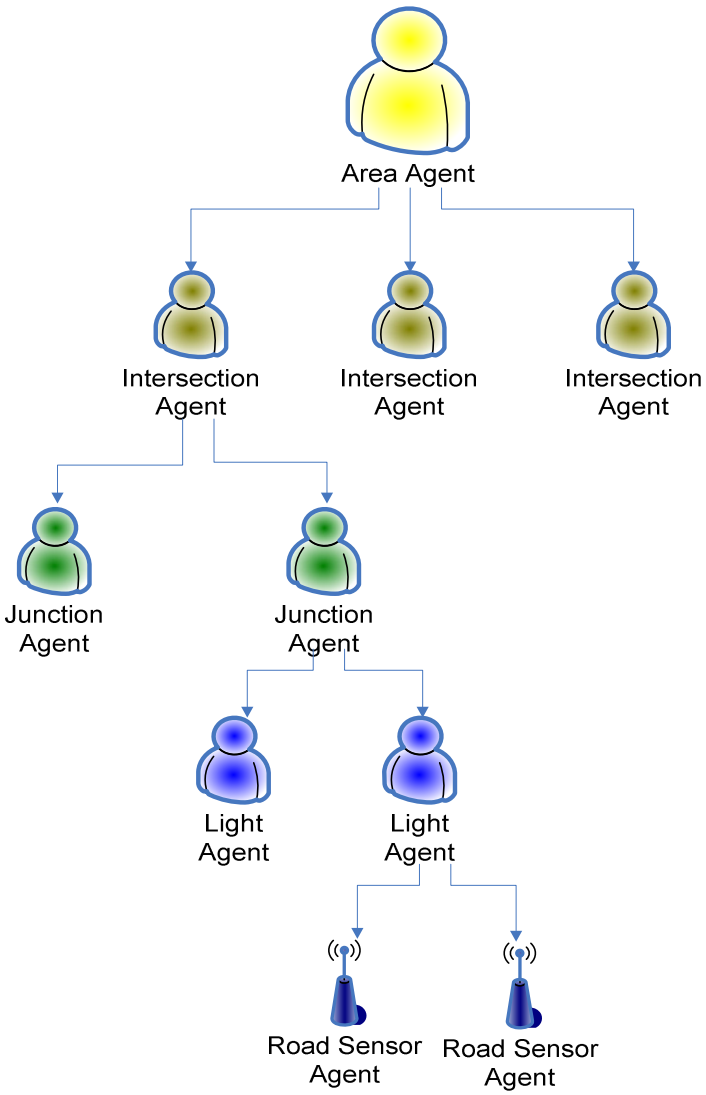


Figure 3.1 The general agent hierarchy of the system.

The objective of hierarchy construction is to decrease the complexity of control. Each upper agent will control its lower level agents by means of peer to peer communication methods. For the following traffic network, the agent schema is formed as follows:

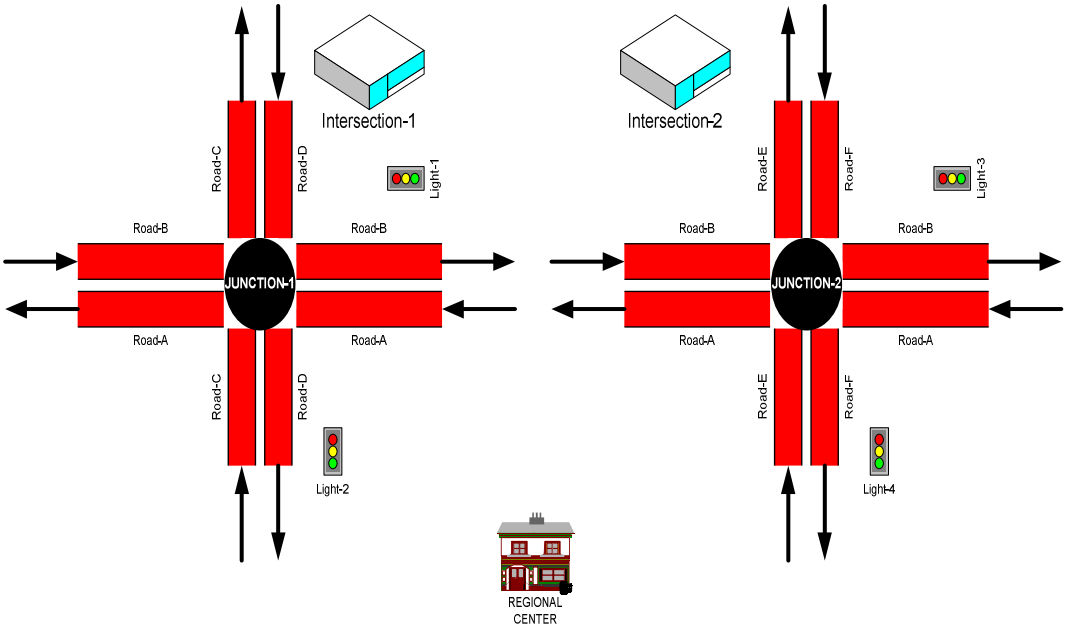


Figure 3.2 A sample traffic network.

Area agent represents the Regional Center in Figure 3.2 that is the global monitor of a target traffic network which consists of the tightly coupled intersections. Intersection agent represents traffic cross-point. In a traffic cross-point, it has to be at least one junction. If more tightly coupled junctions exist in an intersection, they are also controlled by the same intersection agent. This generally happens in intersection environments where there are so many road lanes incoming and outgoing. On the other hand, A Junction agent represents a transition controller: if one side is open, then other side should wait. Light agent represents controller of coupled road lanes on opposite directions and Road agent represents one-way lane traffic.

If this project wants to be executed on a selected network, the following guidelines should be followed for the configuration:

- Each incoming road direction (to the junction) is assigned to one Road Agent
- Each unique traffic light is assigned to one Light Agent
- Each junction is assigned to one Junction Agent
- Each intersection is assigned to an Intersection Agent
- Each tightly coupled intersection network is assigned to one Area agent.

Multi-agent model is the one side of the system. In addition to this, a fuzzy logic controller that is implemented from Road Agent to Junction Agent, neural network progress that is embedded in each Intersection agent and finally, local and global reasoning embedded in Intersection Agent are defined.

3.2 General Properties

- This system will try to optimize vehicle densities (volumes) on target road lanes that are incoming to intersection.
- Optimization will be done by decreasing or increasing of red period of traffic lights. Cycle time is assumed unchangeable. However, it may be open to small shifts.
- Junction to Junction in an intersection and Junction to Neighbor Junction between intersection lists, Light grouping lists are kept in a database.
- Fuzzy Controller is implemented across the agents: Road Agent, Light Agent and Junction Agent.
- To provide learning ability and enhance decision-making, a neural network module is implemented in Intersection Agent.
- Intersection Agent makes a rule-based reasoning based on local junction and neighbor intersection states. The result of reasoning is then used to generate commands to each local junction agent by launching a neural net processor.
- Our junction control design is explicitly focused on a road flow at each time slice. At each slice, the most overloaded one is selected for the optimization. Therefore, if one directional road flow is constantly at high level then it always gets the highest green light and lowest red light periods.
- The general system view is given in the following figures:

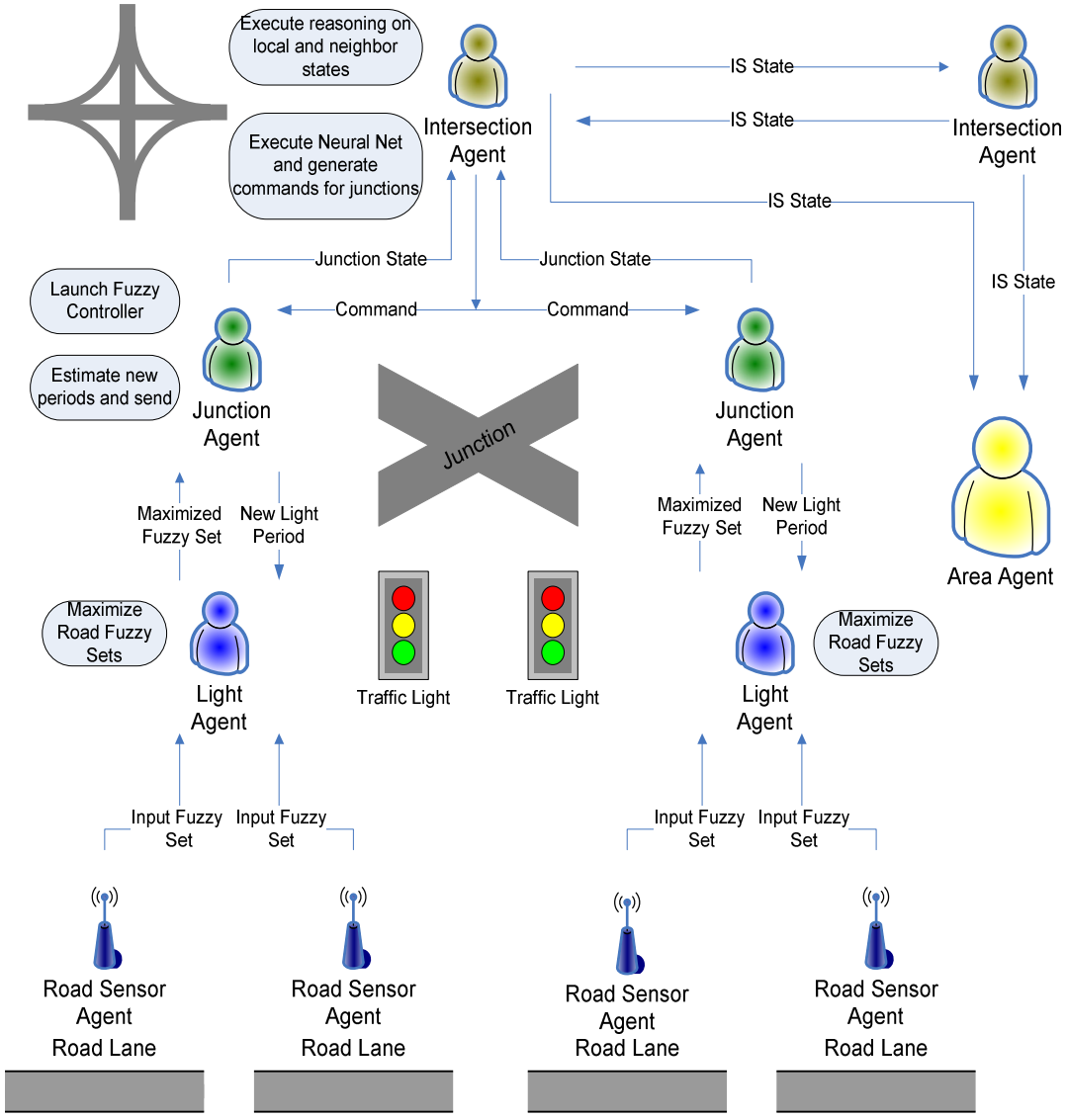


Figure 3.3 The general architectural view of the system.

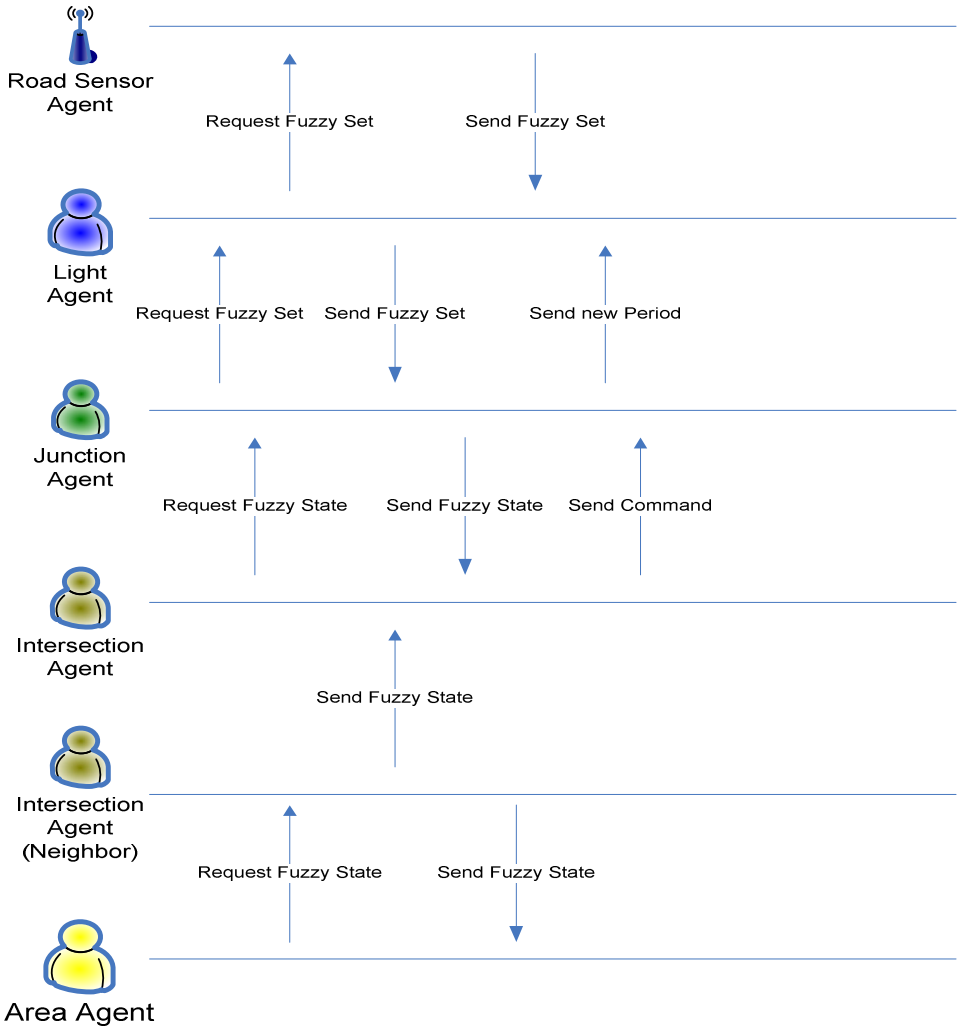


Figure 3.4 The messaging between agents.

3.3 Assumptions

- There is no yellow light period defined in this system. It is assumed as a portion of the green light.
- There is no direct synchronization between intersections. However, on behalf of communication between intersection agents, it is assumed that there is a cascaded synchronization and it may provide some kind of green wave opportunity for the vehicles. Note that green wave is a simple model applied to the traffic lights that have predicted patterns. Green wave algorithms are mostly applied to pre-timed controllers.

- There is no turn movement control in an intersection. On the other side, if turn movements are controlled by traffic lights, they are integrated to the junctions in triple level.
- There are minimum values for the red light periods. It is stated as %10 of cycle time.
- Road lane densities will be generated by a simulator (Random number or a pre configured file). In real life, there are some devices called RTMS (Real Time Monitoring System) that watches the target road lane segment and generates an occupancy-density volume data which shows the vehicle usage rate of road for a period of time.

3.4 Agents

In the following sub-chapters, each agent, its roles, actions, and their interactions with other agents are given in detail.

3.4.1 Road Agent

Road Agent is the bottom unit of the system. These are responsible of sensing one roadway direction. It estimates the density of road that is occupied by vehicles. This estimation is then processed to obtain a fuzzy data set.

The estimation is done instantly. It means that Road Agent waits a request from its master agent, Light-Agent. When it gets the request, it reads density from the sensor and generates fuzzy set, and then sends it back to the Light Agent as a reply.

Road Agent is the starting point of the fuzzy controller implementation. Each density estimated is translated into a fuzzy input set. The fuzzy input graph is as follows:

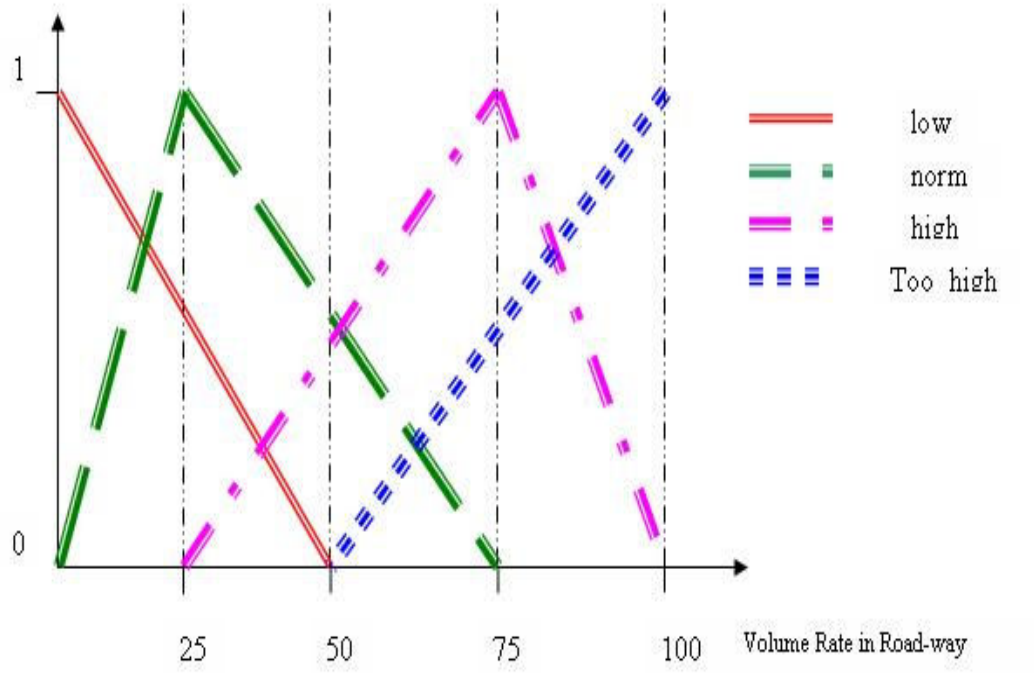


Figure 3.5 The fuzzy input data set graph.

Depending on this graph, the following data structure is defined. Dominant State is also estimated by selecting the highest value group.

Fuzzy Input Set

- Low value rate; (0<x<1)
- Normal value rate; (0<x<1)
- High value rate; (0<x<1)
- Too_High value rate; (0<x<1)

The data structure is filled by a case based evaluation that is given in Appendix A.

The road agent work flow diagram is given in Figure 3.6. As it is seen from the diagram, there is a registration phase at start-up. Master, Light Agent requires the addresses of Road Agents to send and receive messages. To implement this functionality, during start-up, each road agent registers itself to System level name service. The parameters that are used for the registration:

- Agent Name (its unique id)
- Group Name (Sub-Group Name, Ex: L-1)

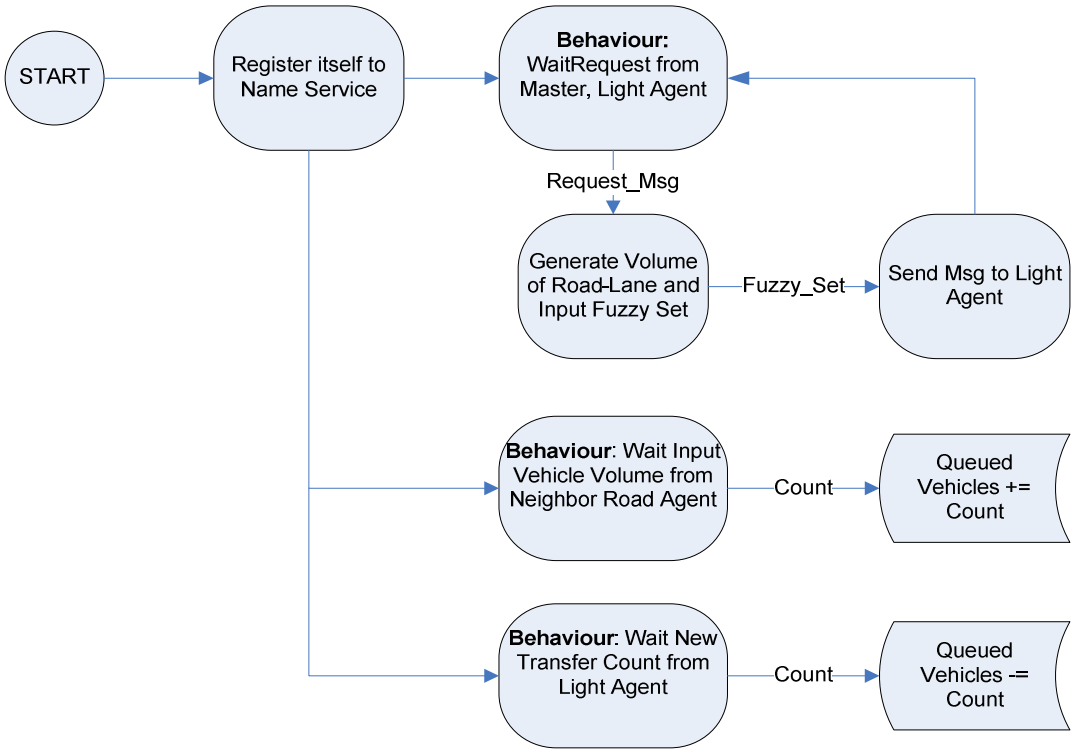


Figure 3.6 The road agent flowchart diagram.

3.4.1.1 Vehicle Flow between Road Agents

In order to get some simulative results, communication channels have been established between road agents. This is required because some road directions have no direct input generator. They get vehicle flow from previous road lanes according to the topological positioning. In order to transfer vehicle flow to these intermediate road directions, a new configuration table has been constructed. This table provides the leaking rate for transferring vehicle densities from source road lanes to target road lanes. Leak rate is parametric and can be changed anytime.

Table 3.1 Road to road connections configuration table

SourceRoad	**TargetRoad**	**LeakRate**	**TargetMaster**
Road-ID	Road-ID	%value	Junction-ID

Road lanes that have input generator (detector) are simulated in our programs using data files. Each data file has same name with Road Agent. Whenever a request comes from the master Light Agent, Road Agent reads its detector information from file and if exists from previous Road agent communication channel and then combines them for the “fuzzification”.

3.4.2 Light Agent

A light agent represents a traffic light in an intersection. It mainly controls its slave Road Agent(s). The work-flow diagram of Light Agent is given in Figure 3.7. When the Light Agent starts, it first registers itself to System-level name service, because Junction Agent, Master of Light Agents should find the addresses of its slave Light Agents to communicate. During registration, as in Road Agent, the following parameters are used:

- Agent Name (Sub-Group Name, Ex: L-1)
- Group Name: (Super-Group Name, Ex: Junction-1)

All actions defined in Light Agent: Waiting Fuzzy Data Request from Junction Agent, Waiting New Red Light Period from Junction Agent and periodically collecting fuzzy data of Road Agents, are executed as parallel and simultaneously.

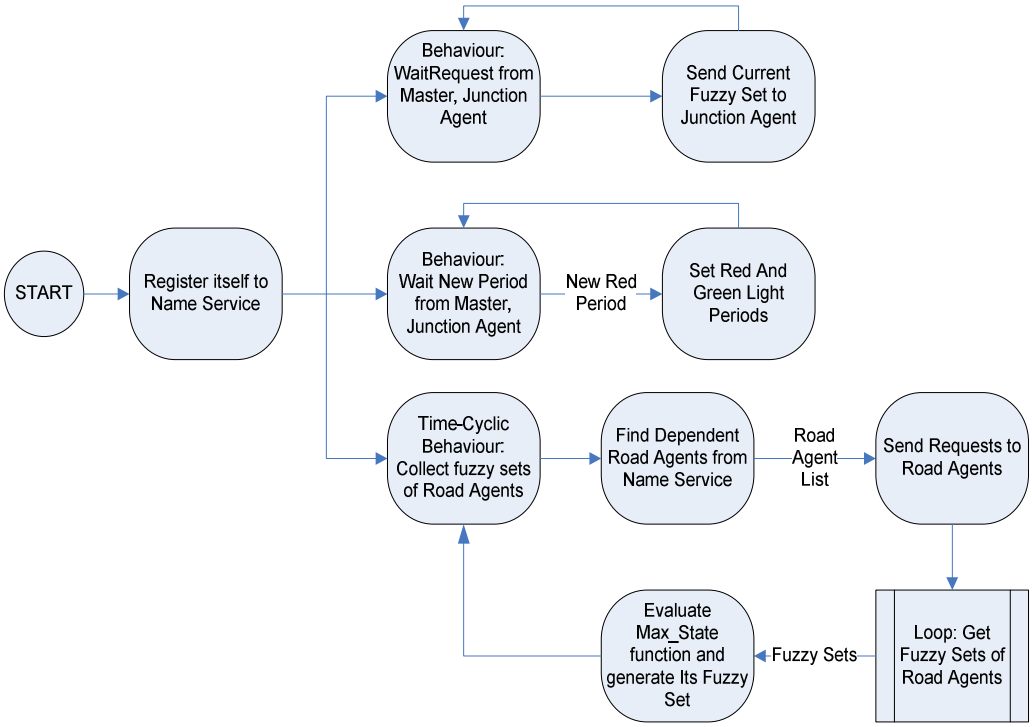


Figure 3.7 The light agent flow diagram.

