SynthECA: A Synthetic Ecology of Chemical Agents

by

Anthony R. P. White
M.A. (Theoretical Physics),
Cambridge University, Cambridge, England
M.C.S. (Computer Science),
Carleton University, Ottawa, Canada

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Ottawa-Carleton Institute of Electrical and Computer Engineering
Faculty of Engineering

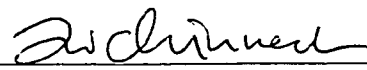
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario K1S 5B6, Canada

August 29, 2000

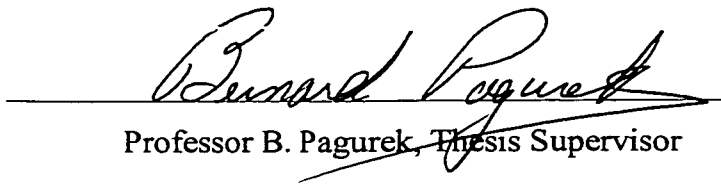
The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

SynthECA: A Synthetic Ecology of Chemical Agents

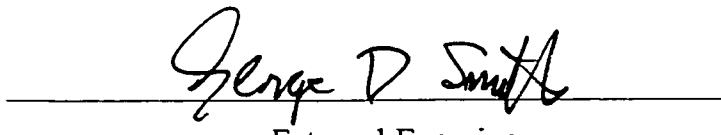
submitted by Tony White, M.A., M.C.S.
in partial fulfillment of the requirements for
the degree of Doctor of Philosophy



Chair, Department of Systems and Computer Engineering



Professor B. Pagurek, Thesis Supervisor



External Examiner

Carleton University
August, 2000

Abstract

Emergence, or self organization, is being increasingly associated with distributed or decentralized problem solving. Naturally occurring systems demonstrate that complex problem solving with simple agents is possible and that such systems are extremely robust to changes in their environment and to the loss of individual agents. Furthermore, such systems are self-managing, requiring no "blind watchmaker" to ensure that the system operates efficiently.

Systems that depend upon the interaction of a large number of simple agents for their problem solving properties stand in stark contrast to monolithic agent systems that have been the target of considerable research. The frailty of symbolic systems, in the face of unforeseen situations and the failure of individual agents provides motivation for the research documented in this thesis. Social insect behaviour, and swarm systems generally, provide the inspiration for the research described here as does the increasing importance of activation-oriented computing.

This thesis proposes an agent architecture that relies on local communication only, all algorithms having no knowledge of the global scene. The agent architecture is analysed in terms of the requirements necessary to support emergent problem solving. The solutions to problems in the Communications domain, specifically routing and fault localization, demonstrate the utility of the proposed architecture along with algorithms that demonstrate the self-management of agent swarms.

Acknowledgements

A thesis is like a race, it has a beginning, middle and end with coaching a key to success. I am fortunate to have been supported by a great set of coaches during my personal "race".

I would like to thank my wife, Nikki, and boys, Colin and Andrew, for their tremendous patience and support during the writing of this thesis. At times I am sure that they wondered when, or if, I would finish. Without their encouragement I certainly would not have done so. Their sacrifice, their understanding, made this document possible.

In all endeavours of this nature, the importance of a supervisor cannot be overstated. I am extremely fortunate to have worked with Professor Bernard Pagurek, from whom I have learnt more than is possible to document here. His willingness to let me find my own way, while ensuring that wayward thinking was quickly corrected, ensures that I will always be in his debt. My ability to undertake research owes much to his guidance and tutelage. Finally, the trust that he placed in me to guide the research of other students has given me a priceless confidence that I could not have expected to acquire as a graduate student. I would also like to thank Andrzej Bieszczad, a good friend whose constant cajolling and encouragement meant that I could never give up even when there appeared to be no end in sight. His energy and constant positive review of my work ensured a timely conclusion to this thesis. Finally, I would like to thank the members of the Perpetuum Mobile Procura research group for their patience and for allowing me to participate in their research. Examining your problems allowed me special insight into my own.

For Mike (1956-1999), a very good friend

Table Of Contents

List of Figures

List of Tables

CHAPTER 1 *Motivations and Contribution*

1.1 Overview

This chapter provides motivations for the contributions made by this thesis. A statement of the problem area considered in this thesis would be: distributed problem solving using mobile agents. The goal of this chapter is to have the reader understand the path the thesis will follow, the importance of the problems solved, and why the ideas form a cohesive structure from which the thesis contributions naturally emerge. Emergence will be a word seen many times in this document and is carefully chosen. As a concept, it is central to the ideas presented here.

Emergence is being increasingly associated with distributed or decentralized problem solving. The pervasive nature of decentralized systems, and their most compelling characteristics, are outlined in the next section. Agents, natural instruments for distributed problem solving, have recently been proposed as a natural evolution of the concept of an object, although no single universally accepted definition of an agent yet exists. A section

on agents, or more specifically agent architecture, motivates the need for less brittle, more natural implementations. The word nature is carefully chosen here as this thesis appeals to naturally-occurring multi-agent systems for inspiration and as providing proof by existence of the value of such systems in the natural order. The section on agents highlights the problems of many of the existing architectures in terms of their knowledge representation and forms of computation.

Representation implies bias. It is clear that digital circuits and electronic signals are not the only means for implementing computers and performing computation. While our thinking has been dominated by Turing and Von Neumann views of computers and computation, other ideas and computational processes have recently challenged views previously thought unimpeachable. A section on forms of computation enhances the motivations for alternative agent architectures outlined in the agent section.

Motivations form the basis for contributions; i.e., statements of problems provide opportunities for solutions and these are provided in section 1.3 of this chapter. The chapter concludes with a review of the organization of the thesis.

1.2 Motivations

The motivations for this thesis broadly fall into three categories: decentralization, agent architectures and forms of computation. These categories are described in the following three sub-sections.

1.2.1 Decentralization

We live in an age of increasing decentralization. Why is this so? What makes

decentralized systems so attractive? Decentralized systems are pervasive, we find them at all levels in society, in naturally-occurring and economic systems and, increasingly, in theories concerning knowledge and self [Minsky 88]. It would seem that we have entered the " ... era of decentralization" [Resnick 94].

The rise of decentralization in society can be seen in the growth of democracy world wide, where decision making in terms of leadership is distributed to a large number of society's members. Decentralization in society also spans organizations; companies have increasingly moved decision making to lower levels in their structures, favouring instead contract-style interactions between sub groups in order to achieve a particular organizational goal. As examples of organizational decentralization we see the rise of quality circles within manufacturing industries, worker control of production lines, elimination of layers of management in many industries and moves towards matrix rather than hierarchical structuring of labour. In the economic area, we see the failures of centrally-planned economies and the move to the free market economy, with price, and the supply and demand of goods providing the driving forces for a decentralized control system.

Decentralization in naturally occurring systems can be found in many systems, spanning all levels of evolutionary complexity, from bacteria to ants and beyond. In such systems, the concept of societies exists; identifiable groups that collaborate in order to achieve some goal. The function and importance of societies will be re-visited later in this chapter. In fact, considering a society as a species, we may think of evolution itself as a

decentralized process; we do not need to invoke some "Blind Watchmaker" in order to have an evolutionary process or system. In fact, Darwin's theory of evolution requires no centralized planner, no overseer to ensure that evolution will occur. It merely relies on environmental change and the fact that effective characteristics in one environment may be a poor choice in another. In other words, it relies upon competition, an inverted form of collaboration. Stated another way, flippers may be an advantage when a watery environment exists but certainly do not provide the best form of locomotion when solid ground dominates the landscape. Dynamic fitness landscapes have a dramatic effect on the complexity and stability of systems [Seth 98]. We shall return to ecosystems and their characteristics later in this chapter.

Decentralization can be seen in theories of Knowledge. The growth in interest in Semiotics, where the observer must be considered part of the theory, implies distributed knowledge as, by necessity, all observers cannot occupy the same place at the same time[1]. Simply put, knowledge is distributed in multiple observers of the system. More recently, the prevailing views in Physics have associated increasing importance with the concept of information and the realization that observations include the observer. Fisher information subsumes Shannon information and Boltzman entropy in order to explain many fundamental physical principles; e.g., Einsteins's Field equations and Quantum Gravity.

Decentralization, then, is truly pervasive. This thesis takes decentralization as a motivating principle; i.e., the removal of all global state from the problem solving process.

1. Even if they did, they still would not have the same sensors.

1.2.2 Agents and Architecture

If we trace the origins of the concept of an agent in the body of work we could call Software History, we find that, originally, the basic software unit was the program. Program control was achieved by jumping from one location in the program to another. All state was global and a single thread of control existed. Structured programming was then developed, where segments of code were encapsulated, local state provided and the subroutine invoked through an external call. The concept of a structure was created. This phase in the history of software was a short one, as it was quickly realized that structures and behaviour (subroutines) were not integrated. This discontinuity in representation led to the programming paradigm that we now know as Object Oriented programming (OOP). Although OOP added other concepts such as inheritance and polymorphism, objects were still passive; messages being sent to an object as a consequence of the activities and actions of an external entity. Objects, in general, do not have their own thread of control.

The passive nature of an object and the overseer style of control implied centralized problem solving. A natural evolution of the concept of an object then occurred wherein an individual thread of control became associated with it and, along with a control element, internal goals. The agent concept was born. This fairly minimal definition of an agent as "an active object with initiative" [Parunak 98] i.e., having localized code, state and control, making application design the process of agent design plus agent coordination. Essentially, agent oriented systems differentiate between two forms of communication: internal and external. Internal communication is generally directed from one object to another. External communication concerns itself with agent coordination.

The appeal of agent architectures exists because of the promise of populations of agents organizing themselves without an overseer controlling element. Even more importantly, such populations should be able to react to dynamically changing environments and cope with unforeseen situations. At first glance, agents are extremely appealing because they reduce the number of behaviours in the active object and, therefore, the potential for unexpected interactions.

Achieving robust agent system behaviour has proven elusive. Why is this? History and origins are, as one might expect, to blame. Firstly, agent system research originated with the Artificial Intelligence (AI) community, a community grounded in the symbolic view of the world. The AI community has been strongly influenced by the Symbol Hypothesis [Newell 80] and first order predicate logic. The Resolution Principle [Robinson 65] entrenched the importance of symbols and theorem proving. Unfortunately, this approach has several limitations that will described in the next section. However, symbolic systems coordinate their activities by exchange of symbolic information and theorem proving. Ordering of symbolic information, and the (almost universally) non-monotonic nature of temporal reasoning makes it very hard to achieve reliable coordination. What happens if an agent dies, or communication proves unreliable? Also, either point to point communication is employed or some communications "centre" where messages are "appropriately routed" to interested parties is implied. This latter option smacks of centralization, thereby re-introducing many of the problems of monolithic behaviour present in pre-OOP systems. Symbolic communication also relies upon the existence of rich ontologies in order that meaning can be made universally accessible. So, is there an

alternative?

Naturally occurring systems of simple agents (such as populations of insects or other animals) offer considerable evidence that centralized, pure symbol processing agent systems are unnecessary. The key word is simple. Agents need not be rational by necessity for complex problems to be solved. A crucial observation is that agents are situated, *they cannot be considered independently of their environment.* In fact, the environment is the communications medium[1] and that agents sense only locally-available information in that medium. The lack of a global overseer is evident in naturally occurring systems. In observing that local sensing of information can be sufficient, it is important to understand what other characteristics allow much simpler agents to succeed where more complex systems have difficulty. Two characteristics of naturally occurring multi agent systems that facilitate complex problem solving are: population-based problem solving with the actions of single agents reinforcing each other and agent mobility driven by some signal gradient within the environment. These, and other important characteristics of naturally occurring multi-agent systems, will be discussed at length in the next chapter. The Ant System, a class of search algorithm based upon the foraging behaviour of ants is described there.

Our motivation, then, is the relative lack of success of the robust coordination of symbolic, rational agents in complex, distributed problem solving and the success of societies of simple naturally occurring agents in complex problem solving. We are

1. As Marshal McLuan put it, "The medium is the message."

motivated to exploit synthetic ecosystems and exploit societies of simple agents. This thesis also observes the lack of research progress is the area of fault tolerance; i.e. the ability of agent systems to deal with the failure of groups of agents. A related problem -- failure on the logical level -- concerns the need to upgrade agent behaviour because of observed deficiencies in current performance.

1.2.3 Representations and Forms of Computation

The previous section alluded to problems of representation in agent knowledge and reasoning with that knowledge. Firstly, a symbolic view of the world, with knowledge represented as well formed formulae and manipulated using first order predicate logic has a fundamental problem: not all properties of a system may be inferred using it [Gödel 31]. Secondly, and far more importantly from a practical perspective, representing dynamic environments results in the Frame problem; a problem that arises as a consequence of the need for specifying state and state transitions. Symbolic systems cannot be situated, as we do not live in a symbolic world; translation from measured data to symbolic representation has to take place. This process results in a further problem, that of sensor fusion.

Connectionist systems, or activation-oriented systems, represent a contrasting style of knowledge representation and reasoning. While avoiding the Frame and Sensor Fusion problems, they suffer from the problem of identification of where knowledge is actually stored. As such, the reasoning process; namely, spreading activation, is difficult to follow and makes an explanation of system activity hard to extract. It is this desire to synthesize symbolic and activation-oriented system representations within a distributed problem

solving framework that motivates the research in this thesis. The question is how to achieve synthesis?

While representation and reasoning influence the way the world is modelled and perceived, the underlying style of computation is the same. Styles and theories of computation have long been dominated by the ideas of Turing and Von Neumann. Until recently, these ideas were thought to be unimpeachable. In the 1990's, the landscape of alternate forms of computation has widened considerably. Firstly, in 1994, Leonard Adelman published a paper in Science describing the "Molecular Computation of Combinatorial Problems." While the Belousov-Zhabotinski reactions had clearly demonstrated that spatial and temporal self organization could produce striking physical effects, potentially being useful as information-processing vehicles, Adelman's work demonstrated actual computation; a DNA computer had been constructed. Arguably, this work can be traced back to Turing's work on Reaction-Diffusion systems and their acknowledged value as information processing systems. Secondly, and coincidentally also in 1994, Shor published a paper entitled, "Algorithms for quantum computation: Discrete logarithms and factoring" wherein an algorithm for factorization of a polynomial using a quantum computer was described. This was important because it demonstrated that the Quantum Computer -- proposed by Deutsch in 1985 -- could do something useful. Quantum Computing was born. In both of these developments we observe massive parallelism with no central coordination. In a DNA computer we see huge numbers of molecules interacting in parallel. In a quantum computer, the interaction of countless quantum states is used to solve a problem.

Both forms of computation rely on a form of reaction and local interaction. In the case of a DNA computation, a reaction is directly observable. In the case of quantum computing, the reactants are quantum states and reactions are the time evolution of the supposition of those states. A reaction has two important characteristics, those of symbol transformation and activation spreading through chemical concentration. Given these observations, and the proven utility of the computational power of chemical systems, we are motivated to use it as a basic block in the computation performed by an agent.

1.3 Contributions

The motivational principles outlined in the previous section drive contributions claimed for this thesis. They are described in the following sub-sections.

1.3.1 Agent Architecture

This thesis proposes an agent architecture that supports distributed problem solving using mobile agents. Problem solving is an emergent property of the system, no single agent is allocated the task of solving a given problem. The problem solving process is driven by the positive feedback (reinforcement) of the actions of many simple agents. The agent architecture explicitly includes the migration strategy of the agent and is strongly influenced by chemical communication in insects.

The contributions in the areas of architectural components are outlined in the next three subsections.

1.3.1.1 Chemicals

The most important contribution of this thesis is the synthesis of activation-oriented

systems and symbol re-writing systems. This synthesis takes place in the idea of a chemical object: a symbol with floating point value -- its concentration.

1.3.1.2 Chemical Reactions

While the chemical provides a concept that allows for symbolic representation of state, transformation of symbols occurs through reactions. These reactions implement a symbol re-writing system and, being active[1], cause transformations to occur in parallel, consequently ensuring that an agent is not biased toward any particular implementation of that computation. We believe this contribution, along with that of the use of the chemical concept, to be the most important in the thesis.

1.3.1.3 Migration Decision Functions

Although a great deal of research has been undertaken in the area of mobile agents, little or no attention has been focussed upon the formal specification of the itinerary associated with a mobile agent. Specifically, there has been no linking of the environment to the migration decision function in the way proposed by this thesis. Further, the identification of the value of societies of agents and the importance of mobility in distributed problem solving is captured in the migration decision function and the chemical communication interaction model proposed in this thesis.

1.3.2 Synthetic Ecologies of Agents

As stated previously, agent based systems have traditionally exploited rational agents -- agents with a knowledge of self and identifiable goals and beliefs -- in their problem

1. A reaction may be thought of as a daemon or thread.

solving processes. This thesis contributes to the on-going debate of agenthood by demonstrating that by engineering ecologies of agents, simple non-deliberative agents can successfully solve complex problems. This thesis demonstrates that models of agent behaviour taken from naturally occurring multi-agent systems can be exploited successfully in network problem solving, the interaction with the environment being key to its success. SynthECA -- synthetic ecologies of agents -- represents an approach to agent system design wherein agents are simple, the environment is closely coupled with the agents themselves, interactions are local and no global state is assumed.

1.3.3 Demonstration of subsumption for SynthECA

Previous multi-agent system work has focussed on groups of agents solving a single problem within a specific domain, although layered architectures have been proposed where problems of increasing abstraction are addressed [Hayzelden 99]. These architectures, in so far as they have been used, have hard-wired connections between layers of the architecture and that connections are internal, not external. However, this thesis demonstrates the use of a subsumption architecture [Brooks 86], [Brooks 91] for a synthetic ecology of agents where there are no hard-wired interactions between architectural layers; interactions occurring through the environment using chemicals as the inhibitory or excitatory stimuli.

1.3.4 Design of a mobile agent framework for SynthECA

This thesis provides a design for SynthECA agents, extending an existing mobile agent framework, with the most important element being the mechanism for communication of

chemical concentration changes. A reactive tuple space implementation is proposed for the local environments used in SynthECA, the design being strongly influenced by LINDA [Gelernter 86] and more recent developments of it used in the Mobile Agent community [Cabri 00].

1.3.5 Problem Solving Algorithms for SynthECA

Any architectural specification without demonstration of its utility in the context of problem solving in some domain is of limited interest. This thesis contributes a number of algorithms for problem solving in the five areas described in the following sub-sections.

1.3.5.1 Routing

Routing without knowledge of network topology is a hard problem, made more complex when constraints on the route to be taken are added. This thesis provides algorithms for point-to-point, point-to-multipoint, and shortest cycle using SynthECA agents.

1.3.5.2 Fault Location

Components responsible for degraded performance are difficult to find in networks. This thesis proposes simple algorithms for fault localization based upon hill climbing in the space of chemicals deposited in the environment. Learning based upon heuristics and reinforcement learning techniques are proposed.

1.3.5.3 Planning

Realizing when a network needs to be re-planned or a transient fault exists in the network is a hard problem. This thesis proposes a number of simple algorithms that deal with the identification of unreliable components and congested regions of the network

which should be re-planned.

1.3.5.4 Agent Density Control

If mobile agent systems are to be self organizing, the problem of self regulation of agent density needs to be addressed for societies of agents. Algorithms are proposed for the self-maintenance of agent density that are loosely based upon Lotka-Volterra dynamics of predator prey systems.

1.3.5.5 Agent Upgrade

If mobile agents are to be permanently deployed in networks and circulate continuously, it is certain that evolved behaviour will be required after some time. This presents three problems. Firstly, how can an agent with the updated behaviour be injected into the network and replace the current agent? Secondly, how can the new agent know that it has subsumed the responsibilities of the previous version of the agent? Finally, how can a previous version of an agent be prevented from re-asserting itself in the network?

This thesis proposes algorithms that address the above problems.

1.4 Thesis Organization

The remainder of this thesis consists of six chapters. Chapter two provides a wide range of background material upon which this thesis is based. Agents, their mobility and forms of communication figure prominently in this chapter as do examples of naturally-occurring systems exhibiting complex problem solving with purely local interaction.

Chapter three presents an agent architecture that is biologically-inspired and based upon

a new problem solving paradigm, that of mobile agents. It analyses the essential characteristics of self organizing systems and demonstrates how the proposed architecture supports emergence.

Chapter four applies the agent architecture of the previous chapter to the problems of routing in networks, including point to point, point to multipoint and protected path routing. The chemical formulation of the previous chapter is applied to demonstrate how multi-priority routing can easily be achieved. Finally, an algorithm that demonstrates how applications with complementary statistical properties can learn to share the same path is presented.

Chapter five provides a fault localization learning algorithm using the architecture of chapter three and couples it with the algorithms presented in chapter four, thereby demonstrating a simple subsumption architecture. The algorithm uses Q-Learning in conjunction with chemical gradient following in order to determine where faults are located. Upon successful diagnosis of the fault, feedback is provided to the environment in order to have future connection requests avoid unreliable components.

While mobile agent systems are appealing for the management of networks, they present management problems of their own. Chapter six discusses the mobile agent management problems from a resource and functional statement, presenting algorithms for the solution of agent density and agent upgrade problems. Issues of mobile agent security are not discussed here as they are well-addressed elsewhere. See, for example, [Mole].

All works of this kind generate as many questions as they provide answers; this thesis

being no different in that regard. In fact, given the wide-ranging synthesis of ideas presenting in the chapters that follow, the body of future work that remains is vast. The conclusions chapter of this thesis provides a number of dimensions to the work that should follow, spanning theory, practice and application. We believe that this thesis takes a single step on a very long journey. However, we believe that many giant strides can be taken by exploitation of ideas from the theoretical work of Prigogine, Eigen, Schuster and others along with the practical research of the rapidly-growing Mobile Agent community.

CHAPTER 2 *Thesis background*

2.1 Overview

This thesis, as is the case for all theses, builds upon the work of others. This chapter introduces material upon which the contributions of this thesis are built and places relevant material in a context where the work reported here is clearly an extension or synthesis of previous ideas. The objective of this chapter is to have the reader understand the areas of knowledge and ideas that contribute to the research reported in this thesis and how they relate to one another.

In order to understand the relevance of the background material, we must first re-iterate aspects of the objectives for the thesis. First, as we have seen in the previous chapter, symbolic multi-agent systems suffer from a number of limitations that betray their origins in Artificial Intelligence. Agents in such systems are rational, having a knowledge of self and a set of goals and beliefs. Limitations of symbolic multi-agent systems include a lack of robustness with respect to individual agent failure, difficulties in dealing with the

Frame problem (i.e., dynamic environments), and problems related to agent coordination. Second, the power of mobile agents as the primary entities in a new distributed problem solving process has yet to be realized.

This chapter first provides a definition of an agent and describes several of the more important agent architectures, choosing the reactive/planning nature of the architecture as discriminant. A section is then provided that reviews several of the more significant agent coordination and control mechanisms that have been developed. Having introduced agenthood in the context of classical symbolic agents, a section then introduces systems that exhibit collective (or swarm) intelligence and where problem solving is considered an emergent property of the system. Systems exhibiting collective intelligence often rely upon mobility and local interaction for their problem solving capabilities and thus it seems natural that the next section should describe the ongoing research into Mobile Agents. By doing this, the ideas of societies of agents and technological exploitation of those ideas are demonstrated.

Being a thesis, and one provided in support of the awarding of a doctor of philosophy degree, it seems appropriate to provide philosophical underpinnings of the ideas presented in later chapters. We do this by briefly describing the work of Varela and Maturana; in particular, the concepts of Autopoiesis and Autonomy. These ideas are important and receiving increasing attention within the Systems Science and Artificial Life communities in that they provide a holistic view of systems and specifically an understanding of the cognitive properties of living systems and societies of agents.

Having appealed to the use of simple, biologically inspired agents and, having noted the existence of alternative forms of computation in chapter 1, research in the area of chemical forms of computation is then presented. This section brings together several pieces of work, spanning Theoretical Computer Science, Theoretical and Algorithmic Chemistry and Self Organizing Systems. The section on Chemical Computation is arguably the most important for understanding the ideas presented in the chapters which follow. Finally, a section on adaptation and learning briefly reviews techniques which are used to enhance the performance of several of the algorithms presented in later chapters. While adaptation and learning are not central to the contributions made by this thesis, it is intended that this work form the initial study of synthetic ecologies of chemical agents, where agents have fixed chemical constitutions. Future work would allow for evolution of agent constitution using, for example, techniques taken from research into Evolutionary Computation (EC) such as Genetic Algorithms (GAs) or Genetic Programming (GP).

Finally, a section provides a summary of the research described in the chapter and how it relates to the chapters that follow.

2.2 Agents and Agent Architectures

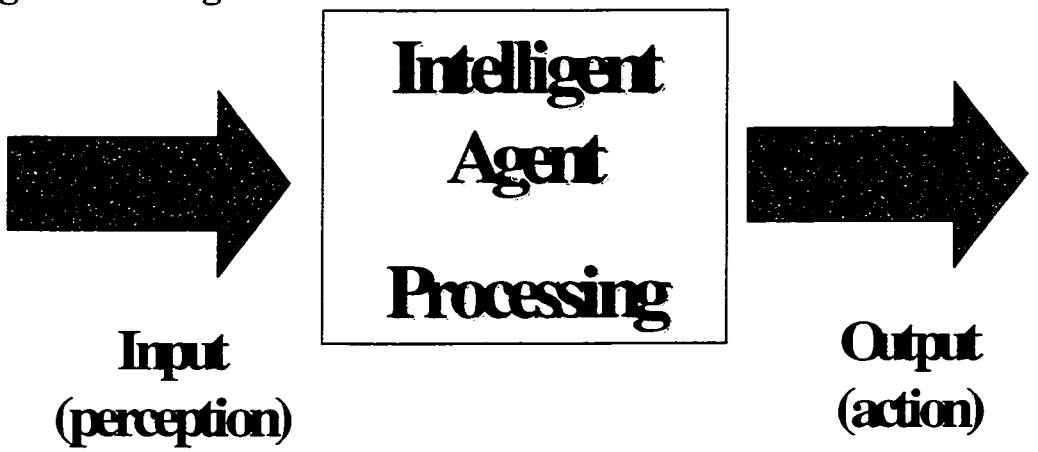


FIGURE 1. **Agent Black Box Architecture**

Figure 1 shows the black box architecture of an agent. Agents are considered to be perception - action entities and are active, having their own independent thread of control. Before introducing background knowledge relating to agents and their architectures, it would seem appropriate to define the essential characteristics of an agent. Unfortunately, no universally agreed definition exists. However, the following characteristics are generally agreed upon as being either mandatory or optional:

- Something that can act on behalf of others.
- Is social, capable of meaningful interaction with other agents (and humans).
- Can make decisions on our behalf.
- Is capable of adapting to changing environments and learning from user interaction.
- Is capable of dealing with unexpected situations.
- Is mobile.

A possible definition has been given as:

"An intelligent software agent is defined as being a software program that can perform specific tasks for a user and possessing a degree of intelligence that permits it

to perform parts of the tasks autonomously and to interact with its environment in a useful manner" [Brenner 98].

An agent, regardless of definition, has three processing phases as shown in Figure 2. The interaction box to the left of the figure represents the actual measurement of some set of aspects of the environment. Processing of this raw input then occurs. First, various sensory inputs from the environment need to be transformed and fused in order to provide input to the information processing component. The information processing component performs whatever reasoning (simple or complex) that is appropriate for the agent in its intended problem domain. The reasoning mechanism depends heavily upon the input it receives. For example, in a symbolic problem solving domain, the information processing component might use forward chaining as a reasoning process, abductive reasoning or an assumption based truth maintenance system (ATMS). In a connectionist agent, a neural network might be used to implement the reasoning process. The result of information processing is a view of the world consistent with the measurements made in the environment. The action module in an agent is responsible for selecting an activity -- a sequence of actions -- that should be undertaken in the environment in order to achieve a particular goal or move the agent closer to the achievement of a longer term objective. The

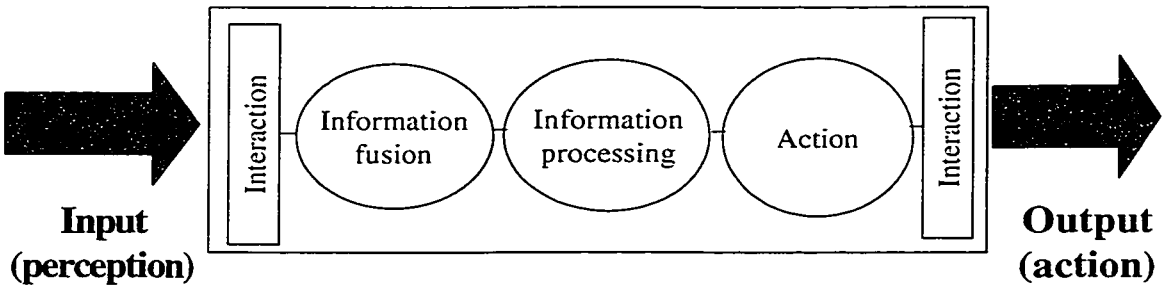


FIGURE 2. **Agent Processing Stages**

representation chosen for the action selection module depends heavily upon the output of the information processing module and, in most cases, uses a similiar representation to it. Hybrid symbol-neural network architectures are rare. The interaction element to the right of Figure 2 represents the conversion of an activity selected by the action mobile into a change in the environment; e.g., moving a leg for a robot.

The following sections present a number of significantly different views of an agent with respect to the processing that it performs in order to select an action.

2.2.1 Belief, Desires and Intentions

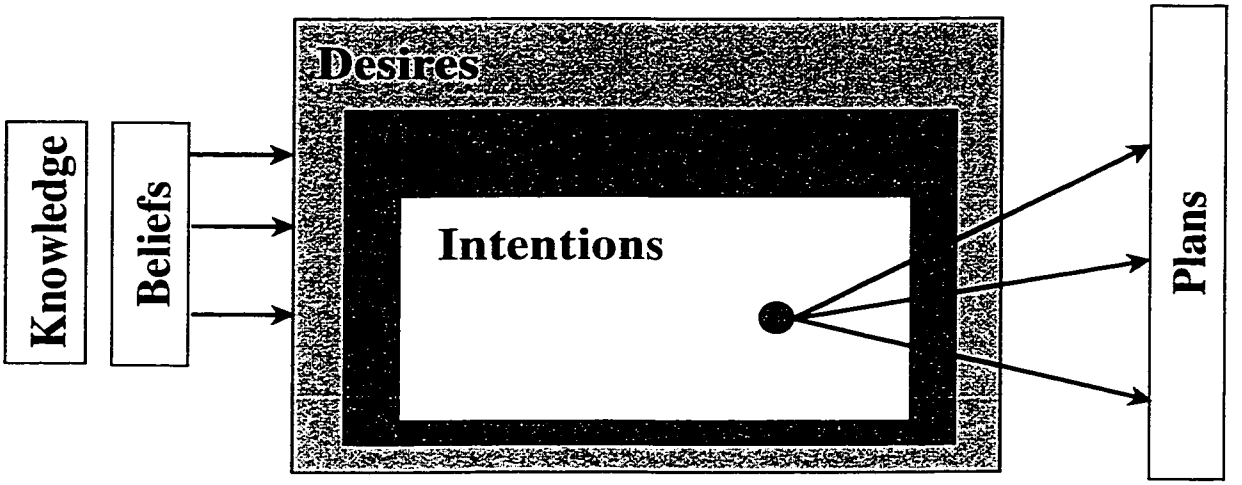


FIGURE 3. **The Belief Desires and Intentions Agent Architecture**

The Beliefs, Desires and Intentions (BDI) architecture shown in Figure 3 is a symbolic architecture due to Georgeff and Rao [Rao 95]. Agents of this type are deliberative and possess a symbolic model of their environment. Beliefs contain the fundamental views of an agent with regard to its environment. An agent uses them to express the expectations of possible future states. Desires are derived from beliefs directly. They contain judgements

of future situations. An agent might, for example, have the desire that a future state occur but another does not. An agent with its formulation of desires has not made any statement on the extent to which these desires are realistic. For example, I may desire to purchase an expensive car (e.g., a Ferrari) but I have yet to acquire sufficient price information and money in order to realize that desire. In other words, an agent may have an unrealistic desire, even though it knows that it may never occur. Conflicting or contradictory desires are possible.

The goals of an agent represent that subset of the agent's desires on whose fulfillment it should act. In stark contrast to its desires, an agent's goals must be realistic and cannot be contradictory. They must be realizable within the processing scope of the agent because they represent those processing alternatives available at a specific time and must be dealt with within a certain window of opportunity. Intentions represent a subset of goals. If an agent decides to follow a specific goal, this goal becomes an intention. An agent may not possess sufficient resources to follow all goals simultaneously, it must assign priorities to outstanding goals and process them accordingly. Finally, plans combine the agent's intentions into consistent units. Obviously, there is a close connection between intentions and plans: intentions form the sub-plans within an overall agent plan. The interation of BDI elements implies a basic architecture for such an agent and this is shown in Figure 4.

The agent's knowledge base is its symbolic view of the environment. The desires, goals and intentions are derived from the knowledge base. This activity is performed by the reasoner, thereby making this a centralized problem solver. The planner takes the

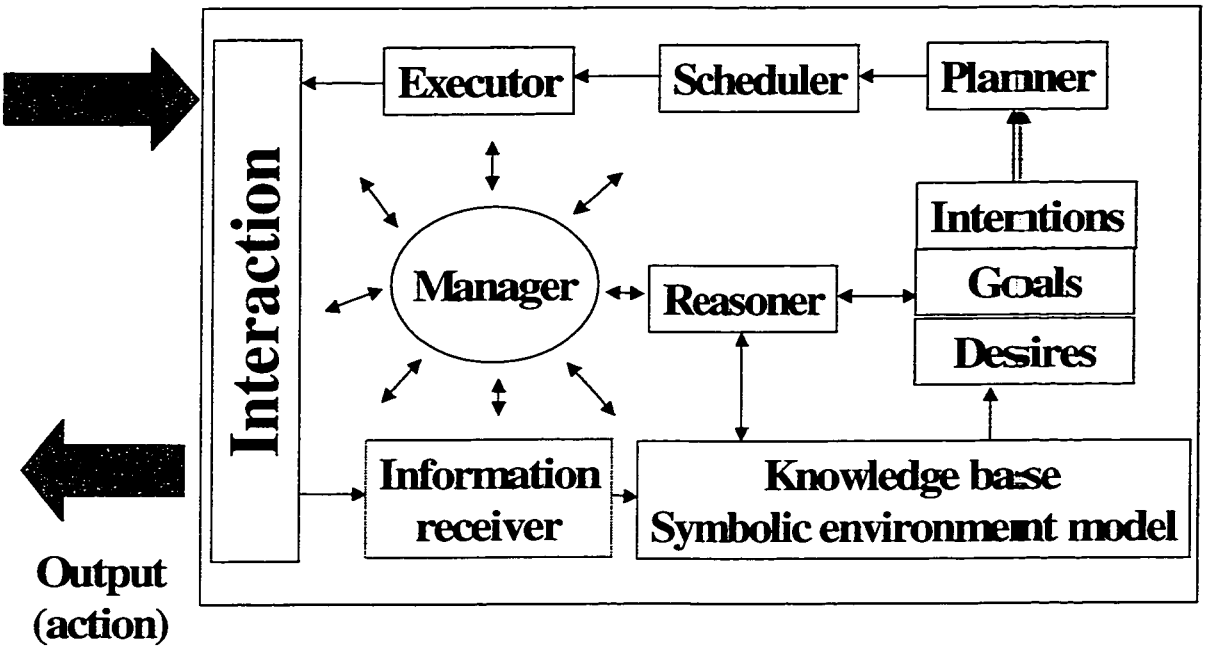


FIGURE 4. **BDI Architecture**

intentions and combines them into a consistent overall plan. Naturally, this is a dynamic, incremental process. The planner tests new intentions for dependencies with existing sub-plans. It may be the case, for example, that the results of an intention may represent the input values of another intention -- a strong dependency. The planner recognizes and makes allowances for this form of dependency. Existing plans are continually adapted to situations that result from the arrival of new intentions.

The scheduler receives the current plans from the planner. Every plan consists of a number of "atomic" actions that must be processed either sequentially or in parallel. The scheduler's job is to decide when specific actions are to be made available for execution. In order to do this, it needs a continuous overview of the resources available to the agent. The scheduler assigns an optimum and latest execution time to every action. The

scheduler also specifies details on the maximum runtime and resource usage. In this regard, it performs very similiar functions to those of an operating system process scheduler.

This information is passed along with the action to the executor module. The executor executes the next outstanding action, monitors its correct processing and terminates its execution. The executor can terminate an action if it requires more computer time than provided by the scheduler. The executor returns the action to the scheduler or planner if it cannot be started before the latest specified time.

As can be seen in Figure 4, these architectures tend to be highly centralized, the manager component orchestrating agent problem solving activities. As such, they rely on global, symbolic knowledge; a characteristic that we find undesirable for robust problem solving. There is also a significant discontinuity between the sensed environment and its manipulated representation which tends to be symbolic.

2.2.2 Subsumption

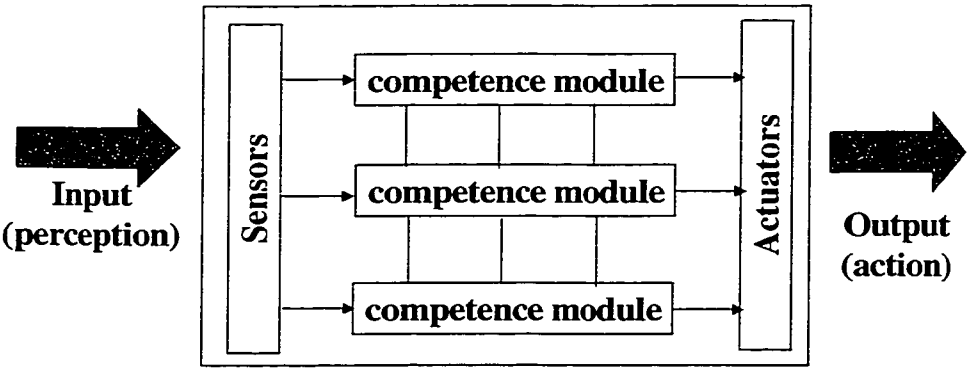


FIGURE 5. **Subsumption Architecture**

In contrast with BDI architectures, subsumption architectures (as shown in Figure 5) do not possess a symbolic representation of their environment. Also, in contrast to deliberative architectures that rely on theorem proving and symbolic manipulation, reactive architectures do not derive their intelligence from internally generated views of the world but from manipulation of direct sensor input. Brooks [Brooks 91] maintains that intelligence exists and increases only through continuing interaction of systems and is not an innate property of individual systems (as is the case with deliberative agents).

A reactive agent need not have a complex structure to act in a complex environment. What is needed is careful observation of the environment and recognition of simple principles and dependencies among them. From these observations, knowledge can be derived that allows the development of task-specific modules that continuously monitor their environment for the occurrence of specific situations, initiating a direct reaction when such a situation is detected. Reactive systems are, then, stimulus-response systems.

Figure 5 shows an architecture of a subsumption-style agent. In it, sensors record information and forward it to task-specific competence modules that react to patterns of sensor input. Connections between competence modules either inhibit or excite the activity of modules about or below them, ensuring that competition for permission to act is resolved in a way that prevents unwanted cyclic or contradictory actions. The output of a competence module is passed to an actuator which transfers the action to the environment.

Subsumption architectures are, therefore, parallel machines. Each competence module operates concurrently; synchronization between them cannot be assumed by simple order

of execution. As such, many dependencies exist and these necessitate action coordination through the links between competence modules. Each competence module within a subsumption architecture is implemented using the principles of Augmented Finite State Machines (AFSM). An AFSM initiates a response as soon as its input signal exceeds a pre-determined threshold value. AFSMs are pure computing units, no symbolic representations or world models are contained in them.

Synchronization of module activities is achieved through a use of inhibitor and suppressor nodes as shown in Figure 6. A suppressor node operates on the signals received by a module and can modify them as required. An inhibitor node can inhibit the output of a specific signal for a certain period of time.

The Subsumption architecture has also been augmented with the so-called Behaviour Language. In it, variables can be associated with groups of AFSMs that are active for a period of time. Registers -- equivalent to enumerated types -- are also provided. However, even with these state-preserving (i.e., memory) mechanisms, substantial limitations of such layered reactive layered architectures have been shown to exist [o.V. 97a], [o.V. 97b].

Maes [Maes 89] has described a similiar architecture -- spreading activation networks -- in which action selection is modelled as an emergent property of the activation/inhibition dynamics among these modules. Maes' spreading activation architecture has a well-developed mathematical model that makes it appealing from an analytical point of view and provides support for learning through dynamic modification of the "strengths" associated with module interactions. However, as observed by Maes herself, spreading

activation networks seem highly dependent upon sensitive global control parameter settings for successful problem solving.

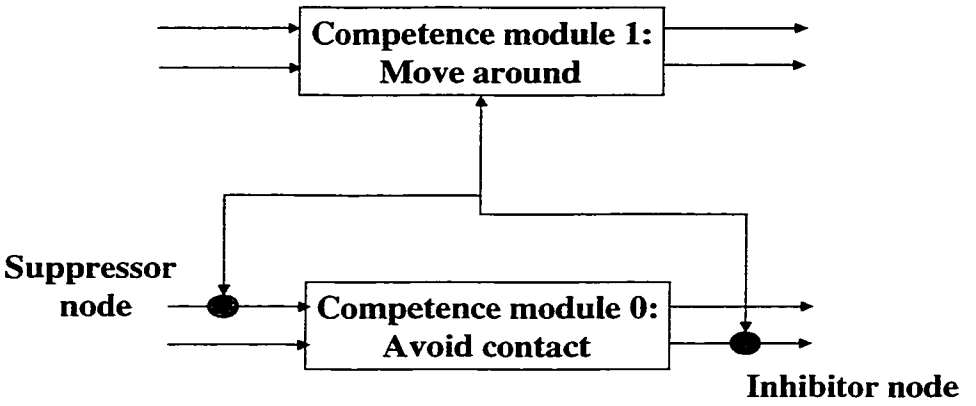


FIGURE 6. **Example layers in a subsumption architecture**

Subsumption architectures are highly relevant to this thesis as they provide for coupled reactive systems that react at differing levels of abstraction. The architecture described in the next chapter depends heavily upon small, reactive agents that are coupled not in a hard-wired way as described here but rather through a "soft" mechanism of chemical reactions that can, if required, vary with time. The connections, as we shall see, are local concentrations of chemicals that are sensed by individual agents.

2.2.3 Hybrid

The previous two sections have highlighted deliberative and reactive agent architectures, these representing the architectural extremes in terms of reaction time. Naturally, elements of these extreme architectures can be combined; two examples of hybrid architectures are provided in the next two subsections.

2.2.3.1 Interrap

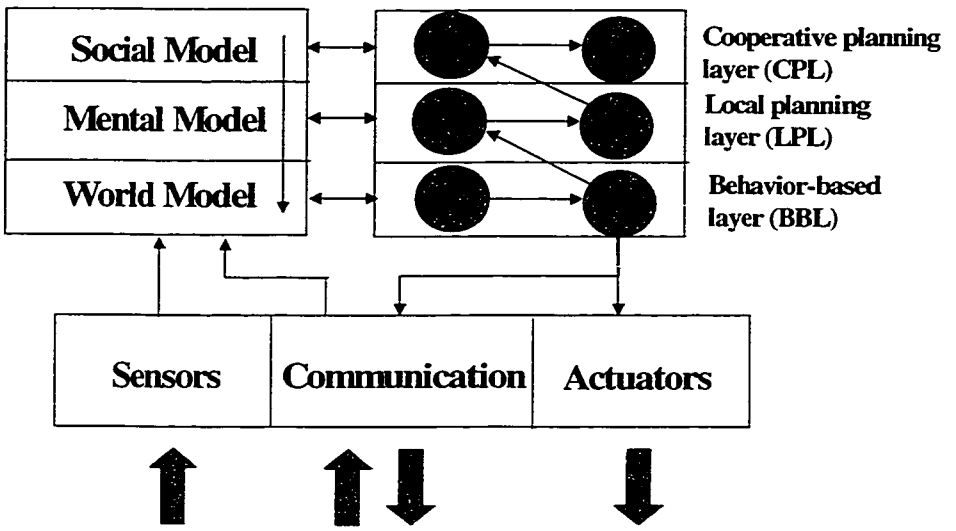


FIGURE 7. **Interrap Architecture**

The Interrap architecture [Muller 96] combines reactive and deliberative architectures in three layers. Interrap is based upon the BDI model, but informally so. As with other BDI models, sensor input forms the basis for agent beliefs.

The beliefs received by an Interrap agent can be grouped within three separate models (see Figure 7). The world model contains the fundamental beliefs relating to the agent's environment. The mental model consists of beliefs that the agent has of itself. The social model consists of beliefs that the agent has of other agents within a multi-agent system. The beliefs contained within the world model are used, for the most part, in a reactive mode. The beliefs in the mental model are used for deliberative purposes (i.e., planning of actions related to self). The beliefs in the social model are used in order to derive actions related to the cooperation with other agents.

The Interrap architecture uses an algorithm to derive situations from general beliefs.

Situations are a subset of beliefs representing states of concrete interest to the agent and are used to drive the goal. Situations can be used by one or more layers depending upon their generality. For example, a local planning situation is based upon (and can be used by) the world and mental models whereas a cooperative situation might be used by all three models.

The Interrap architecture shown in Figure 7 clearly shows that knowledge as well as control is multi-layered. The control process is bottom-up; i.e., a layer receives control over a process when this exceeds the capabilities of the layer below. The local and cooperative layers are used when the situation exceeds the capabilities of the reactive layer. Essentially there is a single thread of control which is passed from one layer to the next up the hierarchy of abstraction.

Each control layer consists of two modules: the situation recognition/goal activation module (SG) and the planning scheduling (PS) module. The SG module performs all steps as shown in Figure 4 up to planning and scheduling with the PS module forming and scheduling a plan to achieve the desired goal. It should be noted that the behaviour-based layer only "sees" the world model whereas the cooperative layer has access to the entire knowledge base consisting of social, mental and world models.

The execution process acts in the opposite direction to the control process; i.e., it passes from the cooperative layer through the local planning layer to the behaviour-based layer. Only the behaviour-based layer has access to the actuators in order to effect changes in the environment. This element of the architecture makes it significantly different from

Brooks' subsumption architecture in which each competence module must compete with other modules for control of actuators and sensors.

It should be noted that the interactions between the PS modules of one layer and the SG modules of the layer above it enable interruption of the execution of a plan in progress. This facility is *essential* for re-planning in a rapidly changing environment where the behaviour-based layer tends to dominate agent actions.

2.2.3.2 TouringMachine

The TouringMachine [Ferguson 92], [Ferguson 95] architecture shown in Figure 8 also possesses three distinct layers organized in the same way as the Interrap architecture of the previous section. However, TouringMachines have a single control framework rather than a hierarchical control framework and all layers compete for control of sensors and actuators (action effectors in Figure 8). Also, the inter-layer connectivity shown in Figure 8 shows that the layers do not represent increasing levels of cognitive abstraction (as suggested by Simon) used in the Interrap architecture.

The reactive layer of a TouringMachine agent provides it with stimulus-response type behaviour as with the Interrap agent. Stimuli for this layer comes purely from sensors. When a rule fires, messages are sent to the modelling and planning layers in order to determine if further processing is required. Also, the control framework "approves" the action associated with the rule before being sent to the agent's effectors.

The purpose of the planning layer is to generate and execute plans. As such, its architecture is quite similiar to that shown in Figure 4. However, in a TouringMachine,

plans are constructed hierarchically, using partial planning, with a least commitment strategy and through access to a schema library rather likes cases in a Case-Based Reasoning system.

The purpose of the modelling layer is to provide an agent with reflective and predictive capabilities. The agent realizes such capabilities by constructing cognitive models of world entities, including itself, which it uses as a platform for explaining observed behaviours and making predications about possible future behaviours. The modelling layer has access to a library of model templates (again case-like) and uses abductive reasoning with conflict resolution for planning.

TouringMachines differ from Interrap agents in that no *explicit* modelling of cooperative

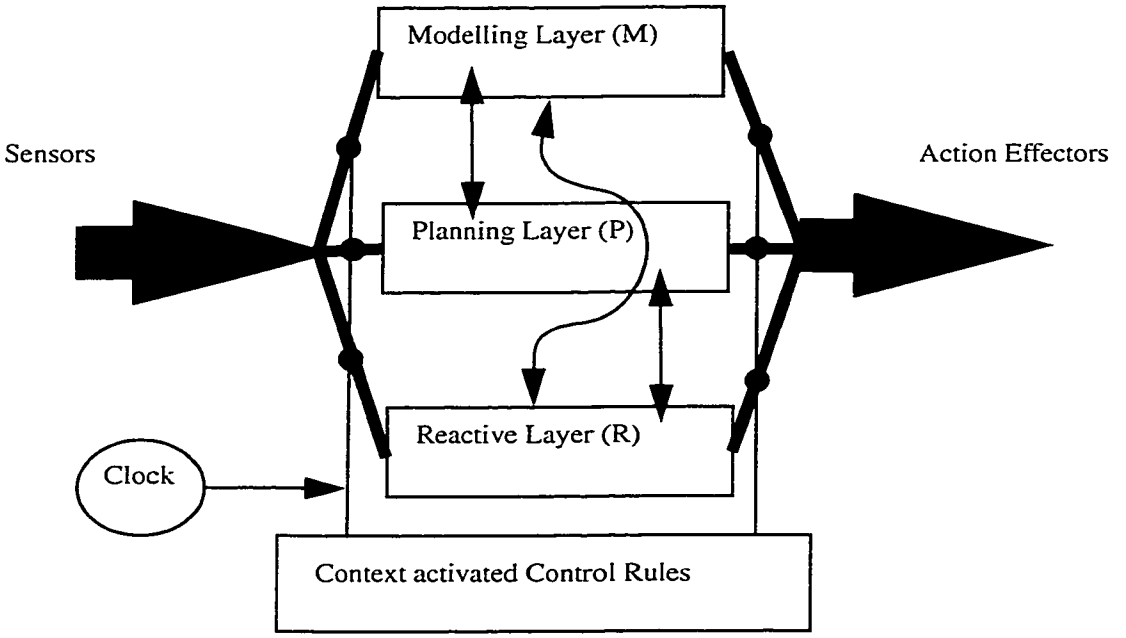


FIGURE 8. **Layer connectivity in TouringMachines**

behaviour is present. If present at all, TouringMachines include models of self and cooperation with other agents in the modelling layer. We believe the separation of models of self and models of cooperation as present in Interrap represent the superior architectural choice.

The importance of the TouringMachine architecture within this thesis is its layered character with direct coupling of sensors to the various reasoning components; facets exploited in the next chapter. The control framework is, unfortunately, global which dilutes the value of the architecture in our view. Also, the claims of layering are somewhat contrived as a result of the communication between reactive and modelling layers.

2.2.4 Mobile Agents

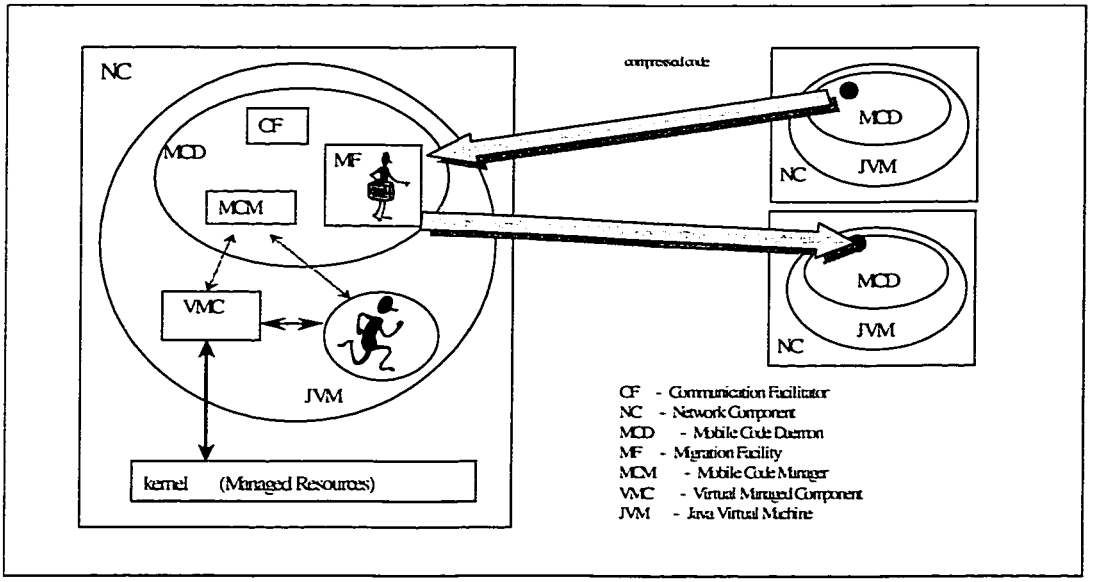


FIGURE 9. **Mobile Agent Architecture**

This thesis deals with mobile agents, agents that move in a network in order to solve problems in a network environment. Mobile agents, we believe, can be viewed in a

number of ways. We can think of mobile agents as moving the computation to the data or, if we concentrate on the interaction of a large numbers of such agents, as an alternative problem solving mechanism where no single agent solves the problem but the solution emerges through the interaction of many simple agents with their environment.

Requirements for a mobile code framework address six fundamental needs: support for legacy solutions, portability, persistent state, security (including secure code distribution and access control lists), access to resources of visited systems and inter-mobile code communication. Due to the fact that substantial funds and effort have been invested in operating network management systems, support for legacy solutions is still needed.

Several elements are necessary for code mobility. We analyze them in the context of Java technology, which provides a very convenient implementation environment for mobile agents owing to the fact that Java virtual machines are supported on many platforms and a rich set of standard classes have been written (e.g., to support communications, remote method invocation, object serialization and security). Maintaining a single program for many platforms is preferred to maintaining a number of programs, each for a different platform. Originally, Java was designed to deliver a secure, robust vehicle to implement applications that run on many platforms in heterogeneous environments.

A mobile program may want to keep its persistent state during mobility. The program may run on one host, pause for a certain period of time and continue on to another host. The program should store its state during the pause and restore it when it continues to

execute. Java supports serialization that allows the state of classes to be stored in the file system or sent through an output stream, so the class can be reconstructed with the same state as before it was serialized.

A fundamental assumption is that every networked device is Java-enabled; that is, it runs its own JVM either as a software process or as a Java chip, or has a proxy that runs a JVM. An example of a mobile code framework architecture can be seen in Figure 9. The most important element of that infrastructure is the mechanism for code migration. The Mobile Code Daemon (MCD) runs as a daemon (thread) inside a JVM. The MCD listens on well-known ports for messages that carry the compressed Java bytecodes of the migrating software agent. The Migration Facility (MF) is the part of the MCD responsible for actual shipment of the code. An agent's navigation model is realized through an agent's logic and interactions with the MCD and MF. When an agent arrives at a port, it is has to pass security checks before it is run as a Java thread. When its objectives have been met, the agent's code is then migrated to another location and the local thread is terminated. The migration is performed by the MCD under the agent's or default control; i.e. the agent maintains an itinerary or permits the daemon to select a migration destination for it.

Other parts of the infrastructure that do not directly support the navigation model include the interface to host resources, *communication facilitators* (*CF*), *security facilitators* (*SF*), *event managers (EM)* and a *naming facility* (*NF*).

Security has always been a major issue in distributed computing and especially so in mobile agents [Mole]. Downloading and executing an untrusted, unsecured program at an

end user's computer may expose private resources to malicious attacks. Taking into account the global nature of networks such as the Internet, security should be of serious concern on every connected computer. Untrusted programs should be authenticated and validated before they are allowed to execute. There are four major security services [Sander 97]: authentication, authorization, data integrity and data privacy. Java provides not only portability but also security features in its Virtual Machine (class loader and bytecode verifier) and an Application Program Interface (API) that facilitates the development of secure mobile agent implementations.

The ability of a mobile agent to access managed resources of visited systems is clearly necessary for a network management. Accessing data on a heterogeneous system may be complex due to non-standard naming and procedures to manage them. Therefore, there must be a standard way that is understandable to the mobile code in performing its tasks without prior knowledge of the underlying system. The framework provides a template that we call a *Virtual Managed Component* (VMC). The mobile agent accesses managed resources indirectly through the VMC. Together with the security facilitator, it constitutes the security model of our agents. The VMC implements access control to the resources of the network component in order to guard sensitive data. A mobile agent is not allowed to access the local file system, launch programs, call system level services of the visited network component, or invoke SecurityManager or ClassLoader classes. With these restrictions, the network component is reasonably safe from attacks of any malicious mobile code; i.e., a piglet. However, research on security in mobile agents is a fertile area of current research.

The fundamental function of the VMC is to interface to the managed resources of a network element, so a mobile agent can get or set their attributes. The interface to the managed resource facility of the VMC controls the access to the attributes of the managed resources. Due to the complexity and heterogeneous nature of network components, there is no standard way in which vendors implement the interface to the devices. The naming schema varies as well. Therefore, there must be a uniform language to allow a universal way of accessing the managed resources. The format of the MIB (Managed Information Base; e.g., the SNMP MIB) may be considered for this uniformity and standardization. If a vendor has implemented its own naming schema, it should provide a translator or interpreter that maps the vendor specific names to ones that are understood by agents. Another facility, which is a part of the VMC, is access control (i.e., authorization) to the managed resources, as well as the rights to extend the VMC. The VMC may also provide a management applet that can be downloaded to the manager and shown on the web browser in a vendor-specific way. Since a network component may have limited storage resources, the VMC may contain a redirection to the location of the applet in a designated remote repository. The vendor may also include recovery procedures in the VMC that may be accessible to other applications. The recovery procedures are used for diagnosis and recovery and can be controlled by a visiting agent or a management applet of the network component. Another facility provided is provisioning procedures. These procedures may be used for a plug-and-play capability of the network component, described in a later section. For instance, the network component may adapt itself to its network environment by automatically installing and configuring its driver that is downloaded from the vendor's

repository. The Jini framework [Jini] provides exactly this type of facility.

The infrastructure implements a security schema that is sufficient for most network management applications. The use of Java is convenient because it has provides a security package that is sufficient to implement four security facilities: authentication, data integrity, data privacy and authorization. The security facilitator makes extensive use of Java security facilities. The JDK 1.1 also comes standard with a default provider, named "SUN". The "SUN" provider package includes an implementation of the Digital Signature Algorithm, and an implementation of the MD5 [RFC 1321] and SHA-1 [NIST FIPS 180-1] message digest algorithm. The authentication uses DSA, which provides a pair of keys, a public key and a private key. The data integrity use of MD5 and SHA-1 ensures the integrity of mobile agent during its transmission.

