Aprendizagem a Partir de Múltiplas Fontes em Grupos Heterogéneos de Agentes

---

Learning From Multiple Sources in Heterogeneous Groups of Agents

Luís Miguel Martins Nunes
(Mestre/M.Sc.)

Afiliações/Afiliations:
FEUP (Doutorando/PhD Student),
LIACC-NIAD&R (Investigador/Researcher),
ISCTE (Assistente/Assistant Teacher)

Contactos/Contacts:
ISCTE, Av. Forças Armadas, 1649-026 Lisbon, Portugal
Luis.Nunes@iscte.pt

Orientador/Advisor:
Professor Doutor Eugénio da Costa Oliveira (Ph.D.)

Tese apresentada para o cumprimento dos requisitos necessários à obtenção do grau de Doutoramento em Engenharia Informática pela Faculdade de Engenharia da Universidade do Porto.

Thesis presented for the fullfilment of the requirements necessary to complete the degree of Ph.D. in Computer Science at Faculdade de Engenharia da Universidade do Porto.

28 de Outubro de 2005 / October 28, 2005

Abstract

The field of Multiagent Systems (MAS) is concerned with software solutions composed of several autonomous elements (agents) that interact and communicate. The scenarios these techniques apply to have special characteristics that bring about new problems, but also provide new tools to develop adequate solutions. One of the research fields that is evolving in parallel and adapting to this new type of paradigm is Machine Learning (ML). Research in ML has been increasingly focused on the development of solutions that can deal with the problems posed by MAS. The contribution of ML to this field is of the utmost importance since adaptability and learning are fundamental in increasing agents' autonomy and flexibility.

This work presents a study concerning the relationship between communication and learning in a certain type of MAS. We focus on problems where we have different teams of agents, solving similar problems at different locations. Each of these teams may use different learning algorithms or heuristic solutions. In the past, learning-agents used solely the environment's feedback as a source of information for learning. MAS provide other sources of information that can increase agents' learning capabilities. Our goal is to determine how the communication of examples and reward information can affect the learning process.

The hypotheses posed in this thesis are: That communication can improve agents' learning performance for several learning algorithms in a specific type of problem; It is possible to enhance the benefits of communication by: using hybrid algorithms to integrate information from different sources, using heterogeneous environments and an adequate selection of information sources. These techniques are tested in three application domains: a "toy-problem" (predator-prey), a simulation with synthetic data (load-balancing) and one using real data (traffic-control).

During this study several variables that influence the performance of the exchange of information during learning where identified, namely: use of batch or specific information; online or offline integration; number of advisors; use of heuristic advisors; heterogeneity of the environment; type of algorithm used in the integration of external information. Although not exhaustive, due to the large number of possible combinations, our research tests the effects of several of these possibilities.

The initial expectations pointed towards the possibility of increasing the speed of learning and the performance by exchanging information, particularly when using heterogeneous environments. It was verified that exchanging information is beneficial, in terms of speed, performance and reliability. Contrary to our expectations the environments' heterogeneity and other tested techniques did not show the desired effects. This

fact is due, mainly, to the near-optimal performance of agents in most environments where agents are allowed to communicate.

Even though there is still a long path to follow in the quest for adequate solutions, this work provides, apart from the above mentioned contributions, a review of the main difficulties found during this research that may be helpful for those that follow this path in the future.

The ultimate goal of this line of research is to endow agents with the capability of learning from more sources than just the environments' feedback in a context where information is abundant. This new perspective of learning in MAS can lead to the development of new learning paradigms, specially suited for MAS, and take us one step further in the construction of autonomous and intelligent agents.

Abstract

A investigação em Sistemas Multi-Agente (SMA) tem como objectivo o desenvolvimento de soluções computacionais, baseadas em elementos autónomos (agentes) que interagem e comunicam entre si. Os cenários a que estas técnicas se aplicam trazem consigo um novo conjunto de problemas, mas também a possibilidade de desenvolver novas soluções. Uma das áreas de investigação que se tem desenvolvido em paralelo é a Aprendizagem Automática (AA). A pesquisa na área da AA foca, cada vez mais, o desenvolvimento de soluções destinadas a lidar com os problemas postos pelos SMA. A contribuição da AA neste campo é da maior importância dado que a capacidade de adaptação é fundamental para melhorar a autonomia e flexibilidade dos agentes.