After Light Agent collects all fuzzy data sets of dependent Road Agents, it forwards them into a maximum function. This function is applied to the members of fuzzy data set one by one. The maximization function is given in Appendix B.

3.4.3 Junction Agent

This agent contains the main intelligence part of the system. It collects the fuzzy data sets of its slave light agents and then applies the fuzzy controller to this data set. The output of the fuzzy controller will be the estimated change rate of red light period for one chosen light agent.

When the Junction Agent starts, it first registers itself to System-level name service, because Intersection Agent, Master of Junction Agents should find the addresses of its slave Junction Agents to communicate. During registration, as in Light Agent, the following parameters are used:

- Agent Name (Sub-Group Name, Ex: Junction-1)
- Group Name: (Super-Group Name, Ex: I-1)

This agent has four parallel actions: “FuzzyCollector”, “WaitIntersectionCommand”, “WaitRequestFromIntersection”.

1. “FuzzyCollector” gets the fuzzy data sets of dependent Light Agents and then processes them to estimate the initial change rate. This action is time-cyclic and the timer frequency is parametric.
2. “WaitIntersectionCommand” waits a command message from the Master, Intersection agent. This message defines the general purpose of the intersection. The incoming data type is directly processed to decrease or increase the red light period of the reference group.
3. “WaitRequestFromIntersection” sends its fuzzy state to Intersection agent upon its request.

The work flow diagrams of Junction Agent actions are given in Figure 3.8, 3.9 and Figure 3.10. The registration part is almost the same like other mentioned agents and it provides that Intersection agent finds the address of this Junction Agent to send messages.

One of the intelligence functionalities is fuzzy logic that is implemented in “FuzzyCollector” Behavior. After collecting fuzzy set values of its dependent Light Agents, they are classified into two groups according to pre-defined correlation table. This table keeps which light agent takes place in which group. An example of grouping of light agents is given below.

Table 3.2 Reference-Opponent light agent groups relation table

Intersection Name	**Junction Name**	**Light Agent Name**	**Group**
IS-1	J-1	L-1	Reference
IS-1	J-1	L-2	Opponent

The group names are Reference and Opponent. Reference group contains the light agents that show green and red signalizations at the same time in the same period, and Opponent group consists of the opposite part in the Junction. This information is pre-requisite for the system. For each road-cross in an intersection or road-cross itself, is a junction and the junction agent delegates it. To define the fuzzy data sets of Reference and Opponent groups, the maximization function that is already used before in Light Agent actions, is applied again.

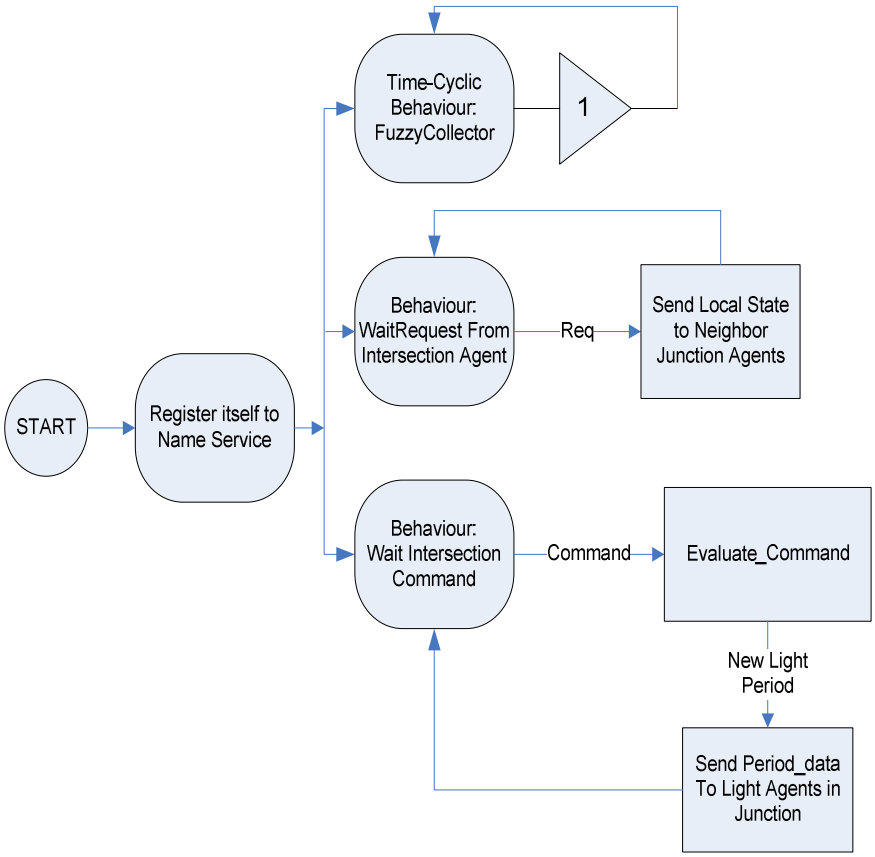


Figure 3.8 The junction agent flow diagram I.

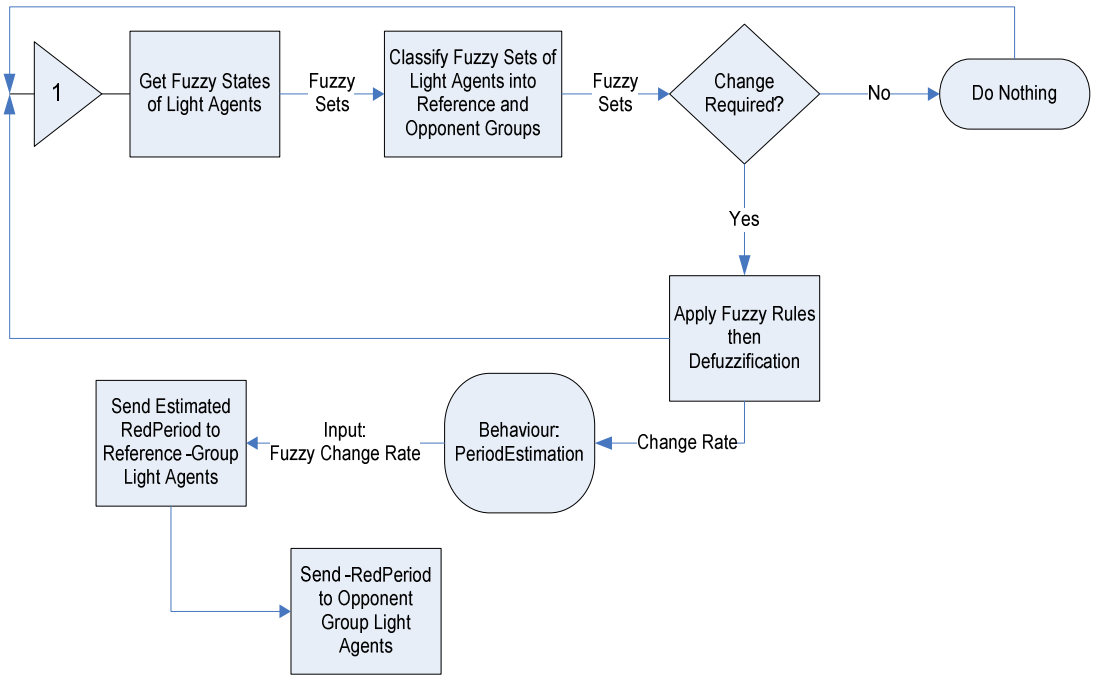


Figure 3.9 The junction agent flow diagram II.

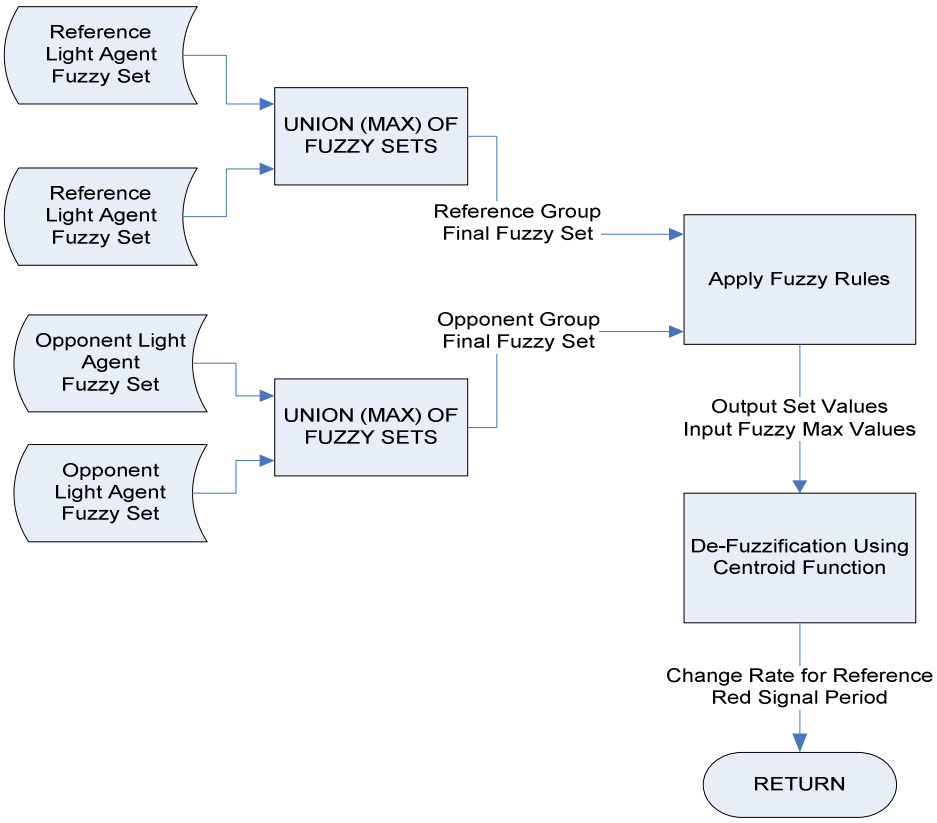


Figure 3.10 The fuzzy logic control processing at junction agent.

3.4.3.1 Distributed Fuzzy Logic Controller

After the classification of the light agents into two groups, fuzzy rules are applied over input fuzzy sets. The output values of rules (Membership Union Values and Output Fuzzy Set Central Mass Values) are then transferred into de-fuzzification process. This process generates the change rate of red light period for Reference group as the result of session evaluation. Execution of rules generates the output fuzzy set.

FUZZY RULE LIST:

a. if Reference is Low and Opponent is Low then Do Nothing (Zero)
b. if Reference is Low and Opponent is Normal then Do Nothing
c. if Reference is Normal and Opponent is Low then Do Nothing
d. if Reference is Normal and Opponent is Normal then Do Nothing
e. if Reference is High and Opponent is High then Do Nothing
f. if Reference is Too High and Opponent is Too High then Do Nothing
g. if Reference is Low and Opponent is High then change period in Positively Medium
h. if Reference is Low and Opponent is Too High then change period in Positively Large
i. if Reference is Normal and Opponent is High then change period in Positively Small
j. if Reference is Normal and Opponent is Too High then change period in Positively Medium
k. if Reference is High and Opponent is Low then change period in Negatively Medium
l. if Reference is High and Opponent is Normal then change period in Negatively Small
m. if Reference is High and Opponent is Too High then change period in Positively Small
n. if Reference is Too High and Opponent is Low then change period in Negatively Large

o. if Reference is Too High and Opponent is Normal then change period in Negatively Medium

p. if Reference is Too High and Opponent is High then change period in Negatively Small

Based on these definitions, the following formulas are generated:

```
//RULE-1
MVal1 = MIN (ReferenceFuzzy.LowValue, OpponentFuzzy.LowValue);
OVal1 = OutputValue[3]; //ZERO
//RULE-2
MVal2 = MIN (ReferenceFuzzy.LowValue, OpponentFuzzy.NormalValue);
OVal2 = OutputValue[3]; //ZERO
//RULE-3
MVal3 = MIN (ReferenceFuzzy.NormalValue, OpponentFuzzy.LowValue);
OVal3 = OutputValue[3]; //ZERO
//RULE-4
MVal4 = MIN (ReferenceFuzzy.NormalValue,
OpponentFuzzy.NormalValue);
OVal4 = OutputValue[3]; //ZERO
//RULE-5
MVal5 = MIN (ReferenceFuzzy.HighValue, OpponentFuzzy.HighValue);
OVal5 = OutputValue[3]; //ZERO
//RULE-6
MVal6 = MIN (ReferenceFuzzy.TooHighValue,
OpponentFuzzy.TooHighValue)
OVal6 = OutputValue[3]; //ZERO
//RULE-7
MVal7 = MIN (ReferenceFuzzy.LowValue, OpponentFuzzy.HighValue);
OVal7 = OutputValue[1];//POSITIVE_MED
//RULE-8
MVal8 = MIN (ReferenceFuzzy.LowValue, OpponentFuzzy.TooHighValue);
OVal8 = OutputValue[0];//POSITIVE_LARGE
```

```
//RULE-9
MVal9 = MIN (ReferenceFuzzy.NormalValue, OpponentFuzzy.HighValue);
OVal9 = OutputValue[2];//POSITIVE_SMALL
//RULE-10
 MVal10 = MIN (ReferenceFuzzy.NormalValue,
 OpponentFuzzy.TooHighValue);
OVal10 = OutputValue[1];//POSITIVE_MED
//RULE-11
MVal11 = MIN (ReferenceFuzzy.HighValue, OpponentFuzzy.LowValue);
OVal11 = OutputValue[5];//NEGATIVE_MED
//RULE-12
MVal12 = MIN (ReferenceFuzzy.HighValue, OpponentFuzzy.NormalValue);
OVal12 = OutputValue[4];//NEGATIVE_SMALL
//RULE-13
 MVal13 = MIN (ReferenceFuzzy.HighValue,
 OpponentFuzzy.TooHighValue);
OVal13 = OutputValue[2];//POSITIVE_SMALL
//RULE-14
 MVal14 = MIN (ReferenceFuzzy.TooHighValue,
 OpponentFuzzy.LowValue);
OVal14 = OutputValue[6];//NEGATIVE_LARGE
//RULE-15
 MVal15 = MIN (ReferenceFuzzy.TooHighValue,
 OpponentFuzzy.NormalValue);
OVal15 = OutputValue[5];//NEGATIVE_MED
//RULE-16
 MVal16 = MIN (ReferenceFuzzy.TooHighValue,
 OpponentFuzzy.HighValue);
OVal16 = OutputValue[4];//NEGATIVE_SMALL
```

Although the output fuzzy values are obtained by the fuzzy rules, we still need the real output value (change rate of red light period). In order to do it, De-fuzzification step is implemented using The Weighted Centroid function.

3.4.3.2 The Weighted Defuzzification Technique

With this method the output is obtained by the weighted average of the each output of the set of rules stored in the knowledge base of the system. The weighted average defuzzification technique can be expressed as

$$x^* = \frac{\sum_{i=1}^{n} m^i w_i}{\sum_{i=1}^{n} m^i}$$

where x^* is the defuzzified output, m^i is the membership of the output of each rule, and w_i is the weight associated with each rule. This method is computationally faster and easier and gives fairly accurate result. This defuzzification technique is applied in fuzzy application of signal validation and fuzzy application on power. If we extract this formula into more understandable format

$$X_0 = \frac{X_1 * M_1 + X_2 * M_2 + \ldots + X_n * M_n}{M_1 + M_2 + \ldots + M_n}$$

OTotal = MVal7*OVal7 + MVal8*OVal8 + MVal9*OVal9 + MVal10*OVal10 + MVal11*OVal11 + MVal12*OVal12 + MVal13*OVal13 + MVal14*OVal14 + MVal15*OVal15 + MVal16*OVal16;

MTotal = MVal1 + MVal2 + MVal3 + MVal4 + MVal5 + MVal6 + MVal7 + MVal8 + MVal9 + MVal10 + MVal11 + MVal12 + MVal13 + MVal14 + MVal15 + MVal16;

ChangeDelta = OTotal/MTotal;

In our model, X values on the right side refer to central mass value of output fuzzy sets and M values refer to union membership values of input fuzzy sets of rules. Each central mass value, X, refers to x-coordinate value of the centroid of triangular output fuzzy set.

The centroid of a triangle is the point of intersection of its medians (the lines joining each vertex with the midpoint of the opposite side). The centroid divides each of the medians in the ratio 2:1, which is to say it is located ⅓ of the perpendicular distance between each side and the opposing point as illustrated in the figure below

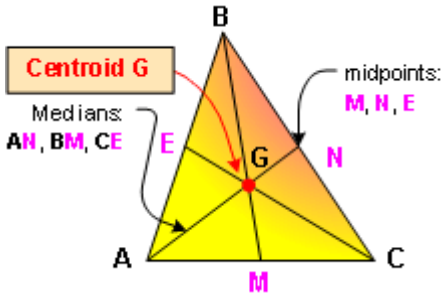


Figure 3.11 The centroid point of a triangle.

The defined output fuzzy set is also given below:

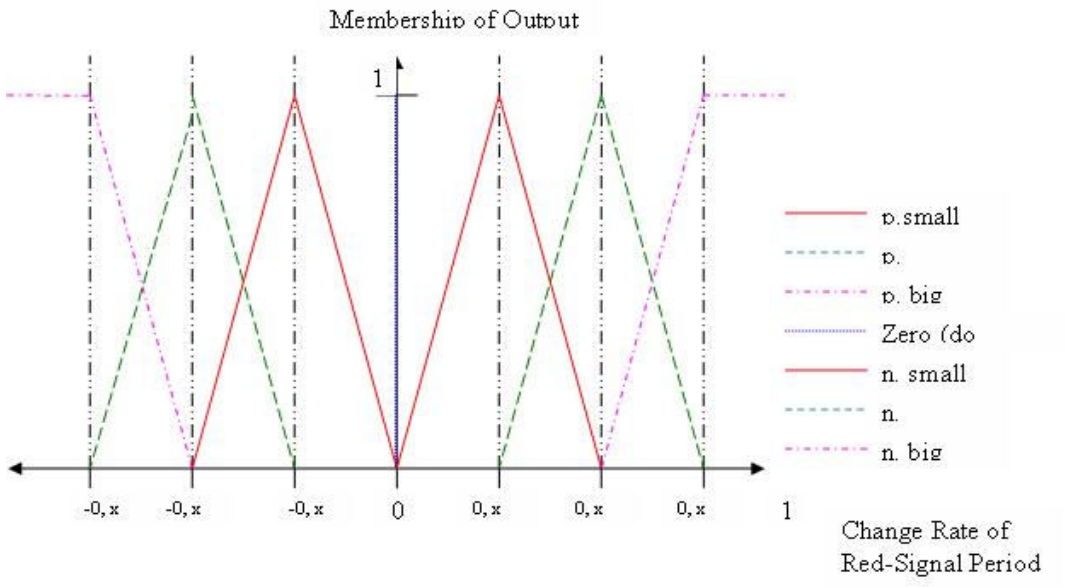


Figure 3.12 The output fuzzy set graph.

Based on the centroid principle, the X coordinate-values of centre of masses of output sets are listed in the following table. (Output sets consist of isosceles triangles and trapezoids)

Table 3.3 Predefined centroid values of fuzzy output sets

Output Value	**Centroid**
Positively large	0.75
Positively medium	0.50
Positively low	0.25
Do Nothing (Zero)	0
Negatively small	-0.25
Negatively medium	-0.50
Negatively large	-0.75

To summarize all stuff, the following fuzzy controller graph is drawn:

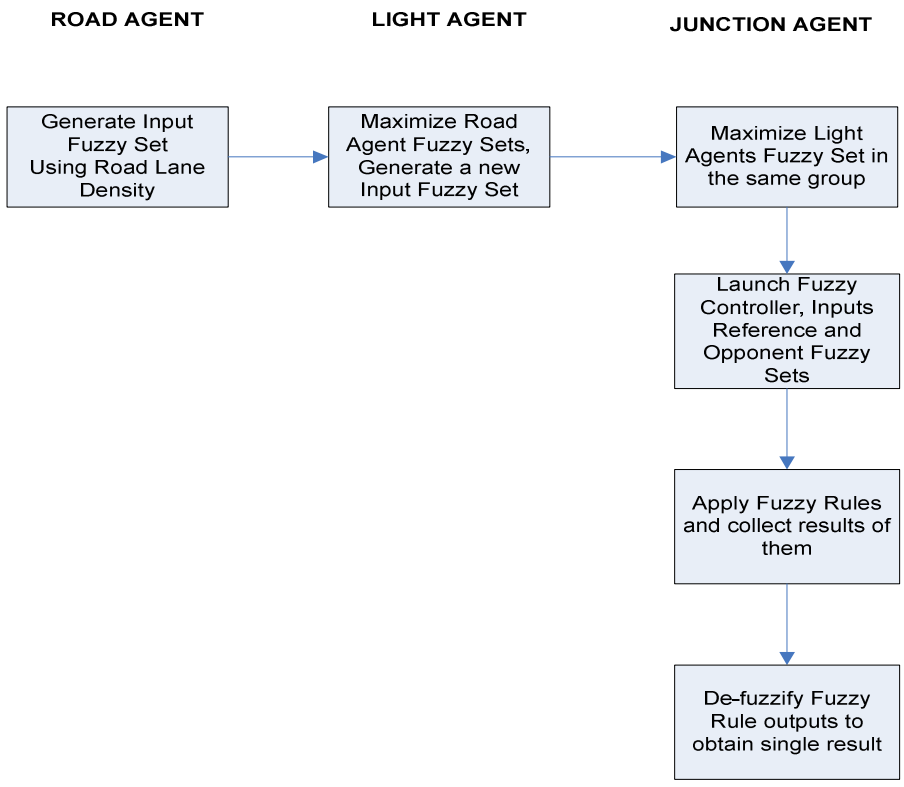


Figure 3.13 The agent-wide fuzzy logic implementation flow.

3.4.3.3 Handling Multiple Road Flows in Junctions

Since the beginning of the project, it was assumed that a Junction controls two primary road directions. However, some junctions including turn movements consist of three or sometimes more road flows. In order to handle these types of junctions, the design internals of Junction Agent have been little changed. For each unique turn movement flow, related light agents are contained in new light groups.

In Junction agent, all combinations of light agent groups are generated and for each combination of the reference and opponent groups, the fuzzy controller is launched repetitively. Each fuzzy controller result output is forwarded into an average function. The final average values are then used to estimate new light period and this output is sent to the related Light Agent group.

With this modification, the project has gained ability to manage controlled turn movements in a junction. However, the turn movement groups are still ignored in correlation configuration tables. It was assumed that turn movement is an implicit action and it should not affect relations between main road flows of junctions and intersections.

3.4.4 Intersection Agent

This agent is the bridge between the global control and local control of the intersections. Global control includes collecting neighbor intersection states and according to states of the neighbor intersections, generating commands for local junctions. Local control includes getting states of sub junctions and generating commands if necessary to balance the traffic load implicitly.