The communication facilitator and the naming facility implement the communication model. To perform its task, a mobile agent may need to collaborate with other agents. Because of agent mobility, the main problem of the inter-agent communication is locating the recipient agent. To resolve this issue, a Naming Facility is provided. The server represents a secretary that always gets a report about the current location of active mobile agents from visited MCDs. Notification of events concerning phases of the lifecycle of an agent are forwarded to the Naming Facility in order that it can track the location of the agent. The NF also provides a mailbox for an agent that is not currently present for a number of reasons, such as the agent is in the process of migrating, the agent has left the network, or the agent has not been created yet. As soon as the recipient agent has migrated

or (re)appeared in the network, the message will be delivered to it.

There are two general mechanisms for inter-agent communication that have been implemented: a blackboard architecture and direct agent to agent communication. The blackboard is a placeholder where agents can post messages and other agents register for notifications of messages of particular types and content. The blackboard (see section 2.2.5.4) can also be a mobile agent in order that it can migrate to make best use of network computing and bandwidth resources. Direct agent to agent communication uses the CF which, in turn, uses the NF in order to locate an agent before delivering the message to the mailbox set up for the recipient agent on a particular network element.

2.2.5 Multi Agent System Coordination, Communication and Control

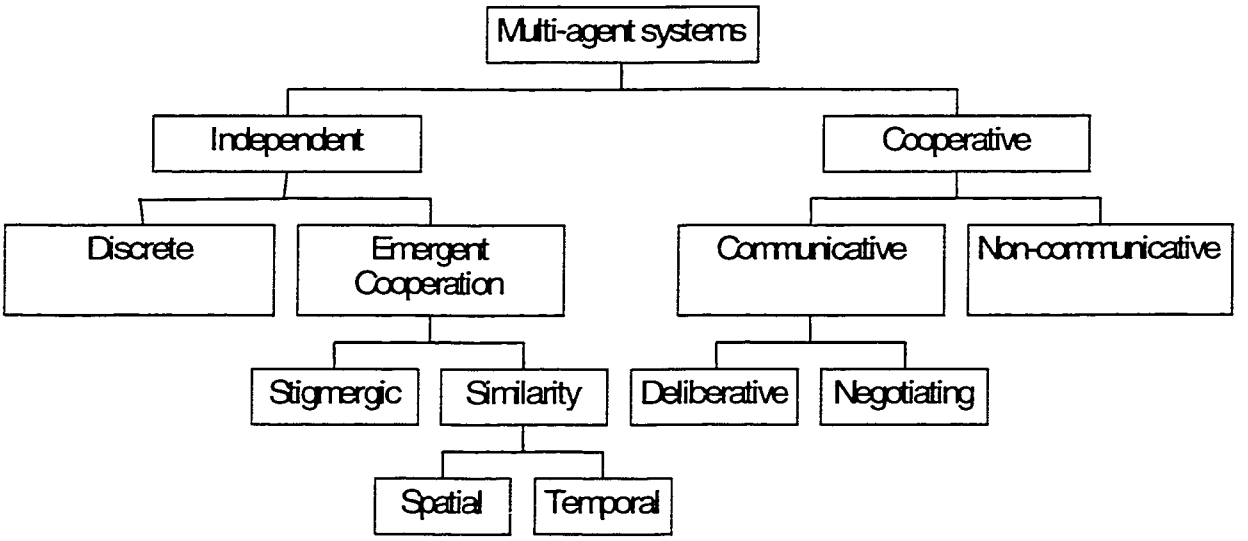


FIGURE 10. **Multi Agent System cooperation and Control Taxonomy**

A taxonomy of cooperation models for multi agent systems is shown Figure 10. This taxonomy is due to Doran [Doran 97]. Our interest in this taxonomy is in two subtrees: Communicative and Emergent Computation. In the sections that follow, examples of ideas

and technologies are briefly described.

2.2.5.1 Market-based control

An abstract definition of a market is a system with locally interacting components (agents) that achieve a measure of coherence when overall, or global, behaviour is considered. In some sense, behaviour is emergent and "problem solving" is decentralized but in reality the market-based mechanism relies of the principle of price stability in the economy. It is through the simple interactions of buying and selling, i.e., trading, that desirable global effects can be achieved. Examples of desirable interactions are stable prices or fair allocation of resources. It should be stressed that market-based systems do not guarantee optimality but are able to facilitate resource allocation with very little information, i.e., price. This makes them attractive candidates for use in complex domains where multi-agent systems are being considered [Clearwater 96].

A market based system of agents generally contains rational agents, this comment being based upon a review of [Clearwater 96] where prototypical applications in the areas of resource allocation in networks, operating systems, factory scheduling and building control are documented. The need for rationality is (in part) due to the limited information that is communicated between agents. Interestingly, it has been noted in [Steiglitz 96] and [Kephart 98] that purely rational agents acting in a market based system can cause wild fluctuations in price and that the addition of speculator (or noisy) agents [Steiglitz 96] can reduce amplitude fluctuations.

Wellman [Wellman 96] has introduced the methodology of market-oriented

programming (analogous to agent oriented programming) and implemented it in the WALRAS platform. Market-oriented programming is based upon the principles of the theory of general economic equilibrium. In market oriented programming, the metaphor of an economy directly computing the behavior of an agent in a multi-agent system is used literally; the distributed computation being implemented as a market price system. Stated another way, agents interact by offering to buy or sell commodities at fixed unit prices. When the system reaches equilibrium, the computational market had computed the allocation of resources throughout the system and dictates the activities and consumptions of the various agents. Herein lies the single biggest weakness of the approach, the implication that equilibrium is even possible. It is understandable that this assumption is made given that economics uses the concept of a *general equilibrium*. However, in reality, economic systems rarely achieve equilibrium and are always in a transient state.

While theoretical results (including convergence properties) are limited for systems implemented using market-oriented programming, the following qualitative observations have been made:

- The equilibration process scales well with the number of agents. *Convergence rate not exponential with number of agents.*
- The equilibration process scales non catastrophically with the number of goods. *Sub-exponential convergence rate with increasing number of goods.*
- Asynchrony reduces oscillation. *Noise is good.*
- Incremental bidding can simplify agent behaviour. *Agents only have price-quantity knowledge, not point-price as would be expected in a real system.*
- Non-convexities can be fatal, and often are. *Non-linearity causes severe convergence problems.*

The last point is the most damming of all. Market-oriented programming often fails

when non-linear relationships exist in the point-price curves. Non-linearity is the norm in most control or design problem scenarios and so this represents a real problem. However, the concepts of information ecologies and attractive properties from the computational point of view, make market-oriented programming an appealing architecture for control problems. It is this observation which makes market-oriented programming relevant to this thesis.

2.2.5.2 Contract Net Protocol

Contract net systems represent a concept that can be used to establish efficient coordination mechanisms between agents integrated in a multi-agent system. A contract net consists of a number of nodes that are formed by the individual agents in the multi-agent system along with a manager which has the responsibility for coordinating activity. As in a market place, a manager asks for bids on a set of pending subtasks and nodes respond to requests that interest them; i.e., tasks that they can perform with the constraints as specified. Task assignment is interactive; all nodes are involved. The goal is to use computational resources (i.e., the knowledge and reasoning capabilities of individual agents) in the most efficient way possible.

The Contract Net Protocol, shown in Figure 11, demonstrates the four phases of the protocol. A bid is first solicited from interested nodes. Each node deliberates on whether it can meet the requirements of the subtask and responds to the manager. Upon receipt of a number of bids, the manager awards a contract to one of the nodes. It may be the case that no bids are received in which case the contract is reformulated (relaxing conditions) and

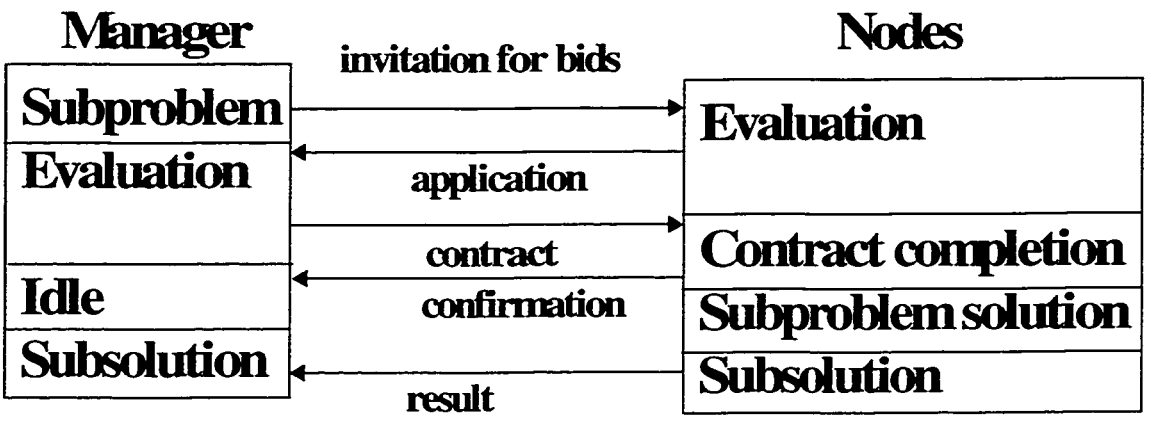


FIGURE 11. **The Contract Net Protocol**

re-broadcast. Upon receipt of a contract, an agent begins processing of the subtask. When complete, the manager is provided with the solution. The manager remains idle during subtask solution. When all subtasks have been solved, the manager assembles the subsolutions into an overall solution to the problem. An example of an initial bid, the bid response and the subsequent awarding of the contract are shown in Figure 12 below.

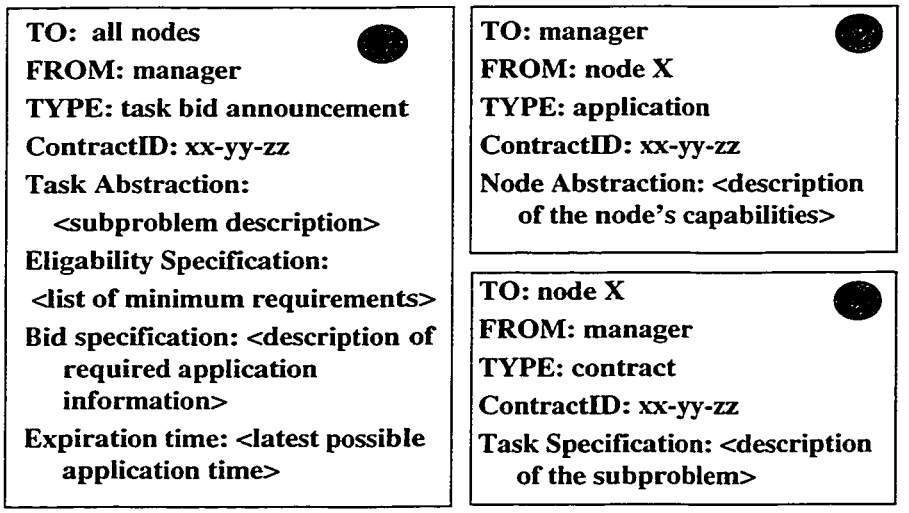


FIGURE 12. **Example Contract Net Protocol Interaction**

2.2.5.3 Knowledge Query and Manipulation Language

While blackboards and reactive tuple spaces provide a shared space for communication, they do not describe the syntax of the information stored there. In order to be useful, a common format needs to be adopted in order that all agents may understand it. This is the purpose of the Knowledge Query and Manipulation Language (KQML) [Labrou 97]. KQML is an example of an Agent Communication Language (ACL). KQML is based upon speech act theory and was the outgrowth of the ARPA Knowledge Sharing Effort. A KQML example is shown in Figure 13. Given the acceptance of the Federation for Industrial Physical Agents OS (FIPA-OS), Extensible Markup Language (XML) and Java code mobility, it has, at best, an uncertain future.

The *syntax* of KQML is not important from the perspective of this thesis. It is, however, included in order to demonstrate the importance of having a single representation for communication between agents. This will be the subject of further comment in the section on Autopoiesis and in the next chapter where the agent architecture proposed in this thesis is discussed.

```
(<Performative>
:content <statement/speechact>
:sender <name>
:receive <name>
:language <text>
:ontology <text>
)
```

Performative corresponds to speech act types.

FIGURE 13. **Example KQML**

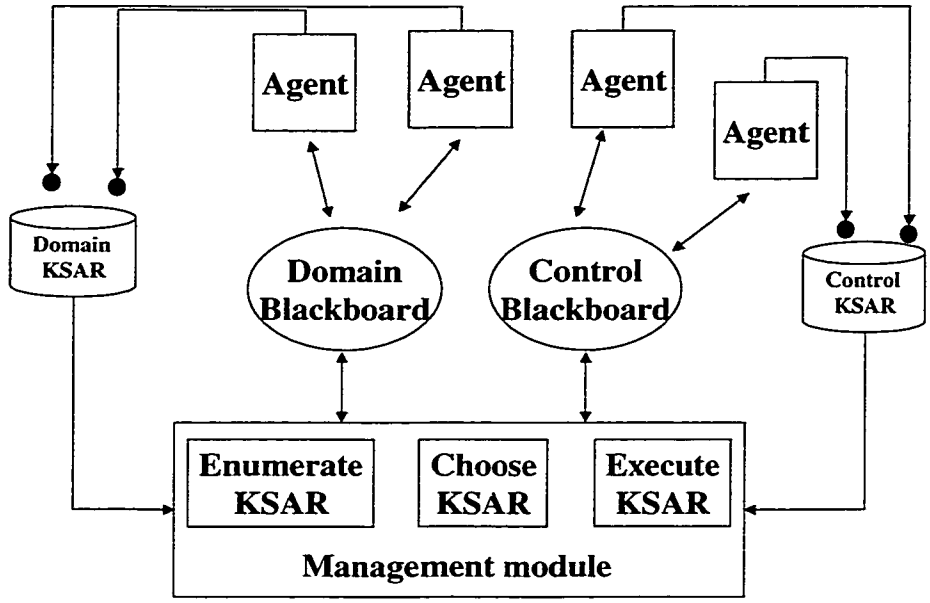


FIGURE 14. **The BB1 Blackboard Architecture**

2.2.5.4 Blackboard Systems

A blackboard system is used to support distributed problem solving by providing all agents within an agent system with a shared work area where they can exchange information, knowledge and data. Communication is not agent to agent but indirect via the blackboard (which may be thought of as shared memory). An advanced blackboard architecture is shown in Figure 14.

Inter-agent communication is achieved by writing something to the blackboard which is then available to other agents in the system. Agents register with the blackboard in order to receive notifications of messages of interest. Typically, blackboard systems contain regions, and an agent registers its interests with one or more of them. Agents may also poll the blackboard looking for items of interest. Naturally, all agents have to understand the same language when using this type of communication vehicle and KQML is often used in

conjunction with the Contract Net Protocol in the efficient implementation of a blackboard system.

The advanced blackboard architecture shown above includes a management component that deals with issues such as which agent should be chosen for solving a subtask and posting of subproblems on the blackboard. Agents report interests in specific subproblems in a Knowledge Source Activation Record (KSAR).

The BB1[1] architecture shown above provides for a dual blackboard architecture. BB1 is a software system, originally invented by Barbara Hayes-Roth in 1983, that embodies the "blackboard control architecture" for blackboard systems. In addition to the traditional properties of blackboard systems, BB1 enables an application to use a uniform reasoning method (event-based triggering and opportunistic control of reasoning operations) to build and modify explicit plans for its own behavior at run time. The domain blackboard deals with the solution of actual problems. Control information is stored in the control blackboard. The structure of the control blackboard does not depend upon the domain blackboard but is determined by system selection of the coordination and control strategy. Separate KSAR databases are maintained for domain and control blackboards. The management module for the BBI architecture performs the functions of: enumeration of all KSARs, KSAR selection, followed by KSAR execution.

While appealing, providing flexible cooperation and communication mechanisms, blackboards nevertheless have drawbacks. The centralized nature of the architecture imply

1. BB1 stands for "Black Board 1" as far as we can ascertain.

that registered agents have to place their information on the blackboard directly regardless of their network location. This can lead to excessive network load. As such, scalability is a real issue.

The importance of the blackboard concept to this thesis is that it provides for a shared communication medium, something required in SynthECA. Also, the blackboard is partitioned, allowing agents to receive only messages of interest to them. This characteristic is also important within the SynthECA architecture introduced in the next chapter.

2.2.5.5 Reactive Tuple Spaces

Related to the previous section, reactive tuple spaces are based upon ideas from Linda [Gelernter 86]. Linda is a parallel processing paradigm based on the concept of "generative communication", which unifies the concepts of process creation and communication. Linda is a coordination language which utilizes a concept known as *tuple space*. A tuple space can be thought as a kind of global associative memory. A tuple space stores objects called *tuples*. A tuple consists of a sequence of typed fields, for example: ("foo", bar, 6, 23.5). Linda provides four basic primitives:

- out(t): placing the tuple t in the space;
- in(t): to remove the tuple t from the tuple space;
- read(t): to access the tuple t in the tuple space non-destructively;
- eval(t): to evaluate the tuple t in the tuple space.

Tuples are associative; i.e., pattern matching occurs for tuples. Full unification may be supported, for example. Further primitives such as register(t) and deregister(t) have been

added to Linda-like tuple spaces in order to support an event notification mechanism.

A reactive tuple space extends the Linda concept by providing a meta tuple space. When a primitive operation occurs on a tuple within the domain space, the meta space is queried using the read primitive for a tuple of the form (ReactObj, Tuple, OpType, AgentID). Typically, the Tuple, OpType and AgentID will be fully qualified, the ReactObj is the reaction object which is associated with the tuple in the meta space. The associated ReactObj is returned to the agent that invokes the reactive tuple space primitive action.

The reactive tuple space mechanism allows tuples to be programmed as the reaction object returned has a standard interface that can be invoked by any agent communicating with the space. Interestingly, a number of tuple space implementations have appeared for mobile or multi-agent systems [Cabri 00], [Cabri 99], [Micmac] supporting the view that it provides an important communication mechanism for such agent systems.

Reactive tuple spaces are important to this thesis as the chemical formulation for communication used in SynthECA is implemented uses a tuple space model. Sensors and effectors, as we shall see in the next chapter, perform the functions of out(t) and read(t).

2.3 Swarm Intelligence

The notion of complex collective behavior emerging from the behavior of many simple agents and their interactions is central to the ideas of Artificial Life [Langton 87]. There are many examples in Nature of social systems where individuals possess simple capabilities which, when compared to their collective behaviors, are much more complex. Such systems span many levels of evolutionary complexity, from simple bacteria [Shapiro

88], to ants [Goss 90], [Franks 89], caterpillars [Fitzgerald 88] and beyond.

The principle of Swarm Intelligence is one that drives many of the contributions and ideas in this thesis. What is it and why is it important to this thesis?

Swarm Intelligence, according to Beni [Beni 89] can be defined as:

"Swarm Intelligence is a property of systems of non-intelligent robots that exhibit collectively intelligent behaviour."

Swarm Intelligence is important in a study of multi-agent systems as agent architectures have to organize themselves and adapt dynamically to changing circumstances. They must do this without an overseer; i.e., top down control from an administrator. It is possible to achieve self organization with rational agents (for example, BDI agents) that attempt to emulate human reasoning by maintaining models of self and by modelling the society of rational agents with which they negotiate. The Interrap architecture described in Section 2.2.3.1 is such an attempt at self organization without a central control element[1]. Rational agents, and rational multi-agent systems still suffer from the limitations of being brittle with respect to agent or message delivery failure; swarm agents are more appealing in this regard.

The characteristics of a swarm are that there is no central controller or data source and that global state is not maintained. There is no explicit model of the environment. Perception is local only and swarm agents (robots in the above definition) are reactive,

1. Nevertheless, a blackboard is often used for inter agent communication which creates a centralized architecture.

belonging to the class of agents favoured by Brooks. Swarm agents are situated; i.e., they sense their environment directly and do not manipulate symbolic representations of the environment. Again, in this regard, they are similiar to competence modules seen in a Brooks subsumption-style robot. Swarm agents do not possess high level goals, generally no single agent within the swarm knows what problem is to be solved. Instead, each agent follows a sequence of simple actions determined by locally sensed input and it is the actions of many such agents that cause problems to be solved. Problem solving is *emergent* or, stated another way, swarms *self organize*.

The properties of a swarm system are such that they are generally robust to the failure of individual agents; i.e., it does not matter that any one agent may not complete its task. In fact, a "poorly-performing" agent will often "die" rather than contribute to the overall problem solving process which further improves the performance of the swarm system. Swarm systems also tend to react well to changing environments, environments that may even cause the extinction of large numbers of agents when dramatic or extreme changes occur. This observation will be revisited when we discuss agent management in chapter 6.

Swarm systems tend to rely on agent to agent interactions through the environment and not direct agent to agent communication as in many symbolic cooperative agent systems. Such systems are necessarily situated, as an agent, to determine its next state, bases its decision upon flows it senses from the environment and these flows depend upon the actions of other agents. No single agent can predict the effect of its actions on the environment because these will depend on the actions of other agents as well. Swarm

agents are not generally reliant upon the actions of any single agent to solve a particular problem. Rather, it is the reinforcement of the actions of one agent by another that naturally causes a solution to emerge. Another important observation about swarm agents is that they move *through* the environment, their migration guided by gradients of flows within that environment. This coupling with the environment makes swarm systems a natural candidate as a model for autonomous mobile agents. Intelligent behavior frequently arises through indirect, environment mediated communication between the agents, this being the principle of stigmergy [Grassé 59].

Crucial, then, to the development of a useful pragmatic theory of swarm systems is the observation that the swarm cannot be considered independently of the environment, modelling of the environment must be included if a swarm is to be correctly engineered. Our view, and one that forms the basis of the architecture described in the next chapter, is that the coupling of agent and environment through time-based dynamical processes such as diffusion and chemical activity allow intelligence to arise in swarm systems. This view is not unique. Recently, Port and vanGelder [van Gelder 95] have articulated the idea that cognition is a time-based dynamical system. In fact, Port and vanGelder propose an alternative to the Physical Symbol Hypothesis [Newell 80] which they call the Dynamical System Hypothesis, the spirit of which runs through all chapters of this thesis.

In order to understand swarm systems, three things need to be analyzed: the agent, the environment and the coupling between the two. We will do this with a series of examples drawn from the world of insect and animal behaviours. The importance of these examples

within the context of this thesis is that they clearly demonstrate that simple agents interacting locally through the environment can generate complex, emergent problem solving behaviour.

2.3.1 Ant Behaviors

Individual ants are behaviorally simple insects with limited memory and exhibiting activity that has a stochastic component. However, collectively ants manage to perform several complicated tasks with a high degree of consistency. Examples of sophisticated, collective problem solving behavior have been documented [Franks 89], [Hölldobler 94] including:

- Forming bridges
- Nest building and maintenance
- Cooperating in carrying large items
- Finding the shortest routes from the nest to a food source
- Regulating nest temperature within a one degree Celsius range
- Preferentially exploiting the richest source of food available.

In the examples listed above, two forms of stigmergy have been observed. Sematectonic stigmergy involves a change in the physical characteristics of the environment. Ant nest building is an example of this form of communication in that an ant observes a structure developing and adds its ball of mud to the top of it. The second form of stigmergy is sign-based. Here something is deposited in the environment that makes no direct contribution to the task being undertaken but is used to influence the subsequent behavior that is task related.

Sign-based stigmergy is highly developed in ants. Ants use highly volatile chemicals

called pheromones (a hormone) to provide a sophisticated signaling system. Ants foraging for food lay down quantities of pheromone marking the path that it follows with a trail of the substance. An isolated ant moves essentially at random but an ant encountering a previously laid trail will detect it and decide to follow it with a high probability and thereby reinforce it with a further quantity of pheromone. The collective behavior which emerges is a form of autocatalytic behavior where the more the ants follow the trail the more likely they are to do so. The process is characterized by a positive feedback loop, where the probability that an ant chooses any given path increases with the number of ants choosing the path at previous times.

Essentially, then, ants following simple (simulated) rules explains the emergence of complex problem solving behaviour.

2.3.2 Ant foraging

Ants are highly successful foraging insects. Societies of ants construct networks of paths that connect their nests with sources of food available within their local environment. When described in terms of a data structure, these networks form minimum spanning trees [Goss 90] and by traversing these paths ants minimize the energy required to bring the food back to the nest. While graph theory has a number of defined algorithms for the construction of minimum spanning trees, they are certainly not used by ants! Rather, this globally optimal data structure arises, or emerges, as a consequence of the simple actions of the individual ants.

The ants' actions are straightforward and consist of five rules [Steels 95]. These are:

1. Avoid obstacles. Ants move around obstacles rather than crash into them.
2. Wander randomly preferring to move in the direction of increasing pheromone density. Sensing pheromones causes an ant to choose the direction of its next step from a distribution which is weighted in favour of the direction of the scent. If no pheromones are detected, execute Brownian motion, choosing each step from a uniform distribution over all possible directions.
3. If the ant finds itself at a food source and is not currently holding any, pick up the food.
4. If the ant finds itself at the nest and is carrying food, drop the food.
5. While holding food, an ant drops pheromone at a constant rate as it walks. The movement strategy continues in rule 2, or movement can be random. Both strategies cause the same emergent behaviour but using the existing pheromone trails works more quickly.

The emergent behaviour -- that of path planning -- arises as a consequence of the Brownian motion of the ants which guarantees that ants can visit all points in a given region. Assuming that the separation of nest and food source are small relative to the range of the ant, ants will find the food and, importantly, the nest. Ants that wander a long distance from the nest and find no food will typically die or, at the very least, never reach the nest again. As such, they contribute nothing to the path planning activity. As food carrying ants are the only ants which drop pheromone, and only when carrying food, all pheromone trails lead to a source of food. Paths to depleted food sources will disappear because pheromones evaporate and diffuse within the environment. Similarly, food

carrying ants that do not return to the nest will never complete a trail. Finally, paths back to the nest will easily be detected by outgoing ants and as these inevitably lead to food, they will be reinforced by those ants once they have picked up food.

While the initial path will not be straight, ants still exhibit a random element in their movement and this wandering will generate short cuts across the initial random walk. Diffusion of the pheromone in the environment causes merging of these short cuts into a straighter path the more it is used. Over time, and the actions of many ants, a minimum spanning tree emerges.

It is evident from the five rules shown above that the ant neither has the goal of computing the shortest path to any individual food source nor does it intend to merge these paths to various food sources in order to form a minimum spanning tree. The behaviour is clearly emergent.

2.3.3 Ant Brood Sorting

Sorting algorithms have been investigated by computer scientists for many years and the induction of programs to perform sorting tasks is still considered a hard problem by the Machine Learning community. How then can an ant colony sort the many items such as eggs, food and larvae without such sophisticated machinery?

Individual ants act according to four very simple rules in order to solve the sorting task [Deneubourg 90]. These are:

1. Wander randomly around the nest (as for rule 1 in the foraging example).
2. Each ant has a small memory of the last several steps. An ant senses nearby objects.

3. When an object is encountered, a decision is made whether to pick up the object based upon whether the ant has seen similiar objects recently (i.e., currently in memory). The probability with which an ant decides to pick up an object is:

$$p_{pickup} = \left(\frac{k_{pickup}}{k_{pickup} + f} \right)^2$$

where f is the fraction of memory that is occupied by objects of the same type as the sensed object and k_{pickup} is a constant. As can be seen, the smaller the value of f, the more likely the ant is to pick up the object.

4. If an ant is carrying something, it decides stochastically whether to drop it or not on each time step. The probability of dropping the carried object increases as the ant ceases to sense objects of that type in the local environment. The probability with which an ant decides to drop an object is given by:

$$p_{drop} = \left(\frac{k_{drop}}{k_{drop} + f} \right)^2$$

where k_{drop} is a constant.

Sorting emerges in this system as a result of the following. As in path planning, Brownian motion allows an ant swarm to explore all regions of the nest. While objects may be distributed randomly initially, the resulting arrangement of objects will not be uniform. These density fluctuations stimulate ants to drop other similiar items. Consequently, the fluctuations in density amplify and new objects are attracted by the

existence of a high concentration of like objects. The stochastic nature of the pick up and drop behaviors facilitates the merging of multiple concentrations as, periodically, ants pick up an object from one high concentration and move it to another region.

Again it is evident from the four rules shown above that an ant does not have sorting as a rational goal. The behaviour is clearly emergent.

2.3.4 Wolves surrounding prey

Wolves, it is known, are capable of preying upon much larger animals by hunting in a pack and surrounding their prey. The so-called predator-prey problem has been well-studied in the field of Distributed Artificial Intelligence (DAI) [Korf 90]. Frequently, the solutions proposed assume reasoning and negotiation (communication) capabilities in the predator that are unproven and even implausible. Wolves, for example, do not have long range communication capabilities.

However, simpler solutions involving swarm-like behaviour are possible. Again, a small number of rules govern the behaviour. These are:

1. For the prey, move to the point that is farthest way from the nearest wolf. As long as its rate of movement is faster than that of a wolf, escape is possible.
2. For wolves, move to the point in the region with the highest score given by:

$$S = d(prey) - k*d(wolf)$$

where k is a constant representing a repulsive force between wolves, *d(prey)* is the

distance to the prey and *d(wolf)* is the distance to the nearest other wolf.

Each individual in this system influences and is influenced by the entire system. The global behaviour of the system depends crucially on the relative speeds of wolves and prey and the value of the repulsive constant, *k*. When balanced, the wolves will, without fail, surround their prey without recourse to sophisticated communication or negotiation strategies. Here, it can be argued, the wolves have the goal of capturing their prey but they certainly do not have the ability to coordinate a sophisticated pack-wide strategy necessitating global communication.

2.3.5 Flocking behaviour

Schools of fish display remarkably coordinated behaviour as do flocks of birds. Large numbers of them stay together, apparently operating as a single entity. Both fish and birds coordinate turns, avoiding collisions with each other and obstacles in their path. Interestingly, no central controller is involved which contrasts strongly with the type of control mechanism employed for similiar many body problems such as are found in air traffic control systems[1]. The key difference, as with the previous three examples, is distribution of activity and the reinforcement of one agents actions on another.

Birds or fish follow three simple rules [Reynolds 87]. These are:

1. Maintain a minimum separation from the nearest object (bird or obstacle).
2. Match velocity to nearby birds.

1. Fish schools may contain thousands of individuals. Air traffic control systems certainly do not seem to scale to such a size.

3. Stay close to the centre of the flock.

The flock, then, is a self sustaining structure. It maintains itself purely through local sensory input and activity. Each entity's actions simultaneously respond to and change the overall structure of the flock while no individual has the goal of flock maintenance. Again, the effects of local action propagate throughout the system causing global coordination.

2.3.6 The Wave

A recent phenomenon at sporting events, the wave represents a remarkable example of global coordination through local interaction in the area of human social behaviour. During the wave phenomenon, one person raises his or her arms and, within seconds, the whole stadium responds in a wave that travels rhythmically around the stands. The wave appears orchestrated but this is in fact not so. The rules governing the behaviour are simple, and rely on limited communication. They are:

1. If the individual to your left has his or her arms raised, raise yours.
2. If the individual k positions to your right has his or her arms raised, lower your arms, if raised.

These two rules are sufficient to maintain a wave of width k indefinitely in a circular structure such as a stadium. Only one individual need raise his or her arms for the wave to be started. The reader might argue that some global communication has occurred in that we all understand the "arms up" cue. However, imagine adding an alien to the crowd who is given only the above rules to guide his or her actions, the wave would still emerge.

2.3.7 Simulation of Swarms: StarLogo

StarLogo is an evolution of the Logo language. Logo was designed as a programming language for children, allowing them to create simple graphics on a computer screen. StarLogo [Resnick 94] can be thought of as a simulation environment designed to stimulate understanding about decentralized systems. StarLogo draws for its inspiration on Cellular Automata. StarLogo is relevant to this thesis in that it represents an environment in which massively parallel computation can be performed for the purpose of exploring self organizing phenomena.

In StarLogo, the intended focus of the system is the investigation of how colony-level behaviours can arise from interactions between individual creatures and their environment. There are two types of entity in the StarLogo world: creatures (turtles) and patches. Associated with the creatures that inhabit the world are behaviours that are encapsulated inside of individual threads of control called daemons. Behaviours belong to the environment and not, as is the case in object oriented systems, to the class of the object or derived from another instance[1]. Patches are discrete regions of the environment with their own state information; e.g., colour, concentration of a scent and the creatures currently located there. Patches may have behaviour, and in this regard are similiar to creatures. The behaviour of a patch can be thought of as the physics of the patch and, as all patches share the same behaviour, as the physics of the universe. While the word scent is used by Resnick -- a word that naturally evokes an image of a chemical being modelled within the system -- the chemical analogy is limited in that creatures do not participate in

1. As in the Self language, for example.

chemical reactions; they merely sense chemicals. This limitation, and the value of a chemical abstraction in computing generally, provides partial motivation for the architecture presented in the next chapter.

Creatures are endowed with sensory apparatus and can sense each other and the contents of a patch. Using sensory input they can orient themselves in the direction of the highest concentration of a given scent thereby moving along a gradient of chemical activity. Arguably, these characteristics are the most important creature capabilities and enable self organization within the StarLogo simulation system. While breeds of creature are supported, creatures generally are limited in that they have no memory and behavior of all breeds is defined within the environment and not associated with the individual breed or "class". This latter characteristic of the system makes it completely unsuitable for implementation in a real network, for example.

2.4 Engineering Swarm Intelligence

This section brings together information regarding the main sources of research in Swarm Intelligence. The most complete set of work has been done by Marco Dorigo, and this has influenced our work considerably.

Swarm intelligence research originates from the work on the emergence of collective behaviours of real ants. Ants have a small amount of cognitive capability, limited individual capabilities, and react instinctively in a probabilistic way to their perception of their immediate environment. They can, for example, find the shortest path between the nest and a food source by laying down on their way back from the food source a trail of an

attracting substance, called pheromone. Ants wandering randomly around the nest can then be attracted by this trail and this will rapidly lead them to the food source. By laying down a pheromone trail of different density depending on the quality of the food source the have found, the colony becomes able to discriminate between food sources of different kinds and qualities.

2.4.1 The Ant System

The Ant System (AS) is a general-purpose heuristic algorithm, which can be used to solve diverse combinatorial optimization problems. This work has been lead by Marco Dorigo, now at the University of Free Brussels, Belgium. For an in-depth technical description of the algorithm, the reader is referred to [Dorigo 91] and, more recently, [Dorigo 96].

The idea of using swarm intelligence as a new computational paradigm for solving engineering problems is quite recent. Marco Dorigo and colleagues were the first ones to propose adapting these ideas to the Travelling Salesman Problem [Dorigo 91]. Many other applications since then have been proposed: graph partitioning [Kunz 94], data clustering [Faieta 94], job shop scheduling [Color 94], robotics [Beni 90], vehicle routing [Bullnheimer 97], graph colouring [Costa 97], the quadratic assignment problem [Maniezzo 94], [Taillard 97]. See [Bonabeau 94] for an overview. Variations on the basic AS have also been proposed. Local search operations have been proposed [Stützle 97], as has the integration of reinforcement learning [Leer 95], or Q-learning [Gambardella 95].

The advantages of swarm intelligence are twofold. Firstly, it offers intrinsically

distributed algorithms that can use parallel computation quite easily. Secondly, these algorithms show a high level of robustness to change by allowing the solution to dynamically adapt itself to global changes by letting the agents (the ants) self-adapt to the associated local changes.

2.4.1.1 Motivations for the Ant System

The Ant System (AS) has several desirable characteristics. It is versatile, in that it can be applied to similar versions of the same problem. For example, there is a straightforward extension from the travelling salesman problem (TSP) to the asymmetric travelling salesman problem (ATSP). It is robust and general purpose. It is a population-based heuristic. As such, it allows the exploitation of positive feedback as a search mechanism, as described in a later section. Consequently, it makes the system amenable to parallel implementations.

These desirable properties are mitigated by the fact that, for some applications, the Ant System can be outperformed by more specialized, domain specific algorithms providing empirical support for the No Free Lunch theorem [Wolpert 99]. Many heuristics share this problem, examples being the much-researched problem solving techniques of simulated annealing (SA), and tabu search (TS). Nevertheless, as is the case with SA and TS, the AS represents a heuristic that can be applied to problems which are similar to a classical problem, such as TSP, but are sufficiently different as to make the application of a domain specific algorithm impossible. The ATSP is an example of such a problem.

The AS is an example of a distributed search technique. Search activities are distributed

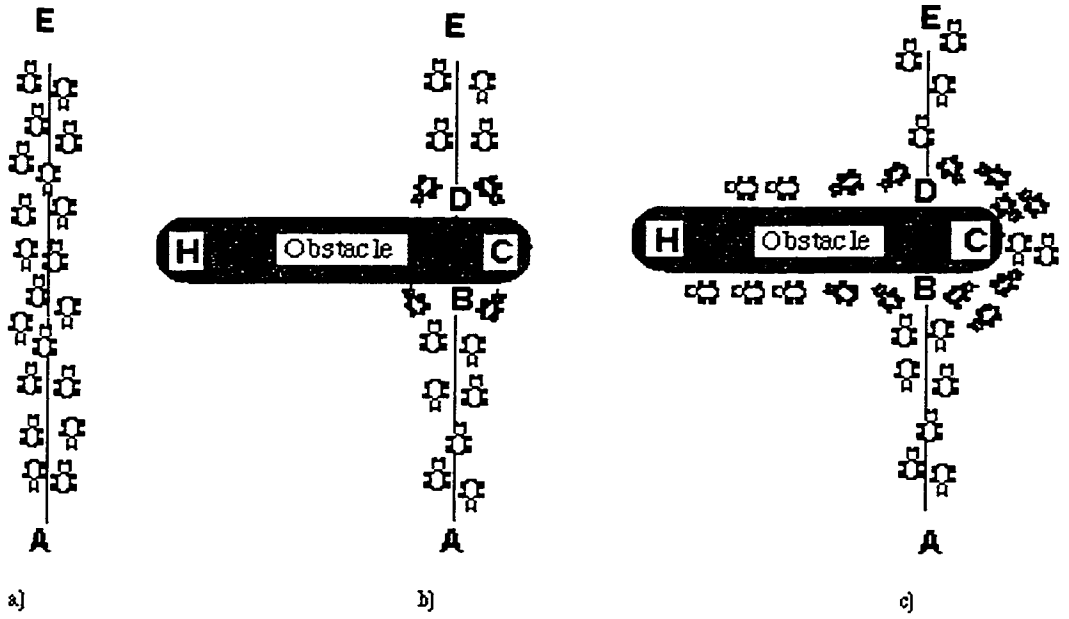
over ant-like agents, i.e., entities with very simple basic capabilities. These agents, in a metaphorical and highly stylized way, mimic the behaviour of real ants. In fact, research on the behaviour of real ants has greatly inspired the AS. The research inspiration for the AS arises from the work of ethologists, who attempted to understand how almost blind animals like ants could manage to establish shortest route paths from their colony to feeding sources and back.

The following paragraphs are a modified explanation of path-emergence as provided in [Dorigo 96].

> "Consider, for example, the scenario shown in Figure 15. Ants are walking along a path, for example from food source A to the nest E, and vice versa, see Figure 15a. Suddenly an obstacle appears and the path is cut off. Therefore, at position B the ants walking from A to E (or at position D those walking in the opposite direction) have to decide whether to turn right or left (Figure 15b). The choice is influenced by the intensity of the pheromone trails left by preceding ants. A higher level of pheromone on the right path gives an ant a stronger stimulus and thus a higher probability to turn right. The first ant that reaches point B (or D) has no preference for left or right branches of the path because there is no pheromone on either path. Because path BCD is shorter than BHD, the first ant following it will

reach D before the first ant following path BHD (Figure 15c).

The result is that an ant returning from E to D will find a stronger trail on path DCB, caused by the half of all the ants, by chance, deciding to approach the obstacle via DCBA and by the already arrived ones coming via BCD. Therefore, they will prefer path DCB to path DHB. Consequently, the number of ants following path BCD per unit of time will be greater than the number of ants following BHD. Hence, the quantity of pheromone on the shorter path will grow more quickly than on the longer



How ants find shortest paths
a) Ants follow a path between points A and E.
b) An obstacle is interposed; ants can choose to go around it following one of the two different paths with equal probability.
c) On the shorter path more pheromone is laid down.

FIGURE 15. **Shortest Path Emergence**

one. Therefore, the probability with which any single ant chooses a particular path is quickly biased towards the shorter one. The result is that, very quickly, all ants will choose the shorter path.

The algorithms that we are going to define in the following sections are models derived from the study of artificial ant colonies. Therefore, we call the system the Ant System (AS) and the algorithms we introduce, ant algorithms. As we are not interested in simulation of ant colonies, but in the use of artificial ant colonies as an optimization tool, our system will have some major differences with a real (natural) one:

1. Artificial ants have some memory.
2. They are not completely blind.
3. They live in an environment where time is discrete.

Nevertheless, we believe that the ant colony metaphor can be useful to explain our model. Consider the graph of Figure 16a, which is a possible AS interpretation of the situation of Figure 1b. In order to fix the AS ideas more concrete, suppose that the distances between D and H, between B and H, and between B and D—via C—are equal to unity. Let C be positioned half the way between D and B (see Figure 16a). Now let us consider what happens at regular discrete points in time: t= 0, 1, 2, ... and so on. Suppose that 30 new ants come to B from A, and 30 to D from E at each time unit, that each ant walks at a speed of one per time unit. Further, suppose that,

while walking, an ant lays down at time t a pheromone trail of intensity one, which, to make the example simpler, evaporates completely and instantaneously in the middle of the successive time interval (t+1, t+2).

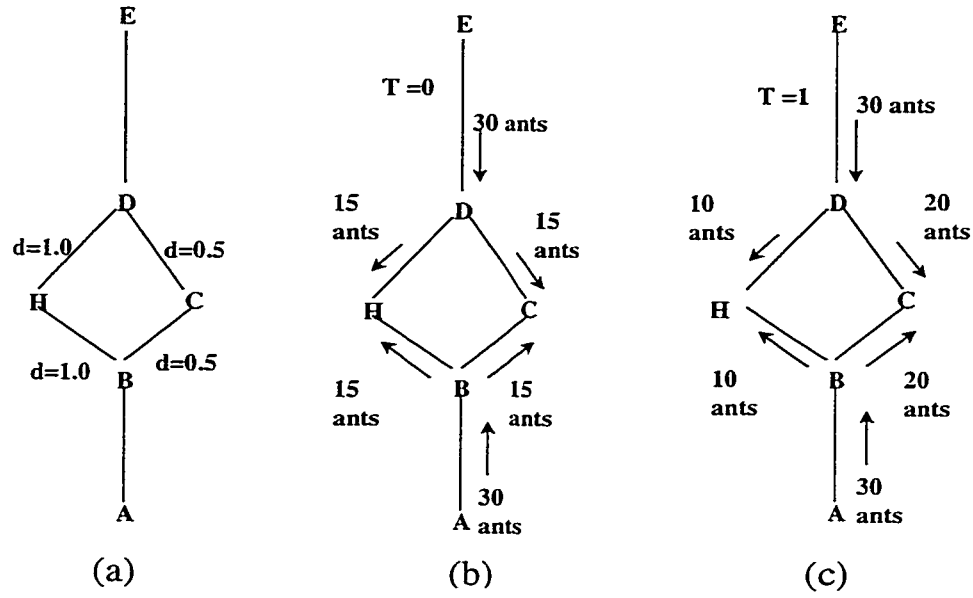


FIGURE 16. **Pheromone Trails**

At t=0 there is no trail yet, but 30 ants are in B and 30 in D. Their choice about which way to go is completely random. Therefore, on average, 15 ants from each node will go towards H and 15 towards C (Figure 16b).

At t=1 the 30 new ants that come to B from A find a trail of intensity 15 on the path that leads to H, laid by the 15 ants that went that way from B and a trail of intensity 30 on the path to C. This trail has developed as the sum of the trail laid by the 15 ants that went that way from B and by the 15 ants

that reached B coming from D via C (Figure 16c). The probability of choosing a path is now biased, so that the expected number of ants going toward C will be the double of those going toward H: 20 versus 10 respectively. The same is true for the new 30 ants in D that came from E.

This process continues until all of the ants will eventually choose the shortest path.

The idea is that, if at a given point an ant has to choose among different paths, those which were heavily chosen by preceding ants (that is, those with a high pheromone level) are chosen with higher probability. Furthermore, high pheromone levels are synonymous with short paths."

2.4.2 Applications of the Ant System

Most work on the Ant System has been applied to the Travelling Salesman Problem[1]. This is a classical problem and is often used for the assessment of heuristics as it is NP-Complete. Dorigo shows that the Ant System can be applied to the TSP, and other problems, which can be expressed in graph-partitioning terms. Much more detail on this work is presented in [Gambardella 95], [Dorigo 96].

Dorigo's conclusions on the results of the Ant System on the TSP are:

Within the range of parameter optimality, the algorithm always finds very good solutions for all of the tested problems.

1. This, however, is changing and several potential applications are being researched.

The algorithm quickly finds good solutions, while not exhibiting stagnation behaviour - the ants continue to search for new, possibly better tours.

With increasing problem size, the sensitivity of the parameter values to the problem dimension has been found acceptable[1].

The work on the TSP is directly transferable to the asymmetric TSP. This problem is significantly more difficult than the TSP. Typically the TSP can be solved for graphs with several thousand nodes, while the ATSP is only solved optimally in cases where there are several dozen nodes.

Application to the ATSP required no modifications to the basic AS algorithm. The results of applying the algorithm to this problem were very close to (within 3.3%) the known optimal tour, and were found within acceptable time. In fact, the application of the AS to other problems have also required little or no modification to the basic algorithm.

2.4.3 Ant System Summary

The Ant System is a relatively new search methodology based on a distributed autocatalytic process that can be applied to the solution of classical optimization problems. The general idea underlying the AS search paradigm is that of a population of agents each guided by an autocatalytic process directed by a greedy force. If an agent were to search alone, the autocatalytic process and the greedy force would tend to make the agent converge to a sub-optimal solution with exponential speed. When agents interact it

1. It is our experience that an AS can be improved by allowing control parameters to self-adapt during the search process. This is described in a later section.

appears that the greedy force can give the right suggestions to the autocatalytic process and facilitate rapid convergence to very good, often optimal, solutions without getting stuck in local optima. We speculate that this behaviour arises because information gained by agents during the search process is used to modify the problem representation. In some sense, the region of the space considered by the search process is reduced. Even if no tour is completely excluded, bad tours become highly improbable, and the agents search only in the neighbourhood of good solutions.

The main contributions of the Ant System are the following:

- Positive feedback is employed as a search and optimization tool. The idea is that, if at a given point an agent (ant) has to choose between different options, and the one actually chosen proves to be good, then in the future that choice will appear more desirable than it was before.
- Synergy can arise and be useful in distributed systems. In AS, the effectiveness of the search carried out by a given number of cooperative ants is greater than that of the search carried out by the same number of ants, each one acting independently from the others.
- The Ant System can be applied to diverse combinatorial optimization problems with a graphical representation.

As already pointed out, the research on behaviour of social animals is to be considered as a source of inspiration and as a useful metaphor to explain our ideas. We believe that, especially if we are interested in designing inherently parallel algorithms, observation of natural systems can be an invaluable source of inspiration. Neural networks, genetic algorithms, evolution strategies, immune networks, simulated annealing are only some examples of models with a "natural flavour". The main characteristics, which are at least partially shared by members of this class of algorithms, are the use of a natural metaphor, inherent parallelism, a stochastic nature, self-adaptation, and the use of positive feedback.

The algorithms introduced later in this thesis can be considered as new members of this class. All this work in "natural optimization" fits within the more general research area of stochastic optimization, in which the quest for optimality is traded away for computational efficiency.

2.5 Autopoiesis

While the previous two sections have provided support for the value of studying naturally-occurring systems and using them for designing multi-agent systems, they provide no insight into systems that are truely autonomous. Other concepts are required in order to explain autonomy.

Autopoiesis, derived from the words auto (meaning self) and poiesis (meaning creation or production), is attributed to Maturana and refined along with his student Varela. Formally, autopoiesis can be defined as:

> "An autopoietic system is organized (defined as a unity) as a network of processes of production (transformation and destruction) of components that produces the components that:
>
> 1. through their interactions and transformations continuously regenerate and realize the network of processes (relations) that produced them; and
> 2. constitute it (the machine) as a concrete unity in the space in which they [the components] exist by specifying the topological domain of its realization as such a network." ([Varela 80], page 79).

While the first part of the definition appears straightforward, the second is certainly not. The language is obscure, possibly owing to the authors' training -- philosophy. The definition, we believe, requires some explanation.

An autopoietic machine exists in space and time and is capable of self creation and maintenance. Such machines specify their own organization and will attempt to maintain that organization over time. It is important to note that autopoietic systems maintain their structure and their organization; i.e., the relationships between structural elements. While not explicitly stated in the definition, it can be inferred from [Varela 80] that the network of interactions imply local information processing, a property which we find attractive and important to the ideas documented here. The definition refers to a network, or graph, of interactions, which could easily be thought of as a set of chemical reactions, for example. Using this as an example, the second part of the definition restated becomes the time and space evolution of the set of chemical reactions describing the system. In other words, the machine exists as a consequence of "running" the program comprised of the processes mentioned in the first point of the definition.

Living systems are autopoietic (as observed by Maturana and Varela) and it is this constitution from which autonomy naturally arises. Autonomy, being more general than autopoiesis, can be defined according to Varela, as:

> "...defined as a composite unity by a network of interactions of components that (i) through their interactions recursively regenerate the network of interactions that produced them, and (ii) realize the network as a unity in

> the space in which the components exist by constituting and specifying the unity's boundaries as a cleavage from the background..."

Again, the language is somewhat obscure but says essentially the following. Autonomy implies a clear separation of the autopoietic system from its environment: think of a cell as an example. The rules of regeneration are stored inside of the boundary between the machine and its environment. From our perspective, the most important word in the above quote is the word *recursive*; i.e., mutual dependencies between structural elements have to be present in order for self-generation to occur. Here, think of an organ as an example: organs regenerate by creation of new cells. Alternatively, think of biochemical pathways in the human body. With input of energy, these pathways regenerate cells and maintain the body's organization. This will be more formally presented when chemical abstractions are described in the next section. A key observation with regard to autopoietic systems is that they are self-maintaining, all other functions are considered less important than the (re-) generation of self. This is a very attractive property when we consider the use of autopoietic principles for control, management and reconfiguration of a network, for example. Varela says:

> "... I see autopoiesis as one possible form of autonomy (or organizational closure, as defined later), and that this term should be restricted to systems, whether natural or artificial, that can be characterized by a network that is, or resembles very closely, a chemical network." ([Varela 80], page 15).

In this quote we have inspiration for the architecture described in the next chapter and a

reason for investigating chemical metaphors as described in the next section. Varela, in the above quote, brings together the ideas that self maintaining systems can be represented chemically, both in terms of state and time evolution of that state, and that chemical reactions enable complex computation.

The importance of chemical reactions needs to be noted here as it implies that chemicals are defined (which are symbols, a point revisited in the next section and following chapter) and a concentration of that chemical exists within the environment. Reactions imply chemical concentration changes, this being the sensory mechanism by which mutual awareness and communication in autopoietic systems occurs. Stated another way, language or communication arises by *attention*; i.e., perception of changes, not absolute values in the environment.

Self organization as order through fluctuations is well known, especially through the work of Prigogine and Nicolis [Prigogine 77] and Eigen [Eigen 79], all of whom consider simple chemical systems in their exploration of self organizing systems. The value of chemistry -- chemicals and chemical reactions -- seems, once again, to present itself.

2.6 Chemical Abstractions

The previous section, while largely philosophical in outlook and content, nevertheless provides support for the value of chemical abstractions. This section explores previous related work that exploits the chemical metaphor. The work presented in this section can be classified along the lines of the completeness of usage of the metaphor. For example, the Chemical Abstract Machine[1] (CHAM) of Berry and Boudol [Berry 92], and

Algorithmic Chemistry of Fontana [Fontana 91], deal with chemicals purely as symbols and reactions as rewriting systems. Prigogine and Eigen, in contrast, are interested in chemical reactions and their properties as they relate to self-organizing systems. In their work, both the chemical and its concentration are studied and self-organization is viewed as patterns of concentrations of particular chemicals; e.g., the Brusselator scheme [Brusselator].

2.6.1 The Chemical Abstract Machine

The Chemical Abstract Machine, a development of the Gamma formalism [Benatre 88], was introduced in order to increase the abstraction with which programs are specified, removing as far as possible any sequential bias in the resulting design.

The Gamma formalism is intended to capture the intuition of computation as the global evolution of a collection of atomic values interacting without constraint. Gamma is, then, a kernel language which relies upon the chemical metaphor. The unique data structure in Gamma is the multiset which can be seen as a chemical solution. A simple program is a pair *(reaction condition, action)*. Execution proceeds by replacing the multiset elements satisfying the reaction condition by the products of the reaction. The result is obtained when a stable state is reached, that is when no further reactions can take place. Gamma is declarative[1] in nature and is designed to express the "idea" of an algorithm rather than its

1. Derived from the Gamma formalism.

1. The Prolog programming language is an example of a declarative language which uses first order logic. However, it fails to qualify as a Gamma language as a result of the serial nature of its clausal evaluation; i.e., clause order is important. If clauses were evaluated without regard to order and in parallel, G-Prolog would provide many of the facilities intended for Gamma. Logic programming implementations of Gamma have been proposed; e.g., Gamma-log.

detailed implementation. It is intended that there should be no hidden control in Gamma. A simple implementation of Gamma would be as shown below:

$$x_1 \ldots, x_n \rightarrow f(x_1 \ldots, x_n) \Leftarrow R(x_1 \ldots, x_n)$$

where $R(x_1, \ldots x_n)$ is the reaction on the multiset $(x_1, \ldots x_n)$, and $f(x_1, \ldots x_n)$ the products

While tuples remain to be processed

do

choose a tuple $(x_1, \ldots, x_n)$ not yet processed;

if $R(x_1, \ldots, x_n)$ then

remove $(x_1, \ldots, x_n)$ from M

replace them by $f(x_1, \ldots, x_n)$

end

of the reaction.

CHAM extends the Gamma formalism with notions of a *membrane* and an *airlock* mechanism. CHAM was originally proposed by Berry and Boudol to describe the operational semantics of process calculi. Membranes are used to encapsulate solutions and to force reactions to occur locally. A membrane can be used to introduce a multiset of molecules to its containing environment. Stated another way, a solution can be transformed into a single molecule. The airlock mechanism is used to describe communications between an encapsulated solution and its environment. The main role of the airlock is to allow a molecule to be visible from outside the membrane and thus take part in a reaction in the embedding solution. The need for membranes and airlocks arose

from the description of Communicating Concurrent Systems (CCS) in CHAM. This added machinery significantly increases the ability of the formalism to deal with large systems as it provides for *scoping* rules.

2.6.2 Other Ideas from Chemical Computation

The attractions of chemical computation have been observed long before the writing of this thesis. Work on Information Chemistry by Winter [Winter 97] of British Telecom indicates the potential for industrial exploitation of speculative ideas of this kind while Fontana's work on Algorithmic Chemistry [Fontana 91] attempts to span logic and the formalized computation described above in a way that formalizes chemical evolution. Work on self organizing algorithms derived from RNA interactions [Banzaf 95] also demonstrates a keen interest in chemical computation.

2.7 Adaptation and Learning

Many techniques exist for adaptation and learning. The two sections which follow represent the techniques that have been employed in this thesis in order to improve the performance of agents in their chosen problem domain using feedback from the results of employing particular actions in the environment. The first section is doubly relevant to this thesis in that it deals with a schema concept -- or set of solutions -- a concept referred to extensively in the section on chemistry in the next chapter.

2.7.1 Genetic Algorithms

Genetic algorithms (GAs) [Holland 75] were originally proposed as a search technique for use in optimization problems. Genetic algorithms are stochastic search algorithms

whose search methods model aspects of Darwinian evolution. As stated in [Davis 87]:

> "... the metaphor underlying genetic algorithms is that of natural evolution. In evolution, the problem each species faces is one of searching for beneficial adaptations to a complicated and changing environment. The 'knowledge' that each species has gained is embodied in the makeup of the chromosomes of its members."

The basic principle behind genetic algorithms is to do what Nature does. The fitter the individual, the more likely it is to survive to be able to reproduce and thereby ensure that its genes survive to the next generation. While this section provides a brief introduction to genetic algorithms, more complete descriptions can be found in [Goldberg 89] and [White 93].

Genetic algorithms use a vocabulary drawn from natural genetics. We talk about *individuals* (or *genotypes*) in a population. These individuals are sometimes also referred to as *chromosomes* or *strings*. In genetic algorithms, a population consists of a set of single chromosomes. Chromosomes are made up of units called *genes* or *features* most often arranged in a linear formation or string. A feature is just a set of genes. Each gene controls the inheritance of one or more traits. The genes are said to express a given feature. Genes of certain features are located at specific locations on the chromosome - these are called *loci* or string positions. A feature which can have several values means that the associated genes can take on many values. The values which a gene can take are called *alleles*.

An evolution process run on a population of such chromosomes corresponds to a search through a space of potential solutions. The search process requires balancing two conflicting objectives: exploiting the best solutions at any point during the process and

exploring the search space. Hill climbing is an example of a strategy which retains as much of the best solution to date as possible when moving to the next solution - only small perturbations are made and the results retained only if improvement is seen. Consequently, it neglects the exploration of the search space and only guarantees to obtain an optimal solution if the solution space contains a single peak. Random search, on the other hand, explores the solution space without regard to the structure of the best solution to date nor to the region of the space where it was obtained. Genetic algorithms combine both solution exploitation and exploration of the search space in an extremely effective way.

Genetic algorithms have been used in a diverse set of optimization problems such as wire routing, travelling salesman problems, network link design [Coombs 87], [Davis 87a], [White 99a] adaptive control and many others.

Genetic algorithms differ from more convectional search algorithms in that they work with a population of chromosomes (solutions) rather than a single chromosome (solution). Genetic algorithms also work indirectly, in that problem parameters are encoded before being manipulated by genetic operators. Finally, genetic algorithms differ from most conventional search strategies in that they use probabilistic transition rules, not deterministic rules. As such they belong to the class of stochastic search algorithms.

Restricting the discussion to chromosomes represented as bit strings, each chromosome is the same length and each gene can take values drawn from {0,1} - the so called binary alphabet. In this case, the chromosome is represented by a bit string and for a chromosome of length l, 2^l states can be represented.

Genetic algorithms are often characterized as algorithms that are better than hill climbing algorithms because they manipulate combinations of bit strings called schemata. If we consider the binary alphabet for the moment, the alphabet of associated schemata would be {0,1,#}, where the # symbol indicates unknown. Figure 17 shows a schema and two possible strings which are examples of it. A schema represents a class of strings.