Este trabalho estuda a relação entre a comunicação e a aprendizagem em certo tipo de SMA. Os problemas focados caracterizam-se pela existência de diversas equipas de agentes, cuja tarefa é a resolução de problemas semelhantes em locais diferentes. Cada uma destas equipas poderá utilizar diferentes mecanismos de aprendizagem ou soluções heurísticas. No passado, a aprendizagem baseava-se apenas na informação devolvida pelo ambiente. Com a introdução do conceito de SMA abrem-se novas possibilidades na utilização de outras fontes de informação. O objectivo deste trabalho é determinar quais os efeitos que a troca de certo tipo de informação (exemplos e recompensas) pode ter na aprendizagem.

As hipóteses postas nesta tese são: que a comunicação pode ser usada para melhorar o desempenho de vários tipos de agentes durante a aprendizagem em determinado tipo de problemas; É possível realçar os benefícios da comunicação através do uso de: algoritmos híbridos para a integração de informação de diferentes fontes, da utilização de ambientes heterogéneos e de uma escolha adequada das fontes de informação. Estas propostas são testadas em três problemas: o problema do "predador e a presa", uma simulação de distribuição de carga por servidores, e uma simulação de controlo de tráfego baseada em dados reais.

Durante o estudo foi identificado um conjunto de variáveis que pode influenciar o desempenho da troca de informação durante a aprendizagem, nomeadamente: transmissão em bloco ou caso a caso; integração imediata ou adiada; número de conselheiros; utilização de conselheiros pré-programados; ambientes homogéneos ou heterogéneos; modo de integração da informação. Embora não sendo exaustiva, devido ao grande número de combinações possíveis, a nossa pesquisa testa os efeitos de várias combinações de valores para as variáveis acima mencionadas.

As expectativas iniciais apontavam para a possibilidade de aumentar a velocidade e o desempenho com a troca de informação, em particular quando fossem utilizados ambientes heterogéneos. Verificou-se que a troca de informação traz benefícios quer

em termos de desempenho quer em termos de fiabilidade. Ao contrário das expectativas iniciais a heterogeneidade do ambiente, bem como outras técnicas empregadas para realçar o desempenho da troca de informação, não surtiram os efeitos esperados. Este facto deve-se, principalmente, ao desempenho *quasi* óptimo dos agentes em ambientes com troca de informação.

Embora haja ainda um longo caminho a percorrer na busca de soluções adequadas, este trabalho contém, além das contribuições mencionadas, uma enumeração das dificuldades encontradas durante esta investigação que poderá ser útil para aqueles que se futuramente dedicarem a este tema.

O objectivo último da linha de investigação prosseguida é a criação dos mecanismos necessários para a utilização da informação disponível em contextos onde esta é abundante. Esta nova visão da aprendizagem em SMA poderá levar ao desenvolvimento de diferentes paradigmas de aprendizagem, adequados às necessidades dos SMA, e ser um passo mais no sentido de sistemas autónomos e inteligentes.

Abstract

La recherche dans le domaine des Systèmes Multi-Agents (SMA) à pour objectif le développement de solutions informatiques basées sur des éléments autonomes (agents) qui interagissent et communiquent entre eux. Ces caractéristiques portent un nouvel ensemble de problèmes mais aussi la possibilité de développer des nouvelles solutions. L'un des domaines de recherche qui s'est développé en parallèle est l'Apprentissage Automatique (AA). La recherche dans le domaine la AA met en évidence de plus en plus le développement de solutions destinées à travailler avec les problèmes présentés par les SMA. La contribution de la AA dans cette domaine est de la plus grande importance vue que la capacité d'adaptation est fondamentale pour améliorer l'autonomie et flexibilité des agents.

Ce travaille analyse la relation entre la communication et l'apprentissage en certains types de SMA. Les problèmes abordés se caractérisent par l'existence de diverses équipes d'agents pour les quelles l'objectif est la résolution de problèmes identiques en locaux différentes. Chaque une des équipes pourra utiliser des différents mécanismes d'apprentissage ou des solutions heuristiques. Dans le passé, l'apprentissage s'est basé uniquement sur l'information de retour fournie par l'environnement. Des nouvelles possibilités dans l'utilisation d'autres sources d'information s'ouvrent avec l'introduction des SMA. L'objectif de cette étude est de déterminer les effets de la communication des exemples et des récompenses sur l'apprentissage.

