

https://dspace.org/

DaTiLis: distributed traffic light simulator

Bergheim, Gavin

2016-07-18
http://hdl.handle.net/10539/20659

Downloaded from a DSpace repository. https://dspace.org/

DaTiLiS:

Distributed Traffic Light Simulator

Gavin Bergheim **90-08688P**

Supervisor: Dr John Bradshaw

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, in partial fulfilment of the requirements for the degree of Master of Science

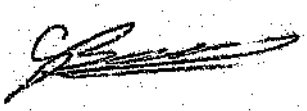
Degree awarded with distinction on 4 December 1996.

Johannesburg, 1996

Declaration

I declare that this research report is my own, unaided work. It is being submitted for the Degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

Signed on 26 November 1996



Gavin S. Bergheim

Abstract

The problem with existing traffic light systems is that they are static and unable to respond to dynamic changes in traffic flow patterns. This report discusses an approach using distributed AI techniques to allow the control system to respond dynamically to the changes in the flow patterns.

A simulated system (Datilis) has been developed. The design of the system is based on the assumption that maximising throughput at individual intersections will result in improved overall performance. The results generated and discussed in this report support this hypothesis.

Acknowledgements

Dr John Bradshaw for many ideas and directions related to this research

Contents

DECLARATION II

ABSTRACT III

ACKNOWLEDGEMENTS IV

CONTENTS V

LIST OF FIGURES VII

LIST OF TABLES IX

INTRODUCTION 1

General Introduction 1

Traffic Control System 1

Operations Research 3

Outline 4

LITERATURE SURVEY 6

Distributed Artificial Intelligence 6

Distributed Domain 7

Distribution and Task Allocation 8

Coherence and Co-ordination 10

Interaction Languages, Structures and Protocols 15

Implementation Languages and Systems 17

Blackboard Structures 19

Multi Agent Planning Literature Review 19

General Planning Systems 20

Interactions and Dependencies 21

Planning and Acting 24

Formal Models of Action 27

Formal Models of Planning Systems 28

Time and the Frame Problem 30

Learning and Reuse 31

Extending the Classical Framework 31

Planning and Execution 32

RESEARCH AND METHODOLOGY 34

RESEARCH 34

DATILIS 36

DATILIS CONTROL STRUCTURES 40

TIMING DATA CONTROL STRUCTURE 46

EXAMPLE PROBLEM 53

TESTING METHODOLOGY 60

RESULTS 63

BASE RESULTS 63

2ND STAGE RESULTS 64

COMPARISON OF RESULTS 66

GENERAL DISCUSSION AND CONCLUSION 69

NETWORKING AND DAI 70

DATILIS AND UNCERTAINTY 71

FURTHER WORK AND ENHANCEMENTS 71

SUMMARY 72

REFERENCES 74

List of figures

Figure 1: An example of a traffic grid.....37
Figure 2: The Direction of Control of an Intersection. Control (green robot) travels from Robot 1 to 2, 3 and 4 and then back to 1 again.....38
Figure 3: The basic conceptual overview of Datlis showing both the blackboard placement and the added communication channels between intersections.....39
Figure 4: The conceptual overview of a single agents blackboard. This diagram shows the structural breakdown of a blackboard. All flow information is shown. Information flows from the min-blackboards North, South, East and West to the timing data control structure and then back again.....41
Figure 5: A diagram showing the fixed communication pathways in the Datlis Simulator. The circle (in the centre of the diagram) touches all neighbouring nodes of the centre intersection, intersection 5. Intersection 5 can only communicate with these neighbouring nodes.....43
Figure 6: Emergency communication between Intersection 1 and Intersection 3. In general Intersection 1 would only be able to communicate with Intersection 2 and Intersection 4.....46
Figure 7: Diagram showing the best case scenario for a intersection. Cars do not only have to be arriving at the next traffic light but could be arriving at any of the traffic lights in the intersection.....50
Figure 8: Diagram showing the average and worst case scenarios for a intersection. As can be seen cars arrive in small burst with a number of gaps between them. Again these cars could be arriving at any of the traffic lights in the intersection.....51
Figure 9: The traffic grid used for example and for the final simulation runs.....54
Figure 10: The route taken by the car in the example simulation. The car enters the traffic grid at Intersection 1 and leaves the traffic grid at Intersection 13.....55
Figure 11: Graphic showing the position of control in an intersection for best, average and worst case scenarios.....58
Figure 12: The results of the example showing the Best / Distributed case Vs the base case. As can be seen from the graph the two lines are clearly divergent.....59
Figure 13: Graph showing all the results of the example with the difference between the best and base results also being shown.....60
Figure 14: Graph showing the base results for Datlis. As can be seen from the Base Time - Best Time line the graph is clearly divergent. Therefore the more cars used in the simulator the greater the difference between the best time and the base time.....63

Figure 15: All results including the results of the simulator when using distributed control methods. As can be seen from this graph the Distributed AI time is clearly better than the Base Time. 64
Figure 16: This graph is exactly the same as Figure 15 but has a additional line indicating the difference between the Distributed Time and the Best Time. As can be seen from the graph as more cars are added to the system the distributed time gets increase compared to the best time. 65
Figure 17: Same graph as Figure 16 but with added blocks showing the areas that are discussed. 66

List of tables

Table 1: Table showing the possible timing values for each traffic light in a intersection. 49

Table 2: The statistical data for the example. Best Time is the fastest time the car can move through the system. Car Move time is the actual times when the car moved through a intersection. Difference column is the difference in time between the best time and actual time. The expected time is the time the car arrived at the intersection. So the difference between the expected time and Car move time is the waiting time at the intersections. 56

Introduction

General Introduction

Most Artificial Intelligence (AI) research investigates intelligent single agent behaviour. DAI can be divided into two primary areas of research , the first is Distributed Problem Solving and the second is Multi-agent Systems (MA). Research in the first area considers how the work of solving a particular problem can be divided among a number of independent modules. Research in the second area concerns itself with co-ordinating intelligent behaviour amongst a collection of autonomous intelligent agents and how they co-ordinate their knowledge, goals, skills, and plans jointly to take action or to solve problems. Multi-agent systems work towards a single goal, or toward separate, individual goals that interact.

The co-ordination of intelligent multi-agent behaviour is the primary focus of this report. The research application, a distributed traffic control simulator, is an example of a system that is distributed but demonstrates no intelligence.

Traffic Control System

A grid of traffic lights controlling the flow of cars can be thought of as a traffic control system. The primary function of the traffic light system is to maintain a reasonable flow of cars in all directions. Traffic lights are situated at each intersection in a grid of roads. These intersections become areas of "bottlenecks" because cars tend to be stopped and queued at these points. Intersections will become saturated as more cars are queued behind the waiting cars and if traffic lights do not allow enough cars to cross the intersections. Since the intersection's traffic lights times are fixed and static the intersection cannot cope with excess traffic or unexpected peaks.

Intersections are also points of conflicting interests for different cars. Each car has to travel through a number of intersections depending on the car's destination. People driving the cars want to arrive at their destinations as quickly as possible and will pick what they think is the best route to get there. These routes may contain intersections that are incapacitated, blocked or saturated. Even if there is no other traffic at an intersection a car may still be forced to wait.

So some of the main problems with a static traffic light system are:

- the inability to respond to unexpected traffic flow.
- If an intersection becomes incapacitated there is no ability to re-route traffic around it.

Through the use of intelligent intersections, the time intervals of the traffic lights can be dynamically changed in response to the conditions at the intersection. Routes of cars can be changed to accommodate intersection breakdowns and compensate for high volume areas. It is therefore hypothesised that through the use of intelligent intersections the volume of cars travelling efficiently through a traffic grid can be increased.

In general it is hypothesed that if the throughput of an individual intersection is maximised and the waiting time of individual cars is minimised then the throughput of the whole system will be maximised. This hypothesis is the basic function for quantifying whether the distributed traffic system has moved from a worse state to a better state or vice versa as compared to the static traffic light system.

The basic hypothesis of maximising the flow through each intersection implies conflicting interest due to the "greedy nature" of the assumption. It is impossible to maximise the throughput of all the intersections as they interact in complex ways in relation to each other. Even with the "benevolent agent assumption" (all agents must fully co-operate with all other agents whenever possible) employed it is still impossible

to minimise the waiting time for all cars and to maximise the throughput of every intersection. So a number of distributed artificial intelligent techniques will have to be used to overcome these limitations or rather to minimise the effect and hence maximise the flow of the distributed system.

Techniques such as task sharing (Smith, 1981) , result sharing (Smith, 1981) and distributed blackboards are some example that can be used to increase the efficiency of a distributed system. To overcome the limitation of uncertainty in the system and limited global knowledge, functionally accurate and co-operative systems (Lesser, 1981) could also be used. Heuristics could also be used to overcome the dynamic nature and sometimes contradictory nuances of the traffic flow.

To summarise, a collection of algorithms and combinations of AI techniques can be used to enhance a simple distributed system. A system is made up of a number of intelligent autonomous agents that communicate and co-ordinate their information to enhance the flow of information through the system. The flow of information is also controlled directly via the autonomous agents.

This dissertation investigates how certain distributed techniques can be used to enhance a traffic control system. The research report also considers how DAI techniques developed for this system could be applied to other similar routing and planning areas such as computer networks.

Operations Research

Another area of research also involved in the traffic light problem is operations research. Through the use of mainly static algorithms, such as linear programming and queuing theory, static solutions to problems are worked out through the operations research model. Operations research presents static solutions to problems where all

variables have been pre-calculated. The assumption is made that there is no change over time to any variable in the system or that the change is very minimal.

Many of the principles of modelling a problem devised in operations research can be used in distributed artificial intelligence design. These principles are (Ravindran, 1987):

1.Do not build a complicated model when a simple one will suffice

2.Beware of moulding the problem to fit the technique.

3.The deduction phase of the modelling must be conducted rigorously.

4.Models should be validated prior to implementation.

5.A model should never be taken too literally.

6.A model should neither be pressed to do, nor criticised for failing to do, that for which it was never intended.

7.Beware of overselling a model.

8.A model cannot be any better than the information that goes into it.

Many of these design philosophies will be discussed in the context of the traffic light simulator in the conclusion of this report. Operations research works in averages and multiple constraints while distributed artificial intelligence has the ability to automatically define solutions depending on the status of the model. Uncertainty and dynamic variables can be controlled and designed within a DAI system, unlike an operations research model.

Outline

Before a more detailed analysis is given of Datilis and Datilis's control and data structures a literature survey is presented. The survey summarises the core backbone of the research and many essential techniques and ideas in the area of DAI and multi agent planning. This is followed by an overview of Datilis and the testing procedures used in the research. The DAI techniques used are discussed in more detail within a

detailed explanation of Datilis. This is followed by the statistical results with a detailed explanation of these results. A general discussion and conclusion of the work ends the report.

Literature survey

Distributed Artificial Intelligence

All distributed situations consist of a collection of agents i.e. robots, sensors, communication agents, where each of these agents has various skills. These group of nodes have an assigned task as a global unit. Each agent, taken locally, has limited knowledge of the global system and shared resources. To co-ordinate their actions, the knowledge sources (agents) need to represent and reason about the knowledge, actions, and plans of other agents. Each computing agent is a semi-autonomous problem solving node that can communicate with other nodes. Nodes work together to solve a single problem by individually solving interacting subproblems and integrating their solutions. The main conflict in any distributed system is the conflict between the complete knowledge needed to ensure coherence and the incomplete knowledge inherent in any distribution of problem solving effort. Any multi-agent system should ideally be able to form and restructure co-ordination frameworks based on emerging contexts and changing problem-solving roles with out the intervention of a programmer (Bond, 1988).

Every Distributed AI (Artificial Intelligence) framework will have some or all of the following basic underlying problems (Bond, 1988):

- How to formulate, describe, decompose, and allocate problems and synthesise results among a group of intelligent agents.
- How to enable agents to communicate and interact.
- How to ensure that agents a. coherently in making decisions or taking action, accommodating the global effects of local decisions and avoiding harmful interaction.
- How to enable individual agents to represent and reason about the actions, plans, and knowledge of other agents.

- How to recognise and reconcile disparate viewpoints and conflicting intentions among a collection of agents trying to co-ordinate their actions.

Interest in Distributed problem solvers arises from the promise of increased speed, reliability, and extensibility, as well as the ability to handle applications with a natural spatial and functional distribution, and the potential for increased tolerance to uncertainty in data and knowledge.

In the following chapters a generalised overview is given of current distributed AI research. Because of the size of the distributed AI survey, the papers have been divided up into a number of smaller sections. These sections are: the distributed domain, distribution and task allocation, coherence and co-ordination, interaction languages and systems and finally blackboard structures.

Distributed Domain

Any distributed system can be viewed as a system of autonomous agents that co-operate to facilitate a solution to a larger problem. A physical distributed framework is best modelled via a system that can take into account all distributed constructs. Individual computational agents have bounded rationality (Bond, 1988), bounded resources for problem solving, and possibly bounded influence, necessitating in co-operation and co-ordination between nodes to solve large problems. Communicating problem solvers can pool their resources and expertise, work in parallel on different parts of a problem to solve it faster, avoid harmful interactions and move information to where it is most needed. In this way a distributed framework enhances and improves performance of certain problems over non-distributed centralised frameworks. Effective speedup in a distributed system will only take place if efficient use of the communications channels are made (Bond, 1988).

Therefore a typical distributed system would have a number of rationales:

- The Distributed system should be **Adaptable,** the system should be able to provide alternative perspectives on emerging situations.
- The system should also be able to **adapt** to changes in the hardware and communication structures.
- The distributed system should be **cost-effective.** A system can involve a large number of simple computer systems of low unit cost. Communications cost will feature highly in the cost effectiveness of the entire system.
- **Concurrency** can increase the speed of computation and reasoning.
- For **protection** and **control,** parts of a system may be separated and isolated from one another. Some problems are better described as **collections** of separate agents.
- Distributed systems may be more **reliable** because they provide redundancy, cross-checking, and triangulation of results.

Distribution and Task Allocation

The problem of allocating particular tasks to particular agents is the problem of assigning responsibility for particular activities. Task allocation may be tackled by the designer, for example, by establishing a collection of specialised agents with predetermined roles, or be done dynamically by a collection of agents themselves, using techniques such as contracting and organisation self-design. Solutions to a distributed system must be globally coherent and this global coherence must be achieved by local computation alone (Cammarata, 1983). This requirement of local computation leads to greater difficulties in solving a given task but much of the power of distributed problem solving comes through effective co-operation and communication.

A dynamic approach to problem solving requires reliable communication and integration between all knowledge sources. Integration dictates how a larger task should be decomposed into smaller sub-tasks. Communication can be both broadcasted or directed, unsolicited or on-demand, acknowledged or unacknowledged or a single-

transmission or repeated-transmission. If communication is unacknowledged, alternative solutions must be planned for in case "receiving nodes" do not act as expected.

Two generalised approaches for co-operation in a distributed network are *task-sharing* and *result-sharing* (Smith, 1981). *Task-sharing* is used for the problem decomposition phase and answer synthesis phases while *result-sharing* is useful because certain sub-problems cannot be solved by nodes working independently without communication with other nodes. There are problems with both *task-sharing* and *result-sharing*.

In the case of *task-sharing*, mechanisms must be designed that give some assurance that individual sub-problems are actually processed, and that processors do not get in each other's way in trying to solve identical sub-problems while other sub-problems are inadvertently ignored. It was suggested that negotiation is a mechanism for dealing with the above problems and the solution was specified by the contract net protocol (Smith, 1981). Use of the contract net protocol makes it possible for a system to be configured dynamically, taking into account such factors as the number of sensor and processor nodes available, their locations, and the ease with which communication can be established (Smith, 1981).

In the case of *result-sharing* there must be some assurances that nodes influence each other in such a way as to converge to a correct solution. Result-sharing was used in Hearsay II (Fennel, 1977). In this system the overall system control was data-directed. This was due to the control being based on events occurring in a globally shared blackboard system. Using this method there is a basic self-correcting nature of information flow. *Result-sharing* offers no mechanism for problem decomposition. Therefore *task-sharing* and *result-sharing* can be used together drawing upon their individual strengths to attack problems for which neither form is sufficiently powerful by itself. Both types of co-operation can achieve stability and fault tolerance if agents overlap tasks and cross check results.

To gain a better understanding of a distributed system Fox (1981) relates the distributed system to human organisations. Therefore task complexity and uncertainty, coupled with resource constraints are important factors in deciding how a system is to be distributed. Complexity is deemed as excessive demands on rationality (Fox, 1981). It appears that complexity and uncertainty are two opposing forces. Complexity leads to heterachical structures where there is abstraction and omission of information. This leads to uncertainty where there is a difference between information available and the information necessary to make the best decision. In most distributed systems there is a trade-off between task complexity and uncertainty. These factors must also be balanced with cost considerations for communications. A system can fail when sharing exceeds capacity, and decision making and co-ordination becomes too complex. As uncertainty increases, the transition from a heterarchy to a hierarchy becomes preferable (Fox, 1981).

Coherence and Co-ordination

In any distributed system there must be coherence and co-ordination to ensure a viable working model. The primary difficulty in establishing coherence and co-ordination is the attempt to achieve them without centralised control or viewpoints. Coherence is defined as the "properties of the total distributed system" while co-ordination is the particular patterns of activity and interaction among the agents (Bond, 1988). Solutions found by the system and sub-systems should be satisfactory and should be efficient given the time constraints the system is under. The system should degrade gracefully if any of the nodes fail. Because each node in a distributed system is operating with limited local knowledge there needs to be co-operation amongst the nodes.