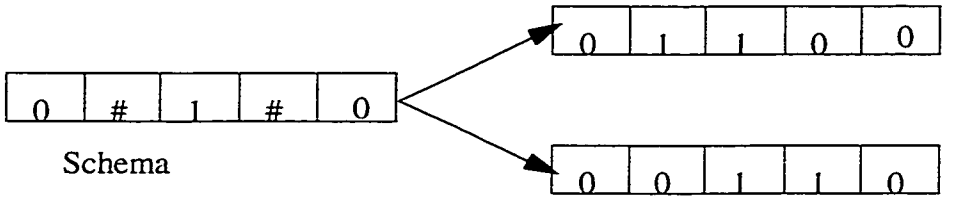


FIGURE 17. **Examples of schema**

The genetic algorithm can best be described by the program below:

```
Program GeneticAlgorithm()
Begin
   initializePopulation();
   evaluatePopulationFitness();
   while notFinished() do
   Begin
      for i:= 0 to populationSize by 2 do
      Begin
         offspring1 := randomRouletteWheel();
         offspring2 := randomRouletteWheel();
         if random() < crossoverProbability then
            newOffspring := crossover(offspring1, offspring2);
         newPopulation[i] := mutate(newOffspring[0]);
         newPopulation[i+1] := mutate(newOffsptring[1]);
      End
      population := newPopulation;
      evaluatePopulationFitness();
   End
End
```

The significant procedural elements of the program are described in the following

sections.

2.7.1.1 initializePopulation

The initializePopulation procedure is quite simple; we create a population of chromosomes where each chromosome is a binary vector of *l* randomly-generated bits.

2.7.1.2 evaluatePopulationFitness

The evaluatePopulationFitness procedure uses the fitness function referred to above in order to compute the fitness of each chromosome in the population.

2.7.1.3 randomRouletteWheel

The randomRouletteWheel function is used to choose which chromosomes from one generation should be reproduced in the next one. This is done by calculating an imaginary roulette wheel where each chromosome has a probability of selection calculated from:

$$p(i) = \frac{f(i)}{f_{avg}}, f_{avg} = \sum_{i=0}^{populationSize} f(i)/populationSize$$

This is called fitness proportional selection. Viewed as a roulette wheel, for a population of five chromosomes, Figure 18 might be visualized:

In order to generate the new population, the imaginary roulette wheel is spun a total of *populationSize* times and each time wherever it stops, that chromosome is reproduced in the new population. Referring to Figure 18, chromosome 5 should be reproduced in the next generation. In fact, chromosome 5 occupies approximately 40% of the roulette wheel and we would expect in the neighborhood of that percentage of the new population to be

chromosome 5. Obviously, several chromosomes will appear multiple times in the new population and it is just this mechanism - survival of the fittest - that causes highly fit members of the population to dominate subsequent generations.

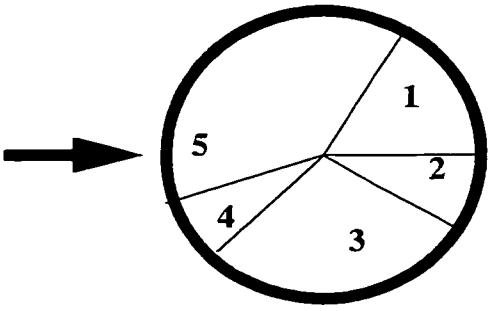


FIGURE 18. **Population Roulette Wheel**

2.7.1.4 crossover

The crossover procedure takes two chromosomes as input and generates two new chromosomes as output. The crossover operation occurs by selecting a position between 1 and l-1 in the first string. This identifies two substrings A_1 and A_2, where A_1A_2 is the same as the original string A. The same position is used for string B, generating the substrings B_1 and B_2. The crossover procedure returns the strings A_1B_2 and B_1A_2. Graphically, this procedure is shown in Figure 19.

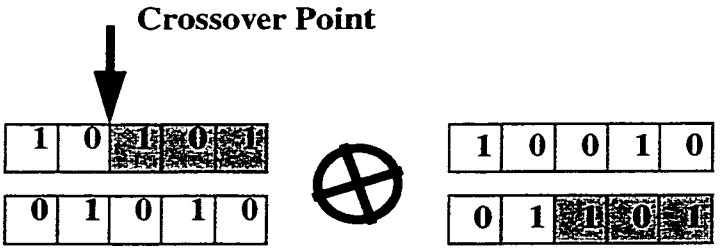


FIGURE 19. **Crossover in action**

As can be seen from the above example, completely different chromosomes can be

returned as a result of the action of this procedure.

2.7.1.5 mutate

The mutate procedure allows modified alleles to be introduced to the new population. The mutate procedure takes a chromosome and *for each bit* executes the following:

```
if random() < mutationProbability then
   toggleBit();
```

The mutation operation ensures that the population maintains some diversity as we proceed from generation to generation.

2.7.2 Reinforcement Learning

In models that employ reinforcement learning, such as the fault localization agent uses in chapter 5, an agent is connected to its environment via sensors and effectors as shown in Figure 20 below. During each interaction step, an agent has a knowledge of its state, *s*, receives input, *i*, performs some action, *a*, and receives some reward, *r*. The action performed by the agent changes its environment which leads to the scalar reward signal, *r*.

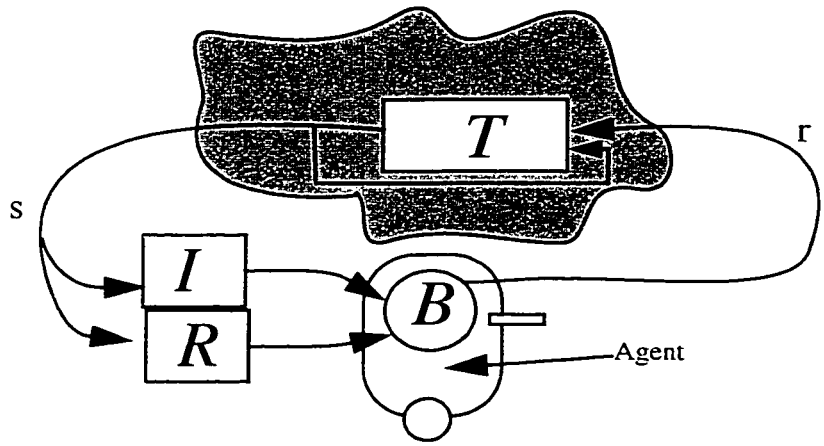


FIGURE 20. **Reinforcement Learning Agent**

The reward signal is also referred to as the reinforcement signal. The goal of reinforcement learning is to adapt the agent's behavior, B, such that it chooses actions that tend to increase the long-run sum of values of the reinforcement signal.

The environment consists of a set of states, *S*, that may or may not be completely observable. The agent has available to it a set of actions, *A*. There is also a set of reinforcement signals; either reals or, more typically, the set {0,1}.

Reinforcement learning is a form of unsupervised learning, where each interaction with the environment causes incremental learning. The agent is given an immediate reward and the new state. However, no information is provided about the *long-term* best action for the initial state. It is necessary for the agent to gather useful experience about the possible system states, actions, transitions and rewards actively in order to act optimally. Another difference from supervised learning is that on-line performance is important: the evaluation of the system is often concurrent with learning. The goal, then, for the agent is to learn the best policy to employ, a policy being a mapping of states to actions.

The problem that has to be solved by a reinforcement learning agent is that of temporal credit assignment, a problem common to many machine learning processes. How long should an agent wait before apportioning credit for a particular action. Many techniques have been proposed, two are described in the following sections.

2.7.2.1 Adaptive Heuristic Critic and TD(λ)

The adaptive heuristic critic is an adaptive version of policy iteration. In it, the value-function computation is no longer implemented by solving a set of linear equations, but is

instead computed by an algorithm called TD(0). The architecture for this approach is given in Figure 21. The architecture consists of two components: a critic (labeled AHC), and a reinforcement-learning component (labeled RL). The reinforcement-learning component can be an instance of any of the k-armed bandit algorithms, modified to deal with multiple states and non-stationary rewards. Instead of acting to maximize instantaneous reward, an agent acts to maximize the heuristic value, v, that is computed by the critic. The critic uses the real external reinforcement signal to learn to map states to their expected discounted values given that the policy being executed is the one currently instantiated in the RL component.

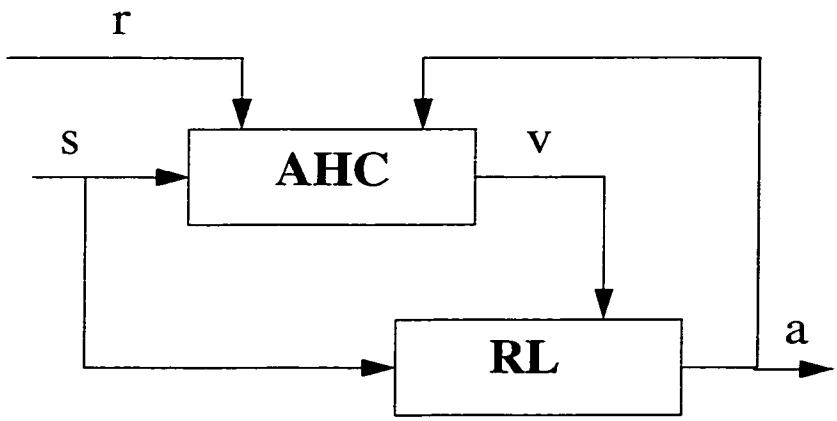


FIGURE 21. **Adaptive Heuristic Critic Architecture**

The correspondence with policy iteration can be seen if we consider the policy, π, implemented by the RL as being fixed with the critic learning the value function V_π for the policy. Once optimized, the critic is fixed and the RL allowed to learn a new policy π'. This is unrealistic, and most implementations allow simultaneous operation of AHC and RL components.

The value of a policy is determined through use of experiences; that is, a knowledge of the initial state (s), the final state (s'), the action taken to move from initial to final states (a) and the reward (r) received from the environment. Sutton's *TD(0)* [Sutton 88] rule is used to update state values:

$$V(s) \leftarrow V(s) + \alpha(r + \gamma V(s') - V(s))$$

Whenever a state s is visited, its estimated value is updated to be closer to $r + \gamma V(s')$, since r is the instantaneous reward received and $V(s')$ is the estimated value of the actually occurring next state. This is analogous to the sample-backup rule from value iteration--the only difference is that the sample is drawn from the real world rather than by simulating a known model. The key idea is that $r + \gamma V(s')$ is a sample of the value of $V(s)$, and it is more likely to be correct because it incorporates the real r. If the learning rate α is slowly decreased, and the policy is held fixed, *TD(0)* is guaranteed to converge to the optimal value function. The *TD(0)* rule is really an instance of a more general class of algorithms called *TD*(λ), with $\lambda = 0$. *TD(0)* looks only one step ahead when adjusting value estimates; although it will eventually arrive at the correct answer, it can take quite a while to do so. The general rule *TD*(λ) is similar to the *TD(0)* rule given above, but is applied to every

$$V(u) \leftarrow V(u) + \alpha(r + \gamma V(s') - V(s))e(u)$$

state according to its eligibility $e(u)$ rather than merely to the previous state. Many

eligibility trace functions have been proposed, one is shown below:

$$e(s) = \sum_{k=1}^{t} (\lambda\gamma)^{t-k}\delta_{s,\,s_k},\ where\ \delta_{s,\,s_k} = \begin{cases} 1, s = s_k \\ 0, otherwise \end{cases}$$

In this thesis, *TD(0)* is the temporal difference mechanism used, owing to its simplicity and the relatively small number of states in our problem state space.

2.7.2.2 Q-learning

The adaptive heuristic critic component of an on line learning system shown in Figure 21, can be implemented using Watkins' Q-learning algorithm [Watkins 89], [Watkins 92]. Q-learning algorithms are easy to implement and are often used in on-line controllers. Defining $Q^*(s, a)$ to be the expected discounted reinforcement of taking action a in state s, then continuing with choosing actions optimally. The Q learning rule is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma\, max_{a'}\, Q(s', a') - Q(s, a))$$

If each action is executed in each state an infinite number of times on an infinite run and α is decayed appropriately, the Q values will converge with probability 1 to Q^*. When the Q values are nearly converged to their optimal values, it is appropriate for the agent to act greedily, taking, in each situation, the action with the highest Q value. During learning, however, there is a difficult exploitation versus exploration trade-off to be made. There are no good, formally justified approaches to this problem in the general case; standard practice is to adopt ad hoc methods. For example, when reinforcement from the environment tends to zero, or falls below some threshold value, we act greedily, taking the

action with the highest Q value in a given state.

2.7.2.3 Discussion

AHC architectures seem to be more difficult to work with than Q-learning on a practical level. It can be hard to get the relative learning rates right in AHC so that the two components converge together. In addition, Q-learning is exploration insensitive: that is, that the Q values will converge to the optimal values, largely independent of how the agent behaves while the data is being collected. The implication is that, although the exploration-exploitation issue must be addressed in Q-learning (as in all learning methods), the details of the exploration strategy will not affect the convergence of the learning algorithm. For these reasons, Q-learning is the most popular and seems to be the most effective model-free algorithm for learning from delayed reinforcement. It does not, however, address any of the issues involved in generalizing over large state and/or action spaces. In addition, it may converge quite slowly to a good policy.

CHAPTER 3

Biologically inspired Architectures for Mobile Agents

3.1 Overview

The advantages of mobile agents have largely been discussed in terms of technology [Chess 97] and the value of individual agent autonomy [Bieszczad 98] as well as a novel problem solving technique [White 99c]. It is possible to view them as an approach to problem solving where mobility and interactions with the network locally are stressed. Similarly, coordination mechanisms for mobile agents have been discussed in terms of blackboard-style algorithms, with the agents tending to be rational, having a knowledge of self and a goal to be achieved [O'Hare 96]. In fact, several implementations of such systems are being investigated by the mobile agent community [Picco 98]. Symbolic systems of this type are often brittle, unable to cope with the failure of a single agent and may depend upon planning by a central agency in order to achieve coordination. Such systems often have to cope with the latency problems inherent in centralized systems. We believe that these limitations undermine the value and power of mobile agent systems.

It is difficult to argue against the effectiveness of many naturally occurring multi-agent systems and, in particular, systems exhibiting mobility. Societies of simple agents are capable of complex problem solving while possessing limited individual abilities [Franks 89], [Hölldobler 94]. They often possess no central coordination of activity; problem solving is distributed. Societies of such mobile agents are found at all levels of evolutionary complexity, from bacteria to ants and beyond. It is common in such societies to observe social coherence although when behavior of the individual is observed, a large stochastic component is present. Stated another way, such societies exhibit *emergent* behavior.

Problem solving by societies of simple agents has a number of common characteristics. Inter-agent communication is local; no single agent has a global view of the world. Communication is also achieved using simple signals and these signals are time dependent; e.g., they usually decay with time. Signal levels provide the driving force for migration patterns. Individual agents sense and contribute signal energy to the environment. In this description of the problem solving process, there are two distinct and important agent characteristics. First, there is the role of the agent within the problem solving process; i.e., how the work of problem solving is distributed to a *diverse* set of agents. Second, the degree to which the actions of one agent reinforce the actions of other agents in the society of problem solvers is significant.

The appeal of swarms of biologically inspired agents for industrial problem solving has recently been appreciated [Parunak 98]. Research into the problems and potential of

multiple, interacting swarms of mobile agents is just beginning [White 98d].

In the remainder of this chapter, we briefly describe the principles of Swarm Intelligence and Stigmergy. We then use these principles as motivation for the **Synth**etic **E**cology of **C**hemical **A**gents (SynthECA) and provide arguments as to the value of the abstraction. SynthECA is then used to indicate how several interacting swarms of agents would be capable of problem solving in networks.

This chapter introduces a new architectural description for an agent that is chemically inspired and proposes chemical interaction as the principal mechanism for inter-swarm communication. Agents within a given swarm have behavior that is inspired by the foraging activities of social insects, specifically ants, with each agent capable of simple actions and knowledge of a global goal is not assumed. The creation of chemical trails is proposed as the primary mechanism used in distributed problem solving arising from self-organization of swarms of agents. The chapter proposes that swarm chemistries can be engineered in order to apply the principal ideas of the Subsumption Architecture [Brooks 91] in the domain of mobile agents. The chapter outlines applications of the new architecture in the domain of communications networks and describes the essential elements of a mobile agent framework that is being considered for its implementation. Application details are presented in chapters 4, 5 and 6.

3.2 Emergent System: Principles and Mechanisms

In the previous chapter the concept of swarm intelligence and self organization of a large number of simple agents was introduced and a number of examples of such systems

described. Many other systems have been documented but this section provides an analysis of the common characteristics of these systems. These common characteristics are then used as motivation for the architecture proposed in this thesis.

In the following paragraphs we discuss principles that should drive a multi-agent architecture such that it should have the potential for exhibiting emergent behaviour. These properties or principles include:

- decentralization and aggregation
- small in size (or function)
- specialized
- behaviourally diverse
- information sharing
- responsive to flows

According to [Kelly 95], the ability of a multi-agent system to solve problems in some domain is distributed across and among the components of such a system. Problem solving is not explicitly embodied in any one agent. Looking at the problem solving process for complex adaptive systems (CASs) from another perspective, they are not -- according to Holland [Holland 96] -- constructed in a monolithic fashion but with small units of behaviour that this thesis chooses to describe as agents. Importantly, decentralization and aggregation are mutually supportive properties. The behaviour in a decentralized, aggregated complex system is distinct from the components' individual behaviours. The global behaviour of the system of agents is non-linear; the interactions are not simply additive. Another way of stating this [Resnick 94] is to observe that a flock is not a large bird (agent); flocking is not part of the behaviour of the individual bird (agent).

Similarly, a traffic jam is not a collection of cars. [Kelly 95] believes that functionality of systems exhibiting emergent behaviour grows incrementally, possibly with layers of higher order functionality being added over time, perhaps similiar in philosophy to the principle of Subsumption [Brooks 91]. In systems such as these, control is bottom up and massive concurrency exists in the system.

In fact, the massive concurrency and multiple simultaneous (possibly conflicting) goals generally facilitate robust behaviour in a dynamic environment. Agents can be thought of as competing for control within the problem solving domain. Diverse behaviour[1] of agents allows for a number of possibilities. Firstly, randomness can help create order [Resnick 94]. Secondly, competitive superiority, i.e., the measurement of the desirability of given behaviours in an environment at a given time can be used to guide the interactions of agents in the future [Kelly 95]. In other words, It is necessary that the environment provide feedback to the agents based upon the actions taken by those agents. Finally, diverse adaptive behaviour facilitates robust problem solving as it allows for constant exploration in an ever-changing environment. However, diverse behaviour may also lead to sub-optimal system response (at least in a formally proveable sense) in a static environment [Kelly 95].

The environment, however, is an active process [Resnick 94] that impacts the behaviour of the system and is not merely a passive communication channel between agents. An example of an active environment is the pheromone concentrations close to an ants' nest.

1. Diverse construction is also a differentiating factor. Changing the construction of an agent alters its ability to implement any one behaviour more or less efficiently; i.e., resource consumption changes.

These volatile chemicals evaporate and diffuse away from the nest. The processes of evaporation and diffusion are not ant behaviours, they are the behaviours of the environment -- in this case, the laws of physics and chemistry. In the case of an active environment, a system should constantly balance stability with change, otherwise it will tend to degenerate. Given response signals from the environment, positive or negative feedback for individual agent actions is possible. Conventional control theory espouses the view that negative feedback is preferable for system stability; however, given the previous sentence indicating the desirability of exploration (disequilibrium), positive feedback may be required for self-organization. In fact, it may be critical. For example, the foraging behaviour of ants depends upon a positive feedback mechanism. Action selection is based, in part, upon signal gradients in the environment and often by an agent moving through that environment. Stated another way, an agent adjusts its migration strategy based upon local differences (or fluctuations) in the strength of signals of interest to that agent.

Holland identifies three mechanisms that he believes necessary for emergent properties in complex adaptive systems. These mechanisms all relate to the sharing of information between agents. Firstly, agents are able to recognize and differentiate among one another. Secondly, the internal structure of an agent in a complex adaptive system enables it to anticipate changes in its environment. Finally, an agent's internal model is made up of small reusable modules that allow it to express a rich set of alternatives with a limited representational vocabulary.

3.3 Goals for and Attributes of a Self Organizing Agent System

Traditionally, functional decomposition of system activities has lead to goals or "things to be achieved" being ascribed to individual agents within the agent system. This has caused two things to occur. Firstly, the environment has tended to be considered as a communication conduit and this, we have seen, ignores its active nature. Secondly, localizing functionality within one (or a few) agents, tends to preclude emergent behaviour as, in some sense, the behaviour is "programmed in." The functional decomposition approach may also be totally in error, as in the example of traffic jam modelling. Where, in such a system, is the functionality for traffic jam creation explicitly represented in the behaviour of the car or in the "car system?" It cannot be, the behaviour is emergent. The temptation to equate the aggregation of behaviour with the aggregation of agent function can, then, lead to erroneous expectations of system behaviour in many cases.

Aggregation of agent behaviour is important for emergence, agent aggregation is not.

The next several sections describe the important agent characteristics that an architecture should provide in order to support self emergence.

3.3.1 Agent Size

Why is agent size important? The answer to this question, and thereby a design principle for our architecture, lies in the observation that human software engineers find it difficult to design large software systems that exhibit minimal decomposition. Monolithic systems become increasingly difficult (and expensive) to maintain and tend to exhibit decreasing

robustness as their overall complexity increases. Large populations of simple agents can exhibit a wide range of behaviours that depend upon agent density and migration rates for example that can lead to a wide range of emergent behaviours. Encapsulation, a concept from the world of Object Oriented programming languages, appears useful here. What *is* required is a clear boundary between the agent and its environment in order that signals may flow (possibly filtered) between the two. Interfaces are important in that they limit the exposed behaviour of an object (or agent). In other words, what we require is containment with selective sharing of information or, in other words, an airlock. The word airlock is chosen deliberately here as it the name of the mechanism used in CHAM (see section 2.6.1) for communication of information from one multiset to another. We should be careful to note here that we are referring to agent-to-agent communication here, not swarm to swarm as the concept of a swarm is a loose one, arising as a consequence of a set of agents sharing a particular (unknown to them) goal.

Agent size is kept small by having limited behaviour within it while providing an ability to communicate through an active environment. Small size therefore implies the need for agent interaction as no single agent knows how to solve a complex problem.

3.3.2 Agent interaction

Agents should be able to communicate effectively; however, point to point communication is not implied here. In order to communicate effectively, a common representation for agent interaction needs to be provided. While KQML (and other) agent communication languages (see section 2.2.5.3) allow efficient message transport in a

standardized way, processing of the message is implementation dependent and the ontology problem remains ever present. What is provided, and is considered important in this thesis, is the ability to match general patterns within the message body.

Such a *representation for interaction* should provide for two things; firstly, it should support pattern matching and secondly, message processing should facilitate agent aggregation. In some sense, our model of a multi-agent system should directly support aggregation, since an environment (and its associated agents) should become a higher level agent (in a logical sense) by defining its inputs and outputs to another environment.

Stated another way, agents should be arranged in *layers*. Another possibility is absorption, the physical integration of one agent within another.

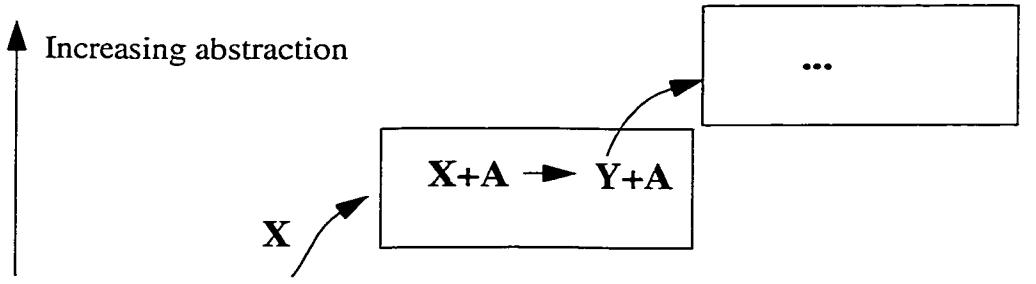


FIGURE 22. **Aggregation using CHAM-like principles**

3.3.3 Agent Aggregation

Using an airlock mechanism to facilitate sharing of information also provides a mechanism for aggregation. If we observe the processing mechanism present in CHAM -- that of the multiset transform -- we see that links between layers are implicitly defined by the transformations defined within the multiset. The symbols allowed through the airlock represent the linkages between a layer above or below it. The transformations within the

agent represent implicit linkages between layers above and below it.

This is shown graphically in Figure 22. In it, the symbol X is exported from a lower layer, is imported, and is transformed to Y which is, in turn, exported to the layer above. The symbol A represents a catalyst -- a precondition -- for the transformation of X to Y.

Absorption also represents a mechanism for aggregation. However, it has a disadvantage. If an absorbed agent represents a building block within a specific problem domain, absorption potentially removes a building block from the pool of agent expertise. Once absorbed, other agents may not be able to function effectively. Absorption can be useful when a more desirable building block is constructed; a process similiar in principle to the use of Automatically Defined Functions (ADFs) in Genetic Programming (GP) [Koza 92].

3.3.4 Agent Scope

Agents in naturally occurring systems sense and act locally. They do this for two reasons. Firstly, the energy requirements for doing so are much lower when comparing local to wider-ranging scope. The technology required for long range communication is also generally more complex. Secondly, the latency associated with long range interactions is much greater than with short range interactions. This latter observation makes learning more difficult as a consequence of the need to make multiple decisions before the reinforcement signal from the initial action is received. Longer response times also expose the agent to a greater risk of capture[1] if it decides to wait for a response from

1. We are thinking here of the classical predator prey environments that exist in nature.

the environment before moving on. Evidence to support this view can be found in the correlation of sensor scope to agent performance [Takashina 96] and in the sorting behaviour of ants [Deneubourg 90].

Local sensing can also reduce memory requirements as shown in the next section. The most important implication of locality of agent sensing and action is decentralized system control. Decentralization and distribution is axiomatic (in terms of importance) to Holland, Resnick and Kelly and, in this thesis, we argue the need for mobility in such agent systems.

3.3.5 Agent Memory

Agents must be capable of forgetting. It is important to note that an agent's state must be limited to a time horizon, i.e., it cannot remember everything that it has experienced. Remembering everything would lead to excessive memory requirements and, as shown below, can actually degrade performance. Equally important is the understanding that knowledge of events decreases in relevance as time passes.

We observe this in nature, pheromones evaporate leading to attenuation of gradients in the environment. Stated in physical terms, order tends to dissipate without energy expenditure, disorder then dominates.

In the simulated sorting behaviour of ants, memory must be limited in order to restrict the number of items carried by the ant and the number of steps "remembered" must be small otherwise sorting performance will be degraded. Also, with too many agents[1] too many items may be carried leaving too few in the environment to form nuclei for

developing piles. In other words, agent memory must be sufficiently large that environmental gradients can be sensed but not so large that these gradients contain no useful information.

Another dimension of memory is that of collective memory, i.e., the spatial and temporal patterns stored within a population of agents. There is obviously a linkage between the memory of the agent and the memory of the collective. For example, in the foraging behaviour of ants, if too few ants are sent out to forage for food, the pheromone trail evaporates more quickly than the reinforcement provided by other ants. In this case, the food source may never be effectively exploited.

3.3.6 Agent Diversity

The examples in section 2.3.2, 2.3.3, and 2.3.4 demonstrate the value of diversity of agent action in problem solving. In the behaviours described in these sections, diversity is achieved by action selection processes that use weighted probabilistic mechanisms. Mathematically, this is equivalent to a population of agents with fixed behaviours where the proportion of agents of each type is described by the aforementioned weighted probabilistic mechanisms. Diversity is valuable in order to allow for a problem space to be searched using many distinct strategies. Also, the effectiveness of a population of agents with high diversity will generally be greater in dynamic environments as no single strategy is optimal in all environments [Wolpert 99]. In other words, any single agent can model and manipulate only a small portion of the overall environment, and a population of

1. If the number of agents equals or exceeds the number of items to be sorted it is probable that all will be picked up and never sorted.

identical agents will do no better, since they will only be able to cover the same subset of the environment's state. A population of diverse agents will cover more of the environment's state and thereby provide better performance.

In sections 2.3.4 and 2.3.5, diversity of agent behaviour is achieved by constraints on agent attributes. In section 2.3.4, wolves "repel" each other such that they are separated just enough to surround their prey without the gaps between them being so large that their prey may slip through it. In section 2.3.5, birds "repel" each other such that they will be able to sense changes in direction and velocity of a small number of birds nearby without crashing into any of them. In some sense, the birds in a flock "attract" each other too; individual birds attempting to match direction and velocity vectors with the centre of the small number of birds close by. It is balance of attractive and repulsive influences that maintains the flock as a whole. The probabilistic choice of action generates diversity.

3.3.7 Environmental Potential Fields

In section 3.3.4 we argued the need for agent mobility in distributed, decentralized systems with limited agent sensory scope. Indeed, in the examples cited in sections 2.3.2 to 2.3.5, the agents all exhibit mobility. In fact, the vast majority of species within the animal kingdom exhibit the characteristic of locomotion and even plants use phototropic and geotropic mechanisms during growth. The question arises, "What drives the migration decision process?"

The answer to this question is to observe the existence of attractive and repulsive fields within the environment. In the example of ant foraging, gradients of pheromones exist

within the environment. In the example of ant sorting, similarity gradients exist in terms of the objects sensed by the ant. Using the examples of the previous section, wolves and birds perceive repulsive fields defined by the proximity of other wolves or birds respectively. An important observation is that the potential fields are created and maintained by the agents themselves. In the case of ant foraging, ants lay pheromones when returning to the nest. In flocking, wolves or birds moving closer together cause an increase in the local repulsive field.

While the fields themselves are important, their maintenance by agents within the system is equally important from the point of view of self-organization of multi-agent systems. As the examples presented have shown, natural agent-based systems organize themselves with striking efficiency. The Second Law of Thermodynamics tells us that closed systems become progressively more disorganized over time. It is not obvious, therefore, that a large collection of agents will organize itself to do useful work or solve complex problems.

While the addition of energy to the system is necessary to avoid the consequences of the Second Law, it is not enough to ensure emergent behaviour. Rather, agents within natural systems can organize themselves at the macro (global) level because their actions are coupled to a dissipative or disorganizing force at the micro (local) level. The system reduces entropy as seen at the macro level by generating more than sufficient entropy at the micro level to ensure that the Second Law is not violated. In other words, entropy leaks from the macro level (which is where problem solving activities are performed and

emergent behaviour is observed) to the micro level (where stochastic behaviour is the norm and not affected).

This leakage effect generates a flow field that the agents can perceive and reinforce [Kugler 87]. Perception of the field allows migration decisions to be made. Insect colonies deposit pheromones whose molecules, spreading through the environment by diffusion, generate, and subsequent leak, entropy. This diffusion process generates a flow field that the insects can perceive and to which they can orient themselves in making further pheromone deposits. The interplay of entropy, flow fields and macro and micro behaviours is shown in Figure 23. In emergent systems there are three fundamental processes: micro dissipation, macro perception of the micro flow field, and macro reinforcement of the micro dissipative field.

Dissipation of the signal field may also fall within the responsibilities of the agents. For example, the movement of currency in a market economy is from purchasers to sellers.

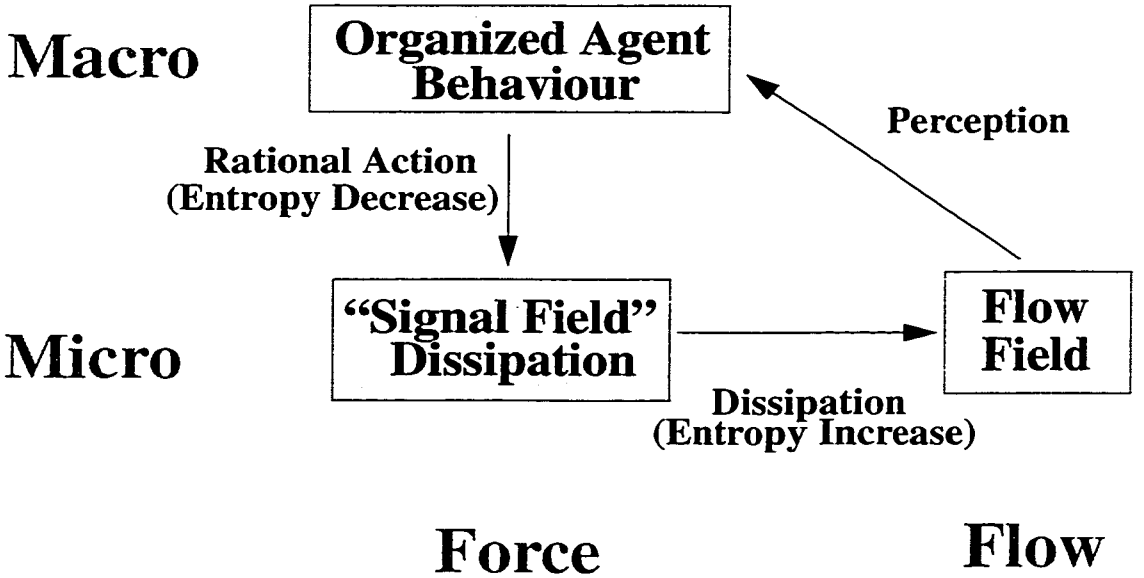


FIGURE 23. **Macro Organization through Micro Dissipation**

Entrepreneurs see the resulting the flow field -- or rather its fluctuations -- and orient themselves to it, resulting in the emergence of structures such as local economies, markets and supply chains.

To summarize then, agent communities should implement mechanisms with the following three characteristics in order to have the potential for self organization:

- A signal must flow either among agents or through the environment, thereby setting up a gradient field.
- The agents must be able to perceive the field and orient themselves to it.
- The agents' actions must reinforce the field, providing positive feedback.

Market oriented programming [Clearwater 96] exploits these principles using currency as the dissipative signal field. [Drogoul 95] exploits similiar dissipative principles using simple agents to play a game of chess.

3.4 Introduction to Architectural Specification

As we have seen in the previous chapter, Nature provides us with several examples of social systems comprising individuals exhibiting simple behaviors while the society exhibits complex problem solving capabilities.

Naturally occurring social systems provide considerable inspiration for artificial systems that display emergent behavior and this chapter has discussed the important attributes of such systems and their environments. Systems exhibiting emergent behaviour promise to provide guiding principles for, and engineering solutions to, distributed systems management problems found, for example, in communications networks.

This chapter continues with a presentation of our multi-swarm architecture and how it

fulfills the requirements for emergence analyzed earlier. Following a presentation of the architecture, we describe a scenario drawn from the communications domain where a multi-swarm architecture has been used. Implementation issues, both by simulation and through the use of a mobile agent toolkit, are then briefly described. The scenario presented is analyzed in the following two chapters.

3.5 Agent System Architecture

In our system, ant-like agents solve problems by moving over the nodes and links in a network and interacting with chemical messages deposited in that network. Chemical messages have two attributes, a label and a concentration. These messages are the only medium of communication used both between swarms and individual swarm agents. Data and chemicals are considered synonymous in our system.

Agents in our multi-swarm system are of limited intelligence; i.e., they belong to the 'lightweight' category of agents, and are capable of only simple behaviors. Such agents are reactive in nature and have the ability to sense and modify their environment locally. Our agents stand in stark contrast to agents supporting the Belief-Desire-Intention (BDI) model [Shoham 93]. However, we freely acknowledge the desirable nature of hybrid reactive-reflective architectures such as the Touring Machine architecture [Ferguson 95] and, in fact, our lightweight agents interact with stationary agents on platforms used for management and planning in our networks. Having the capability for mobility, ant-like agents are potentially able to modify local environments on network elements (or components) in the entire network that they inhabit. Agents communicate locally, when

co-resident on a node and only through their local chemical environment.

Agents in our system can be described by the tuple, A=(E,R,C,MDF,m). They have a uniform architecture consisting of five components:

- emitters (E),
- receptors (R),
- chemistry (C),
- a migration decision function (MDF),
- memory (m)

Analyzing the above components, we can see that SynthECA agents may be thought of as cells, cells which migrate through a network performing their individual, specialized functions through interactions with other cells. Cells of the same type may be thought of as swarms and emergent behaviour arising in two ways: through the collective dynamics of a swarm and through the aggregate behaviour of swarm collectives. The idea of a cell is appealing because of the cell membrane and associated cellular structures. The cell membrane facilitates the movement of particular chemicals from inside the cell to its surrounding environment. We would tend to view our agents as prokaryote cells -- rather like bacteria -- having a simple structure with no nucleus. However, the membrane bound organelles of the eukaryote cell -- shown in Figure 24 -- could be thought of representing the chemistry in our agent architecture. Once again we see the influence of the Chemical Abstract Machine -- the airlock mechanism providing similiar functionality to that of a

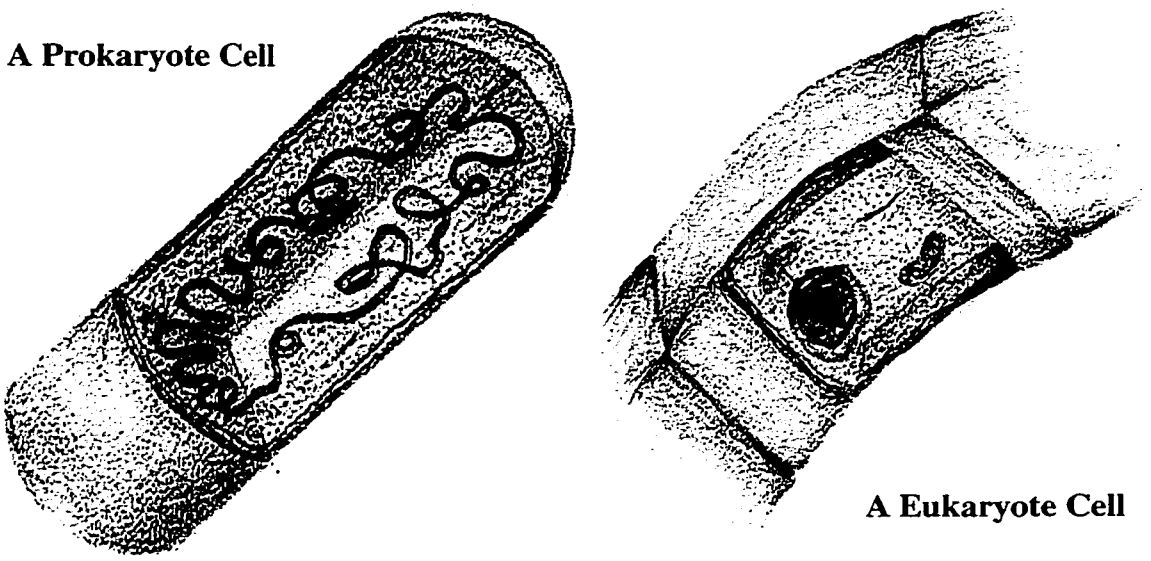


FIGURE 24. **Examples of Cell Types**

cell membrane and the membrane-bound organelles representing the multiset transform.

The metaphor of a chemical reaction plus that of a prokaryote cell along with import of reactants and export of products provides for the layered architecture of SynthECA agents that is shown in Figure 25. In it, we see the interaction of swarms of agents generating and processing chemicals that are used as excitatory and inhibitory signals to swarm levels above and below it. This figure has a considerable number of architectural similarities when compared to that of the Subsumption architecture shown in Figure 5 on page 25 and Figure 6 on page 28. Swarms become competence modules and linkages between them are replaced by activating or deactivating signals represented by concentrations of specific chemicals. We will return to the concept of layering later in the chapter.

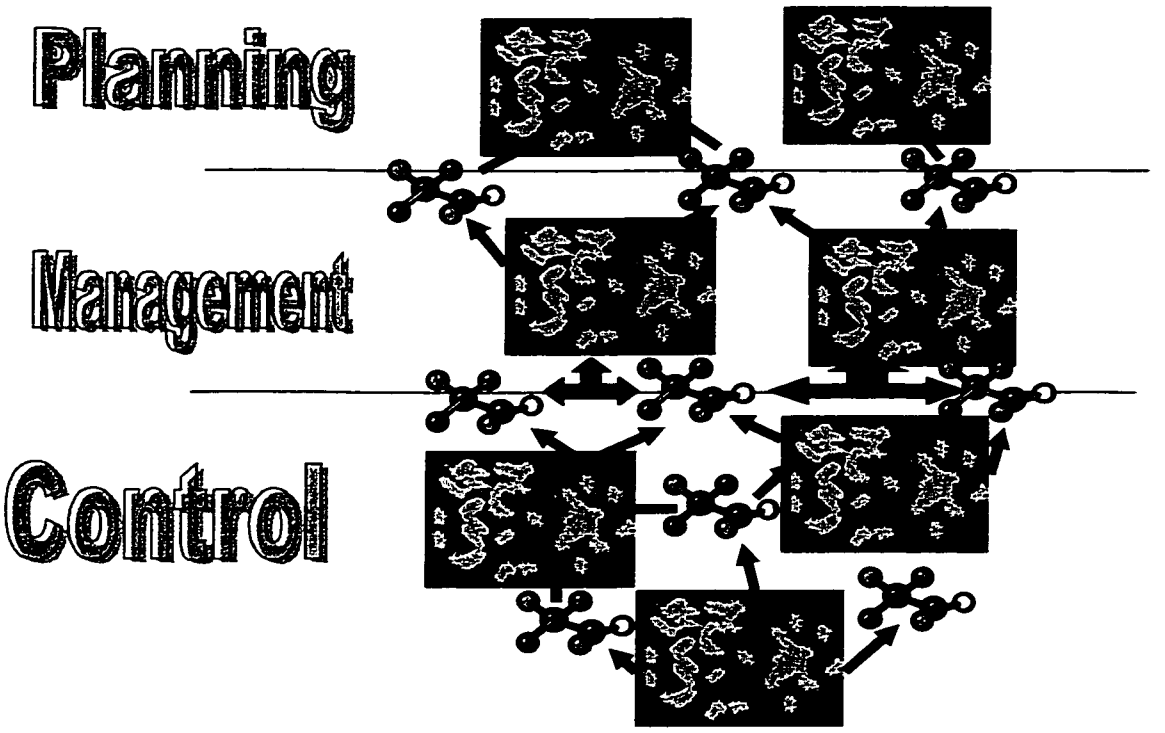


FIGURE 25. **Layered Architecture for SynthECA Agents**

3.5.1 Chemicals and the Chemical Universe

Central to the architecture is the concept of a chemical. The chemical concept is used in order to provide communication between agents and to create dissipative fields within the environment. The chemical concept is used to provide communication with and sensing of the environment and provides the driving force for agent mobility. A chemical consists of two components, an encoding and a concentration. The encoding can be thought as the description of the chemical, in the same way that C_2H_5OH is the internationally recognized description for ethanol.

Essentially, the chemical encoding is a data structure, G(Al), defined on some alphabet, Al as a production of some grammar, G. A data structure consists of one or more locations, with connections between them. The alphabet describes the contents of each and every

location within the data structure. For example, a chemical could be defined by an array of size *N* with an alphabet given by the set, {1, 0, #}. In this case, each element of the array would be either 1, 0 or #. Another example might be a n-ary tree, T, with the alphabet, {1, 0, #, $*_n$}, where each node in the tree contains one of the alphabet set.

Chemicals participate in reactions and support generalized pattern matching, defined by matching rules. Considering the n-ary tree example of the previous paragraph, the # symbol matches the contents of a single location and the $*_n$ symbol matches a subtree. The subscript here allows for matching and subsequent reuse elsewhere in a chemical reaction; e.g., taking part of a reactant and having it appear within a product of the reaction. We will return to this point in Section 3.5.4 on page 114, the section dealing with the chemistry of an agent.

More formally, the following definitions apply.

Chemistry: The chemistry (or chemical universe) associated with SynthECA is defined by the tuple Cu = <Al, G, Map>, where Al is the alphabet for the chemistry, G the grammar used to generate chemicals or productions, and Map the mapping rules for the alphabet.

Alphabet: The alphabet, Al, is a set of symbols associated with a chemical universe. This set defines the contents of each and every location within the chemical structures that can be generated using G. The set consists of two classes of symbols. *Grounded* symbols have the property that they only match themselves and no other symbol in the alphabet. Stated mathematically, g is a grounded symbol if

$\{b{:}m(g) \equiv b, g \in Al, b \in Al\} = \{g\}$, where m(g) is the mapping function applied to g. *Unifier* symbols have the property that they match other members of the alphabet set. Stated formally, the symbol u is a unifier if, $\{b{:}m(u) \equiv b, u \in Al, b \in Al\} - \{u\} \neq \varnothing$ .They may or may not match with themselves. *Simple unifiers* match a subset of the alphabet set only excluding themselves. *General unifier* symbols match a subset of the alphabet, including themselves.

Chemical: We define a chemical as Ch(Enc,Re), where Ch(Enc,Re) represents a chemical with an encoding Enc and a concentration Re represented by a positive, real number. The encoding Enc is given by G(Al), a production of the grammar, G, defined for the alphabet Al.

Grounded Chemical: A Grounded Chemical, $Ch(Enc_g,Re)$, is one whose encoding contains only grounded members of the alphabet. Stated another way, grounded chemicals may only be mapped to themselves within the chemical set by application of the mapping rules. If $S = \{Ch(Enc, Re){:}MR_e(Enc) \equiv Enc_g\}$, then the cardinality of the set S is 1. Chemicals that do not possess the grounded property are referred to as unground.

Mapping rules: The mapping rules for a chemical universe consist of two elements, the encoding mapping rules, MR_e, and the concentration mapping rule, MR_c. The encoding mapping rules define equivalence relations between sets of productions that are considered the same. The concentration mapping rule defines how the concentrations of chemicals, Ch(Enc,R) are computed given the concentrations of the grounded chemicals

produced by application of the encoding mapping rules. If $S_g = \{Ch(Enc_g, Re_g): MR_e(E) \equiv Enc_g\}$ for some encoding Enc, then Re is given by $Re = MR_c(S_g)$. Trivially, Ch(Enc, Re) = $Ch(Enc_g, Re)$ if Enc is grounded, regardless of the mapping rule used[1]. This result arises directly from the cardinality of the set *S* being 1. It should be noted that a closed chemical world is assumed, an environment that does not contain a specific grounded encoding effectively describes the situation $Ch(Enc_g, 0)$. The concentration mapping rules defined are: *min*, *max*, *avg*, *plus*. These rules return the minimum, maximum, average and sum of concentrations of chemicals in S_g respectively.

Expressiveness: The expressiveness of a chemical is the cardinality of the set S_g.

Binary Alphabet: The Binary Alphabet defines Al = {1, 0, #} with the mapping rule $\{b: m(\#) \equiv b, u \in Al, b \in Al\} = \{1, 0\}$, and the symbols 1 and 0 being grounded. This encoding has been inspired by those typically used in Genetic Algorithms [Goldberg 89] and Classifier Systems [Holland 86].

Important Chemical Universes: We are interested in two chemical universes in this thesis. First we define the Binary Array Chemistry of order N, BAC(N), and second we define the Binary Tree Chemistry of size N, BTC(N).

Binary Array Chemistry of order N: BAC(N) uses the Binary Alphabet. The grammar

1. Pathological functions for MR_c could be defined such that this does not hold, hence the restricted set provided.

for chemical productions uses an array of size N and selects a member of the alphabet for each position in the array. The encoding mapping rules are simply the application of the alphabet mapping rule to all positions in the array. The concentration mapping rule is plus.

Binary Tree Chemistry of order N: BTC(N) uses the Binary Alphabet. The grammar for chemical productions creates a node with two children, selecting a member of the alphabet for the contents of the node and recursively applying the grammar to the children until N nodes have been generated. The encoding mapping rules are simply the application of the alphabet mapping rule to all positions in the tree. The concentration mapping rule is plus.

Properties: These two chemical universes have a number of interesting properties. First, they are both closed. A closed universe is one in which the set of all possible chemicals has finite cardinality. An open chemistry is one with an infinite cardinality. Second, the cardinality of BAC(n) is 3^n while the cardinality of BTC(n) is $\frac{3^n}{n+1}\binom{2n}{n}$. The relative expressiveness of chemicals within these two chemistries is quite different. In both BAC(n) and BTC(n), the expressiveness of a chemical is 2^m, where m is the number of # symbols in the data structure. However, even though the absolute expressiveness of the two chemistries is identical, the relative expressiveness of the two chemistries when measured as a fraction of their universe size is $(n+1)/\binom{2n}{n}$.

Open chemistries are also of interest but go beyond the scope of this thesis. An open

chemistry allows for the generation of complex structures from simple building blocks. We will comment on the utility of open chemistries in the final chapter.

3.5.2 Emitters

The emitters associated with an agent are used to generate chemicals that are deposited where the agent is currently located. Using ants and their foraging behavior as an example, pheromones are laid down as the ant returns from searching for a source of food. Emitters have an associated Emitter Decision Function (EDF) which is used to decide the rate of production of an emitted chemical. The emitted chemical is digitally encoded, having an associated pattern that uses the Chemistry associated with the agent; e.g., the Binary Chemistry. The hash symbol in the alphabet allows for matching of both one and zero and is, therefore, the "don't care" symbol. A chemical encoding including one or more "don't care" symbols can be thought of as a generalized chemical or an instance of several classes of chemical.

The function of an emitter is to alter the local environment inhabited by the agent. Using the above alphabet it is possible, for example, for an agent to generate a digital chemical with the encoding 1#01 which will be sensed by an agent (as we shall see in the next section) with a 1101 receptor and by an agent with a 1001 receptor.

An emitter can be either on or off depending upon its internal state; i.e., the concentrations of chemicals stored within agent memory.

3.5.3 Receptors

The receptors associated with an agent are used to sense chemicals that are present in the

agent's local environment and chemical changes that occur in it. Using, once again, ants and their foraging behavior as an example, pheromones are sensed by the ant as it searches for a source of food. Receptors have an associated Receptor Decision Function (RDF) that is used to determine the sensitivity to the chemical in question and it is possible to associate actions with a receptor. The sensed chemical is digitally encoded, once again having an associated pattern that uses the agent Chemistry; e.g., the Binary Chemistry. It is possible, therefore, to engineer wide spectrum sensors that detect many chemicals. For example, a receptor engineered to sense the encoding 10## will be able to detect the chemicals having the 1000, 1001, 1010 or 1011 encoding. Like an emitter, a receptor can be either on or off depending upon its internal state.

3.5.4 Chemistry

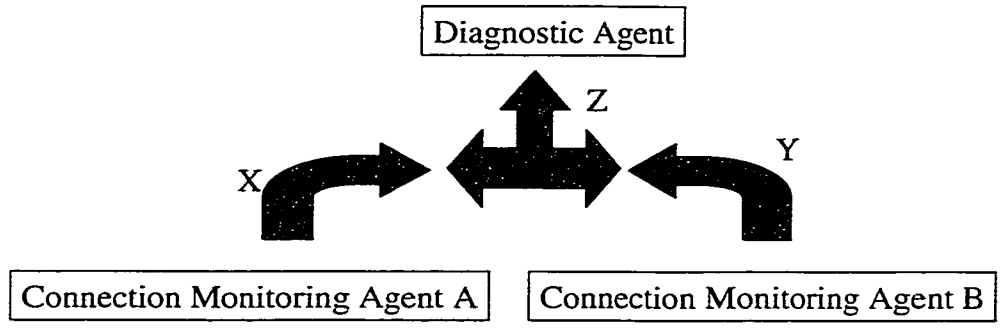
The chemistry associated with an agent is the set of chemical reactions that can occur within the agent. While the reaction set is limited to, at most, two reactants or products, larger reactions can be synthesized by building chains of these five types of reaction. Reactions are read from left to right. This is important when applying the mapping rules associated with a chemical universe as will be demonstrated in an example later in this section. There are five types of reaction that can occur within an agent. These are shown on the next page.

$$X \rightarrow \text{'}nothing\text{'} \quad (1)$$
$$X + Y \rightarrow Y \quad (2)$$
$$X + Y \rightarrow Z \quad (3)$$
$$X + Y \rightarrow X + Z \quad (4)$$
$$X + Y \rightarrow W + Z \quad (5)$$

Reaction Types

The first reaction can be thought of as evaporation of a chemical. An example of such a reaction would be the evaporation of pheromone from an ant trail. The second type of reaction is the catalytic breakdown of a chemical, with Y representing the catalyst. An example of such a reaction might be a parasitic interaction between two types of agent such as could be observed when one ant is trying to throw another 'off the scent' when competing for finding a path to a given food source. Another example, this time from the telecommunications domain, is the scenario where an agent representing higher priority traffic reduces the concentrations of lower priority traffic's pheromones (that are used to mark a given route) in order to have the lower priority traffic find an alternate route. The third reaction type represents the fusion of two chemicals and it is this type of reaction that we envisage being used to communicate information from one layer of a multi-swarm hierarchy to another. This type of reaction provides a mechanism which multi-swarm systems could use to implement Subsumption Architectures [Brooks 86], [Brooks 91]. This is shown diagrammatically in Figure 26.

Figure 26 shows two connection monitoring agents that have quality of service monitoring for a specific connection as their primary responsibility. They detect decreasing quality of service for a shared network resource on, say, a link. As a result, they



Monitoring agents A and B lay down increasing quantities of pheromone with decreasing quality of service on a given link. Diagnostic agent senses increasing levels and initiates diagnostic activity when threshold exceeded.

FIGURE 26. **Type 3 Reaction Example**

lay down quantities of X and Y that indicate an increasing level of dissatisfaction with the quality of the connection. A diagnostic agent encoding a type three reaction has one or more receptors that allow for the detection of X and Y, allowing for the generation of Z.

The fourth reaction type represents a catalytic reaction where one chemical is converted to another but *only* in the presence of a mediating chemical, the catalyst. This type of reaction can be thought of, in computational terms, as providing a conditional construct where, only if we have a certain confidence in a given state can we perform a specific transformation of one chemical to another. The fifth reaction type is the most general in that the two reactants are converted to two products that are distinct from the reactants. Depending on whether a given chemical is part of one swarm layer or the swarm layer above it, the five types of chemical reaction can be considered as providing both inhibitory and excitatory stimuli to the upper swarm layer. For example, Figure 26 can be viewed as providing an excitatory stimulus to the fault detection swarm layer considered being

above the connection-monitoring layer within our multi-swarm architecture.

All of the reactions use digitally encoded chemicals, i.e. all chemicals use the {1, 0, #} alphabet. Hence, reactions of the form below are supported[1].

$$011 + 1\#0 \rightarrow 1\#0$$

The above reaction, an example of a type two reaction, allows for the catalytic breakdown of the 011 chemical to occur via either of two catalysts, namely 110 or 100. Unification occurs between chemicals on a bit-by-bit basis and is carried through from reactants to products, i.e., the "don't care" symbol in a given position within two reactants or between reactants and products can be either 1 or 0 in a single reaction. Using the above reaction as an example, four possibilities exist. These are shown below:.

$$011 + 110 \rightarrow 110$$
$$011 + 100 \rightarrow 100$$
$$011 + 110 \rightarrow 100$$
$$011 + 100 \rightarrow 110$$

The first two reactions are implied, with the "don't care" symbol being unified to 1 and 0 respectively. However, the latter two reactions are not implied, as in both reactions the "don't care" symbol in the second position has to unify to both 1 and 0.

All of the reaction types have an associated reaction rate, i.e. a measure that determines the speed with which the reaction can occur. Reaction rates are temperature dependent, with the dependence characterized by Arrhenius' equation, shown on the next page, where k is the rate coefficient, A is a constant, E_a is the activation energy, R is the universal gas

1. Note that the concentrations of the two chemicals have been assumed to be one in this example.

constant, and T is the temperature (in degrees Kelvin).

$$k = A\exp(-E_a/(RT))$$

Temperature and energy are considered to be essentially the same in our system. Consequently, a unique chemical encoding that can be generated by chemical reactions (as any other) has been chosen to represent temperature. By using the same representation for energy and chemicals, endothermic and exothermic reactions can be used to cool and heat the system respectively. Endothermic reactions are characterized by a decrease in temperature and, as such, reaction types 2, 3, 4 and 5 on page 115 can represent this type of reaction. This is shown in the example reactions below, where T is meant to represent the energy consumed by the reaction, i.e. it appears on the left-hand side of the reaction.

Similarly, reaction types 3, 4 and 5 as is shown in the example reactions above may represent exothermic reactions. In these reactions T once again represents the energy generated by the reaction, i.e. it appears on the right-hand side of the equation.

$$T+Y \rightarrow Y$$
$$T+Y \rightarrow Z$$
$$X+T \rightarrow X+Z$$
$$X+T \rightarrow W+Z$$

Endothermic

$$X+Y \rightarrow T$$
$$X+Y \rightarrow X+T$$
$$X+Y \rightarrow W+T$$

Exothermic

Changing the *local* temperature of the system changes the degree to which swarms interact. Low temperatures see little interaction between swarms whereas high temperatures see high levels of interaction. It should be stressed that temperature is a local characteristic of the environment and no attempt is made to make this information

globally available. The temperature chemical can be thought of as a local control parameter limiting or promoting agent interaction, i.e. providing inhibitory or excitatory stimuli within the multi-agent system.

In our system, all agents are provided with a temperature receptor by default, thereby being able to sense the local temperature. However, this need not be the case, one could imagine a design where internal and external agent temperatures were maintained.

3.5.5 Memory

The memory associated with an agent stores the chemicals and their concentrations that are held internally to the agent. It is the holder of the state of the agent. Symbolic information can also be stored in memory; however, the agent alone may use this type of information. These types of agent cannot communicate such information to the environment. Only chemicals for which emitters or receptors are not provided are stored within agent memory.

3.5.6 Migration Decision Function

The Migration Decision Function (MDF) is a function or rule set that is used to determine where an agent should visit next. The MDF typically uses chemical and link cost information in order to determine the next hop in its journey through the network or may simply follow a hard-coded route through the network. This latter migration strategy is often referred to as an itinerary in the mobile agent literature. Alternatively, when migrating, the agent may use the default migration node available to it. An important consideration in designing an MDF is that it should take advantage of gradients in

chemicals that are present in the network. In doing so, agents may take advantage of the actions of other agents. Particular agents may want to move up a gradient (attraction) or down a gradient (repulsion). The MDF may take advantage of the pattern matching properties of the language used for the chemistry of the system. For example, consider the Binary Alphabet introduced earlier and used as part of a Binary Chemistry of order 2. We might include a term in the MDF consisting of the chemical 1#.

Consider a scenario where an agent has the choice of two links. Link 1 has concentrations Ch(10, 0.1) and Ch(11, 0.7). Link 2 has concentrations Ch(10, 0.5) and Ch(11, 0.6). Thus, an agent moving up a gradient indicated by the 1# pattern would follow link 2 because Ch(1#, 1.1) is sensed for that link. Similarly, an agent moving up the gradient indicated by the 11 pattern would follow link 1.