L'hypothèses proposées dans cet étude sont: que la communication peut améliorer la performance des différents types d'agents durant l'apprentissage, dans certain problèmes; Est possible de rehausser les bénéfices de la communication avec l'usage: des algorithmes hybrides pour l'intégration de la information de différents sources, la utilisation des environnent hétérogènes, et une bonne choix de les sources d'information. L'hypothèses son testées en utilisant trois problèmes: "prédateur et la proie", une simulation de distribution de charge par des serveurs, et une simulation de contrôle de trafic basée sur les données réelles.

Durant cet étude, un ensemble de variables qui peuvent influencer la performance des échanges d'information pendant l'apprentissage on été identifiées, comme par exemple: transmission en bloque ou cas a cas; intégration immédiate ou stockage pour intégration en bloque; nombre de conseillers; utilisation de conseillers préprogrammées; environnements homogènes ou hétérogènes; mode d'intégration de l'information. Bien quelle ne soit pas exhaustif a cause du très grand nombre de combinaisons possibles, notre recherche teste les efforts de plusieurs combinaisons de valeurs pour des variables ci-dessus présentées.

Les expectatives initiales laissaient entrevoir la possibilité d'augmentation de la vitesse et de la qualité résultant des échanges d'information particulièrement en utilisations sur d'environnements hétérogènes. On a pu constater que l'échange d'information apporte de bénéfices soit en termes de performance soit en termes de fiabilité. Contrairement aux expectatifs initiales, l'hétérogénéité de l'environnement et d'outres techniques utilisées, n'ont pas produit les effets espérés. Ce fait peut être due à la performance quasi-optimale des agents quand ils échangent des informations.

Bien que cette recherche n'est pas finie on à énumérée les principales difficultés et étudié des différents solutions.

L'objectif final de cette recherche est la création des mécanismes pour la utilisation de la information obtenue par touts les agents qui approche un problème. Cette recherche pourra porter le développement des nouvelles méthodes de apprentissage plus approprié aux besoins des SMA.

The sooner you make your first 5000 mistakes
the sooner you will be able to correct them

in "The Natural Way to Draw",
Kimon Nicolaides, 1941

Acknowledgments

To my parents and wife without whom this would not have been possible.

To my advisor for the freedom he allowed me in choosing my path and his good advice and support at all times.

To all my colleagues and staff at FEUP/NIAD&R, ISCTE/DCTI and also to my friends, for their help and useful comments, with a special thanks to Rui Lopes, Luís Botelho, Ricardo Ribeiro, Pedro Figueiredo, Joaquim Esmerado, Nuno David, Luís Mota, Alexandre Almeida, José Moura and Luís Paulo Reis.

Our thanks, also, to the Traffic Control Department of Câmara Municipal de Lisboa for making the necessary data available for the traffic-control simulation. A special acknowledgement to the Open Software Foundation and all contributors to the software used during this work. Our thanks to the anonymous reviewers that made a serious analysis of our work and whose comments have been very helpful in the progress of this research. Our thanks to FCT/PRODEP, DCTI and LIACC for the financial support necessary for the research activities and for granting us the necessary time to develop them.

I could not finish these acknowledgements without thanking the persons from whom I have learned the most valuable lessons throughout these years, both in my profession as well as in life. Their competence, professionalism and character have been an example to me and to all those fortunate enough to have worked with them. To my great teachers, Professors Eugénio da Costa Oliveira, Luís Borges de Almeida, Fernando Côrte-Real and Manuel Menezes de Sequeira, my gratitude and respect.

Definitions, Abbreviations and Acronyms

The first item of the description, in parenthesis, is the section where the concept is defined in the text.

Action (section 2.2) An agent's decision that has consequences in the environment.

Action-dependent feature (section 6.2) A state feature that has a different value depending on the action being considered. *See also* Action and State.

Adaptable Parameters (section 2.2) *See* Hypothesis Parameters.

Agent (section 1.1) An autonomous element of a MAS environment whose decisions can affect state transitions. In this text all references to agent should be interpreted as Learning Agents.

Area (section 2.2) *See* Location.