Intersection Agent controls pre-defined junction agents in a tightly coupled road-crosses. It can be seen as a super-junction agent. All junction agents send their local state information to Intersection agent and then it checks neighboring relations. Moreover, it gets status information from its neighbor intersection agents.

Neighboring relations are based on after-before position of junctions and intersections.

Intersection Agent generates local and global information in two separate cycles. If more than one junction exists in an intersection topology, the local information is generated based on their states. To generate global information, at least one neighbor intersection should be defined in the target area. The work flow cycles of Intersection Agent are given in Figure 3.14.

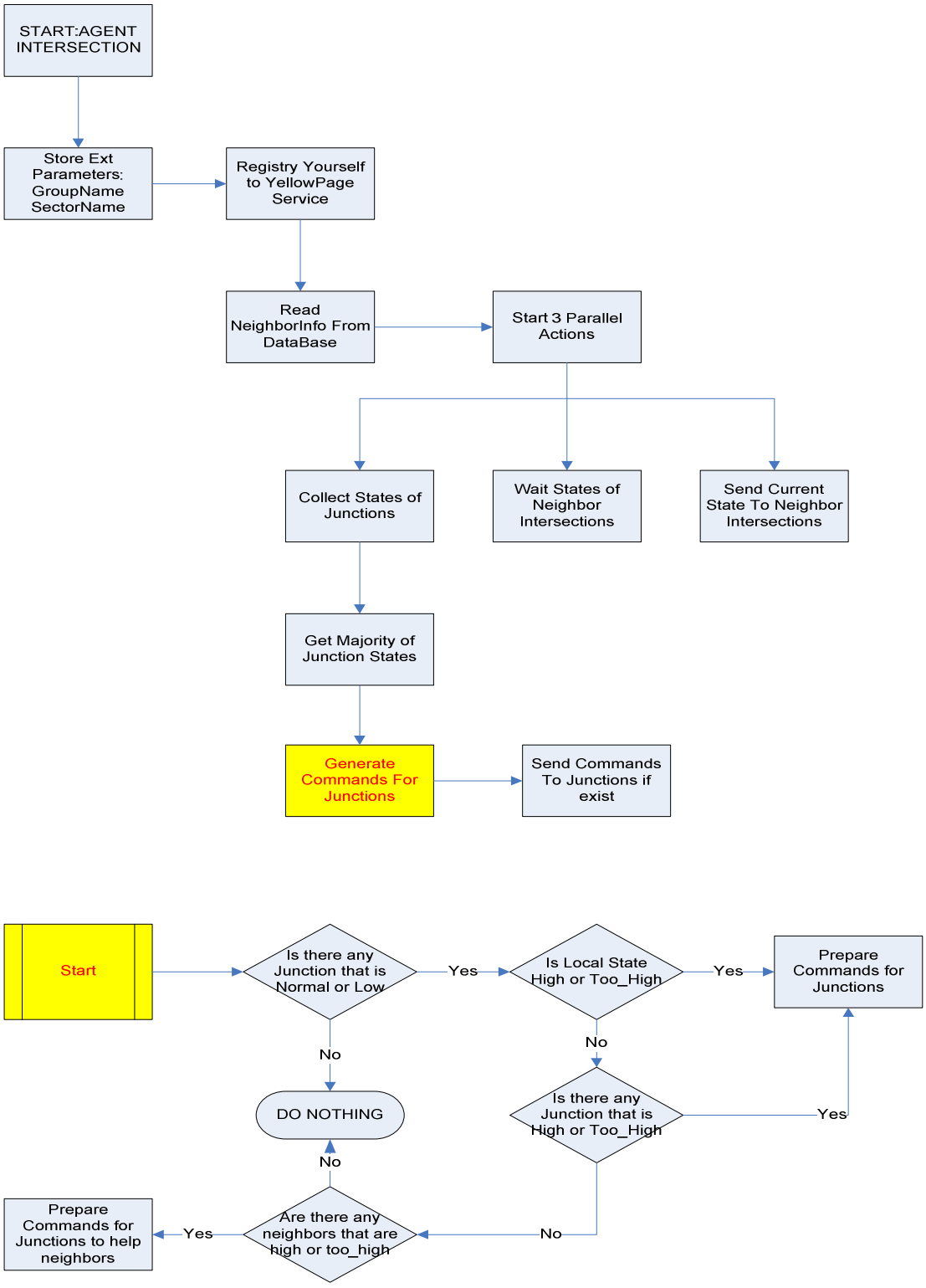


Figure 3.14 The intersection agent work flow cycles.

As it is expected, intersections consist of only one junction. In that case, it has only one critical role: getting states of neighbors. If one of the neighbors is in high

density, and its local state junction(s) are in normal level, it can change the light periods of junction to help the neighbor in trouble.

The registration part is similar to the other agents. When the Intersection Agent starts, it first registers itself to System-level name service, because Area Agent, Master of Intersection Agents and other neighbor agents should find the addresses of this Intersection Agent to communicate. During registration, as in Junction Agent, the following parameters are used:

- Agent Name (Its unique id, Ex: IS-1)
- Group Name: (Name of Sector, Ex: Sector-1)

Intersection Agent keeps two relations tables. First one lists the members of neighbor intersections. The layout of it is as follows. This table is used to send local intersection to neighbors.

Table 3.4 Neighbor list table for an intersection

Intersection Name	**Neighbor-Name**
IS-1	IS-2
IS-1	IS-3
IS-1	IS-4

The other relation is about the links between junctions in an intersection and between intersections. The data in this table is used during rule-based reasoning to help sub-junctions or neighbors in trouble. According to Figure 3.15 and Figure 3.16, the complete junction to junction relations are as follows.

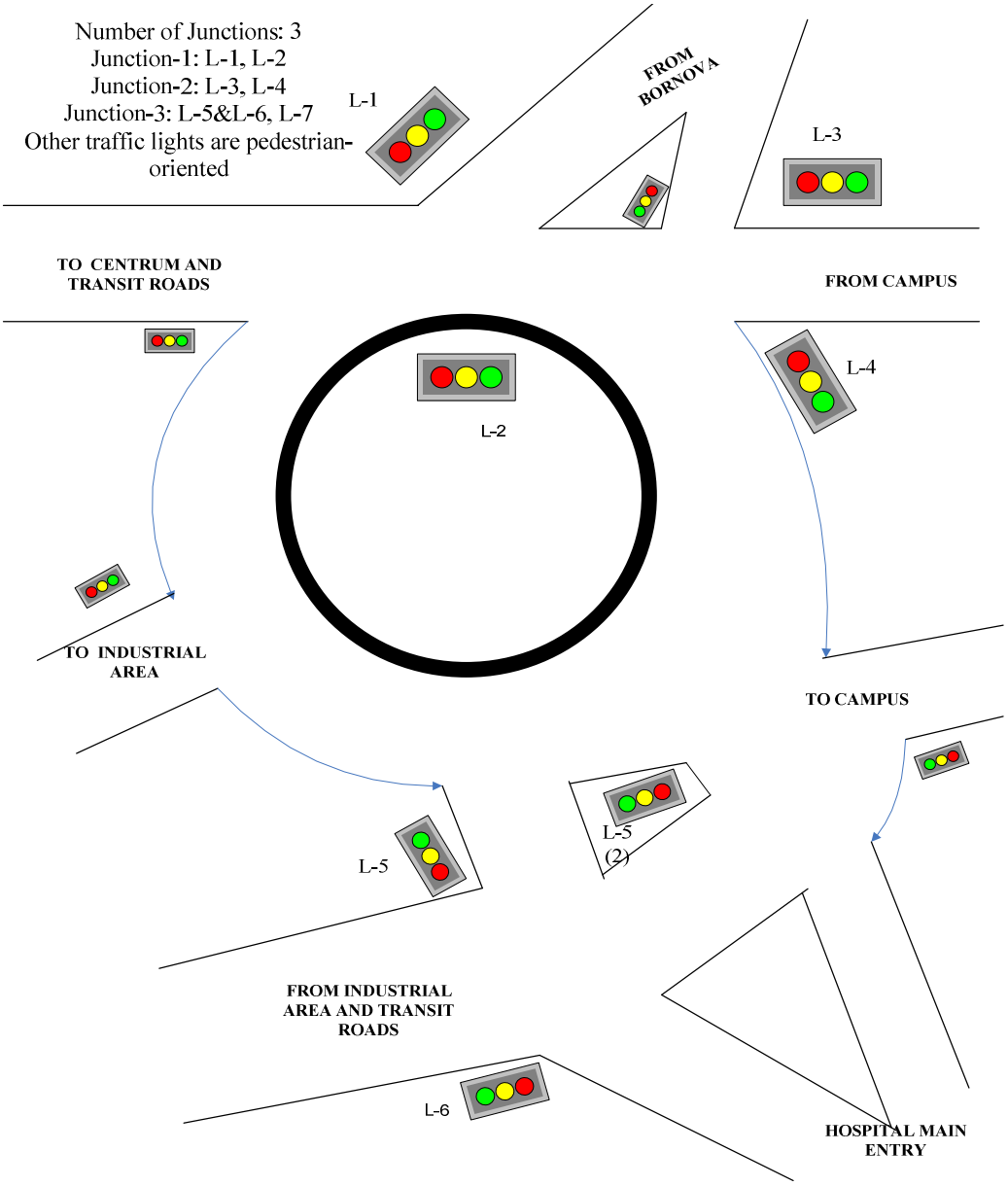


Figure 3.15 Ege university hospital intersection and sub junctions.

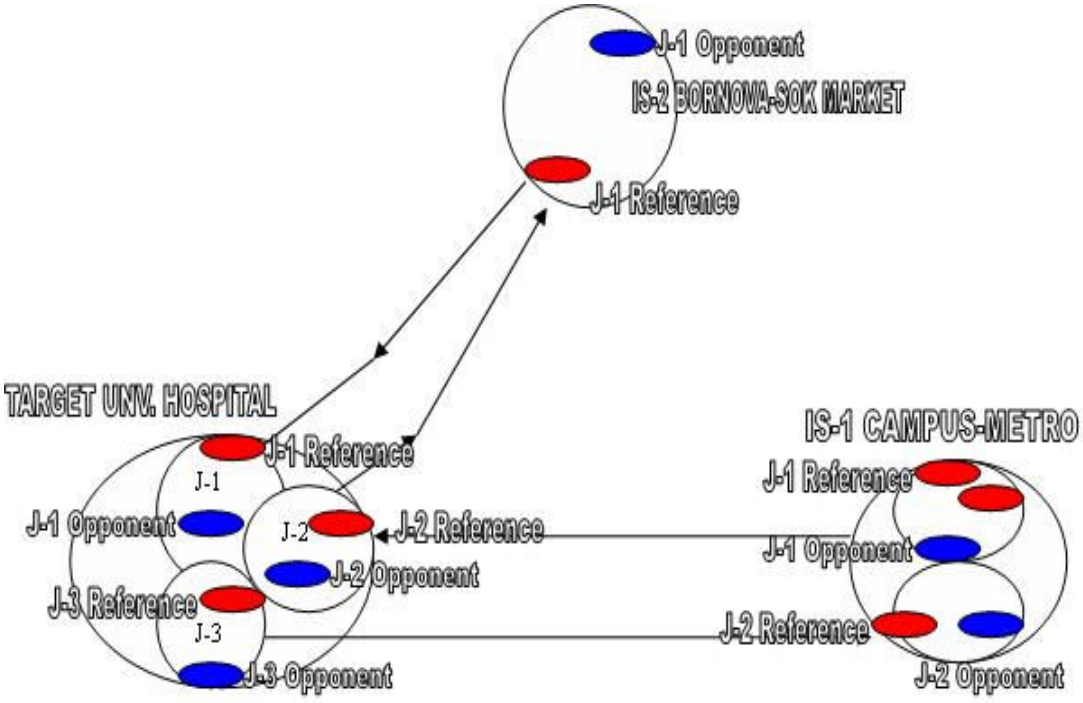


Figure 3.16 Junction to junction and neighbor intersection relations for the sample intersection.

However, most of them are ignored or modified because they have connections to both parts of junctions (reference and opponent). If both junction sub groups have a connection to the same part of other another junction or neighbour intersection, it can not take a place in this relation table. For example, if a junction-sub group is in trouble, adjacent junction can be taken under evaluation. If both parts of this junction have direct relation with sub-group in trouble, it will have no sense. Therefore, only a subset of these relations is taken into the table as in the following structure.

Table 3.5 Complete relations

Local Junction ID With Sub Group Name	**Linked To**	**Relation Type**	**Neighbor**	**Position Type** (according to local junction)
J-2 Reference	J-1 Reference	Neighbor	IS-2	Next
J-2 Opponent	J-1 Reference	Neighbor	IS-2	Next
J-2 Reference	J-1 Opponent	Local	-	Next
J-2 Opponent	J-1 Opponent	Local	-	Next
J-3 Reference	J-2 Reference	Neighbor	IS-1	Next

J-3 Opponent	J-2 Reference	Neighbor	IS-1	Next
J-3 Reference	J-2 Opponent	Local	-	Next
J-3 Opponent	J-2 Opponent	Local	-	Next
J-1 Reference	J-1 Reference	Neighbor	IS-2	Previous
J-1 Opponent	J-2 Reference	Local	-	Previous
J-1 Opponent	J-2 Opponent	Local	-	Previous
J-2 Reference	J-1 Reference	Neighbor	IS-1	Previous
J-2 Reference	J-1 Opponent	Neighbor	IS-1	Previous
J-2 Opponent	J-3 Reference	Local	-	Previous
J-2 Opponent	J-3 Opponent	Local	-	Previous
J-3 Reference	J-1 Opponent	Local	-	Previous
J-3 Reference	J-1 Reference	Local	-	Previous

Table 3.6 Main junctions to neighbors (local junction or neighbor intersection) link table

Junction-SubGroup	**Linked To**	**Relation Type**	**Neighbor**	**Position Type** (according to local junction)
J-1 Reference	J-1	Neighbor	IS-2	Prev
J-1 Opponent	J-2	Local	-	Prev
J-2 Reference	J-1	Neighbor	IS-1	Prev
J-2 Opponent	J-3	Local	-	Prev
J-3 Reference	J-1	Local	-	Prev

Each intersection agent launches its rule based algorithm after collecting local junction and neighbor intersection states. This algorithm is given in the next page.

After generating commands in fuzzy value rank, they are forwarded into a neural net so that more accurate and realistic light periods will be generated. The neural net details are given in the following chapters.

3.4.4.1 State Reasoning

The given algorithm below shows the flowchart of the Junction and Neighbor intersection state reasoning occurs at Intersection Agent.

```
Collect States of Junction Light Groups
My_State = Majority of Local Junction States
Collect States of Junction Light Groups of Neighbor Intersections
 If LocalState is high or too_high AND Junction_Count > 1 then
     Call Manage_LocalState(Junction)
 Else
     Call Manage_Neighbors(Intersection);
End if
Manage_LocalState(Junction)
Begin
For I 1 to N (Number of Junctions)
If (Junction[I].State is High or Too_High) then
        Link[] = GetLinksFromDB(Junction[I], LOCAL);
        For J 1 TO N (Number of Links)
            XState = Link[j].State;
            XEdge = Link[j].EdgeType;//Opponent or Reference
            XPos = Link[j].Pos;//Prev or Next
            XIndex = getJunctionIndex(Link[j].Name);
            If XState is NOT Too_High then //Normal, Low or High
                XFactor = ABS(Junction[I].State – XState);
                ChgRate[ XIndex ] += XFactor* Value[XPos][XEdge];
            End if
        End For
        End if
End if
End For
End
```

```
……
Manage_Neighbors(Intersection)
For I 1 to N (Number of Intersections)
If (Neighbor[I].State is High or Too_High) then
        Link[] = GetLinksFromDB(Neighbor[I], NEIGHBOR);
        For J 1 TO N (Number of Links)
                XState = Link[j].State;
                XEdge = Link[j].EdgeType;//Ref or Opp
                XPos = Link[j].Pos;//Prev or Next
                XIndex = getJunctionIndex(Link[j].Name);
                If XState is NOT Too_High then //Normal, Low or High
                        XFactor = ABS(Neighbor[I].State – XState);
                        ChgRate[XIndex] += XFactor* Value[XPos][XEdge];
                End if
        End For
        End if
End if
End For
End
```

As you see, at each cycle, only one command is generated for a junction. The priority is firstly given to the internal part of intersections. Value array consists of constant values. The table values are as follows.

Table 3.7 Command evaluation matrix used at reasoning phase

XPos (Prev = 0, Next = 1)	***XEdge (Ref = 0, Opp = 1)***	***Value***	***Comment To Junction Cycle***
0	0	-1	Increase red period of Reference Group
0	1	1	Decrease red period of Reference Group
1	0	1	Increase red period of Reference Group
1	1	-1	Decrease red period of Reference Group

- The objective of the value table is to define the direction of change rate for the reference light agent group (controlled by the related Junction Agent).
- XPos value is defined by targeting neighbor pos according to local group (XEdge)
- XFactor can take a value between 0 and 3 (LOW = 0, TOO_HIGH = 3). In that case, change rate at one time is max 30%. And min 0%.

3.4.4.2 Neural Network Integration

In our application, a back-propagation neural network is integrated to the Intersection agent. **Back-propagation** is a supervised learning technique used for training neural networks (Rich & Knight, 1991, p. 500-507). It is most useful for feed-forward networks (networks that have no feedback or simply, that have no loop connections). The technique is as follows.

a. Present a training sample to the neural network.

b. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.

c. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.

d. Adjust the weights of each neuron to lower the local error.

e. Repeat the steps above on the neurons at the previous level, using each one's "blame" as its error.

In order to provide learning ability and attach other environmental parameters into the new light period decision making cycle, a neural network was attached to the Intersection agent work flow model. For our neural network, Three layers will be defined (one input, output and one hidden layer) Table 5 shows the input elements of the neural network

Table 3.8 Input elements of the designed neural network

Neural Net Input Parameters	Description
Command Rate coming from Reasoning Engine:	001 to 110 (-3≤fCommand ≤ 3)
Target ID Type:	(Local Junction or Neighbor Intersection) Either 0 or 1
Is Holiday:	0 or 1
Day Of Week:	001 to 111 (from Monday to Sunday)
Hour of Time:	00000 to 10111 (from 0 to 23)
Minute of Time:	000000 to 111011 (from 0 to 59)
Junction ID to give support:	(2^n+1 input neurons, n refers to the number of junctions in target intersection)
Target ID to get help:	(2^n+1 input neurons, n refers to the maximum number of junctions or neighbors for the target intersection)

All these inputs attached to the input layer of the designed neural network are connected to the neurons in the hidden layer and the output of the hidden nodes is connected to the neuron in output layer. The result of the output node represents the change rate for the red light period of target Junction Reference Group. To obtain accurate results, the training data must be well produced in high and effective densities.

The output of the ANN will be a new Light Period: 0< fPeriod <1. There will be a constraint on fPeriod. By using the supervised learning training samples, it will be eligible to generate output between 0 and Total_Cycle/100. The real output of neural network will be then multiplied by 100 and the final light period for reference junction group will be obtained.

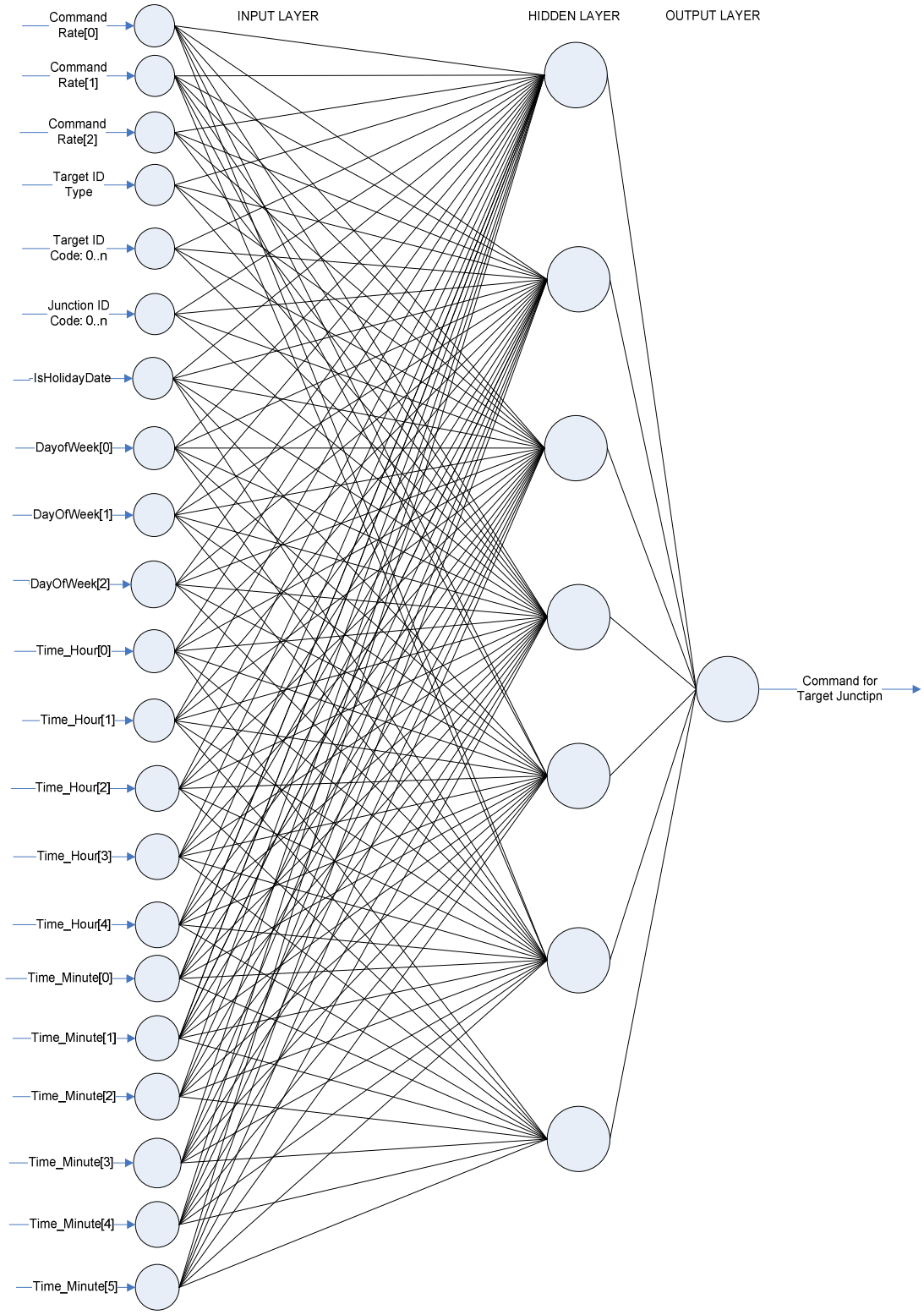


Figure 3.17 The neural network lay out.

The generated command rates are then sent to the target junctions. Each command is directly put in charge by Junction Agent to accelerate traffic.

3.4.4.3 Training data for Supervised Neural Networks

In each Intersection Agent, there will be an artificial neural network module running. In order to enable ANN (Artificial Neural Network), it should be trained first and if possible with a package of huge data. However, providing this kind of data with help of simulators is almost impossible. Therefore unsupervised network models might be under consideration in future. This kind of networks learns without an external teacher (note that in supervised networks, the output expected is given in training data), simply by detecting the similarities of the input patterns and categorizing (i.e. projecting) them on a (1D or 2D) map. In an Unsupervised learning, a data set of input objects is gathered. Unsupervised learning then typically treats input objects as a set of random variables. A joint density model is then built for the data set.

3.4.5 Area Agent

This agent is located in the root of hierarchy. This agent will have no automatic contribution to the system. It will be manual controller and watcher of the system. All intersection agents will send their local states to this agent. After collecting these local states, the operator will see the total picture and he will be able to send some commands: increase red, increase green in some levels to the intersection agents (like fuzzy output set names). Intersection Agents will get these commands to administer the junction agents in global manner. Also, the Area Agent will be able to send the real period values to the intersection agents and telling them to disable or enable their automated system.