Nodes should avoid unnecessarily duplicating the work of others, sitting idle while other nodes are swamped with work, or transmitting information that will not improve overall performance. Full global coherence can only be maintained by nodes broadcasting all changes in there status. Because of the real-world communication channels and limited bandwidth there is no practical means to ensure full global

coherence (Bond, 1988). Coherent co-operation requires that, at any given time, each node performs a sequence of tasks to generate a tentative partial solution that is compatible with the solutions being generated at other nodes. By iteratively exchanging their potentially incomplete, inaccurate, and inconsistent partial solutions, the nodes eventually converge on an overall network solution. Effective co-ordination implies some degree of mutual predictability and lack of conflict. To achieve all this a planning module is needed.

There exists two major paradigms for planning in a distributed system. The first is planning for multiple agents while the second is distributed problem solving. In Multiple agent planning, a single intelligent agent constructs a plan to be carried out by a group of agents, and then hands out the pieces of the plan to the relevant individuals. Distributed problem solving is where a group of intelligent agents act together to construct, and possibly execute, the final plan (Rosenschein, 1982). The use of a global "controller" to make decisions for a distributed system is not an option because it would be a severe communication and computational bottleneck and would make the network susceptible to complete collapse if it fails (Durfee, 1987b). This global method of planning could be used in smaller more compact areas of the organisational structure thereby limiting communication to a smaller number of nodes. In the event of failure the "supervisor" can be replace by another node.

Rosenschein (1982) proposes a method whereby an intelligent agent constructs a plan to be executed by other agents while Georgeff (1983) proposes a method for synthesising multi-agent plans from simpler single-agent plans. Georgeff describes a relatively simple method for achieving this synchronisation, given that the plans of the individual agents have already been constructed.

Both of these planning systems could be used in conjunction with one another, as both have weaknesses and strengths which are complimentary to each other. A hierarchical organisational structure is well suited to Rosenschein's (1982) method where there is one node controlling a smaller sub-section of nodes in an area. The

supervisory root node will then contain a single global plan for the hierarchical tree. Georgeff's method then could be used to correlate and synchronise the plans of all the supervisory nodes.

Synchronisation of plans is a major issue in both of the above methods. Stuart (1985) proposes a system which augments plans with synchronising primitives to ensure conflict avoidance and co-operation. With the use of synchronising primitives, a distributed system could resolve conflicts and produce plan's which are as unrestrictive as possible.

Each agent has its own set of desires and goals, and even though there is the *benevolent agent assumption* (each agent will to the best of its abilities attempt to solve the given problem and will not interfere with the actions of other nodes), agents may not necessarily help other agents with information or with actions. Because of the physical distributed nature of a system, communication makes mutually beneficial activity possible which would otherwise have been impossible to co-ordinate (Rosenschein, 1989). Nodes may compete or conflict with each other because each must locally interpret the network goals. Where all agents are assumed to have a single goal, the complexity of the interaction might be better handled by a framework that recognises and resolves sub-goal conflict in a general manner (Rosenschein, 1989). This can be handled by using a payoff matrix that includes communication. The framework presented by Rosenschein (1989) contains methods for interaction that explicitly accounts for communication and promises, and allows multiple goals amongst agents. This framework could be used for coherence and control where through the use of communication and binding promises, agents are able to co-ordinate their actions more effectively. In most systems it is assumed that all agents are benevolent, therefore the payoff matrix representation (Genesereth, 1986) without communication amongst agents would not be useful.

When agents work on interacting tasks, their activities should be co-ordinated so that they co-operate coherently. This coherence is very difficult to achieve since each

agent has only a limited view of the overall task, communication should be kept to a minimum and there is no single dominating "controller" to co-ordinate the agents. Therefore there is no means of ensuring full global coherence. Goals must be achieved by using partial global coherence. This is done by increasing the coherence of local control decisions and by reducing control uncertainty through sophisticated local processing and selective communication decisions. An increase in co-ordination and coherence should result from making each agent more capable locally, and from making it evaluate each of its potential actions in the light of how that action affects general network/system problem solving (Durfee, 1987b). Nodes can get closer to optimality overall by making their local decisions less redundant and/or less disruptive of the work of other nodes. Each problem solving node integrates knowledge of the problem domain with knowledge about network co-ordination. This allows nodes in the system to make rapid, intelligent local decisions based on changing problem characteristics with only a limited amount of intercommunication to co-ordinate these decisions. Each agent must consider how the timing of their local decisions will affect the activities of other agents because sub-systems may need to synchronise or exchange information. Co-operation requires intelligent local control decisions so that each node performs tasks which generate useful subproblem solutions.

A system that uses many of the distributed AI techniques is the Distributed Vehicle Monitoring Testbed (**DVMT**) (Durfee, 1987b). The **DVMT** system uses multi-layered blackboards. The blackboard system of control is used because of the sophisticated local processing capability coupled together with selective communication decisions which reduces control uncertainty.

Co-ordination in a blackboard system is based on an organisational view of individual node activity. Nodes need not have detailed models of each other's problem solving activities. Dynamic improvements to the organisational structure may be achieved using meta-level knowledge. Meta-level messages convey high-level co-ordination information. The basic organisational structure of a distributed system can

broadly define network responsibilities, types of tasks a node may perform, resources assigned to a node and other nodes with which a node may need to synchronise.

By planning task sequences, a node can predict what results it may generate, when it will be idle, what resources it needs, or exactly when it will be executing specific tasks. Meta-level communication enable planning predictions to be exchanged, and network coherence improves as nodes work to generate compatible results, maximise the number of tasks which meet their deadlines, avoid resource conflicts and synchronise task executions (Durfee, 1987b). The independent data-directed nature of the blackboard system can potentially generate a significant amount of parallel activity (Fennel, 1977).

The use of partial global plans (PGPs) as described by Durfee and Lesser et al. (1987a) can enhance a distributed system. Through the uses of PGPs a unified framework that supports all styles of co-operation can be constructed. The use of PGPs specifies how nodes will act and interact. Each basic style of co-operation can be viewed as using partial global plans in some way. Developing and maintaining PGPs involves computation and communication overhead but this is acceptable since PGPs substantially reduce the number of agents executions needed to solve complex situations (Durfee, 1987a).

Because of the dynamic nature of a distributed system conflicts are expected amongst agents, and no realistic collection of agents can be expected to have precisely complete, constant and compatible view-points at all times. Through the use of functionally accurate and co-operative (FA/C) style of processing attempts are made to reduce the amount of conflicts and to deal with them as they arise. Because consistency checking is such an important part of the FA/C approach, it is natural to think of dealing with distribution-caused uncertainty and errors as an integral part of the network problem-solving process (Lesser, 1981). Self-directed control as enhanced by the FA/C style of processing has the added benefit of increased system robustness in the face of communication and node failure and increases system responsiveness to

unexpected events. Treating the distributed problem as a search process, where there may be a number of alternative paths to an acceptable solution, can enhance the functionality and co-ordination of the overall solution. Through use of abstraction communication cost can be reduced (by reducing inter-node communication requirements). Uncertainty in a FA/C system is resolved through Incremental Aggregation and Automatic Error Resolution although it has been shown that some errors could not be resolved (Lesser, 1981). Theoretically this incremental method of problem-solving would allow a distributed system to detect and reduce the impact of incorrect decisions caused by incomplete and inconsistent local data bases and by hardware malfunctions.

Interaction Languages, Structures and Protocols

There must be methods built into a distributed system for interaction among agents of varying types, grain size, and capabilities. The protocols used should help determine the content of the information transmitted, rather than simply provide a means of sending bits from one node to another. Co-operation cannot be established between nodes simply by indicating how they are to communicate, it must also be indicated what the different agents must say to each other. Through negotiation interested parties can exchange information and come to an agreement.

Task distribution can be viewed as an interactive process, a discussion carried on between a node with a task to be executed and a group of nodes that may be able to execute the task. Task distribution is viewed as a form of contract negotiation (Davis, 1983). Through use of the contract net approach, which distributes both control and data, and allows the system to reconfigure itself by taking into account the hardware available. Because of the nature of the contract net a hierarchichal structure develops dynamically with each node having the ability to take on the role as manager or contractor. A manager is responsible for monitoring the execution of a task and processing the results of its execution. A contractor is responsible for the actual execution of the task. Nodes can take the task of manager and contractor simultaneously.

Through the use of the *common internode language* (CIL) (Davis, 1983) all nodes in a distributed system have a common basis for communication. The contract net and negotiation may also be enhanced through the use of multistage negotiation as explained by Conry et al. (Conry, 1986).

Through multistage negotiation, a planner is able to recognise when a problem is overconstrained and to find a solution to an acceptable related problem under these conditions. A planning problem is overconstrained if satisfaction of one top level goal precludes the satisfaction of others. When a number of alternative choices for goal satisfactions are known, detection of an overconstrained situation is not possible without either multistage negotiation or a global view. Multistage negotiation provides a mechanism for reaching a consensus among those nodes with conflicting goals concerning an acceptable satisfying condition. This makes it possible to detect and to resolve subgoal interactions in a distributed system with limited communication bandwidth and no single locus of control. Multistage negotiation generalises the contract net by recognising the need to iteratively exchange inferences drawn by an agent about the impact of its own choice of what local tasks to perform in satisfaction of global goals. Multistage negotiation produces a co-operation strategy similar in character to the Functionally Accurate/Co-operative paradigm in which agents iteratively exchange tentative and high level partial results (Conry, 1986).When satisfaction of a goals requires the commitment of resources, conflicts may arise among goals competing for limited resources.

Through the use of Dialogue-Games as described by Levin et al. (1977) the communication protocols can capture shared, and conventional knowledge that can be used to achieve dialogue. Dialogue-Games could be a viable alternative to other communication protocols or could be used as a subset or enhancement of other communication protocols. For a "Dialogue-Game" framework to work a number of representations for a particular type of interaction have to be described. Therefore

careful study would have to be done on either a similar implemented system or a system in the design phase.

An alternative or enhancement to a blackboard system is a system based on a distributed database system (Cromarty, 1987). In many ways a distributed database system is similar to a blackboard system as there is many of the same problems such as communication costs, fault tolerance, database consistency and multiple data views. This system could be implemented using a Hyperstar architecture which can be thought of as a logical extension of a hierarchical star topology. Hyperlinks provide communication protocols, rapid data paths, or shortcuts through a hierarchical graph structure. Because there is a cost associated with such links resource allocation techniques are used to determine weather a given pair of nodes qualify. All nodes are given the latent ability to decide if a hyper-link can be created. In this way hyper-links can enhance robustness and survivability. With this approach it is possible to construct one or more database / blackboard manager agents and theses databases / blackboards may either be internal to the agents that access them or distinct remote information management agents.

Implementation Languages and Systems

An object-oriented language ABCL/1 (Yonezawa, 1986) is well suited for the implementation of a distributed system. The language incorporates different type of message passing which can be used for testing and comparison of different methodologies. The message passing structure also relates the priority of messages and can be used to maintain object structures for emergencies, i.e. if an emergency packet need priority scheduling express message passing can be used. Using ABCL/1 the semantics of the message parsing among object is transparent and faithful to the underlying computational model. Objects can be created dynamically, message transmission is asynchronous, and the network topology can change dynamically. There is also concurrent activation's of independent objects. Mechanisms for synchronisation are provided for. The language can cater for such mechanism that have not been explicitly provided for, i.e. through the use of compound scripts, mechanisms

can be created for totally new control strategies such as multi-stage negotiation and control structures for **PGPs**.

For a more comprehensive system to work a number of capabilities must be added to ABCL/1. These capabilities include the ability to build different systems at different levels of granularity. The language must also have the ability to map agents to processors, handle inter-agent communication, and provide a language for describing agents, tracing and instrumentation, a facility for remote demons, and a collection of system agents which construct user-agents from descriptions, monitor execution, handle errors, and interface to the user. A system such as MACE (Gasser, 1987) could be used to handle all of the above requirements. MACE was designed as a testbed for DAI experiments. As explained by Gasser et al. (1987) MACE is used to implement both distributed blackboard system as well as contract net systems.

Another system that offers these benefits is Agora (Bisiani, 1986). Agora is useful for systems that are heterogeneous, in rapid development and are computationally expensive. Agora supports heterogeneous systems by providing a virtual machine that is independent of any language and allows a number of different programming models and can be efficiently mapped into a number of different computer architectures. Agora is suitable to development because the complexity of the system can be hidden by building reusable custom environments that guide a user in describing, debugging and running an application. Also the structure of the system can be modified while the system is running. Knowledge sources are described in such a way that allows Agora to match the available architecture and its resources with the requirements of the computer.

An addition to any system is a diagnosis module (Hudlicka,1987) that enables the system to reason about its own behaviour. The aim of the diagnosis is to identify inappropriate control parameter settings or faulty hardware components as the causes of observed misbehaviour.

Blackboard Structures

Conventional blackboard architectures incorporate a shared common data area or blackboard as the common medium for memory and interaction among a collection of knowledge sources. Flexibility and efficiency are often conflicting goals in a blackboard system. Both flexibility and efficiency can be achieved by maintaining a sufficient level of abstraction between the application code and the blackboard implementation. The techniques needed to do this are available in GBB (Corkill, 1987). Flexibility in a blackboard system is the ability to change the blackboard database implementation.

COPS (Corkill, 1987) is an environment for distributing problem solving. It provides the facilities for distributing blackboards and problem-solving demands processed over the networks of heterogeneous computers. Set of rules, called ambassadors can be placed at each blackboard, to handle matters of interest to remote processes. With the introduction of concurrent execution in a blackboard system, new mechanisms have to be built for scheduling knowledge source activities, synchronising knowledge source interactions, and accessing shared data. This can be done using a transactional blackboard architecture (Ensor, 1986).

Multi Agent Planning Literature Review

An AI planning system is charged with generating a plan which provides one possible solution to a specified problem. Higher life forms must be able to anticipate the future and form plans of action to achieve their goals. Reasoning about actions and plans can thus be seen as fundamental to the development of intelligent machines that are capable of dealing effectively with real-world problems. Agents planning or deciding on possible actions can choose from an enormous repertoire of actions and these in turn can influence the world in exceedingly complicated ways. The world can change its state only by the occurrence of an event or action. The basis of planning is to find, out of all possible actions that an agent can perform, which, if any, will result in the world's behaving as specified by the goal condition. Thus planning could be viewed

as a straightforward search problem. There are some problems with designing an efficient planning system :

- Representing and reasoning about time, causality and intentions.
- Physical or other constraints on suitable solutions.
- Uncertainty in the execution of plans.
- Sensation and perception of the "real world" and holding beliefs about it.
- Multiple agents who may co-operate or interfere

In the following sections a literature survey of existing planning systems will be discussed.

General Planning Systems

Three areas of scientific study were combined initially combined into the development of AI planning systems: human cognition, operations research, and theorem proving. Early attempts at planning tried to combine all three of these areas into a general problem solver.

One of the first general problem solving programs was GPS (Newell, 1963). GPS deals with a task environment consisting of objects which can be transformed via various operators. The problem solver detects differences between objects and organises the information about the task environment into goals. The GPS system led to a number of insights that have continued to influence the systems being built today. The main technique is that of means-ends analysis to direct search.

A problems solver that uses a resolution theorem prover similar to GPS is STRIPS (Fikes, 1971). STRIPS attempts to find a sequence of operators in a space of world models that transforms a given initial model into a model in which a given goal formula can be proven to be true. A resolution theorem prover is used to answer questions about particular models. STRIPS uses means-ends analysis to guide it to the desired

goal-satisfying model. The combination of means-ends analysis and formal theorem-proving methods allows objects much more complex and general than any of those used in GPS and provides more powerful search heuristics than those found in theorem proving programs. In STRIPS the process of theorem proving is separated from the process of searching through a space of world models. Problems arise when the solution is too far away from the boundary of known abilities. This is because the combinatories of search will then make a solution unlikely.

General purpose problem solvers, such as STRIPS and GPS, cannot solve any reasonably complex set of problems in a reasonably complex domain. Attempts to traverse a complex problem space can be caught in a 'combinatorial quagmire' (Sacerdoti, 1973). A problem solver deals not with the problem domain itself, but with some representation of that domain. Unfortunately, a straightforward transcription of a complex problem domain will yield a complex representation. However, a well chosen transcription can lead to a simpler representation. A problem domain can be represented as a hierarchy of abstraction spaces in which successively finer levels of detail are introduced. Thus significant increases in problem-solving power is achieved. The problem solver ABSTRIPS, a modification of STRIPS, can define an abstraction space hierarchy from the STRIPS representation of a problem domain, and it can utilise the hierarchy in solving problems. This approach to problem solving would be first to search through the abstraction space and then when a solution to the problem in the abstraction space is discovered, account for the details of the linkup between the steps of the solution. ABSTRIPS relies heavily on being able to produce good plans at the highest level.

Interactions and Dependencies

The general planners were often called linear planners because the planners produced plans in which the actions were fully ordered. In the synthesis of a plan or computer program, the problem of achieving multiple goals simultaneously presents special difficulties, since a plan to achieve one goal may interfere with attaining others. A single linear plan could be considered a base plan that can be used for tackling a

second goal. Multiple goals lead to interaction in plans. It was recognised that interaction problems resulted from a poor choice of the order in which goals were tackled. This led to research on problems arising when a second goal interfered with the successful achievement of the first goal.

Waldinger (1975) proposed a method whereby two goals can be achieved simultaneously. First a plan is developed to achieve one of the goals and then modified to achieve the second goal as well. A systematic program modification technique is used to support this strategy. The technique required the introduction of a special "skeleton model" to represent a changing world that can accommodate modifications in the plan. The skeleton model is a framework in which the partial plans that has been constructed is regarded as a central part of the model.