Artificial Neural Network (ANN) (section 2.1.1) Evaluation function used mainly with the Backpropagation algorithm. The name is also used to define a computing paradigm that encompasses several types of functions, inspired by the research on how human brain-cells work, as well as the learning algorithms that apply to them.

Backpropagation (section 2.1.1) Supervised Learning Algorithm that adapts the parameters of an ANN to minimize the difference between the current response to a given input and the desired response.

Connectionist Q-Learning (section 2.1.3) Adaptation of Q-Learning, used for large search-spaces, that replaces the Q-table with an ANN.

Epoch (section 2.2) Period of time, consisting on a fixed number of turns. *See also* Turn and Trial.

Environment (section 2.2) Virtual or physical space in which the agents are immersed. *See also* Location.

Evaluation Function (section 2.2) A function that maps states to actions and contains adaptable parameters (called Hypothesis). The Evaluation Function can also be referred to as adaptive function. *See also* Hypothesis Parameters and Hypothesis.

Evolutionary Algorithms (EA) (section 2.1.4) Stochastic search algorithms, inspired by Darwin's theory, that represent each hypothesis as a specimen in competition with others for survival. Performance is seen as a measure of fitness and the fittest specimens are allowed to breed, so as to iteratively improve the best specimen or a whole population. This, broad, definition is subdivided in several sub-categories such as: Genetic Algorithms, Evolutionary Strategies and Genetic Programming *See also* Genetic Programming.

Genetic Programming (GP) (section 2.1.5) A sub-category of Evolutionary Algorithms where the base structures can be interpreted as program trees. *See also* Strongly Typed Genetic Programming.

Hypothesis (section 2.2) Set of parameters of an Evaluation Function that determines the mapping of states to actions.

Hypothesis Parameters (section 2.2) Adaptable components of an Evaluation Function. *See also* Hypothesis.

Learning Algorithm (section 2.1) Process of change that affects the Hypothesis based on acquired information and leads to an improvement of a certain measure of quality with respect to a certain class of tasks.

Learning Parameters (section 2.2) Set of label-value pairs that influence how a certain Learning Algorithm will change the Hypothesis.

Location (section 2.2) A sub-component of the environment where a team of Agents acts.*See also* Area.

Multiagent Systems (MAS) (section 1.1) Aggregation of autonomous computing entities (agents) jointly working in the same environment to reach, both, individual and collective goals. The name of a research area the focuses on problems where distributed and autonomous software/hardware is used. *See also* Agent.

Nash equilibrium (section 3.2.2) Situation that refers to a given set of hypothesis, where changes in one of the hypotheses alone will cause a decrease in performance.

Policy (section 2.2) A policy is the mapping made by an Evaluation Function according to a given Hypothesis *See also* Hypothesis and Evaluation Function.

Q-Learning (QL) (section 2.1.3) Reinforcement Learning algorithm based on the estimation of the quality associated with pairs of states and actions.

Reward (section 2.2) Scalar number, in this work always in $[0, 1]$, that measures an estimated quality of an action/state, or a sequence of actions/states.

Reward-based Learning (section 2.1) Class of learning algorithms in which the feedback information comes in the form of a quality evaluation. When given an example (usually called *state* in this case) the agent chooses an action and gets a reward (which may be delayed in time). The agent's objective is to maximize this reward, i.e. learn to choose actions that cause the environment to give high rewards as feedback. This label is proposed by Panait and Luke (2003).

State (section 2.2) A snapshot of the values of certain characteristics of the environment.

Strongly Typed Genetic Programming (GP) (section 2.1.5) A class of Genetic Programming algorithms where the nodes in the program trees have types. Mutation and crossover are bound by the use of these types. *See also* Genetic Programming.

Supervised Learning (section 2.1) Class of learning algorithms in which the feedback information is in the form of a desired response for each presented example. The agent's job is to emulate, generalize (and in some cases, describe the underlying rules that govern) the behavior represented by the example-response pairs.

Trial (section 2.2) A sequence of events consisting of a given number of epochs *See also* Epoch and Turn.

Trust (section 4.2) In this thesis trust is a coefficient that is proportional to the estimated efficiency of the advice given by a certain advisor.

Turn (section 2.2) Period of time between two observations of the environment, in which the agent may issue one or more actions *See also* Epoch and Trial.