According to the given explanations, the Area Agent will have two primary behaviors:

Send Commands: This action waits for the interaction of operator. When the system operator selects to send global command based on local states of intersections, he fills the required parameters. For each intersection, commands can

have different content. Like the declarations of fuzzy output sets, the command will be as follows:

1. Increase your capacity in small, medium or large levels
2. Decrease your capacity in small, medium or large levels.

When Intersection gets this command, it will inform its sub-junctions to reach that goal. Another option for the system operator is to define red and green light periods of a specific intersection with enabling or disabling automated signalization process. After parameters have been defined, the prepared message is sent to target intersection agent. Thus, the system will be able to provide pre-timed signalization work type.

The "Send Commands" behavior will show a dialog and from this dialog, the system operator will be able to send command to the target intersection.

Collect States: With this action, the local states of intersections are collected, and the system operator watches the traffic flow between intersections. Intersection agents define their local state again using the maximization function over junction agent states.

There is an obligatory registration part in Area Agent, because it is the root agent. However, it registers itself using its name. It also searches naming service to find the addresses of Intersection Agent to communicate. The work flow diagram of Area Agent is given in Figure 3.18.

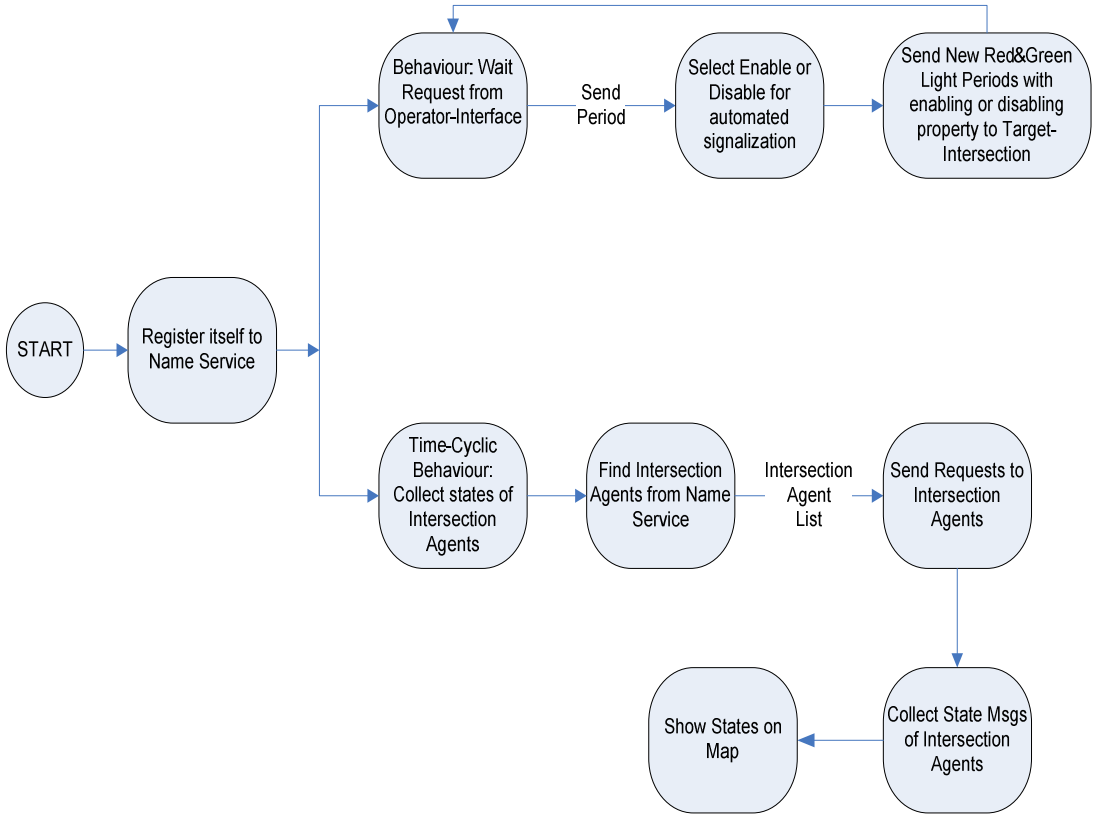


Figure 3.18 The area agent flow diagram.

Till now, all system internals are given in detail. In order to show how the model works, four sample scenarios are defined in Appendix D.

Scenario 1 figures out a primitive local intersection that consists of 1 junction. Scenario 2 figures out a bit more complex local intersection that consists of 2 junctions. Scenario 3 figures out a complex intersection that is a type of triple junction control including turn movements. Scenario 4 figures out a more complex intersection that consists of 3 junctions and 1 neighbor.

CHAPTER FOUR
IMPLEMENTATION

After completing the design cycle of the proposed model, the software development and simulation have been implemented. There are four parts of this development work: Coding Environment, Agent Library process, Neural Network process, Traffic Simulator search. In the following chapters, these processes are detailed.

4.1 Coding Environment

Software development of the project has been implemented over Windows XP PRO/SP2 OS platform. As IDE environment, Borland Java Builder 2005 Foundation edition v11.0 was used. As it is seen from IDE name, Java run-time and development environments (v1.5) were used. Java SDK is selected as primary language for all development work because it has platform independence over common operating systems and its object-oriented nature is very close to the Agent paradigms. (Encapsulation, independent execution, message based communication that triggers the target events) Moreover, more components and libraries are freely available on Java platforms.

The project covers some configuration and implicitly estimated data structure. All these information is kept in an MS Access 2003 database (The current view of the database is given in Appendix C). The database management of MS Access is simple and Java is connected to the Access database by JDBC-ODBC driver. MS Access database program was chosen because there is no huge collection of data in this system and only the configuration of traffic topologies and agent workflow parameters are kept in tables. System runs interactively on real-time data.

4.2 Agent Library Process

Agent programming is popular in coding and design parts of the software development cycle. Shortly it is a methodology that provides a context for componentizing application functionality through the abstraction of these features called as agents. In agent programming, agent is the basic element of distribution. Each agent serves as an independent component with its own local state and execution model. The agent designer can choose to assign a particular set of the functionalities to an agent, specify the types of events and messages that the agent may invoke and/or respond to, and implement those triggers and/or responses. Once an agent-based system is designed, it can be distributed flexibly across multiple CPUs and nodes in a network.

Before selecting target agent platform, four library options have been examined in this project: OpenCybele, Zeus, AgentKernel, and Jade. After the first review of agent languages and depending on requirements of our project, “OpenCybele” was chosen as agent library because of its easy to use and simplicity properties. However, during the practices with “OpenCybele”, some problems occurred and when a help is demanded from the creators of the environment, unfortunately no reply has been received. Therefore the second alternative, JADE, has been taken under investigation and finally it was seen that it was the best selection to continue.

Jade is a Java Agent DEvelopment Framework to develop multi-agent systems in compliance with the FIPA (Foundation for Intelligent Physical Agents) specifications. It is free software with GNU General Public License.

Jade has been fully coded in Java, and an agent programmer should code his/her agents in Java. Jade is made of various Java packages, giving application programmers both ready-made pieces of functionality and abstract interfaces for custom, application dependent tasks. Java was the programming language of choice because of its many attractive features, particularly geared towards object-oriented

programming in distributed heterogeneous environments; some of these features are Object Serialization, Reflection API and Remote Method Invocation (RMI).

4.2.1 JADE Features

JADE is built on Java platform. Its architectural position is given below.

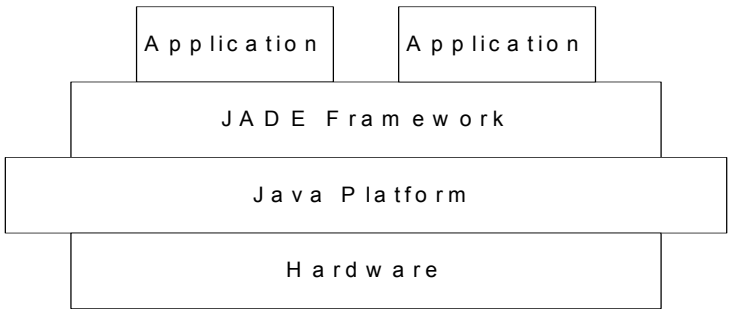


Figure 4.1 The Jade agent system infrastructure.

The standard model of an agent platform, as defined by FIPA, is also represented in the following figure. The Agent Management System (AMS) is the agent who exerts supervisory control over access to and use of the Agent Platform. Only one AMS will exist in a single platform. The AMS provides white-page and life-cycle service, maintaining a directory of agent identifiers (AID) and agent state. Each agent must register with an AMS in order to get a valid AID (Bellifemine, Caire, Trucco & Rimassa, 2007).

The Directory Facilitator (DF) is the agent who provides the default yellow page service in the platform. The Message Transport System, also called Agent Communication Channel (ACC), is the software component controlling all the exchange of messages within the platform, including messages to/from remote platforms.

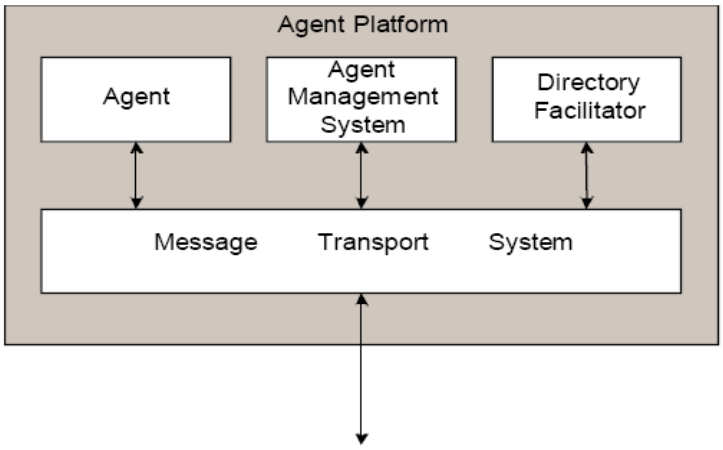


Figure 4.2 The Jade agent platform interactions.

Jade fully complies with this reference architecture and when a Jade platform is launched, the AMS and DF are immediately created and the ACC module is set to allow message communication. The agent platform can be split on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host. Each JVM is a basic container of agents that provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host. The main-container, or front-end, is the agent container where the AMS and DF lives and where the RMI registry, that is used internally by Jade, is created. The other agent containers, instead, connect to the main container and provide a complete run-time environment for the execution of any set of Jade agents.

An Agent super class is defined in Jade library and all user agents are inherited from this Agent class. An agent must be able to carry out several concurrent tasks in response to different external events. In order to make agent management facility efficient, every Jade agent is composed of a single execution thread and its tasks are modeled and can be implemented as Behavior objects. Jade Behavior class hierarchy is given below.

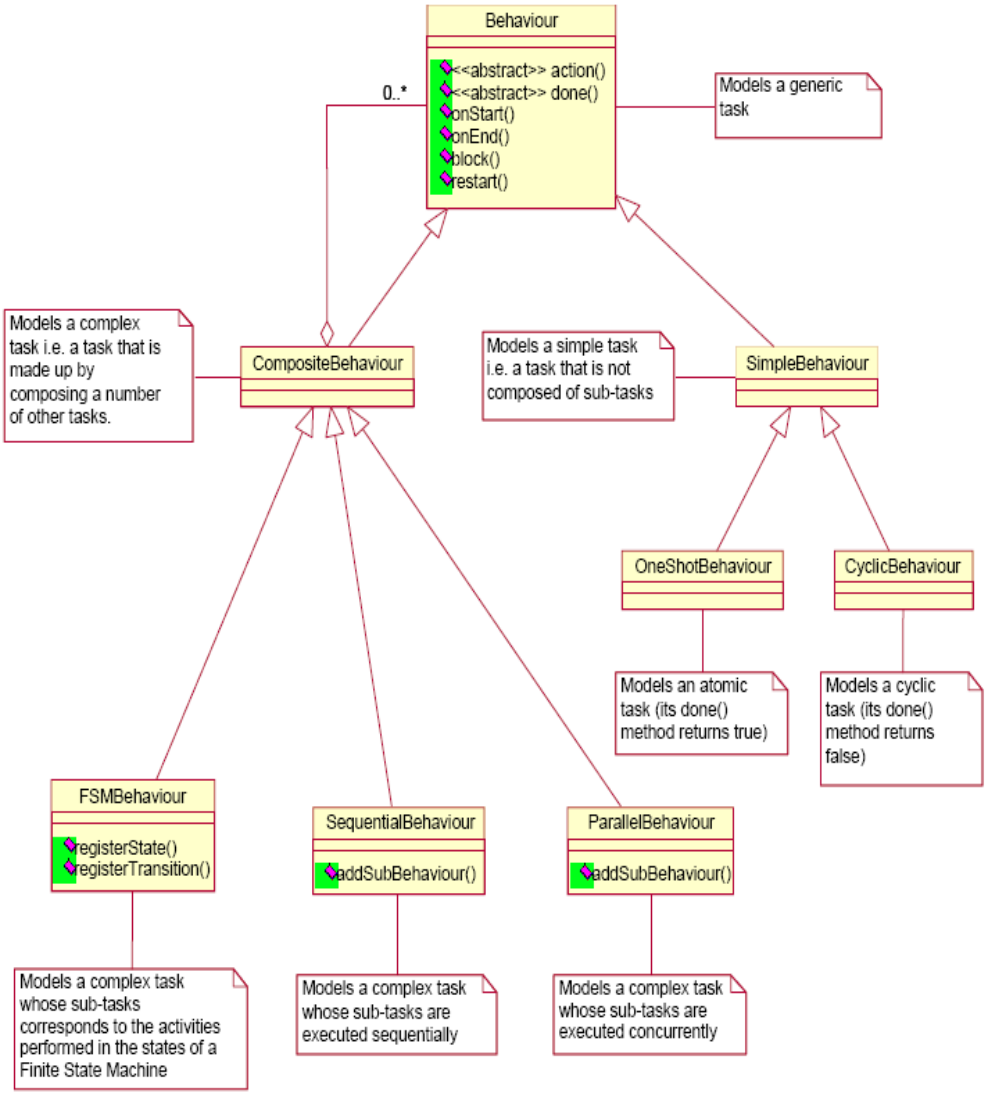


Figure 4.3 The Jade agent behavior class hierarchy.

4.3 Neural Network Process

In order to give learning ability to Intersection agents at global decision-making progress, a neural network model has been defined. Instead of writing the neural-net code directly, a free Java library search has been done.

Some of the examined ANN libraries are Matlab Neural Network Tool, NNUtility, JavaNNS, Neurak, NumMap7 and Joone.

A group of libraries given above were based on C/C++ language. Therefore, they were eliminated because of our Java programming environment. Some of them require license fee and some libraries look complex and not easy to use quickly. At the end of all evaluations and comparisons, Joone was selected as primary neural network library. It is Java based Object Oriented Neural Engine that is most popular and recommended free engine. It has both Java API access and GUI interface.

4.3.1 Joone Features

Joone is a free Neural Network framework to create, train and test artificial neural networks. The aim is to create a powerful environment both for enthusiastic and Professional users, based on the newest Java technologies.

Joone is composed of a central engine that is the fulcrum of all applications that are developed with Joone. Joone's neural networks can be built on a local machine, be trained on a distributed environment and run on whatever device. Everyone can write new modules to implement new algorithms or new architectures starting from the simple components distributed with the core engine (Marrone, 2005)

The kernel of the engine is the Layer object. It is composed by N neurons (that can be set by the attribute 'rows'). Imagine a feed-forward neural net composed by three layers like the following graph:

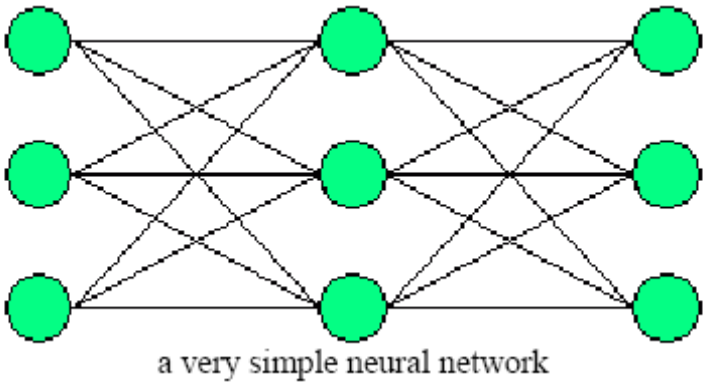


Figure 4.4 The simple neural network scheme.

In order to build this neural net with Joone, three Layer objects and two Synapse objects are created:

```
SigmoidLayer layer1 = new SigmoidLayer();
SigmoidLayer layer2 = new SigmoidLayer();
SigmoidLayer layer3 = new SygmoidLayer();
FullSynapse synapse1 = new FullSynapse();
FullSynapse synapse2 = new FullSynapse();
```

Then we complete the net connecting the three layers with the synapses:

```
layer1.addOutputSynapse(synapse1);
layer2.addInputSynapse(synapse1);
layer2.addOutputSynapse(synapse2);
layer3.addInputSynapse(synapse2);
```

Here you can see, each synapse is the output synapse of a layer and the input synapse of the next layer in the net. The general properties of Joone Neural Net package is listed as follows.

- The Joone's framework is built with a modular architecture: the 'core engine' is separated from the visual interface and permits easily to implement any new application based on it.
- Joone is portable, being written in 100% pure Java. It can run in any environment, from big multiprocessor machines to small palmtop devices.
- The neural networks based on Joone are usable stand-alone (separated from the framework that has created or trained them).
- The Joone's based neural networks can be transported using common protocols (like http or ftp) to run on remote machines
- The framework is expandable with more components to implement new learning algorithms or new architectures.
- With Joone, it's possible to implement any kind of optimization; there are two main methods to find the best solution to a given problem (i.e. to find the best neural network): local optimization and global optimization techniques. The local

optimization is obtained applying some 'internal' mechanism (the most famous is the momentum), the global optimization, instead, try to find the best solution applying some external technique to select the best performing NN among a predefined group of NNs (like genetic algorithms). Both are implemented with Joone, and many new optimization techniques can be experimented thanks to its expansibility.

- Joone's core engine is based on a multithreaded engine, capable to scale using all the computing resources available.
- Joone provides the professional users with a distributed environment to train many neural networks in parallel on several machines.
- Joone is freely usable. Its license is the Lesser General Public License (LGPL).

4.4 Software Specification

Agent Roles and Fuzzy Controller have been fully implemented in Java software development platform. Agents and their tasks have been coded using Jade libraries. Screen simulation parts have been done by using Java Swing Component. For neural network studies, Joone API has been integrated to the simulator program

Communication between agents is handled by a service provided by Jade. It’s called Yellow Page Service. All agents register themselves to this service at startup. Depending on their names, the sender agent of a message finds the address of the receiver and then sends it to this receiver. The implemented simulation software is designed for running on a single computer. However, Jade has support to send messages over network.

Agents are hierarchically constructed in this model. The some part of the hierarchical relations is kept in MS Access 2003 database tables (see Appendix C, LAgents and ISAgents tables). The other part of the relations is given at startup batch file. A sample batch runner is as follows:

"java jade.Boot -nomtp R-A:JadeTestAgent.RoadAgent (Road-A Light-2 0) R-B:JadeTestAgent.RoadAgent (Road-B Light-1 1) L-1:JadeTestAgent.LightAgent (Light-1 Junction-1) L-2:JadeTestAgent.LightAgent (Light-2 Junction-1) J-1:JadeTestAgent.JunctionAgent (Junction-1 Intersection-1) R-C:JadeTestAgent.RoadAgent (Road-C Light-4 2) R-D:JadeTestAgent.RoadAgent (Road-D Light-4 3) R-E:JadeTestAgent.RoadAgent (Road-E Light-3 4) L-3:JadeTestAgent.LightAgent (Light-3 Junction-2) L-4:JadeTestAgent.LightAgent (Light-4 Junction-2) J-2:JadeTestAgent.JunctionAgent (Junction-2 Intersection-1) IS-1:JadeTestAgent.IntersectionAgent (Intersection-1 Sector-1) I-Demo:JadeTestAgent.ISDemoSimulator (SIMULATOR)"

The description of the startup text is given below:

"java" addresses the java run time starter.

"Jade.Boot" addresses the run-time start parameter of Jade Agent platform.

"mtp" addresses external Messsage Transport Protocol activation for communication between agents. If "nomtp" is given, only the default settings defined in jade main container is used.

"AAA:BBB.CCC (X Y Z)" can be parsed in the following order:

AAA: refers to the agent name recorded in page service

BBB: the name of the namespace where the agent resides.

CCC: The software module name of the agent

X, Y, Z: refers to the arguments for Agent startup.

The general view of the class diagram implemented in this project is given in the following figure (Figure 4.5).

As it is seen in the picture, there are 5 types of agents defined. Each Agent in Jade environment represents a separate process and each process has a type of sub thread that is called as Behavior.

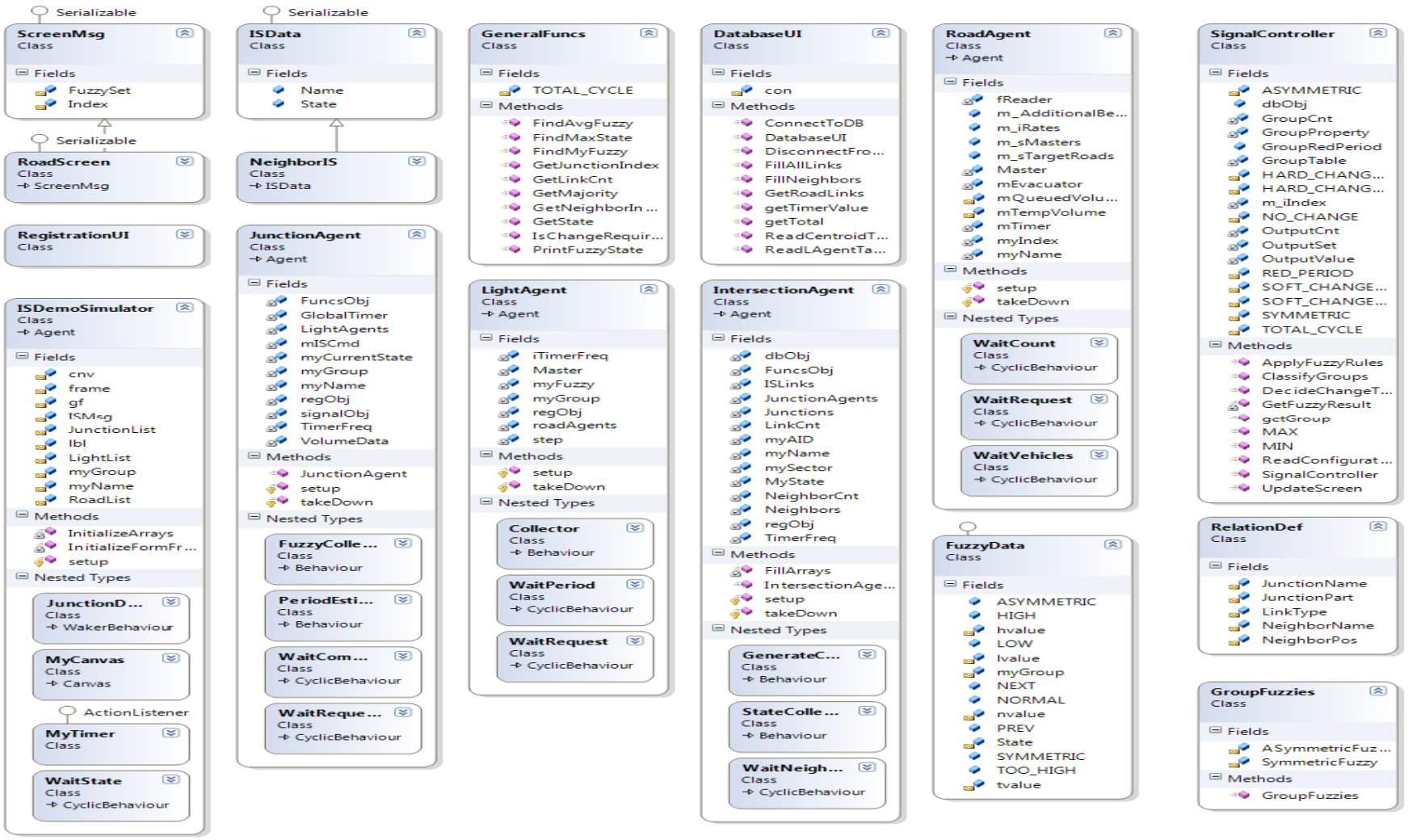
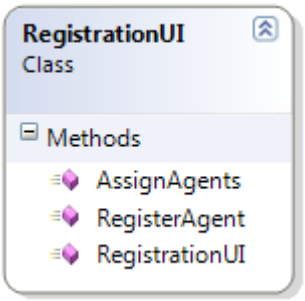


Figure 4.5. The class diagram of the implemented software simulation

The descriptions of some significant classes are given below:

RegistrationUI: This class is used to register agents against Jade Yellow Page Service. Its layout is given below.