3.5.7 Agent Action Primitives

Agent action primitives are invoked during evaluation of the receptor and emitter decision functions. The focus of the architecture proposed by this thesis is self emergence. As such, the principles and mechanisms needed to support it are described. Moreover, the agent action primitives will, by and large, be domain specific and, therefore, should not be part of a high level specification such as provided here.

However, a number of important generic primitives are provided. First, a migrate primitive is provided and causes the migration decision function associated with the agent to be evaluated in the current environment resulting in a decision to move to an adjacent node. Second, a clone primitive is provided. This creates a duplicate of the agent invoking

the primitive, giving the child agent an independent thread of control. Finally, a diffuse primitive is provided. This primitive allows a proportion of a chemical to be removed from the current environment and divided equally among the environments of all adjacent nodes.

3.5.8 Observations of the SynthECA Architecture

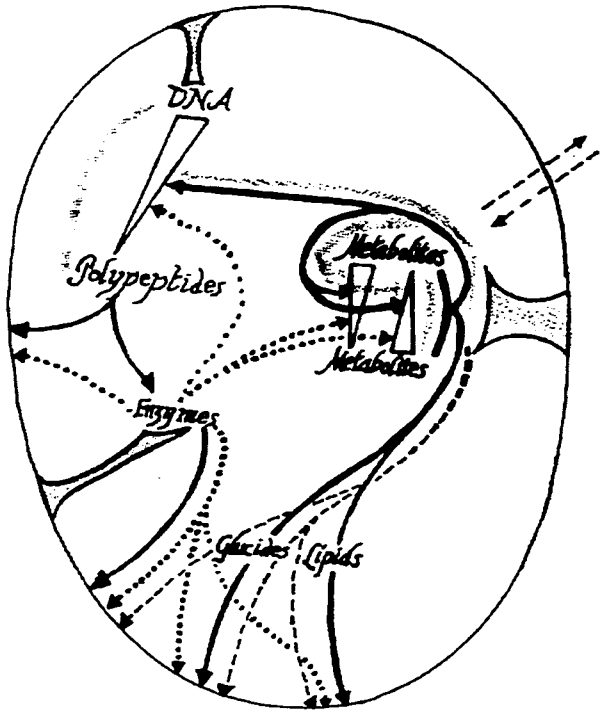


FIGURE 27. **A Cellular Autopoietic Network**

The cellular figure to the left is taken from Maturana and Varela and is used by them to introduce autopoietic concepts. The essence of this figure is autopoiesis and the relationships within a cell that make self creation possible. The reader cannot fail but to notice the striking similarities between it and the agent architecture described in section 3.5. The cell chemistry, reaction pathways and chemical exchange with the cell environment are all present.

Section 3.3 on page 95 put forward a number of desirable characteristics for an agent architecture such that collections of appropriate agents might possess the property of self organization. In this section we argue as to how the architecture described in this chapter provides these characteristics.

Agent Size: In SynthECA, it is intended that agents are reactive, i.e., they do not contain planning, execution and scheduling components as described in Section 2.2.1 on page 22 and shown in Figure 4 on page 24. Knowledge in a SynthECA agent is represented in two ways: in the chemistry of the agent and in the receptor and effector decision functions. Given the small number of chemical reactions, receptors and effectors, the expertise of a SynthECA agent is considerably smaller than a BDI agent.

Agent interaction: Effective communication is provided through the exploitation of the chemical metaphor. Chemical reactions ensure that the architecture is not biased towards any serial processing; all reactions occurring in parallel. The chemical metaphor also provides for effective coupling of agent and environment as it is explicitly represented in agents' chemical reactions and the pattern matching properties associated with chemicals. Effective interaction is supported by encapsulation, the agents' effectors and receptors providing this.

Agent Aggregation: Aggregation is supported by layering. Layering is facilitated by the use of a single form of communication -- the chemical. All interfaces between layers are defined in terms of chemical signals; the receptor and effector mechanisms providing the airlock mechanism described in CHAM. Encapsulation of this form ensures straightforward replacement of one agent type within another, as demonstrated in Figure 28. The encapsulation argument may be extended across multiple layers in a straightforward way. Aggregation as implied by Subsumption is supported through the alphabet used to describe chemicals. By allowing pattern matching, we facilitate the

integration of a wide range of signals from lower level agents by a higher level swarm. Similarly, if a chemical, C, means the same thing to a number of lower level agents, i.e., they have a partial ontology in common, integrating C into the higher level agent allows it to subsume the lower level behaviours.

Describing this functionally, and with a concrete example, we could think of a layer in which agents perform routing functions, and static agents monitoring the quality of service of the connections which these routing agents create. A layer above this would contain agents responsible for fault location. These agents would sense quality of service changes as deposited by the connection monitoring agents, locate and fix faults in the network, and deposit "fault indicating" chemicals in the environment. These same fault indicators, along with routing chemicals are used by the routing agents in order to bias routes away from unreliable components. Finally, network planning agents would sense fault indicating and congestion indicating chemicals in order to initial partial re-planning of the network. In essence, we have three layers: control, management and planning. This will be demonstrated in the next three chapters and is the essence of Figure 25 on page 108.

Agent Scope: The scope of an agent is the portion of the environment that it can sense. Limited, or local, scope is supported directly; agents may only sense their immediate environment and act locally. Agent scope includes nearest neighbour nodes in a graph. This is necessary in order to sense gradients in the local environment.

Agent Memory: Memory contents are constrained to be symbols of internal interest and chemicals that are not communicated to the local environment. Acting and sensing locally

tends to devalue the long term storage of sensory input from previous environmental interactions.

Agent Swarms: The architecture supports the concept of a swarm by allowing for multiple autonomous agents with the same receptor and effector apparatus that enables agent actions (placing concentrations of particular chemicals in the environment) to affect the behaviour of other swarm agents; positive reinforcement is supported.

Agent Diversity: Restricting the discussion purely to the communication mechanisms described by this agent architecture, diversity is easily supported within a swarm and between swarms. For example, an MDF with a probabilistic component to it as shown in equation 6, where p_{ij} represents the probability of moving from node i to node j, S_j is the concentration of a specific chemical, s, on the j^{th} node as sensed by the agent, C_{ij} is the

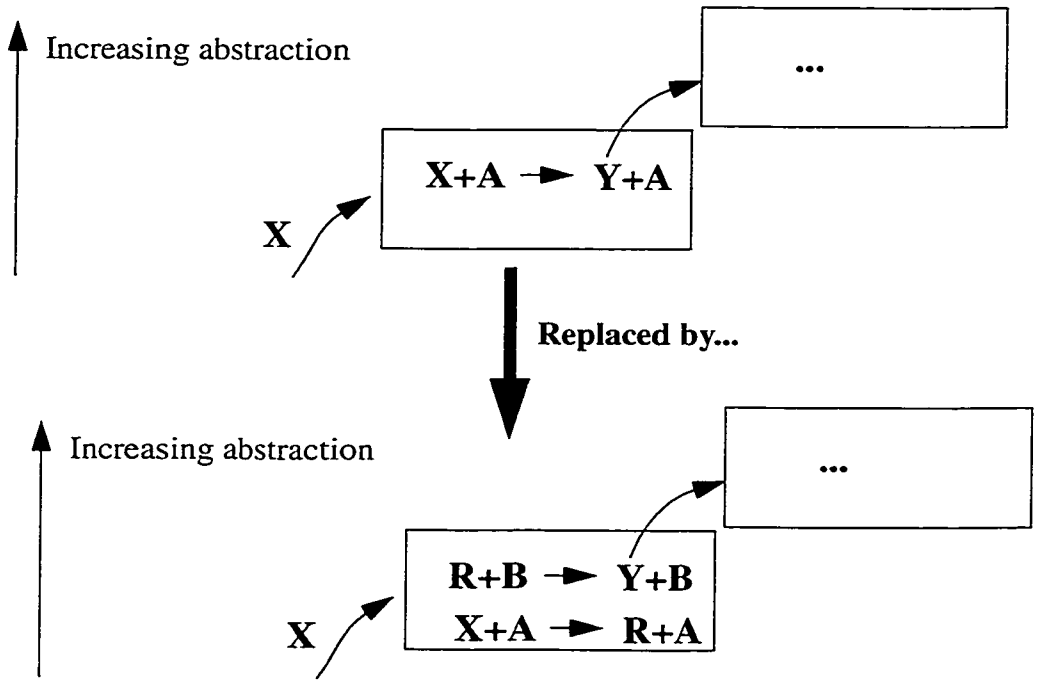


FIGURE 28. **Aggregation facilitated by chemical interaction**

cost of the link from node i to node j, and a and b are constants, obviously supports

$$p_{ij} \propto S_j^{-a} C_{ij}^{-b} \qquad (6)$$

diversity through non-determinism similiar to Resnick's Randomness Principle. If a swarm of agents were to be constructed with this form of MDF, the swarm would exhibit diverse behaviour as a consequence of individual agents executing a biased random walk within the network.

Inter-swarm diversity is achieved through a number of mechanisms, although they are all derived from basic sensory differences. First, using different effectors and receptors causes varying perception of the same environment. Second, the points of interaction between swarms can cause diverse behaviour as a result of one agent affecting another. This is best demonstrated with an example.

Imagine two agent types representing servers. Both agent types sense a common chemical, C_s. Now consider the migration decision functions for the two agent types. These MDFs are deterministic and state, "If the concentration of C_s exceeds $t(C_s)$, then migrate to a random neighbour." Let the rate of production of C_s be 1 per unit time for the first agent, 0.5 for the second and $t(C_s)$ to be just less than 1.5. Finally, we assume the complete evaporation of C_s between time steps. With this scenario, the two agent types that end up on the same node at the same moment in time will spontaneously migrate to random nodes whereas two agents of the second type may coexist owing to the different threshold values. Hence diversity of behaviour between swarms.

Environmental Potential Fields: One of the most important observations made in this chapter is that of the need for potential fields within the environment and the ability for agents to sense them and orient themselves to those fields. SynthECA contains potential fields defined through the connectivity of the network; i.e., the local neighbourhood is defined, and the laying down of concentrations of particular chemicals on the nodes in that network. Differing concentrations of particular chemicals between nodes then define gradients within the network. The potential fields can be either attractive or repulsive, this being determined by the agent through its migration decision function. For example, consider the MDF shown in equation 6. The field as defined by the concentration of chemical s such that $S_j > 1$ can be considered attractive in the sense that the agent moves towards nodes of high concentration of s if $a < 0$, and repulsive in the sense that the agent moves away from nodes of high concentration of s if $a > 0$. If a is zero, the agent is indifferent to the field. An agents' perception of the potential fields within a network may also be complex as a result of the pattern matching capabilities of the language chosen for chemical expression. Consider, for example, the alphabet {1, 0, #} and two chemicals 10 and 11. Now consider two nodes, the first with concentrations 0.5 and 1.0 and the second with 1.0 and 0.5 for chemicals 10 and 11 respectively. Obviously there are gradients for both chemicals between the two nodes. However, consider an agent that has an MDF that is defined by equation 7. The index of the node chosen for migration, i, at some time, t, is

$$i(t) = max_j(Y_j(t)) \qquad (7)$$

determined by the node with the maximum concentration of the chemical $Y_j(t)$, in this

example represented by the encoding 1#. Looking at the concentrations for the two base chemicals (10 and 11), we see that there is no gradient as Y for both potential destination nodes is 1.5 (=1.0+0.5).

Self organizing criticality: Chemical systems such as those described in [Eigen 79] exhibit the kind of insensitivity to initial conditions that characterize self organizing critical systems [Bak 96]. While no convergence or stability proofs are included in this thesis, the wide body of literature on the analysis of chemical systems, it should be possible to exploit it in the analysis of systems based upon this architecture.

3.5.9 Agent Operation

While the chemistry of an agent appears similar to a classifier system at first glance, it is only superficially so. Firstly, the agent chemistry is fixed and no Bucket Brigade algorithm ([Goldberg 89], for example) or similar apportionment of credit scheme is intended to operate in order to modify the chemistry. An agent's chemistry is *fixed*, having been engineered in order to achieve a given function within the mobile agent subsumption architecture. Secondly, an agent operates continuously and all reactions operate in parallel in order to modify the local environment. This is quite different from the way in which message processing occurs within a classifier system. Upon arrival at the node, an agent registers interests in particular chemicals. Chemical concentration changes caused by agent chemical reactions are communicated to the local environment for which the agent has emitters. These concentration changes are then automatically communicated to other agents resident at the node as a result of their registration for notification of chemical

concentration changes. Once the agent has performed its task on a particular node; e.g., measurement of quality of service of a connection or simply sensing the concentration of a specific chemical, the MDF is invoked in order to determine the node to migrate to in the network. No fixed residency time is assumed; some agents will remain at a node for long periods of time, others will not.

3.5.10 Agent Design

Section 2.2.4 on page 33 in the previous chapter described the Mobile Code Toolkit developed here at Carleton University within the Perpetuum Mobile Procura group. It is a natural candidate as a framework for design and implementation of the agent architecture described earlier. The principal Java data structures introduced in support of this design are described in Appendix B.

In this section we describe the main elements of the design and how the agent-environment coupling works. Figure 29 shows the interactions of the main elements of the design. This figure does not include short and long term memory for reasons of clarity. However, it should be understood that all elements of the architecture interact with these elements. Short term memory stores those chemicals that are shared between the local environment and the agent; i.e., cross the boundary using Emitters. Long term memory stores those chemicals that are considered internal to the agent and other information which is considered essential to the operation of the agent. The boundary of the agent shown in Figure 29 logically represents the cell membrane; in terms of implementation this is provided by the encapsulation mechanism present in the Java programming

language. The Emitter and Receptor blocks shown crossing the cell membrane in Figure 29 represent the coupling to the environment which is implemented as a shared communication medium -- a tuple space. The Receptors are implemented using the Observer Pattern, and are observers of the Environment. The Emitters are also observers of the Chemistry of the agent and are notified of changes in the concentrations of chemicals that are to be communicated to the Environment.

The Receptor Decision Function (RDF) associated with a Receptor is invoked whenever the Receptor is notified of a change in chemical concentration. The RDF may reason with the absolute concentration value or the change since the last notification. The actions associated with an RDF allow for the interaction with a Virtual Managed Component

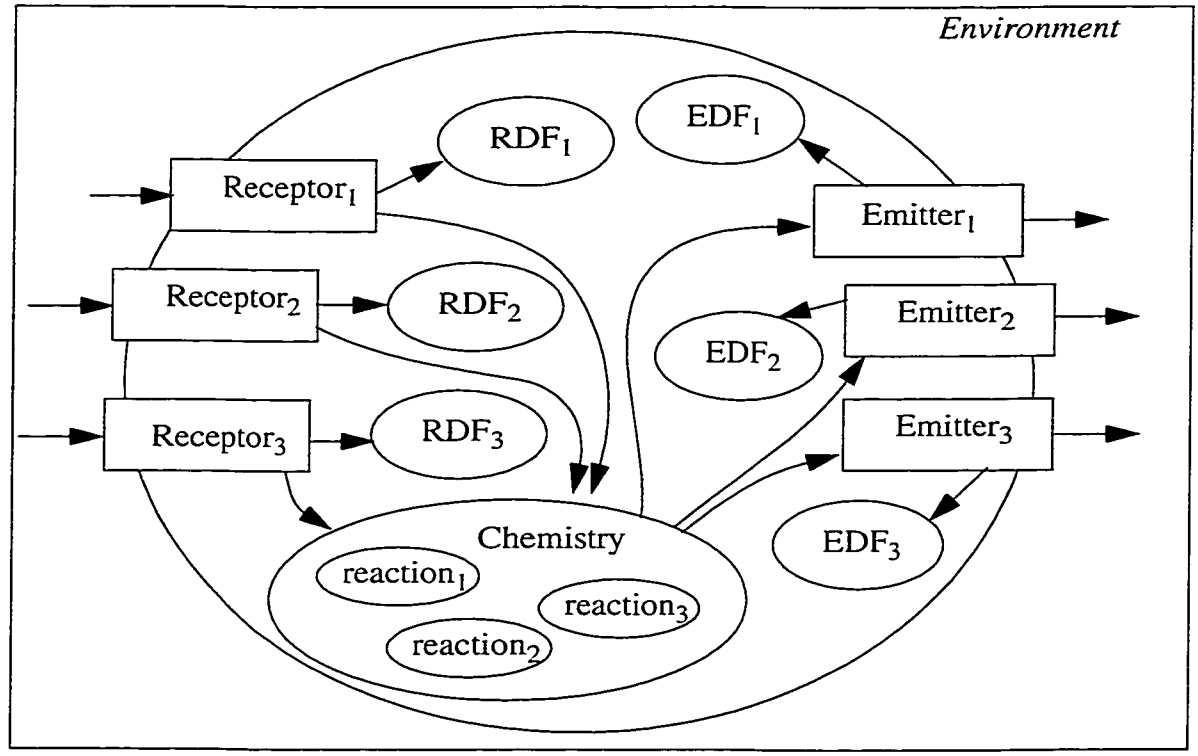


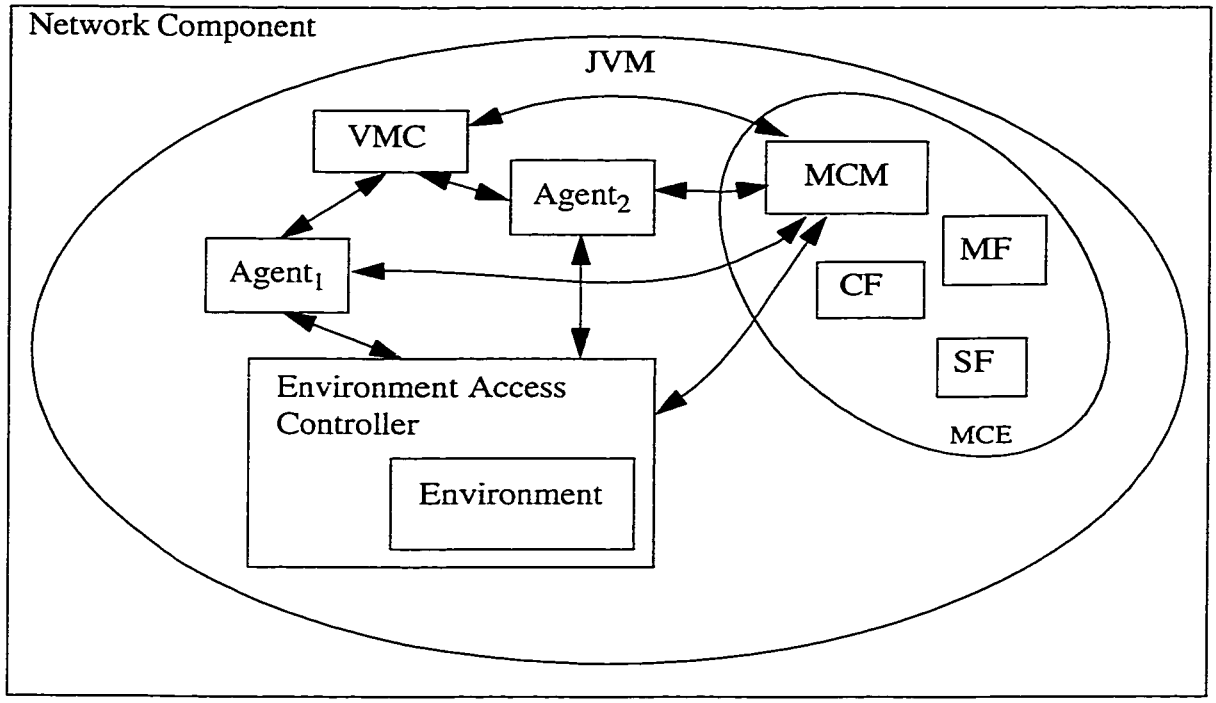
FIGURE 29. **Agent Architecture and Interactions**

(VMC) resident of the component. The VMC provides access to the underlying component's resources as described in Section 2.2.4 on page 33. These interactions allow for state changes on the network component.

The Chemistry processor provides access to and control of chemical reactions that are active within the agent. One of the main functions of the Chemistry entity is the calculation of changes to the concentrations of chemicals involved in reactions. In an analogue world these changes would be monitored continuously; however, we approximate this process digitally by having the Chemistry processor signal concentration changes to registered Emitters after a processor clock tick. The Chemistry processor also communicates chemical concentration changes to the Emitter, which, in turn, communicates them to the Environment via the Environment Access Controller.

Once signalled with a chemical concentration change, an Emitter first communicates the concentration change to the Environment. Upon completion, the Emitter then invokes its associated Emitter Decision Function (EDF). The actions associated with an EDF may cause interactions to occur with a resident VMC and analogous state changes to take place as with an RDF.

Agents communicate with the Environment through an Environment Access Controller (EAC), the EAC being implemented as a type of VMC. This mediation allows chemical concentration changes to be aggregated over an Environment clock tick. The reason for doing the aggregation is the same as for the Chemistry Processor. The VMC shown in Figure 30 is intended to show that agents may also interact with traditional data sources,



Legend.

JVM	Java Virtual Machine
MCE	Mobile Code Environment
VMC	Virtual Managed Component
MCM	Mobile Code Manager
CF	Communication Facilitator
MF	Migration Facilitator
SF	Security Facilitator

FIGURE 30. **Agent-Environment Coupling**

such as Management Information Bases (MIBs).

The interactions between the Mobile Code Manager, agents, VMC and Environmental Access Controller shown in Figure 30 indicate the central role played by the MCM in the Mobile Code Environment. For example, the MCM is used as a local naming service for

finding the EAC and the network component VMC. The Environment of Figure 29 is represented by the Environmental Access Controller shown in Figure 30, which, in turn manipulates the underlying Environment.

3.5.11 Agent Lifecycle

In this section we define the lifecycle of an agent from the point of view of arrival at a node, interaction with the local environment and subsequent migration. A simplified lifecycle for the agent is shown in Figure 31. The agent arrives at a node using the MCT code transport mechanism. This registers the agent with the agent framework locally. The callback onInit() or onRestore() is then invoked in order to allow the agent to initialize or to restore persistent state after migration respectively. The local Environmental Access Controller (a virtual managed component) is then located by querying the Mobile Code Manager and the agent registers its interest in changes of specific chemicals. These activities are performed using a pool of worker threads defined within the Mobile Code Framework. The callback onStart() is then invoked, this being the main control loop for the agent in which reactive processing takes place. A separate thread processes this method and two further agent threads are created (the Chemical Communication threads) that handle communication between the agent and the local chemical environment. The onStart() method creates the Chemical Processor which is responsible for the processing of chemical reactions within the agent. When a chemical change occurs within the agent, the agent's onOutChange(Chemical) method is invoked using the output Chemical Communication thread. The onOutChange(Chemical) method acts as a dispatcher, mapping the Chemical received to the appropriate Emitter Decision Function.

When a chemical change in the environment occurs that is of interest to the agent, the agent's onInChange(Chemical) method is invoked asynchronously. The processing of the change occurs within the input Chemical Communication thread. This onInChange(Chemical) method acts as a dispatcher, mapping the Chemical received to the appropriate Receptor Decision Function.

Once local processing is complete, the agent will choose to invoke the migrate primitive from within either an Emitter Decision Function or a Receptor Decision Function. When this occurs, the migrationDecisionFunction() method is invoked, and the next destination computed. Once computed, the onMigration(Location) callback is invoked in order to allow for local housekeeping to occur on the node. The agent is then migrated. If

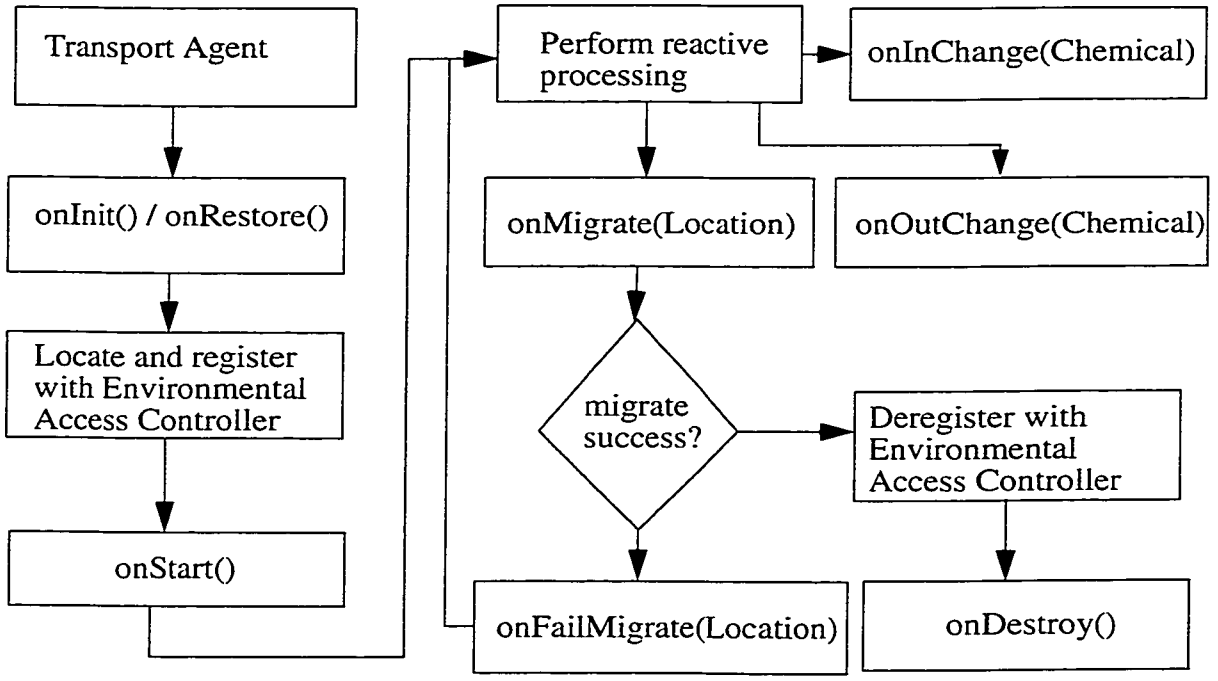


FIGURE 31. **Agent Lifecycle**

successful, the local copy of the agent deregisters with the Environment Access Controller and the callback onDestroy() is invoked before all agent threads are destroyed and the Mobile Code Manager removes all references to the agent. If migration fails, the agent's onFailMigrate(Location) callback is invoked and processing continues on the current node.

3.6 SynthECA Scenario

As an example of a multi-swarm interacting system moving on a network using the architecture described in this chapter we have chosen to investigate route finding, maintenance and fault detection in a communications network. In our environment we have a *completely* distributed view of the network. Such a view is *highly* desirable as it makes management of these networks easier and scalable.

In our system, drawn from the domain of transmission networks, we are attempting to create connections between nodes in the network, monitor them for quality of service degradation and diagnose the location of faults when they occur. It is assumed, and this can be the case, that a network manager does not exist and so no global view of the network is maintained. Consequently, a distributed route finding solution as represented by the Ant Search class of algorithms is a good candidate for route finding.

To date, three applications of the ant metaphor in the domain of routing have been documented [White 98a] (used in this thesis), [Schoonderwoerd 97] and [Di Caro 97]. The work reported in [Schoonderwoerd 97] embraces routing in the circuit switched networks while [Di Caro 97] deals with packet switched networks. Both [Schoonderwoerd 97] and

[Di Caro 97] propose the control plane as the domain in which their systems would most likely operate. Di Caro and Dorigo [Di Caro 97], in particular, provide compelling experimental evidence, based upon simulation, as to the utility of AntNet in the network routing problem domain by comparing ant-based routing with the current and proposed routing schemes used in NSFNET. The scenario described here is somewhat different and applies to a management context such as is found in a Synchronous Optical Network (SONET) transmission network. A routing table is maintained in all of the aforementioned work, a rather conventional and limiting view from the perspective of the ideas proposed here.

For the context of this thesis, we are interested in forming a connection between a source and one or more destinations for the purpose of creating a link in a logical network. It should be noted that this path may be protected, i.e. two node and link diverse paths may exist between a given source and destination. This (possibly protected) path, in turn, can be used as a resource, a link, in the next logical layer.

3.6.1 Agent Classes

In our system we have three agent classes related to route finding, one class concerned with connection monitoring and one class that has the function of detecting network poor quality of service conditions. These will now be described in terms of the (E,R,C,MDF,m) formalism introduced earlier in this chapter.

The three agent classes related to route finding are explorers, allocators and deallocators. The function of an explorer agent is to find a path from a given source to a specific

destination. The metaphor used to describe the behavior of explorer agents is that of ants foraging for food. Explorer ants possess a single emitter (e) and three receptors (r_1, r_2, r_3). The emitter and receptor r_1 are both tuned to a single chemical or pheromone (T). The receptor r_2 is used to measure the costs of links in the network (C). The receptor r_3 is used to detect the perceived quality or reliability associated with links in the network (Q).

The explorer agent has two distinct modes of operation. When moving towards the requested destination, the emitter is turned off and the receptors are used to detect the connection-specific chemical and link costs respectively. The agent's memory is used to store the links traversed by the agent. When moving back towards the source node having reached the required destination, the receptors are turned off and the emitter is turned on. A single chemical reaction (c) is defined for the explorer agent. This reaction allows for the generation of the pheromone used to reinforce an emerging path. The MDF used by the agent is defined by a series of equations that specify the probability with which a given link will be used for agent migration. Consider a swarm of explorer agents represented by the set {k}. The probability with which an explorer agent (k) chooses a node j to move to when currently at the i^{th} node at time t is given by:

$$p_{ijk}(t) = [S_{ijr}(t)]^{\alpha_{kr}}[C_{ij}]^{-\beta}[R_{ijk}(t)]^{-\delta} / N_{ik}$$

$$N_{ik}=\Sigma_{j\,\varepsilon(L(i)-Tabu_k)}[S_{ijr}(t)]^{\alpha_{kr}}[C_{ij}]^{-\beta}[R_{ijk}(t)]^{-\delta}$$

where α_{kr}, β and γ are control constants and determine the sensitivity of the search to

pheromone concentration, link cost and component quality respectively. N_{ik} is simply a normalization factor that allows p_{ijk} (t) to be interpreted as a probability. L(i) is the set of integers, {n} such that there exists a link between the i^{th} and n^{th} nodes. S_{ijr}(t) is the quantity of pheromone, s, present on the link between the i^{th} and j^{th} nodes for the r^{th} chemical at time t. C_{ij} is the cost associated with the link between the i^{th} and j^{th} nodes. R_{ijk} (t) is the quality or reliability measure associated with the link and the j^{th} node for the k^{th} agent at time t. $Tabu_k$ is the set of links already traversed by the k^{th} agent. The expression (L(i) - $Tabu_k$) represents the indices of the set of nodes not yet visited by the k^{th} agent. The C function is meant to represent the cost to the user for consuming bandwidth on a given link while the R function represents the confidence that we have in the various components involved in the connection being able to transport data effectively.

When explorer agents return to the source node, a decision is taken as to whether a path has emerged. Essentially, if a given percentage of the last n agents have followed a single path then path emergence is considered to have occurred. Other algorithms were also considered and these are presented in the next chapter. Once emerged, an allocator agent is sent into the network in order to create the connection; i.e., allocate bandwidth and associate a set of path elements with the route to be used.

The allocator agent has no emitters or receptors. It has a simple memory that stores the route that has emerged. An allocator operates in two modes: forward and backward. The

allocator agent has a simple MDF that simply pops the first entry from the list of links used in the route that is stored in memory. The agent allocates resources for the connection at each node in forward mode. In backward mode the allocator performs no action at each node.

A deallocator agent has an identical (E, R, C, MDF, m) description to that of an allocator agent. The only difference between the two agent types is the action performed at each node when in forward mode. A deallocator agent releases resources in forward mode in contrast to the action performed by the allocator agent and is sent when confidence in the existing route falls below a given threshold, the connection is no longer required or the route is no longer viable. This might be due to component failure, for example.

Further details regarding the connection allocation algorithm, explorer, allocator and deallocator agent behaviors will be provided in the next chapter.

Evaporator agents also circulate within the network. The function of these agents is to evaporate chemical concentrations relating to connection finding. They are equipped with a single receptor capable of sensing all connection-related chemicals. They implement a type one reaction in order to effect chemical evaporation. Evaporator agents are required in order to ensure that we do not "greedily" choose the first path found but allow a balance of "exploration and exploitation" to occur in route finding. Evaporation agents are examples of agents where the "don't care" symbol is used in the emitter/receptor description. Evaporator agents have an MDF that allows them to cycle through all nodes in the network in a periodic fashion. Static evaporator agents were also considered.

Explorer agents continue to search for better routes through the network even after a connection has been set up. Also, once set up, the end-to-end quality of service for the connection is monitored from the source node. When significant changes in quality of service are observed, a monitoring agent is sent out into the network in order to modify the Q values for the components used in the connection.

Monitoring agents have either one receptor (r_1) or one emitter (e_1). If the quality of service of the connection has increased, the monitoring agent with a receptor is sent. If the quality of service of the connection has decreased, an agent with an emitter is sent. The monitoring agent's emitter generates the "quality of service" chemical, q, using a single chemical reaction in situations where quality of service has decreased and evaporates existing concentrations of q chemical when quality of service has significantly improved. The receptor senses the q chemical. The monitoring agent has a simple MDF that simply pops the first entry from the list of links used in the route that is stored in memory.

The final agent type in the current system design is the fault location agent. Fault location agents circulate through the network and monitor the q chemical concentrations on nodes and links, denoted by $Q_{ij}(t)$ in the equation below. Fault location agents have a single receptor (r_1) and no emitters. They do not have an associated chemistry; i.e., they are merely observers of network state. The MDF associated with fault detection agents is probabilistic in nature and is given by the equation:

$p_{ij}(t) = Q_{ij}(t) / \Sigma_k Q_{ik}(t)$ for f% of the time and random otherwise.

Random migrations are made for (100-f)% of the time in order to ensure that the entire

network is reached in reasonable time. A probabilistic choice, based upon Q values, is made for f% of the time in order to revisit parts of the network that are experiencing poor quality of service. It should also be noted that oscillation between two high Q components is explicitly prevented; i.e., a fault location agent cannot return to a previously visited network element for t migrations. This list of tabu elements is stored in agent memory in a similiar fashion to that of explorer agents.

The function of a fault location agent is to observe components with high Q values. When the observed Q value exceeds a threshold value, the agent initiates diagnostic activity by executing rules associated with r_I.

3.7 Implementation

A Smalltalk simulation has been built for the scenario described in the previous section. This simulation has been used to investigate the interaction of the many parameters that characterize the system; e.g., reaction rates, number of agents, agent generation frequency and several others. A brief description of the Simulation and its user interface can be found in Appendix A.

3.7.1 Results and Discussion

The author's research [White 98a], [White 98c] provides experimental evidence that the basic system described can effectively compute routes and plan connections in a network. While only simulated results are available, the system has demonstrated that routing solutions to the point-to-point, point-to-multi-point and protected path problems (a problem equivalent to the shortest cycle) for a variety of graph topologies can be

effectively computed. The results in [White 98a] indicate that 15% fewer blocked connections are typically observed when comparing standard shortest path routing to the ant-like agent routing described.

Some care has to be taken in the choice of system parameters. Our research is ongoing in the area of self-adaptation of system parameters; e.g., chemical and link cost sensitivities and results for this activity are reported in [White 98c]. We are investigating the sensitivity of our swarm systems to the number of agents engaged in problem solving as well as considering the effects of noise; i.e., unreliable agent knowledge.

As connection quality of service changes, connections are dropped and new routes quickly found with traffic rapidly moving away from regions of the network that have proven unreliable. Further, the fault location agent, detecting q chemicals concentrations from multiple connection monitoring agents, quickly identified the faulty components within the network. A simple diagnostic example is shown in Figure 32.

In Figure 32, two connections are defined, one from A to B and another from C to D. Both connections experience poor quality of service and use monitoring agents to drop q-chemical in the network. The numeric labels on the nodes and edges represent the concentrations of q-chemical for that component. As can be seen in Figure 32, node E sees twice the q-chemical as it is the common element for the two paths. The fault detection agent therefore initiates diagnostic activity on this component. Further details on the use of this architecture and its implementation for diagnosis can be found in [White 98b], [White 99b] and in chapter 5.

Hill climbing in the space of q chemical as the primary strategy for fault localization leads the fault detection agent to 'zero in' on faulty components far more quickly than random search. Consider the situation shown in Figure 32. Imagine that there is a single fault detection agent present at A. If we allow migration based only on q-chemical concentration, the agent is forced to move to E; i.e. the detection agent reaches the faulty node in a single move. Consider random migration as an alternative policy. Using this policy, the agent has only a $1/3^{rd}$ probability of reaching E directly and has a far higher expected number of moves before it reaches the faulty component. We have experimented with a number of small graphs (10-20 nodes) with a variety of connection patterns and have found the hill climbing strategy to take less than $1/4^{th}$ of the number of moves that a random strategy would require.

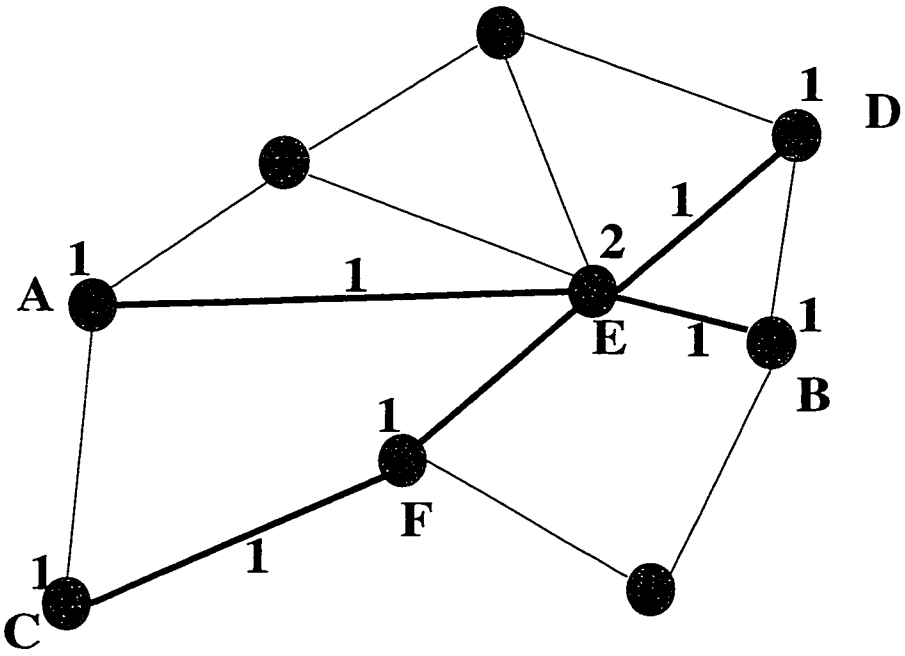


FIGURE 32. **Example Fault Localization**

While the simple example described above demonstrates the utility of chemical interference – in this case constructive – and a hill climbing migration strategy, it should be noted that a simple detection mechanism based upon a concentration threshold leads to false diagnoses. We have investigated the use of Reinforcement Learning and Discrete Learning Automata as a means of learning the correct diagnosis for a network state. In these learning systems, a vector of q-chemical concentrations on the node and its links represents state. Referring to Figure 32 once again, the state vector for node E would be (2,1,0,0,1,1,1), where we start with the concentration of q-chemical on the node followed by the links beginning with the link from A to E and moving in a clockwise direction. Feedback to the learning system is the result of whether corrective action resulting from diagnosis improves the state of the system; i.e. q chemical concentrations are reduced. No change in q-chemical concentration implies a misdiagnosis. Detailed descriptions of these two interacting swarms are forthcoming in the following two chapters.

3.8 Summary

This chapter has described a number of naturally occurring multi-agent systems and proposed a number of requirements for emergent problem solving by such systems. Several sections in this chapter have analyzed the important characteristics of the naturally occurring multi-agent systems and the SynthECA architecture based upon them.

This chapter has also provided a formal description of a multi-agent system that relies on Swarm Intelligence and, in particular, has proposed trail laying behavior in order to solve problems in a communications network, mapping specific roles onto SynthECA agents.

We have demonstrated how fault detection can arise as a result of trail laying behavior of simple agents and we have proposed the use of ideas from Subsumption as guiding the design of multi-swarm systems.

While the ideas presented in this chapter are conceptually appealing, considerable work remains to demonstrate the utility of the approach. This is provided in the next three chapters.

CHAPTER 4

SynthECA agents for Management and Control in Networks

4.1 Overview

This chapter describes how biologically inspired (SynthECA) agents can be used to solve control and management problems in Telecommunications. These agents, inspired by the foraging behavior of ants, exhibit the desirable characteristics of simplicity of action and interaction. The collection of agents, or swarm system, deals only with local knowledge and exhibits a form of distributed control with agent communication effected through the environment as represented and described in the previous chapter. In this chapter we explore the application of ant-like agents to the problem of routing in circuit switched telecommunication networks.

4.2 Introduction

Routing in telecommunication networks has mainly been static to date. Early efforts to introduce adaption into the routing algorithms of the Arpanet caused oscillations or required excessive distribution of statistical information. With the advent of new

technologies and, in particular, high capacity networks, much more flexibility will be required. Distance learning, video on demand, world wide information services such as WWW servers and many other services create dynamic network load which (ideally) should be taken care of by new algorithms. This is a new area of research with many economic and technical implications. The purpose of this chapter is to explore what swarm intelligence, and SynthECA agents specifically, might offer to solve these dynamic and fundamentally distributed optimization problems.

The advantages of swarm intelligence are twofold. Firstly, it offers intrinsically distributed algorithms that can use parallel computation quite easily. Secondly, these algorithms show an ability to react quickly to changing network conditions by allowing the solution to dynamically adapt itself to global changes by letting the associated swarm agents self-adapt to the associated local changes.

4.3 Motivations

Networks today have a wide range of applications running on them with applications having diverse statistical properties. This presents a significant challenge when deciding which applications should share node and link resources for the purpose of optimizing quality of service and, more generally, making best use of network capacity. Many would assume that making best use of network capacity implies load balancing; however, this is a simplistic assumption and ultimately it is user perception of the quality of service offered by the network that is important. Anyone who has used real audio, or IP telephony services on the Internet will certainly appreciate this.

The motivation for the algorithms in this chapter is that certain applications provide better quality of service with a lower bandwidth requirement when sharing the resources of the supporting network. This arises as a consequence of complementary statistical properties or "natural synergy" of the particular applications. At least two distinct approaches to the solution of this problem are possible.

An off-line, or planning solution is possible. In this approach, the set of connections to be created is known in advance and routes for them computed to optimize a fitness function. Typically, the fitness function used seeks to balance load across nodes and links in the network and may take account of constraints of the devices themselves and user routing preferences. See, for example, [Mann 95]. A hybrid approach, using local and global search, can be found in [Clark 97]. Such approaches are characterized by the requirement for a global view of the network including a need for notification of changes to the topology of the network. Methods for simulating a dynamic environment include optimizing network routing for a set of connection scenarios, or by using fuzzy input values.

On-line approaches are also possible. In traditional networks, routing protocols are often used that attempt to maintain a global view of the network. See, for example, [Tanebaum 96] or [Bertsekas 87] for discussions of Link State and Distance Vector routing. Several agent-oriented approaches have recently been proposed that rely upon Reinforcement Learning [Boyan 94] or appeal to principles drawn from Swarm Intelligence [Schoonderwoerd 97], [Di Caro 97], [Di Caro 98], [Bonabeau 98], [Heusse 98] and

[White 98a], [White 98c]. In an on-line approach, agents compute routes for connections in order to optimize their connection routing cost, where cost may represent an aggregate statistic of delay, utilization, reliability and other factors. In these latter approaches, a common characteristic can be observed. A global view of the network is not maintained, no exchange of global information is permitted and we deal only with information that can be measured locally. Also, these approaches are robust with respect to the loss of individual routing agents. These characteristics have very positive implications with respect to the robustness and scalability of the approach. Beyond the routing domain, and more generally, the appeal of swarms of biologically-inspired agents for industrial problem solving have recently been appreciated [Parunak 98]. Research into the problems and potential of multiple, interacting swarms of agents is just beginning [White 98d].

Given the above comments, SynthECA agents seem a natural choice for this problem domain.

4.3.1 Problem Description

Routing is the problem of finding paths between nodes in a communications network. In Figure 33, a "telephone call" is to be set up between two points by finding a route that connects the source and the destination. Links are able to carry application traffic up to the total capacity of the link. In this thesis, it is assumed that bandwidth is used through simple aggregation. Nodes are assumed to be capable of satisfying the switching demands placed upon them. Links have an associated cost function. This is typically based on distance (e.g., traffic delay), and/or on error rates.

A connection request is characterized by the kind of traffic (voice, data, and video) which is required. This implies bandwidth requirements and potentially other constraints such as maximum error rate, maximum number of hops, or maximum delay. Different kinds of traffic can tolerate different delays and error rates, and these can be used as parameters in a cost function, along with the link's actual costs.

Point to multi-point routing is required in applications such as distance learning. In this case, a single source node transmits and receives information from a number of destination nodes, as in the case of a teacher giving a lecture to a number of students in a distance learning application.

Multi-path routing is required in domains where data streams should not be disrupted when a single component fails in the network. In these cases, data is routed via at least two node and link distinct paths and the data streams merged at the destination[1].

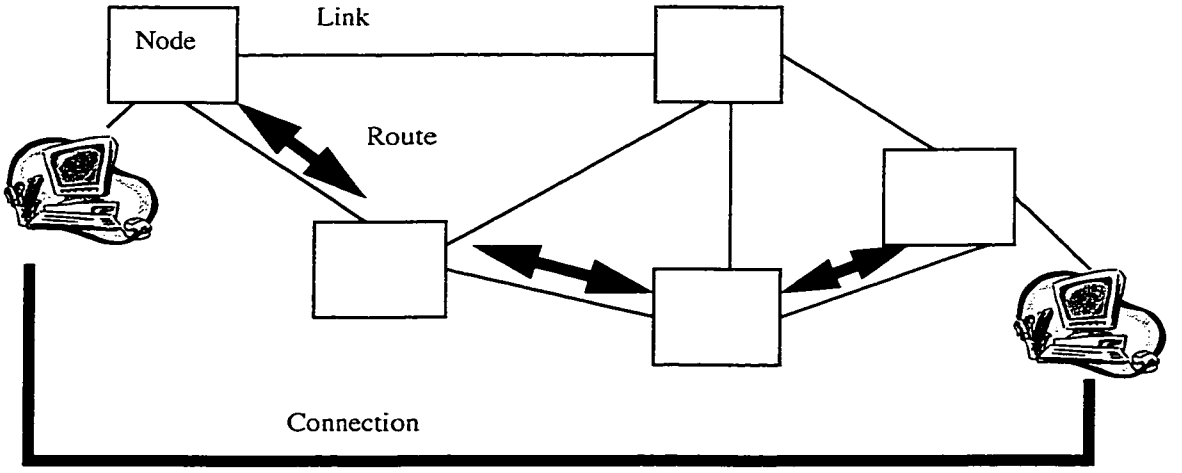


FIGURE 33. **A Network Connection**

The communications network will be idealized here as a weighted graph where the vertices correspond to switching nodes, the arcs to the physical links, and the associated weights to the cost value associated to every link. Both point to point and point to multi-point routing amount to finding a minimum spanning tree in a graph.

4.3.2 Description of the Algorithm

The swarm algorithm solution to this routing problem relies on the movements of artificial agents on the associated graph designed to make the global shortest path emerge. When a connection request is made, a new colony of SynthECA agents is created and associated Connection Creation Monitoring Agents (CCMA) are positioned on the source and destination nodes. The CCMA on the destination node is created when the first explorer agent arrives. The functions of the CCMA are to decide when a path has emerged and when the current path is no longer the shortest path and that path re-planning should occur. These artificial agents correspond to a special class of automata called reactive agents (as described in Section 2.2.2) that react to their local perception of the environment by stochastically adopting predefined behaviours. The shortest path then emerges from the movement and interaction of all these agents. This corresponds to the first algorithm that focuses only on establishing a single point-to-point connection.

There are three classes of routing-related agent, as introduced in the previous chapter. They are:

Explorer agents: these agents search for a path from a source to a destination.

---

1. This is a simplistic view of multi-path routing but will suffice for the algorithms proposed in this chapter.

Allocator agents: these agents allocate resources on the links used in a path from a source to a destination.

Deallocator agents: these agents deallocate resources on the links used in a path from a source to a destination.

As soon as we wish to establish multiple point-to-point connections, the problem becomes more constrained since the connections consume bandwidth and that after a while, some links might run out of available bandwidth. The extension is quite straightforward, the graph arcs have an associated available bandwidth and a condition is added for the agents to use a given arc: it should have enough bandwidth. Every time a path has emerged and a connection has been established the amount of available bandwidth is decreased on every arc of the path thereby adding additional bandwidth constraints to the graph.

We assume here that all nodes are capable of running an extended Mobile Code Environment as described in sections 2.2.4 and 3.5.10.

Let us now define the local rules of the artificial agent automata.

4.3.2.1 Point to Point Routing

For this case, the algorithm is quite straightforward. If the network is symmetric[1], both source and destination nodes of the connection are considered as host to a static agent called the Connection Creation Monitoring Agent (CCMA). If the network is assymmetic,

1. A network is symmetric if its link costs obey the relation, $C_{ij} = C_{ji}$.

only the source node contains a CCMA. The explorer agents are created by the CCMA and leave the node in order to explore the network following their local rules:

On each node, they choose a path using their MDF with a probability proportional to the heuristic value (function of the cost and the pheromone level) associated with the link. The details of the MDF used are shown in equations 8-11 on page 157,

The agents cannot visit a node twice (they keep a tabu list of their visited nodes in memory) and cannot use a link if there is insufficient bandwidth available. The tabu list is stored in agent memory.

Once the destination is reached, the agents return from whence they came by popping their tabu list. On their way back, they lay down a pheromone trail.

When the CCMA on the source node judges that a shortest path has emerged, it sends a special kind of agent, the allocator, in order to allocate the bandwidth on all links used between the source and the destination. A path is considered to have emerged when the majority of returning explorer agents follow it. The allocator follows a fixed path consisting of nodes and links that form the connection.

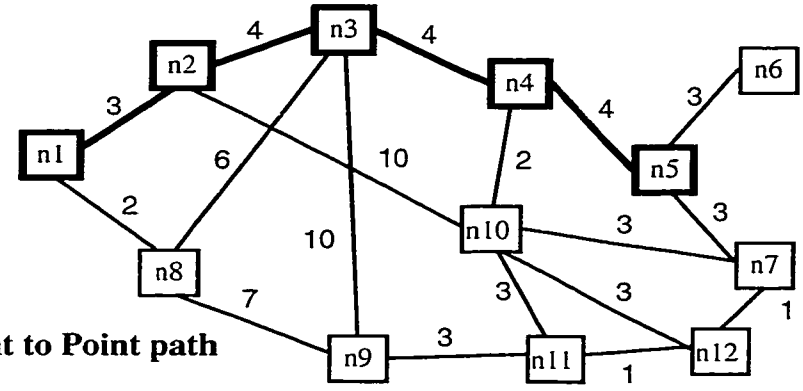


FIGURE 34. **Point to Point path**

Pheromone laid down on a link will evaporate over time. This is controlled by a constant evaporation rate, r. In reality, a SynthECA agent is deployed on each node that contains a single type one equation and this evaporates all chemicals, albeit at the same rate. An alternative solution was implemented wherein several evaporator agents circulate within the network but this solution was found to consume more network processing resources without noticeable improvement of the quality of solutions found.

4.3.2.2 Multiple Point to Point Routing

This scenario is similar to the previous one except that we have now several connections at the same time. Each connection corresponds to a different species and these species do not interact except by allocating bandwidth, and therefore, by imposing constraints on the other species by changing the environment. A species in SynthECA corresponds to a chemical encoding.

4.3.2.3 Point to Multi-point Routing

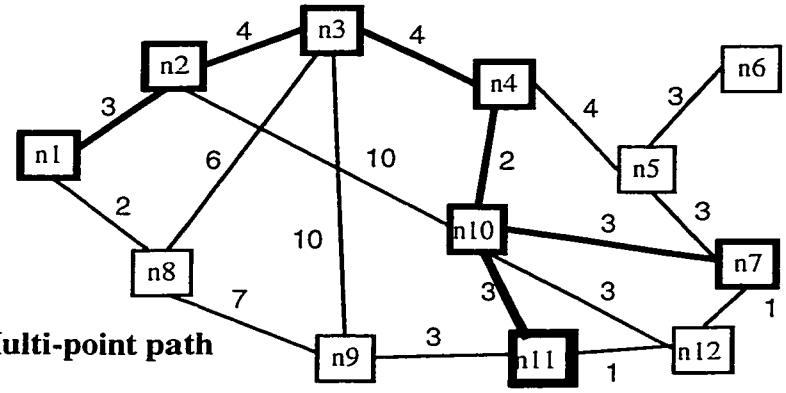


FIGURE 35. **Point to Multi-point path**

Point to multi-point connections (an example is shown above) can be regarded as multiple point to point connections starting from the same source node. The only modification to the previous point to point algorithm concerns the allocator. Rather than

sending a different allocator for each destination, identical allocators are sent from the source toward the destinations. Only the first allocator passing on the link will allocate the bandwidth, and fan out points (also known as multi-cast nodes) are created on bifurcation nodes. This is achieved by the allocator dropping an 'allocated' chemical on the node which is sensed by any allocators that follow.

An alternative explorer algorithm was considered. In the latter algorithm, the each explorer is given all of the destinations and return to the source node when one of them has been reached.

4.3.2.4 Cycle (or Multi-path) Routing

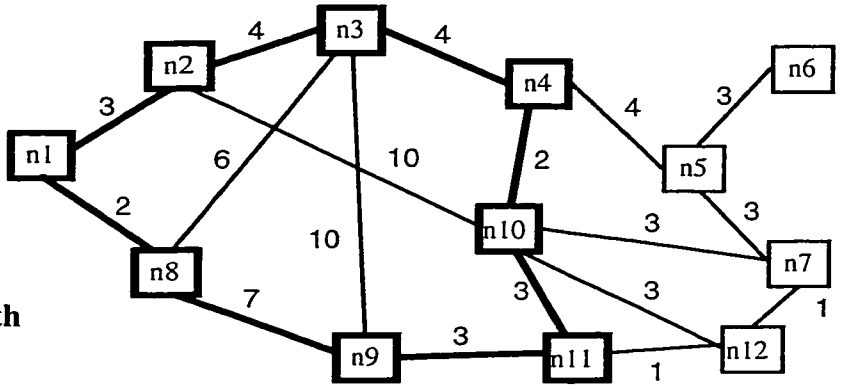


FIGURE 36. **Cyclical path**

Cycle or multi-path routing can be regarded as two node and link disjoint paths (excluding source and destination nodes) that connect a source node to a destination node. An example is shown in Figure 36. The only modification to the original algorithm for the explorer agent is that upon reaching the destination it turns around and finds a path back to the source node that does not use any of the nodes or links used in the outward journey. Paths of this type are frequently constructed for the purpose of fault tolerance, i.e., an element on one path may fail but the circuit remains viable as a vehicle for the

transportation of data.

4.3.3 Detailed explorer agent rules of behaviour

There are two phases to the movement of an explorer agent: an outward exploring mode and a backward trail-laying mode. The (simulation) algorithm used by an explorer agent for a *single* connection (where only a single chemical is used and so the third index on S is dropped) is shown below. Comments are shown in italics within curly brackets.

1. Initialize the route finding simulation do:
   Set t:= 0
   For every edge (i,j), set $S_{ij}(t) := 0$, $cr_k := 0$
   Place m agents on the source node.
   Explorer agents are created at frequency e_f
   end
2. *{Start the explorer agents from the source node}*
   Set i := 1 {tabu list index}
   For k := 1 to m do
   Place starting node, s, of the k^{th} agent in $Tabu_k[i]$.
3. *{Migrate the explorer agents one node at a time}*
   Repeat until destination reached:
   Set i := i + 1
   For k :=1 to m do
   Choose node j to move to with probability, $p_{ijk}(t)$
   Move the k^{th} agent to node j.
   Update explorer agent route cost: $cr_k = cr_k + C_{ij}(u)$
   If $cr_k > cr_{max}$ then kill the k^{th} explorer agent.
   Insert node j in $Tabu_k[i]$.
   At destination go to 4.
   end
4. While i > 1
   Move to node $Tabu_k[i]$.
   Update pheromone levels: $S_{ij}(t) := S_{ij}(t) + ph(cr_k)$
   i := i - 1
   end

5. At the source node do:
   If the path in $Tabu_k$ is the same as b% of
   paths in PathBuffer then create and send an allocator agent
   If $t > T_{max}$ then create and send an allocator agent for shortest path found
   end

In the single connection algorithm above, the following symbols are used:

$S_{ij}(t)$ is the quantity of pheromone present on the link between the i^{th} and j^{th} nodes,

$C_{ij}(u)$ is the cost associated with the link between the i^{th} and j^{th} nodes at utilization, u.

cr_k is the cost of the route for the k^{th} explorer agent.

$Tabu_k$ is the list of edges traversed.

T_{max} is the maximum time that is allowed for a path to emerge.

PathBuffer is the array of paths obtained by the (up to m) explorer agents.

cr_{max} is the maximum allowed cost of a route.

$ph(cr_k)$ is the quantity of pheromone laid by the k^{th} explorer agent.

$p_{ijk}(t)$ is the probability that the k^{th} agent will choose the edge from the i^{th} to the j^{th} node as its next hop given that it is currently located on the i^{th} node.

More generally, for multiple simultaneous connection finding, with one chemical used for each connection to be computed, the probability[1], $p_{ijk}(t)$, with which the k^{th} agent chooses to migrate from its current location, the i^{th} node, to the j^{th} node at some time, t, is given by:

$$p_{ijk}(t) = F_{ijk}(t) / N_{ijk}(t), \; R < R^* \qquad (8)$$
$$= H_{ij}(t)$$
$$N_{ijk}(t) = \Sigma_{l \text{ in } A(i)} F_{ilk}(t) \qquad (9)$$
$$F_{ijk}(t) = \Pi_r [S_{ijr}(t)]^{\alpha_{kr}} [C_{ij}(u)]^{-\beta} [R_{ij}(t)]^{-\delta} \qquad (10)$$

1. A more conventional representation of this probability might be $p_{jk|i}(t)$, indicating the conditional nature of the probability.

$$F_{ijk}(t) = \Pi_r [S_{ijr}(t)]^{\alpha_{kr}}[C_{ij}(u)]^{-\beta}[R_{ij}(t)]^{-\delta}, j = j^{max} \quad (11)$$

$$= 0 \qquad \text{otherwise}$$

where:

α_{kr}, β,δ are control parameters for the k^{th} agent and r^{th} chemicals for which the k^{th} agent has receptors, $\alpha_{kr} = 0$ if the agent does not have a receptor for the r^{th} chemical,
$N_{ijk}(t)$ is a normalization term,
A(i) is the set of available outgoing links for node i,
$C_{ij}(u)$ is the cost of the link between nodes i and j at a link utilization of u,
$R_{ij}(t)$ is the reliability of the link between nodes i and j at time t.
$S_{ijr}(t)$ is the concentration at time t of the r^{th} chemical on the link between nodes i and j,
R is a random number drawn from a uniform distribution (0,1],
R* is a number in the range (0,1],
$H_{ij}(t)$ is a function that returns 1 for a single value of j, j*, and 0 for all others at some time t, where j* is sampled randomly from a uniform distribution drawn from A(i),
$F_{ijk}(t)$ is the migration function for the k^{th} agent at time t at node i for migration to node j,
j^{max} is the link with the highest value of the product: $\Pi_r [S_{ijr}(t)]^{\alpha_{kr}}[C_{ij}(u)]^{-\beta}$.