The general strategy proposed was " in order to construct a plan to achieve P and Q. Construct a plan to achieve P, and then modify that plan to achieve Q as well. In the course of the modification, the relation P is "protected": no modifications that might make P false are permitted. If no satisfactory modification is found, the same strategy is attempted with the roles of P and Q reversed". This practice of retrieving techniques according to the form of the goal and then trying them each in turn until one is successful is called "pattern-directed function invocation". A problem solver organised around these principles can only be aware of only one goal at a time.

Regression plays an important part in Waldinger's (1975) program modification technique. Important relationships and the plan itself are modelled explicitly while other relationships may be inferred using regression rules. Although one serious limitation of Waldinger's (1975) method is that the only way to modify plans is to add instructions to the plan, there is no way to modify existing instructions. By forcing the plan to be linear at all intermediate stages the planner must do unnecessary search with backtracking, or sophisticated plan optimisation to find the correct order in which to attack conjuncted conditionals. An alternative method to the "skeleton model" is the procedural net information system.

The procedural net (Sacerdoti, 1975) is a information structure that represents a plan as a partial ordering of actions with respect to time. By avoiding premature commitments to a particular order for achieving subgoals, interaction problems may be solved. A plan in this approach may be thought of as a partial ordering of actions with respect to time. The procedural net is a collection of nodes which are linked to form hierarchical descriptions of operations, and to form plans of action. Each node points to a body of code which if simulated reveals the action of the node. Initially the planner builds a procedural net that consist of a single goal node to achieve the given goal. This one-step plan may then be expanded. The planner expands the procedural net by expanding each node of the plan. The nodes are expanded in order of their position in the time sequence. Thus by creating subplans for each node in the plan, a new, more detailed plan will be created.

Before this detailed plan is presumed to work, the planning system must take an overall global consensus. This global examination is done by critics. These constructive critics add constraints to the plans. There are specific critics for certain areas such as "Resolve Conflicts" critics which examine portions of the plan that represent conjuncts to be achieved in parallel. There is the "Use Existing Objects" critic which will replace formal objects (a variable object) by real objects wherever possible. This approach attacks conjunctive goals by making a "linear" assumption. That is, conjunctive goals are assumed to be independent and additive, and so to achieve the overall goal each conjunct may be achieved in sequence. As plans fails a collection of critics can be developed to prevent a recurrences of failed plans. The planning system does not deal with disjunctive subgoals and also fails to deal with "non-linearizable interactions". MOLGEN (Stefik, 1981) also uses a least commitment strategy for handling interactions.

Non-hierarchical programs suffer from the tyranny of detail. Abstraction, which is the basis for hierarchical planning, is a way of suppressing detail. A hierarchical planner distinguishes between important considerations and details. MOLGEN uses an

approach to hierarchical planning termed constraint posting. MOLGEN uses constraints to represent the interactions between sub-problems. Constraints are dynamically formulated and propagated during hierarchical planning. These constraints are used to co-ordinate the solutions of nearly independent sub-problems. The constraint posting approach depends on the view of systems as aggregates of loosely coupled subsystems (Stefik, 1981). MOLGEN model the design of its systems in terms of operations on constraints. The power of constraint programming comes from the ability to plan hierarchically by introducing new constraints and variables and the ability to anticipate interference between sub-problems and to eliminate the interfering solutions. Constraints may be interpreted as conditions to be satisfied or as partial descriptions and commitments from the perspective of plan refinement. Constraints can also be interpreted as a communication medium for expressing interactions between sub-problems. There are three basic operations on constraints:

1. constraint formulation,
2. constraint propagation,
3. constraint satisfaction.

The first is used for adding new constraints. The second for creating new constraints from old constraints and the third is the operation of finding variables so that a set of constraints on a variable is satisfied. MOLGEN has the ability to formulate constraints dynamically as it runs. MOLGEN has certain problems, first is that it has no ability to propagate logically equivalent variations of constraints it creates and it also does not use constraints to describe processes.

Planning and Acting

The planners described previously have a certain basic assumption: All change in the environment must be knowable to the planner (so that the final stage after planning can be demonstrated to hold true). As planning domains get more complex, allowing other agents or outside forces to introduce change (violating the above assumption), the notion of separate generation and execution of plans cannot be guaranteed to work.

Certain actions cannot be represented as a change of world state. Therefore a planning problem then becomes a task whose actions cannot be immediately carried out. Reducing such a task amounts to finding a collection of sub-tasks such that the problem task may be carried out by doing all the sub-tasks. If execution is included in the problem solving process, then at some point one plan must be selected to guide action in the "real world". If it fails, the state of the world will in general have been changed enough by the attempt so that the alternative plans are out of date. A new addition to the STRIPS robot problem system overcomes this problem.

Fikes, Hart and Nilsson (1972) propose a addition to the STRIPS system which allows a robot to react "intelligently" to unexpected consequences of actions. This is done by replacing problem specific constants by problem independent parameters. The generalised plan is stored in a format called the triangle table. A triangle table is a lower triangular array where rows and columns correspond to the operators of the plan.

Through the use of this table two benefits are derived. The first is as a "macro action" (Fikes, 1972), that can be used as a single component of a new plan to solve a problem and the second allows robots to react "intelligently". The plan execution scheme used with the triangle table has two properties. When new information obtained during plan execution implies that the remaining portion of the plan need not be executed. The executor should recognise such information and omit the unneeded plans steps, and the second property is when execution of some portion of the plan fails to achieve the intended results, the executor should recognise the failure and either direct re-execution of some portion of the plan or as a default, call for a re-planning activity. Through the use of a triangle table a problem solving program can "learn" old solutions and use them both to monitor real-world execution and to aid in the solution of new problems. Mathematical decision theory can also be used to make decisions under conditions of uncertainty.

The central idea of mathematical decision theory is that a numerical utility function can be used to evaluate decisions. Decision theory helps a symbolic-problem solver search for the best plan that achieves a given goal. Therefore a integrated problem-solving framework would incorporate aspects of mathematical decision theory into symbolic problem-solving techniques (Feldman, 1977). Using numerical measures the utility of a plan can be used to compare the merits of competing plans, it can be used as a guide a search for good plans. In this method search is used for all bounding and measurement of plans. The problem with this technique and search in general is the ability to generate a good utility to guide the search process. Therefore techniques for guiding searching of plans is appropriate for maintaining a balance between action and planning. The solution is to face up to the necessity for treating "choice between alternatives" as a basic situation of problem solving (McDermott, 1978). Since there is no general criterion for such a choice, the choice process is reliant on the underlying mechanism of the problem solver.

NASL (McDermott, 1978) was developed to use heuristic means to provide choice rules for a search planning process. Through the use of certain notations for talking about problems and plans and a problem solver which operates by manipulating a data base containing formulas in the notation, NASL forms a basis for a heuristic problem solver that copes with uncertainty. An extension to the system NASL, SCRAPS (Hendler, 1985) is based on marker passing.

It is possible through the use of marker-passing to approximate a system that can examine the primitive actions of a plan quickly, and in parallel, while looking for special features. Integrating marker-passing within a problem solver enables a system to avoid backtracking. The marker-passer is used to choose correct plans, reject plans violating constraints or to modify existing plans. Marker Passing, a computational model of spreading activation, can be though of as the marking of nodes adjacent to some node in memory, and then marking all nodes adjacent to those. By marking first one node and then another we can see how they are related by examining those

concepts marked by both. Marker Passing is a very powerful paradigm with certain properties. It is able to examine many possibilities efficiently. It returns information that the problem-solver uses for making choices.

Formal Models of Action

As shown in the previous chapter, action, must be considered part of the essential framework of a problem-solver. Action should therefore be formally defined in terms of the planning model. A general framework would be concerned with the representation of action within the larger context of general knowledge representation and does not generally specify practical methods for building planning systems. Rather, formal models attempt to stretch the limits of what situations can be represented and to show how complex knowledge about action can be. The most influential formal model for planning is situation calculus.

Situation calculus introduced the first temporal representations of plans. This was accomplished by indexing each proposition by the situation in which it was true. Through situation calculus a general model of "action as state change" was presented which is formalised by the "result" function which maps an initial situation to a resulting situation. Situation Calculus has been used as a framework for much research. McDermott (1982) develops a representation of action retaining the ideas of situations, but adds a rich temporal structure.

McDermott defines a model by continuous change by using a temporal model in which situations correspond to numbers on the real number line. This first-order temporal logic provides analyses of causality, continuous change in quantities, the persistence of facts, and the relationship between task and actions.

In addition, it allows indeterminacy by using a branching time model ("chronicle tree") to represent the set of possible futures. Temporal intervals are constructed as sequences of all the situations between two other situations, and event types are then

represented simply by the set of all intervals over which the event occurs. Because of this explicit temporal model, McDermott has no problem stating that two actions or events may happen simultaneously and has no difficulty representing possible future states that may have to be considered in order to construct a reasonable plan. Allen (1984) also introduces time to define the notion of action, but takes a different approach from situation calculus.

Allen takes the notion of a time intervals as primitive and builds a representation directly from intervals rather than from a notion of a situation. This formalism can be used to characterise the different types of events, processes, actions, and properties that can be described in simple english sentences. Allen represents events as the set of intervals over which they occur and shows that simultaneous actions and future external events can be represented. Rather than use a branching time model, Allen uses a linear time model, which allows for only one future, and lack of knowledge about the futures is used to simulate indeterminacy. Due to this certain problems such as standing still can easily be modelled. Although, there are still problems with these formal models such as building a system that can reason about interactions, with temporal logic, still remains a difficult problem.

Formal Models of Planning Systems

Without the formalization of the actual planning systems, a problem solver would not be viable. Formal models give the ability and techniques needed to develop actual planning systems and the ability to classify existing systems. A formal model would also indicate whether a given model is sound or unsound.

Lifschits (1990) gives a formalization of one on the most general planning systems STRIPS. STRIPS is a problem solver which operates with world models represented by a set of formulas of first-order logic. A STRIPS system is defined by an initial world model and by a set of operators. The description of each operator consists of its precondition, its add list (the list of formulas that must be added to the current world

model), and its delete list. The formalization of STRIPS is a much more complex task than it first might appear because the operations of STRIPS, namely, adding and deleting formulas from the database, cannot be captured directly as logical interference. Therefore any designer of a STRIPS systems must be vary careful about what operations they allow in order to avoid deriving incorrect results. For example, a designer cannot possibly include in a delete list all arbitrarily complex formulas that may become false after the corresponding actions is performed. Lifshitz (1990) defines semantics for these operations that allows the notion of soundness to be defined and yields a precise definition of the "good" uses of such a system.

Ronsenchein (1981) explores some theoretical issues of robot system planning from the perspective of propositional dynamic logic. A generalised notation of "progression" and "regression" of conditions through action is developed. Rosenchein (1981) develops a framework which can be used to clarify gaps in existing "nonlinear" and "hierachical" planning strategies. Actions are formalised as state change, as in situation calculus, but within the dynamic logic framework as developed for the semantics of programming languages. A formal definition is given of planning problem and then defines what it means for a plan to solve a problem.

Allen and Koomen (1983) develop a different approach to non-linear planning. Rather than maintaining a partial ordering of states, actions and the necessary consequences, it is shown that all this information can be stored in Allen's interval logic. This approach allows the problem solver to gather constraints on the ordering of actions without having to commit to an ordering when a conflict is detected. Beside being able to explicitly represent simultaneous actions, they show that that the resolve-conflict critic could be formalised simply as a procedure for maintaining the consistency of a temporal database. A STRIPS like action formalism is used to create a more generalised world model with the ability to be used by existing problem-solving methods i.e. means-end analysis, decomposition.

Korf (1987) views problem solving as search using subgoals, macro-operators and abstraction as knowledge sources. Korf classifies the different forms in which subgoals my interact and how these forms affect the heuristic search. From this research, results include the identification of sub-goal distance as a fundamental measure of problem difficulty, a multiplicative time-space trade-off for macro-operators, and a analysis of abstraction which concludes that abstraction hierarchies can reduce exponential problems to linear complexity.

Time and the Frame Problem

The most important formal problem facing planning systems that use a rich representation of the world is the frame problem. This term generally refers to reasoning about how the world changes as actions occur. Specifying the direct effects of actions is not so problematic, the frame problem arises from the need to specify what items in the world may or may not be indirectly affected by the action.

Shoham and McDermott (1988) identify the general form of certain classes of problems that arise in formal temporal reasoning. It is argued that the frame problem arises from the conflicting desires to reason both rigorously and efficiently about the future and that this conflict does not depend on the particular underlying temporal formalism. Two formalism-independent problems are identified, called the qualification problem and the extended prediction problem, which subsume the frame problem. These two problems, unlike the frame problem, do not depend on any particular underlying formalism.

Hayes (1975) is one of the first papers to specifically address the frame problem. The paper contains a survey of the problem, showing how it arises in several contexts and relates it to some traditional problems in philosophical logic. Partial solution to the frame problems are outlined and compared. Hayes (1975) formalises a system of "frame-rules" that are used to partially overcome the frame problem. The frame-rules are a attempt to formalise solutions to the frame-problem. Although maintaining consistency when defining the "frame-rules" is still a matter of the axiom-writer's art.

Learning and Reuse

Learning is concerned with the ability to use past experience to help modify behaviour. An example of this is a STRIPS system which could learn to group sets of operators into single "macro-operators" which could then be used in new planning situations.

Hammond (1986) describes a case-based approach to planning. In such a system a old plan is chosen and modified to run in a different situation. This work concentrates on guiding the search for the old plan and then using the mapping strategy to produce a new plan. Memories of past successes are accessed and modified to create new plans. Memories of past failures are used to warn the planner of impeding problems and memories of past repairs are called upon to tell the planner how to deal with them. The ideas of memory, learning and planning are implemented in the case-based planner CHEF, which creates new plans in the domain of Szechwan cooking.

The work on CHEF concentrated primarily on finding relevant plans. Another important issue is how the plans that are retrieved can be reused in the current situation. The PLEXUS (Alterman,1986) system uses information about a new context to guide the reuse of an existing plan. PLEXUS is an adaptive planner. An adaptive planner can take advantage of the details associated with specific plans, while maintaining the flexibility of a planner that works from general plans. Adaptive planners foreground specific plans, but gain flexibility, in situations where the old plan and the planner's current circumstances diverge, by having access to more general plans. Adaptive planning works with increased amounts of knowledge and achieves these twin goals by exploiting the structure of that knowledge.

Extending the Classical Framework

Because of the need for problems solvers to handle conditional iteration, multi-agent co-ordination, and complex domain interactions many of the classical planning frameworks have been extended.

GEMPLAN (Lansky, 1989) works by utilising heuristics that exploit the naturally occurring structures within a domain. The planner uses "locality" or domain structure, to provide a way of partitioning the planning search space into smaller, localised search spaces. This is motivated by the need to reduce overall complexity of planning systems. The domains structural semantics are used to guide reasoning about regional interactions and to cope with the problem of re-planning in dynamic environments. GEM models a domain as a set of interrelated events clustered into regions of activity (Lansky, 1989).

Pednault (1990) describes a method how some multiagent, dynamic-world planning problems my be formulated as single-agent, static world problems. Thereby permitting them to be solved with techniques originally intended for the latter. This approach requires that dynamic worlds be modelled in the same manner as their static counterparts, and that simultaneous actions be modelled as sequential actions. The former is accomplished by introducing time as a parameter in the description of dynamic worlds. A boundary conditions is used to model simultaneous conditions as sequential ones. A boundary conditions determines the course of events up to the next boundary conditions. The effect of simultaneous actions is then modelled by modifying boundary conditions sequentially. Pednault introduces a language, ADL, that combines the notational convenience of the STRIPS operator language with the semantics and expressive power of the situation calculus.

Planning and Execution

Rapidly changing environments need to be handled by certain mechanisms. These mechanisms work by giving up complex planning for shallow planning or by planning using tightly coupled sensing and action.

The system PRS (Procedural Reasoning System) (Georgeff, 1987) was created with the ability to operate under real-time constraints. This system consists of a

database containing current beliefs or facts about the world, a set of current goals or desires to be realised, a set of procedures describing how certain sequences or actions and tests may be performed to achieve given goals or to react to particular situations, and an interpreter for manipulating these components. PRS uses a partial planning strategy, procedural knowledge and metalevel (reflective) capabilities. The partial hierarchical planning strategy and the reflective reasoning capabilities of PRS work well under real time constraints, yet still allow the system to plan ahead when necessary. Because PRS expands plans dynamically and incrementally, there are frequent opportunities for it to react to new situations and changing goals.

In the following section a more detailed look at the Datilis control structures will be discussed using many of the basic rationales and structures from the literature survey.

Research and Methodology

Research

This research is based on the ideas of a dynamic system of interacting agents or as it is better known, a multi-agent system. The main goal of this research is to show that the control structures for a dynamic system can be built within a economical structure and can be utilised as a viable alternative to a static system or one based on the principles of operations research and queuing theory. Through the use of a dynamic system, 'data' can be modified and handled at the data points (intelligent agents). In this way each agent can have localised control with a centralised command structure in place to direct the complete system.

There are two main distinct sections to this project. The first is an objective look at using a distributed control structure in a live system and comparing the results to a base static system. In this way a true objective comparison can be made between the two systems. The second stage has to do with the uncertainty of a system. If a traffic light breaks down and an intersection is unusable most cars in a non-dynamic system would either be stopped or be forced to take "unfamiliar" routes around the intersection. With a dynamic system a car could be re-routed according to both the destination of the car and the most viable routing information available to the traffic light system. These two sections will be discussed in further detail later on.

Due to the fact that no "live system" could be used for testing a simulator was constructed and used as the basis for the experimentation. The simulator had to have a number of basic design goals to make it a viable working model for the research. The methodology of modelling a problem and then obtaining results is similar to the operations research methodology. The model or simulator had to have a number of basic design methodologies to make the results as valid as possible.