ISDemoSimulator: This class is used to manage simulator screen activities. It is basically a thread. However, it is implemented b Jade library utilities. That's why it seems Agent. Another property of this class is that for each intersection topology, it should be newly designed.

ISData: This class is used to send intersection state information to neighbors.

JunctionAgent: This class is used to manage Junction Agent behaviors.

LightAgent: This class is used to manage Light Agent behaviors.

RoadAgent: This class is used to manage Road Agent behaviors.

IntersectionAgent: This class is used to manage Intersection Agent behaviors.

GeneralFuncs: This class contains some common functions used by agents. Some functions are related with fuzzy controller estimations.

DatabaseUI: This class is a collection of functions to access database and its tables. It is used by all agent modules.

FuzzyData: This class represents fuzzy input set sent from Road Agents to Light Agents and then Light Agents to Junction Agents.

SignalController: This class is used by the Junction Agent to control the estimation of new light period. Fuzzy Controller and De-fuzzification cycles are implemented in this class.

RelationDef: This class is used by the Intersection Agent to keep relations between internal junctions and junction to neighbor intersections. The data kept in this table is evaluated by the Reasoning engine integrated to the Junction Agent.

GroupFuzzies: This class is used by the Junction Agent to forward it into Fuzzy Controller. Junction Agent uses this structure as input to the fuzzy controller. Each group may refer to a Light Agent in a junction topology. If some Light Agents work at the same splits and offsets, they become the member of the same group and the group fuzzy values are estimated using average function.

In order to give a view of the programming design, the code module of the Light Agent that is written by Java and Jade is given in Appendix E.

4.5 Traffic Simulators

After the system is fully coded in software, two types of requirement have been emerged: Sample input data for road lane densities and displaying intersection work flow cycle on a screen simulator.

A simulator research has been done for a couple of months. It was seen that to find a general simulator product that meets all requirements above will be not easy. If the requirements are evaluated one by one;

Sample input density for road lanes: In order to overcome this issue, TSIS (Traffic Software Integrated System) simulator utility is taken under consideration. With this product an intersection model can be built and then after interpreting the model, the sample input data for each road lane in constructed topology can be obtained. TSIS simulator has a visualization component too. However, it is not open to external world and can not be directly used from development environment (ex: Java programs). More information about TSIS is given later.

Screen Simulator Demo: An intersection monitor module referencing to the Area Agent has been fully coded and implemented in the system. The programmer should change the style of the monitor for each target intersection or area environment. Therefore, it requires a programming effort to display new scenarios.

4.5.1 TSIS Simulator

The Traffic Software Integrated System (TSIS) is an integrated suite of traffic models for input development (TRAFED), simulation (CORSIM), and animation (TRAFVU). The suite is somewhat analogous to Microsoft Office that integrates a word processor, presentation, and spreadsheet. TSIS is widely recognized as one of the most successful analysis tools supported by FHWA (Federal Highway Administration). With the advent of TSIS, several tools were integrated into a common Windows interface that provides the user with a familiar look-and-feel compared to standard Windows applications.

The advantages of operating CORSIM within the TSIS environment include an intuitive, user-friendly graphical interface; scrollable screen input; better memory management; and on-line context-sensitive help that encompasses the TSIS, TRAFED, TRAFVU and CORSIM Users Guides. TSIS is designed to support CORSIM simulator, its input processor (TRAFED) and output processor (TRAFVU). The user can however extend TSIS functionality by adding other traffic engineering or analysis software tools that have been designed to conform to TSIS traffic tool interface (Owen, Zhang, Rao & McHale, 2000).

In this project, TSIS may be used for two purposes. The first one is sample data collection. TSIS provides detector installations on road links. By counting vehicles or densities, data sampling can be done on time period basis. After each complete simulation, TSIS generates an output file and this file includes detector occupation results. Manually extracting these results into related road lane files may provide a way to test our software with real data solve input sample data requirement problem.

The second usage type is to make animation and directing this animation by an external program (Win32 compiled DLL). TSIS provides a mechanism by which an external application can interface directly with the CORSIM simulation tool. This type of application has become known as a CORSIM run-time extension (RTE). The original run-time extensions were tailored for signal timing studies. However, the concept has been expanded to support freeway monitoring, incident detection and ramp metering run-time extension packages.

4.5.2 TSIS in Detail

A sample Intersection edited in TSIS (TRAFED) is given below.

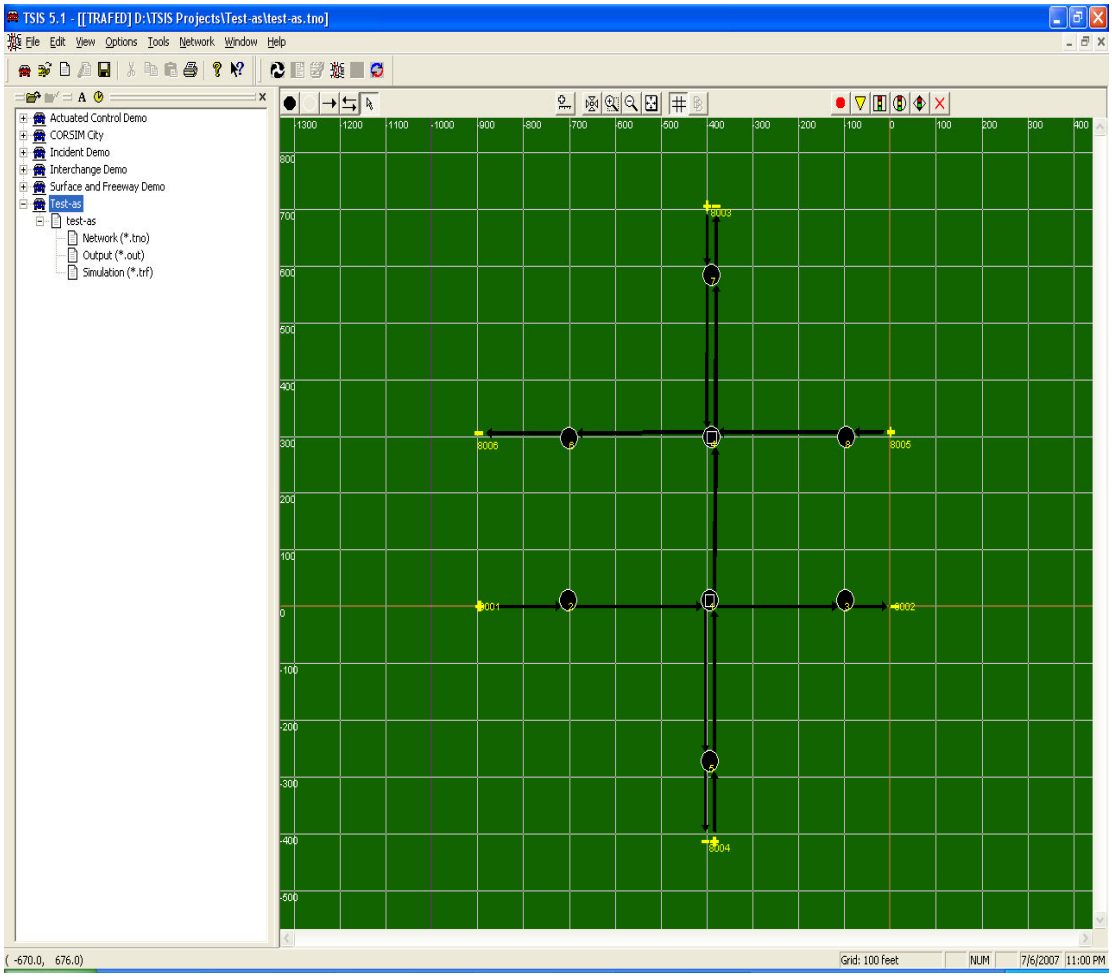


Figure 4.6 A sample TSIS traffic network editor view (TRAFED).

For input sampling, detectors are set on target road directions. A detector installation and configuration window is given below.

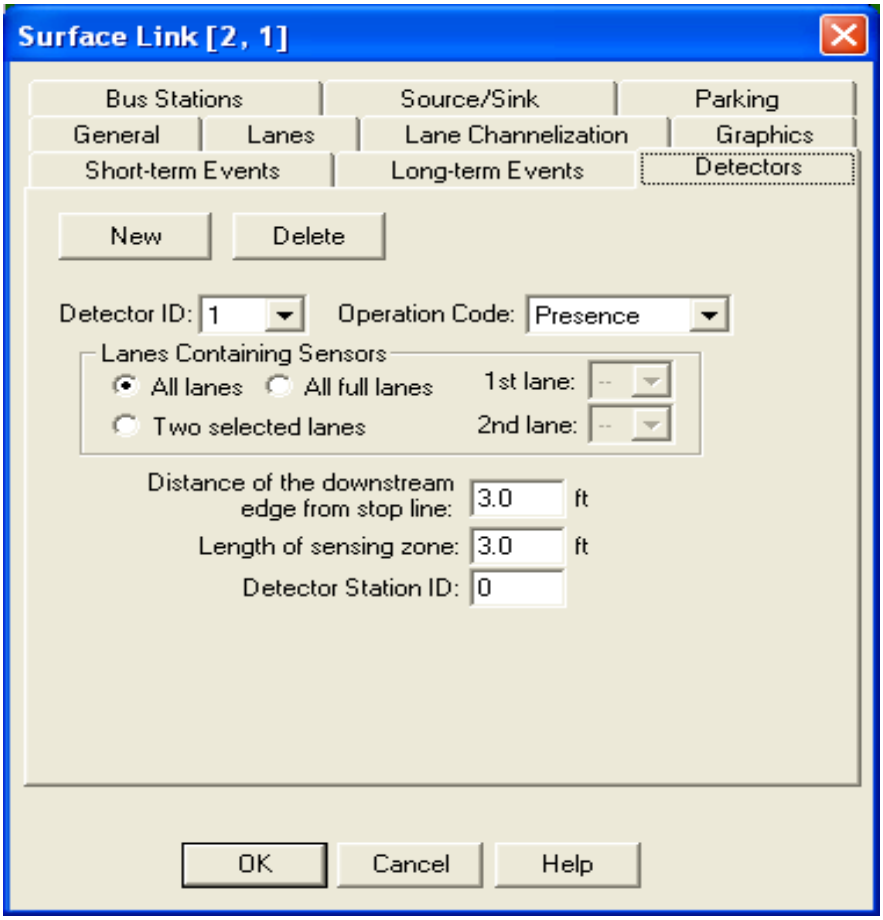


Figure 4.7 The detector install and setup page of TSIS.

The traffic light controllers and incoming road lane directions are set using the following dialogs (Figure 4.8, Figure 4.9).

Some tests have been done using the detector outputs of TSIS. However, it didn't satisfy the expectations because there are lots of parameters in TSIS environment. Each one should be precisely defined to construct the required intersection topology and get realistic results.

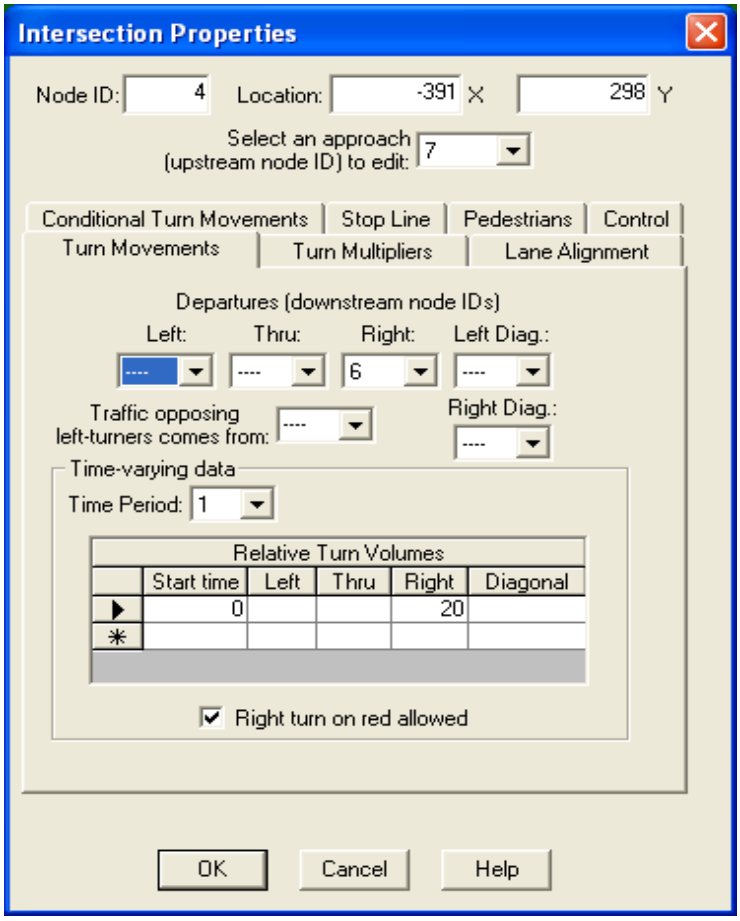


Figure 4.8 The TSIS intersection control parameters dialog.

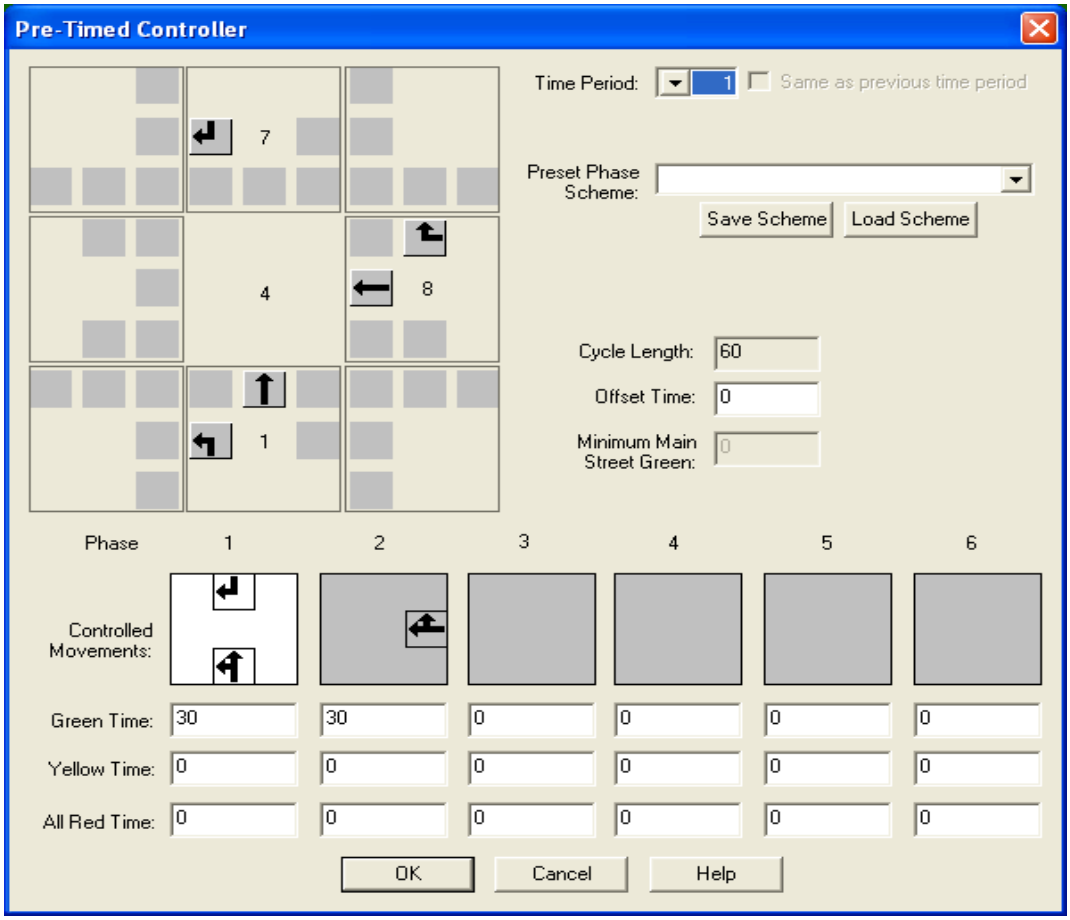


Figure 4.9 The TSIS traffic light controller configuration pages.

At the end of each simulation time period (set by the user before the simulation), the following detector output type is listed in out file.

AT A SELECTED TIME-PERIOD

DETECTOR TYPE	APP LINK	VEHICLE COUNT	CUMULATIVE TIME (SEC)	OCCUPANCY (%)**	SPEED (MPH)
PRESENCE	(2, 1)	22	44.3	73.8	14.5
PRESENCE	(5, 1)	8	47.1	78.5	14.1
PRESENCE	(7, 4)	17	41.2	68.7	9.7
PRESENCE	(8, 4)	29	50.7	84.5	16.0
PRESENCE	(1, 4)	23	49.6	82.7	13.1

** OCCUPANCY IS CALCULATED AS (CUMULATIVE TIME) / (TIME SINCE BEGINNING OF SIMULATION)

CHAPTER SIX
CONCLUSIONS

The general objective of this project research is to establish a know-how regarding the applicability of intelligent systems in traffic lights management and control and then the final goal is to develop a system that can effectively react upon versatile road and intersection traffic patterns. The system (can be called “smart traffic lights management“) is capable of optimizing and adjusting traffic lights, therefore improving vehicular throughput and minimize delays.

The target controller model has been fully implemented and simulated against the pre-timed controllers. With all this completed study, it was shown that the proposed solution model generates realistic and optimal schedules for any selected road traffic network.

The proposed solution is quite different from many other adaptive models in terms of the independent nature of the intersection topologies, neuro-fuzzy approach embedded into multi-role Agents, and global state evaluations that are taken into the decision mechanisms of the local level agents.

Analyzing the obtained results, it is seen that the proposed adaptive traffic controller solves the congestion problems efficiently. It boosts the performance specifically when one road has constant vehicle volume and the other road has a continuously changing vehicle curve, either in increasing or decreasing mode. If the traffic pattern changes occur sharply and very frequently, the new controller can’t handle the traffic and in those cases it shows the characteristics of the pre-timed controllers.

It seems that the controller could be very productive if one road has low and the other one has high densities at micro control (basic junction) level. In addition to this property, it is seen that when the controller reaches the limits of the signalization periods, it stays at that level until the other road direction gets congested.

Adaptive Traffic Controllers are not a global unique solution for all traffic problems. Their primary purpose is to regulate the unexpected vehicle flow very efficiently. If traffic at a predefined location is predictable, using fixed pre-timed controllers or their derivatives (green wave etc.) will be more satisfactory than Adaptive ones. Therefore, our newly designed controller should be tested and used for intersections that have non-uniform traffic patterns.

As a future study, some enhancements could be applied to the solution model. Neural Network internals are well defined and integrated to the Intersection Agent. However, it still requires more training data to give accurate and reliable results. Training data and training cycles are fundamental problems of the neural network applications. In order to eliminate it, an unsupervised neural-net (Dean, *et al.*, 1995, p. 183) might be preferred.

Another enhancement is possible through comparisons with other adaptive systems. This project focused on the comparisons with pre-timed controllers. In order to compare the results of the proposed model with other adaptive systems, other popular simulation tools such as TSIS (Traffic Software Integrated System) simulation program, see (Owen, *et al.,* 2000), that contains CORSIM traffic simulator tool might be employed. Thus, the strength of the proposed solution might be better appreciated.

Moreover, the proposed controller might be tested with real world intersection data. Nowadays, many traffic radars named RTMS (Real Time Monitoring System) are already installed on roads and junctions by the municipalities and private organizations. Requesting these data for academic purposes and evaluating with our proposed controller would be productive to see the performance.

REFERENCES

Abdulhai, B., Pringle, R., & Karakoulas, G., J. (2003). *ASCE Journal of Transportation Engineering*, Volume 129, Number 3, 278-285.

Athmaraman, N., & Soundararajan, S. (2005). Adaptive Predictive Traffic Timer Algorithm, *Proceedings of the 2005 Mid-Continent Transportation Research Symposium*, AN: 01004337, ITS Section.

Bellifemine, F., Caire, G., Trucco, T., & Rimassa, G. (June 18, 2007). *Jade Programmers Guide v3.5*. Retrieved December 15, 2007, from http://jade.tilab.com/doc/index.html

Bigus, J. P., & Bigus, J. (1998). *Constructing Intelligent Agents with Java: A Programmers Guide to Smarter Applications*. John Wiley & Sons Inc.

Brule, J.F. (April 27, 1985). *Fuzzy Systems, A Tutorial*. Retrieved July 24, 2005 from http://www.jimbrule.com/fuzzytutorial.html

Camurri, M., & Mamei, M. (2006). Urban Traffic Control with Co-Fields. *The Third International Workshop on Environments for Multi Agent Systems - EMAS 2006*.

Chiu, S., & Chand, S. (1999). Adaptive Traffic Signal Control Using Fuzzy Logic, *Proceedings FUZZ-IEEE'99,* 1371-1376.

Cools, S., Gershenson, C., & D'Hooge, B. (2005). Self-organizing traffic lights: A realistic simulation, *arXiv:nlin/0411066v2[nlin.AO]*

Cunningham, R., & Dowling, J. (2006). Self-Optimization in a Next Generation Urban Traffic Control Environment, *ERCIM News - Special: Emergent Computing*, 2006, vol. 64, 55-56

Dean, T., Allen, J., & Aloimonos, Y. (1995). *Artificial Intelligence Theory and Practice*, New York: The Benjamin/Cummings Publishing Company, Inc.

Deitel, H.M., & Deitel, P. J. (1998). JAVA How to Program (Second Edition), Prentice Hall.

Dresner, K, & Stone, P. (2006). Multi Agent Traffic Management: Opportunities for Multi Agent Learning, *SpringerLink Book: Learning and Adaption in Multi-Agent Systems, Volume 3898/2006, 129-138*.

Erciyes, K., & Sahan A. (2004). A Real-time Total Order Multicast Protocol. *4th International Conference on Computational Science, ICCS'2004.* SpringerLink, Volume 3036, 357-364

Findler, N., & Stapp, J. (1992). A distributed approach to optimized control of street traffic signals, *Journal of Transportation Engineering*, vol.118: No.1, 99-110.

Flanagan, D. (1999). *Java in a Nutshell, A Desktop Quick Reference* (Third Edition), USA: O'Reilly.