Unsupervised Learning (section 2.1) Class of learning algorithms where there is no feedback information, the learning agent's job is to divide a set of examples into a certain number of classes according to a given measure (distance between examples in most cases). The evaluation of the results depends on the application, although in most cases what is requested is that the learning agent finds a division that results in a compact and well defined partition of the example space.

Symbols

One of the main concerns related to notation was to maintain the coherence of the meaning for the symbols used, with the exception of chapters 2 and 3, "Background Concepts" and "Related Work", respectively, where notations followed, as much as possible, the most common norms for each specific subject. In other chapters coherence had to be balanced with other concerns such as: clarity, size of equations and readability, which forced minor adjustments in different contexts. For these reasons, the indexes and function parameters are sometimes omitted, whenever they are not relevant for the current explanation. Even though all symbols are defined the first time they are used in each section, here we list their default meaning through the text. Whenever a symbol has a different meaning this will be explicitly defined.

Variables are uncapitalized, e.g. (x), vectors are indicated using a bar above the name, e.g. $\bar{x}$. The first letter of symbols representing sets is capitalized.

The indexes i,j and k are mainly used as the identifier of agents in a set. k is mainly used as the identifier of an agent selected according to a certain *criterium*. t is always used to define the time of a certain event or the index of a sequence of values for a given variable.

The first item of the description, in parenthesis, is the section of the symbol's first appearance in the text.

$\mathcal{A}_{i,t}$ (section 2.2) The set of actions available for agent i at time t.

$a_{i,t}$ (section 2.2) Action take by agent i at time t. The action may also be represented as a vector $\bar{a}_{i,t}$ representing the adequacy or probability of choosing each possible action.

$\mathcal{B}$ (section 2.2) Set of learning parameters that control a certain learning function.

$\mathcal{E}_{i,t}$ (section 2.2) Previous experience obtained by agent i until a certain time t. Each element of this set contains tuples with, at least, three values, namely: state, action and reward.

$\mathcal{F}_{k,i}$ (section 2.2) Evaluation Function of type k used by agent i.

$\mathcal{H}_{i,t}$ (section 2.2) Hypothesis in use by agent i at time t.

$\mathcal{H}_i^*$ (section 2.2) Set of optimal hypothesis parameters for a given agent i considering the current state and dynamics of the environment.

$\hat{\mathcal{H}}_i^*$ (section 2.2) Hypothesis with the best estimated performance found by agent i at time t.

$\mathcal{L}_i$ (section 2.2) Learning function that transforms the hypothesis' values, based on previous experience.

$\pi_{i,t}$ (section 2.2) Snapshot of the policy used by agent i at time t.

Φ_l (section 2.2) The set of all agents acting in location l.

$Q_t(\bar{s}, a)$ (section 2.1.3) Estimated quality of taking action a at state $\bar{s}$ at a given time t.

$\bar{Q}_t(\bar{s})$ (section 2.1.3) Estimated quality of taking each of the possible actions at state $\bar{s}$ and time t.

$R_{l,n}$ (section 2.2) Reward achieved by the team controlling location l in epoch n.

$r_{i,t}$ (section 2.2) Reward achieved by agent i after issuing an action at time t.

$r_{l,n}$ (section 2.2) Reward achieved by a team of agents in location l at epoch n.

$\bar{s}_{i,t}$ (section 2.2) State of the environment from the point-of-view of agent i at time t.

$\bar{\mathcal{S}}_{l,t}$ (section 2.2) State of the environment at location l and time t.

T (section 2.1.3) Temperature parameter for Boltzmann selection.

$w_{n,ij}$ (section 2.1.1) Weight of an ANN connecting unit i of layer n to unit j of layer $n+1$.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the past decades the concepts of *decentralization* and *autonomy* concentrated the attention of researchers in most areas of computational sciences. The facts in favor of decentralized and autonomous software solutions are well known and have been proved in practice by its growing use. The main factors that are usually pointed out in favor of these solutions when compared to centralized ones, are:

- Better resistance to failure, e.g. distributed network management;
- More efficient access to decentralized resources, e.g. distributed Databases;
- Decrease in the number of critical bottlenecks/elements, e.g. distributed routing;
- A large number of small and cheap computational elements can overpower centralized processing solutions, e.g. SETI program (SETI, 2005);
- The ability to cope with situations where direct human/centralized control is not possible, e.g. Mars Rover project (MarsRover, 2005);

Although centralized solutions still have — and will certainly keep — their place in many areas, it is increasingly obvious that there are a number of problems where decentralized software is a better option. In some cases it is the only option. It is important to notice that when speaking of these concepts we are, in most cases, referring to "a certain degree of decentralization/autonomy" and that the efficient balance of this degree is always a problem-dependent feature.