- The first goal was to have a system that simulated the distributed system as closely as possible to the original system i.e. all agents were given equal time, there is no definable biased towards any one agent. All Parallel activity is handled correctly from the agent's viewpoint.
- The distributed simulator should handle all distributed events evenly. I.e. if events are generated they are handled for the time period they were generated for rather than in the order the events were generated. In this way the fairness of the previous clause is ensured. All Parallel activity is handled correctly from an event's viewpoint.
- The simulator should be as extensible as possible. Due to this an object oriented approach was used in the design and implementation of the system. In this way the model could be used to validate other techniques using the same platform.
- The system should be as generic as possible. In this respect the simulator can be used for other areas such as networking etc. while the control structures stay the same. This is also a research goal to show that the DAI techniques can be used within other areas with limited complications.

Problems arise with the specifying of a "complete simulator" and the frame problem (Shoham, 1988). The traffic light system is an open system. It is not possible to compensate for all unknown variable in the representation of the system. This means that as complete as the simulator is, it will never accommodate all the necessary information to give a full simulation of an actual traffic light system. The only way to get fully accurate results is to use a actual traffic light system as a testbed. All simulation models have the same limitations within the bounds of the frame problem.

A number of compromises were made when building the simulator. The reasoning behind this was two fold. Because of the size of the system, the simulated area was reduced to a workable level. A limited number of intersections were used. The more intersections the higher the complexity of the system and the longer the simulation takes.

Certain facts were assumed to be static which is not the case in the real life system. An example of this is the time taken between traffic lights. In real life cars speeds can vary between intersections. Cars generally slow down gradually when approaching a red light as this gives the light the opportunity to change colours or speed up to just catch a robot. In the simulator the "average of all times" for the section of road is assumed and this time is constant for all cars i.e. all cars take a set time to travel any section of road.

The simulator was built up with the concept of "object" extensibility. Each component of the simulator is built upon a "removable object", with very well defined interfaces between the object and the main control loop. Each object has the ability to totally change the 'thinking' patterns of the complete system or rather to alter the structure of the total system. Through the use of objects corresponding to different constructs the simulator has the ability to reconfigure almost any part of itself. This means if needed the simulator could be used to simulate a contract net as opposed to the blackboard structure that was used.

In the next section a more detailed look at the specific Datilis system is given.

Datilis

DATLIS (Distributed Traffic Light Simulator) comprises of a number of intersections (logical agents) interacting in a matrix network. Each intersection has a maximum of four and a minimum of two connections. All intersections have the ability to be "fully intelligent" and therefore have the capacity to both change the route of the cars and the timing of the robots at each intersection.

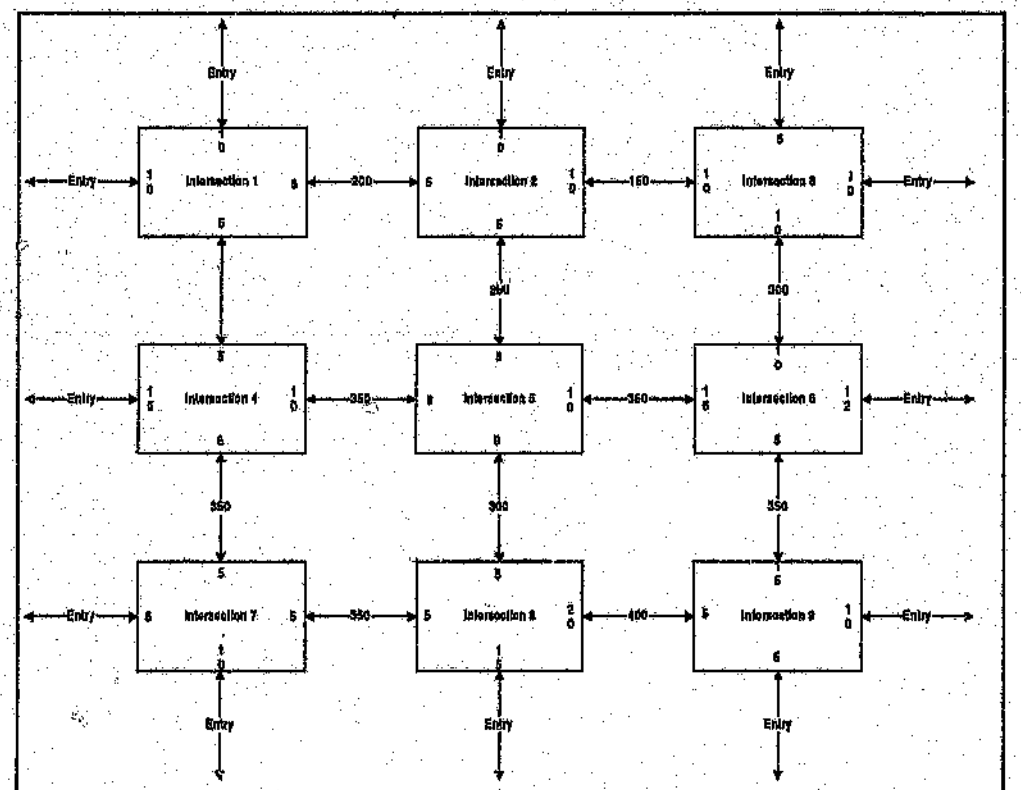


Figure 1: An example of a traffic grid

Figure 1 is an example of a traffic grid that contains nine intersections. All timing values for this example were randomly generated. Randomly generated values present a problem in simulating a real traffic light system. Timing for the robots would not be totally randomly generated in a real life situation but based on the flow of cars from a high concentration of cars to one of lower concentrations i.e. a centre of a city has high volume of cars that move outwards towards the suburbs.

Unlike a real traffic light system Datilis utilises a rotating-time control system whereby each intersection's traffic lights have a limited time in which to allow cars to travel through the intersection. Both the time for each connection and the order in which the connections are handled by each intersection is fixed. In this way all robots in the intersection can have totally independent times and both turning and traffic flowing in single directions can be more easily controlled and enhanced i.e. Figure 2.

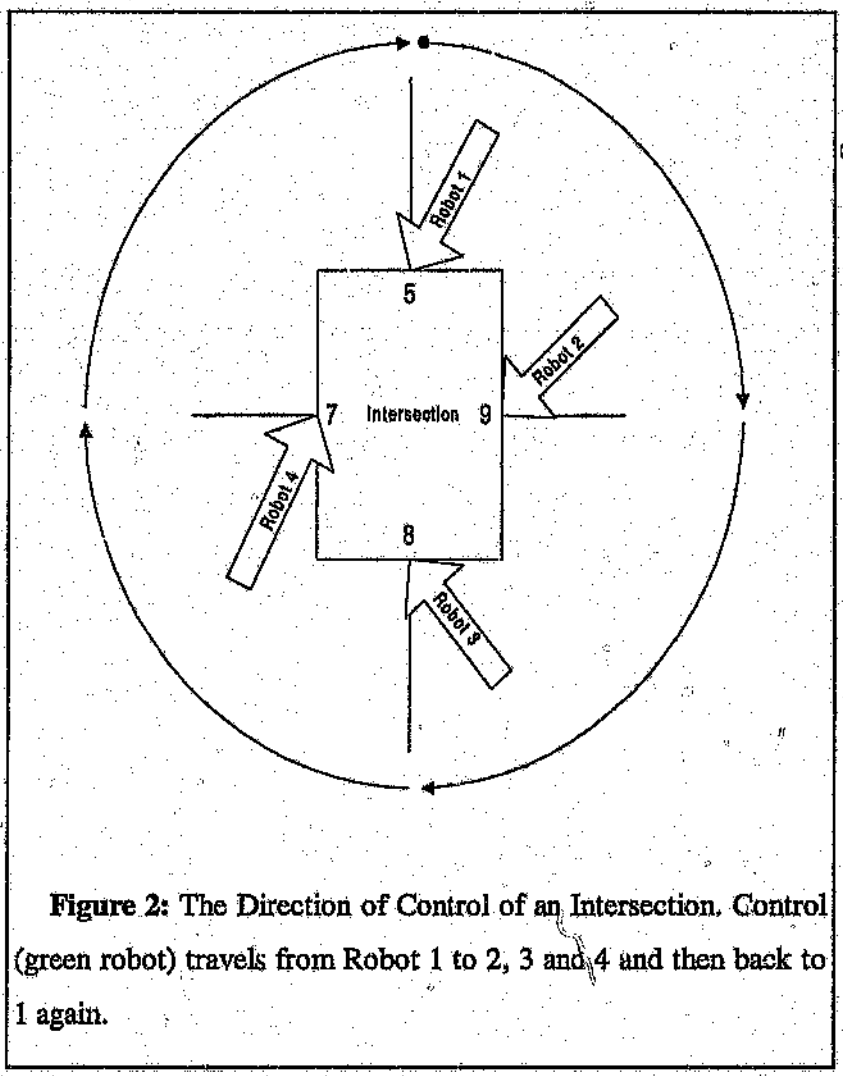


Figure 2: The Direction of Control of an Intersection. Control (green robot) travels from Robot 1 to 2, 3 and 4 and then back to 1 again.

Problems arise from this flow control method when cars travel on average in straight lines. In the usual flow control method i.e. when robots on both sides of an intersection are green, double traffic would be permitted through the intersection and cars would turn when both the facing traffic light . . . reen and there was no car crossing the intersection from the opposite direction. The advantages of the simulated flow control is the ability to control cars that turn in the intersection. With only one robot green at a time, all cars travelling through the intersection have right of way and can turn immediately without waiting for traffic to clear. This does not effect the simulated results as both the base results and test results were generated using this

same flow control. Although this still does abstract the simulator from the real-life situation by a small margin.

Using "intelligent" enhancements in each intersection of Datilis, makes both the timing constraints of the traffic lights and routing of the cars dynamic. The areas of communication and co-operation become essential to the functioning of the total system. With the addition of communication and multi-agent planning (for co-operation) the total complexity of the system increases proportionally to each attribute added. The intelligent enhancements used by Datilis are based upon DAI techniques and are discussed in more detail when explaining Datilis's control structures.

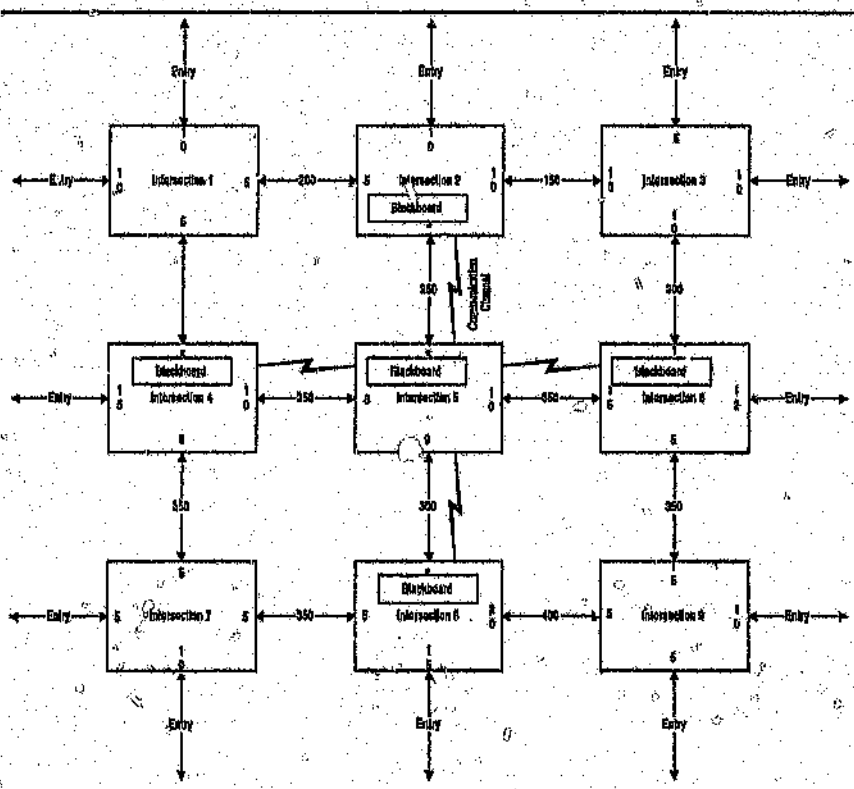


Figure 3: The basic conceptual overview of Datilis showing both the blackboard placement and the added communication channels between intersections.

Datilis Control Structures

Datilis is built up through a number of objects that correspond to structures in a real life traffic system, plus objects that correspond to the structures needed for the distributed control structures. There is generally one single agent that controls each intersection. These agents control an intersection's traffic lights. Agents are used as the main control structures for Datilis and are a very important part of the design of the simulator.

For communication and co-operation a distributed blackboard system was used. Due to the nature of the distributed traffic light system a single distributed blackboard is unfeasible. A single blackboard would have to contain information pertaining to the flow of all cars in the entire simulator. Therefore a single distributed blackboard would be unable to control many of the distributed dependencies in a real time situation. One of the basic design goals of Datilis was a system that is feasibly possible to implement as a real-time system. It was decided to implement a distributed blackboard system along the lines of the physical nature of the traffic light system. Therefore in each intersection blackboards were inserted to keep information relating to the local flow of cars of the intersection. To build up area information blackboards store and share extra information with the blackboard of the nearest neighbour. An external communication channel was setup between all nodes to facilitate in the exchange of the blackboard information. A graphical example of this structure is given in Figure 3.

Local blackboards contain information that pertains to itself and its nearest neighbours. Each blackboard contains all information about each traffic light that the agent controls. The information stored is the arrival time of the car from the previous intersection and the waiting time in the current intersections queue. Neighbouring agents have open communication pathways to only other neighbouring nodes. In this way, as cars move physically from intersection to intersection the information relating to that car is transferred electronically from blackboard to blackboard. These

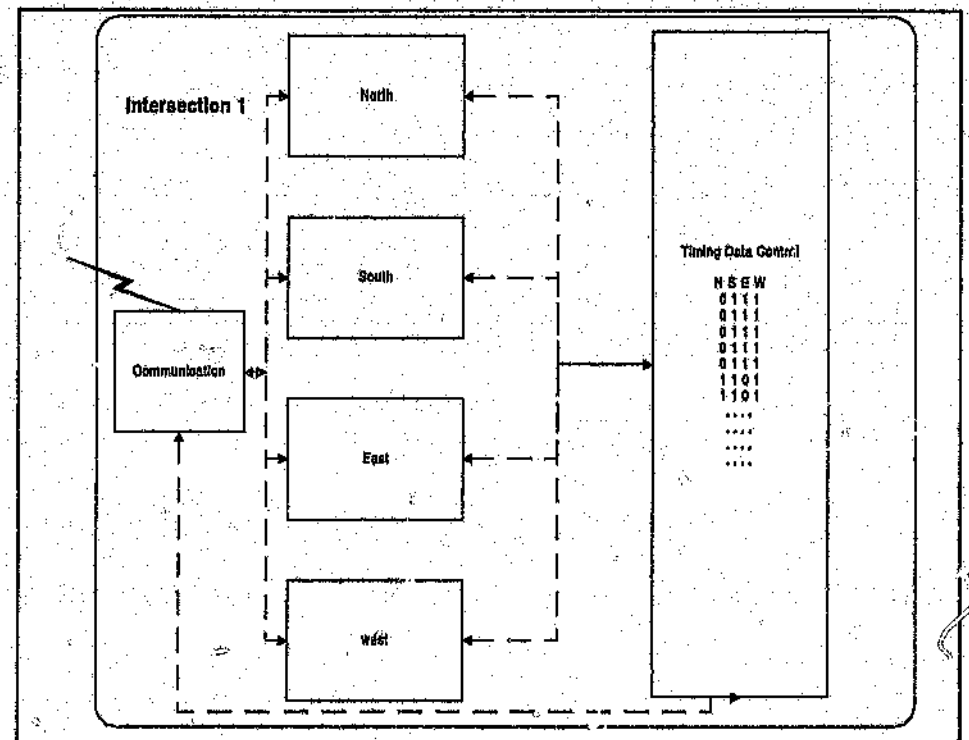


Figure 4: The conceptual overview of a single agents blackboard. This diagram shows the structural breakdown of a blackboard. All flow information is shown. Information flows from the min-blackboards North, South , East and West to the timing data control structure and then back again.

communication pathways are always connected and hence each neighbour also has the ability to request information from its neighbour.

Global information of the system can be built up using information from each blackboard in each intersection. Because all the blackboards hold information pertaining to its nearest neighbours there is a redundancy of information that can be used to overcome breakage's in the network. Each blackboard also has the capacity to maintain two blackboards and hence if there is a physical fault with an agent, the agent blackboard can be replicated by neighbouring agents.

As shown in Figure 4 above, the black board structure is divided into five main areas. The North, South, East and West mini-blackboards store the information on the incoming cars relating to the corresponding traffic light direction. Cars are inserted in a first in-first out (FIFO) queue in these mini-databases. Overtaking is not permitted in the simulator and hence cars are always queued in the same order that the cars came in as. There is one extra structure used by the agent, the timing data control structure. This is the heart of the agents decision making ability and is discussed in more detail in the section "Timing Data Control Structure".

The structure of Datilis is similar to the hyperstar architecture (Cromwell, 1986). All blackboards are connected in concentric structures while Datilis still has the ability to insert higher level controlling agents (not intersections) above the current intersection's agents. As shown in Figure 5 standard communication takes place in fixed pathways. Unlike the hyperstar architecture, Datilis has no central agent controlling all the agents.