It should be noted that equations 8-11 take into account the reliability of the link through the terms $R_{ij}(t)$. The reliability measure arises from a reliability pheromone being deposited in the network by fault detections. These agents are the subject of the next chapter and are described at length there.

The explorer agent has two modes of behaviour. If travelling towards its destination, it finds links at each node which the agent has not yet traversed, and which have enough bandwidth available for this connection. It selects a link from this set based on the probability function $p_{ijk}(t)$. Having selected a link, the selected link is added to the tabu list. The cost of the journey so far is updated and then the link to the next node is

traversed. An explorer agent that stays without being able to move for more than a given number of iterations simply dies. Similarly, an agent whose journey cost exceeds a given threshold also dies. The robustness of the algorithm to the failure of individual agents is a highly attractive feature of the algorithm.

At the destination, the explorer agent switches to trail-laying mode. When travelling back to the source node the agent pops the tabu list and moves over the link just popped dropping pheromone at a constant rate proportional to the cost of the route found.

The CCMA at the source node maintains a set of statistics relating to the set of routes that have been found so far in both point-to-point and point to multi-point connections. This is achieved by querying the returning agents (and agents which are sent from the destination node to this node) about the path which they took. This information is maintained by their tabu list. The node records the frequency of agents following a particular route over a given time period (a moving window). It also records details such as the total cost of the route. A good route is one for which a proportion of agents in the current time window exceeds a specified limit; e.g., 95%. When the limit is exceeded, the node sends out an allocator agent that creates the connection by allocating resources in the network. An alternate algorithm used buckets. That is, routes of a given cost in a certain range were all considered equivalent. When a specified limit of all routes lie within one bucket, an allocator is sent with the route chosen randomly from those within the bucket.

Once an allocator agent is dispatched, if it does not succeed in establishing the connection because, for example, another connection used all the available bandwidth, it

simply backtracks. In the meantime, explorer agents continue to explore the problem space. The CCMA at the source node may send out an allocator agent again when the path emergence criteria are satisfied, or may choose to delay sending the allocator out again until the problem space settles to a steady state.

4.3.4 Experimental Results

The section reproduces and discusses the results of an investigation into the effects of the sensitivity parameters: α and β. Our experiences using the algorithm suggest that its important parameters are the sensitivity to pheromone, α, and the sensitivity to cost, β. It is prudent to question the validity of the results based on such a small number of graphs. However, observations over many runs show that the analysis of the results represents a reasonable picture of what happens on different graphs and connections. These findings are consistent with other ant search applications [Kunz 94], [Color 92] which have also identified the need for appropriate choice of control parameters. In order to investigate the sensitivity of the algorithm to these parameters, we decided to fix the other parameters. The values used are shown in Table 1:

TABLE 1. Simulation Parameters

Parameter	Value
Agent creation frequency	Every 10 cycles
Quantity of pheromone dropped	10 units
Emergence criterion	90% follow a path
Number of agents created	15
Path buffer size	40 agents
Pheromone evaporation rate	1.0 units/cycle
Maximum search time	400 cycles

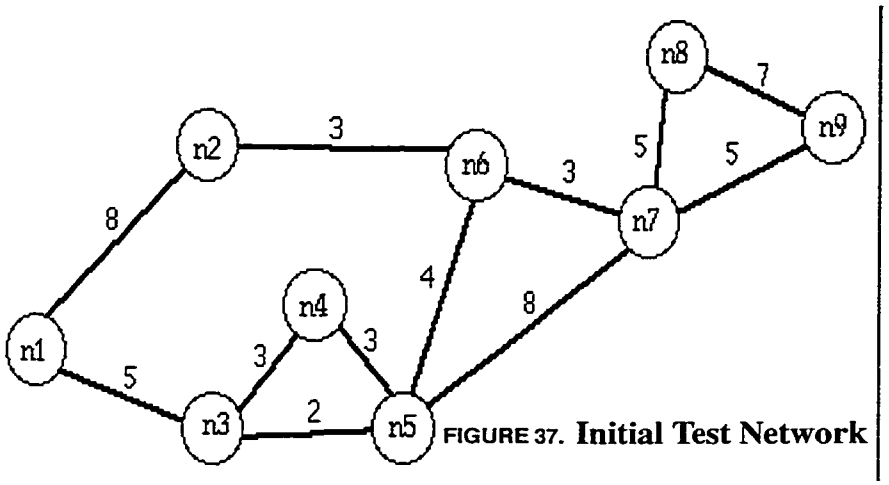


FIGURE 37. **Initial Test Network**

The graph we used and its associated weights are shown in Figure 37. The initial test network contained only nine nodes and, in this investigation, links where assumed to have sufficient capacity to form the connection. The integer values associated with the links in Figure 37 are the cost of using that link in forming the connection. The connection requested in this study was from node n1 to node n9. One hundred runs were performed in order to assess the robustness of the algorithm. In all experiments, the path that emerged was n1-n2-n6-n7-n9. This route has the lowest cost (19) and shortest path (least number of hops) although the later is an artifact of the graph selected. Results for a selection of α and β values can be seen in the table below, all measurements given as the number of agents needed to achieve path convergence The numbers in the table represent the number of cycles required to converge to a solution.

TABLE 2. **Results of Initial Experiments**

α,β	2,1	2,2	4,2	8,2
Minimum	75	75	130	130
Maximum	400	220	195	190
Mean	220	175	159	150
Std Dev.	45	25	14	10

When the algorithm starts, the cost part of the MDF is important, and essentially a greedy heuristic comes into play. The actual value of α is unimportant, as only a small amount of pheromone (close to zero) is present on the links, therefore the only factor influencing the choice of links is the actual cost on that link. The choice of next hop depends almost exclusively on link cost.

As routes are found, pheromone is laid on the links that form the path. The amount of pheromone laid is inversely proportional to the total cost of the route, and acts as a global measure of its 'goodness'. This brings the reinforcement part of the probability function into play. The sensitivity to pheromone, α, influences the choice links, and links with more pheromone are more likely to be chosen.

The first result set shows a broad spread of times to find solutions, with a very high standard deviation. The mean is also the highest of the set. The high standard deviation means that confidence in a path is not high, and the system continues to use the greedy heuristic to explore other paths. In this particular case, we consider the standard deviation too high. The second and third set of results show similar results for minimum, maximum and mean, but with standard deviation being the most different. The system finds results quickly, and reinforces good solutions, but confidence in the solutions found is at a level such that other solutions are not rejected, but continue to be explored. These seem to be the best sets of results. The final set of results show a fast, almost deterministic algorithm. This makes it undesirable for dynamic routing or as a candidate for a system that reacts to change quickly.

It is important that other solutions are explored and evaluated, as this is a stochastic approach. Reducing the standard deviation shows that the algorithm is more deterministic, which is not desirable. On the other hand, too high a standard deviation is also undesirable, as good solutions are not sufficiently reinforced.

Observations and analytical analysis show that with high values of α, the system becomes 'locked into' the solution found first, which in this experiment appears to be the best solution. This leads to problems later when bandwidth is removed from a link on that path. The system reorganizes, but when the bandwidth is reinstated, the previous solution, which is typically the best solution, is not found as the high sensitivity to pheromone is forcing the choice of paths where ants have previously been. ***However, it is important to note that with all values*** α ***and*** β ***routes are found.***

We conclude that suitable values for α and β are: (α=2, β=2) or (α=4, β=2).

These combinations ensure that a solution is found quickly, good solutions are reinforced, and that better solutions still have a reasonable chance of emerging. In order to add support for this choice, a meta GA[1] was run. The meta GA encoded values of α and β in the range 0-8 using increments of 0.0125. Connections in the network of Figure 37 were computed for the source-destination pairs: (n1, n9), (n2, n9), (n3, n8) and (n5, n8). The meta GA computed each pair 100 times, using as fitness the mean number of agents required to compute the pair plus two times the standard deviation of that value. A population size of 20 was used and 10 generations were run. The resulting fittest (α,β) pair

1. For more details on the concept of a meta Genetic Algorithm consult [Goldberg 89], for example.

was (4.25, 2.5). While not conclusive, these values are similiar to those used in other Ant System research. The sensitivity to these parameter settings also seemed similiar to Dorigo's AS system.

4.4 Further Enhancements to the Basic Algorithm

4.4.1 Introduction

This section discusses some of the finer details of the algorithm, and specific enhancements which improve the behaviour of the algorithm in a wide variety of circumstances.

As stated previously, there are three kinds of agents: Explorers, Allocators and Deallocators. Explorers explore the graph, selecting their movement according to an MDF described earlier. After some time, large proportions of the agents are likely to follow the same path. The criterion used to determine if a solution has emerged is quite simple: more than 90% of the agents have to follow the same path. This path is the solution given by the algorithm. It was found that in a network with several paths of equal cost (or nearly equal) the algorithm could oscillate between solutions, causing path emergence times to lengthen considerably. In order to solve this, the bucket method was employed, i.e., paths within a certain range were considered equivalent and when the bucket contents represented the convergence threshold, an allocator agent was dispatched to establish the connection by reserving the required amount of bandwidth on each of the links of the solution chosen randomly from those within the bucket.

The connection is deemed to have been established when the allocator returns. However,

the allocator agent may fail to allocate resources, as bandwidth may be consumed by other connections during the time the last explorer returns to the CCMA and the decision to send an allocator is made. In this case, the allocator agent backtracks along the already-allocated path fragment, releasing resources until it arrives back at the source node. The CCMA then freezes the allocation decision ("backs off") for a period of time, continuing to send explorer agents into the network.

4.4.1.1 How agents choose the next link

The previous sections demonstrated that the desired agent behaviour could be achieved using a probabilistic function to govern the agents' movement. For a particular explorer located at a node in the graph, the probability of choosing a link (assuming enough bandwidth is available on that link) is proportional to the amount of pheromone on it. In addition, the 'cost' of the link is taken into account. For each link, the probability of selection is proportional to the amount of pheromone raised to the pheromone sensitivity, α, and multiplied by the cost of that link raised to the cost sensitivity, β.

Experiments were undertaken with $\beta = 0$ and with $\alpha = 0$. In the former case, the time to converge on the shortest path was considerably higher, often by a factor of 5. The sensitivity to cost allows the agents to find better solutions in the early stages of the search (this is the so-called 'greedy heuristic'). With $\alpha = 0$, essentially the agents do not collaborate and gain nothing from the changing pheromone levels in the environment. So, when acting alone, the algorithm does not benefit at all from being population based and only finds shortest paths by chance (if at all) depending upon the local graph topology.

When α is non-zero, the agents form an interacting population and through a feedback mechanism (the pheromone trail), the search effort is distributed among the agents which tends to avoid local optima. The use of both cost and pheromone sensitivity greatly improves the results of the algorithm.

All the links connected to the current node are first considered. Then, the links already visited (i.e., in the *tabuList*) are removed. This *tabuList* is updated each time the agent moves: its previous position is added to the *tabuList*. This way, an agent can not go twice on the same position. We use this to avoid cycles in the exploration. Then, the links without sufficient available bandwidth for the connection are removed from the search. Acknowledging the bandwidth constraint during explorer search significantly decreased path emergence times when compared to an algorithm where the constraint was applied back at the source node by the CCMA.

4.4.1.2 Pheromone

The way the pheromone is dropped on a graph is different than with real ants. Some species of ant drop pheromone both to and from a food source. However, explorer agents only drop pheromone on their way back to the nest, in backward mode.

Contrary to real ants, the aim is now to find the path with the lowest cost and not the shortest path. The amount of pheromone depends on the cost of the way they have found, and is inversely proportional to the cost of the path. In this way, a cheaper path will receive a larger amount of pheromone and will hence attract more agents. This is, in effect, a positive feedback loop.

Search using positive feedback would lead to run-away situations unless there were some other control mechanism. The mechanism used in this approach is pheromone evaporation. This way, previous solutions that have become out of date can be 'forgotten'. If this were not the case, the algorithm would become locked onto an early solution (local optimum). Imagine that one path was discovered at an early stage but that, now, fewer and fewer agents use it. The evaporation process will ensure that the pheromone disappears if pheromone is not dropped quickly enough, thereby ensuring the dominance of the emerging path.

4.4.1.3 Source and Destination Nodes

For a symmetric graph, half the agent population begins at the source node and search for the destination node; for the other half, the opposite is the case. Formally, the first agent reaching the destination node initiates the reverse search. Therefore, agents search from both source and destination nodes. In this way local optima due to the structure of the graph is minimized.

When explorer agents are created, they start at their creation node. They look at all the links connected to that node, and choose the next based on the probability function described earlier. The agent moves from the current position to the next position, if there is one, or else commits suicide (as there is nowhere else for it to go). The conditions for suicide are either insufficient bandwidth or all remaining links have already been traversed.

4.4.1.4 Point to Multi-point Connections

Such connections have to be created when you want to send the same message from the same source point to many destination nodes. If you create *n* point to point connections between the source node and the destination nodes, on some links you will have a high redundancy: you can repeat up to *n* times the same message and you will use *n* times the same amount of bandwidth. To avoid such a redundancy, you can allocate the amount of bandwidth for one message once on the common part of the path, and then duplicate the message when the different paths diverge. This provides an economy of bandwidth. The form of routing is equivalent to finding a minimum spanning tree of a graph.

In order to achieve this, each connection needs to know its associated connections (i.e., the other limbs of the point to multi-point connection). This is very important for the allocator, which must ensure that the allocator agents created by the other connections do not allocate bandwidth if others in that spanning tree have already done so.

4.4.1.5 Agent Species

In order to reuse what the previous connection explorer agents have done, a species id was given to each connection. This species id was determined by the source and destination node addresses. Therefore, two connections between the same points having receive the same species id.

In that way, connections with the same species id will use the same pheromone. For new connections, there may already be pheromone between the source and destination in the graph. The agents do not have to search from the beginning, as at least one solution to the search is already present. This modification can save considerable search time.

4.4.1.6 Load Balancing

The aim of load balancing is to find a route between the source and the destination, while trying to have a homogenous partition of the traffic. The aim is to avoid having empty links while others are full. The problem is then to favour a path with low occupancy. The first thing is to drop more pheromone on empty links than on full ones. We modified the algorithm by multiplying the real cost of a link by a Cost Function. This function is based on the occupancy of the link (used bandwidth / total bandwidth).

Four cost functions were mainly used during all of these experiments. The first of them is the *defaultCostFunction*. This is a constant function and it is shown in Figure 38d. This function represents the cost of a simple routing algorithm without a special weight given to the occupancy of the link. This cost function is equivalent to the standard behaviour of the algorithm described previously.

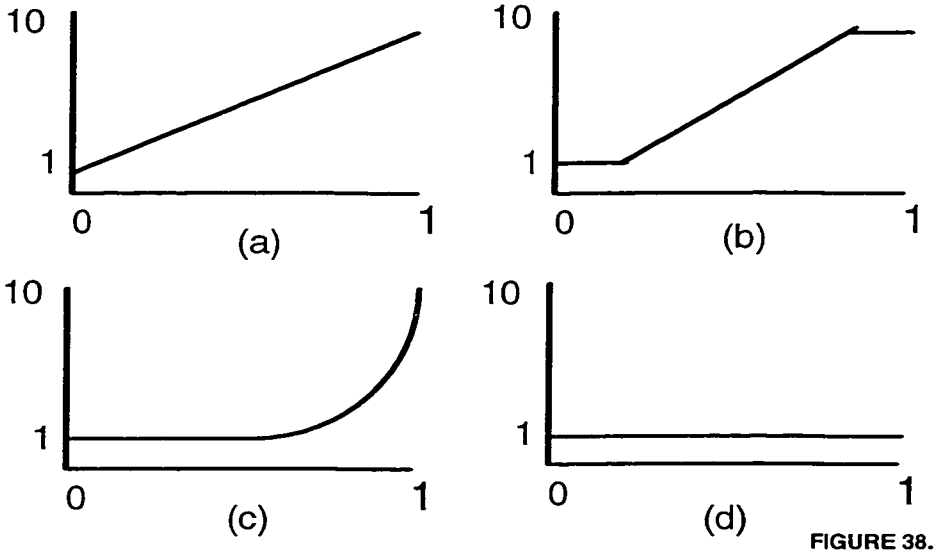


FIGURE 38. **Link Cost Functions**

The cost functions we investigated are shown in Figure 38a, 38b and 38c share two characteristics. They return one if the link occupancy equals zero and they return 10 if the

link occupancy equals one.

This normalization seems to compare the effects of each cost function for the same conditions. We have only chosen continuous functions.

The mathematical functions used for Figure 38a, 38b and 38c are given by the equations in Figure 39. The mathematical function for Figure 38d is simply the constant 1. The results of using this modification are presented in a later section. The mathematical form of Figure 39c requires some comment in that it closely resembles the response time characteristic associated with the closed form M/M/1 queuing result. Hereafter, we will refer to the functions in Figure 38a, 38b and 38c as *linear*, *non-linear* and *quartic* respectively.

$$\forall x \in [0, 1], f(x) = 9x + 1 \qquad (12)$$

$$0.25 < x < 0.75, f(x) = \left(18x - \frac{14}{4}\right) \qquad (13)$$

$$\forall x \in [0, 1], f(x) = x^3(x + 8) + 1 \qquad (14)$$

FIGURE 39. **Mathematical Cost Functions**

4.4.2 The Genetic Algorithm-like approach

In certain cases, the absolute best path (we call it path #1) is not available due to very heavy traffic allocation on it. The algorithm therefore finds another path (path #2). During the time needed to find path #2, the agents have dropped a large amount of pheromone on the trail. Due to the pheromone sensitivity of the agents, this path is more likely to be followed. Sometimes, the agents can be attracted by that trail in such a way that they will not explore other links any more. The problem is that, if certain changes happen in the

graph, for instance path #1 becoming available, the algorithm will not be aware of them because its agents will not explore new links.

To avoid to be locked into such a local optimum, the pheromone sensitivity has to be reduced at that moment. With lower pheromone sensitivity, the agents are more likely to explore other links. The problem is when to decide that the sensitivity has to be lowered and what new pheromone sensitivity to give to the agents.

Each explorer agent encodes its α and β sensitivity values that are used in the calculation of $p_{ijk}(t)$. Initially, these values are selected randomly from a given range. Hence, the population of m agents initially sent out into the network has a range of sensitivity values. When these agents return to the source node, having found a route to a given destination, the route cost, cr_k, is used to update the fitness value, $f(\alpha,\beta,k)$, associated with the (α,β) pair. The equation used to update $f(\alpha,\beta,k)$ is given by:

$$f_{new}(\alpha,\beta,k) := f_{old}(\alpha,\beta,k) + \gamma(cr_k - f_{old}(\alpha,\beta,k)),$$

where $0 < \gamma < 1$.

It can easily be seen that an agent returning with a lower route cost than the current $f(\alpha,\beta,k)$ will cause $f(\alpha,\beta,k)$ to decrease. However, several agents must return with the same cr_k value before $f(\alpha,\beta,k)$ approaches cr_k. A discrete space for (α,β) was chosen in order to ensure that updates to $f(\alpha,\beta,k)$ would occur. A discounted feedback mechanism as shown above is required in this system because of the stochastic nature of the search. The same (α,β) encoding may result in several different cr_k values and $f(\alpha,\beta,k)$ represents the

average over all possible routes found in the network. Obviously, with γ set to zero it is possible to ignore previous searches with a given encoding.

As stated earlier, the source node retains a path buffer that contains m paths. The source node also retains m (α,β) pairs and their associated fitness values. When new agents need to be created and sent out to explore the network, the fitness values are used to create new (α,β) pairs. First, parent (α,β) encodings are selected based upon their f(α,β,k) values. The lower the value of f(α,β,k), the more likely the (α,β) encoding is to be chosen. A new encoding is then created by single point crossover and mutation operators using a mechanism well known in Genetic Algorithms (GAs) [Goldberg 89]. Example crossover and mutation operators are shown in Figure 40.

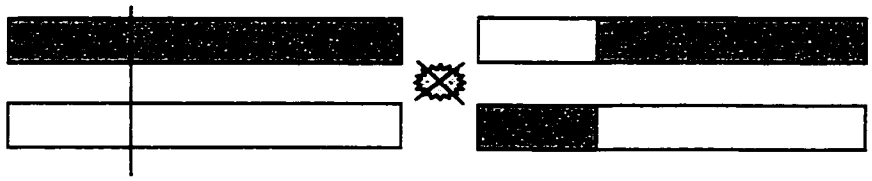


FIGURE 40. **An example of Crossover**

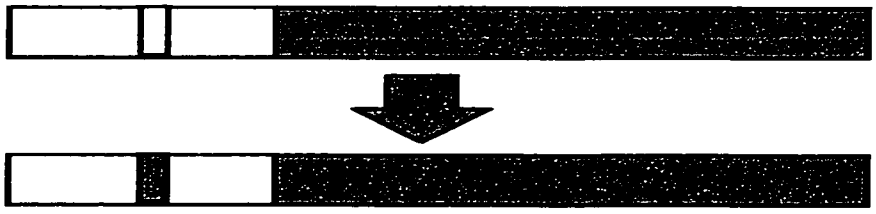


FIGURE 40. **An example of Mutation**

The figure above shows the genotype of two agent parents that encode α and β. A single crossover point is chosen (more are possible) and two offspring are generated. One offspring is discarded and the other undergoes mutation as shown above.

The way we have achieved this is with a Genetic Algorithm-like process. As stated above, each agent has its own cost and pheromone sensitivity. At the very beginning, all the agents have random sets of parameters that are defined within a given range. Seeding the population with the (α,β) values established using the meta-GA was also performed, with slightly improved results.

When an agent returns, its set of parameters is stored along with the cost of the route found. Its parameters are linked to the cost of the path found. This cost has the same role as the fitness function of a GA. When creating a new agent, the sets of all of the last returning agents are considered. An intermediate population of parameters is created: each set has a probability of being chosen proportional to its fitness (the cost of the associated path). Some random parameters are automatically added to the population. Given that the encoding is a bit string, the spectrum of parameter values is discrete, a property which is essential for the use of the updating equation for f(α,β,k). The negative values allow agents to flee the main trail and therefore to explore new links. If these values are useful (i.e., they cause better paths to be found) they will be stored for future 'breeding', otherwise they will be forgotten. Then the genetic operators such as mutation and crossover are carried out. Finally, agents with the corresponding set of parameters are created and sent out to explore the graph.

These random elements add an extra degree of variability to parameters and this is needed to avoid the convergence of the parameters.

We have observed that the algorithm is able to adapt to a new situation much more

quickly when using these adaptive parameters. The time needed to discover the new path was about 2 or 3 seconds instead of half a minute to minutes before[1], depending of the value of the fixed pheromone sensitivity.

This approach differs from a conventional GA in that, in this algorithm, we are trying to *avoid* the convergence of the population because it tends to lead to local optima. This is perhaps closer to work on co-evolving populations because the environment of the agents (the network and its representation, the graph) is modified by their actions. There is considerable inter-play between the pheromone laying activities of one agent with the cost of a path found by another agent and, therefore, the fitness associated with the (α,β) encoding of pheromone and cost sensitivity values.

4.4.3 Experimental Setup

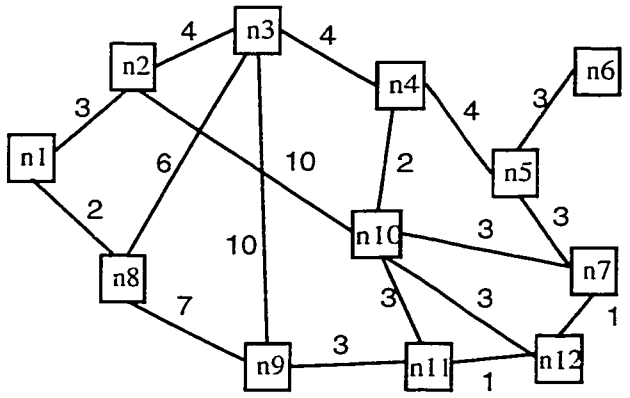


FIGURE 41. **Experimental Network 1**

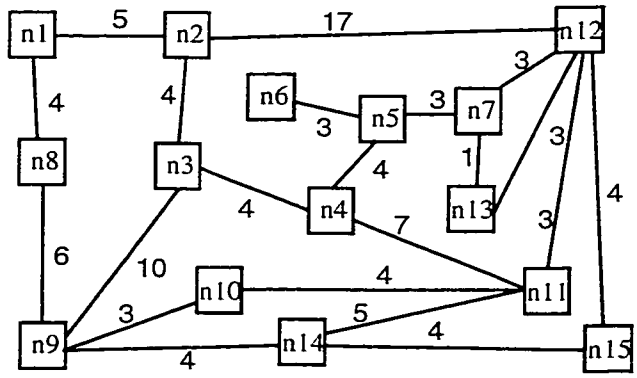
Two graphs were used during experimental investigation of the adaptive system for the three problems outlined earlier: point to point, point to multi-point and multi-path. These

1. In cases where a large number of equivalent paths are present in the network and the 90% simple threshold for path emergence is used.

are shown in Figure 41 and Figure 42. The numbers associated with the edges in these networks represent the costs of the edges at zero edge utilization. Each edge is considered to have a capacity of 63 units.

For problem one, the point to point path finding scenario, ten randomly generated traffic profiles were created for all source-destination pairs with bandwidth requirements sampled uniformly from the set {0, 2, 4, 6, 8, 10} bandwidth units. A bandwidth requirement of zero units was taken to mean that no path need be calculated for the source-destination pair. Paths were calculated such that the utilization of the network increasing by the bandwidth requirements of the traffic as paths emerged. All paths were computed in parallel. Initial network edge utilizations of 0, 30, and 50% were considered in order to test the effects of four different cost functions. For problem three, the same randomly generated traffic profiles were used for experimentation. For problem two, ten randomly generated traffic profiles were created with either 2,3 or 4 destinations. Bandwidth requirements for the point to multi point requests were identical to problem one.

FIGURE 42. **Experimental Network 2**



A population size of 50 was used with path emergence considered to have occurred when 90% of the population follow a given path. A maximum of 100 cycles (or generations) of the path finding algorithm was allowed before path calculations were stopped and 20 agents per cycle were sent out into the network for path finding. The value of α was allowed to vary in the range -0.25 to 3 and the value of β was allowed to vary in the range -0.125 to 1.5. A total of 16 bits was allowed for the encoding of α and also for β. When adaptive search was contrasted with its non-adaptive counterpart, with constant α and β, values of 2 and 1 respectively were used. These constant values were found to be a reasonable compromise for path finding. A value of 10 was chosen for Q, the constant of proportionality for the quantity of pheromone to be laid. An indirect representation was used with mapping of bit strings into floating point values in the above ranges in such a way as to cover the ranges uniformly. Values of 0.8 and 0.01 were used for the probabilities of crossover and mutation respectively. Single point crossover was used as the crossover operator.

Four cost functions were used in the experiments. These are shown in Figure 38. These cost functions are all functions of the utilization of the capacity of the network edge. It should be noted that the equation implied by Figure 38d is a constant implying that the cost is independent of edge utilization.

4.4.4 Results for Problem 1

Tables 3, 4 and 5 contain the results of experiments for constant α and β for the two experimental graphs with different initial edge utilizations for the path finding problem.

Tables 6, 7 and 8 contain the results of experiments where α and β were allowed to adapt during search. The mean and standard deviations of run times are given for the various cost functions used. By comparing tables 3 and 6, 4 and 7, 5 and 7, the results indicate that

TABLE 3. **0% Initial Utilization,** $\alpha\beta$ constant

		graph1	graph2
38a	Mean (secs)	25.12	31.22
	std dev. (secs)	2.68	3.42
38b	Mean (secs)	32.67	37.94
	std dev. (secs)	4.11	4.62
38c	Mean (secs)	27.56	30.22
	std dev. (secs)	2.76	3.34
38d	Mean (secs)	22.18	25.89
	std dev. (secs)	2.07	3.01

TABLE 4. **30% Initial Utilization,** $\alpha\beta$ constant

		graph1	graph2
38a	Mean (secs)	26.11	32.21
	std dev. (secs)	2.99	3.98
38b	Mean (secs)	27.33	31.43
	std dev. (secs)	3.22	4.76
38c	Mean (secs)	22.99	29.16
	std dev. (secs)	2.41	3.11
38d	Mean (secs)	22.18	25.89
	std dev. (secs)	2.07	3.01

TABLE 5. **50% Initial Utilization,** $\alpha\beta$ constant

		graph1	graph2
38a	Mean (secs)	25.55	32.28
	Std dev. (secs)	2.71	3.72
	Mean (secs)	33.69	38.96

38b	Std dev. (secs)	4.21	4.81
38c	Mean (secs)	33.11	35.65
	Std dev. (secs)	3.99	4.29
38d	Mean (secs)	22.18	25.89
	Std dev. (secs)	2.07	3.01

TABLE 6. **0% Initial Utilization,** αβ adaptive

		graph1	graph2
38a	Mean (secs)	18.91	23.99
	Std dev. (secs)	1.52	2.98
38b	Mean (secs)	28.11	28.01
	Std dev. (secs)	2.31	2.41
38c	Mean (secs)	20.09	24.22
	Std dev. (secs)	1.55	2.11
38d	Mean (secs)	17.1	19.67
	Std dev. (secs)	1.39	1.65

TABLE 7. **30% Initial Utilization,** αβ adaptive

		graph1	graph2
38a	Mean (secs)	19.11	24.11
	Std dev. (secs)	1.6	2.77
38b	Mean (secs)	21.2	25.22
	Std dev. (secs)	1.91	3.51
38c	Mean (secs)	17.01	22.11
	Std dev. (secs)	1.61	2.41
38d	Mean (secs)	17.1	19.67
	Std dev. (secs)	1.39	1.65

TABLE 8. **50% Initial Utilization,** αβ adaptive

		graph1	graph2
38a	Mean (secs)	20.18	24.91
	Std dev. (secs)	2.01	2.22
	Mean (secs)	23.99	28.99

38b	Std dev. (secs)	2.31	2.54
38c	Mean (secs)	24.95	27.11
	Std dev. (secs)	2.88	3.11
38d	Mean (secs)	22.18	25.89
	Std dev. (secs)	2.07	3.01

the adaptive system is clearly superior to the simple non-adaptive algorithm. The adaptive algorithm is typically 25% faster than the simple algorithm in finding paths for the two experimental graphs across the range of initial utilizations chosen. The standard deviations for search times also decrease when comparing the adaptive to the non-adaptive systems, again indicating that adaptation of α,β values has improved predictability of the search process. The results for 0, 30 and 50% initial edge utilization for cost function 6c being the same in tables 1, 2 and 3 can be explained by the fact that this cost function does not vary with the utilization of the network edges. By the same argument, tables 4, 5 and 6 show no variation in search times for cost function 38d. The mean edge utilization results are not included here due to space restrictions. However, cost function 38c provided the best overall results when reviewing the standard deviation on edge utilization.

4.4.5 Results for Problem 2

Tables 9 and 10 contain the results of experiments for constant and adaptive α,β for the two experimental graphs respectively. In both of these tables an initial edge utilization of zero was used. Results for an initial edge utilization of 30% and 50% are not presented as they exhibit similar patterns to those provided for problem one and provide no further

insight into the effects of the cost function chosen.

TABLE 9. **0% Initial Utilization,** αβ constant

		graph1	graph2
6a	Mean (secs)	62.91	80.01
	std dev. (secs)	6.99	9.01
6b	Mean (secs)	82.01	99.11
	std dev. (secs)	11.1	11.91
6c	Mean (secs)	70.65	80.11
	std dev. (secs)	7.01	9.91
6d	Mean (secs)	56.99	65.85
	std dev. (secs)	5.61	7.91

TABLE 10. **0% Initial Utilization,** αβ adaptive

		graph1	graph2
6a	Mean (secs)	45.31	55.91
	std dev. (secs)	3.41	7.15
6b	Mean (secs)	67.41	67.19
	std dev. (secs)	5.45	5.55
6c	Mean (secs)	47.34	56.67
	std dev. (secs)	3.51	5.01
6d	Mean (secs)	41.4	42.99
	std dev. (secs)	3.03	3.97

Once again, comparing tables 9 and 10, the adaptive system is clearly shown to be superior to the simple AS.

4.4.6 Results for Problem 3

Tables 11 and 12 contain the results of experiments for constant and adaptive α,β for the two experimental graphs respectively for the problem of finding a cyclical path between two nodes in the network. In both of these tables an initial edge utilization of zero was

used. Results for an initial edge utilization of 30% and 50% are again not presented are they exhibit similar patterns to those provided for problem one and thus provide no further insight into the effects of the cost function chosen.

TABLE 11. **0% Initial Utilization,** αβ constant

		graph1	graph2
6a	Mean (secs)	51.5	64.98
	Std dev. (secs)	5.94	7.11
6b	Mean (secs)	68.91	76.98
	Std dev. (secs)	8.97	10.01
6c	Mean (secs)	60.12	64.09
	Std dev. (secs)	5.92	7.98
6d	Mean (secs)	48.3	56.74
	Std dev. (secs)	4.91	6.71

TABLE 12. **0% Initial Utilization,** αβ adaptive

		graph1	graph2
6a	Mean (secs)	38.56	49.02
	Std dev. (secs)	3.01	5.87
6b	Mean (secs)	57.6	55.37
	Std dev. (secs)	4.32	4.58
6c	Mean (secs)	43.09	49.41
	Std dev. (secs)	3.07	4.76
6d	mean (secs)	34.12	41.56
	std dev. (secs)	2.9	3.21

As tables 11 and 12 clearly show, the adaptive system outperforms the basic path finding system with constant α and β values and exhibits improvements that are similar to those

observed for problems one and two.

It should be noted that when the search for a path starts, the cost element of the probability function dominates the calculation of $p_{ijk}(t)$, and an almost greedy heuristic comes into play, i.e., the least cost edge is probabilistically chosen. The actual value of α is relatively unimportant, as only a small amount of pheromone is present on the edges. Therefore, the main factor influencing choice of links is the actual cost of that edge. As routes are found, pheromone is laid on the edges that form that path. The amount of pheromone laid is inversely proportional to the total cost of the path found, and acts as a global measure of 'goodness'. As the quantity of pheromone rises, this brings the reinforcement part of the probability function, $p_{ijk}(t)$, into play. The sensitivity to pheromone, α, influences the choice of edges, and edges with higher pheromone concentrations are more likely to be chosen. It can easily be seen, therefore, that the importance of edge cost and pheromone concentration varies throughout the search process and that adaptation of the appropriate sensitivity parameters likely to be desirable.

Finally, allowing negative values for α and β has introduced the ability for agents to choose low pheromone concentration, high cost, edges. This has improved the ability of the system to recover from situations where agents have laid down large concentrations of pheromone on a non-optimal path early on in the search process.

4.5 Load Balancing Experimental Results

In order to carry out reproducible experiments, a special connections manager was developed. This is described briefly in Appendix A. With it, the user can choose the

connections within the list of all the connections created which one he wants to use and in what order. He has to define the cost functions he wants to compare, and the number of runs. Then, the program will establish the first connection of its list. When the path is allocated, this connection is stopped (without removing its agents or its allocation of bandwidth from the network: the important point is that the modifications due to the former connection still remain in the graph) and the next connection is launched, and so on, until the list is empty.

The criterion used to measure if the load balancing of an experiment is good or not was the standard deviation of the occupancy of the links in the graph. A hypothetically perfect routing, with all the links at the same level of occupancy, will have a standard deviation in link utilization of zero. A very bad routing, with all the line empty but one having a utilization close to 100% will have a very high standard deviation. The criterion for good routing solutions *is the lower the standard deviation, the better the load balancing*. Therefore, we used the standard deviation as a means to measure the results of our experiments.

The use of a species id was not good for those experiments because instead of spreading, the connections having the same species id tried to follow the pheromone trail of its kind. This is the reason why we used connections with different bandwidth (even slightly - in the order of 0.1 difference) in all those experiments.

The first experiment (experiment #1) was on the graph shown in Figure 41, with a set of 8 different connections between n1 and n12. In a second experiment (experiment #2),

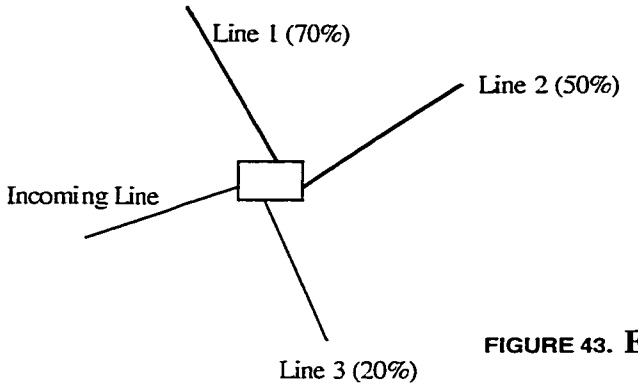
performed on the graph shown in Figure 42, we used another set of connections that seemed more realistic for us. In reality, you have a lot of connections using a small percentage of the total bandwidth. This set was made of:

- one connection whose bandwidth represents 20% of the maximum bandwidth available,
- 2 connections whose bandwidth represents 10% of the maximum bandwidth available,
- 4 connections whose bandwidth represents 5% of the maximum bandwidth available,
- 12 connections whose bandwidth represents 1% of the maximum bandwidth available.

For experiment #1, the first thing that was obvious, was that there was a spreading effect in using a cost function different from the *defaultCostFunction*. Secondly, it appeared that there was not one optimal cost function for all occupancy ranges. In fact, for an initial occupancy of 0%, the *linear* cost function gave the best results. For an initial occupancy of 25%, the *non-linear* cost function gave the best results. Finally, the *quartic* cost function provided the best results for the 50% initial occupancy.

We then tried to improve the algorithm. We had seen that multiplying the cost by a cost function could have a spreading effect, and then we tried to *force* this spreading effect. We modified the way the next link was chosen. In the basic algorithm, each possible next link has a selection value based on the probability function as discussed in previous sections.

In the enhanced algorithm, this selection value will be multiplied by a factor. This factor favours a link if it has a lower occupancy than the other possible next links (the factor has to be greater than 1 in that case). Conversely, it lowers the likelihood of selection if that link has a higher occupancy (the factor has to be lower than 1 in that case). We use an exponential function whose exponent is the difference of the occupancy of the current link



$$factor_1 = e^{\frac{(0.5-0.7)}{T}} \cdot e^{\frac{(0.2-0.7)}{T}} = e^{\frac{0.7}{T}}$$

$$factor_2 = e^{\frac{(0.7-0.5)}{T}} \cdot e^{\frac{(0.2-0.5)}{T}} = e^{\frac{0.1}{T}}$$

$$factor_3 = e^{\frac{(0.7-0.2)}{T}} \cdot e^{\frac{(0.5-0.2)}{T}} = e^{\frac{0.8}{T}}$$

FIGURE 43. **Example Factor Calculation**

with the occupancy of all the other candidates.

The forcing factor is given by the equation above. As can be seen from the functional form, low occupancy links are exponentially favoured over high occupancy links. Figure 43 demonstrates the calculation of the factor for an example node. It can easily be seen that Line 3 with 20% has the best factor, whereas Line 1 has the worst, as expected.

In the experiments with this factor, the results of all cost functions were improved, but the ranking was different: for low occupancy, *the non-linear* cost function was better than the *linear* function. The *quartic* cost function was still the best for higher occupancy values. As a result of these experiments we were not sure that such an improvement for the standard deviation was worth it if the total cost was too high, or the time needed too large.

In order to assess the utility of the forcing factor, a number of experiments using the network in Figure 41 were performed. T was varied in these experiments, so we defined a total gain, taking into account the gain for the standard deviation, the total cost, and the time needed for the connection emergence. This is shown in Figure 44.

In the total gain term, the gain for time is divided by 4 because this parameter seemed to

us less relevant than the cost and the standard deviation of occupancy.

The choice of the reference is highly significant for each gain. In order to be able to compare all four cost functions with different values for T, we selected the results of an experiment without factor effect, using the *defaultCostFunction* as cost function for the reference gain measure.

$$Gain_{SD} = 1 - \frac{SD}{SD_{Ref}}$$

$$Gain_{Time} = 1 - \frac{Time}{Time_{Ref}}$$

$$Gain_{Cost} = 1 - \frac{Cost}{Cost_{Ref}}$$

$$Gain_{Total} = Gain_{SD} + \frac{Gain_{Time}}{4} + Gain_{Cost}$$

FIGURE 44. T-factor gain calculations

From all these experiments, it appeared that there would not be an obvious winner. We can only choose the best compromise. In reality, the occupancy of network links are rather high, the best compromise could be using *quartic* cost function and T=0.5. In order to confirm that the results were not biased by the choice of the set of connections, we carried out experiment #2 and it appeared that the same set of parameters would be the best compromise.

Figure 45 shows the gain of each used cost function with varying values for T. From the left to the right, there are the results of our experiments with the following initial occupancy: 50%, 25% and 0%. The total gain is represented for each cost function (*defaultCostFunction*, *linear*, *non-linear* and *quartic*). These values are the mean values

computed over 100 runs for each set of parameters.

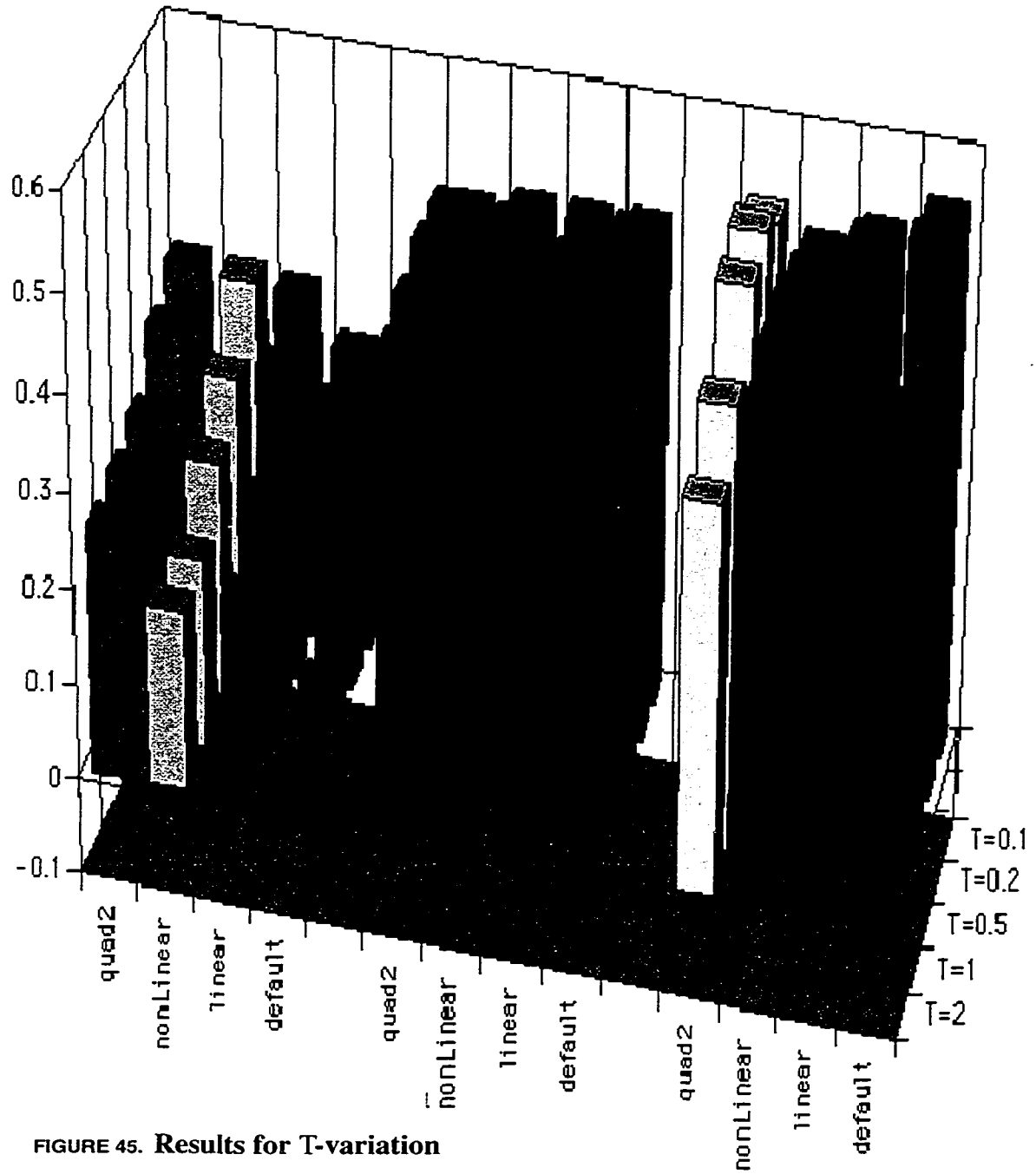


FIGURE 45. **Results for T-variation**

4.6 Other Algorithm Improvements

4.6.1 Sensitivity to Parameters

The algorithm is sensitive to both the agent parameters and the particular graph on which the algorithm operates. In order to reduce this, we focused on evolving the most important parameters (sensitivity to cost and pheromone). This was successful in reducing the sensitivity to the size and complexity of the graph.

Other parameters that can affect the efficacy of the algorithm are:

number of agents created and frequency of agent creation

These two parameters are very sensitive to the size and complexity of the graph. By default, 10 agents are created every 10 'ticks'. It is often better to create agents more frequently, *although it is not always the case that more agents lead to a better solution.* Indeed, these parameters are likely to be sensitive to more than the network; it appears that they are sensitive to the particular connection requested too.

In order to reduce the sensitivity to these parameters, the CCMA monitors the progress of the path finding process. It keeps statistics on two aspects of the search process: the convergence of the process itself and the rate of pheromone density increase in the network. The convergence of the path finding process is measured as a percentage of the number of returned agents -- within the window stored -- that follow the same path. If this percentage does not increase as search progresses, the number of agents being generated is increased using a growth factor with a logarithmic characteristic. Similarly, if the density of pheromone in the network is not increasing as measured by the returning agents, it

means that the rate of evaporation is too high. In this case, the CCMA increases the amount of pheromone to be dropped, using a growth factor with a logarithmic characteristic.

The actual amount of pheromone dropped by an agent is this amount divided by a measure of how good the found route was (i.e., some function of it's cost). Therefore, this parameter is sensitive to the cost distribution on the graph, and is related to the 'Orders of Magnitude' problem, below.

agent return buffer size (n)

agent percentage required for emergence (m)

Deciding whether a path has emerged is a difficult problem. In the research reported here, it is particularly difficult as, from the outset, we decided to avoid using a global monitor. Therefore, one of the nodes (the designated source node) was responsible for determining whether a path had emerged, this being the responsibility of the CCMA.

We used a simple scheme: of the last n agents to return, m followed the same route. This mechanism works reliably as a tool for detecting emergence. However, where there are several similar solutions to a problem, it takes a long time for one to dominate over another (it tends to oscillate).

A refinement of this emergence criterion analyzed the amount of the graph that has been explored. This is done by recording the number of distinct routes that the agents find. As the search stagnates, fewer new paths are found (i.e., the standard deviation of the set of

routes found approaches zero), we assume that no more new paths will be found, and then choose the current best available solution. Therefore, these parameters are not a particular problem. Another variation on this refinement was to sort the returning agents according to the cost of the path found, placing agent paths into buckets of a known width. When the threshold percentage (m returning agents of the last n) are placed into the same bucket, the path is said to have emerged.

4.6.2 Orders of Magnitude

There are two parts to this problem: cost magnitudes and pheromone magnitudes. Consider the following. Link AB has a cost of 1, while link AC has a cost of 100. This leads to the situation where AC is significantly less likely to be chosen than AB. It can be argued that this is exactly the desired behaviour. However, without careful consideration of relative costs on the network, this can lead to poor utilization of resources. Therefore, we recommend that costs on links be kept within one order of magnitude, as this will ensure good searching of the network. This may require a level of normalization before using the algorithm. *Should this approach be deployed in networks we would expect vendor agreement such that this problem be avoided.*

The other magnitude problem is to do with pheromone. When a large number of agents follow a particular route, a very large amount of pheromone can build up on that path. Indeed, the amount can be so great that it is impossible for the agents to escape that path. This problem is more significant than the cost magnitude problem, as it becomes very problematic on a network where connections come and go, freeing bandwidth on other

links. When the agents become locked into a route due to a high amount of pheromone, and another, far better route becomes available, the agents will never find that route. A solution to this problem is to ensure that the concentration of particular chemicals is bounded.

The use of a genetic algorithm allied to a simplified Q-learning mechanism served to avoid some of this, as it allows for trails with large quantities of pheromone to be ignored by using a repulsive term in the MDF. However, even with adaptive coefficients it algorithm can still become locked prematurely into solutions. This is a classic problem of exploitation versus exploration that is present in all search algorithms.

4.6.3 Complexity

The algorithm described here has yet to be analyzed mathematically and is beyond the scope of this thesis. It is complex and will, in all liklihood, require techniques from Statistical Mechanics to prove convergence. However, Dorigo's work with the Ant System has clearly shown that the technique scales to large problems -- in the 1000's -- which is large for a network of the type considered here. The number of possible paths in a network scales exponentially with increasing number of network nodes and so it is important that the search be efficient. We are confident that Ant Search results will transfer to the routing problem described here.

4.7 Application Oriented Routing

The statistical properties associated with applications using the resources of a network have natural synergies when considering measured quality of service. However, it is

difficult to know in advance which applications will benefit from resource sharing. This section introduces the concept of application oriented routing, an approach to routing where routing agents learn to collaborate in order to optimize the quality of service experienced by applications sharing network resources. The section proposes a multi-swarm problem solving architecture as described in this chapter and based upon SynthECA principles, with individual swarms responsible for determining network connections and learning to collaborate using a coevolutionary process as a consequence of monitoring the quality of service of the resulting connections.

4.7.1 The Model

The model for collaboration between applications for the purpose of improved quality of service draws its inspiration from the agent architecture described in previous chapter and its application in the previous section.

In the agent-oriented model of [White 98a], [White 98c] and restricting the discussion to point-to-point connections only, ant-like agents drop pheromones which mark the route from source to destination. Pheromones are represented as strings with each bit position consisting of a symbol from the alphabet {1,0,#}, the hash symbol matching both 1 and 0. All pheromone bit strings are of the same length. No significance is associated with particular bit patterns, agents are unable to infer, for example, the destination of a routing agent from the pheromone that it uses. The quantity of pheromone dropped is a function of the cost of the route; the shorter the route, the greater the quantity of pheromone dropped. Successive agents reinforce low cost routes from source to destination by using an MDF

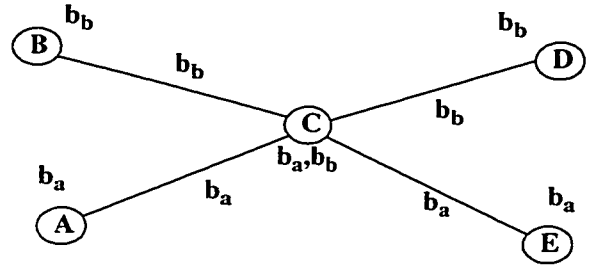


FIGURE 46. **Example Network Fragment**

as shown in equations 8-11. In this investigation, a single pheromone was used for route finding; i.e., r equals 1 in equations 10 and 11. A static agent resident on the connection source node monitors the progress of the routing activity and decides when a path has emerged, subsequently sending out an allocator agent to reserve resources in the network when emerged. The nodes and links -- the resources used by the connection -- are then remembered; this being equivalent to the knowledge of the service dependency model described in [White 98b] and which is the subject of the next chapter.

Collaboration in our model is achieved by having routing agents sense each others' pheromones. In this way, pheromones dropped by explorer agents associated with one connection will be sensed by explorer agents associated with another connection, the net effect being for the two agents to collaborate in routing; i.e., there will be a tendency for them to share parts of their routes. Sensing of each others' pheromones is straightforward owing to the presence of a the # symbol in the pheromone encoding[1], and uses the standard classifier template matching technique implicit in the Binary Chemistry of Order N.

1. Remember, we are using the Binary Chemistry.

Referring to Figure 46, which represents a small part of a much larger network, two circuits are defined, one from A to E and one from B to D. The labels, $\mathbf{b_i}$, represent the encoding of the pheromones used to find a path from source to destination.We call this encoding the connection encoding, CE. The label $\mathbf{b_i}$ is all we know about the application. Hence, in Figure 46, we see that the path from A to E uses the edges A-C and C-E, and the nodes A,C,E. Similarly, the path from B to D uses the edges B-C and C-D, along with nodes B,C and D. The connection monitoring agent sent from A to E would return with information regarding the service dependency model for the connection A-D that impinges upon it. Note, that the pattern $\mathbf{b_b}$ cannot be used to infer that the connection B-D has D as its endpoint.

Once the connection has been created; i.e., a circuit has been set up, the set of pheromones sensed during the exploration phase is noted. We call this the collaboration set, CS. The *end to end* quality of service (QoS) is then monitored continuously by a static QoS agent resident on the source node. The average quality of service experienced during the lifetime of the call is considered to be a measure of the fitness associated with the circuit.

When a significant change in measured QoS is observed, a circuit monitoring agent is sent out to traverse the nodes and links used by that circuit. It remembers the connections that are using the resource for each node and link; i.e. the pheromones for circuits that are currently active. This set we call the notification set, NS. Upon return to the source node, the set CS is compared with NS in order to observe differences. Typically, notification

occurs as a result of connection setup or tear down. In both situations the sets differ by a single encoding. We call this encoding the notification encoding, NE.

Using this information, if the QoS has increased, we decrease the hamming distance between the CE and NE encodings by applying the rules for each bit in the encoding using the transformation rules in Table 13.

TABLE 13. Increasing QoS transformation rules

CE	NE	CE transformed	Probability
0	1	# 0	p_{10i} $1-p_{10i}$
1	0	# 1	p_{01i} $1-p_{01i}$
1	#	1 #	$p_{1\#i}$ $1-p_{1\#i}$
#	1	1 #	$p_{\#1i}$ $1-p_{\#1i}$
0	#	0 #	$p_{0\#i}$ $1-p_{0\#i}$
#	0	0 #	$p_{\#0i}$ $1-p_{\#0i}$
#	#	1 0 #	$p_{\#\#1i}$ $p_{\#\#0i}$ $1-p_{\#\#1i}-p_{\#\#0i}$

If the QoS has decreased, we first determine the minimum covering encoding, MCE, for the collaboration set, CS. The MCE is the least general unifier for the set of encodings in CS. Having computed the MCE, we increase the hamming distance between the CE and

NE encodings using the transformation rules in Table 14. We define the generality of an

TABLE 14. Decreasing QoS transformation rules

MCE	NE	CE transformed	Probability
0	1	0	1
1	0	1	1
1	#	1	1
#	1	0 ##	$p_{\#1d}$ $1-p_{\#1d}$
0	#	0	1
#	0	1 #	$p_{\#0d}$ $1-p_{\#0d}$
#	#	1 0 #	$p_{\#\#1d}$ $p_{\#\#0d}$ $1-p_{\#\#1d}-p_{\#\#0d}$

encoding to be the number of hash symbols it contains. If the generality of the MCE is zero, the above table implies that no transformation can occur as all probabilities associated with 1 or 0 are 1. In this situation we compute the bit position within the MCE encoding whose value we are least certain of and transform that. We do this with probability p_f. In the situation where multiple encoding positions have equally uncertain values, we choose randomly between them.

4.8 Experimental Setup

Two graphs were used during experimental investigation of the model described in the previous section These are shown in Figure 41 and in Figure 42. The numbers associated with the edges in these networks represent the costs of the edges at zero edge utilization. Each edge had a capacity of 63 units. The network nodes and links were totally reliable

and never failed or exhibited degraded performance.

For the point to point path finding scenario, ten randomly generated traffic profiles were created for all source-destination pairs with bandwidth requirements sampled uniformly from the set {0, 2, 4, 6, 8, 10} bandwidth units. A bandwidth requirement of zero units was taken to mean that no path need be calculated for the source-destination pair. Paths were calculated using the ASGA algorithm [White 98c], with the utilization of the network increasing by the bandwidth requirements of the traffic as paths emerged. All paths were computed in parallel. Initial network edge utilizations of 0%, and a constant cost function were used. A constant cost model was used in order to remove the effects of congestion on routing. Only the capacity limits of the edges were considered as constraints. Although unrealistic in actual networks, our desire in this investigation was to determine whether collaboration would naturally emerge and, most importantly, facilitate higher average network QoS when compared to no collaboration. Four bits were used to encode the routing pheromone.

A population size of 50 was used with path emergence considered to have occurred when 90% of the population follow a given path. A maximum of 100 cycles (or generations) of the ASGA algorithm was allowed before path calculations were stopped and 20 agents per cycle were sent out into the network for path finding. The value of α_k was allowed to vary in the range -0.25 to 3 and the value of β_k was allowed to vary in the range -0.125 to 1.5. A total of 8 bits was allowed for the encoding of α_k and also for β_k. A value of 10 was chosen for Q. An indirect representation was used with mapping of bit

strings into floating point values in the above ranges in such a way as to cover the ranges uniformly. Values of 0.8 and 0.01 were used for the probabilities of crossover and mutation respectively. Two-point crossover was used as the crossover operator as it was found to be slightly superior to the one point crossover used in previous experiments. The values of the probabilities in Table 13 and Table 14 were set to 0.2. The value of p_f was set to 0.5.

4.8.1 Results and Discussion

Results for three typical runs of the system are shown in Figures 47, 48 and 49. Each chart consists of two curves. The raw time series, with each data point marked, is displayed along with a log fit to the data. Each graph clearly shows the overall improvement in time of the average network quality of service. While periods of declining average QoS can be seen -- most strikingly in example 3 -- the overall trend is for improvement over time.