Gabric, T., Howden, N., Norling, E., Tidhar, G., & Sonenberg, L. (1994). Multi Agent Design of a Traffic Flow Control System, *Technical Report 94/24, Department of Computer Science*, *University of Melbourne*

Huang, D., Huang, W. (2003). Optimization of Traffic Lights at Crossroads, *International Journal of Modern Physics C*, vol. 14, Issue 05, 539-548

Lee, J., Lee, K., Seong, K., Kim, C., & Lee-Kwang, H. (1995). Traffic Control of Intersection Group Based On Fuzzy Logic, *Int. Conf. 6th IFSA '95, Sao Paulo*, 465 – 468

Lin S., & Chen D. (2005). Apply Adaptive and Cooperative multi-agent system to urban traffic signal control, *IEICE Trans Inf & Syst.*, E88-D: 119-126

Liu, Z. (2007). A Survey of Intelligence Methods in Urban Traffic Signal Control. *IJCSNS International Journal of Computer Science and Network Security*, vol.7 No.7, 105-112

Liu H.X., & Oh J.S. (2001). On-line Traffic Signal Control Scheme with Real-time Delay Estimation Technology, *Journal of Eastern Asia Society for Transportation Studies (EASTS)*, 4(4): 107-119

Marrone P. (February 3, 2005). *The Joone (Java Object Oriented Neural Engine) Complete Guide*. Retrieved January 22, 2006, from http://www.joone.org

Nikraz, M., Caire, G., & Bahri, P.A. (2006). A Methodology for the Analysis and Design of Multi-Agent Systems using JADE. *International Journal of Computer Systems Science & Engineering*. Special Section: Software Engineering for Multi-Agent Systems.

Owen L., Zhang Y., Rao L., & McHale, G. (2000). Traffic Flow Simulation using Corsim, *Proceedings of the 2000 Winter Simulation Conference*, 1143-1147

Pearson, R. (November 1, 2001). *ITS Traffic Signal Control*. Retrieved May 15, 2005, from *http://www.calccit.org/itsdecision/serv_and_tech/ Traffic_signal_control/ traffsigrep_print.htm*

Posio, J. (June 2003). Role of Intelligent Techniques in Transport Management – A Survey, *European Network on Intelligent Technologies for Smart Adaptive Systems (EUNITE) Roadmap*, *University of Oulu, Finland*

Rich, E., & Knight, K. (1991) *Artificial Intelligence* (International Edition), Singapore: McGraw Hill

Roozemond, D.A., & Rogier, J.L.H. (2000). Agent controlled traffic lights, *ESIT-2000 Aachen*, 77-81

Shen W., Norrie, D.H., Barthes, J. (2000). *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing* (First Edition). USA: CRC

Stergiou, C., & Siganos D. (April 17, 1987). *Neural Networks Course Guide*. Retrieved March 17, 2006 from http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

Taale, H., Back, T., Preuß, M., Eiben, A.E., De Graaf, J.M., & Schippers, C.A. (1998). Optimizing traffic light controllers by means of evolutionary algorithms, *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing,* Verlag in series H

Tanenbaum, A.S., & Van Steen, M. (2002). Distributed Systems: Principles and Paradigms, New Jersey: Prentice Hall

Tavladakis, K., & Voulgaris, N.C. (1999). Development of An Autonomous Adaptive Traffic Control System, *European Symposium on Intelligent Techniques ESIT'99*. Traffic W AB-03

Thorpe, T.L. (1997). Vehicle Traffic Light Control Using SARSA. *Masters Thesis, Department of Computer Science*, Colorado State University

Van Katwijk, R.T., Van Koningsbruggen, P., De Schutter, B., & Hellendoorn J. (2005). Test bed for multiagent control systems in road traffic management, *Transportation Research Record: Journal of the Transportation Research Board*, no. 1910**,** 108-115

Vega, J., R., P. (October 20, 2004). *ECE210 Statics Course Guide*, Retrieved October 16, 2006, from http://math.la.asu.edu/~rpacheco/COURSES/ECE210/ece210/Statics7.doc

Weiss G., (Ed.). (1999). *Multi Agent Systems: An Introduction to Distributed Artificial Intelligence.* Cambridge: The MIT Press

Wiering, M., van Veenen, J., Vreeken, J., & Koopman, A. (2004). Intelligent Traffic Light Control, *Technical Report UU-CS-2004-029,* Utrecht University, The Netherlands

APPENDICES

Appendix A - Input Fuzzy Set Generation

Input fuzzy set data structure that is used by the Agents is given below.

```
STRUCT FUZZY SET
{
    Low_Value : 0<x<1
    Normal_Value: 0< x <1
    High_Value: 0< x <1
    TooHigh_Value: 0< x <1
    DominantFuzzySet: Low < x <Too High
}
```

The fuzzy object generated from this struct is filled by the Road Agent using the following rules:

```
CASE volume < = 25
    If (volume <= 16) then
        DominantFuzzySet = "Low";
    Else
        DominantFuzzySet = "Normal";
    Endif
    TooHigh_value = 0
   High_value = 0
   Normal_value = volume/25
   Low_value = ABS(volume-50)/50

CASE volume < = 50
   DominantFuzzySet = "Normal"
   TooHigh_value = 0
   High_value = (volume-25)/50
```

```
        Normal_value = (75-volume)/50
        Low_value = ABS(volume-50)/50

    CASE volume < = 75
         DominantFuzzySet = "High"
        TooHigh_value = (volume-50)/50
        High_value = (volume-25)/50
        Normal_value = (75-volume)/50
        Low_value = 0

    CASE volume <=100
         If (volume <= 83) then
                DominantFuzzySet = "High"
         Else   DominantFuzzySet = "Too_High"
         Endif
        TooHigh_value = (volume-50)/50
        High_value = ABS(volume-100)/25
        Normal_value = 0
         Low_value = 0
```

Appendix B - Maximization Function for Agent Fuzzy Sets

This function is applied in Light and Junction Agents to generate their own fuzzy set from member Road and Light Agent fuzzy sets correspondingly.

Maximization Function Algorithm:

```
MAX = LOW(0);
FOR (I = 1 to N)
        IF (LOW(I) > MAX) THEN MAX = LOW(I);
END FOR
LIGHT_FUZZY_LOW = MAX;
MAX = NORMAL(0);
```

```
FOR (I = 1 to N)
        IF (NORMAL(I) > MAX) THEN MAX = NORMAL(I);
END FOR
LIGHT_FUZZY_NORMAL = MAX;
MAX = HIGH(0);
FOR (I = 1 to N)
        IF (HIGH(I) > MAX) THEN MAX = HIGH(I);
END FOR
LIGHT_FUZZY_HIGH = MAX;
MAX = TOOHIGH(0);
FOR (I = 1 to N)
        IF (TOOHIGH(I) > MAX) THEN MAX = TOOHIGH(I);
END FOR
LIGHT_FUZZY_TOOHIGH = MAX;
```

With this evaluation, Light Agent produces a new fuzzy data set over tightly coupled road fuzzy sets.

Table A.1 Sample road-to-light agent fuzzy data process ccenario

	Road-A	Road-B	MAX(RoadA, RoadB) ➔ **Light Agent Fuzzy Set**	
Road Volume	47	67	-	**-**
LOW	0,06	0,00	0,06	**0,06**
NORMAL	0,56	0,16	0,56	**0,56**
HIGH	0,44	0,84	0,84	**0,84**
TOO-HIGH	0,00	0,34	0,34	**0,34**
Dominant Fuzzy Set	Normal	High	High	**High**

Appendix C - Database Graph for Configuration Tables

In this thesis, there is no historical data kept. However, for road, light, junction and intersection relations, a group of configuration tables are used. For simplicity, an MS Access database is preferred. The picture of the database and some explanations are given below.

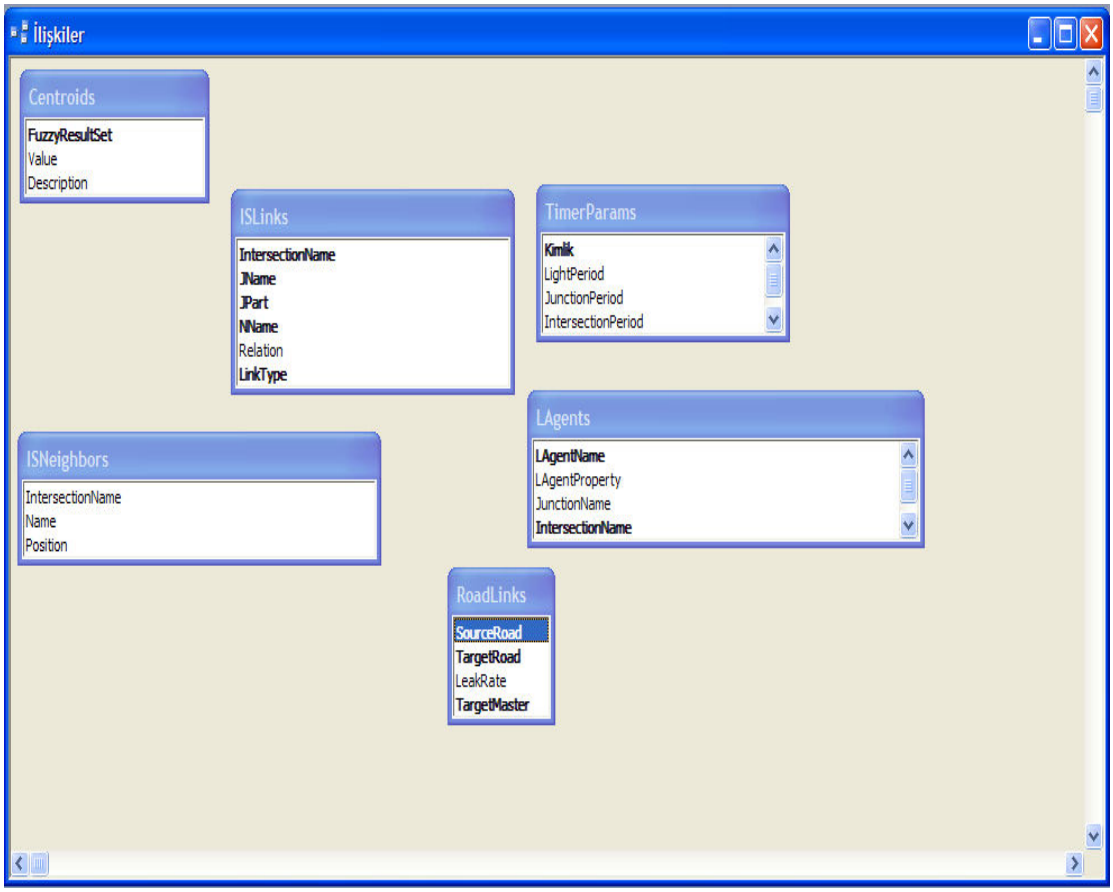


Figure A.1 The tables in configuration database.

- *Centroids:* keeps some values of de-fuzzification process
- *TimerParams:* keeps the values of agent timers for collecting sub-states.
- *ISNeighbors:* keeps the list of neighbors of intersection. Position shows the relation whether next or previous to this intersection
- *RoadLinks:* keeps the relations between incoming and outgoing road directions. The relations show that the leak rate that defines the vehicle volume level transferred from one road to the other one.

- *LAgents:* keeps the list and classification data of light agents in a junction. LAgentProperty shows the group: Group-Reference or Group-Opponent
- *ISLinks:* keeps the relation between junctions and neighbor intersections. (Which junction group is connected to which junction or neighbor intersection?) LinkType column shows the relation whether it is intersection or junction connection. RelationType column show the target object, whether it is Prev or Next. JPart column shows the junction group (reference or opponent), NPart column shows the adjacent junction group if the link type is junction. This table is open to change because of ambiguous relations between intersections and junctions.

Appendix D - Sample Traffic Network Scenarios

Scenario 1: The Simplest Junction

The implemented system was initially tested for a base junction scenario that consists of two complementary traffic lights and one junction and pre-selected data sets. The sample scenario schema is illustrated in Figure A.2.

Initial test configuration is as follows: Total Cycle is set to 60 seconds. Data collection frequencies are 30, 45, 60 seconds corresponding to Light, Junction and Intersection Agents. Road max capacity is assumed 100 vehicles and a vehicle can pass through the intersection in 1 second: 1 vehicle/sec

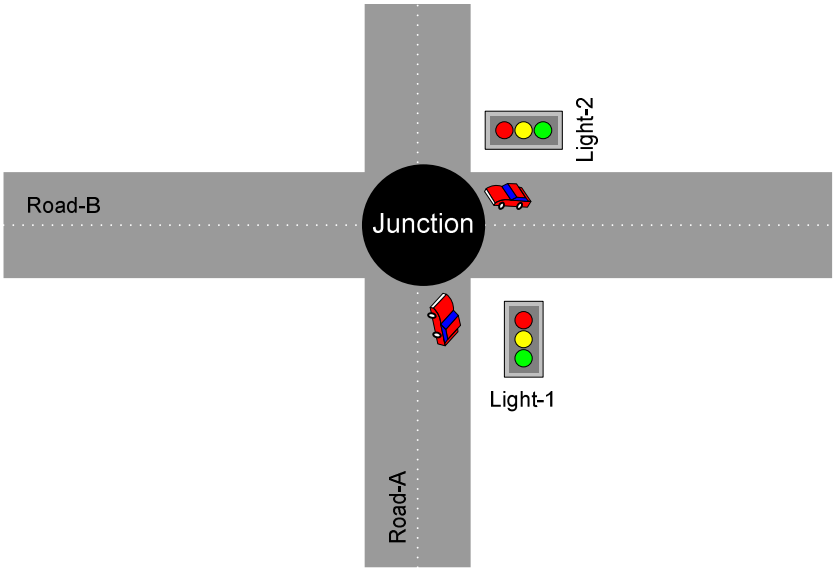


Figure A.2 The sample intersection graph for scenario 1.

Based on these settings and according to the given data sample sets in Table A.4 and Table A.5, two work schemes are presented here.

The first data set of Scenario 1 is given in Table A.2. In this table, Road-A has low vehicle input volume, other one is changing dynamically. The result graph (Figure A.3) shows that our adaptive light controller outperforms the fixed pre-timed light sharply just after Road-B input vehicle count has started to decrease.

Table A.2 The sample data set 1 and results of scenario 1

Time (min)	**Road-A**	**Road-B**	**Pre-timed, in wait**	**Actuated by our system, in wait**
1	10	10	0	0
2	10	20	0	0
3	10	30	0	0
4	10	40	10	10
5	10	50	30	28
6	10	60	60	51
7	10	50	80	51
8	10	40	90	41
9	10	30	90	21
10	10	20	**80**	**16**

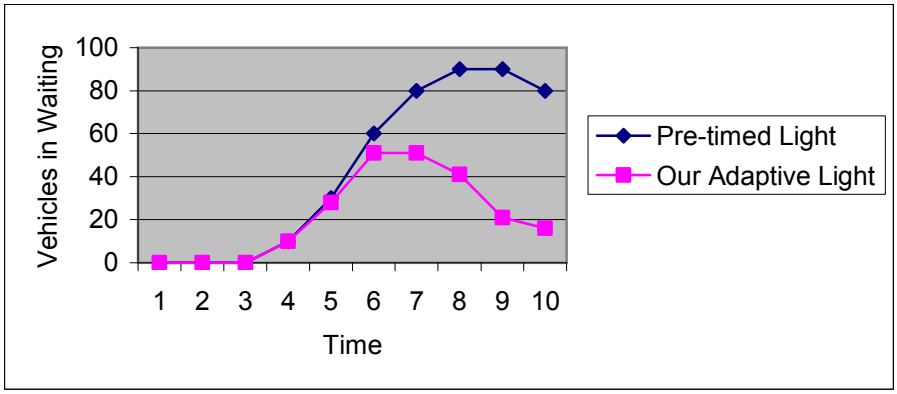


Figure A.3 The result graph of data set 1 execution.

The second data set for Scenario 1 is given in Table A.5. In this data set, while the Road-A has low vehicle input volume, the other one changes dynamically, and the result graph (Figure A.4) shows that our adaptive light outperforms the fixed pre-timed light again in the same manner.

Table A.3 The sample data set 2 and results of scenario 1

Time (min)	**Road-A**	**Road-B**	**Pre-timed, in wait**	**Actuated by our system, in wait**
1	10	60	30	30
2	10	50	50	49
3	10	40	60	38
4	10	30	60	18
5	10	20	60	8
6	10	10	60	12
7	10	20	60	16
8	10	30	60	20
9	10	40	70	22
10	10	50	**80**	**24**

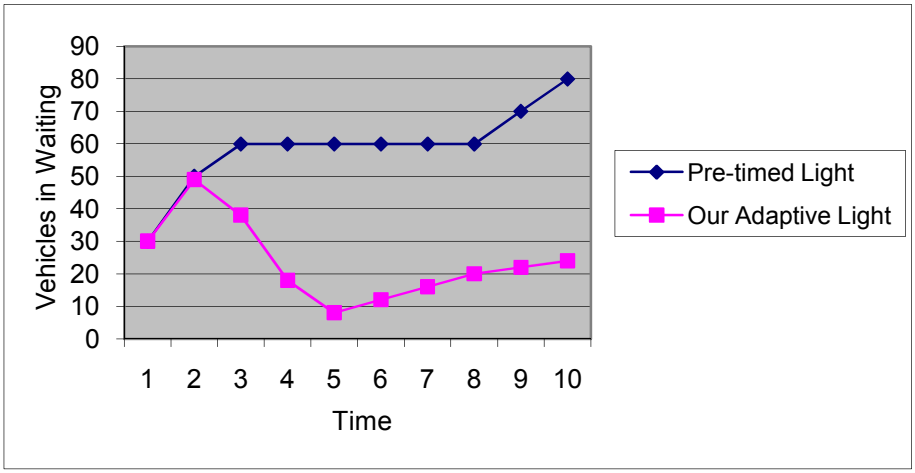


Figure A.4 The result graph of data set 2 execution.

The results show that for the basic controller scheme, the proposed controller boosts the performance when compared to the pre-time traffic controller. For low volumes, the adaptive controller shows almost the same performance. However, for high vehicle volumes, the new controller outperforms pre-timed controllers.

Scenario 2: An Intersection with No Neighbor and Two Junctions

The sample scenario is applied to the following figure given. According to this figure, the following agent structure is generated:

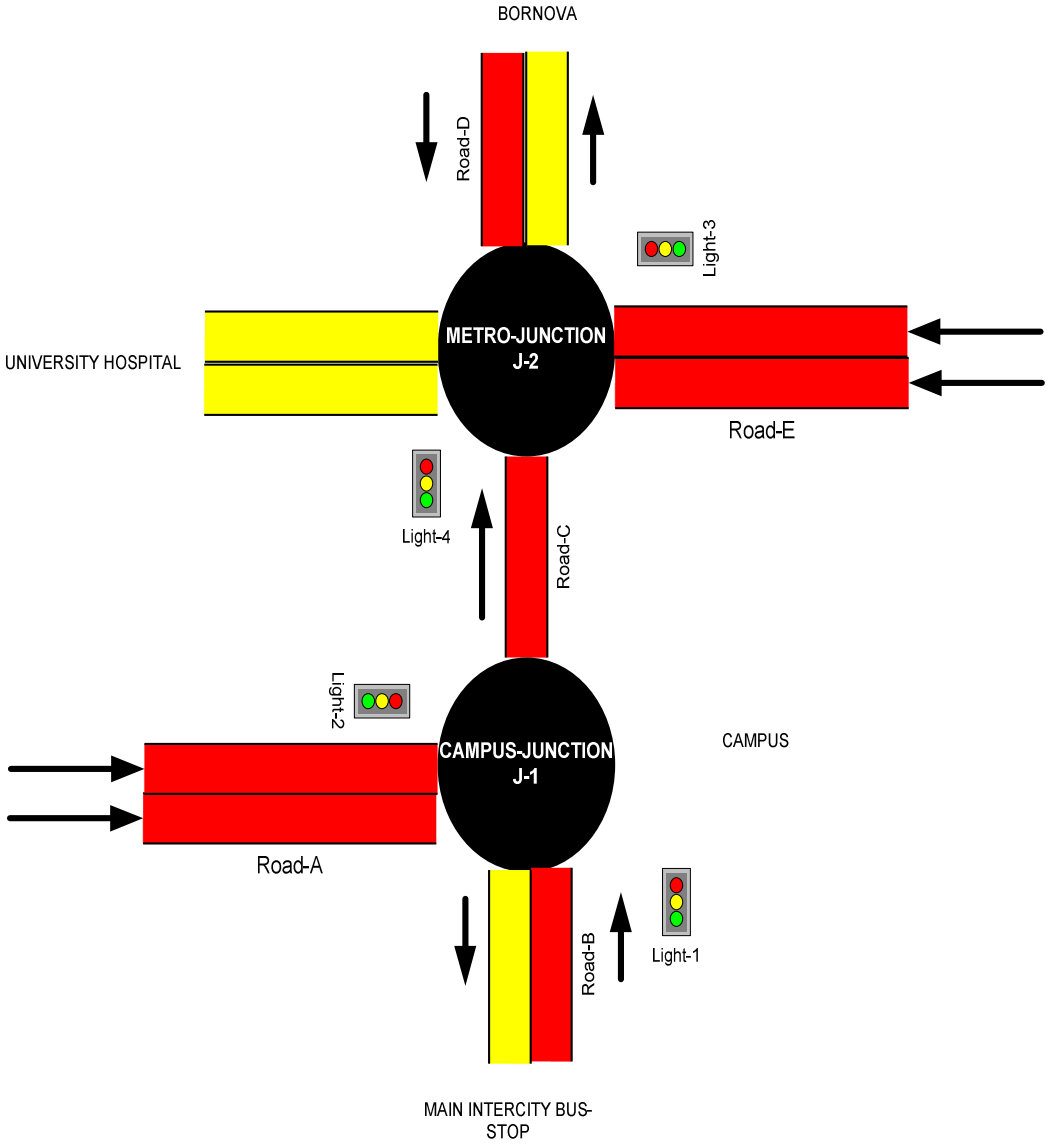


Figure A.5 The sample intersection 2 (road lanes with yellow color are out of scope).

1 Intersection Agent (top-controller)

2 Junction Agents (middle-level controller, J-1, J-2)

4 Light Agents (2 for J-1, 2 for J-2)

Light-1: J-1 Reference

Light-2: J-1 Opponent

Light-3: J-2 Reference

Light-4: J-2 Opponent

5 Road Agents (2 for J-1, 3 for J-2)

Road-A: Light-2

Road-B: Light-1

Road-C: Light-4

Road-D: Light-4

Road-E: Light-3

Data Collection period parameters:

- Light Agent collects states of road agents each 30 seconds
- Junction Agent collects states of light agents each 45 seconds
- Intersection Agent collects state of Junction agent each 60 seconds.