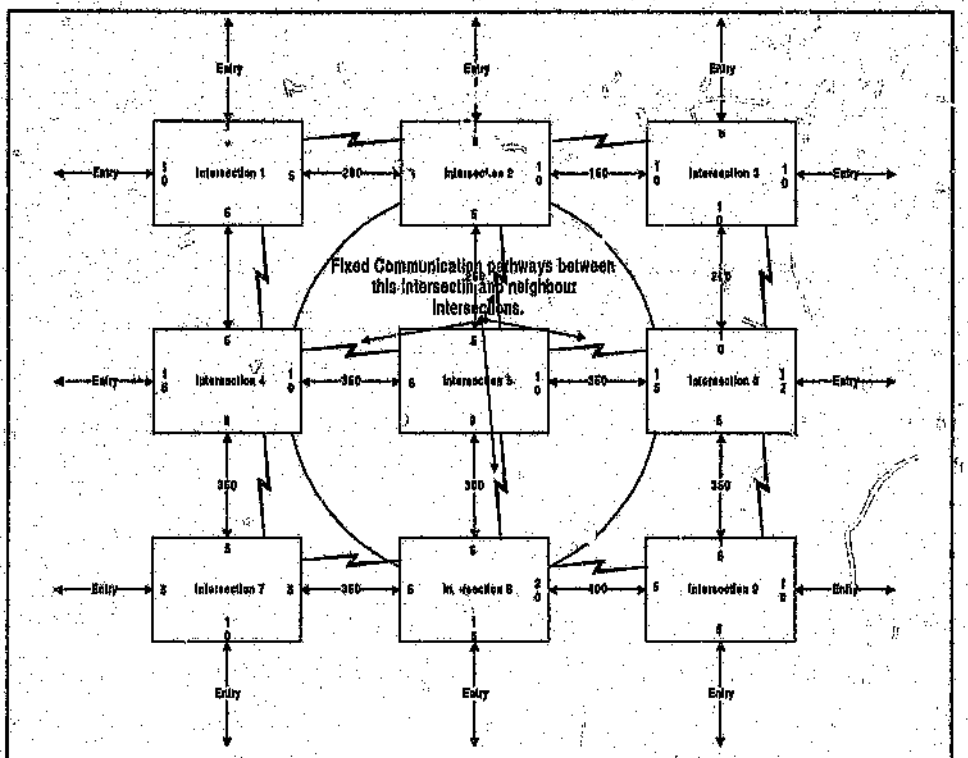


Figure 5: A diagram showing the fixed communication pathways in the Datilis Simulator. The circle (in the centre of the diagram) touches all neighbouring nodes of the centre intersection, intersection 5. Intersection 5 can only communicate with these neighbouring nodes.

Built into the system is also the ability in emergencies for an intersection to communicate with any other intersection. This communication is over and above the fixed communication pathways used in standard communication model i.e. Figure 5 and Figure 6. This added emergency communication model gives the network the ability to handle emergency vehicle's and any major intersection breakdowns. As an emergency vehicle travels through the intersections messages can be sent from the intersection to all intersections on the route to the vehicles destination. This communication ensures that all robots are green along the route and thus give each intersection along the route adequate re-planning time to cope with the cars waiting at the intersection. In the case of a intersection breakdown and a intersection's blackboard ceasing to function, neighbouring nodes could communicate directly with each other and work-around the inoperable intersection.

With the ability to communicate with any intersection certain problems pertaining to contract nets become essential e.g. synchronising communication. For example if intersection 1 sends a message to intersection 3 then intersection 1 must first know the physical address of intersection 3 i.e. where to send the message. In the standard communication model each intersection knows the physical addresses of all neighbouring intersections. When communicating outside of these neighbouring intersections in the emergency communication model an address server is needed to co-ordinate all special communications.

The address server is a special agent in the simulator i.e. it keeps track of all live intersections and automatically forwards all messages to relevant intersections. The address server maintains a level of indirection between the communication network and the intersection agents in the network. In this way the physical location of a agent may change in the network but the means of sending communication is still through the address server.

Each agent in Datilis acknowledges all special communications (i.e. please send routing information messages) and also periodically each intersection sends a global message informing the address server that its is "alive" and its routing information for its messages. All the routing information is stored in the address server. If there is no acknowledgement from any agent a message is sent directly to that agent. This occurs three time. If there is no acknowledgements the intersection is assumed to be "dead" and unusable.

The communication channel in each agent in Datilis is a global device and can be used via various protocols. Each agent's communication object inherits a number of basic design philosophies and abilities. Also due to the object orientated nature of the design of the simulator, each agent could also have plug in modules, in this way making each agent individually suited to its needs i.e. a agent may have different abilities depending on its position and placement in the grid. Each agent must have a

object that describes the basic communication protocol. While the communication module has the ability to transfer 'basic strings' from one agent to another the agent must have the ability to recognise the 'strings', decode and obtain the relevant information from the messages. Also each agent has the ability to dynamically swap in and out different modules if needed during a execution run (in real-time).

In conclusion Datilis is very extensible. Datilis's objects attempt as much as possible to encapsulate all items relating to the real traffic lights physical objects. Each function designed for each object can be re-written for a physical real-time system with a minimum amount of hassle. Communication and agent control is facilitated through the use of an address server. A level of indirection is maintained to cope with the uncertainty and dynamic nature of the system.

To utilitise the knowledge encapsulated in the blackboard a hybrid technique was developed. This technique uses the timing data control structure to modify the time duration's of each traffic light. This technique is discussed further in the next section.

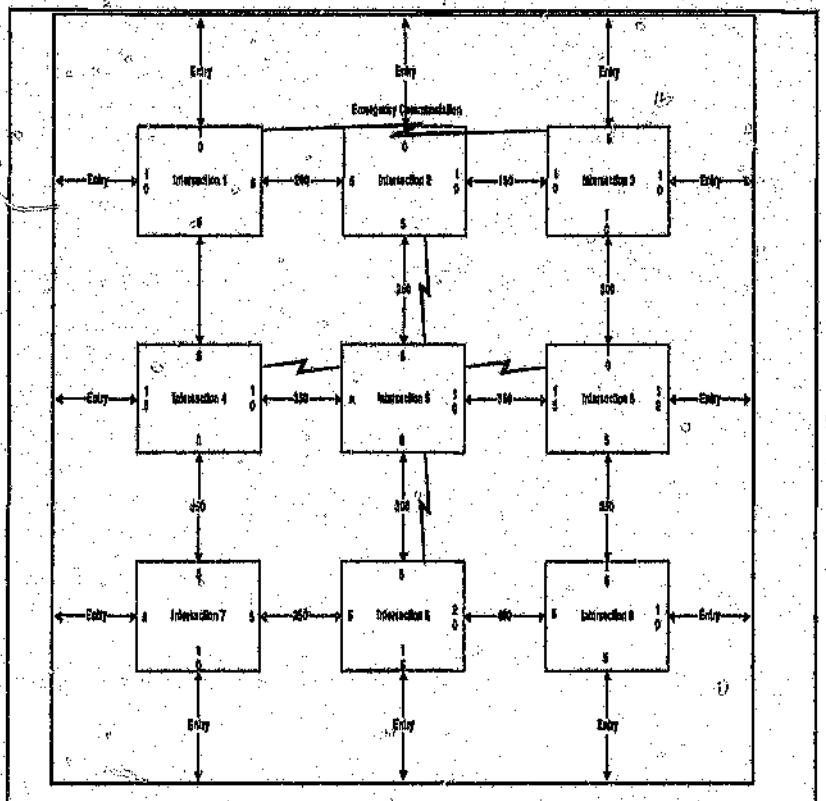


Figure 6: Emergency communication between Intersection 1 and Intersection 3. In general Intersection 1 would only be able to communicate with Intersection 2 and Intersection 4.

Timing Data Control Structure

Through the use of the information available in the local blackboard's the planning module of each intersection will use a new hybrid technique called boundary bashing and pegging. Boundary bashing and pegging is used to cut down on the combinatorial quagmire (Sacerdoti, 1973) of the search area available to the planning module.

Boundary bashing and pegging is a hybrid technique based upon the ideas presented by Pednault (1990) and many of the ideas presented in the planning and acting section of the literature survey. A technique was needed to constrain the planner to areas of

interest, with the ability to find and then 'forget' the area of planning in a real-time environment.

The ideas behind boundary bashing and pegging come from a conglomeration of ideas gleamed from the multi-agent literature review. Two papers contain the essential ideas behind the final system that was encapsulated. The idea of pegging was based on a paper on SCRAPS (Hendler, 1985). SCRAPS uses a system called marker passing. Marker passing is used to mark adjacent nodes in memory. By marking first one node and then another we can see how they are related by examining those concepts marked by both.

In Datilis the control of the intersection occurs in cycles. A single cycle can be thought of as the control being given to the north facing traffic light i.e. turning green , passing through the next three traffic lights and then passed back to the north facing robot. The next cycle occurs as soon as control is passed back to the north facing robot. The idea behind the pegging system is to encapsulate these cycles into 'nodes' similar to the marker passing nodes. Through the use of pegs, adjacent nodes are demarcated and then compared to give the final timing lengths for the traffic lights.

Areas of interest in the Datilis system are the boundaries of the cycles or nodes. Boundaries determine the course of events up to the next boundary condition. This is similar to the use of boundaries as described by Pednault (1990). The constraining of the traffic lights times is done by modifying the boundary conditions sequentially (Pednault, 1990). Boundaries are demarcated through the use of pegs. These pegs are 'bashed' into position through the activities of the planning agent and the information concerning incoming cars which are inserted into the timing data control structure.

Each agent in a intersection uses the timing data control structure of the blackboard for all decision making procedures. Too keep a records of changes over a period of time and to stop unrelated areas of the plan being changed, pegs are inserted into the

boundary areas of the timing data control structure. The timing data control structure of Datilis is time structured i.e. all items stored in the structure are ordered by the time the event will happen.

Pegs have two usage's by the planning agent, the first is a logical use and the second is a physical use. The logical use of pegs is for boundary marking and is usually moved through boundary bashing. The physical use is for the building of a real-time system. A quick method for access was needed into the timing data control structure so decision making and planning could be done in real time. The pegs work as a hashing table (based on time) into the timing data control structure. In this way a direct access scheme was built up to facilitate the ability of acting and planning in real time.

All cycles in the timing data control structure are pegged. This allows Datilis to plan for subsections of upcoming events while previously pegged plans can be executed in real-time. In this respect Datilis plans and acts at the same time, constraining events to certain actions before continuing with further planning. Using pegs as placeholders allows previously "decided" sub-plans to be quickly scanned and re-planned if needed. Pegs allow a significant amount of parallel activity through the use of multiple accesses to the timing data control structure.

Due to the time constraints of the searching engine of the planning agent the boundary bashing and pegging system is limited via time constraints. These limitations facilitate the ability of the system to react in 'real-time' through the limiting of the time taken by the searching mechanism. Physically pegs are restricted to a limit of 60 seconds between pegs. Therefore each cycle in the timing data control structure is limited to a maximum of 60 seconds. This does not mean that a cycle must be 60 seconds long but can rather be any time from 0 to 60 seconds long. A further restriction used to facilitate the 60 second cycle is that each traffic light in the intersection can only be activated for a maximum of 15 seconds. All restrictions are shown in Table 1. Values larger than these restrictions could be used but a solution to

the timing problem could become incomputationally infeasible. This area, of larger restrictions, would be further researched in further work.

Traffic Light	Time Allowed
North	0-15 seconds
South	0-15 seconds
East	0 - 15 seconds
West	0 -15 seconds
	Total: 0 - 60 seconds

Table 1: Table showing the possible timing values for each traffic light in a intersection.

With the pegs in place and the timing data control structure in place it is possible to assign times to each traffic light depending on the amount of cars in a traffic lights queue.

A best case scenario for a planner would be if there were blocks of cars moving together as a units. For example if ten cars where moving through the traffic grid together they could all be moved through a intersection together and the traffic light could always be allocated 10 seconds for the group i.e. Figure 7.

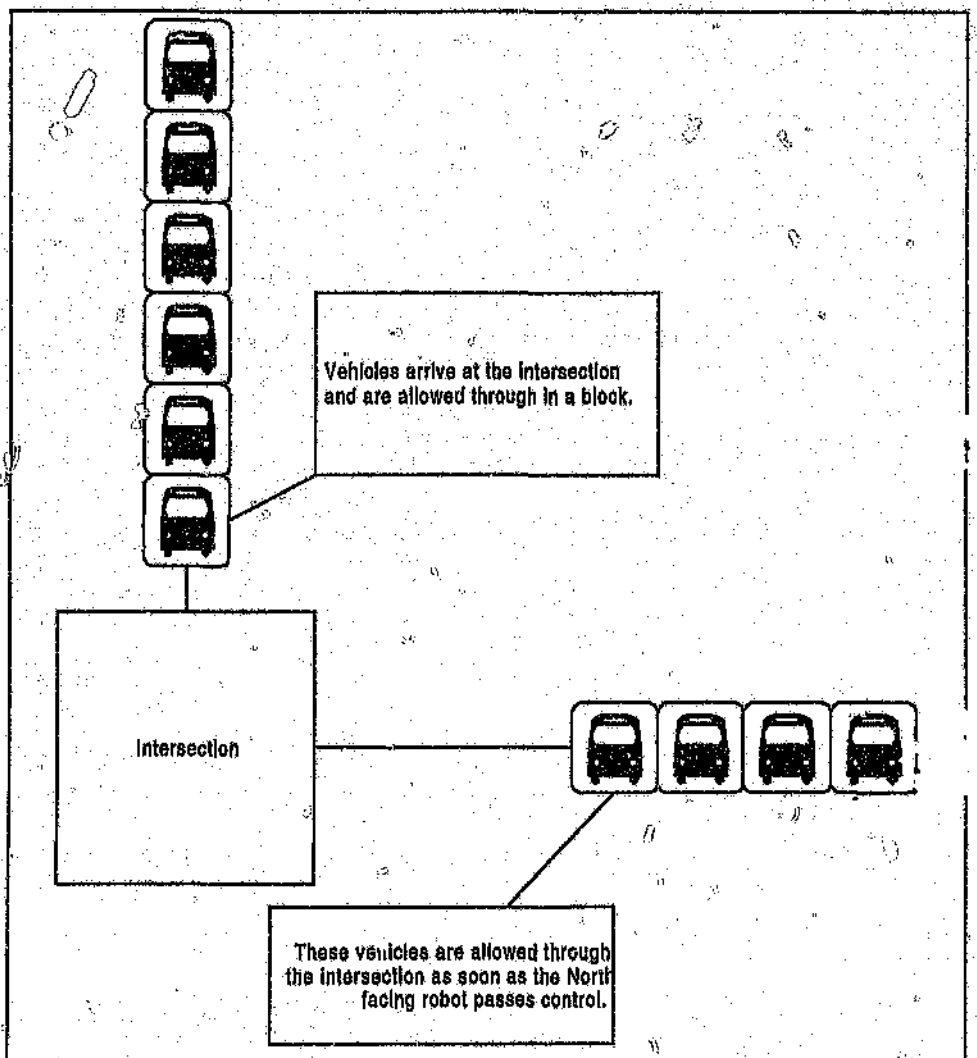


Figure 7: Diagram showing the best case scenario for a intersection. Cars do not only have to be arriving at the next traffic light but could be arriving at any of the traffic lights in the intersection.

Although the best case scenario is computationally easy to solve it does not always occur as a pattern in the traffic system. An average and worst case scenario ,which tends to occur more often, would arise when there are gaps between cars. An example of this is if there is 5 cars and then a 2 second gap and then another 5 cars arriving at a intersection. The absolute worst case would be single cars arriving at a intersection with time gaps between all the cars. An example of these cases is given in Figure 8.

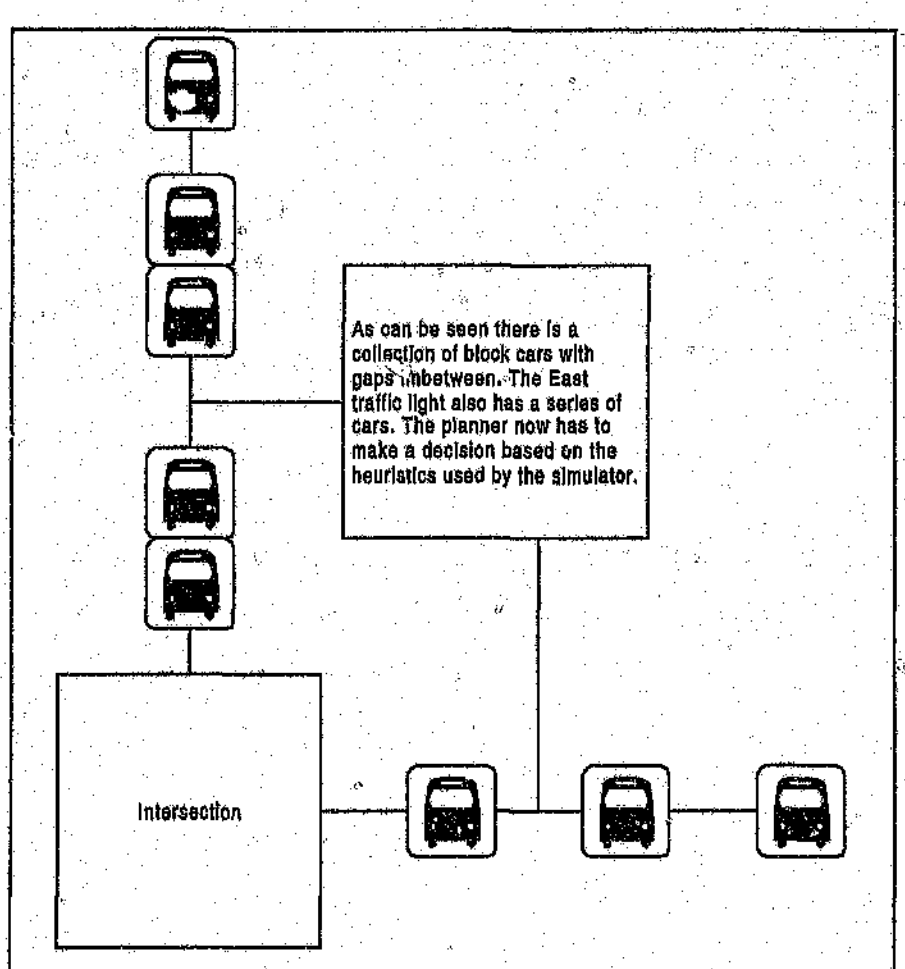


Figure 8: Diagram showing the average and worst case scenarios for a intersection. As can be seen cars arrive in small bursts with a number of gaps between them. Again these cars could be arriving at any of the traffic lights in the intersection.