Analysis of the experimental data reveals that temporary declines in QoS occur as a result of two major factors. The first is the over generalization in application collaboration. In other words, applications associate with too many other applications too

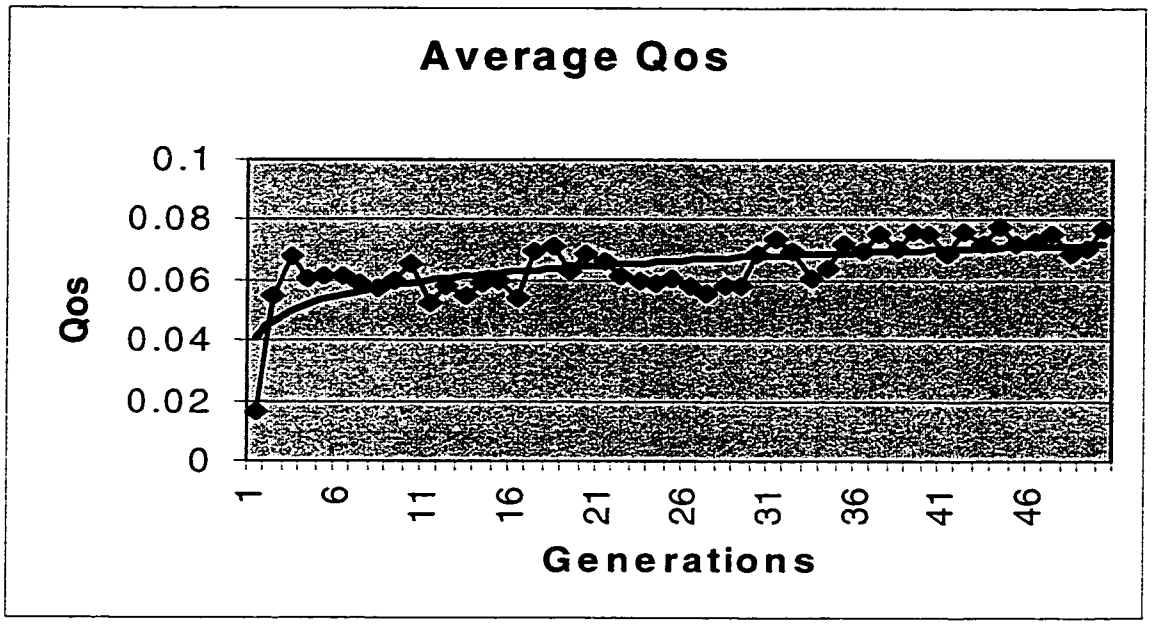


FIGURE 47. **Average Qos Example 1**

early in the adaptive process. The second effect is the problem of decreasing QoS in the situation when the generality of the MCE is zero. In this case, we tend to force

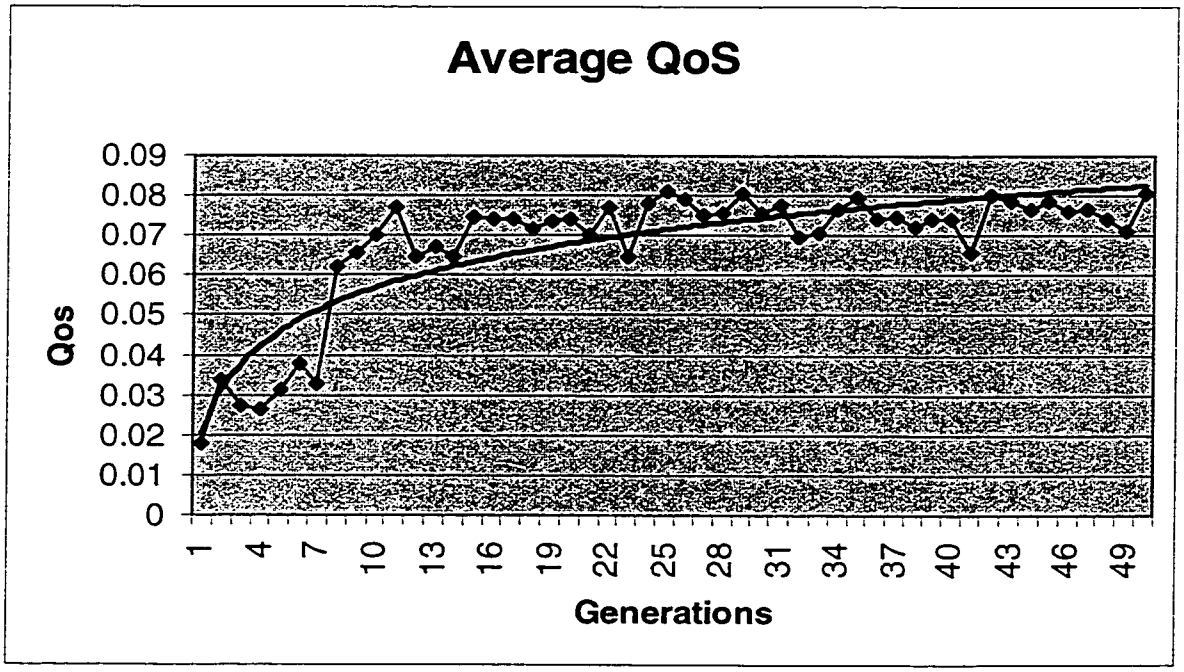


FIGURE 48. **Average Qos Example 2**

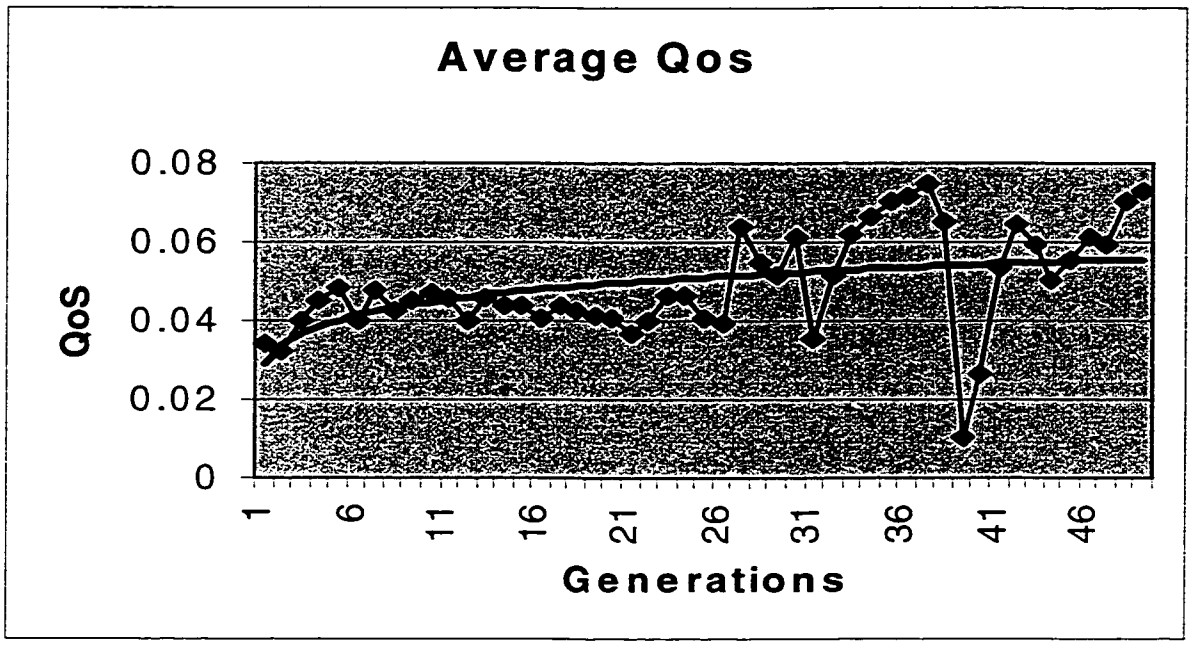


FIGURE 49. **Average Qos Example 3**

applications which would have collaborated to no longer collaborate.

The experimental results presented have demonstrated that learning to exploit application synergy can improve average network quality of service. However, considerable experimental work remains in order to demonstrate conclusively that this approach would be viable for a real network. In particular, an exhaustive experimental analysis of how the probability assignments in Tables 13 and 14 affect the performance of the system needs to be undertaken. The model proposed represents one example of how we may learn synergies between applications. Equally importantly, but as yet untried, is the problem of learning anti-synergies; i.e., learning which applications should avoid each other. In fact, this system represents a special class of a more general class of coevolutionary systems where the agents have an MDF that contains terms for collaboration neutral, collaborative and anti-collaborative behavior. In these more general

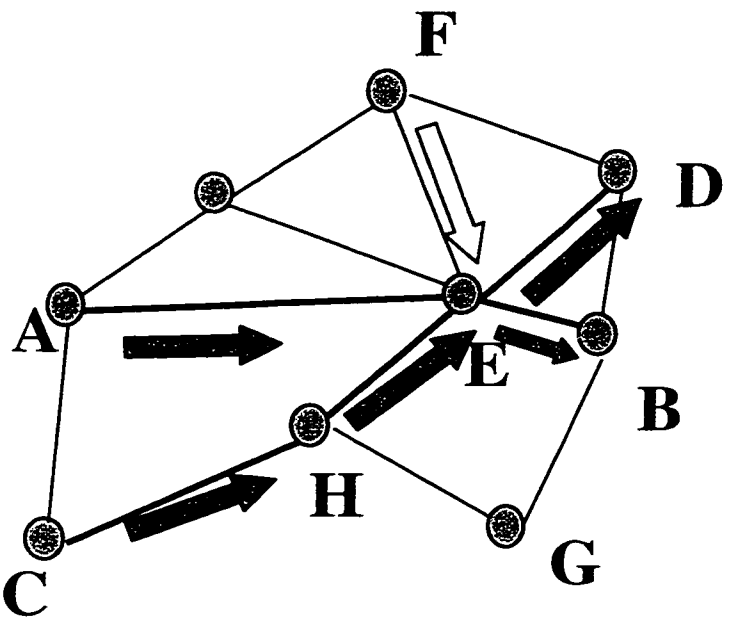


FIGURE 50. **Example of Multi-priority Routing**

systems, equation 10 would be the product of three pheromone and one cost terms. The three pheromone terms would represent collaboration neutral, collaborative and anti-collaborative behavior. The α_{kp} terms for collaboration and collaboration neutral terms would be negative, and the anti-collaboration term positive. We believe that identification of these three forms of behavior, with individual sensitivity coefficients, will improve the performance of the system.

4.9 Multi-priority Routing

In many networks we would like to route application traffic with varying priorities. In the model proposed here, the idea is to move existing traffic to longer (as measured by the link costs) routes when a higher priority connection request enters the system. Consider, for example, the scenario as represented in Figure 50. Here, two routes have already been

computed: AB and CD. These are indicated with the dark arrows. A new connection request now enters the system (FG) as shown by the white arrow. The idea, then, is to have the AB and CD explorer agents (which continue to explore the network even after route allocation) to recognize that the environment has changed and to deallocate the existing route and find one that avoids the higher priority traffic. However, having said this, we want to provide this functionality by using the same stigmergic principles detailed in the previous sections.

Considering only the higher priority connection finding algorithm for a moment, this may be achieved in a number of ways using the SynthECA architecture.

4.9.1 Adding a Priority Pheromone

In this approach, the explorer algorithm proceeds as before, with one modification. The MDF of the explorer agent does not sense the lower priority connection pheromones but has two receptors instead of one used in the basic routing system. In fact, we use a two class pheromone system. The first class of pheromones has the same meaning as in the earlier sections of this chapter, i.e., it is a member of the Binary Array Chemistry of order N. The second class of pheromones is a different length chemical, i.e., it is a member of the Binary Array Chemistry of order N^p, where $N^p \neq N$. In this way, the utilization (and therefore the cost) of a link appears lower to a higher priority connection when compared to a lower priority connection. This requires that chemicals have a well-defined and standardized encoding. As before, all explorer agents find the shortest path using the algorithms described in the previous sections and, upon path emergence, an allocator is

sent out to allocate resources in the network. The allocator differs here from the previous design. In a multi-priority system, the allocator deposits pheromone on its way back from the destination having allocated resources for the connection. The reason for the allocator depositing pheromone on its way back from the destination is that we want to ensure that resources have been allocated before communicating the existence of the new connection to other connections in the network. The allocation pheromone, or a-chemical, *is* sensed by lower priority connection explorer agents. Upon return of the allocation agent, the lower priority explorer agents, or lp explorers as we shall refer to them, begin to experience higher costs for links on their paths that have had resources allocated for the higher priority connection. With the added costs, lp explorers will deposit smaller quantities of their connection pheromones, indicating a reduced confidence in the path. If confidence in the path is sufficiently reduced, i.e., it is no longer the shortest path, the majority of lp explorers will begin to follow the new shortest path. At this point, the CCMA will send out an allocation agent for the new shortest path and a deallocation agent for the old shortest path. Hence, the higher priority connection has forced the movement of lower priority connections off of its path.

The above is probably best illustrated with an example. Consider again the network and connections shown in Figure 50. Let the order of the connection and priority chemistries be 8 and 4 respectively. The latter implies that 4 priority levels exist[1]. Let all edges in the network have unity cost, i.e., $C_{ij}(u) = 1$, except edges FD and GH which have $C_{ij}(u) = 2$.

1. The receptors sensing the priority-related chemicals will sense only the priorities above them. Hence a priority 1 receptor will be of the form: 1###, while priority 4 will have the form 1111. Hence priority 1 receptors will sense priority 4 traffic, and is thus considered low priority traffic.

The paths indicated by dark arrows for connections AB and CD are then the shortest paths and will be found by explorer agents. Now, consider the introduction of the high priority connection FG. The shortest path for this connection is FE-EB-BG. Assuming that the allocator for this connection drops 3 units of the a-chemical, lp explorer agents for the AB connection will begin to experience higher costs for the AE-EB path, now 4, which makes the path longer than the shortest path, now AE-ED-DB with cost 3. After some time, the CCMA agent for the AB connection will observe that the majority of its explorer agents follow the new shortest path and will send deallocation and allocation agents out into the network in order to move the connection to the new shortest path.

Obviously, the above example has been constructed to demonstrate the path moving algorithm. However, it is sensitive to the quantity of a-chemical deposited in the network by the allocator agent for the higher priority connection. If the concentration of a-chemical is too low, the lower priority connections sharing links with the high priority connection will not be forced to move. This limitation can be overcome by having the CCMA monitor the quality of service associated with the connection. If the measured quality of service falls below the required service level agreement, an agent is sent out into the network that deposits higher concentrations of the a-chemical, thereby increasing the effective cost of the links as seen by the lp explorer agents.

Returning to the above example, assume that the high priority connection allocation agent deposited only 1 unit of the a-chemical. In this case, the cost of the path for the AB connection does not appear to change, and no re-routing occurs. The CCMA for the high

priority question, seeing its quality of service to be below the agreed value, sends an agent out into the network to deposit a further quantity of the a-chemical, say 2 units. The rate of increase of a-chemical concentration achieved by sending out multiple agents that deposit a-chemical we call the *priority momentum factor.*

At this point, the quantity of a-chemical on the high priority connection links is 3, as before, and re-routing consequently takes place. With a lower priority momentum factor the required number of agents depositing a-chemical to achieve re-routing will be higher, but re-routing will inevitably occur when the route is made "expensive enough" as seen by the lp explorer agents.

4.9.2 Using Pheromone Decomposition

In this approach -- inspired by the parasitic behaviour of certain species of ant -- the high priority connection explorer agents decompose the routing pheromones that they encounter on links that they traverse. As before, explorer agents compute shortest path routes through the network using the algorithms previously described. However, after allocating resources for the route which emerges, post-allocation explorers have an chemistry enhanced with a type 2 reaction. A type 2 reaction is a catalytic decomposition reaction, $X + Y \rightarrow Y$. These modified explorer agents decompose the routing pheromones associated with lower priority connections. In order to do this, a substring of the encoding used must be allocated to priority. It is equivalent to "gluing together" the two chemicals proposed in the approach described in the previous section. Using the encoding examples of the previous section, we could use (8+4) 12 bits for a routing pheromone, where the last

four bits would be the priority bits. In this case, a priority 4 connection would possess a receptor for the encoding ##########1, thereby being able to sense all class 1, 2 and 3 routing pheromones. Note that encodings for a particular priority class have a zero in the appropriate position in order to avoid the decomposition of the self chemical. The self chemical, i.e., the connection's own routing pheromone, is the catalyst. The importance of the catalyst here is the higher the concentration of it, the more it decomposes of the lower priority non-self routing chemicals. Thus, on links used by the high priority connection, decomposition will occur at a higher rate. By coupling the decomposition rate to the concentration of self, off-path links are affected to a lesser extent than on-path links.

As with the previously described approach, the sensitivity to individual parameters may meet that re-routing of lower priority traffic does not occur quickly. The solution, once again, is to employ a momentum term. If, after a number of post-allocation agents have been sent out, the measured quality of service is still below that specified in a service level agreement, the rate of decomposition of non-self routing chemicals is increased. Ultimately, with this mechanism, the increasing differential decomposition rates between the on- and off-path links will cause re-routing of the lower priority traffic.

4.10 Using SynthECA Routing Agents on Real Networks

There are two primary mechanisms for utilizing this approach on a real ATM network or transmission network: finding routes on the network itself, and finding routes in the management system. The algorithm appears to work best on networks where connections are relatively long lived; i.e., computing permanent virtual circuits rather than switched

virtual circuits.

4.10.1 SynthECA in the Network

For the algorithm to be able to operate on a real network, several mechanisms must be available:

A method of assigning source and destination nodes and informing those nodes of their status (probably via a management system or an embedded communications channel).

A method of packaging agents, their execution state and their data so that they can be transported across a link (c.f. mobile agents in Grasshopper [Grasshopper] or the framework designed here in the Systems and Computer Engineering Department of Carleton University [Susilo 97]).

On each node, a computing platform and a method for providing information about connected links to the agents.

For each link (probably recorded at each end node) a mechanism for recording pheromone concentrations for each species and evaporating this over time[1].

On the source node, a computing platform, the CCMA, must be present in order to determine whether a route has emerged.

All of the above points can be addressed with the application of Java, a mobile agent framework and the use of many of the Virtual Managed Component (VMC) ideas found in [Bieszczad 98], [Susilo 97], [Pagurek 00] and [Schramm 98]. The extensions to the

1. Alternatively, evaporator agents could circulate and reduce the pheromone concentrations.

Carleton University mobile agent toolkit described in Section 3.7 provide a potential solution to the problem of recording pheromone concentrations and communicating them to interested agents.

Besides these requirements, some consideration must be given to the amount of bandwidth taken up by these agents. The agents themselves are relatively small, although there may be a large number of them circulating in the network at any time. Whether code is transferred with each and every agent or resides on the nodes is a matter or engineering. The former approach avoids the problem of agent code upgrade (a problem discussed at length in chapter 6) while the latter approach reduces the network bandwith consumed by the mobile agent. An intermediate solution is possible wherein the code resident on the node is considered cached and forgotten after a period of time. There may also be many connection requests being satisfied concurrently. The amount of bandwidth available in the network is likely to be several orders of magnitude higher than the total bandwidth available, and so in many situations, these concerns are probably unfounded[1].

However, when the network utilization is approaching its capacity, sending many agents may be deemed impractical, as it eats up too much bandwidth which could be more productively used. From the previous sections, it is clear that the real benefits of the algorithm show when the network is approaching its capacity, and therefore it is valuable to retain the algorithm.

A compromise is to assign a small amount of bandwidth on each link permanently to the

1. For example, channels in Wavelength Division Multiplexed (WDM) networks are 2.5Gb/sec.

algorithm that will make best use of it, i.e., define an embedded management communications channel. Furthermore, when the network has sufficient capacity, the algorithm can make use of an additional amount to maximize its performance. If the network operator decides to use the final 0.01 percent of the bandwidth for network traffic, the swarm algorithm can be suspended and the management channel given over to network traffic.

A further benefit of having agents roaming the network is that they can gather transmission delay and cell loss information. This can be fed back to the management system, as statistics about how well the algorithm and network are operating.

4.10.2 SynthECA in the Management System

While the previous section has suggested employing SynthECA agents directly in the network, another approach is running the ASGA algorithm on the management system using snapshots either of the network, or by querying the nodes for information (although the latter is likely to be unacceptable). We can view the network snapshots as a simulation of the network. Network snapshots can be obtained using standard graph algorithms such as depth first search, for example.

This method is analogous to the simulator already implemented, in that the agents operate on a model of the actual network. However, as we deal with snapshots, it is possible that some information will be incorrect. The allocator agent must be analyzed and time dependencies between allocating on the network model and allocating real bandwidth on the actual network must be reduced. We would expect that a network view

would only be required for replanning of the network using techniques described in [White 99a].

This latter point raises the issue of the integration between the algorithm, the management system and the network. The algorithm results must be close enough to the real network in order for the solutions to be valid for any length of time on the real network. It is clearly not desirable to work on an out-of-date network model, nor is it for allocation to be attempted when the real network says that it is not possible.

4.11 Summary

This chapter has shown that this multi-agent search technique, called Swarm Intelligence, embodied in SynthECA agents, can solve routing problems on networks.

The main strengths of the algorithm are its robustness, the simple nature of the agents, and that it continues searching for new solutions even if a very good one was found. The algorithm also appears to perform well in problems with a large graph. It should also be noted that Dorigo's work on TSP has indicated that the AS is insensitive to the parameter settings of the system. While the system described here is different in detail to Dorigo's, we have experienced the same insensitivity with respect to parameter settings.

In an enhancement of the routing algorithm, we were able to obtain a load-balancing algorithm. All our experiments were on general graphs, therefore this kind of heuristic could be used as a general purpose tool (c.f. Dorigo's application of the same algorithm to the TSP, QAP, JSP with minimal changes).

The Swarm Intelligence dynamic routing algorithm showed strengths of versatility and

generality. For example, adding multiple priority levels to the algorithm proved straightforward. These can also be weaknesses: some other very specialized algorithms can outperform this algorithm. Its real strength is in the way it can adapt to new situations. Real ants are able to quickly find a new path when their environment changes, and the swarm algorithm also exhibits this behaviour. For instance, it can be shown that the explorer agents react quickly to changes in available bandwidth (and hence cost) by avoiding a particular link. This kind of heuristic would be very relevant in cases where the "environment" is changing rapidly, with a large flow of data. This is the case for problems like air traffic management [Ndovie 94].

Adaptive behaviour could be used in managing incidents, network failures or networks with rapidly changing topologies[1]. The fact that it can adapt to totally new situations may be very useful. This way, during the failure of some of its physical elements, a network would be able to manage routing. It is interesting too, that the agents can behave and act without human intervention. During network failures, this algorithm could behave as the "immune system" of the network and insure that during incident the routing of data would still be done.

The interaction of a diagnostic agent swarm and the routing swarms as detailed in this chapter are the subject of the next.

1. Such topologies are found in satellite networks and low power radio networks such as those proposed for home use. See, for example, [Bluetooth 99].

CHAPTER 5

Distributed Fault Location Using SynthECA Agents

5.1 Overview

This chapter describes how multiple interacting swarms of adaptive mobile agents can be used to locate faults in networks. The chapter proposes the use of distributed problem solving using learning mobile agents for fault finding. The chapter uses the SynthECA architectural description for an agent that is biologically inspired and proposes chemical interaction as the principal mechanism for inter-swarm communication. Agents have behavior that is inspired by the foraging activities of ants, with each agent capable of simple actions; global knowledge is not assumed. The creation of chemical trails is proposed as the primary mechanism used in distributed problem solving arising from the self-organization of swarms of agents. Fault location is achieved as a consequence of agents moving through the network, sensing, acting upon sensed information, and subsequently modifying the chemical environment that they inhabit. Elements of a mobile code framework that is being used to support this research, and the migration mechanisms used for agent mobility within the network environment, are described.

5.2 Introduction

The telecommunication networks that are in service today are usually conglomerates of heterogeneous, very often incompatible, multi-vendor environments. Management of such networks is a nightmare for a network operator who has to deal with the proliferation of human-machine interfaces and interoperability problems. Network management is operator-intensive with many tasks that need considerable human involvement; fault diagnosis being one of the most important. Legacy network management systems are very strongly rooted in the client/server model of distributed systems. This model applies to both IETF [Case 90] and OSI [Yemini 91] standards. In the client/server model, there are many agents providing access to network components and considerably fewer managers that communicate with the agents using specialized protocols such as SNMP or CMIP. The agents are providers (servers) of data to analyzing facilities centered on managers. Very often, a manager has to access several agents before any intelligent conclusions can be inferred and presented to human operators. The process often involves substantial data transmission between manager and agent that can add a considerable strain on the throughput of the network. The concept of *delegation of authority* has been proposed [Yemini 91] to address this issue. Delegation techniques require an appropriate infrastructure that provides a homogeneous execution environment for delegated tasks. One approach to the problem is SNMPscript [Case 93]. However, SNMPscript has serious restrictions related to its limited expression as a programming language and to the limited area of its applicability (SNMP only). Although *delegation* is quite a general idea, the static nature of management agents still leaves considerable control responsibility in the

domain of the manager and generally implies the availability of an accurate global view of the network. Legacy network management systems tend to be monolithic, making them hard to maintain and requiring substantial software and hardware computing resources. Such systems also experience problems with the synchronization of their databases and the actual state of the network; i.e, the network view is generally out of date. Although the synchronization problem can (potentially) be reduced in severity by increasing the frequency of updates or polling, this can only be achieved with further severe consequences on the performance of the system and the network.

An emerging technology that provides the basis for addressing problems with legacy management systems is network computing based on Java. Java can be considered a technology rather than merely as another programming language as a result of its 'standard' implementation that includes a rich class hierarchy for communication in TCP/ IP networks and a network management infrastructure -- the Java Management Application Programmer Interface, (JMAPI). Java incorporates facilities to implement innovative management techniques based on mobile code [Bieszczad 98]. Using this technology and these techniques it is possible to address many interoperability issues and thereby work towards plug-and-play networks by applying autonomous mobile agents that can take care of many aspects of configuring and maintaining networks. For example, code distribution and extensibility techniques keep the maintainability of networks and their management facilities under control. The data throughput problem can be addressed by delegation of authority from managers to mobile agents (or code, we use the terms interchangeably) where these agents are able to analyze data locally without the need for

any transmission to a central manager. We can limit the use of processing resources on network components through adaptive, periodic execution of certain tasks by visiting agents. The goal is to reduce, and ultimately remove, the need for transmission of a large number of alarms from the network to a central network manager. In other words, our research focuses on proactive rather than reactive management of the network. In particular, this chapter focusses on an important facet of Network Management; namely, the location and diagnosis of faults in the network.

While Java technology provides a device independent agent execution environment, the use of mobile code in Network Management and the use of groups of agents in particular, generate a number of issues which must be addressed. First, how is communication between agents achieved? Second what principles guide the migration patterns of agents or groups of agents moving in the network. Finally, how are groups of agents organized in order to solve network-related problems? These questions motivate the research reported in this chapter; the answers being provided by the use of SynthECA agents.

The remainder of this chapter is organized in the following way. The concept of a service dependency model is first introduced. Second, we briefly describe extensions to the mobile code infrastructure described in Section 2.2.4 on page 33 that facilitate device management. A SynthECA agent architecture utilizing mobile code for network management is subsequently introduced and briefly described. Following this introduction, algorithms for the localization of network faults are then provided, along with an exploration of their characteristics in a number of network scenarios.

5.3 Service Dependency Modeling

In order to drive the problem solving process -- that of fault location -- a model of faults, or a concept of services and dependencies between them, is required. However, it is our goal to have the fault model learned rather than provided to the system by an operator.

Within the context of this chapter, a network is said to provide services; e.g., private virtual circuits (PVCs). When a service is instantiated; e.g., a new PVC is created, it consumes resources in that network and subsequently depends upon the continued operation of those resources in order for the service to be viable. From a fault finding perspective, a service can then be defined in the following way:

$$S \alpha \{(R_i, p_i)\}$$

where S is the service, R_i is the i^{th} resource used in the service, p_i is the probability with which the i^{th} resource is used by that service and the relational operator means depends upon. A resource R_i might be a node, link or other service.

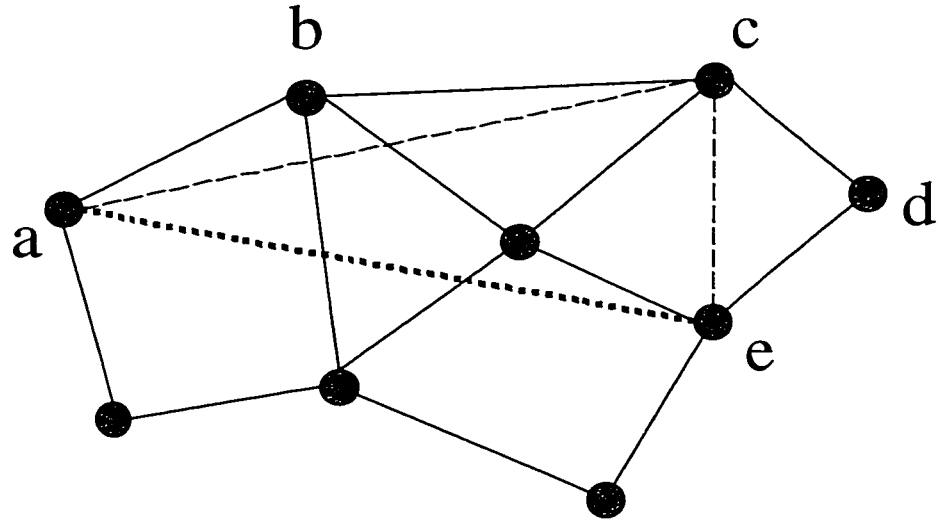


FIGURE 51. **An example virtual network**

For example, a PVC that spans part of a network might depend upon the operation of several nodes and T1 links. The links, in turn, might depend upon the correct operation of several T3 links that carry them in a multi-layer virtual network. An example of such dependencies is shown in Figure 51.

Three layers within a multi-layer virtual network are partially represented in the figure on the previous page. The link *ae* represents a PVC. This link depends upon links in the layer that supports it, in this case the T1 layer represented by links *ac* and *ce*. These links, in turn, depend upon links in the T3 layer. In the case of link *ac*, its dependencies include links *ab* and *bc*. The link *ce* depends upon the T3 links *cd* and *ce* for its operational definition. An agent-oriented solution to the PVC configuration problem can be found in [Pagurek 98] and [Boyer 99].

5.4 Agent System Architecture

In the system described here, SynthECA agents solve problems by moving over the nodes and links in a network and interacting with "chemical messages" deposited in that network. These messages are stored within VMCs and are the principal medium of communication used between both swarms and individual swarm agents. Chemical messages are used for communication rather than raw operational measurements from the network in order to provide a clean separation of measurement from reasoning. In this way, fault finding in a heterogeneous network environment is more easily supported. Also, chemical messages drive the migration patterns of agents, the messages intended to lead agents to areas of the network that may require attention. As we shall see later, this system

can be thought of as sitting on top of the routing system described in the previous chapter and represented by Figure 25 on page 108.

5.4.1 Agent Classes

The agent classes defined in the system described here are intended to implement an active diagnosis system [Ishida 96]. In active diagnosis systems, monitoring and diagnostic activity is undertaken by agents working in a distributed manner in a sensor network. The agents perform these activities on a timely basis rather than just when a fault is detected. Ishida also describes an immunity-based agent approach to active diagnosis that exploits the metaphor of an immune system for active diagnosis. In some sense, a fault finding system can be thought of as an immune system and agent classes as examples of B-cells and T-cells. In fact, SynthECA agents are characterized by the cellular metaphor rather closely as they consist of chemical reactions encapsulated within a cell membrane that consists of effectors and receptors.

The agent system described here consists of four agent classes. First, condition sensor agents (CSAs) are defined. A CSA is an example of a netlet [Bieszczad 98]. The function of a CSA [Schramm 98] is to measure one or more parameters associated with a given component and determine whether a specific condition is true or false. CSAs interact with VMCs on network components by measuring parameters associated with the network component; e.g. the utilization of links connected to the node or the utilization of the node itself. CSAs are adaptive and learn to (a) avoid components where no valid sensory information is available and (b) visit components more frequently that are likely to cause

the condition of interest to evaluate to true. While the first situation appears strange at first reading, it must be noted that we are dealing with heterogeneous networks where parameters supported by one vendor may not be supported or provided by another[1]. Therefore, it is likely that CSAs will be vendor specific or apply to a subset of all components in the network at best. Also, it is intended that our CSAs should be self-configuring. Being netlets, they are injected into a mobile code region from a network management workstation and are not directed to visit particular components. It is essential, therefore, that CSAs are capable of learning an applicable (to them) map of the network. A CSA's ability to modify the frequency with which it visits a component facilitates variable frequency polling of components. The more the condition for a CSA evaluates to true, the more likely the agent is to visit the component. However, we prevent a CSA from spending all of its time on a single component. In this way, CSAs spend more of their processing effort on components with potential performance problems rather than allotting equal time to all components. A CSA may also leave chemical messages on devices that it visits. In this way it is possible for two such agents, one for device type one and the other for device type two, to measure different parameters but generate the same chemical message for use by the fault finding agents. The separation of measurement from reasoning is clearly an advantage here.

It is worth noting that CSAs are capable of interacting with the old manager/agent schema for network management. This can easily be implemented using VMCs. For

1. A review of the private part of an SNMP MIB for a small number of devices confirms just how diverse devices can be.

example, an application that uses a local VMC and implements an SNMP protocol handler can be installed inside the MCD. Thereafter, it can act as an SNMP agent.

Another possibility that has been implemented within the MCE is a handler of an extension protocol [Pagurek 00]. The DPI protocol was chosen for implementation. The DPI protocol was chosen as it is a 'lightweight' protocol and avoids the BER encoding/ decoding that is part of SNMP. In this research, a VMC extension registers with an SNMP agent and, acting as an SNMP subagent, provides data in response to SNMP requests. This scenario is shown in Figure 52.

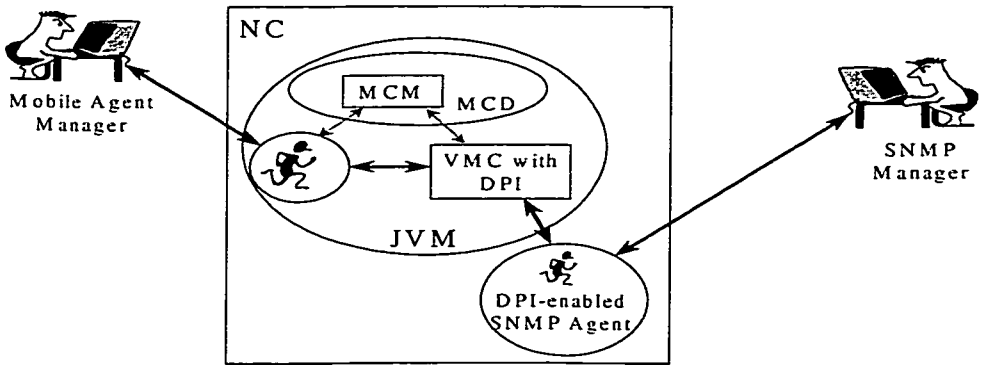


FIGURE 52. **A Sensor Agent talks to an SNMP Agent**

Both of these ideas could also be applied in situations where inter-working with a legacy system is required. It is possible to associate simulated network components with actual devices running legacy agents through properly engineered VMCs. This might be the situation where the actual device does not support a Java environment.

Second, Service Monitoring (SMA) and Service Change agents (SCA) are defined. A service monitoring agent is responsible for monitoring characteristics of a set of instances

of a service; e.g. the quality of service on one or more PVCs. These agents are static and reside where the service is being provided; e.g., at the source of a PVC. A service monitoring agent knows the service dependency model for the service. This model is computed during configuration of the service. This model, using the example of the previous chapter, is constructed by explorer agents as they move throughout the network. A service monitoring agent detects changes in the characteristics of the monitored service and, if the change is considered significant, a service change agent is sent into the network in order to mark the resources on which the service depends with a chemical message. The concentration associated with the chemical message reflects the change in value of the characteristic of the monitored service. If the change in the measured characteristic for the service is considered beneficial, a negative concentration will be associated with the chemical message; i.e., the chemical will be 'evaporated'. If the change in the measured characteristic for the service is considered detrimental to the service, a positive concentration will be associated with the chemical message; i.e., an existing trail will be reinforced or a new one created. Given that resources will be shared by multiple services, it is easy to see that the resources common to two services will see twice the change in chemical concentration when the SMA detects a significant change. *It is this process of chemical interference that allows localization of a fault to be inferred.* A simple example of chemical interference used for fault localization is shown in Figure 53. In this example, a fault on node E has resulted in degraded quality of service for the two connections present in the network. The SMAs for the two connections have detected the degraded quality of service and sent out service change agents to mark the resources (in this case

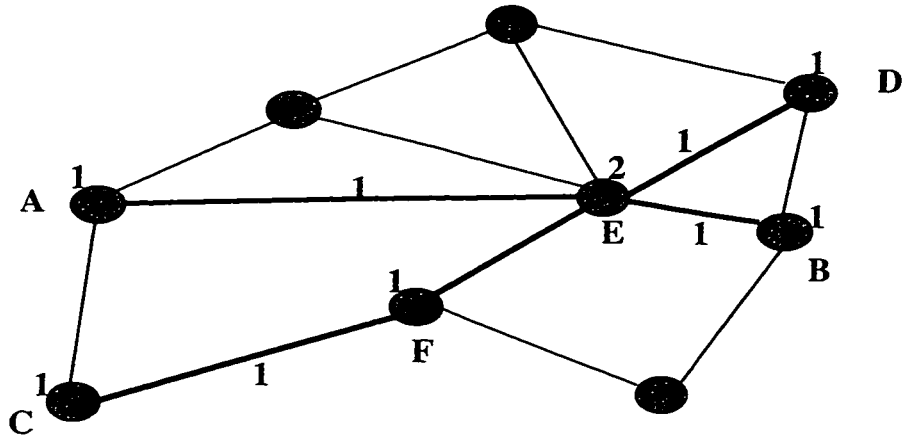


FIGURE 53. **Localization of a Fault by Chemical Interference**

nodes and links) that might be at fault. Figure 53 shows the concentration of a chemical message that represents the change in quality of service on the network nodes and links. Where a node or link has no associated chemical concentration, it means that it is zero. Figure 53 clearly shows that the highest concentration of the chemical is to be found at node E. This hints at the problem solving mechanism; namely, hill climbing in the space of increasing concentration of chemical associated with the fault.

Problem identification agents -- other netlets that circulate continuously throughout the network -- use the trail of chemical messages laid down in the network in order to determine the location of faults and to initiate diagnostic activity. These agents form the final class of agents defined. The value of communicating problems to network operators rather than a stream of alarms has long been understood [White 92], [White 96], [White 97]. In this previous work, a static knowledge base system has been developed where the knowledge base is composed of a set of problem classes with communication by

messaging between them. A problem class represents a model of one or more potential faults in the network. Instances of problem classes are intended as hypotheses regarding a fault in the network and a winner-take-all algorithm, where the instance explaining the most alarms is considered the most likely problem, is used to discriminate between competing hypotheses.

Mapping a single problem class to a problem agent, and using inter-agent communication for inter-problem message passing, seems a natural progression of this work. Rather than being alarm driven as reported in previous research, problem agents respond to the chemical messages laid down in the network and migrate from component to component based upon the concentrations associated with these chemical messages. In some sense, agents move to alarms (chemical messages in the network) rather than alarms being sent to a central manager for processing.

5.4.2 Problem solving by agents

Several problem agents have been implemented. First, a PVC *Quality Of Service* problem agent (qos-agent) has been built. As indicated in the previous section, these agents hill climb in the space of the chemical laid down by SCAs. The qos-agents circulate continuously in the network, moving and performing diagnostic activity at various of the nodes that they visit. At the beginning of our research, these agents would initiate diagnostic activity on a component when a concentration threshold was reached and this threshold implied that at *least* two SCAs had visited the component. This, however, has the potential for large numbers of incorrect diagnoses and was quickly discarded as a

viable heuristic even though the qos-agents do eventually find the faulty component. A variation of this simple heuristic was also tried, wherein the qos-agents adjusted their diagnostic thresholds up and down depending upon the number of incorrect diagnoses. While better than the simple heuristic described above by more than 50%, it was still considered inadequate.

A much-improved solution to the problem is the introduction of reinforcement learning techniques (see [Watkins 92], for example) to the agent architecture. A reinforcement learner is introduced at each node in the network and implemented as part of the VMC. The system state, Γ, associated with the reinforcement learner consists of the vector -- the s-vector, or s -- of concentrations of q-chemical on the node and connecting links, the actions, a, available to the system and the Q values associated with a vector of q-chemical concentrations. For example, in Figure 53, the vector (2,1,0,0,1,1,1) could be used to partially define the state of the node E and its network links. The actions available in a given state are to diagnose a component (node or link) or not to do anything. Diagnostic actions are also stored within the VMC [White 99d]. Action selection is based upon the Q value associated with the q-vector. The Q value is the long term expected utility associated with taking a particular action when in a given state and is quite different from the concentration of the q-chemical deposited in the network. The reinforcement signal within the system is provided by the SCAs. If the qos-agent selects the correct component for diagnosis, the SMAs will detect the change in quality of service and send SCAs into the network in order to modify the concentrations of q-chemical on the various nodes and links that form part of the circuit (more generally the service dependency model). This

change will, in turn, be sensed by the qos-agent residing on the node where diagnostic activity was initiated and it will increase the Q value associated with taking that action in that state. We assume that the fault correction activity, if initiated on the correct component, will be successful. Diagnosis is not the focus of this chapter, fault location is. We are assuming here that if the correct component is identified, diagnosis will occur and the fault will be corrected. This assumption implies that we are not at all interested in the way in which a particular node provides the service being monitored; the node is a "black box." If an incorrect component is chosen for diagnosis, two situations are possible. First, if we assume that diagnostic actions cannot make the quality of service of the connection degrade further, then changes of that kind that the qos-agent sees are not as a result of its actions. It does not use these signals to update the value associated with choosing that diagnostic action. They are assumed to be the result of a fault elsewhere in the network. We do not assume single faults in our system; several may be present in the network at the same time. Second, it is possible that no improvement in quality of service is seen by the SMAs whose circuits depend upon the component being diagnosed. In this situation, the qos-agent "times out" and applies a negative signal (reward) to the action associated with the initial state. It then attempts (up to) two further diagnoses before migrating to a new node. Should one of the remaining diagnoses improve the quality of service for the circuits depending upon the component diagnosed, the feedback is applied in a discounted fashion to the one or two diagnoses that preceded it. This apportionment of the reinforcement signal is done to take account of latency effects in the network.

In the equation on the next page, $Q(s,a)$ represents the long term discounted reward for

an agent taking an action, a, when in state s. $\Delta Q(s,a)$ represents the change in the Q value

$$\Delta Q(s, a)=\alpha(r+\gamma\, max_{a'}\, Q(s', a')-Q(s, a))$$

for the state, s, having taken the action a. Learning occurs through the modification of the Q values and by choosing actions that are biased towards higher Q values. The terms α and γ are constants that determine the learning rate of the system. Algorithmically, Q values are updated by taking a specific action in a given state and noting the change in the system state. The reward, r, is the observed change in the quality of service on the component and its associated links. Changes in quality of service are aggregated over a time window, t_w, which is the time in which improvements in quality of service are seen, up to a threshold value of t_{wmax}. An aggregation time window is needed in order to take account of the latency of the network; SCAs need time to lay down changes in concentrations of q-chemical. However, we do not wait forever as, in the case of an incorrect diagnosis, no change will occur. During the aggregation time window, it is the qos-agent that senses changes in the local concentration of q chemical. It is the qos-agent that applies the aggregate q chemical concentration changes as a change in the Q value for the state using the equation above; i.e., it is the agent that performs the feedback for the reinforcement learning system.

If a degradation in quality of service is seen during diagnosis, a reinforcement signal is applied to the state up to, but not including, that change. The reasoning here is that a successful diagnosis can only improve the quality of service for everyone and that no diagnostic activity causes quality of service to degrade further. Therefore, as soon as a

quality of service degradation is seen, it must be the result of a second, independent failure.

The *Q(s,a)* function is stored as a lookup table with a entry for every possible state-action pair. In this system, there are two actions associated with the node and each incident link, the actions being to diagnose the component or do nothing. We designate these d_i and $\overline{d}_i$ for diagnosis and non-diagnosis of the i^{th} component respectively. The table is initialized such that for a given state, s, the function $Q(s, d_i) = ks_i$ and $Q(s, \overline{d}_i) = \frac{\bar{k}}{\bar{k} + s_i}$ respectively. The values k and $\bar{k}$ are constants for the system, with $k > 0$ and $\bar{k} > 0$. This form of initialization for reinforcement learning we call Intelligent Initialization Reinforcement Learning (I^2RL). Normal initial procedures for reinforcement learning randomly assign Q values to the various actions. I^2RL was found to be 22% superior to this owing to the strong correlation between the concentration of q chemical associated with the component and the expectation of diagnosis. Generally, we would expect monotonically increasing functions for initialization of d_i and monotonically decreasing functions for initialization of $\overline{d}_i$ to provide improved algorithm performance when compared to standard initialization. Stated more simply, we would expect high concentrations of q chemical to be associated with a higher probability of diagnosis. Defining *dim(s)* to be the dimension of s; that is, the number of components diagnosable by the reinforcement learner, an action in any state, s, is chosen with probability, $p(d_i|s)$,

using the Q value associated with a given action, d_i or $\overline{d}_i$, using the distribution given by:

$$p(d_i|s) = \frac{e^{Q(s,d_i)} - 1}{\sum_i e^{Q(s,d_i)} - dim(s) + e^{Q(s,\overline{d}_i)}}$$

$$p(\overline{d}_i|s) = \frac{e^{Q(s,\overline{d}_i)}}{\sum_i e^{Q(s,d_i)} - dim(s) + e^{Q(s,\overline{d}_i)}}$$

As stated above, diagnostic actions are initiated by interaction with the component through a VMC. When such activity is initiated, and the diagnostic activity is successful as measured by improved quality of service, the concentration of the 'reliability' chemical, or r-chemical, is increased on the component. The amount of r-chemical deposited on the device is proportional to the time taken to receive the positive reinforcement signal; i.e., the time it takes for an SCA to visit the device and change the quantity of q-chemical. The reinforcement signal is positive if the SCA evaporates q-chemical and negative if it deposits q-chemical. If no signal is received within a period of time, it is assumed that an incorrect diagnosis has occurred and a negative reinforcement signal is applied.

The migration decision function associated with qos-agents depends only upon the quantity of q-chemical on the node and its incident links. Defining $S_{ijq}(t)$ to be the concentration of the q-chemical on the component from i to j at time t, the MDF for the fault location agent can be written:

$$p_{ij}(t) = \frac{S_{ijq}(t)}{\sum_k S_{ikq}(t)} \qquad \text{for f\% of the time, random otherwise}$$

Random migrations are made for (100-f)% of the time in order to ensure that the entire network is reached in reasonable time. A probabilistic choice, based upon *S* values, is made for f% of the time in order to revisit parts of the network that are experiencing poor quality of service. It should also be noted that oscillation between two high *S* components is explicitly prevented; i.e., a fault location agent cannot return to a previously visited network element for t migrations.

A *Chronic Failure* problem agent has been defined in the system that senses the c-chemical for the purpose of identifying components that experience multiple faults in short periods of time. The concentration of c-chemical is used within the migration decision function of explorer agents to determine where new connections should be made. In order that c-chemical concentrations do not increase unchecked, a CSA has been included in the system that periodically visits components and 'evaporates' c-chemical concentrations.

Finally, an *Overload* problem agent has been defined, but not implemented. This agent hill climbs in the space of the concentration of a chemical generated by CSAs that circulate in the network, monitoring component and link utilization parameters. Again threshold driven, it is intended that persistently over-utilized components are identified in order to facilitate re-planning of the network.

To summarize, SMAs detect changes in quality of service. Significant changes in quality of server result in an SCA being sent into the network to mark components in the service dependency model with concentrations of q-chemical. Qos-agents circulate in the

network, hill climbing in the direction of increasing concentrations of q-chemical. Probabilistically, they choose to initiate diagnostic activity on components. If a diagnosis is correct, the SMAs will see a significant improvement in quality of service and, once again, send SCAs out into the network in order to change the concentrations of q-chemical. The qos-agent diagnosing the fault will sense the improvement and use it to reinforce the Q value associated with taking that action in that state, thereby making a correct diagnosis more likely in the future.

5.5 Results

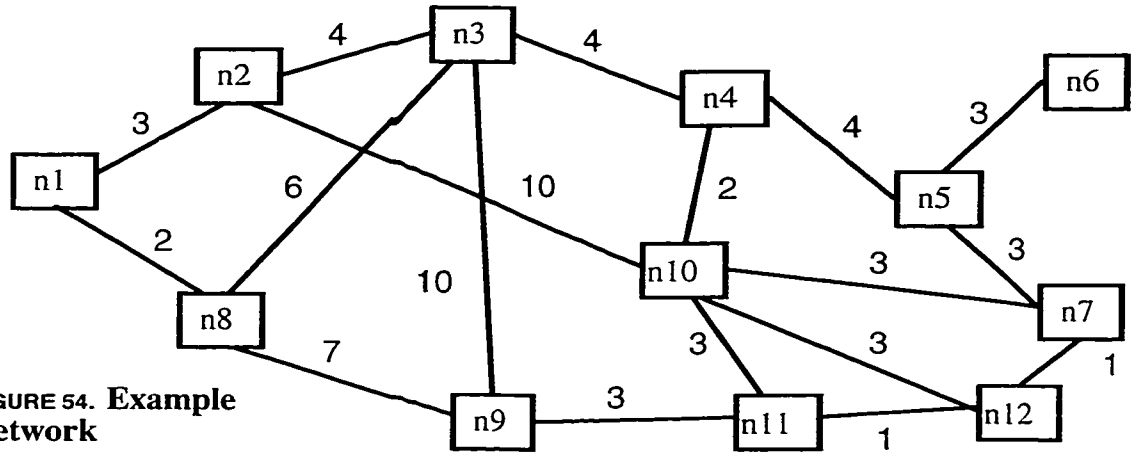


FIGURE 54. **Example Network**

The fault location system described briefly in the previous sections was applied to a number of small networks, one of which is shown in Figure 54. Each component in the network was assumed to have a probability of causing degraded performance of 0.1 and 5 distinct quality of service degradation levels were defined. The experimental setup and nature of traffic patterns that were applied to this network are defined in the previous chapter. Quality of service changes were randomly injected into the network in order to test the response of the system. A reinforcement learner was initialized using I^2RL on each

node such that the most likely action chosen for any state was the diagnosis of the component associated with the highest individual component of that vector (assuming > 2).

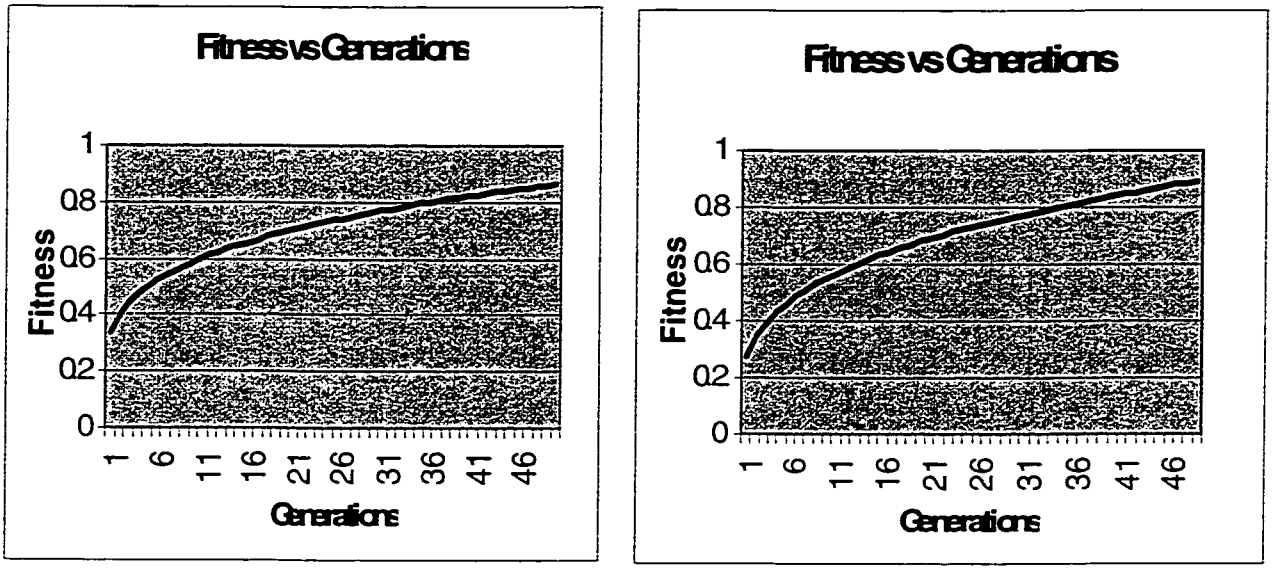


FIGURE 55. **Fitness results**

The number of qos-agents was varied from 1 to 5. The reason for this is that qos-agents acting independently can cause incorrect feedback to be seen by one another and thereby degrade learning performance. This is the so-called Tragedy of the Commons problem often observed in multi-agent learning systems; see [Wolpert 99] for example. While a single qos-agent would eventually visit and diagnose the correct component, this would lead to unacceptable fault location times in large networks. However, having too many agents causes inferior learning performance owing to the poor nature of the reinforcement signals. Increasing the number of qos-agents increases the probability that the successful diagnosis by one agent will be seen as a positive reinforcement signal by another. [Wolpert 99] provides a useful analysis of the properties of a MAS with reinforcement learning that

overcomes these problems. Examples of learning performance for two typical runs are shown in Figure 55 for two qos-agents in the system. The curves shown represent the trend in performance, not the raw experimental data.

Several experiments were performed with varying numbers of qos-agents in the system. For the size of network shown in Figure 54, two agents were found optimal in the sense that the converged performance of the reinforcement learners was superior to that of all other qos-agent configurations. The variation of converged performance with the number of qos-agents is shown is Figure 56. The difference in converged performance between one and two agents is small but two agents are slightly superior. In addition, the time to diagnose the location of a fault is lower.

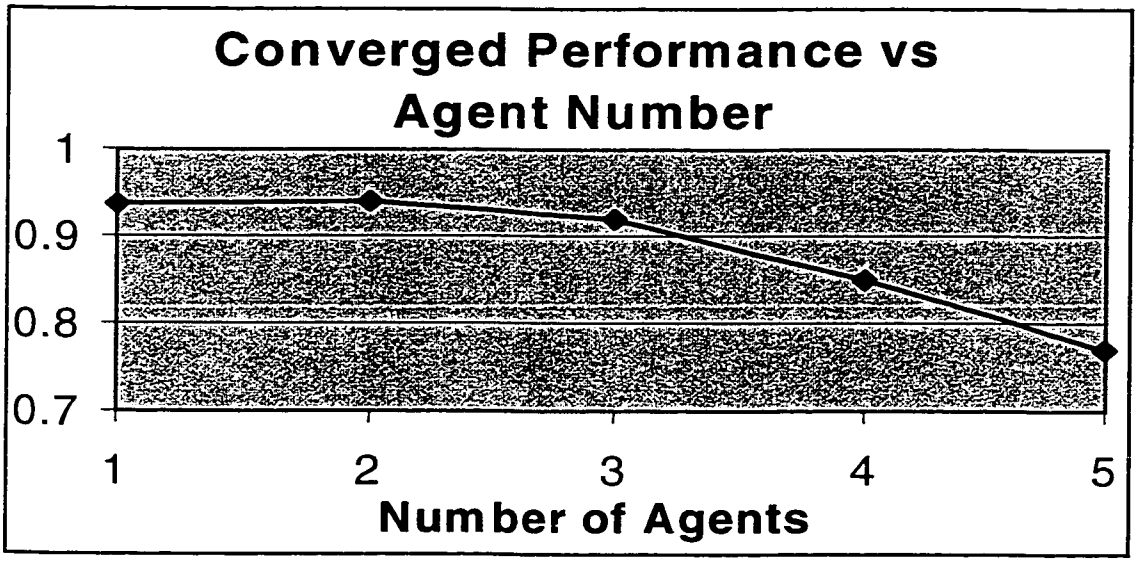


FIGURE 56. **Variation of Performance with Number of Agents**

5.6 Interaction with the Routing System

Referring to equations 10 and 11 on page 157, we observe that the explorer agent MDF depends upon the concentration of the reliability chemical (the r-chemical) on a particular

node or link. The higher the concentration of the r-chemical, the less likely an explorer agent is to use the path element, if all other things are equal. What this implies is that explorer agents will begin to avoid areas of the network experiencing poor quality of service. This, in turn, implies two things. First, if routes are static; i.e., they do not change once allocated, new connections being generated will tend to avoid the unreliable components. Second, if we allow explorer agents to continue to roam the network after routes are allocated, routes can be dynamic, resulting in path deallocation when sufficient concentrations of r-chemical are deposited on unreliable components. By allowing explorer agents to continue to explore the network, we allow the use of network resources to vary based upon the changing reliability of routing components.

The $R_{ij}(t)$ term in equations 10 and 11 on page 157 can be thought of as providing a coupling between the control and management planes in the network, with the flow of information moving from the management to the control plane. Another flow exists, this being the creation and destruction of q-chemical concentrations, and this flow moves from the control to the management planes. The $R_{ij}(t)$ term can be thought as providing an inhibitory signal in the control plane, while the q-chemical concentrations laid down by SCA agents provide an excitatory signal for the management plane. These flows, and their implied signals, create a two level subsumption architecture as described in the section on "Subsumption" on page 25.