START

STEP-1: Each 30 seconds,

Road agents send their fuzzy sets to Light Agents (upon their request)

SAMPLE

Road-A: Volume➔40

Fuzzy_Set (LOW:0.2, **NORMAL:0.7**, HIGH:0.3, TOO_HIGH:0)

Dominant State ➔ Normal

Road-B: Volume➔30

Fuzzy_Set (LOW:0.4, **NORMAL:0.9**, HIGH:0.1 , TOO_HIGH:0)

Dominant State➔Normal

Road-C: Volume➔25

Fuzzy_Set (LOW:0.5 , **NORMAL:1.0**, HIGH: 0, TOO_HIGH: 0)

Dominant State➔Normal

Road-D: Volume➔80

Fuzzy_Set (LOW:0, NORMAL:0, **HIGH:0.8**, TOO_HIGH:0.6)

Dominant State➔High

Road-E: Volume-➔35

Fuzzy Set (LOW:0.3, **NORMAL:0.8**,HIGH:0.2, TOO_HIGH:0)

Dominant State➔Normal

STEP-2: Each 45 seconds,

Light Agents send their fuzzy sets to Junction agents (upon their request)

Light-1: = MAX(Road-B) = Road-B

Fuzzy_Set (LOW:0.4, **NORMAL:0.9**, HIGH:0.1 , TOO_HIGH:0)

Dominant State➔Normal

Light-2: = MAX(Road-A) = Road-A

Fuzzy_Set (LOW:0.2, **NORMAL:0.7**, HIGH:0.3, TOO_HIGH:0)

Dominant State➔Normal

Light-3: = MAX(Road-E) = Road-E

Fuzzy Set (LOW:0.3, **NORMAL:0.8**,HIGH:0.2, TOO_HIGH:0)

Dominant State➔Normal

Light-4: = MAX(Road-C,Road-D)

Fuzzy_Set (LOW:0.5, NORMAL:1.0, **HIGH:0.8**, TOO_HIGH:0.6)

Dominant State➔High

Junction-2:

Group Reference (Light-3), Dominant State➔**Normal**

Group Opponent (Light-4), Dominant State➔**High**

Launch Fuzzy Controller

Fuzzy Controller:

Reference-Set (Light-3)➔ (LOW:0.3, **NORMAL:0.8**,HIGH:0.2, TOO_HIGH:0)

Opponent-Set (Light-4)➔ (LOW:0.5, NORMAL:1.0, **HIGH:0.8**, TOO_HIGH:0.6)

Fuzzy Rules Evaluation

- Rule-1: MinVal1 = 0.3, Oval1 = 0
- Rule-2: MinVal2 = 0.3, Oval2 = 0
- Rule-3: MinVal3 = 0.5, Oval3 = 0
- Rule-4: MinVal4 = 0.8, Oval4 = 0
- Rule-5: MinVal5 = 0.2, Oval5 = 0
- Rule-6: MinVal6 = 0, Oval6 = 0
- Rule-7: MinVal7 = 0.3, Oval7 = 0.6
- Rule-8: MinVal8 = 0.3, Oval8 = 0.9
- Rule-9: MinVal9 = 0.8, Oval9 = 0.3
- Rule-10: MinVal10 = 0.6, Oval10 = 0.6

- Rule-11: MinVal11 = 0.2, Oval11 = -0.6
- Rule-12: MinVal12 = 0.2, Oval12 = -0.3
- Rule-13: MinVal13 = 0.2, Oval13 = 0.3
- Rule-14: MinVal14 = 0, Oval14 = -0.9
- Rule-15: MinVal15 = 0, Oval15 = -0.6
- Rule-16: MinVal16 = 0, Oval16 = -0.3

De-fuzzification➔Apply Centroid Principle

OTotal = 0.18+0.27+0.24+0.36-0.12-0.06+0.06 = 0.93

MTotal = 4.7

Change_Rate = 0.93 /4.7 = 0.19, %19 increase of red period of reference light group.

If Red of Light-3 was 45 sec, it becomes now 53 sec. And it means 53 seconds (up from 45 sec) green for Light -4 which is in High state

Neural-Net:

ISCommand➔0

Change_Fuzzy➔0.19 (based on reference group)

DayOfWeek➔XXX

Time➔XXXXX, XXXXXX

IsHoliday➔X

Output* 100 will be real red period value for reference light agents.

Junction-1:

Group Reference (Light-1), Dominant State➔**Normal**

Group Opponent (Light-2), Dominant State➔**Normal**

No Trouble Do Nothing (optional) or

Light-1: = MAX(Road-B) = Road-B

Fuzzy_Set (LOW:0.4, **NORMAL:0.9**, HIGH:0.1 , TOO_HIGH:0)

Dominant State➔Normal

Light-2: = MAX(Road-A) = Road-A

Fuzzy_Set (LOW:0.2, **NORMAL:0.7**, HIGH:0.3, TOO_HIGH:0)

Dominant State➔Normal

Rule Results:

0.2, 0.4, 0.2, 0.7, 0.1, 0, 0.3, 0, 0.3, 0, 0.1, 0.1, 0, 0, 0, 0

Output Set Centroids

0, 0, 0, 0, 0, 0, 0.6, 0.9, 0.3, 0.6, -0.6, -0.3, 0.3, -0.9, -0.6, -0.3

OTotal = 0.18 + 0 + 0.09 + 0 + -0.06 -0.03 + 0 + 0 + 0 + 0 = 0.18

MTotal = 2.4

Change_Rate = 0.18 /2.4 = **0.07, %7 increase of red period of reference light.**

STEP-3: Each 60 seconds,

ISLinks database table content is as given as follows.

Table A.4 Junction-to-junction, Junction-to-intersection links table

Junction-SubGroup	**Linked To**	**Relation Type**	**Neighbor**	**Position Type (according to local junction)**
J-2 Opponent	J-1	Local	-	Prev

Junction Agents send their fuzzy states to Intersection Agent (upon its request)

Junction-1:

State➔Normal

Reference➔Normal

Opponent➔Normal

Junction-2:

State➔High

Reference➔Normal

Opponent➔High

Now execute the Intersection Agent state reasoning algorithm:

- Local State is High, so call manage_localstate()
- There is no link to Junction-2, so skip process

If J-1 were high and J-2 were Normal,

- Local state is High, so call manage_localstate()

- There is a link found to Junction-1
- Send command to J-2 (%10 increase of reference red period), means that %10 increase of green period of J-2 Opponent. It causes less traffic to J-1.

There is a table below that shows several results according to the values of input Road Agents:

Table A.5 The Execution flow of the system for sample scenario 2

	Road Agents	**Fuzzy Flow**	**Fuzzy Output**	**Junctions**	**Reasoning at IS**
T-1	**Road-A: 90** **Road-B: 70** **Road-C: 50** **Road-D: 30** **Road-E: 10**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) High **Group Opponent (Light-2, Road A):** (Low:0, Normal:0, High:0.4, Too_High:0.8) Too_High **Junction-2** **Group Reference (Light-3, Road E):** (Low:0.8, Normal:0.4, High:0, Too_High:0) Low **Group Opponent (Light-4, Road C,D):** Road-C (Low:0, Normal:0.5, High:0.5, Too_High:0) Road-D (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) MAX = (Low:0.4, Normal:0.9, High:0.5, Too_High:0) Normal	Junction-1: MTotal = 2.2 OTotal = 0.21 **Change = 0.21/2.2 = %9 increase of reference red period.** Junction-2: MTotal = 2.9 OTotal = 0.42 **Change = %14 increase of reference red period or do nothing (optional)**	Junc-1: Too_High Junc-2: Normal Evaluation for Junc-1 **(Too_High-Normal) = 2*10** **Change[Prev,Opp]**	**%20 increase red period of reference for Junc-2.**
T-2	**Road-A: 70** **Road-B: 30** **Road-C: 10** **Road-D: 50** **Road-E: 90**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) Normal **Group Opponent (Light-2, Road A):** (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) High	Junction-1: MTotal = 2.6 OTotal = 1.11 **Change = 1.11/2.6 = %42 increase of reference red period.** Junction-2:	Junc-1: High Junc-2: Too_High Evaluation for Junc-2 **No Links to J-2**	**No command**

		Junction-2 **Group Reference (Light-3, Road E):** (Low:0, Normal:0, High:0.4, Too_High:0.8) Too_High **Group Opponent (Light-4, Road C,D):** Road-C (Low:0.8, Normal:0.4, High:0, Too_High:0) Road-D (Low:0, Normal:0.5, High:0.5, Too_High:0) MAX = (Low:0.8, Normal:0.5, High,:0.5 Too_High:0) Normal	MTotal = 3.0 OTotal = -1.53 **Change = %51 decrease of reference red period.**		
T-3	**Road-A: 50** **Road-B: 70** **Road-C: 30** **Road-D: 90** **Road-E: 10**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) High **Group Opponent (Light-2, Road A):** (Low:0, Normal:0.5, High:0.5, Too_High:0) Normal **Junction-2** **Group Reference (Light-3, Road E):** (Low:0.8, Normal:0.4, High:0, Too_High:0) Low **Group Opponent (Light-4, Road C,D):** Road-C (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) Road-D (Low:0, Normal:0, High:0.4, Too_High:0.8) MAX = (Low:0.4, Normal:0.9, High,:0.4 Too_High:0.8) Too_High	Junction-1: MTotal = 2.0 OTotal = -0.36 **Change = -0.36/2.0 = %18 decrease of reference red period.** Junction-2: MTotal = 4.0 OTotal = 1.32 **Change = %33 increase of reference red period**	Junc-1: High Junc-2: Too_High Evaluation for Junc-2 **No Links to J-2**	**No Command**
T-4	**Road-A: 10** **Road-B: 30** **Road-C: 50** **Road-D: 70** **Road-E: 90**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) Normal **Group Opponent (Light-2, Road A):** (Low:0.8, Normal:0.4, High:0,	Junction-1: MTotal = 2.2 OTotal = -0.09 **Change = -0.09/2.2 = %4 decrease of reference red**	Junc-1: Normal Junc-2: Too_High Evaluation for Junc-2 **No Links to J-2**	**No Command**

		Too_High:0) Low **Junction-2** **Group Reference (Light-3, Road E):** (Low:0, Normal:0, High:0.4, Too_High:0.8) Too_High **Group Opponent (Light-4, Road C,D):** Road-C (Low:0, Normal:0.5, High:0.5, Too_High:0) Road-D (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) MAX = (Low:0, Normal:0.5, High:0.9, Too_High:0.4) High	**period or do nothing (optional)** Junction-2: MTotal = 2.9 OTotal = -0.54 **Change = %18 decrease of reference red period**		
T-5	**Road-A: 90** **Road-B: 50** **Road-C: 10** **Road-D: 30** **Road-E: 70**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0, Normal:0.5, High:0.5, Too_High:0) Normal **Group Opponent (Light-2, Road A):** (Low:0, Normal:0, High:0.4, Too_High:0.8) Too_High **Junction-2** **Group Reference (Light-3, Road E):** (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) High **Group Opponent (Light-4, Road C,D):** Road-C (Low:0.8, Normal:0.4, High:0, Too_High:0) Road-D (Low:0.4,Normal:0.9,High:0.1,Too_Hig h: 0) MAX = (Low:0.8, Normal:0.9, High:0.1, Too_High:0) Normal	Junction-1: MTotal = 1.8 OTotal = 0.57 **Change = 0.57/1.8 = %31 increase of reference red period.** Junction-2: MTotal = 3.0 OTotal = -1.35 **Change = %45 decrease of reference red period**	Junc-1: Too_High Junc-2: High Evaluation for Junc-1 **(Too_High-High) = 1*10** **Change[Prev,Opp]**	**%10 increase red period of reference for Junc-2.**
T-6	**Road-A: 10** **Road-B: 90** **Road-C: 30** **Road-D: 70** **Road-E: 50**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0, Normal:0, High:0.4, Too_High:0.8) Too_High	Junction-1: MTotal = 2.0 OTotal = -1.32 **Change = -1.32/2.0 = %66**	Junc-1: Too_High Junc-2: High Evaluation for Junc-1	**%10 increase red period of reference for Junc-2.**

		Group Opponent (Light-2, Road A): (Low:0.8, Normal:0.4, High:0, Too_High:0) Low **Junction-2** **Group Reference (Light-3, Road E):** (Low:0, Normal:0.5, High:0.5, Too_High:0) Normal **Group Opponent (Light-4, Road C,D):** Road-C (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) Road-D (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) MAX = (Low:0.4, Normal:0.9, High:0.9, Too_High:0.4) High	**decrease of reference red period.** Junction-2: MTotal = 3.6 OTotal = 0.12 **Change = %3 increase of reference red period.**	**(Too_High-High) = 1*10 Change[Prev,Opp]**	
T-7	**Road-A: 30** **Road-B: 10** **Road-C: 70** **Road-D: 90** **Road-E: 50**	**Junction-1** **Group Reference (Light-1, Road B):** (Low:0.8, Normal:0.4, High:0, Too_High:0) Low **Group Opponent (Light-2, Road A):** (Low:0.4,Normal:0.9,High:0.1,Too_High: 0) Normal **Junction-2** **Group Reference (Light-3, Road E):** (Low:0, Normal:0.5, High:0.5, Too_High:0) Normal **Group Opponent (Light-4, Road C,D):** Road-C (Low:0,Normal:0.1,High:0.9,Too_High: 0.4) Road-D (Low:0, Normal:0, High:0.4, Too_High:0.8) MAX (Low:0, Normal:0.1, High:0.9, Too_High:0.8) High	Junction-1: MTotal = 2.2 OTotal = 0.09 **Change = 0.09/2.2 = %4 increase of reference red period or do nothing (optional)** Junction-2: MTotal = 2.2 OTotal = 0.57 **Change = %25 increase of reference red period.**	Junc-1: Normal Junc-2: High Evaluation for Junc-2 **No Links to J-2**	**No Command**

After the detailed evaluations, the following summary results table is generated:

Table A.6 Final results summary for sample scenario 2

Tests	Reference	Opponent	Junction Result	IS Result
T-0	**Light-1: Normal**	**Light-2: Normal**	**%7 increase red period of Reference (J-1)**	Local State is High, and No link to J-2, No Command
	Light-3: Normal	**Light-4: High**	**%19 increase red period of Reference (J-2)**	
T-1	**Light-1: High**	**Light-2: Too_High**	**%9 increase red period of Reference (J-1)**	Test-1, Local State is Too_High, Command➔%20 increase red period of reference for J-2.
	Light-3: Low	**Light-4: Normal**	**%14 increase red period of Reference (J-2)**	
T-2	**Light-1: Normal**	**Light-2: High**	**%42 increase red period of Reference (J-1)**	Test-2, Local State is Too_High, No Link to J-2, No Command
	Light-3: Too_High	**Light-4: Normal**	**%51 decrease red period of Reference (J-2)**	
T-3	**Light-1: High**	**Light-2: Normal**	**%18 decrease red period of Reference (J-1)**	Test-3, Local State is Too_High, No Link to J-2, No Command
	Light-3: Low	**Light-4: Too_High**	**%33 increase red period of Reference (J-2)**	
T-4	**Light-1: Normal**	**Light-2: Low**	**%4 decrease red period of Reference (J-1)**	Test-4, Local State is Too_High, No Link to J-2, No Command
	Light-3: Too_High	**Light-4: High**	**%18 decrease red period of Reference (J-2)**	
T-5	**Light-1: Normal**	**Light-2: Too_High**	**%31 increase red period of Reference (J-1)**	Test-5, Local State is Too_High, Command➔%10 increase red period of reference for J-2.
	Light-3: High	**Light-4: Normal**	**%45 decrease red period of Reference (J-2)**	
T-6	**Light-1: Too_High**	**Light-2: Low**	**%66 decrease red period of Reference (J-1)**	Test-6, Local State is Too_High, Command➔%10 increase red period of reference for J-2.
	Light-3: Normal	**Light-4: High**	**%3 increase red period of Reference (J-2)**	
T-7	**Light-1: Low**	**Light-2: Normal**	**%4 increase red period of Reference (J-2)**	Test-7, Local State is High, No Link to J-2, No Command
	Light-3: Normal	**Light-4: High**	**%25 increase red period of Reference (J-2)**	

A part from the step by step scenario work given above, for the same intersection scenario another test has been realized. The sample data set for the execution is as follows. Note that the all roads have a steady flow and Road-C is an internal road direction that's why it's labeled with a T. The values given in parentheses specify the

transferred vehicle count for pre-timed controller scheme. (The Leak rate parameters: %60 of Road-B, %70 of Road-A). In Table A.8, the numbers of queued vehicles are listed according to the tested environments (Pre-timed controller, TSIS Simulator environment, the proposed model)

Table A.7 The another data set for sample scenario 2

Time (unit)	Road-A	Road-B	Road-C	Road-D	Road-E
1	20	40	0	40	20
2	20	40	0 + 38(T)	40	20
3	20	40	0 + 38(T)	40	20
4	20	40	0 + 38(T)	40	20
5	20	40	0 + 38(T)	40	20
6	20	40	0 + 38(T)	40	20

Table A.8 The queued vehicles after the execution of data set

Time (unit)	Road-A	Road-B	Road-C	Road-D	Road-E	Total
New Solution	0	30	0	60	21	111
Pre-timed Controller	0	60	38	60	0	158
Corsim Controller	4	7	39	25	6	81

As you see from the table, the new solution outperforms pre-timed controller, however, the Corsim controller (TSIS-integrated simulator) performs better than the solution model. With high probability, it occurred because of the wrong settings of the simulator environment because the Corsim simulator had been configured based on pre-timed controller model and it should have approximated to the pre-timed controller.

Scenario 3: SOK MARKET Intersection – Bornova

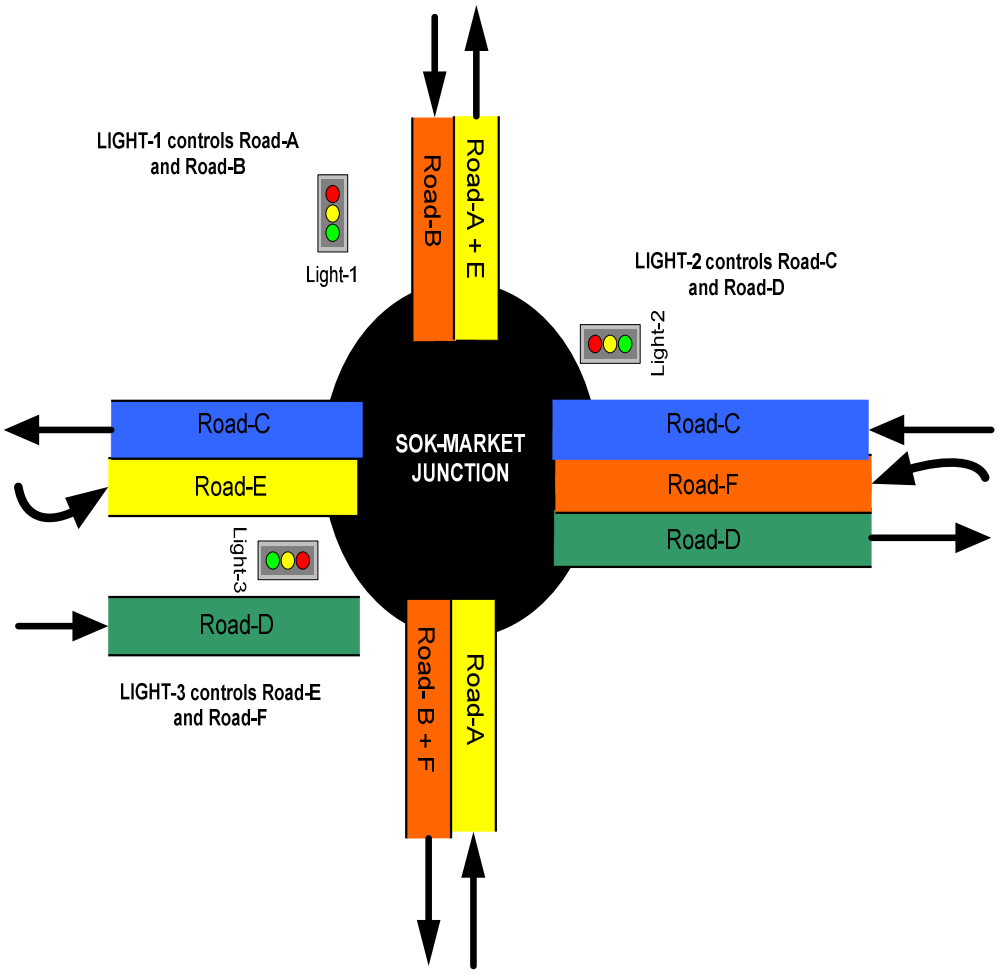


Figure A.6 The sample triple junction scenario (yellow lanes are out of scope).

According to this figure, the following agent structure is generated:

1 Intersection Agent (IS-1, top-controller, has no sense in this configuration)

1 Junction Agent (middle-level controller, J-1)

3 Light Agents (L-1, L-2, L-3, each one controls a road-cross point)

6 Road Agents

Road-A: Light-1

Road-B: Light-1

Road-C: Light-2

Road-D: Light-2

Road-E: Light-3

Road-F: Light-3

Data Collection period parameters:

- Light Agent collects states of road agents each 30 seconds
- Junction Agent collects states of light agents each 45 seconds
- Intersection Agent collects state of Junction agent each 60 seconds.
- Total Cycle for intersection is 90 sec. Initially each Light agent launches 30sec green and 60 sec red signal.
- The vehicle count above 100 in queues is ignored, because road capacities are assumed as 100.

The first sample data table is as follows:

Table A.9 Input vehicle counts data set I for sample scenario 3

	VEHICLE COUNTS INCOMING					
	Road-A	Road-B	Road-C	Road-D	Road-E	Road-F
Time-1	20	40	20	60	40	10
Time-2	20	40	20	55	40	15
Time-3	20	40	20	50	40	20
Time-4	20	40	20	45	40	25
Time-5	20	40	20	40	40	30
Time-6	20	40	20	35	40	35
Time-7	20	40	20	30	40	45
Time-8	20	40	20	25	40	50
Time-9	20	40	20	15	40	55
Time-10	20	40	20	10	40	60

By processing this data set, the following result graph is obtained. As you see from the graph, at the end of simulation, our system outperforms fixed-time split system and moreover, it uniformly distributes the waiting vehicles on roads.

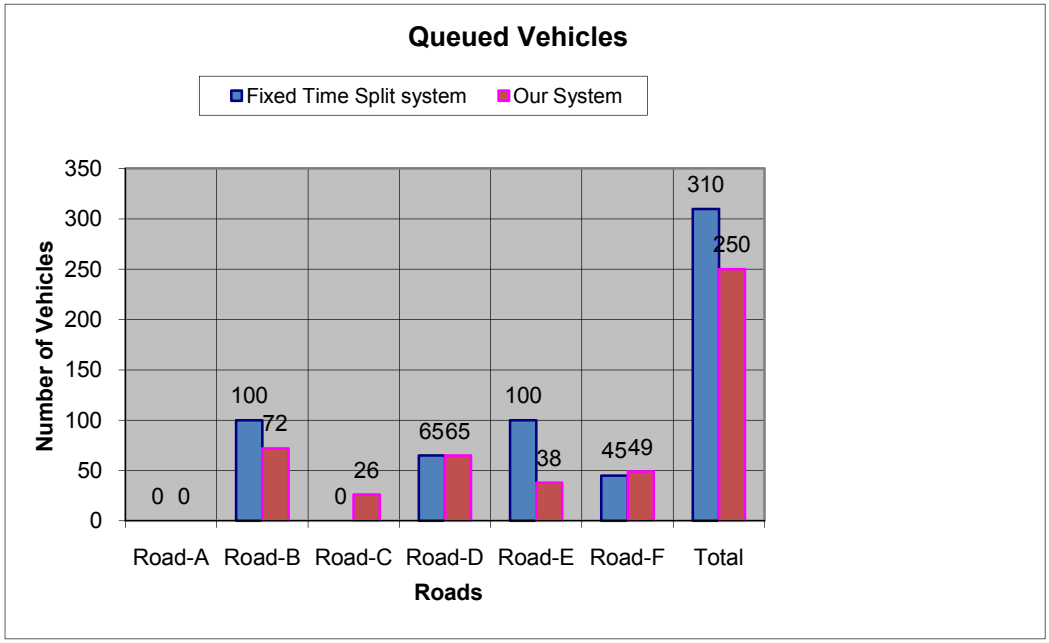


Figure A.7 The result graph of the vehicle data set 1 of sample scenario 3.

Another data set and output graph is given below:

Table A.10 Input vehicle counts data set II for sample scenario 3

	VEHICLE COUNTS INCOMING					
	Road-A	Road-B	Road-C	Road-D	Road-E	Road-F
Time-1	40	50	10	60	10	10
Time-2	40	50	15	55	15	15
Time-3	40	50	20	50	25	20
Time-4	40	50	25	45	30	25
Time-5	40	50	30	40	35	30
Time-6	40	50	35	35	40	35
Time-7	40	50	40	30	45	45
Time-8	40	50	45	25	50	50
Time-9	40	50	55	15	55	55
Time-10	40	50	60	10	60	60

By processing the given data set the following result graph is obtained.