Handling the best case scenario is relatively easy for the planner. Each block of cars up to 15 cars (15 seconds , 1 car a second) is allowed through at one time. Therefore the first heuristic used by the planner is to assign a time for a traffic light based on the maximum amount of cars in a row. An example of this case is given in Figure 7. As can be seen the planner will always allow a block of cars through first, even if a bigger second block is waiting to go at a different traffic light. This is not a primary heuristic but often is used in conjunction with the second heuristic.

The second heuristic is based upon the average and worst case scenarios. A decision must be made if gaps should be catered for in the current cycle or in the following cycle. An example of this is if there is a block of 5 cars and then a 2 second gap and then another block of 5 cars, then the question becomes, should the traffic light stay green for 12 seconds or only 5 seconds and transfer control to the next traffic light. If the planner decides to wait for the next cycle the cost of waiting the full cycle must be calculated into the overall decision.

As mentioned above the first heuristic used always allows the maximum block of cars (without breaks) in a 15 second range to pass through the intersection at one time i.e. set the traffic lights time to the maximum amount of cars in a block up to 15 seconds. Taking this into account the second heuristic is generated: If there is a block of cars waiting at the next traffic light(s) in the current pegged area, that is bigger than the block of cars after a gap at the current traffic light, control will be passed onto the next traffic light(s). For completeness a gap is defined as a space of 1 second or above between cars.

The time for the traffic lights is re-planned for every incoming car that is inserted into the blackboard structure. As a message comes in about a car, the car is inserted into both the traffic light structure (North, South , East , West i.e. Figure 4) and into the timing data control structure. Once this information has been entered re-planning / planning can occur.

Because of the design of Datilis's heuristics, high volumes of cars travelling in any one direction is well handled. In response to a continuous flow of traffic from any one direction and no other traffic on the network, Datilis would set the facing traffic light green permanently. This is achieved by making the time for all other traffic lights 0 seconds for each cycle and 15 seconds for the facing traffic light.

In conclusion, the timing data control structure regulates the timing constraints of the traffic lights. Boundary bashing and pegging is used to constrain the time of the traffic lights to the passage of the cars through the intersection through this structure. When a message is sent informing an agent of a new car, the planning modul. of an agent uses boundary bashing and pegging to constrain the traffic lights for certain periods relating to the arrival time of the car.

Example Problem

To clarify the previous section and to further clarify the results that are presented in the next section an example of a simulated run is given. This example problem uses the same grid as the one used for the final results.

The traffic grid used is a four by four matrix giving four centre nodes i.e. Figure 9. Centre nodes are areas of higher volume of cars as they are used for all through traffic i.e. generally cars will travel from one border of the grid to another travelling through the centre intersections. When generating the grid random times were used for all traffic lights. For more realistic results the robots active time is generally a small percentage of the time taken to travel from one robot to another.

In this example results were generated for a single car travelling through the traffic system. The car enters the system at intersection 1 and then travels through to intersection 13 and then out the system i.e. Figure 10 . The car travels through ten intersections. In the example the results are trivial as there is only one car travelling through the grid when the results were generated. The ideas behind this result is central to the research and therefore it was decided to leave only one car in this example.

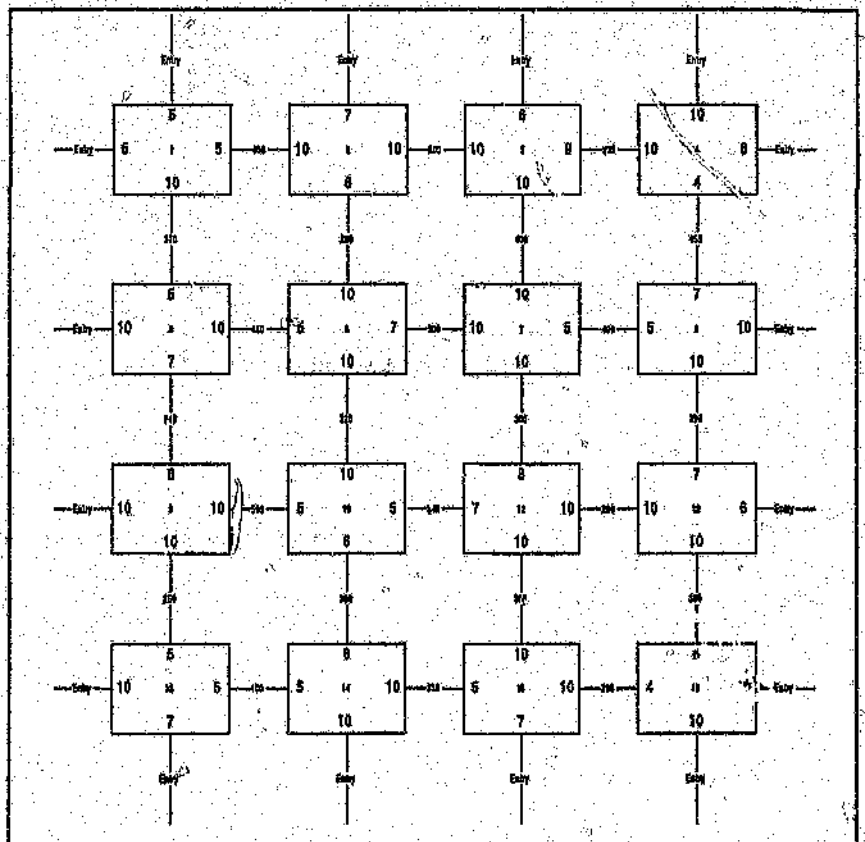


Figure 9: The traffic grid used for example and for the final simulation runs.

All results are assumed to start from 0 seconds i.e. when the car enters the system it is assumed that the timing starts and the simulation clock is set to time 0.When the results are generated for multiple cars the first car starts at time 0 and the rest of the cars time are incremented from there. In all results the north facing robot is initiated at time 0.

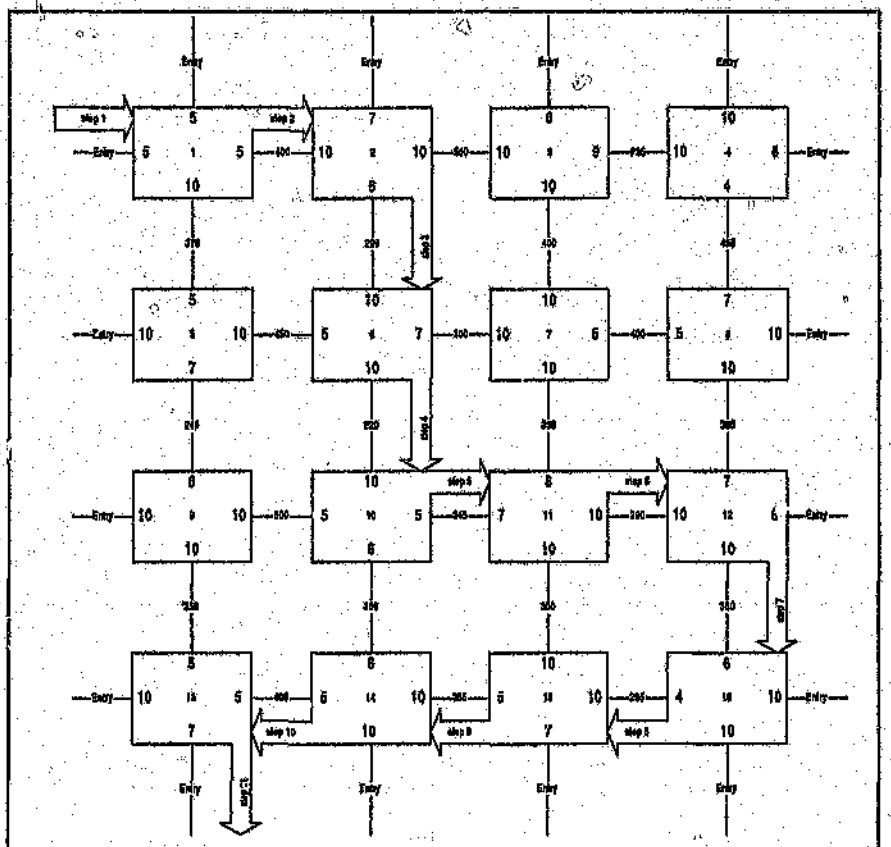


Figure 10: The route taken by the car in the example simulation. The car enters the traffic grid at Intersection 1 and leaves the traffic grid at Intersection 13.

The best time the car can take to complete this route is 2975 seconds. This is derived by taking the starting time plus the time needed for the car to enter the system. For example if a car was entering from the west side of a intersection at time 0 and the north robot was also activated at time 0, the first chance the car would have to enter the system is once the north, east and south robot have been activated and deactivated. This time is considered to be part of the base time as a car is entering the system from a unknown area and hence no pre-planning could be done. So in this example at time 0 the car is stopped at intersection 1 by the west facing intersection. Also at time 0 the north facing robot is activated for 5 seconds. Then at time 5 in the simulator the east facing robot of intersection 1 is activated for 5 seconds and then at time 10 the south facing robot is activated. Therefore only at time 20 on the simulator does the example car enter the grid (pass through intersection 1).

Because all the cars enter the grid from unknown areas that are not controlled by the distributed system all time taken to enter the system is added to both the base and the distributed AI results received and this time is unchangeable. From this time on, for the best time, all times are added assuming that the car arrives at all intersections and the robot is always green. Therefore the base time is taken by adding up all the times taken to travel the roads along the route plus the time taken to enter the grid.

Best Time	Car Move Time	Difference (Car move time - best time)	Expected Time
0	0	0	0
20	20	20	20
400	420	20	420
690	710	20	710
910	936	26	930
1255	1288	33	1281
1545	1578	33	1578
1925	1980	55	1958
2220	2282	62	2275
2575	2641	66	2637
2975	3053	78	3041

Table 2: The statistical data for the example. Best Time is the fastest time the car can move through the system. Car Move time is the actual times when the car moved through a intersection. Difference column is the difference in time between the best time and actual time. The expected time is the time the car arrived at the intersection. So the difference between the expected time and Car move time is the waiting time at the intersections.

The actual time taken by the car was 3053 seconds, where the difference between the base time and the actual time was 78 seconds. As can be seen from Table 2 the difference of the base and actual results (column 2) increases as the car travels further along its route. This is an expected result and means that the actual time taken by the car is divergent from the base time the longer the car is stopped in the traffic system.

This is due to the fact that at best the actual time will be the same as the base time. This case occurs when all facing robots are green when the car arrives at the intersections. The worst case would be when the car just misses its facing robot when arriving at the intersection and this would occur for all intersection that the car arrives at. In this case the full time of all three other robots would be added to the time. An average or general case would be that the car would sometimes catch a green robot and sometime miss it by the maximum amount. Each robot missed means the actual time taken gets further from the best time and hence the average pattern is divergent. The best, average and worst case is show in diagram format in Figure 11.

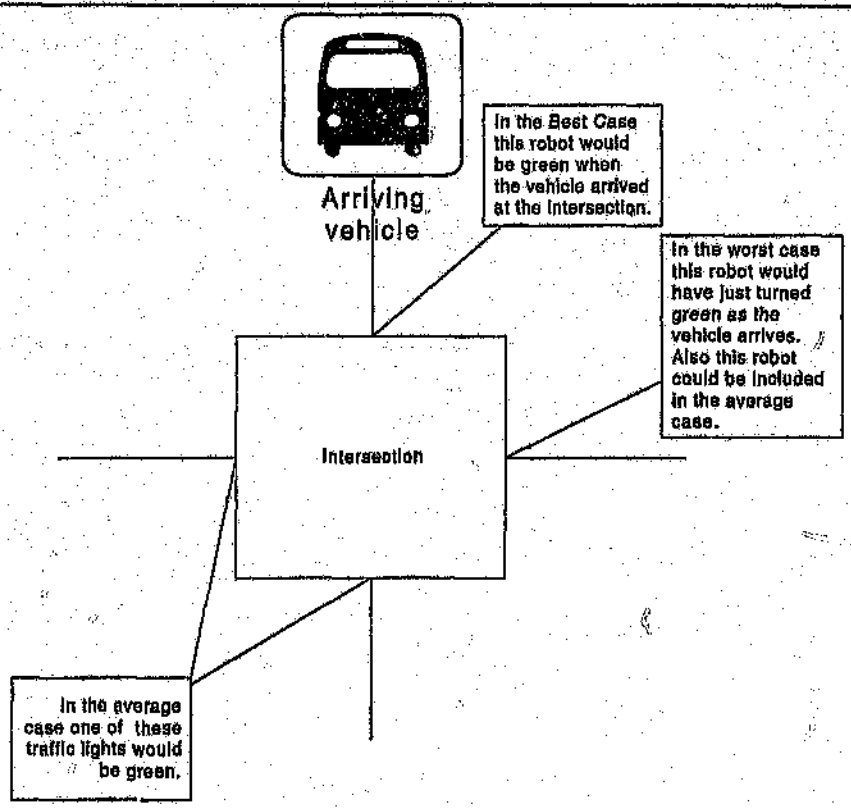


Figure 11: Graphic showing the position of control in an intersection for best, average and worst case scenarios.

The fourth column of Table 2 is the expected time of the car. This time is the actual time the car arrives at the intersection If the car arrives at time 420 and leaves at time 425 then the car has stopped for 5 seconds. This column is used as a indicator to show at which intersections a car has been stopped.

In this example, with the distributed artificial intelligent system enabled, the best time is equal to the actual time. Each intersection the car comes to is automatically changed to green for the car as there is only one car in the system. Hence the result for this example could be said to be optimal. With the difference in time between the base case and the distributed case being 78 seconds.

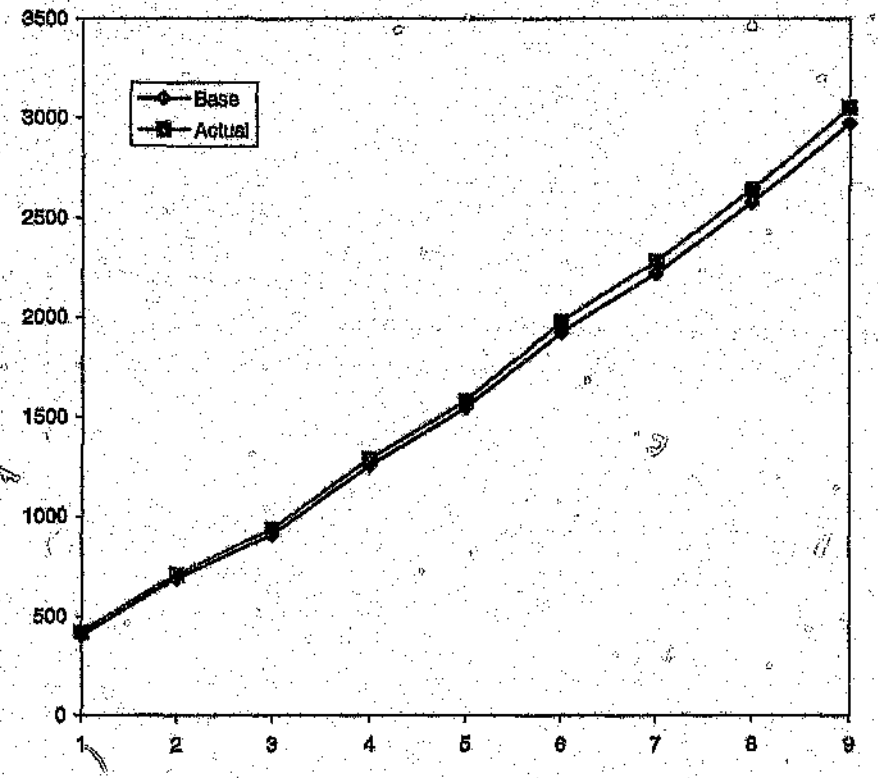


Figure 12: The results of the example showing the Best / Distributed case Vs the base case. As can be seen from the graph the two lines are clearly divergent.