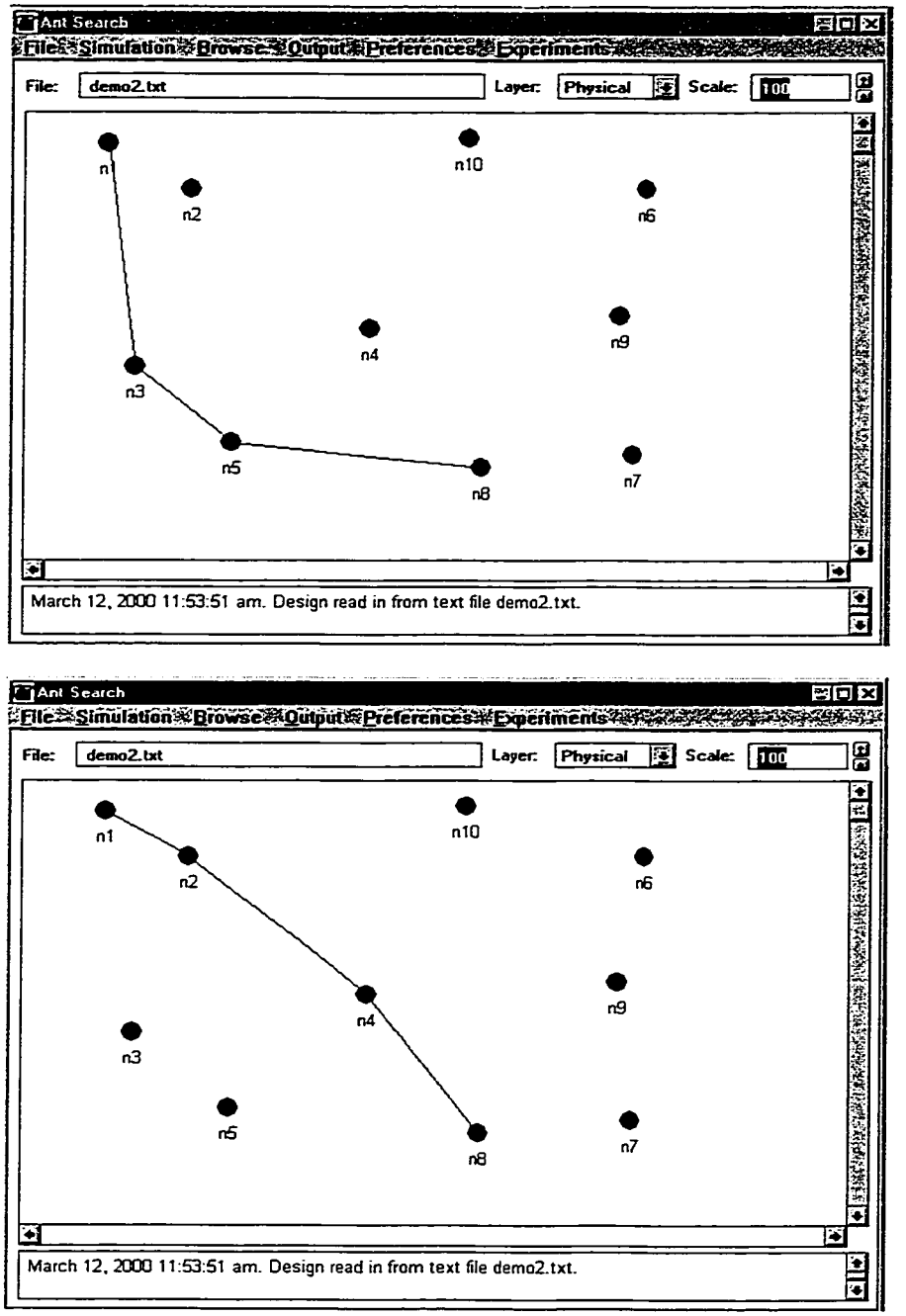


FIGURE 57. **Before and After routes for n3 'Failure'**

Figure 57 shows the before and after routes for a dynamically routed connection between n1 and n8. After n3 experiences significant quality of service degradation and

that degradation is corrected, the explorer agents for the connection detect that n1-n3-n5-n8 is no longer the shortest path (because of the r-chemical concentration laid down on node n3). This causes deallocation of the original path and the creation of a new path n1-n2-n4-n8.

It should be noted that the original explorer agents are in no way made aware of the activities of fault location agents, they merely sense the concentrations of r-chemical laid down once the quality of service degradation has been corrected. The concentration of r-chemical has acted as an inhibitory signal with respect to n3 being used as part of the route for the connection between n1 and n8.

In order to ensure that a route is only recalculated when necessary, a route is not recalculated until the percentage of returning explorer agents falls below a deallocation threshold. In the previous chapter, an allocation threshold of 90% was used with the associated deallocation threshold set at 50%. The dead band (50-90%) ensures that we do not oscillate between routes; highly desirable given the expensive nature of the connection-finding process.

5.7 Summary

This chapter has presented a multi-agent system for network management that relies on Swarm Intelligence and, in particular, trail laying behavior in order to locate faults in a communications network. This architecture promotes the idea of a clear separation of sensing and reasoning amongst the classes of agents used and promotes the idea of active, or collective, diagnosis. A chemically inspired messaging system augmented with an

exploitation of the ant foraging metaphor have been proposed in order to drive the mobile agent migration process. The chapter has demonstrated how fault location determination can arise as a result of the trail-laying behavior of simple problem agents. An implementation of this architecture has demonstrated that mobile agents can be effectively used to find faults in a network context. The service dependency model concept, along with the introduction of reinforcement learning techniques for the learning of models of fault location, have shown that global models of the network need not be provided in order that effective fault location can occur. However, our research has observed the interaction between multiple qos-agents and our future work will consider mechanisms based upon Wolpert's Collective Intelligence (COIN) research in order to overcome this.

CHAPTER 6

Management of Distributed Agents

6.1 Overview

This chapter concerns itself with the management of agents moving throughout a network for the purpose of solving problems using distributed computation. In any system that supports distributed computation in an unreliable network, there is a need to address issues of agent density and upgrading.

An excessive number of agents in a network can significantly degrade the functioning of that network, while too few may also compromise performance. Generally, it is sufficient to maintain agent density within a range, a single point value is unnecessarily restrictive[1]. In an unreliable network we find that links or network nodes may fail with a resulting loss of any agents executing on the node or in transit between two nodes. Some might argue that the computing infrastructure should support reliable computation and transport. However, this adds significantly to the complexity of the mobile agent infrastructure

1. In fact, point control is often the cause of oscillatory behaviour in a controlled system.

required and seems contrary to the philosophy of this thesis where tolerance to individual agent loss is a goal. Maintaining accurate statistics on agent numbers and position is really unnecessary for solving this problem if we view it from a decentralized viewpoint. In fact, load balancing generally may be viewed from this same point of view as we will see in a section presented later in this chapter.

It is also too commonly the case that software, and agents are not likely to be exceptional here, is flawed either logically or functionally. This necessitates software modification of individual agents or complete replacement of the agent as these flaws are discovered and new agents are constructed. A computing infrastructure supporting agent versioning is a more challenging problem still, in that we know neither the positions of individual agents nor the versions actively moving through the network. This presents a serious halting problem in that we do not know the numbers of particular versions of the agent that are active in the network. It is not possible, then, to know when the upgrade process is complete. In other words, any algorithm or solution technique should be capable of upgrading older versions of agents for all time.

It is difficult to conceive of an environment that automatically upgrades agent software when changes are available which does not rely on some form of global information. For example, the current mechanism for software upgrade used on personal computers is to check periodically with the supplier of the software and download improvements when available. While mirror sites partially solve the load balancing problem, the solution is still a central one, one in which global information is held on the supplier's web site. Should the supplier move or disappear completely, the upgrade process fails. This,

obviously, is an inferior solution. A decentralized solution to the versioning problem would have no such limitation, relying instead on only local information.

This chapter proposes the use of algorithms that exploit the SynthECA formalism, relying exclusively on local information and the emergent behavior of large numbers of agents.

6.2 Density Control

The importance of having an appropriate number of agents in a network performing a given task cannot be overstated. For example, in the routing problem described earlier, it was noted that if too few agents were sent out into the network, the effects of pheromone evaporation dominated reinforcement of the pheromone concentration causing no clear route to emerge. This can be countered by forcing convergence towards a route through increasing the frequency of agent generation (or increasing the number produced in a single generation). This momentum effect ensures the timely convergence of the algorithm to a minimum length route.

The above scenario is a less interesting example of density control when compared to the general situation with agents moving through the network and never terminating their problem solving activity. It is less interesting in that density control resides on the source node where a single agent is responsible for generating explorer agents; in some sense control is centralized. It is also straightforward in that the reliability of the network is unimportant as a single lost explorer agent will not affect the overall route obtained. All that may happen is that explorer agents may not reach their intended destination or will fail to return to the source node, thereby only partially applying their reinforcement signal

for a particular route.

Consider, then, the problem of multiple swarms of problem solving agents in an unreliable network. Clearly, if steps are not taken to inject new swarm agents over time, agent density will tend to zero. The argument is straightforward. Assuming that agent movement is random and uncorrelated, given a non-zero component failure rate, λ_f, following a Poisson distribution, a network of n nodes, with m agents, the number of agents failing per unit time is: $m\lambda_f/n$. This expression, being unconditionally greater than zero given n,m greater than zero, ensures that the probability that the number of agents in the network at time t_1 is less than the number of agents in the network at time t_0, $t_1 > t_0$ is one.

Having established the need for agent replacement, a mechanism for that replacement is required. We propose the addition of a Density Control Agent (DCA) class. The purpose of the density control agent class is to circulate continuously in the network depositing chemical signals in that network such that agent classes whose density is being controlled will automatically adjust their numbers to meet the target density range. The DCA class uses a random migration decision function in order to explore all parts of the network equally and is responsible for controlling its own density. It also remains at each node for a randomly generated residency period chosen from a uniform distribution in order to avoid correlations between agent actions. This is an extremely important observation as, without it, significantly greater oscillations are observed with possible population extinction. Every agent class that is density controlled generates a visit chemical that is sensed by the DCA. The DCA controls its own density in order to solve the problem of

managing management class agents. Therefore, the DCA also generates a visit chemical. Visit chemical concentrations are associated with the node. The visit chemical leaves a trail of activity for the density-controlled problem solving agents that is integrated across a number of network nodes by DCA agents for the purpose of generating birth or death signals that are in turn sensed by the density controlled problem solving agents. Birth or death signals are, naturally, chemical in nature and these chemicals do not evaporate.

DCA agents generate birth signals when the aggregated visit chemical concentration for a particular density control problem solving agent class falls below a threshold value. Visit chemical concentrations evaporate over time, this forming the dissipative field that makes the density control mechanism work. This is a crucial part of the control process as, without it, visit chemicals would accumulate forever leading to the rapid extinction of the entire agent population. In fact, we make use of this observation in solving the agent upgrade problem described later in this chapter. DCA agents generate death signals when the aggregated visit chemical concentration for a particular density control problem solving agent class exceeds a threshold value. Both types of signals are generated on the node where the appropriate threshold condition is violated. An exponential averaging process is used to aggregate visit chemical concentrations.

When a density controlled problem solving agent senses a birth signal, it clones itself, generating a new agent with the parent agent consuming the birth signal. When a density controlled problem solving agent senses a death signal, it chooses to die according to a probability distribution, having first consumed the death signal.

6.2.1 Results

In order to demonstrate the utility of the above algorithm, the two networks used for routing experiments were revisited for density maintenance. Two classes of agent, including the DCA and both with random migration decision functions, were allowed to circulate within the network. A single agent of each class was injected into the network and allowed to stabilize to the "natural" value for the network. In later experiments, extra agents were injected into the network periodically in order to see if the density correction algorithm could return the density to the appropriate value for the network. This had the added side effect of ensuring that at least one DCA agent would be present in the network.

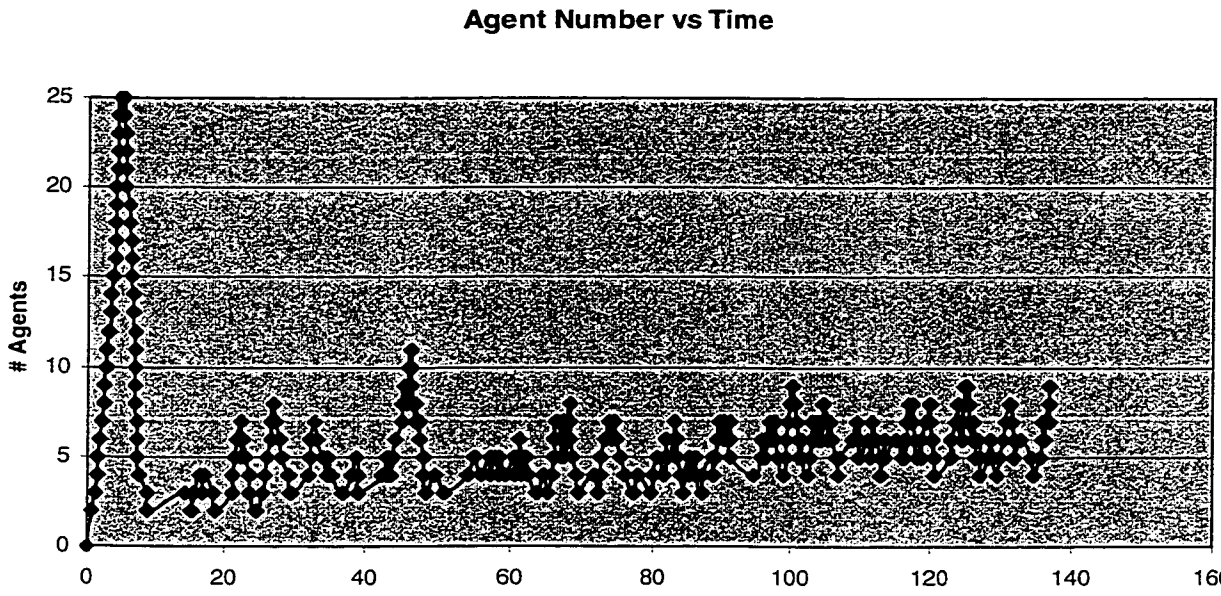


FIGURE 58. **Agent Management Graph 1**

The minimum concentration threshold value was set at 1, the maximum at 5. The rate at which visit chemical was deposited on the node was 1.5 units per visit, the evaporation rate was set to 0.8 units per simulation time step.

Figure 58 above shows the variation of agent number with time for graph1 used in the

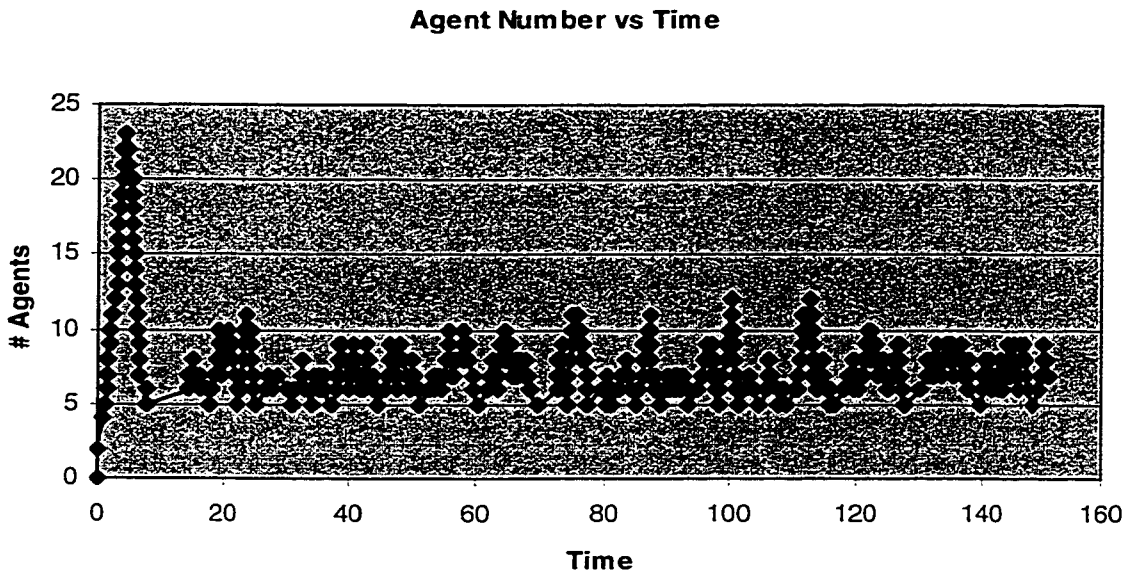


FIGURE 59. **Agent Management Graph 2**

routing experiments previously described. The agent type plotted represents a very simple agent that has a random migration pattern and is designed to measure the concentration of visit chemical, nothing more. The focus, in this study, is the management of agent number, and not problem solving per se. The variation of agent number with time represents the self regulation of agents in the system, no further agents beyond the initial seed agent were injected into the network.

The rapid rise in agent number initially is due to the fact that the network contains no visit chemical traces for the agent type. As a consequence of this, birth signals will be generated for all nodes visited and a large number of new agents will be generated. Avoiding this transient is possible by injection of the DCA *after* the network has stabilized, i.e., after the problem solving agents have had time to colonize the network and lay down visit chemical traces that mark their presence in the network. Even with the start

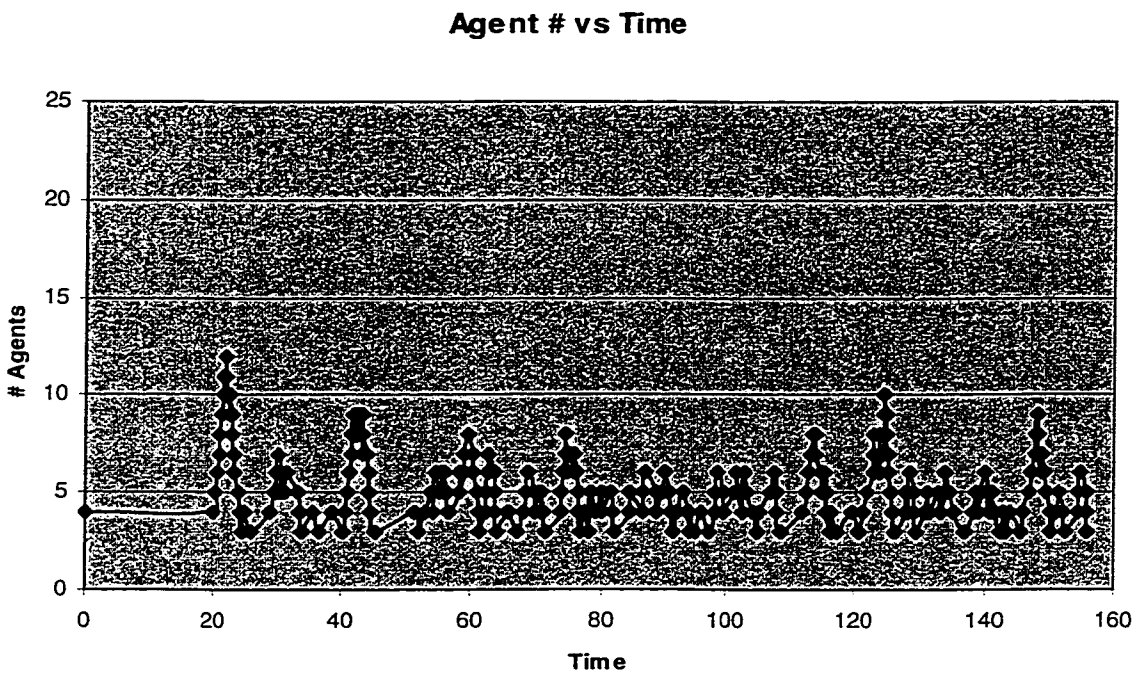


FIGURE 60. **Agent Management with Transient Initialization for Graph 1**

up transient, the network quickly recovers and settles down to a number of agents that oscillates around 5. Similar behavior can be observed in Figure 59, where agent density control is applied to graph2. In the experiment shown in this figure, the agent number oscillates around 7, with the number of agents never falling below 5. The oscillation can be further damped by choice of exponential averaging constant and by adjustment of the minimum to maximum visit chemical concentration threshold ratio. In the experiments charted in Figure 58 and Figure 59, a ratio of 5 was used to control the agent density.

The dynamics of this system have very similiar characteristics to Lotka-Volterra [LV 25] multi-species competition systems, systems which display similiar initial transients and oscillatory steady state behaviour. This should not be at all surprising as the DCA agents act as predators by providing a death signal and by providing "food" in the form of a birth

signal. These systems do not display asymptotic stability (i.e. converge to an attractor) but follow a trajectory enclosing one, the system never forgetting its initial conditions.

Figure 60 and Figure 61 demonstrate the utility of having density control come online once the network has been colonized by problem solving agents. In these experiments, density control was disabled during the first 20 time units of the simulation. The difference between the experiments charted in the two figures is that Figure 60 injects 4 DCA agents initially while Figure 61 injected 6 DCA agents at the start of the experiment. Interestingly, Figure 61 shows the system moving from one stable state to another at approximately 72 time units. Continuing the simulation beyond 160 time units saw no further changes in state. Contrasting the dynamics of this system with those displayed in Figure 58 clearly shows a much smaller transient and more rapid stabilization to the steady state network behaviour. The number of agents injected initially seems to make

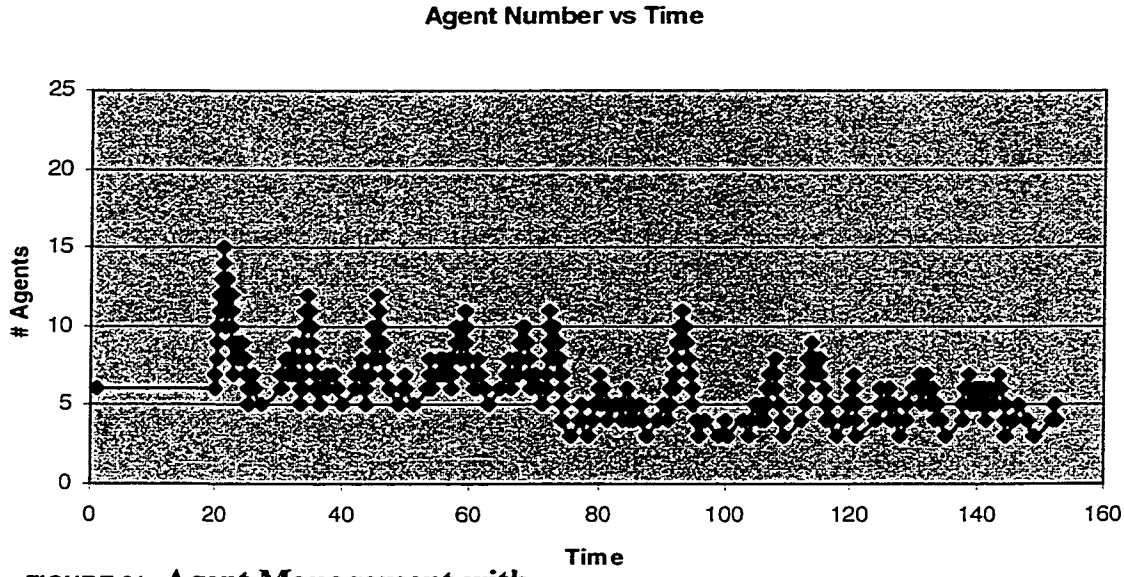


FIGURE 61. **Agent Management with Transient Initialization using 6 agents for Graph 1**

minor differences to the final stable network state. Experiments were run wherein up to 30 agents were injected initially; the system still converged to a mean number of agents of 6.

Experiments were conducted where agents were occasionally injected into the network in order to test the stability of the agent density management algorithm. As Figure 62 shows, injecting agents after the network has settled down merely causes the agent density to find a new stable point, which may, of course, be the same as the original point. This is to be expected in that many systems have several basins of stability. This characteristic is an attractive feature of the system in that we can alter the stable system trajectory by injection (or removal) of agents. In fact, as suggested earlier, we would propose the periodic injection of a density control agent in order to ensure that the system never remains locked at zero population for that agent. Figure 63 shows the destruction of agents

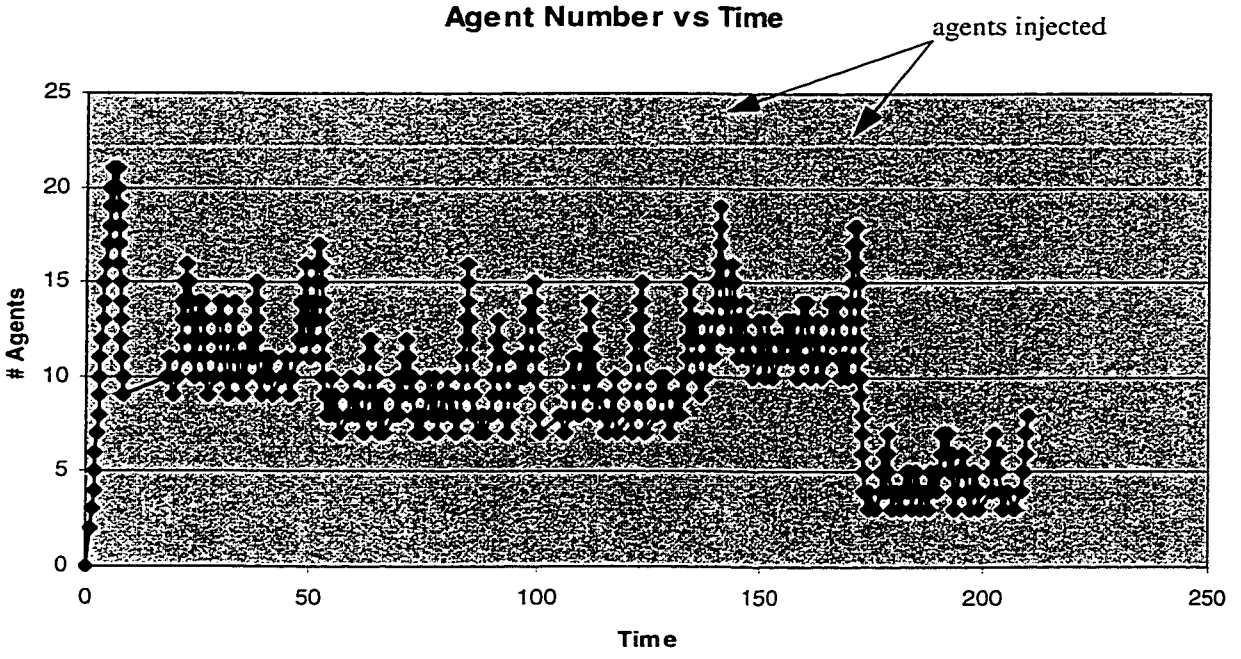


FIGURE 62. **Agents Injected after Settling Period for Graph 1**

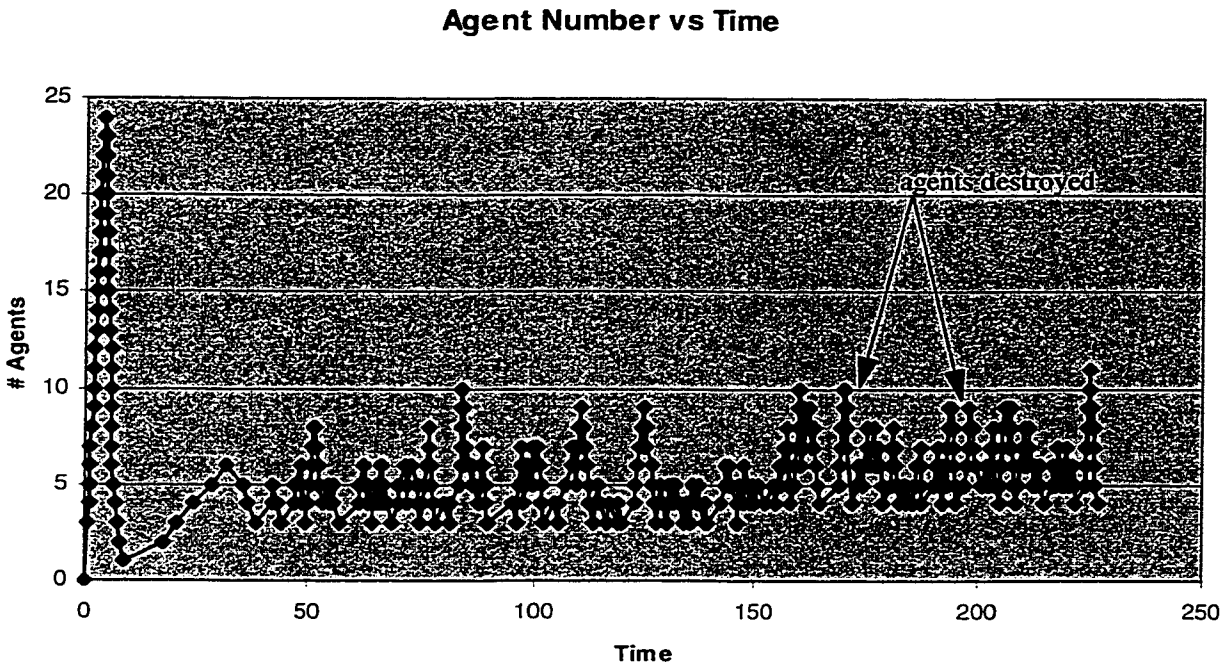


FIGURE 63. **Agents Destroyed after Settling Period for Graph 1**

in the network after the settling period. This scenario represents the situation that initially inspired the density management algorithm, namely the loss of agents as a consequence of network component failure. Figure 63 shows two failures, at approximately 175 and 200 time units, where multiple agents are lost. Clearly, the algorithm has performed well, with the natural trajectory of the system being quickly restored. Obviously a failure of *all* components in the network would cause the loss of all agents; however, the scenario demonstrated in Figure 63 actually represents a failure of 20% of the network which, in all likelihood, represents an extreme case.

More general experiments, with random single node failures, provided equivalent support for the robustness of the density control algorithm and results are not included here as, it was felt, the scenario described in the previous paragraph provide a more

dramatic illustration of the robustness of the algorithm.

6.3 Agent Upgrading

It is unfortunately the case that agents, or more generally software, never have completely correct behaviour when first deployed. The problem of upgrading software in an operational environment is challenging and is currently the focus of considerable research [Ning 99], [Gang 00]. The software upgrade problem presents a unique challenge when the software is an agent and that agent is mobile, as we have no knowledge a priori of the location of any agent. The software upgrade problem is further complicated by no knowledge of the number of agents to be upgraded and their source of injection into the network. This latter piece of information is important as it implies that older, incorrect versions of a software agent may be injected into the network once the upgrade process has been supposedly completed. Together, these problems present a significant research problem.

In the afore-mentioned research, the upgradeable software (agent) is constructed using a number of well-known Design Patterns [Gamma 95] and relies on standard client-server interactions. Unfortunately, this mode of interaction does not apply to the systems of interest in this thesis and reduces the utility of the research reported within the context of swarm systems. However, the research does describe two distinct types of swapping: application and module. While [Ning 99] and [Gang 00] focus on module-level swapping, this thesis chooses to adopt application level swapping as a consequence of the mode of interaction of the agents in the network.

The reasoning behind this choice is clear. This thesis is concerned with the interaction of

swarms of problem solving agents with the solution process being independent of the actions of a single agent. The algorithms proposed in this thesis are, as stated previously, robust with respect to the failure of an individual agent. Hence we may view the agent upgrade problem as one of "failing" the faulty agent and injecting one with corrected behaviour.

Not knowing the source of the agent originally injected into the network leads to the disturbing and inevitable conclusion that the upgrade process is potentially never complete but has phases: dormant and active. The words "dormant" and "active" are deliberately chosen in order to invoke an easy analogy with the human immune system. The human immune system relies on a recognition of self and non self, using as markers specific proteins that are attached to the exterior of T and B cells in the body. Continuing with the view of the previous paragraph, and the language of the immune system, faulty agents become the viruses to be detected and destroyed. Faulty agents, then, become the non self, with the corrected agents representing self.

Referring now to the previous section on density control, and using the above immune system analogy, a potential mechanism for removal of the faulty agent is to exploit its response to the death signal and visit chemical concentrations. That is, increase the levels of the visit chemical for the faulty agent such that the death signal for that agent is generated throughout the network by an associated DCA agent. This is achieved by having different visit chemicals for the faulty and corrected agents with the encodings so chosen that the faulty agent senses the visit chemical of the correct agent. Hence, fault agents will tend to see higher concentrations of visit chemical when compared to the corrected agent;

the corrected agent will tend only to see its own. An example best illustrates this.

Consider two agent versions, v_f and v_c, representing faulty and corrected versions respectively having visit chemical encodings ####1 and ###11 respectively. Assuming the Binary Array Chemistry of order 5, if we were to have concentrations c_f and c_c of the two chemicals, the faulty agent would sense $c_f + c_c$, whereas the corrected agent would see c_c. In other words, the faulty agent would see higher concentrations of visit chemicals and would be more likely to see the death signal as a result of exceeding the upper bound on visit chemical concentration. Similarly, the faulty agent would be less likely to see the birth signal as a result of higher visit chemical concentration.

This example can easily be extended. Consider a perfect agent that corrects faults in agent version v_c, say v_p. Let v_p have the visit chemical encoding ##111 and concentration c_p. Again assuming the Binary Array Chemistry of order 5, if we were to have concentrations, the faulty agent would sense $c_f + c_c + c_p$, whereas the corrected agent would see $c_c + c_p$. In this case, both v_f and v_c would experience a reduced number of birth signals and elevated number of death signals. As the number of v_p agents increases, the tendency is for the number of v_f and v_c agents to decrease, eventually causing the imperfect agent types to disappear.

This mechanism works because of the encoding scheme used for the three agent types. It relies upon the masking of the existence of previous versions of the agent through the increasing number of bits being specified as we move to higher and higher versions of the agent. Obviously, this limits the number of versions that might be accommodated with a

Binary Array Chemistry. However, a potentially infinite data structure such as can be provided by the Binary Tree Chemistry removes this limitation. Obviously this is a more expensive solution from a computational viewpoint -- pattern matching by bit position versus matching by subtree -- but does solve the more general problem when an unbounded number of agent versions need to be supported. The results in the next section deal with a Binary Array Chemistry only.

6.3.1 Agent Upgrading Results

This section presents experimental results that demonstrate the applicability of the agent upgrade algorithm. As indicated earlier, this algorithm exploits the masking of chemical concentrations of one version of the agent by another. In the results presented below, the two networks used in the density control experiments were also used as a simulation testbed for the agent upgrade algorithm.

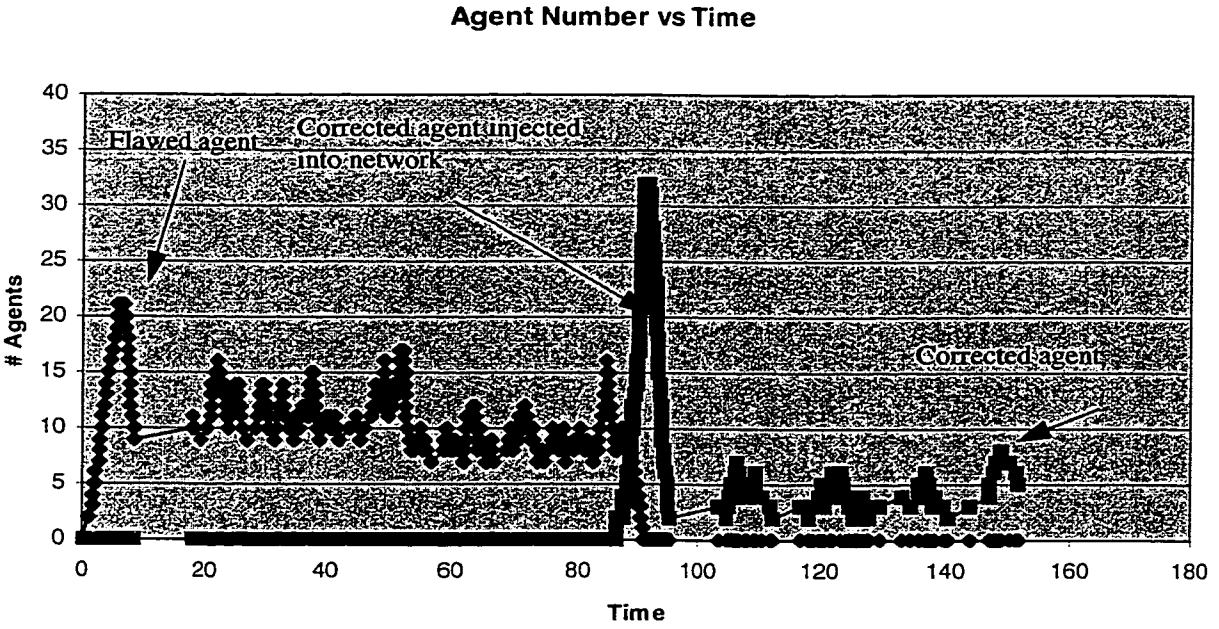


FIGURE 64. **Corrected Agent Replaces Flawed Agent**

In Figure 64, three flawed agents are injected into the network and are subject to the density control algorithm. Eventually the number of agents stabilizes around seven. At 83 time units, a single corrected agent is injected into the network and quickly establishes dominance over the flawed agent using the visit chemical masking mechanism introduced in the previous section. It is not possible for the population of flawed agents to recover once the number of flawed agents reaches zero as the birth process is one of cloning. Only through injection of a new flawed agent into the network from some external source can the flawed population temporarily recover.

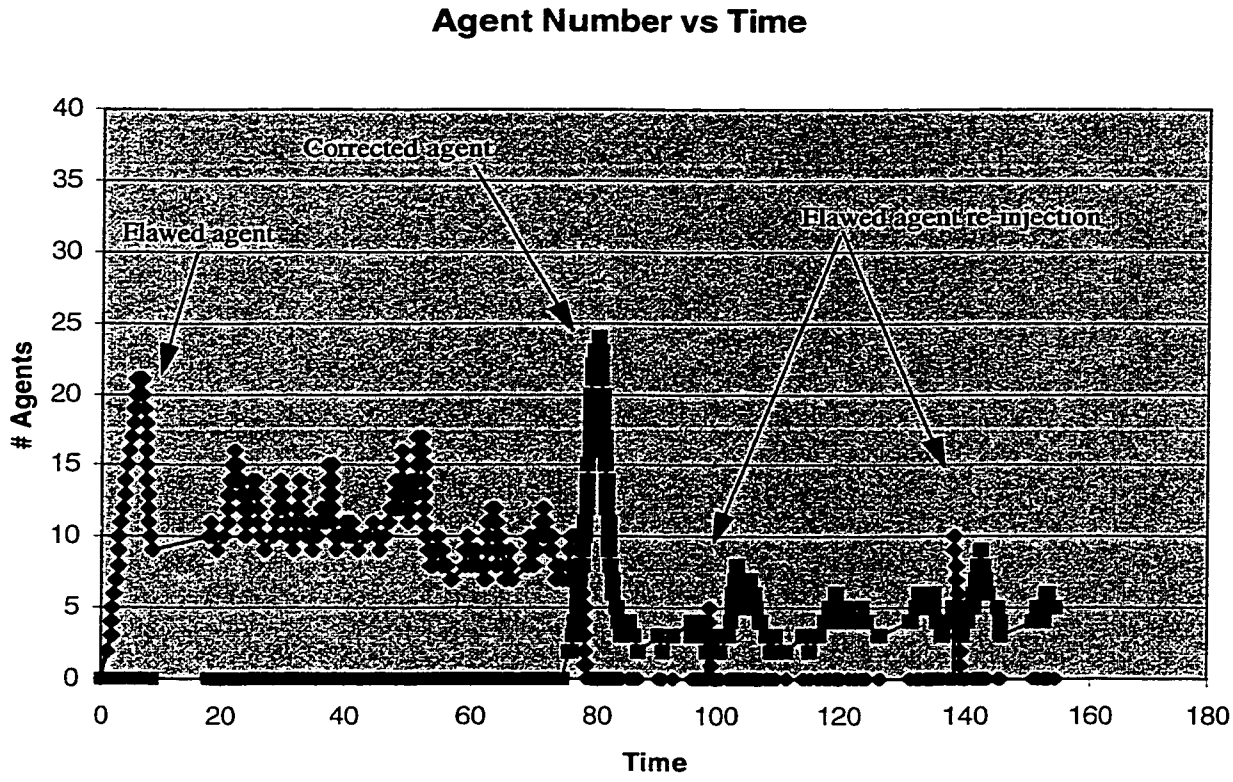


FIGURE 65. **Corrected Agent Replaces Flawed Agent After Re-injection**

In Figure 65, the flawed agent type is re-injected into the network following the colonization of the network by the corrected agent. The flawed agent is quickly removed

from the network on both occasions despite (relatively) large numbers of them being injected. While results are not included here, the number of flawed agents re-injected did not affect the final state of the network; merely the time to achieve it. In all cases, the flawed agents were eventually removed from the network, leaving only the corrected agent population. Clearly, this demonstrates that the corrected agent resists recolonization of the network by the flawed agent. The rejection of the flawed agents by the network is faster upon re-injection as a result of the established population of corrected agents.

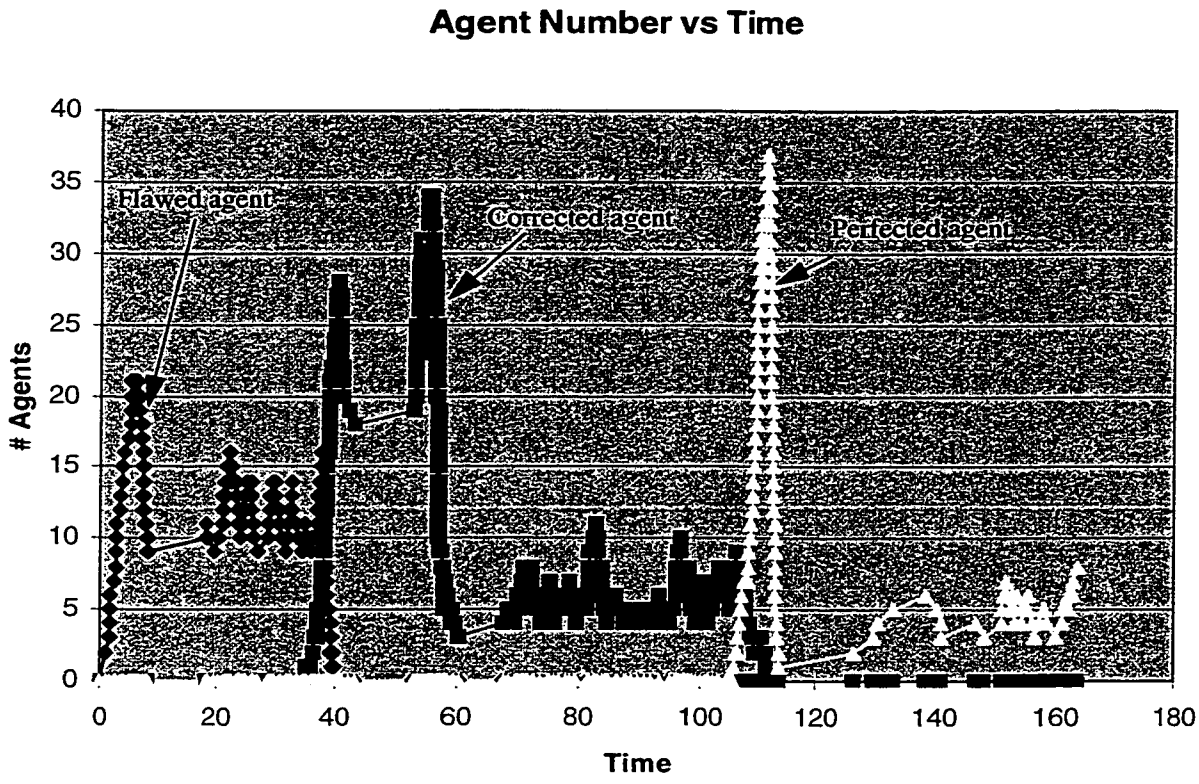


FIGURE 66. **Perfected Agent Replaces Flawed Agent and Corrected Agent**

In order to demonstrate the robustness of the algorithm in the presence of multiple agent versions, experiments were conducted wherein a third version -- the "perfected agent" -- was injected into the network after the corrected agent population had replaced the initial

population of flawed agents. As can be seen in Figure 66, the perfected agent quickly established a dominant position in the network causing the corrected agent population to vanish completely.

Once again, subsequent re-injection of flawed or corrected agent populations did not affect the dominant position of the perfected agent. This is clearly shown in Figure 67 where corrected and flawed agents are re-injected into the network following the perfected agent establishing itself in the network. Reviewing the interval 25 to 45 time units in Figure 67 also shows that the algorithm can deal with multiple agent populations simultaneously trying to establish themselves. During this time interval, flawed, corrected and perfected agent types are moving throughout the network; however, the corrected

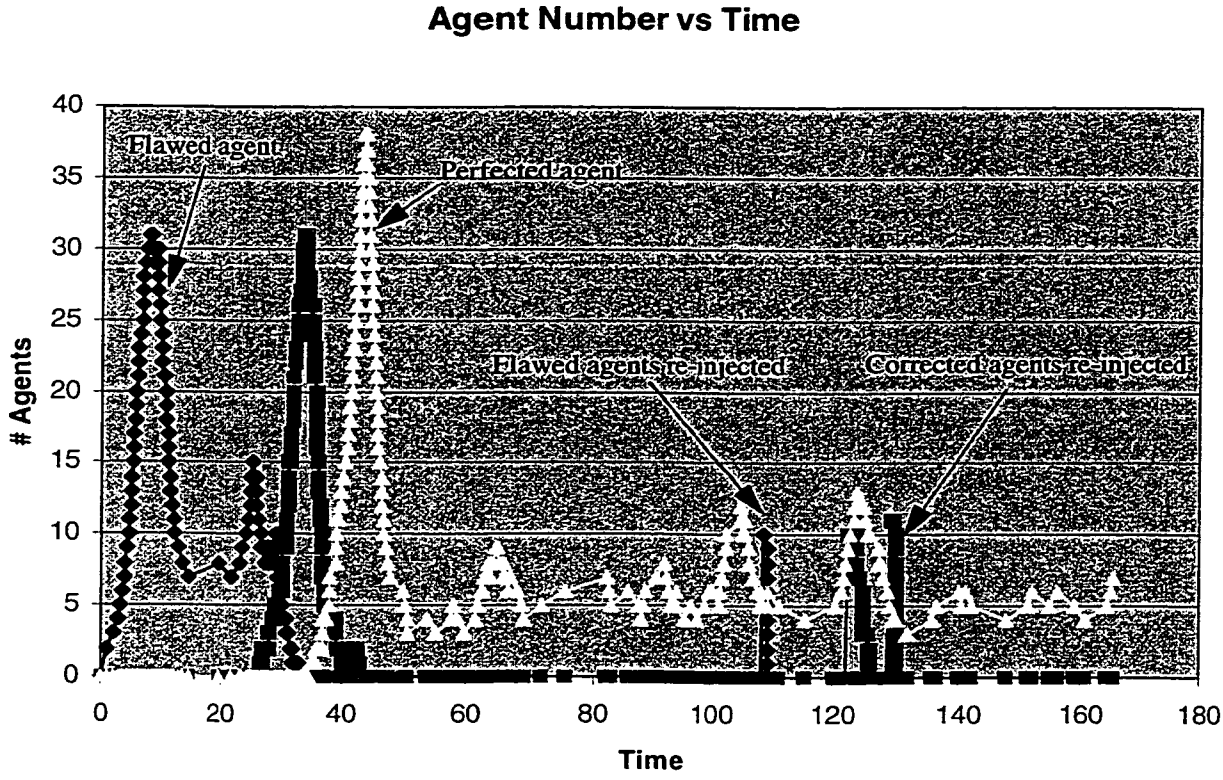


FIGURE 67. **Perfected Agent Replaces Flawed and Corrected Agents After Re-injection**

agent population is quickly extinguished even before stabilization can occur. Despite the version control algorithm's simplicity, it seems to be remarkably effective in maintaining the correct dominant version in the network.

While the above results have demonstrated the effectiveness of the upgrade algorithm, a comment regarding the rate of production of visit chemicals for successive versions of the agent needs to be made. As stated previously, given two agent versions, v_f and v_c, with encodings as previously described, v_f will see higher concentrations of visit chemical when compared to v_c. If the rates of production per nodal access of visit chemical are r_f and r_c for faulty and corrected agents respectively, and the visit chemical sensitivity ranges are (l_f,u_f) and (l_c,u_c) respectively, we can accelerate the removal of faulty agents by setting the corrected agent visit chemical production rate such that $r_c > u_f$. If this relation holds, the faulty agent will tend to see quantities of visit chemical that exceed its upper density bound, resulting in the generation of the death signal for the faulty agent class in almost all instances. The word almost applies here as we have to allow for the effects of evaporation in our system. Between the departure of a corrected agent and the arrival of a faulty agent evaporation can reduce the concentration of the visit chemicals sensed by the faulty agent below u_f.

In order for the corrected agent population to maintain density in a similiar way to the flawed agent population with the relationship $r_c = u_f$ holding other parameters have to be modified too. If we assume the rate of evaporation remains unchanged, the following relationships have to hold:

$$u_c = \frac{u_f r_c}{r_f}$$

$$l_c = \frac{l_f r_c}{r_f}$$

6.4 Summary

In a network managed completely by swarms of mobile agents, two important observations need to be made. First, the number of agents is unknown and second, the positions of mobile agents are unknown. These characteristics make management of the swarm populations extremely challenging.

This chapter has addressed two questions related to the management of mobile agent swarms.

The first question relates to the maintenance of population density and we have presented algorithms that maintain population density in a network having unreliable components. The algorithms presented rely on only local knowledge and we have shown by experiment that populations quickly settle to a mean population around which they oscillate. The oscillation is natural and unavoidable, having very similiar characteristic to the dynamics observed for Lotka-Volterra predator prey systems. The algorithms present are robust with respect to the introduction of agents after the network has stabilized as well as loss due to component failure.

The second question deals with the upgrading of agents over time. Agents, being software entities, rarely have correct behaviour when first introduced into service and often require upgrades. This chapter describes a number of algorithms that solve the upgrade problem by taking advantage of the density control algorithm in a form of

parasitic behaviour where a corrected agent "fools" the flawed agent population into believing that there are more of them than there are; the flawed agent population quickly decaying to zero. The upgrade algorithm, given an unbounded data structure such as is provided by the Binary Tree Chemistry, is capable of solving the problem of an infinite number of versions and is resilient to the re-introduction of older agent versions.

CHAPTER 7 *Conclusions and Future Work*

7.1 Introduction

Mobile agents are currently thought to provide a new type of distributed problem solving paradigm [White 99c] where simple agents interact and solutions emerge. This thesis has introduced a novel way of looking at distributed problem solving by exploitation of several physical and biological systems. The integration of ideas from social insect behaviour with chemical computation has provided a powerful problem solving framework, the utility of which has only been briefly explored in this thesis. Its motivations for choosing the systems exploited arise from the network problems solved in the previous chapters. In a wider problem domain, however, it would no doubt prove desirable to exploit elements of other naturally-occurring systems. Nature, after all, provides striking examples of diversity in the systems that exist today and elements of that diversity have been described in sections 2.3.2 to 2.3.6. This thesis, therefore, does not provide a definitive set of techniques for distributed problem solving but hints at the essential qualities of such techniques. In this regard, the chapter on architecture, and the

background material which preceeds it, form the backbone of the contributions made by this thesis. Figure 23 on page 103 is arguably the most important figure in this thesis as it captures the essence of what makes swarm systems work. The specification of the SynthECA architecture that closely follows the figure then provides an architectural expression of that figure; the sections on Chemistry and agent sensory apparatus being the most significant.

This thesis makes contributions in a number of areas. Firstly, the thesis provides a synthesis of activation-oriented systems and symbol rewriting systems through the use of chemicals and chemical reactions. An agent architecture is developed with these concepts that embodies swarm problem solving and embraces the Chemical Abstract Machine formalism well known to Computer Science. The architecture also supports the concept of subsumption, or layering, through the use of chemical signalling in order that complex systems of interacting swarms may be developed. Most noteworthy in the architecture is the complete absence of any global knowledge or centralized control. There is no "blind watchmaker" in this thesis.

While agent architectures are interesting from an academic viewpoint, their utility should be measured, in part, by their problem solving potential. This thesis presents a number of applications drawn from the communications domain that demonstrate the power of the SynthECA architecture.

The promise of mobile agents has yet to be realized for several reasons, not least of which is the significant security questions and need for multi-language execution environments that they raise. However, from a network engineering standpoint, questions

of agent lifecycle management and population control are vitally important and this thesis provides simple answers to them. In fact, the deceptive simplicity of the ideas and algorithms presented here makes one want to ask the question, "Can it really be this simple?" We make no apology for the simplicity of the solutions provided here; appealing instead to the argument regarding the importance of representation in problem solving. Obviously Nature provides us with a large number of examples of successful systems that exhibit complex, emergent behaviour where individual components appear simple. Complexity from simplicity is a recurring theme in much of the Artificial Life research that is active today.

However, this thesis, as all others, starts by posing a single question only to find that numerous related questions quickly appear. This thesis, we believe, is a starting point from which a rich vein of research into distributed problem solving using stigmergic communication can proceed.

7.2 Future Research

This section describes a number of future directions for the research described in this thesis.

7.2.1 Search techniques

In the chapter on routing, an algorithm was developed in order to speed up the convergence of the basic Ant Search algorithm. While successful in the routing domain, work is ongoing to apply the ASGA system to the TSP and asymmetric TSP as proposed in [Dorigo 96]. Also, [Gambardella 95] have proposed Ant-Q, a family of combinatorial

optimization algorithms that represent the synthesis of Q-Learning and Ant Search. Future work will extend ASGA to incorporate Ant-Q algorithms in order to determine the utility of adaptation in this new family of search algorithms. Particle Swarm optimization [Eberhart 95] -- a technique based upon similiar reinforcement mechanisms to the Ant Search method -- may also benefit from self adaptation of its controlling parameters. As has been noted by Dorigo himself, only a single form of the decision function used in ant migration has been explored and a systematic exploration of the interaction of Ant System parameters has yet to be performed. Considerable experimental work remains to be done here. While the TSP results using Ant Search have been very encouraging, a theoretical understanding remains elusive. We believe that [Millonas 94] work using ideas from Statistical Physics may well prove useful in constructing a sound theoretical framework.

7.2.2 Other Applications Areas

Given that the SynthECA architecture consists of agents moving on a graph, problems involving search within a graph may well benefit from its application. Specifically, there are other applications within the telecommunications domain besides routing and load balancing, for which Swarm Intelligence may prove useful.

In addition to the quadratic assignment problem and the job shop scheduling problem, which have obvious uses within and outside telecommunications, some work has been done on temporal graphs and correlation.

7.2.2.1 Alarm Correlation

The concept behind using swarm intelligence for alarm correlation is that alarms arrive

separated in time and space, and this can be thought of as forming a temporal graph. Finding correlations between alarms, i.e., diagnosing problems is equivalent to partitioning the graph into a number of cliques. Swarm Intelligence has been used for graph partitioning [Kunz 94].

The factors that effect the partitioning of alarms include:

- temporal closeness
- topological closeness — within and between network layers
- logical closeness — known direct causal relationships

These factors could be introduced into a swarm intelligence algorithm, and the sensitivity to each value can be varied using a power relationship with a sensitivity factor (raising to a power works well for routing and TSP).

As ants explore the graph, they will lay a pheromone trail representing the partitioning they have found. Other ants have a tendency to follow this trail, based on a sensitive factor, as for the other grouping factors.

Over time, the pheromone trails will be reinforced as more ants follow that path. This can be seen as a positive feedback search technique, which is controlled by evaporating the pheromone at a constant rate. This means that infrequently used relationships between alarms will evaporate toward zero, thus making their use more unlikely.

Solutions correspond to large accumulations of pheromone between nodes in a closed ring formation, the clique. The decision as to whether a large enough ring of pheromone has emerged has yet to be investigated.

Unlike routing problems where paths are limited to following arcs between nodes, the

alarm correlation problem allows ants to explore any appropriate alarm nodes on the graph. Clearly, this can be large, but it is anticipated that the complexity of the problem can be reduced by using a sliding window technique. This removes 'old' alarms over time, but can also be modified to 'forget' alarms that have been resolved, or to remove 'irrelevant' alarms. This sliding window technique can be seen as a global controller over the algorithm.

Use of this graph partitioning technique should lead directly to solutions where there are multiple faults, as this corresponds to multiple partitions within the graph.

7.2.2.2 World Wide Web

The section on "Application Oriented Routing" on page 190 describes an algorithm -- the AOR algorithm -- that improves quality of service where applications learn to share path elements in order to take advantage of their complementary statistical properties. The algorithm, based upon evolving chemical encodings, has been shown experimentally to improve the average quality of service experienced by applications in a number of networks.

The AOR algorithm, we believe, can be applied to the vexing problem of meaningful surfing on the World Wide Web. While finding an initial web page of interest is hard in itself, with search engines currently being exploited in order to attract a surfer to particular pages, the problem of which link to follow once on a particular page remains largely unaddressed. Ideally, we would like to take advantage of previous surfers' choices in link following in order to improve the quality of the surfing experience. This can be thought of as a routing problem where we want to follow links that were found of most interest to a

class of user with a complementary search question.

By encoding search questions using chemicals, possibly using hash coding for the keywords, and page summarization using a technique similiar to Extractor [Extractor], the AOR algorithm could be used to retrieve a set of web pages that follow a particular path in the web. Essentially, this would provide for a form of collaborative filtering across the World Wide Wide (WWW).

There are a number of privacy issues related to the above as such a technique would necessitate the leaving of a trail of information for each search performed by the user. Such information might subsequently be exploited for commercial or other reasons. However, it would doubtless improve the quality of the surfing experience which is currently being degraded at an alarming rate.

7.2.3 Open Chemistry Research

While the SynthECA architecture allows for closed and open chemical universes, this thesis has chosen to explore the application of a closed, binary chemistry. The simple binary chemistry used throughout this thesis was sufficient to solve routing, fault location and agent management problems[1] but this choice presupposes that no new or evolved behaviour is required in the system.

Many problem domains, e.g., automatic programming, require a problem representation that supports variable solution size and complex pattern matching. An example of a solution technique that deals with variable size solutions is Genetic Programming [Koza

1. In theory, an open chemistry is potentially required for the agent upgrade algorithm.

92]; a technique that has proven to be remarkably effective in the solution of many difficult automatic programming problems; e.g, the design of electronic circuits.

Given Genetic Programming as the inspiration for using variable size representations, an open chemical universe might well prove useful in generating autopoietic systems as described by Varela and recently clarified by McMullin. A line of research of this nature, where chemical reaction pathways are generated in support of stable information processing structures, have application in environments that are intended to be self sustaining, such as communication networks and the WWW.

7.2.4 Learning

This thesis has concentrated on the specification of an agent architecture that supports collaboration through communication using the environment. We have not considered the evolution of agent swarms themselves[1].

Consider for a moment that agent swarms are defined as being capable of executing a set of functions, {F}, and terminals, {T}, and that actions, A, are defined as the interpretration of a sequence of functions and terminals. These actions are associated with receptors. In essence, this describes the encoding strategy used in Genetic Programming. Given, then, that we have already defined an encoding for chemicals and chemical reactions, it would be possible to define a complete encoding for the agent. Given a complete agent encoding, it would then be possible to evolve agent swarms to perform a given task, given a problem domain and definition of fitness in solving a problem in that domain.

1. Arguably the AOR algorithm represents sensor/receptor evolution.

We believe that research in Artificial Life is already moving in this direction [Creatures99] and that more realistic agent-based simulation software will soon emerge from it. While swarm systems have been applied to games involving strategy; e.g., chess [Drogoul 95], learning of the form described above might well prove promising in improving the performance of such systems.

7.2.5 Theory

This thesis has concentrated on an architectural specification and proving its utility by application in an important problem domain, that of the management and control of communication networks. However, the contributions contained in this thesis need to be supported by sound theoretical analyses for different network architectures and agent chemical constitutions. It is clear that all theoretical literature relating self organization and non-linear irreversible processes is relevant here. Most noteable amongst this body of work would be Prigogine's results on self organization in nonequilibrium systems [Prigogine 77], self organizing criticality [Bak 96] and the ecology of computation [Huberman 88].