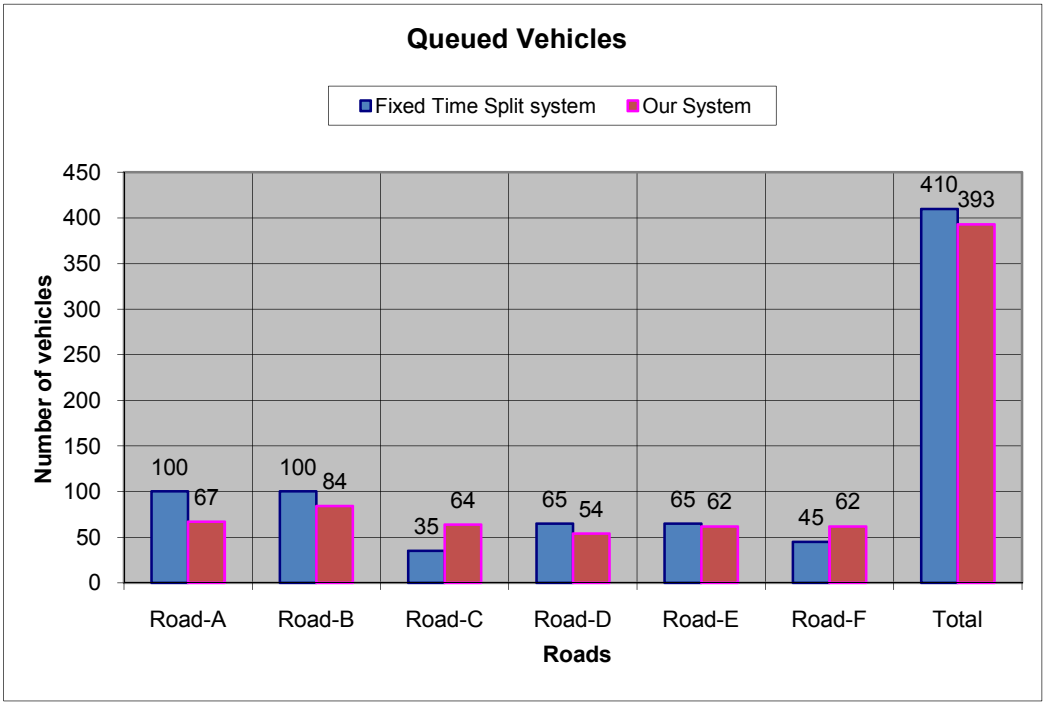


Figure A.8 The result graph of the queued vehicles for data set 2 of sample scenario 3.

As it is in the previous graph, at the end of simulation, our system outperforms fixed time-split system and again it uniformly distributes the waiting vehicles on roads. Because of fuzzy and autonomous nature, our system may produce different results for the same data set. For example, when the second test is repeated with same date, the program generated the following results. Probably it is related with minimum and maximum assumptions that are affecting results regarding to small changes.

Table A.11 The result table of the queued vehicles for data set 2 of sample scenario 3

At the end of time period (Second Test)	Road-A	Road-B	Road-C	Road-D	Road-E	Road-F	Total
Fixed Time Split system	100	100	35	65	65	45	410
Our System	67	92	75	58	43	43	378

Scenario 4: Ege University Hospital Intersection – Campus Link

The intersection diagram is given below:

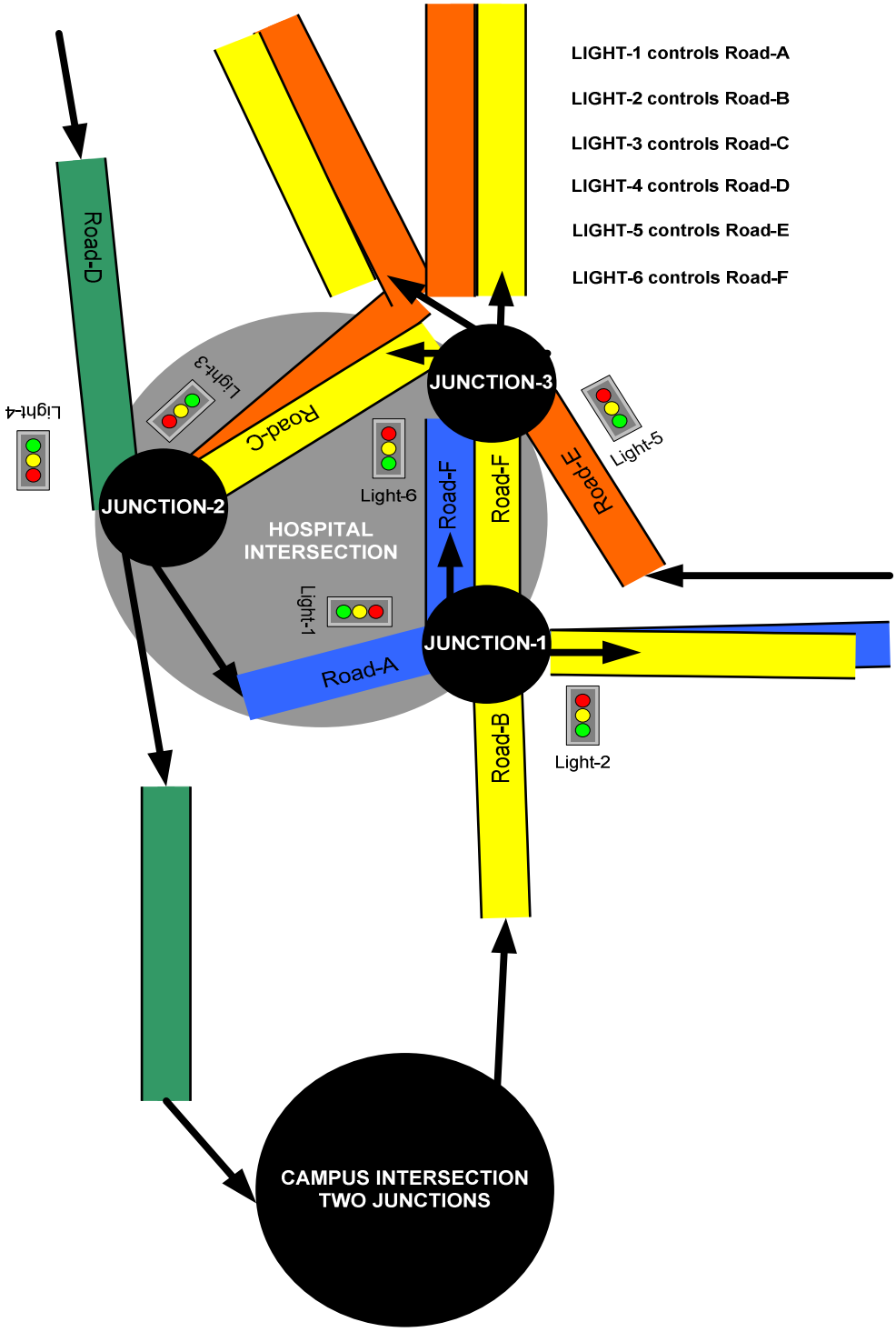


Figure A.9 A three junctions topology scenario in a single intersection.

According to this figure, the following agent structure is generated.

1 Intersection Agent (IS-1, top-controller)

3 Junction Agent (middle-level controller, J-1, J-2, J-3)

6 Light Agents (L-1, L-2, L-3, L-4, L-5, L-6, each one controls one road-cross point)

6 Road Agents

Road-A: Light-1

Road-B: Light-2

Road-C: Light-3

Road-D: Light-4

Road-E: Light-5

Road-F: Light-6

Data Collection period parameters:

- Light Agent collects states of road agents each 30 seconds
- Junction Agent collects states of light agents each 45 seconds
- Intersection Agent collects state of Junction agent each 60 seconds.
- Total Cycle for each junction is 60 sec. Initially each Light agent launches 30sec green and 30sec red signal.
- The vehicle count above 100 in queues is ignored, because road capacities are assumed as 100.

The first sample data table is given as follows.

Table A.12 The vehicle data input set I of the sample scenario 4

	VEHICLE COUNTS INCOMING FROM FILE					
	Road-A (from C,D)	Road-B	Road-C (from E,F)	Road-D	Road-E	Road-F (from A,B)
Time-1	30	50	30	60	10	30
Time-2	0 – 33(T)	50	0 – 8(T)	55	15	0 – 61(T)
Time-3	0 – 28(T)	50	0 – 15(T)	50	25	0 – 63(T)
Time-4	0 – 27(T)	50	0 – 18(T)	45	30	0 – 60(T)
Time-5	0 – 26(T)	50	0 – 18(T)	40	35	0 – 59(T)
Time-6	0 – 22(T)	50	0 – 19(T)	35	40	0 – 58(T)
Time-7	0 – 20(T)	50	0 – 20(T)	30	45	0 – 54(T)
Time-8	0 – 17(T)	50	0 – 20(T)	25	50	0 – 54(T)
Time-9	0 – 15(T)	50	0 – 21(T)	15	55	0 – 51(T)
Time-10	0 – 10(T)	50	0 – 21(T)	10	60	0 – 51(T)

T represents transferred vehicle count according to pre-timed controller scale.

In order to transfer vehicle flow to intermediate road lanes (shown with 0 values in the table given above), the database configuration table has been filled with some default values. The table keeps leak rates between source and destination road lanes. Whenever source lane passes vehicles, it sends a portion of passed vehicles to target road. During run-time, these values are used for fuzzy estimations. The Road Link table diagram for this scenario is given below.

Table A.13 Road to road connection leak rates for internal road links

RoadLinks DB TABLE			
SourceRoad	**TargetRoad**	**LeakRate**	**TargetMaster**
Road-A	Road-F	70	Light-6
Road-B	Road-F	80	Light-6
Road-C	Road-A	10	Light-1
Road-D	Road-A	50	Light-1
Road-E	Road-C	20	Light-3
Road-F	Road-C	20	Light-3

After processing the given data set against the new controller and the fixed-time split systems, the following result graph is generated.

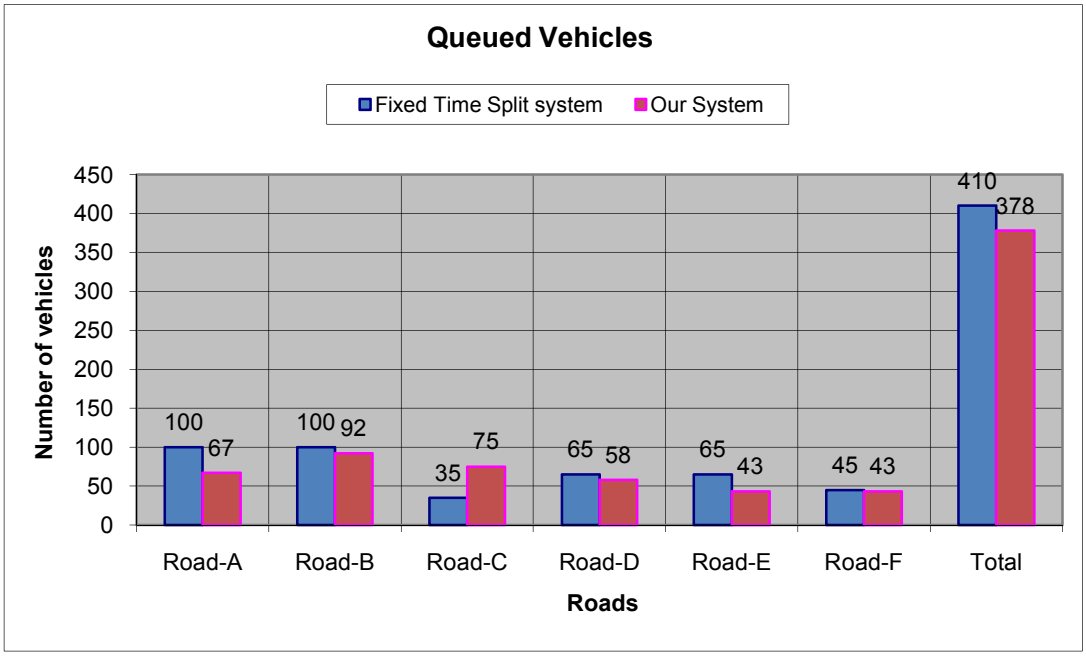


Figure A.10 The result graph of data set 1 of Sample Scenario 4.

As it is seen from graph, our system outperforms fixed-time split system. Some small inconsistencies can have been caused by intermediate transitions. However, in general, our system solution makes more vehicles passed through the intersection.

Based on the same scenario, another data set and output graph is given below.

Table A.14 The vehicle data input set II of sample scenario 4 (T represents the transferred vehicle count according to the pre-timed controller scale)

	VEHICLE COUNTS INCOMING					
	Road-A (from C,D)	Road-B	Road-C (from E,F)	Road-D	Road-E	Road-F (from A,B)
Time-1	30	40	30	60	20	30
Time-2	0 – 33(T)	40	0 – 10(T)	55	20	0 – 53(T)
Time-3	0 – 28(T)	40	0 – 15(T)	50	20	0 – 55(T)
Time-4	0 – 26(T)	40	0 – 15(T)	45	20	0 – 52(T)
Time-5	0 – 24(T)	40	0 – 14(T)	40	20	0 – 50(T)
Time-6	0 – 21(T)	40	0 – 14(T)	35	20	0 – 49(T)
Time-7	0 – 19(T)	40	0 – 14(T)	30	20	0 – 47(T)
Time-8	0 – 16(T)	40	0 – 13(T)	25	20	0 – 45(T)
Time-9	0 – 14(T)	40	0 – 13(T)	20	20	0 – 43(T)
Time-10	0 – 11(T)	40	0 – 13(T)	15	20	0 – 43(T)

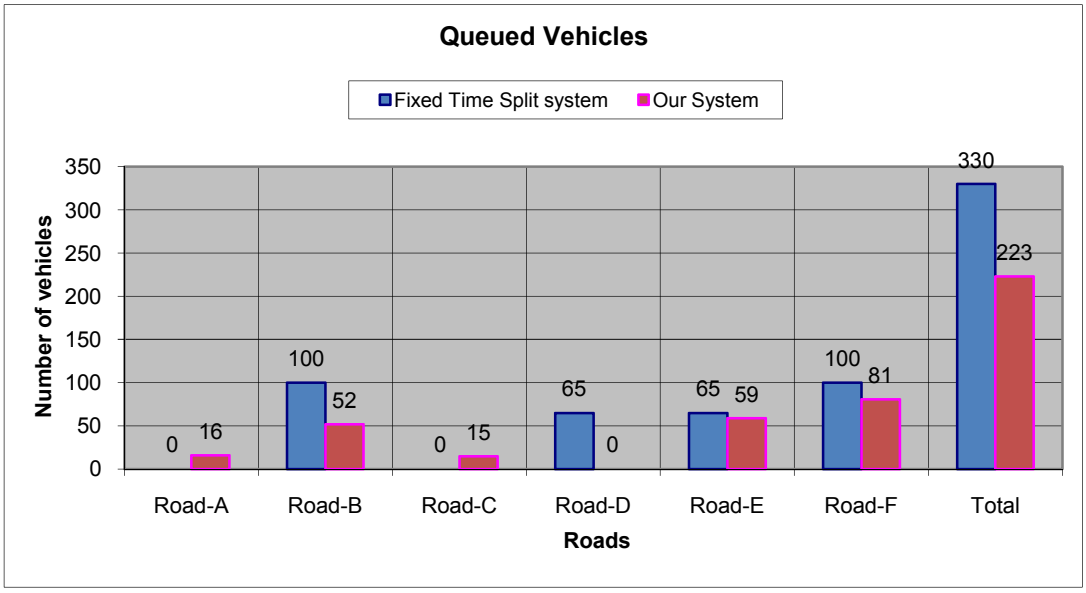


Figure A.12 The result graph of data set 2 of sample scenario 4.

The results show that our system outperforms the fixed-time system again. Moreover, in this case a bit more clearly and sharply.

Appendix E – Light Agent Program Code

```
package JadeTestAgent;
/**
 * <p>Title: Light Agent </p>
 * <p>Description: all behaviors of Light Agent coded here</p>
 * <p>Copyright: Copyright (c) 2008 by A.Ş.</p>
 * <p>Company: DEU University</p>
 * @author unascribed
 * @version 1.0
 */
import jade.core.Agent;
import jade.core.AID;
import jade.core.behaviours.*;
import java.util.*;
import java.io.IOException;
import jade.domain.DFService;
import jade.domain.FIPAException;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;
import jade.lang.acl.UnreadableException;

public class LightAgent extends Agent
{
  private AID[] roadAgents = null;
  private int   step = 0;
  private int   iTimerFreq;
  private RegistrationUI regObj;
  private String myGroup = "None", Master = "None";
  private FuzzyData myFuzzy;
  protected void setup()
  {
    Object[] args = getArguments();
    DatabaseUI  dbObj = new DatabaseUI();
    regObj = new RegistrationUI();
    if (args != null && args.length > 1)
    {
      myGroup = (String) args[0];
      Master = (String) args[1];
    }
    AID myAID = getAID();
    System.out.println("LAgent:is ready,
MyGroup:"+myGroup+"Master:"+Master);
    regObj.RegisterAgent(Master,myGroup,myAID,this);
    dbObj.ConnectToDB();
    iTimerFreq = dbObj.getTimerValue("Select LightPeriod From
TimerParams","LightPeriod");
    dbObj.DisconnectFromDB();
    myFuzzy = new FuzzyData();
    myFuzzy.State = 1;//Default is normal,
    myFuzzy.myGroup = myGroup;
    addBehaviour(new WaitRequest());
    addBehaviour(new WaitPeriod());
    addBehaviour(new TickerBehaviour(this,iTimerFreq)
    {
        protected void onTick()
```

```
        {
            DFAgentDescription  template = new DFAgentDescription();
            ServiceDescription  sd = new ServiceDescription();
            sd.setType(myGroup);
            template.addServices(sd);
            try
            {
                DFAgentDescription[] result =
DFService.search(myAgent,
                        template);
                if (result.length > 0)
                {
                    roadAgents = new AID[result.length]; //no escape
                    regObj.AssignAgents(result, roadAgents);
                    myAgent.addBehaviour(new Collector());
                }
                step = 0;
            }
            catch(FIPAException fe)
            {
                fe.printStackTrace();
            }
        }
    });
  }
  protected void takeDown()
  {
    System.out.println( myGroup + " terminating");
  }
  /**********************************************/
  private class WaitRequest extends CyclicBehaviour
  {
    public void action ()
    {
      byte State = 0;//
      FuzzyData  msgData = new FuzzyData();
      MessageTemplate m1 =
MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
      MessageTemplate mt = MessageTemplate.and(m1,
MessageTemplate.MatchLanguage(Master));
      ACLMessage msg = myAgent.receive(mt);
      if (msg != null)
      {
        String title = msg.getContent();
        ACLMessage  reply = new ACLMessage(ACLMessage.INFORM);
        reply.addReceiver(msg.getSender());
        System.out.println(myGroup +": send its state value:" +
myFuzzy.State + " to "+ Master);//getAID().getName()
        reply.setLanguage(Master);
        msgData.State = myFuzzy.State;
        msgData.myGroup = myGroup;
        msgData.hvalue = myFuzzy.hvalue;
        msgData.lvalue = myFuzzy.lvalue;
        msgData.nvalue = myFuzzy.nvalue;
        msgData.tvalue = myFuzzy.tvalue;
        try
        {
          reply.setContentObject(msgData);//String.valueOf(myState)
        } catch (IOException e)
```

```
        {
          e.printStackTrace();
        }
        myAgent.send(reply);
      } else block();
    }
  }
  /**************************************************/
  private class WaitPeriod extends CyclicBehaviour
  {
    public void action()
    {
      int RedPeriod = 0;//
      MessageTemplate m1 =
MessageTemplate.MatchPerformative(ACLMessage.INFORM);
      MessageTemplate mt = MessageTemplate.and(m1,
MessageTemplate.MatchLanguage(Master));
      ACLMessage msg = myAgent.receive(mt);
      if (msg != null)
      {
        RedPeriod = Integer.parseInt(msg.getContent());
        System.out.println(myGroup + ":Getting new Period Values (in
SECONDs), Setting Red-Light:"+ RedPeriod +
        " Green-Light:"+ (GeneralFuncs.TOTAL_CYCLE - RedPeriod));
        InformRoadAgents(GeneralFuncs.TOTAL_CYCLE - RedPeriod);
      }
      else block();
    }
    private void InformRoadAgents(int RedPeriod)
    {
      DFAgentDescription  template = new DFAgentDescription();
      ServiceDescription  sd = new ServiceDescription();
      sd.setType(myGroup);
      template.addServices(sd);
      try
      {
          DFAgentDescription[] result = DFService.search(myAgent,
                  template);
          if (result.length > 0)
          {
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
            msg.addReceiver(result[0].getName());
            msg.setLanguage(myGroup);
            msg.setContent(String.valueOf(RedPeriod));
            myAgent.send(msg);
          }
      }
      catch(FIPAException fe)
      {
          fe.printStackTrace();
      }
    }
  }
  /**************************************/
  private class Collector extends Behaviour
  {
    private int repliesCnt = 0;
    private MessageTemplate mt,m1;
    private FuzzyData[] VolumeData;
```

```
      public void action()
      {

System.out.println("..........................................
...");
         ACLMessage cfp = new ACLMessage(ACLMessage.REQUEST);
         for (int i = 0; i < roadAgents.length; ++i)
           cfp.addReceiver(roadAgents[i]);
         cfp.setLanguage(myGroup);
         myAgent.send(cfp);
         m1 = MessageTemplate.MatchLanguage(myGroup);
         mt =
MessageTemplate.and(m1,MessageTemplate.MatchPerformative(ACLMessage.
INFORM));
         VolumeData = new FuzzyData[roadAgents.length];
         while (repliesCnt < roadAgents.length)
         {
            ACLMessage reply = myAgent.receive(mt);
            if (reply != null)
            {
                FuzzyData fuzzyObj;
                try
                {
                  fuzzyObj = (FuzzyData) reply.getContentObject();
                  AID sender = reply.getSender();
                  String name = sender.getName();
                  System.out.println(myGroup + ":Received State from
" + name + ":" + fuzzyObj.State);
                  VolumeData[repliesCnt] = fuzzyObj;
                  repliesCnt++;
                } catch(UnreadableException e3)
                {
               System.err.println(getLocalName()+ " catched exception
"+e3.getMessage());
                }
            }
            else block();
         }//end of action.
         if (repliesCnt >= roadAgents.length)
         {
           step=2;
           String GlobalFuzzy = "Unknown";
           GeneralFuncs  FuncsObj = new GeneralFuncs();
           FuncsObj.FindAvgFuzzy(VolumeData,myFuzzy);
           System.out.println(myGroup + "->Low:"+ myFuzzy.lvalue+"
Normal:"+ myFuzzy.nvalue+" High:"+ myFuzzy.hvalue+" TooHigh:"
           +myFuzzy.tvalue);
           GlobalFuzzy = FuncsObj.PrintFuzzyState(myFuzzy.State);
           System.out.println(myGroup + ":My fuzzy state is " +
GlobalFuzzy);
           byte[] index = myGroup.getBytes();
           InformScreen(GlobalFuzzy,(byte)(index[index.length-1]-
0x30));
         }
      }//end of action
      public boolean done()
      {
       return (step == 2);
      }
```

```
    private void InformScreen(String sFuzzy, byte index)
    {
      DFAgentDescription  template = new DFAgentDescription();
      ServiceDescription  sd = new ServiceDescription();
      sd.setType("SIMULATOR");
      template.addServices(sd);
      try
      {
          DFAgentDescription[] result = DFService.search(myAgent,
                  template);
          if (result.length > 0)
          {
            ScreenMsg sm = new ScreenMsg();
            sm.Index = index;
            sm.FuzzySet = sFuzzy;
            ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
            msg.addReceiver(result[0].getName());
            msg.setLanguage("Light");
            msg.setContentObject(sm);
            myAgent.send(msg);
          }
      }
      catch(FIPAException fe)
      {
          fe.printStackTrace();
      }
      catch(IOException fe)
      {
        fe.printStackTrace();
      }
    }
 }//end of RequestPerformer
}
```