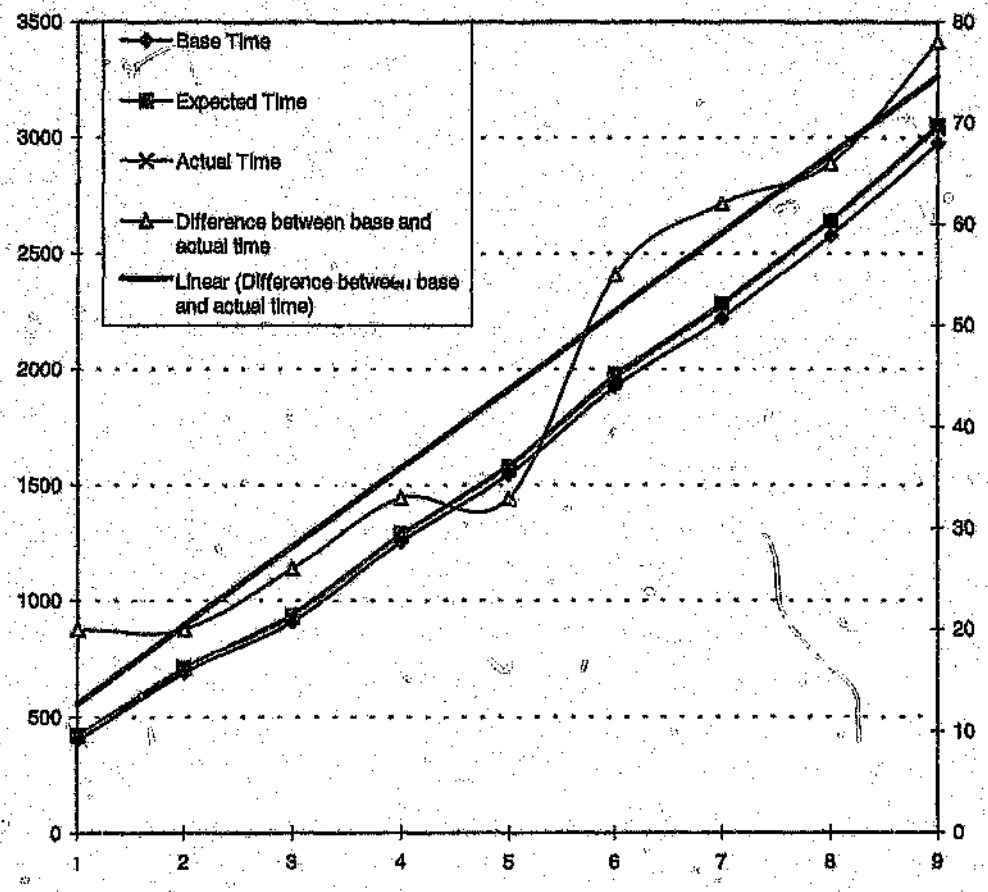


Figure 13: Graph showing all the results of the example with the difference between the best and base results also being shown.

From this example it can be seen that clearly there was 78 seconds saving on this run. A more detailed look at the testing methodology is presented in the next section.

Testing Methodology

This research was conducted in two phases. Both phases relate to certain levels of functionality in Datilis. The first phase is the base phase. All results received from this phase are used as the base results for all comparisons. This phase is equivalent to the original traffic light system without distributed control.

Datilis was used to simulate a number of "runs" of cars flowing through the system. All of the "runs" are dependent on the number of cars travelling through the system. In this research there were six major sets with five runs for each set. The divisions of the sets is based on the number of cars. In set one there was 300 cars, set two had 350 then the next set 500, 1000, 2000 and then 3000 cars. There were 5 runs made for each set to average out the times for the results.

All cars routes, and the intersections traffic lights times were generated. These figures were stored and used for both testing phases. The same starting information for both stages is essential for the validity of the final analytical result. Due to the exactness of both the starting point of the simulator run and the exactness of the main essential data the final result indicates the exact difference between a traffic light system without distributed control and a system with distributed control.

Phase two extends phase one with the addition of a car warning system. This is the first level of distributed control and the main area of study. Before a car arrives at a intersection the forwarding intersection's agent will send a communication warning the intersection of the incoming traffic and the amount of cars arriving. In this way the timing values of the intersections connections can be changed to best handle the incoming traffic. A planning system must be incorporated into each intersection abilities to handle the timing changes.

This phase builds anticipation into the control system. A simile would be that of a traffic officer at each intersection with a walkie-talkie connecting him to his nearest neighbour. In this case neighbours would communicate information about the cars on the way to the traffic officer's intersection.

Testing is done with the same data used in phase one and the same testing protocol. The result are compared and graphed in the next section.

Phase three includes both the two previous phases but allows bi-directional communication and involves the phase two planning module. In this stage a warning message is sent as before but the receiving agent can send back a response. This response can either be 'yes, send the car' or 'no, do not send the car'. The receiving intersection, depending on its current situation, will make decisions based on the results of the intersection's planning agent. If the response from a receiving agent is negative then the forwarding agent will have to decide if route changes should be made to the waiting cars. This phase allows the ability to guide cars through alternatives routes depending on traffic flow or breakages to the intersections or at the intersection points.

Unlike the previous two phases, phase three is not analytical in nature. There is no comparison to phase one because phase one could not cope with the uncertainty that is handled in phase three. For example if a intersection had to go down (no cars were permitted through the intersection) in phase one there would be no way to control the routes of the cars and hence they would still attempt to travel through the downed intersection. With the addition of further distributed control it would be possible to change the routing information of the cars and hence reroute cars around the downed intersection.

Extended distributed control is an area of research that follows on from the previous two phases. In this report this area is briefly touched upon and would form a logical extension of this work.

Results

Base Results

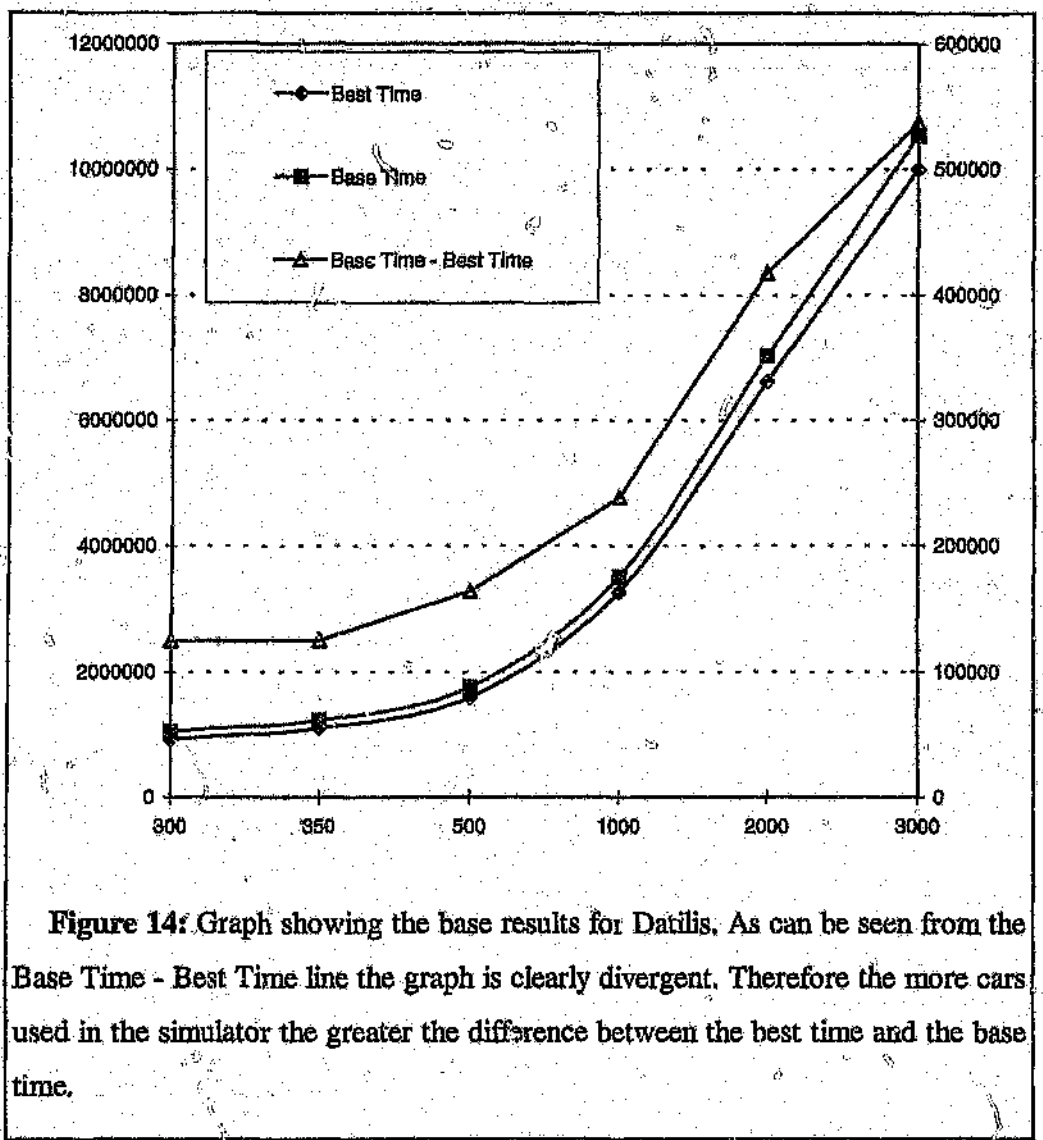


Figure 14: Graph showing the base results for Datilis. As can be seen from the Base Time - Best Time line the graph is clearly divergent. Therefore the more cars used in the simulator the greater the difference between the best time and the base time.

As can be seen from Figure 14 the results are divergent as expected. These results indicate the base result with no distributed control. From the graph it can be seen that as more cars are used in the simulator the greater the divergence from the best time. This is due to the fact that there is a greater number of cars that individually add their individual waiting times to the final time. The more cars there are the greater the

congestion at each intersection. The divergent nature of the results are indicated by the Base Time - Best Time line. These results are further discussed in the Comparison of Results section.

2nd Stage Results

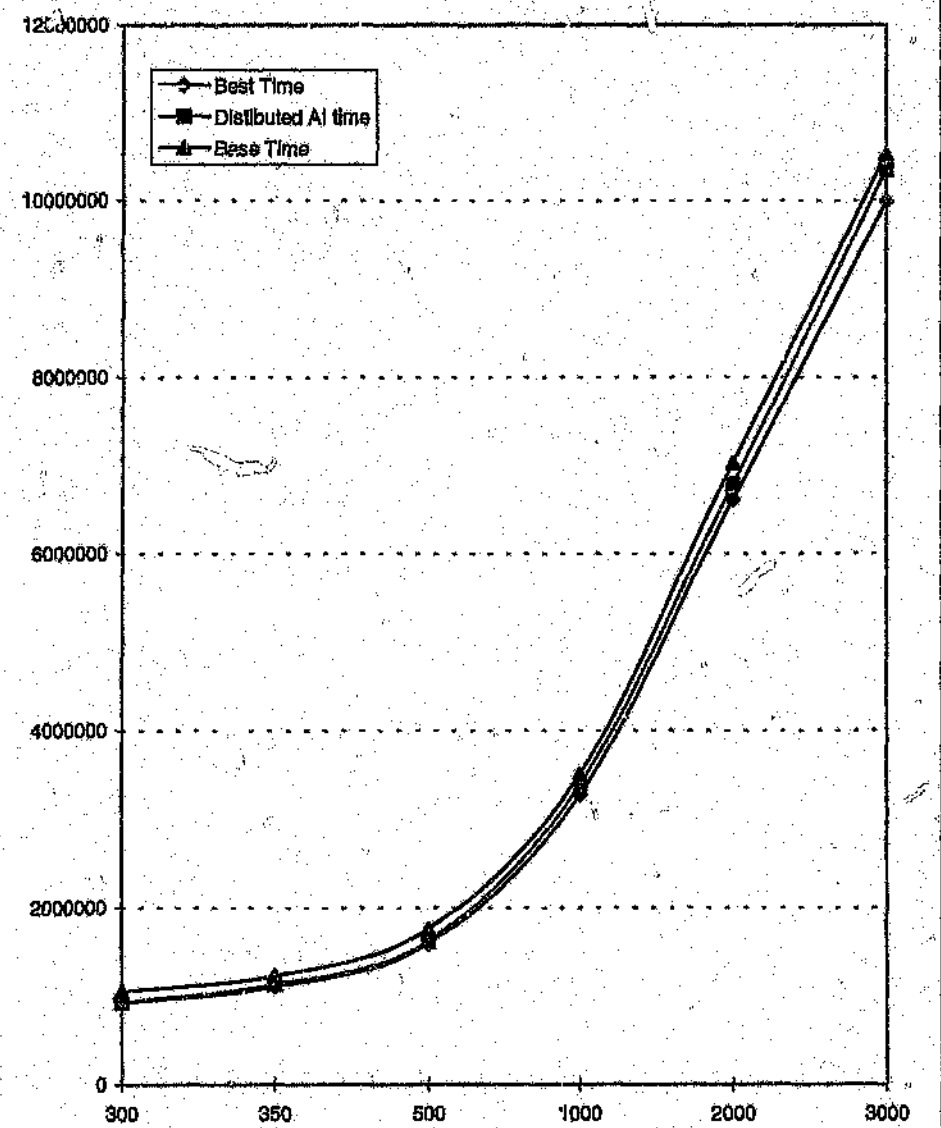


Figure 15: All results including the results of the simulator when using distributed control methods. As can be seen from this graph the Distributed AI time is clearly better than the Base Time.

The base results are compared in this graph to the results of using the distributed control structures. As can be seen from Figure 15 the distributed AI time is placed between the best time and the base time. This is an expected results as this mean with distributed control there is a saving of time in the traffic grid as compared to the base time. This result is discussed in more detail in the Comparison of Results section.

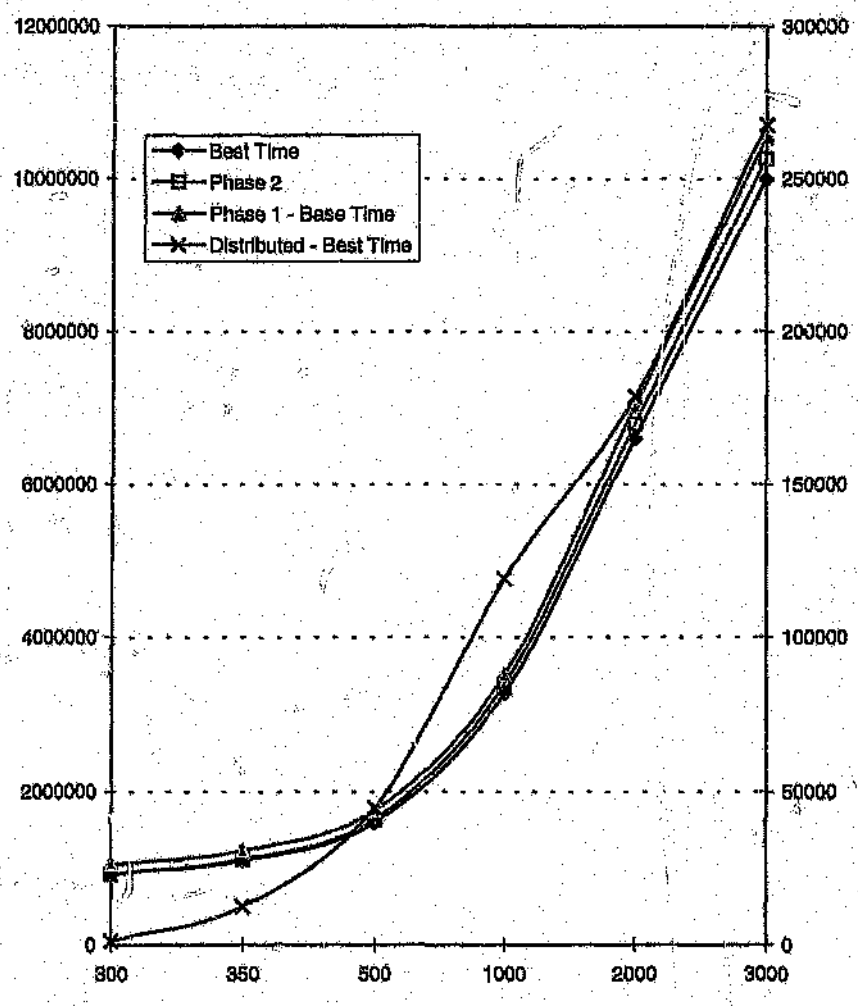


Figure 16: This graph is exactly the same as Figure 15 but has a additional line indicating the difference between the Distributed Time and the Best Time. As can be seen from the graph as more cars are added to the system the distributed time gets increase compared to the best time.

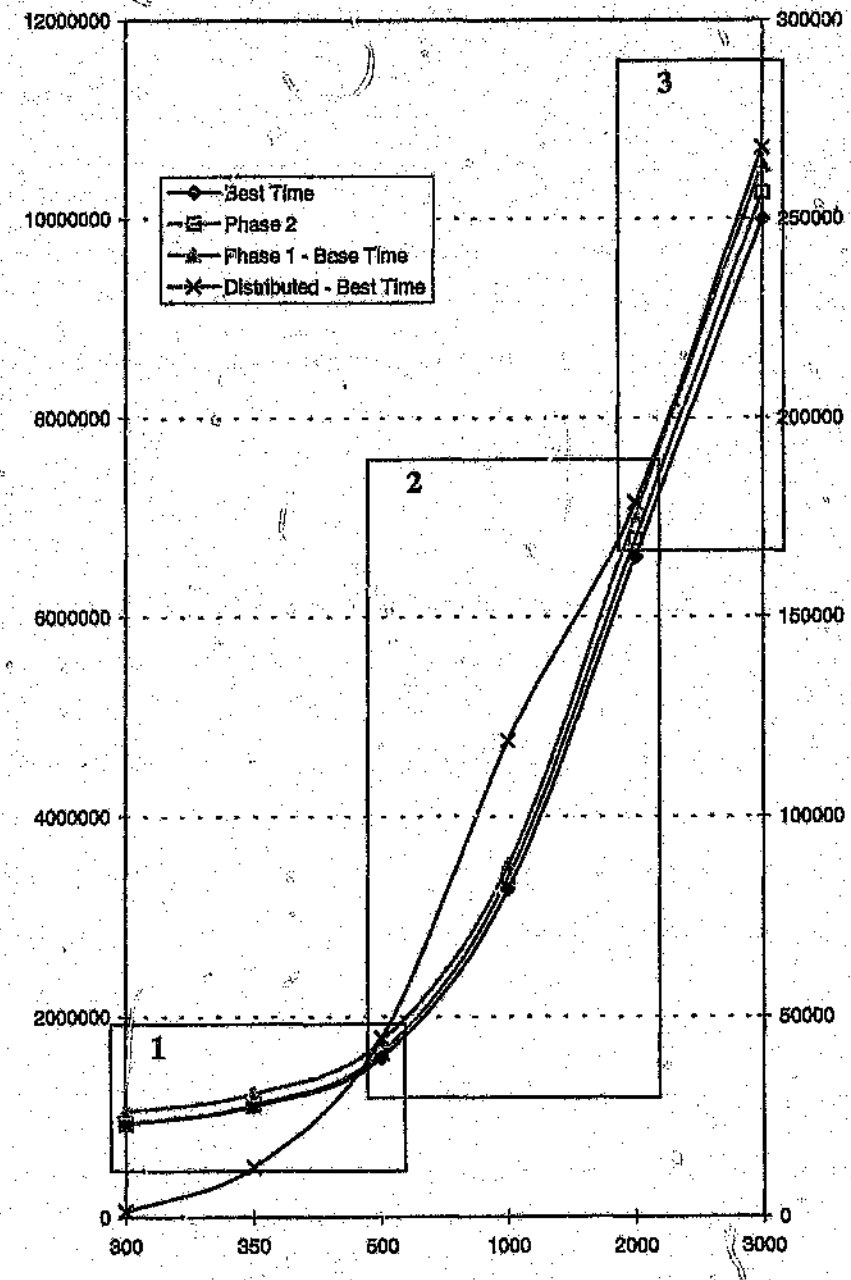


Figure 17: Same graph as Figure 16 but with added blocks showing the areas that are discussed.