Considerable work on molecular evolution, and specifically the identification of reaction pathways which make such evolution inevitable, is of considerable importance. The identification of the Hypercycle [Eigen 79], and its underlying theoretical analysis, constitute a body of work that could be applied to an analysis of classes of SynthECA architectures.

7.3 Summary

This thesis has been inspired by the seemingly effortless way in which naturally occurring systems solve complex problems with simple components. It has never ceased to amaze us that chemical reactions and diffusion provide a rich environment for information processing. As a result, this thesis has brought together several ideas from areas spanning Computer Science, Biology, and Chemistry and applied them to a series of problems in the Engineering domain.

The thesis began with observations concerning the increasing importance of decentralization in systems thinking and the movement toward agent-oriented and mobile computing. With these observations, and the acknowledged success of simple social insects[1] in the solution of complex problems, we were led to the simple agent architecture we called the Synthetic Ecology of Agents, or SynthECA. In SynthECA, chemicals and chemical reactions are modelled within the agent by symbol re-writing and diffusion represented by a migration decision function. We have, we believe, demonstrated that information processing can occur in SynthECA systems in the same way that Reaction-Diffusion systems [Sherstinsky 94] are capable of complex information processing.

While the proof by existence that this thesis provides for the information processing characteristics of the chemically-inspired architecture presented is an interesting starting point, we believe, more importantly, that it sets out an agenda for future research into multi-agent (swarm) systems.

We look forward to contributing to this research effort.

---

1. Nature's mobile agents.

References

[Appleby 94] Appleby S. and Steward S., Mobile Software Agents for Control in Telecommunications Networks, *BT Technological Journal* 12 (2), pp. 104-113, April, 1994.

[Bak 96] Bak, P. How nature works; the science of self-organized criticality. Copernicus, New York, 1996.

[Banzaf 95] Banzaf, W., Self Organizing Algorithms derived from RNA Interactions, Evolution and Biocomputation: Computational Models of Evolution, Lecture Notes in Computer Science Vol. 899, Springer Verlag 1995.

[Beckers 92] Beckers R., Deneuborg J.L and Goss S. 1992. Trails and U-turns in the Selection of a Path of the Ant Lasius Niger. In *J. theor. Biol.* Vol. 159, pp. 397-415.

[Benatre 88] Banatre J-P., Coutant A, Le Metayer D. A parallel machine for multiset transformation and its programming style. In Future Generation Computer Systems 4, pp. 133-144. North-Holland, 1988.

[Beni 89] G. Beni and J. Wang, Swarm Intelligence in Cellular Robotic Systems, *Proceedings of the NATO Advanced Workshop on Robots and Biological Systems*, Il Ciocco, Tuscany, Italy, 1989.

[Beni 90] Beni G., Key issues of the theory of cellular automata as applied to swarm intelligence, Proceedings of the 1990 IEEE International Symposium on Intelligent Control, Philadelphia.

[Berry 92] Berry, G. and Boudol, G., The Chemical Abstract Machine, Theoretical Computer Science, 96(1), pp. 217-248, 1992.

[Bertsekas 87] D. Bertsekas and R. Gallager, *Data Networks*, ISBN 0-13-196825-4, Prentice Hall, pp.297-406, 1987.

[Bieszczad 98] Bieszczad, A. and Pagurek, B., Network Management Application-Oriented Taxonomy of Mobile Code, to be presented at the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, February 1998.

[Bieszczad 97] Bieszczad, A. and Pagurek, B., Towards plug-and play networks with mobile code, to be presented at the International Conference for Computer Communications ICCC'97, November 19-21, 1997, Cannes, France.

[Bluetooth 99] URL: http://www.bluetooth.com

[Bonabeau 94] Bonabeau E., and Theraulaz G., Intelligence Collective, Hermes, Paris, 1994.

[Bonabeau 98] Bonabeau E., Henaux F., Guérin S., Snyers D., Kuntz P. and Théraulaz G., Routing in Telecommunication Networks with Smart Ant-Like Agents. *Proceedings of Second International Workshop on Intelligent Agents for Telecommunication Applications (IATA '98)*, 1998.

[Boyan 94] Boyan J.A. and Littman M. L., Packet routing in dynamically changing networks: A reinforcement learning approach. In Cowan, J. D., Tesauro, G., and Alspector, J. (eds.), *Advances in Neural Information Processing Systems 6 (NIPS)*. Morgan Kaufmann, 1994.

[Boyer 99] Boyer, J., Pagurek, B., White, T., Methodologies for PVC Configuration in Heterogeneous ATM Environments Using Intelligent Mobile Agents. In Proceedings of the 1st Workshop on Mobile Agents and Telecommunications Applications (MATA '99), October, 99.

[Brenner 98] Brenner, W., Zarnekow, R., and Wittig, H., Intelligent Software Agents: Foundations and Applications, Springer Verlag, Heidelberg, 1998.

[Brooks 86] Brooks, R.A., Achieving Artificial Intelligence Through Building Robots, A.I. Memo 899, MIT A.I. Lab, 1986.

[Brooks 91] Brooks, R.A., Intelligence Without Representation, Artificial Intelligence, Vol. 47, pp. 139-159, 1991.

[Brusselator] http://www.cmp.caltech.edu/~mcc/STChaos/,

http://www.cherwell.com/cherwell/products/simulation/modelmaker/bruss.htm

[Bullnheimer 97] Bullnheimer B., R.F. Hartl and C. Strauss, Applying the Ant System to the Vehicle Routing Problem. 2nd Metaheuristics International Conference (MIC-97), Sophia-Antipolis, France, 1997.

[Cabri 00] Cabri G., Leonardi L., Zambonelli F.,
Mobile-Agent Coordination Models for Internet Applications
IEEE Computer Magazine, Vol. 33, No. 2, February 2000.

[Cabri 99] Cabri G., Leonardi L., Reggiani G., Zambonelli F.,
Design and Implementation of a Programmable Coordination Architecture for Mobile Agents, Proceedings of the TOOLS EUROPE '99 Conference, Nancy (F), June 1999.

[Case 90] Case, J. D., Fedor, M., Schoffstall, M. L. and Davin, C., Simple Network Management Protocol, RFC 1157, May 1990.

[Case 93] Case, J.D., and Levi, D. B. , SNMP Mid-Level-Manager MIB, Draft, IETF, 1993.

[Chess 97] Chess D., Harrison C., and Kershenbaum A., Mobile agents: Are they a good idea? In Mobile Object Systems: Towards the Programmable Internet, pages 46-48. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222

[Clark 97] Clark, I. J., and White, A. R. P., Real Time Control Architecture for Admission Control in a Communication Network, US Patent Application 08/873497, 1997.

[Clearwater 96] Clearwater S. H. (ed.), Market-Based Control: A Paradigm for Distributed Resource Allocation. World Scientific, 1996.

[Color 92] Colorni A., Dorigo M. and Maniezzo V. An Investigation of Some Properties of an Ant Algorithm. Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92), Brussels, Belgium, R.Männer and B.Manderick (Eds.), Elsevier Publishing, 509-520, 1992.

[Color 94] Colorni A., Dorigo M., Maniezzo V., and Trubian M., Ant System for Job-shop Scheduling, Belgian Journal of Operations Research, Statistics and Computer Science, 1994.

[Coombs 87] Coombs, S., and Davis, L., Genetic Algorithms and Communication Link Speed Design: Theoretical Consideration, In Proceedings of the Second International Conference on Genetic Algorithms, pp. 252-256, 1987.

[Costa 97] Costa D. and Hertz A., Ants Can Colour Graphs. Journal of the Operational Research Society, 48, 295-305, 1997.

[Creatures99] http://www.cyberlife.co.uk/

[Davis 87] Davis, L. (editor), Genetic Algorithms and Simulated Annealing, Morgan Kaufmann Publishers Inc, Los Altos, CA, 1987.

[Davis 87a] Davis, L., and Coombs, S., Genetic Algorithms and Communication Link Speed Design: Constraints and Operators, In Proceedings of the Second International Conference on Genetic Algorithms, pp. 257-260, 1987.

[Deneubourg 90] Deneubourg, J.L., et al, The Dynamics of collective Sorting: Robot-Like Ants and Ant-Like Robots, in From Animals to Animats: Proc. First Int. Conference on Simulation of Adaptive Behavior, J-A. Meyer, and S.W. Wilson, eds., Paris, France, pp. 356-363, 1990.

[Di Caro 97] Di Caro G. and Dorigo M., AntNet: A Mobile Agents Approach to Adaptive Routing. Tech. Rep. IRIDIA/97-12, Université Libre de Bruxelles, Belgium, 1997.

[Di Caro 98] Di Caro G. & Dorigo M., AntNet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317-365, 1998.

[Dijkstra 59] Dijkstra E.W. A Note on Two Problems in Connexion with Graphs In *Numerische Mathmatik* vol. 1, 1959.

[Doran 97] Doran J. E., Franklin S., Jennings N. R. and Norman T. J. "On Cooperation in Multi-Agent Systems" The Knowledge Engineering Review, 12(3), 309-314. October, 1997.

[Dorigo 91] Dorigo M., Maniezzo V. and Colorni A., The Ant System: An Autocatalytic Optimizing Process. Technical Report No. 91-016, Politecnico di Milano, Italy, 1991.

[Dorigo 92] Colorni A., Dorigo M. and Maniezzo V. Distributed Optimization by Ant Colonies. Proceedings of the First European Conference on Artificial Life, Paris, France, F.Varela and P.Bourgine (Eds.), Elsevier Publishing, 134-142, 1992

[Dorigo 96] Dorigo M., Maniezzo V. and Colorni A. The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, 26, 1, 29-41, 1996

[Drogoul 95] Drogoul A. In *From reaction to cognition*, lecture notes in AI 957, C. Castelfranchi & J.P. Müller (Eds), pp. 13-27, Springer-Verlag, Berlin-Heidelberg, 1995.

[Eberhart 95] Eberhart R. and Kennedy J., A New Optimizer Using Particles Swarm Theory, Proc. Sixth International Symposium on Micro Machine and Human Science (Nagoya, Japan), IEEE Service Center, Piscataway, NJ, 39-43, 1995

[Eigen 79] Eigen, M. and Schuster, P., The Hypercycle: A Principle of Natural Self Organization, Springer Verlag, New York, 1979.

[Extractor] http://extractor.iit.nrc.ca/

[Faieta 94] Faieta B., and Lumer E., Diversity and Adaptation in Populations of Clustering Ants, proceedings of Conference on Simulation of Adaptive Behaviour, Brighton, 1994.

[Ferguson 92] Ferguson, I.A., The Touring Machine, PhD thesis, Cambridge University, England, 1992.

[Ferguson 95] Ferguson, I.A.,. On the Role of BDI Modelling for Integrated Control and Coordinated Behavior in Autonomous Agents. Journal of Applied Artificial Intelligence, 9(4), 1995.

[Fitzgerald 88] Fitzgerald T.D., Peterson S.C. Cooperative foraging and communication in caterpillars, *Bioscience*, 38, pp. 20-25, 1988.

[Fontana 91] Fontana W., Algorithmic Chemistry, Artificial Life II, pp.159-209, 1991.

[Franks 89] N.R. Franks, Army Ants: A Collective Intelligence, *Scientific American*, Vol. 77, 1989.

[Gambardella 95] Gambardella L.M. & M. Dorigo. Ant-Q: A Reinforcement Learning Approach to the Traveling Salesman Problem. Proceedings of ML-95, Twelfth International Conference on Machine Learning, Tahoe City, CA, A. Prieditis and S. Russell (Eds.), Morgan Kaufmann, 252-260, 1995.

[Gamma 95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., and Booch, G., Design Patterns : Elements of Reusable Object-Oriented Software, Reading MA: Addison-Wesley, October, 1995.

[Gang 00] Gang, A., Software Hot Swapping Techniques for Upgrading Mission Critical Applications on the Fly, Master of Engineering Thesis, Department of Systems and Computer Engineering, Carleton University, April, 2000.

[Gelernter 86] D. Gelernter, Domesticating Parallelism, *IEEE Computer*, 19(8):12-16, August 1986.

[Gödel 31] In S Feferman, *Gödel's Collected Works* (1986), 1-36.

Also: http://www.math.hawaii.edu/~dale/godel/godel.html

[Goldberg 89] Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[Goss 90] Goss S., Beckers R., Deneubourg J.L., Aron S., Pasteels J.M. 1990. How Trail Laying and Trail Following Can Solve Foraging Problems for Ant Colonies, in Hughes R.N. (ed.) *NATO ASI Series, Vol. G 20, Behavioural Mechanisms of Food Selection*, Springer Verlag, Berlin.

[Grassé 59] Grassé P.P., La reconstruction du nid et les coordinations interindividuelles chez Bellicoitermes natalenis et Cubitermes sp. La theorie de la stigmergie: Essai d'interpretation des termites constructeurs. In Insect Societies, Vol. 6, pp. 41-83, 1959.

[Grasshopper] Mobile Agent Platform- The OMG-MASIF conformant mobile agent platform in Grasshopper, URL: http://www.ikv.de/products/grasshopper/

[Hayzelden 99] Hayzelden and Bigham (Eds.), Software Agents for Future Communications Systems, Springer Verlag, 3-540-65578-6, 1999.

[Heusse 98] M. Heusse, D. Snyers, S. Guérin, and P. Kuntz, Adaptive Agent-Driven Routing and Load Balancing in Communication Networks, Ants'98, Brussels, Belgium, October, 1998.

[Holland 96] Holland, J., H., and Mimnaugh, H., Hidden Order: How Adaptation Builds Complexity, Helix Books, 1996.

[Holland 86] Holland, J. H., Escaping Brittleness: the Possibilities of General-Purpose Learning Algorithms applied to Parallel Rule-Based Systems. In Machine Learning, an Artificial Intelligence Approach, Volume II, edited by R.S. Michalski, J.G. Carbonell and T.M. Mitchell, Morgan Kaufmann, 1986.

[Holland 75] Holland, J. H., Adaptation in Natural and Artificial Systems, University of Michigan Press, 1975.

[Hölldobler 94] Hölldobler B. and Wilson E.O., Journey to the Ants. Bellknap Press/Harvard University Press, 1994.

[Huberman 88] Huberman, B.A., The Ecology of computation. North-Holland, Amsterdam, 1988.

[Ishida 96] Ishida, Y., Active Diagnosis by Immunity-Based Agent Approach, Proceedings of the Seventh International Workshop on Principles of Diagnosis (DX 96), Val-Morin, Canada, pp. 106-114, 1996.

[Jini] http://www.sun.com/jini/

[Kelly 95] Kelly, K., Out of Control : The New Biology of Machines, Social Systems and the Economic World, Perseus Press, 1995.

[Kephart 98] Kephart J. O., Hanson J. E., Levine D. W. , Grosof B. N., Sairamesh J., Segal R. B., and White S. R., Dynamics of an Information-Filtering Economy, Proceedings of *Second International Workshop on Cooperative Information Agents '98*, Paris, July 4-7, 1998.

[Korf 90] Korf, R., Planning as search: A quantitative approach. Artificial Intelligence, Vol. 33, No. 1, 1987, pp. 65-88. Reprinted in Readings in Planning J. Allen, J. Hendler, and A. Tate (Eds.), Morgan Kaufmann, pp. 566-577, 1990

[Koza 92] Koza, J., R., Genetic Programming : On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems), Bradford, 1992.

[Kugler 87] Kugler, P. and Turvey, M. T., Information,. natural law, and the self-assembly of rhythmic movement, Hillsdale, N.J., Erlbaum. 1987

[Kunz 94] P. Kuntz and D. Snyers, Emergent Colonization and Graph Partitioning. *Proceedings of the Third International Conference on Simulation of Adaptive Behavior: From Animals to Animats 3*, MIT Press, Cambridge, MA, 1994.

[Labrou 97] Labrou R., KQML as an Agent Communication Language, in "Software Agents", Jeffrey Bradshaw (editor), AAAI/MIT Press, 1997. Authors: Tim Finin, Yannis Labrou and James Mayfield.

[Langton 87] Langton, C.G., Artificial Life, Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Things, Los Alamos, New Mexico, Addison Wiley, 1987.

[Leer 95] Leerink L. R., Schultz S.R. and Jabri M.A., A Reinforcement Learning Exploration Strategy based on Ant Foraging Mechanisms. *Proceedings of the Sixth Australian Conference on Neural Networks*, Sydney, Australia, 1995.

[LV 25] http://www.ento.vt.edu/~sharov/PopEcol/lec10/lotka.html

[Maes 89] Maes P., A Spreading Activation Network for Action Selection, Intelligent Autonomous Systems-2 Conference, Amsterdam, December 1989.

[Maniezzo 94] Maniezzo V., A. Colorni and M. Dorigo, The Ant System Applied to the Quadratic Assignment Problem. Tech. Rep. IRIDIA/94-28, Université Libre de Bruxelles, Belgium, 1994.

[Mann 95] Mann J., White T. and Turner J., Optimal Route Finding in ATM Networks Using Genetic Algorithms, in *Proceedings 7th BNR Design Forum*, December, 1995.

[Micmac] http://micmac.mitel.com/micmac.htm

[Millonas 94] Millonas M. M., Swarms, Phase Transitions and Collective Intelligence, In Artificial Life III (ed. C. G. Langton). Santa Fe Institute Studies in the Sciences of Complexity, Proc. Vol XVII. Reading, Massachusetts: Addison-Wesley, 1994.

[Minsky 88] Minsky, M., L., The Society of Mind, Simon and Schuster, 1988.

[Mole] http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/security.html

[Muller 96] Muller, J.P., The design of intelligent agents: a layered approach, Lecture Notes in Computer Science, Vol. 1177, Springer Verlag, Heidelberg e.a. 1996.

[Ndovie 94] Ndovie, B., Multi-agent cooperation in air traffic control: A functional analysis. DAKE Centre Technical Report DAKE/-/TR-940013.0, University of Keele, Dake Centre, University of Keele, Keele, Staffs ST5 5BG, UK, February 1994.

[Newell 80] Newell, A., Physical Symbols Systems, *Cognitive Science 4*, 135-183, 1980.

[Ning 99] Ning, F., Software Hot Swapping, Master of Engineering thesis, Department of Systems and Computer Engineering, Carleton University, September, 1999.

[o.V. 97a] o.V., Methodological Assumptions of Subsumption, URL: http:// krusty/eecs/umich.edu/cogarch/brooks/method.html

[o.V. 97b] o.V., Notes: The subsumption architecture, URL: http:// www.janus.demon.co.uk/alife/notes/subsump.html.

[O'Hare 96] O'Hare G. M. P. and Jennings N. R. (eds.), Foundations of Distributed Artificial Intelligence, ISBN 0-471-00675-0, John Wiley & Sons, 1996

[Pagurek 98] Pagurek B., Li Y., Bieszczad A., and Susilo G., Configuration Management In Heterogeneous ATM Environments using Mobile Agents, Proceedings of the Second International Workshop on Intelligent Agents in Telecommunications Applications (IATA '98).

[Pagurek 00] Pagurek, B., Wang, Y., White, T., Integration of Mobile Agents with SNMP: How and Why. Proceedings of the IEEE/IFIP Network Operations and Management Symposium, Hawaii, May 2000.

[Parunak 98] H. Van Dyke Parunak, Go to the Ant: Engineering Principles from Naturally Multi-Agent Systems, to appear in *Annals of Operations Research.* Available as Center for Electronic Commerce report CEC-03, 1998.

[Picco 98] Picco, G. and Baldi, M., Evaluating the Tradeoffs of Mobile Code Design Paradigms in Network Management Applications. In *Proceedings of the 20th International Conference on Software Engineering (ICSE'98)*, Kyoto (Japan), R. Kemmerer and K. Futatsugi, eds., April 1998, IEEE CS Press, ISBN 0-8186-8368-6, pp. 146-155.

[van Gelder 95] van Gelder, T. and R. Port, It's about time: An overview of the dynamical approach to cognition, *Mind as motion: Explorations in the dynamics of cognition*, in R. Port, and T. van Gelder eds., Cambridge, MA: MIT Press, 1995

[Prigogine 77] Nicolis, G. and Prigogine, I. Self-Organization in Nonequilibrium Systems: From Dissipative Structures to Order Through Fluctuations, Wiley, New York, 1977.

[Rao 95] Rao A. S. and Georgeff M., *BDI Agents: from theory to practice.* In Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), pp. 312--319, San Francisco, CA, June 1995.

[Resnick 94] Resnick, M., Turtles, Termites and Traffic Jams, Explorations in Massively Parallel Microworlds, MIT Press, 1994.

[Reynolds 87] Reynolds, C. W. Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings) pp. 25-34, 1987.

[Robinson 65] Robinson, J. A. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23-41, January 1965.

[Sander 97] Sander, T., Tschudin, C., Towards Mobile Cryptography. Technical Report 97-049, Institute of Computer Science, Berkley, 1997.

[Schoonderwoerd 97] Schoonderwoerd R., Holland O. and Bruten J., Ant-like Agents for Load Balancing in Telecommunications Networks. Proceedings of Agents '97, Marina del Rey, CA, ACM Press pp. 209-216, 1997.

[Schramm 98] Schramm, C., Bieszczad, A. and Pagurek, B., Application-Oriented Network Modeling with Mobile Agents, Proceedings of the IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Louisiana, February 1998.

[Seth 98] Seth, A.K., The Evolution of Complexity and the Value of Variability In Proceedings of the 6th International Conference on Artificial Life (ALIFE VI), eds. C. Adami, R. Belew, H. Kitano, and C. Taylor, MIT Press, pp 209-221, 1998.

[Shapiro 88] Shapiro, J. A., Bacteria as multi cellular organisms, *Scientific American*, pp. 82-89, 1988.

[Sherstinsky 94] A. S. Sherstinsky, M-Lattice: A System for Signal Synthesis and Processing Based on Reaction-Diffusion, Sc.D. thesis, Massachusetts Institute of Technology, Cambridge, MA (1994).

[Shoham 93] Shoham, Y., Agent-oriented programming. Artificial Intelligence, 60(1):51-92, 1993.

[Steels 95] Steels, L., and Brooks, R.. Building Situated Embodied Agents. The Alife route to AI. Lawrence Erlbaum Assoc., New Haven. 1995.

[Steiglitz 96] Steiglitz K., Honig M. L., Cohen L. M. A Computational Market Model Based on Individual Action. In S. Clearwater (ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[Stützle 97] Stützle T. and Hoos H. 1997. The MAX-MIN Ant System and local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Global Optimization. *2nd Metaheuristics International Conference (MIC-97)*, Sophia-Antipolis, France - July 21-24, 1997.

[Susilo 97] Susilo, G., *Infrastructure for Advanced Network Management based on Mobile Code,* Technical Report SCE-97-10, Systems and Computer Engineering, Carleton University, June 1997.

[Susilo 98] Susilo, G., Bieszczad, A. and Pagurek, B., Infrastructure for Advanced Network Management based on Mobile Code, Proceedings IEEE/IFIP Network Operations and Management Symposium NOMS'98, New Orleans, Luisiana, February 1998.

[Sutton 88] Sutton R. S., Learning to predict by the method of temporal differences. *Machine Learning*, 3(1):9-44, 1988.

[Taillard 97] Taillard E. and L. M. Gambardella, An Ant Approach for Structured Quadratic Assignment Problems. 2nd Metaheuristics International Conference (MIC-97), Sophia-Antipolis, France, 1997.

[Takashina 96] Takashina T. and Watanabe S. The locality of information gathering in multiagent system, In Proceedings of the 2nd International Conference on Multi-agent Systems (ICMAS '96), December, 1996.

[Tanebaum 96] Tanenbaum A.S., *Computer Networks*, 3rd Edition, ISBN 0-13-349945-6, Prentice Hall, pp. 345-373, 1996.

[Varela 80] Varela, F. G., and Maturana, H. R., Autopoiesis and cognition: the realization of the living. Dordrecht, Reidel. 1980 141p.

[Watkins 89] Watkins J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, 1989.

[Watkins 92] Watkins J. C. H. and Dayan P. Q-learning. *Machine Learning*, 8(3):279-292, 1992.

[Wellman 96] Wellman M. P., Market-oriented programming: Some early lessons. In S. Clearwater (ed.), *Market-Based Control: A Paradigm for Distributed Resource Allocation.* World Scientific, 1996.

[White 92] White, T., and Bieszczad, A., The Expert Advisor: An Expert System for Real Time Network Monitoring, European Conference on Artificial Intelligence, Proceedings of the Workshop on Advances in Real Time Expert Systems Technology, August, 1992.

[White 96] White T., and Ross, N., Fault Diagnosis and Network Entities in a Next Generation Network Management System, in Proceedings of EXPERSYS-96, Paris, France, pp. 517-522.

[White 97] White T. and Ross N., An Architecture for an Alarm Correlation Engine, Object Technology 97, Oxford, 13-16 April, 1997.

[White 93] White, A., R., P., Integrating Automata with Genetic Algorithms in order to provide Adaptive Operators, M.C.S. thesis, Carleton University, 1993.

[White 98a] White T., Pagurek B. and Oppacher F., Connection Management using Adaptive Mobile Agents, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), July, 1998.

[White 98b] White T., Bieszczad A., Pagurek B., Distributed Fault Location in Networks Using Mobile Agents. Proceedings Intelligent Agents for Telecommunications Applications (IATA '98), July, 1998.

[White 98c] White T., Pagurek B and Oppacher F., ASGA: Improving the Ant System by Integration with Genetic Algorithms. Proceedings of the Symposium on Genetic Algorithms (SGA '98), July, 1998.

[White 98d] T. White and B. Pagurek, Towards Multi-Swarm Problem Solving in Networks. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS '98)*, pp. 333-340, July, 1998.

[White 99a] White, T., Mann, J., and Smith G. D., Genetic Algorithms and Network Ring Design, Annals of Operational Research 86 (1999), pp. 347-371.

[White 99b] White T. and Pagurek B., Emergent Behaviour and Mobile Agents. In Proceedings of the Workshop on Mobile Agents in Coordination and Cooperation at Autonomous Agents '99, Seattle, May 1st-5th, 1999.

[White 99c] White T., and Pagurek B., Learning agents for diagnosis, Proceedings of the Pacific Rim International Multi-Agent Workshop, December, 1999.

[White 99d] White T., Pagurek B., and Bieszczad A., Network Modeling For Management Applications Using Intelligent Mobile Agents, Journal of Network and Systems Management, September, 1999.

[Winter 97] Shackleton M. A. and Winter C.S. Information Chemistry. In Proceedings of the Fourth European Conference on Artificial Life (ECAL 97), Brighton, UK, July, 1997.

[Wolpert 99] Wolpert D. H., Wheeler K. R. and Tumer K., General Principles of Learning-based Multi-Agent Systems. In Proceedings of the 3rd Annual Conference on Autonomous Agents (AA '99), Seattle, pp. 77-83, May, 1999.

[Yemini 93] Yemini, Y., The OSI Network Management Model, IEEE Communication Magazine, pages 20-29, May 1993.

[Yemini 91] Yemini, Y., Goldszmidt, G. and Yemini, S., Network Management by Delegation. In The Second International Symposium on Integrated Network Management, Washington, DC, April 1991.

APPENDIX A *Swarm Simulation Environment*

A.1 Overview

This appendix describes the swarm simulation environment that has been built in order to conduct experiments in multi-swarm problem solving in networks. The simulation has been written in Smalltalk, and provides a rich graphical environment for viewing the problem solving process. The environment has been designed with extensibility in mind; the addition of new problem solving swarms being of primary interest. Not all user interface components are described in this appendix, only those that relate directly to multi-swarm problem solving are included.

A.2 Introduction

The main window of the simulation environment is shown in Figure 68 below. The main

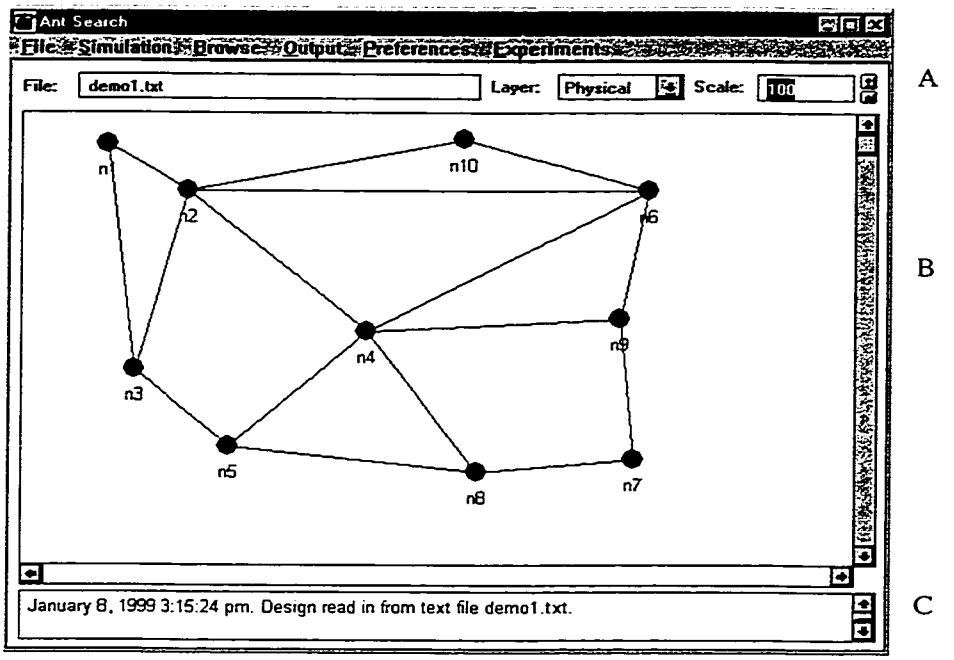


FIGURE 68. **Simulation Main Window**

window is the starting point for any simulation; several other dialogs may be accessed via the pull down menu displayed at the top of the window. These dialogs and their associated functions will be described in subsequent sections.

The anchor window consists of three main regions. The region designated A in Figure 68 consists of the name of the file from which the network was loaded, the layer of the network being displayed and the scale at which the network is being displayed. It is possible to simulate multiple layers of a network; for example, the physical layer and logical network layers. However in this thesis, only the physical layer was modelled. A graphical view of the network is displayed in Region B. User interaction with the nodes

and links displayed is possible. Nodes and links can be dragged within the network display region, and popup menus are defined for each of the displayed components. A background popup menu is also defined. These menus will not be not described here in the interests of brevity. Finally, Region C displays a scrolling list of status messages associated with the current session of the simulator. The beginning and end of an activity are marked with messages being added to the status region, along with important warning or error messages. Error indications are also associated with the display of a modal dialog that forces user acknowledgement before the simulation session is allowed to continue.

The following sections in this appendix briefly document the main visual components of the simulation environment.

A.2.1 Principal Simulation Dialogs

A.2.1.1 The Experiment Manager Dialog

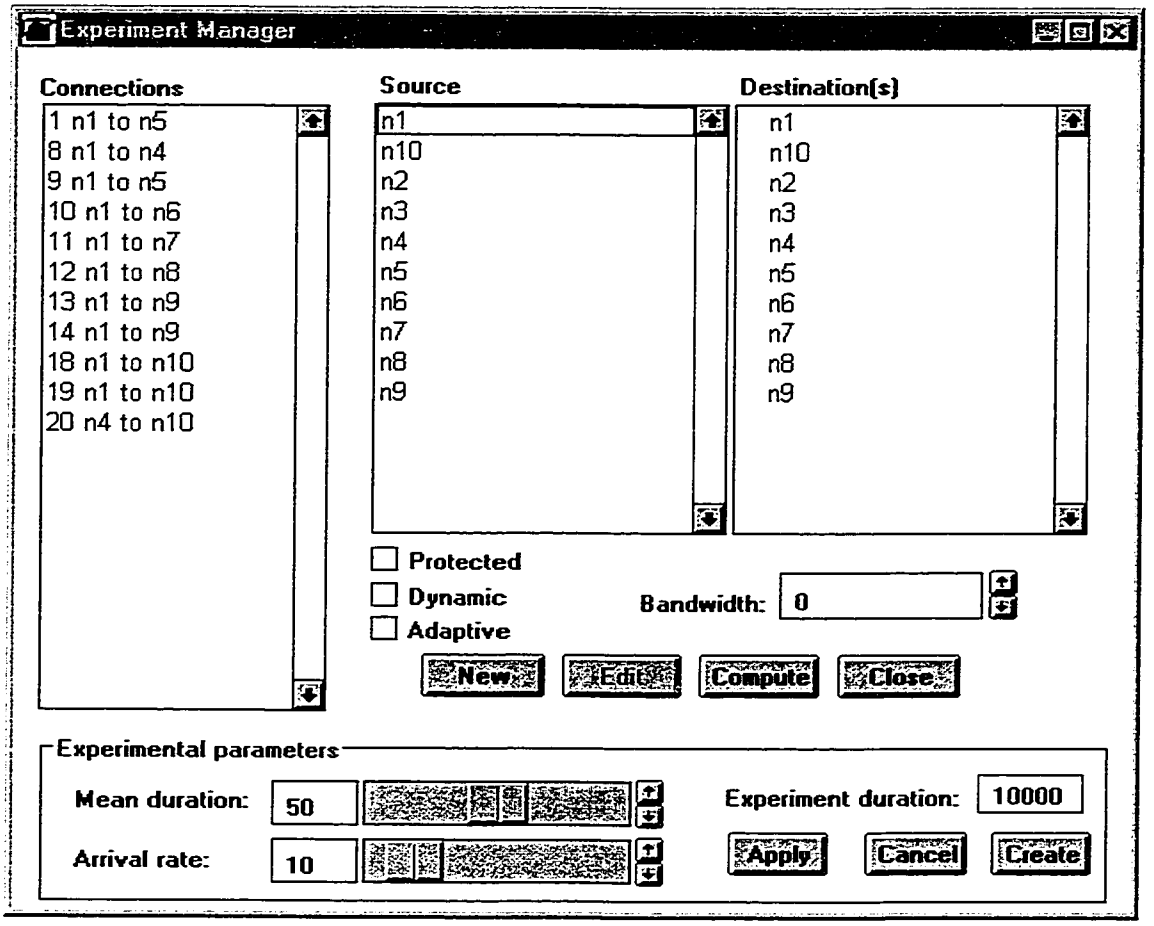


FIGURE 69. **Experiment Manager Dialog**

The Experiment Manager Dialog allows a user to set up a number of connections each of which have a mean arrival rate and duration. A Poisson distribution is assumed for duration and arrivals. This dialog facilitates the creation, management and evaluation of a large number of simultaneously active connection requests and is intended to represent a "realistic" network situation.

A.2.1.2 Connections Notebook

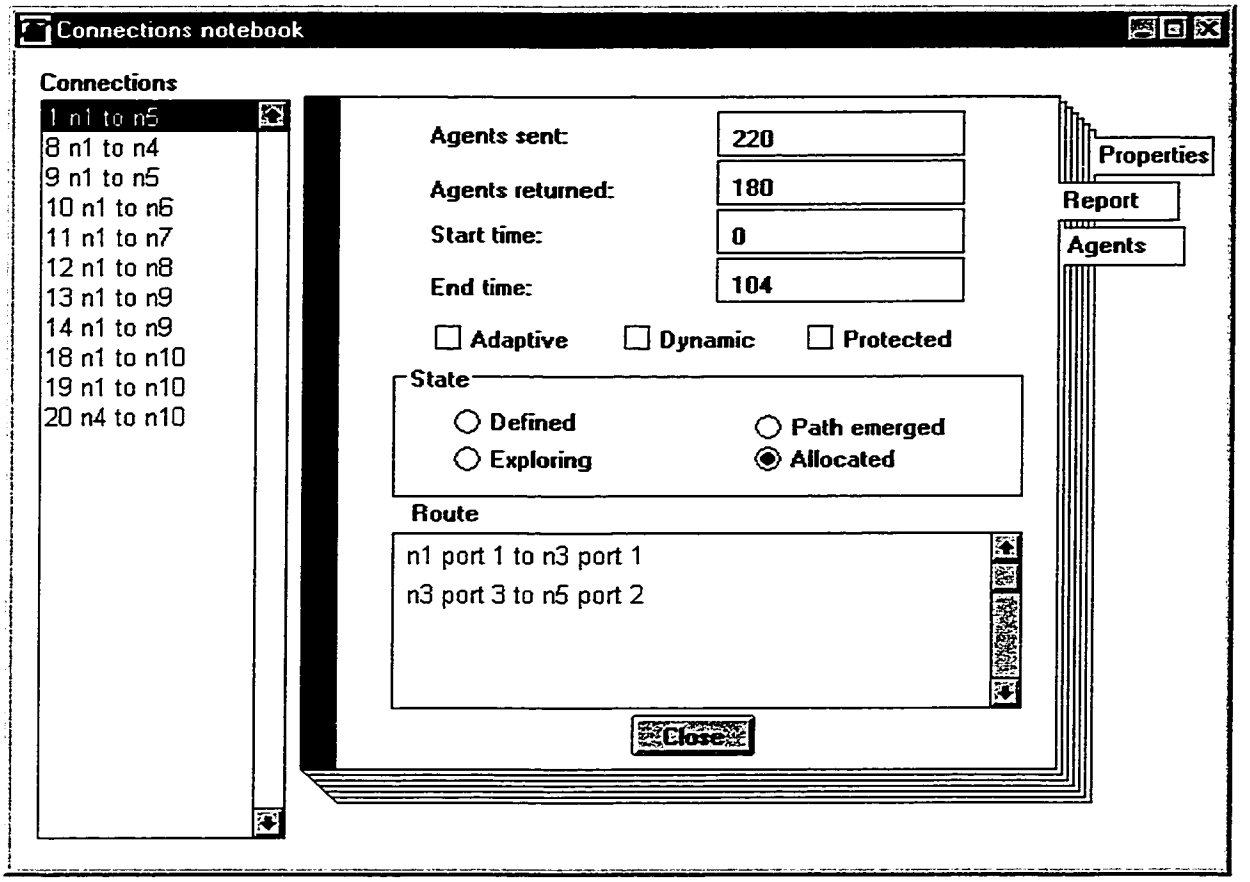


FIGURE 70. **Connections Notebook**

The Connections Notebook allows the solution process to be monitored for a selected connection. The number of agents sent, returned and the state of path search are displayed. The full life cycle, from connection definition to allocation can be viewed and the path which emerges is shown in text format in the Route section of the dialog. The Agents tab allows the location of all agents associated with the connection to be displayed. The Properties tab allows the properties associated with the connection to be displayed.

A.2.1.3 Links Notebook

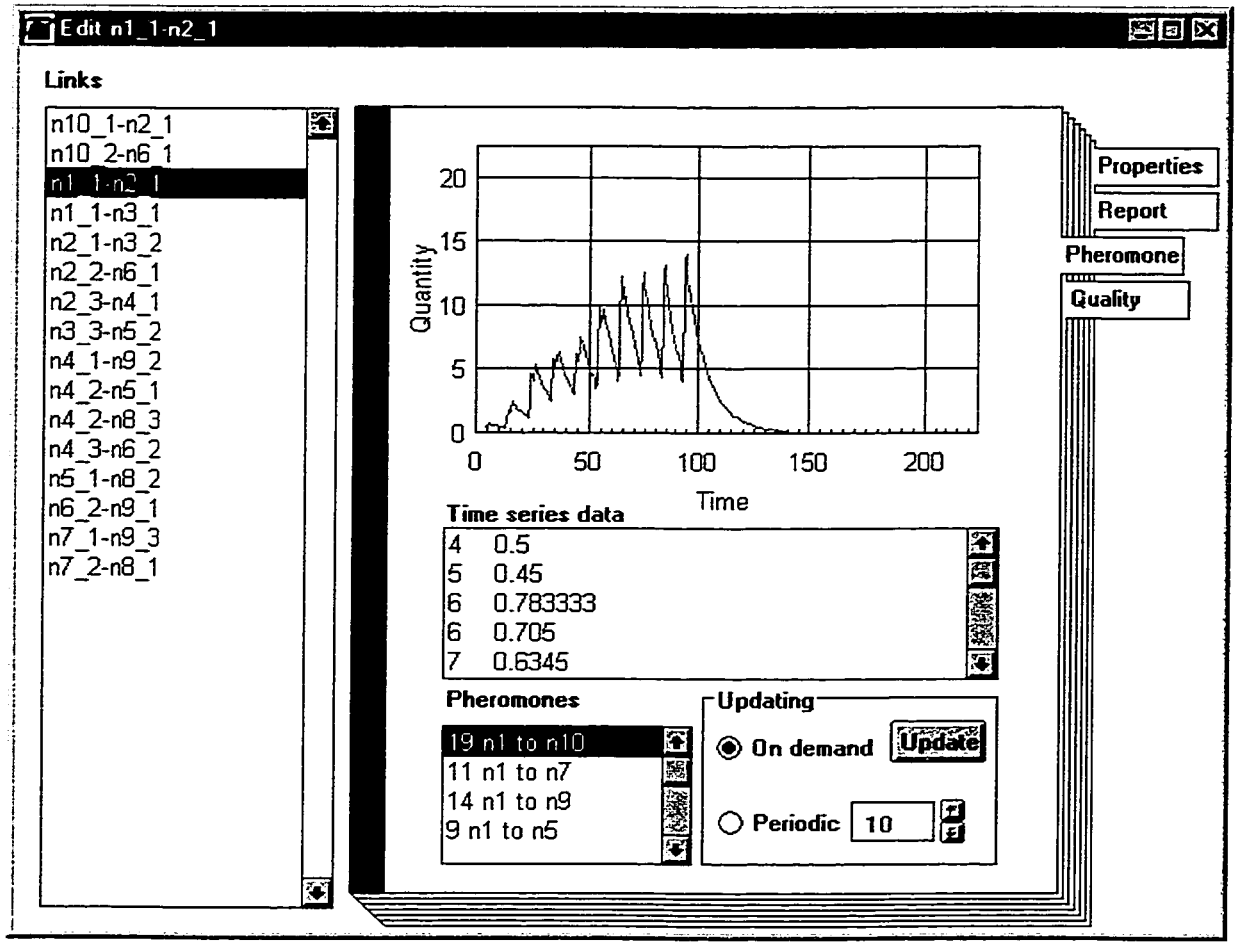


FIGURE 71. **Links Notebook**

The Links Notebook allows the solution process to be viewed from a link's perspective. The view shown above is the quantity of pheromone that is present on the selected link (n1_1, n2_1) for the selected connection (n1 to n10). The time series data shown in the above figure may also be exported to Microsoft Excel for further analysis. The Report tab displays the connections that are using the link. The properties tab displays the properties of the link. The quality tab is to indicate the quality of service for the link. This aspect of the link is used to inject "faults" into the network by changing q-chemical concentrations on various network entities.

A.2.1.4 Nodes Notebook

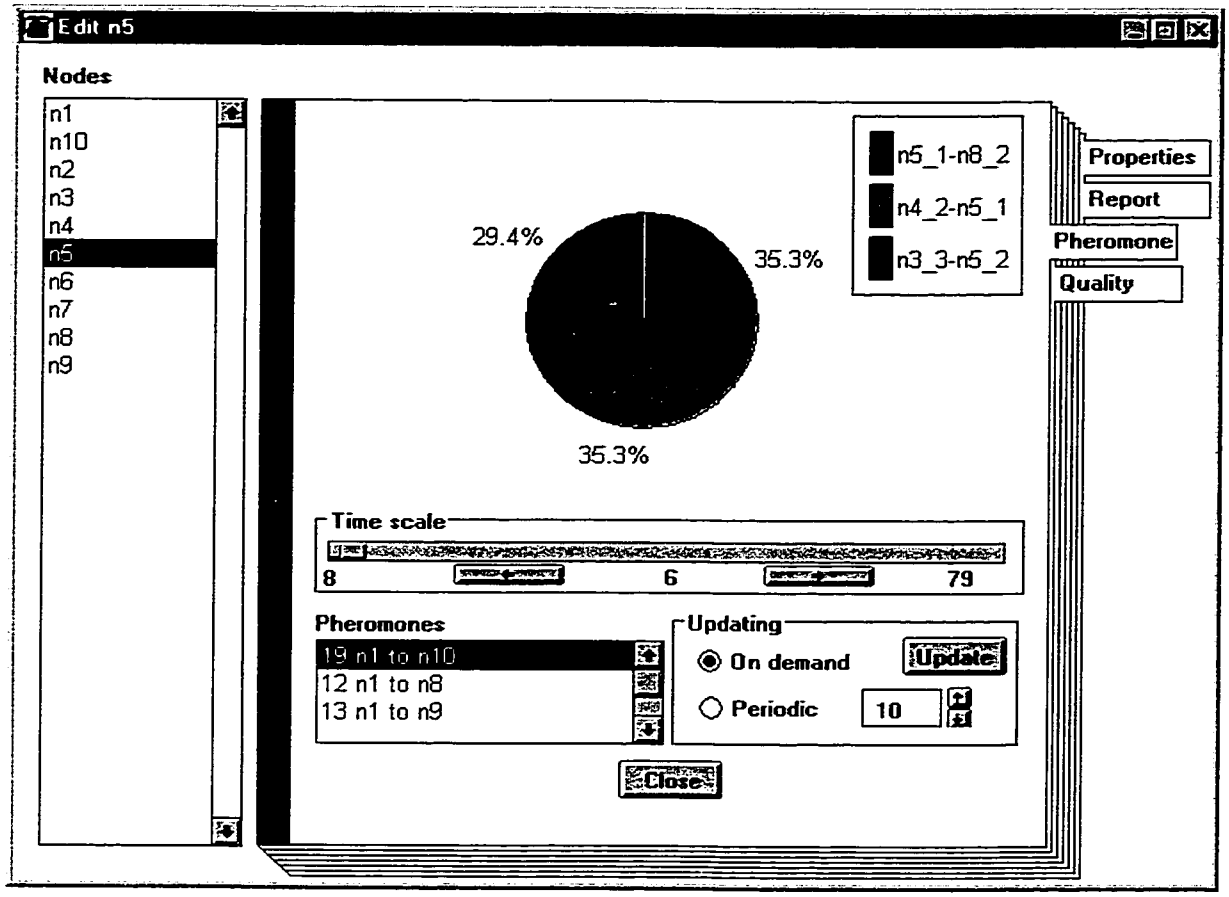


FIGURE 72. **Nodes Notebook**

The Nodes Notebook allows the solution process to be viewed from a node's perspective. The view shown above is the quantity of pheromone that is present on the selected node for the selected connection (n1 to n10). The pie chart provides a graphical view of the probability with which an agent will select a given link. The time series data for the node may be viewed by moving the slider, the pie chart being updated to reflect the distribution at that moment. The Report tab displays the connections that are using the node. The properties tab displays the properties of the node. The quality tab is to indicate the quality of service for the node. This aspect of the node is used to inject "faults" into the network by changing q-chemical concentrations on various network entities.

A.2.1.5 Simulation Control

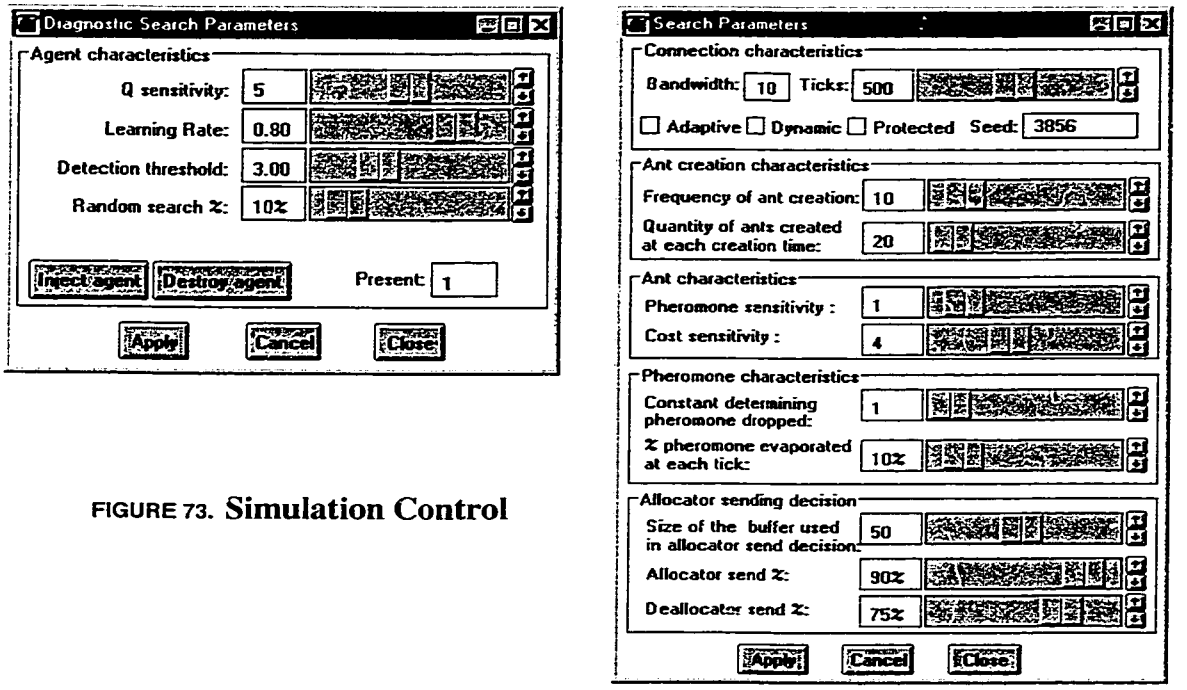


FIGURE 73. **Simulation Control**

There are two simulation control dialogues that allow experimentation with system parameters. These are shown above in Figure 73. These controls allow various routing and diagnostic parameters to be adjusted in order to assess their effects upon solution quality and speed of emergence. Parameters of particular interest are the density of routing and diagnostic agents, the rate of production of agents and the sensitivity to various pheromones.

A.2.1.6 Simulation Output

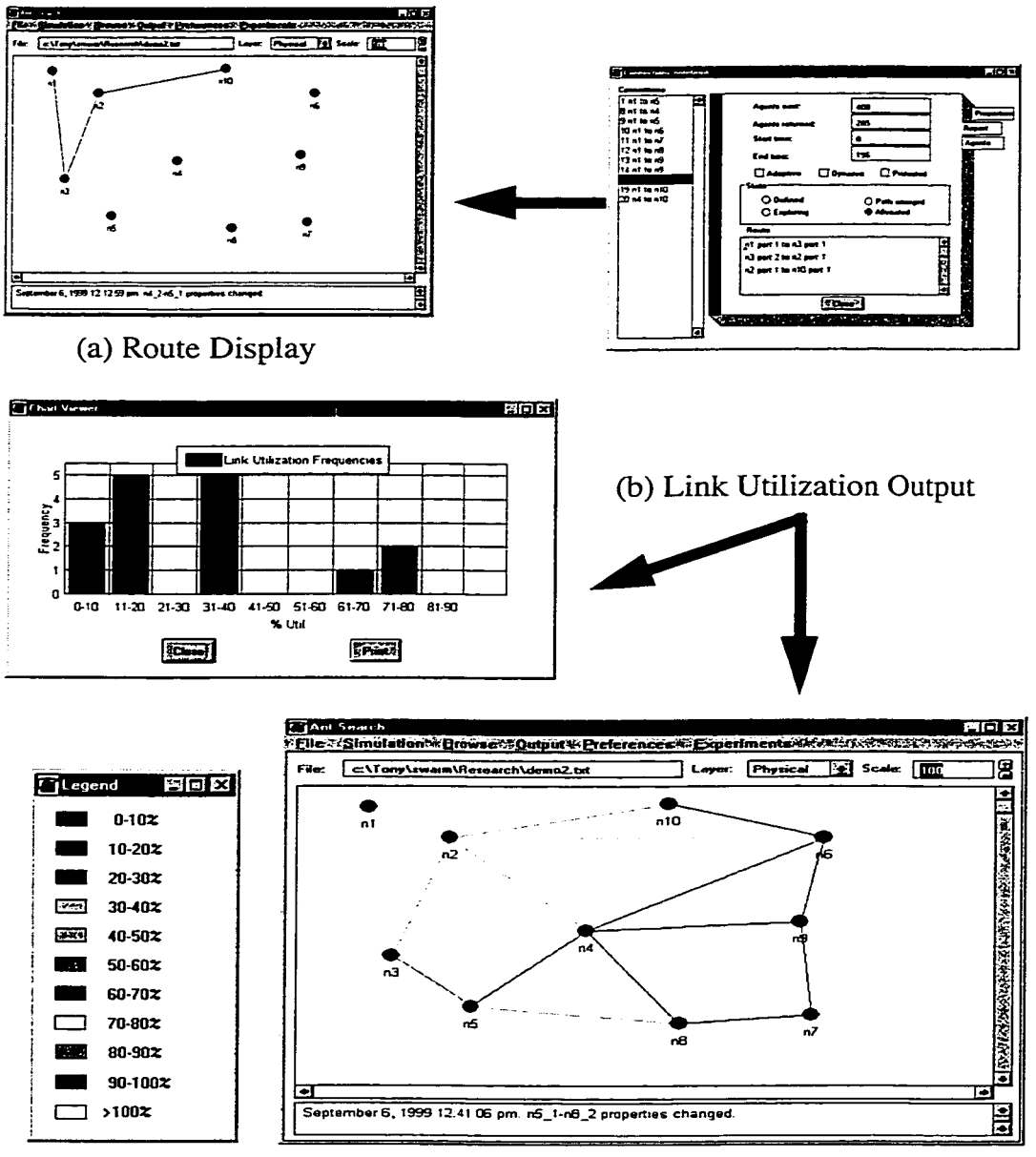


FIGURE 74. Output Displays

Figure 74a shows the route displayed for a particular connection selected from those defined in the simulation. Figure b shows the link utilization distribution for a particular simulation. along with an overall utilization of the links. A display showing only those links exceeding a user-defined threshold can also be displayed.

APPENDIX B *SynthECA Implementation*

B.1 Overview

This appendix describes the implementation of principal data structures that are used in the SynthECA environment. The environment relies heavily on the implementation of the Mobile Code Toolkit which is not described here. Considerable information on the Mobile Code Toolkit is maintained on the WWW at URL: http://www.sce.carleton.ca/netmanage/perpetuum.shtml.

B.2 SynthECA classes

B.2.1 ChemicalEncoding

The ChemicalEncoding class implements the chemical universe used in our implementation. The Binary Array Chemistry of order N was used in our implementation. The most important method in this class is the equals(ChemicalEncoding) method which returns true if the two encodings are equal, false otherwise.

B.2.2 Chemical

The Chemical class implements a chemical within our environment for the Binary Array Chemistry of order N. It has two attributes: encoding and concentration. The encoding attribute is an instance of the ChemicalEncoding class and the concentration is a float whose value is bounded by zero.

B.2.3 ChemicalMembrane

The ChemicalMembrane abstract class represents the interface between the agent and the local chemical environment. This class has two attributes: encoding and decisionFunction. The encoding attribute is an instance of the ChemicalEncoding class and the decisionFunction is an instance of a Receiver object. The Receiver class is part of the Mobile Code Toolkit and will not be described at length here. However, a Receiver object is an implementation of the Delegate design pattern and consists of a delegated object and method that are used to provide behavior not directly provided by the delegating object. In this case, the delegated object implements the decision function, not the membrane object.

This mechanism is used in order to have emitter and receptor decision function methods implemented in one place -- the agent class -- rather than proliferate classes with a single method implementing the decision function.

B.2.4 Reaction

The Reaction class is used to define the reactions that can occur between reactants, thereby generating products. This class has five attributes: reactants, products, rate, active

and name. The reactants and products are instances of the Array class and these arrays contain instances of Chemicals. Within the context of this class, the concentration associated with the Chemical is used to describe the number of molecules of the encoding that participates in the reaction. The rate attribute is a floating point value that determines the rate constant for the reaction. The active attribute is a boolean that is used to switch a reaction on and off. In some sense, it is equivalent to setting the rate attribute to zero when the active attribute has the value false. The name of the reaction is a user friendly description of the reaction that is intended to capture its function within the overall framework of chemical processing provided by a Chemistry object.

B.2.5 ChemicalEnvironmentInterface

The ChemicalEnvironmentInterface implements the interface to a ChemicalEnvironment. This interface allows a user to get, set, increment and decrement chemical concentrations. It is implemented by the ChemicalEnvironment and EnvironmentAccessController.

B.2.6 ChemicalEnvironment

The ChemicalEnvironment class stores the associations between chemical encodings and their concentrations. It implements the ChemicalEnvironmentInterface, described above. It has a single attribute, environment, an instance of Hashtable. This class implements the mapping rules defined for the Binary Chemistry of order N and is responsible for computing the concentration associated with a ChemicalEncoding.

B.2.7 ChemicalProcessor

The ChemicalProcessor class controls processing of chemical reactions within an agent. As such, it consists of the following attributes: reactions, tick, active, env and outputs. The reactions attribute is an instance of the Array class containing Reaction objects. The tick attribute is an integer that determines the frequency with which the processor updates its outputs. The env attribute is the chemical environment as perceived by the agent and is an instance of the ChemicalEnvironment class. The outputs attribute is an instance of an Array that, in turn, contains ChemicalMembrane instances.

B.2.8 ChemicalCommunicationInterface

The ChemicalCommunicationInterface defines two methods used for communication between an agent and the local chemical environment. The two methods: onInChange(Chemical) and onOutChange(Chemical) process chemical environmental changes that cross the agent's cell membrane.

B.2.9 EnvironmentAccessController

The EnvironmentAccessController class is a sub-class of VirtualManagedComponent (a Mobile Code Toolkit class) that implements the ChemicalEnvironmentInterface. The EnvironmentAccessController class is responsible for maintaining the concentrations of chemicals within the chemical environment defined for an individual node. This class has two attributes: controller and environment. The controller attribute is an instance of the SynthECAgent class that has the responsibility for controlling access to the environment and can be thought of as a resident agent representing a nodal chemistry. The environment attribute is an instance of the ChemicalEnvironment.

B.2.10 SynthECAgent

The SynthECAgent class is a abstract sub-class of the Mobile Code Toolkit SuperNetlet class that implements the ChemicalCommunicationInterface. The SynthECAgent class has the following private attributes: emitters, receptors, chemicalProcessor, localChemicalEnvironment and agentChemicalEnvironment. The emitters and receptors attributes are instances of the Array class that store ChemicalMembrane instances. The chemicalProcessor attribute is an instance of the ChemicalProcessor class and is responsible for the chemical reactions implemented within the agent. The localChemicalEnvironment is an instance of the EnvironmentAccessController class that permits access to the chemical information which is local to the node. The agentChemicalEnvironment attribute is an instance of the ChemicalEnvironment, and contains chemical information local to the agent.

Concrete sub-classes were defined for the scenarios implemented in this thesis.