These results can be roughly broken down into three parts. As can be seen from Figure 17 the corresponding sections have been marked by three blocks. Each of these blocks correspond to very rough sections of results. The blocks break down the graph into sections corresponding to the number of cars. Block 1 includes the results from 300 to about 700 cars. Block 2 from 700 to 2000 cars and block 3 from 2000 to 3000 cars.

As can be seen from the results the best time line is divergent from the base time line. This is more clearly indicated in Figure 14. From this it can be deduced that as more cars are run on the simulator the greater the time difference between the best time and the base time. This result follows as a logical conclusion to the results presented in Figure 13.

If a comparison is now done with phase 2 result time i.e. the distributed AI time line, it can clearly be seen that the results fall roughly into the three blocks shown in Figure 17. In block 1 the phase 2 line is on or very near the best time line. For block 2 the phase 2 line is moving away from the best time line but is still much closer to this line than the base line. In block 3 the phase 2 line has moved to roughly halfway between the best time line and base time line and is clearly getting closer to the base time line. A more detailed analysis of each block is presented in the next three paragraphs.

In block 1 there is roughly between 300 - 700 cars in the simulator. With the use of distributed control structures the average time taken for the cars to travel through the traffic grid is close to optimal. Near the 700 car mark the distributed time is moving away from the optimal line (best time line) but is still significantly better than the base results. Therefore block 1 shows the best results of the simulator.

In block 2 there is roughly between 700 - 2000 cars in the simulator. With the use of the distributed control structures the average time taken for the cars to travel

through the traffic grid is roughly 25% worse than the best time line and 75 % better than the base time line. By the end of this block the distributed line has moved to roughly the middle of the best time line and the base time line. This means that for this block the distributed control structures are still performing well and giving a 75% - 50% increase in performance in the traffic system.

In block 3 there is roughly between 2000-3000 cars in the simulator. With the use of the distributed control structures the average time taken for the cars to travel through the traffic grid is roughly halfway between the best time line and the base time line i.e. about 50%. By the end of the block the distributed time line is closer to the base time line then the best time line. This means as more cars are simulated the worse the distributed structures handle the flow of cars and the closer the distributed time will get to the base time.

In conclusion, as more cars enter the simulator the effectiveness of the agents in each intersection is curtailed. If a intersection is inundated with cars there is no ability to manoeuvre cars. Hence the simulated time taken for the simulator when using distributed control moves closer to the time taken without distributed control. In simple terms it could be said that even with distributed control a large influx of cars will lead to a traffic jam which takes a large amount of time to handle. In the case of a smaller amount of cars the distributed control structures were highly effective and led to effective time saving in a traffic light system.

General Discussion and Conclusion

An agent must construct representations, albeit economical ones, that accurately cover its perceptions of the environment. When an agent represents other agents, it must represent what they believe, desire and intend(Bond, 1988). The other agents have beliefs about this agent, and its beliefs and representations of them, ad infinitum. Believing that there are other agents in its environment will cause an agent to act differently, e.g., benevolently and predictably. First, a benevolent agent might clean up its environment, if it believes that other agents will act similarly, because it knows that it can operate more efficiently in a more orderly environment.

For an agent to act predictably the agent should have a self model with which its beliefs are consistent, i.e. agents beliefs should be consistent with each other and agents actions should be consistent with its beliefs (Bond, 1988). Also the agent should have a representation of what other agents believe about it, and should act in accord with it. By acting predictably, an agent will reduce conflicts with other agents, thereby increasing not only its own effectiveness, but also the effectiveness of all of the agents.

Distributed systems can be thought of as composed of intelligent entities with intentions, beliefs, and know-how. These entities interact through high-level communications to effect each other intentions and beliefs and thereby actions.

In this report a simulator, Datilis, was designed and built using the principles mentioned above. Datilis was built along the principles of extensibility and object functionality. Each agent was designed to act predictably and consistently. The blackboard system of control was used in Datilis because of the sophisticated local processing capability coupled together with selective communication decisions which reduces control uncertainty. The independent data-directed nature of the blackboard system generates a significant amount of parallel activity that is well utilised by Datilis.

Through the use of communication each agent in Datilis effects other agents intentions and beliefs and thereby there actions.

Datilis was also built upon the principles suggested by Ravindran (1987):

1.Do not build a complicated model when a simple one will suffice

2.Beware of moulding the problem to fit the technique.

3.The deduction phase of the modelling must be conducted rigorously.

4.Models should be validated prior to implementation.

5.A model should never be taken too literally.

6.A model should neither be pressed to do, nor criticised for failing to do, that for which it was never intended.

7.Beware of overselling a model.

8.A model cannot be any better than the information that goes into it.

Each of these principles were used and verified in the construction and the evaluation of Datilis.

Networking and Dai

Many computer networks can be enhanced through many of the same ideas used in this project. Many distributed structures i.e. contract network, blackboard system, could be incorporated into a computer network.

The question of what is to be used as a communications channel comes to the forefront when using a computer network. In most distributed systems a computer network is used as the communication medium. In a computer network a decision must be made whether a separate communication channel should be built or if the existing network will be used for distributed communication as well. A communication channel in most cases needs to work considerably faster than the system it is working on. In a

computer network communication takes place at the same speed as the data being transferred. The advantages of the distributed control structures may outweigh the disadvantages of building a new communication channel.

Certain areas of networking already use some of the techniques used in DAI. These areas include network load balancing and the DNS system which is used extensively on the internet. Also some techniques used in networking can be / are integrated into distributed AI.

Datilis and uncertainty

The traffic light system is a open system i.e. it is a system that is large and unpredictable and is therefore difficult to be completely characterised. This system cannot be meaningfully closed and bounded for analysis purposes. No matter how well defined the traffic light system, it will always be subject to new information from outside controlled area i.e. the frame problem.

Further work and enhancements

Certain enhancements could be made to the traffic control system to make it more viable. Certain of these techniques were implemented in the Datilis system to check for feasibility but limited testing was undertaken. The first addition could be that of a controlling agent in each car that could be used as a backup structure for any intersection or it could undertake planning on its own. In this way a "intelligent" car would know its own route. Intersections could communicate directly with the car passing on relevant information. Uncertainty is more easily handled as routes can be passed directly to the car rather that using the robots and turning arrows to direct the cars.

Assuming a car has a small communication radius or the car can only communicate with the intersection when crossing or waiting at a intersection, the agents in the cars

can be used as a limited communication channel between the intersections in case of communication failure in the primary communication channel.

The search and limiting heuristics used with the planner to make decisions could be modified in a number of ways to obtain different variations in the total planning system. These heuristics were very simplistic in there design and implementation.

A learning model could be incorporated into the system assuming there was some sought of repetition that could be used as a basis of analysis .i.e. if the traffic move win regular patterns. To keep down the overhead of the planning system a knowledge base can be used rather than replanning. Through the use of a knowledge base pattern analysis can occur and set times for the traffic lights can be generated depending on the time of the day. With the ability to use a knowledge base Datilis would be able to use a less bandwidth for communication and less CPU time when planning.

Summary

In summary this research shows that distributed artificial intelligent techniques can enhance a standard distributed system. Through the use of the sample testbed system, Datilis, a distributed traffic control system, standard techniques such as blackboard systems and multi-agent planning were implemented and tested. The sample system was built up in two phases. The second phase, the multi-agent planner, is the main focus of the research.

Measuring the results of the proposed model was a quantitative process. The first phase was used to generate all "base" results. These results were used for comparison with the second phase. Phase two extended the base phase within the framework of uncertainty and sophisticated local control. Through the use of distributed blackboards information is compiled and stored for the use of the multi-agent planner. Distributed AI is used to enhance the control handling of the distributed system.

Datilis was used to evaluate the research hypothesis:

"if the throughput of individual intersections is maximised and the waiting time of individual cars is minimised then the throughput of the whole system will be maximised."

In conclusion, for low volume densities the distributed control structures were highly effective and even with very high volumes improved performance can be seen.

References

Allen, J F. (1984) **Towards A General Theory of Action and Time,** Artificial Intelligence, vol. 23, pp 123-154.

Allen, J F. and Koomen, J. A. (1983) **Planning Using a Temporal World Model,** Proceedings of the Eighth International Joint Conference of Artificial Intelligence, pp. 741-747.

Alterman, R. (1986) **An Adaptive Planner,** AAAI 1986 Proceedings, pp. 65-69.

Georgeff M. And Lansky A L. (1987) **Reactive reasoning and Planning,** AAAI 87 Proceedings.

Feldman, J A and Sproull R F. (1977) **Decision Theory and Artificial Intelligence II: The Hungry Monkey,** Cognitive Science 1, pg 158-192.

Fikes, R E And Nilsson, N H. (1971) **STRIPS: a New Approach to the Application of Theorem Proving to Problem Solving,** Artificial Intelligence 2, pp 189-208.

Fikes, R E, Hart P E, and Nilsson N J. (1972) **Learning and Executing Generalized Robot Plans,** Artificial Intelligence 3(4), pp 251-288.

Hayes, P J. (1975) **A Representation for Robot Plans,** Advance papers of IJCAI-75, Tbilisi, USSR.

Hammond, K J. (1986) **CHEF: A Model of Case-based Planning,** AAAI 1986 Proceedings, pp261-271.

Hendler, J A. (1985) **Integrating Marker-Passing and Problem Solving,** Proceedings of the Cognitive Society Conference.

Korf, R E. (1987) **Planning as Search: A Quantitative Approach,** Artificial Intelligence, Vol 33, North-Holland Publishing Co.

Lansky, A L. (1989) **Localized Representation and Planning,** The Proceedings of the 1989 Stanford Spring symposium Workshop on Planning and Search.

Lifschitz, V. (1990) **On The Semantics of Strips, Readings in Planning,** Morgan, Kaufmann Publishers, Inc. San Mateo, California.

McDermott, D. (1978) **Planning and Acting,** Cognitive Science 2(2), pp 71-109.

McDermott, D. (1982) **A Temporal Logic for Reasoning About Processes and Plans,** Cognitive Science 6, pp. 101-155.

Newell, A And Simon, H A. (1963) **GPS, A Program that Simulates Human Thought,** Computers and Thought, McGraw-Hill, New York.

Pednault E P D. (1990) **Formulating multiagent, dynamic-world problems in the classical, planning framework,** Readings in Planning, Morgan Kaufmann Publishers, Inc. San Matei, California.

Rosenschein, S J. (1981) **PLAN SYNTHESIS: A logical Perspective,** IJCAI 1981 Proceedings, pp. 331-337.

Sacerdoti, E D. (1973) **Planning in a Hierarchy of Abstraction Spaces,** Advance papers of IJCAI-73, Palo Alto, CA ., USA.

Sacerdoti, E D. (1975) **The Nonlinear Nature of Plans,** IJCAI 1975, pp 206-214.

Shohan, Y. and McDermott, D. (1988) **Problems in Formal Temporal Reasoning,** Artificial Intelligence 36, pg. 49-61.

Waldinger, R. (1975) **Achieving Several Goals Simultaneously,** SRI AI Center Technical Note 107, SRI, Menlo Park, Ca.,USA.

Stefik, M. (1981) **Planning with Constraints (MOLGEN: Part 1),** Artificial Intelligence 16, pg 111-140.

Bisiani, R. (1986) **A software and Hardware Environment for Developing AI Applications on Parallel Processors.** *Proceedings of 1986 conference of the American Association for Artificial Intelligence*, pages 742-747.

Bond, A H. And Gaser, L. (1988) **Readings in Distributed Artificial Intelligence.** *Morgan Kaufmann Publishers, Inc.* pp. 3-35.

Cammarata, And McArthur, D. And Steeb, R. (1983) **Strategies of co-operation in distributed problem solving.** *Proceedings of the 1983 International Joint Conference on Artificial Intelligence*, pp.767-770.

Conry, S E. And Meyer, R A. And Lesser, V R. (1986) **Multistage Negotiation in Distributed Planning.** *Technical Report 86-67, Department of Computer and Information Science, University of Massachusetts, Amherst, MA*, December.

Corkill, D D. And Gallagher, K Q. And Johnson, P M. (1987) **Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures.** *Proceedings of 1987 Conference of American Association for Artificial Intelligence*, pages 18-23.

Cromarty, A S. (1987) **Control of Processes by Communication over Ports as a Paradigm for Distributed Knowledge-Based System Design.** *Benjamin/Cummings Publishing Company, Inc.*

Davis, R. And Smith, R G. (1983) **Negotiation as a Metaphor or Distributed Problem Solving.** *Artificial Intelligence*, 20(1):63-109.

Durfee, E H. And Lesser, V R. (1987a) **Using Partial Global Plans to Coordinate Distributed Problem Solvers.** *Proceedings of the 1987 International Joint Conference on Artificial Intelligence*, pp.875-883.

Durfee, E H. And Lesser, V R. And Corkill D D. (1987b) **Coherent Cooperation Among Communicating Problem Solvers.** *IEEE Transactions on Computers*, C-36:1275-1291.

Ensor, J R. And Gabbe, J D. (1986) **Transactional Blackboards.** *International Journal for Artificial Intelligence in Engineering*, 1(2):99-109.

Fennel, R D. And Lesse, V R. (1977) **Parallelism in Artificial Intelligence Problem Solving: A case Study of Hearsay II.** *IEEE Transactions on Computers*, C-26(2):98-111.

Fox. M S. (1981) **An Organisational View of Distributed Systems.** *IEEE Transactions on Systems, Man and Cybernetics*, 11:70-80.

Gasser, L. And Braganza, C. And Herman, N. (1987) **Implementing Distributed AI Systems Using MACE.** *Proceedings of the Third IEEE Conference on Artificial Applications*, pages 315-320.

Genesereth, M R. And Ginsberg, M L. And Rosenschein. (1986) **Cooperation without Communication.** *Proceeding AAAI-86*, pp.51-57.

Georgeff, M P. (1983) **Communication and Interaction in Multi-Agent planning.** *Proceedings AAAI-83* pp.125-129.

Hudlicka, E. And Lesser, V. (1987) **Modelling and Diagnosing Problem Solving System Behaviour.** *IEEE Transactions on Systems, Man and Cyberbetics*, 17(3):407-419.

Kornfeld, W A. And Hewit C E. (1987) **The Scientific Community Metaphor.** *IEEE Transactions on Systems, Man and Cybernetics*,17(3):407-419.

Leao, L V. And Talukdar, S N. (1986) **COPS: A System For Constructing Multiple Blackboards.** International Journal for Artificial Intelligence in Engineering, 1(2):70-79.

Lesser, V R. And Corkill, D D. (1981) **Functionally Accurate, Cooperative Distributed Systems.** *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):81-96.

Levin, J A. And Moore, J A. (1977) **Dialogue-Games: Metacommunication Structures for Natural Language Interaction.** *Cognitive Science*,1:395-420.

Morgenstern, L. (1987) **Knowledge Preconditions for Actions and Plans.** *Proceedings of the 1987 International Joint Conference on Artificial Intelligence*, pp.867-874.

Rosenschein, J S. (1982) **Synchronization of Multi-Agent Plans.** *Proceedings AAAI-82*, pp. 115-119.

Rosenschein, J S. And Genesereth M R. (1985) **Deals Among rational Agents.** *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*, p.91-99.

Smith, R G. And Davis, R. (1981)**Frameworks for Cooperation in Distributed Problem Solving.** *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):61-70.

Stuart, C J. (1985) **An Implementation of a Multi-Agent Plan Synchronizer.** *Proceedings of the 1985 International Joint Conference on Artificial Intelligence*, pp.1031-1033.

Yonezawa, A. And Briot, JP. And Shibayama, E. (1986) **Object-Oriented Concurrent Programming in ABCL/1.** *Proceedings of the 1986 Conference on Object-Oriented Programming Systems and Languages*, pages 258-268.

Ravindran, A. And Phillips, D. And Solberg, J. (1987) **Operations Research Principles and Practice.** *Second Edition, John Wiley & Sons.*

Author: Bergheim Gavin.
Name of thesis: DaTiLis- distributed traffic light simulator.

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2015

LEGALNOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page)for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.