The field of Multiagent Systems (MAS) is currently the ground for most of the research involving this type of software. We will not go into the discussion of "what is an agent?". This discussion has been a controversial one in the recent past. Now, that the dust has settled, there are a certain number of characteristics that seem to be consensual about this type of software, namely: a relative degree of autonomy in the

decision process, the ability to, pro-actively, interact with the environment and other agents and the fact that it is immersed in a "society".

The MAS area merges the research from several other areas, some of which were unrelated in the past. One of these is Machine Learning (ML) that was also faced with the problems and advantages put forth by MAS. Some authors argue that true autonomy can only be achieved through adaptability and learning. But, complex MAS, add several challenges to the, already difficult, problem of single-agent automated learning, namely:

- Partial observability;
- Non-static environments, due to:
  - Constant changes in other agents' policies;
  - External/Unpredictable events;
- Integration of, often conflicting, local solutions;
- Large search-spaces (when considering the information collected by all agents);

Of course most of these problems had already been approached, but they were usually tackled individually. Their appearance as an ensemble creates a new problem, for which some of the proposed solutions are ineffective. Given the growth in popularity of MAS the demand for adequate solutions is now higher than ever. The reviews of Sen (1997), Weiss and Dillenbourg (1999) and Kazakov and Kudenko (2001) corroborate this view, mentioning learning and cooperation between autonomous agents as important subjects of research that deserve attention and are likely to become the main features of this new generation of software.

Several approaches have been attempted for these problems, but none which considers the possibility of joining the efforts of teams that use different learning algorithms by exchanging information between them. The strength of diversity has been proven in nature as in human societies. Communication was once the main breakthrough in human evolution (some say, the main difference between humans and other animals) and is now a key factor in the advance of science. Is learning in MAS an exception, or can these two concepts (communication and heterogeneity) be used to our advantage in this problem too? This work will attempt to take us (and the reader) one step further, however small it may be, towards an answer to this question.

1.2 Thesis Question

As we mentioned in the previous section there are several new problems but also new advantages to explore in MAS. One of the main advantages is that there is more information available in MAS than in traditional ML scenarios. Establishing if/how

this additional information may be used advantageously to improve an agents' learning process is the main concern of this work.

As hinted above the question we strive to answer is: "**(How) can heterogeneous teams of learning agents benefit from exchanging information during the learning process?**" or, rephrasing it as a working hypothesis: Can communication improve agents' learning performance, for several learning algorithms in a specific type of problems? Is it possible to enhance the benefits of communication by: using hybrid algorithms to integrate information from different sources? using heterogeneous environments and/or an adequate selection of information sources?

It is important to establish that when we speak of "heterogeneous teams" we mean that each team may use different learning algorithms. We have not approached the problems related to heterogeneity of learning algorithms within the group. There are no restrictions in our proposals that preclude these scenarios, but no effort was done to investigate or solve the particular problems posed by intra-team heterogeneity.

It is also important to define that our approach is in a middle ground between the definitions of *Concurrent Learning* (MAS where each agent learns on its own) and *Team Learning* (where teams learn as a whole). These definitions are proposed by Panait and Luke (2003).

In the course of this research several questions were raised. This report will try to guide the reader through the problems that emerged and propose a set of solutions that will be evaluated and discussed. Some of the questions discussed in this work are:

- What information is useful to the learning task, other than the one directly provided by the environment?
- How can agents select the appropriate information source for a given situation?
- How can different types of information be integrated in an agent's learning process?
- What are the consequences of communication between agents that use different learning algorithms in similar problems?

Much of the related work, focused on section 3 of this report, only considers parts of the problem, specific instances of single-agent environments, the use of expert advisors, or situations where all agents use similar learning algorithms. In this work we have tried to broaden the spectrum of possible environments and agent types. It is likely that specific solutions to some of our test-problems may show a better performance than our approach. Nevertheless, this study is important and, to the best of the author's knowledge, original work.

The problems under scrutiny are related to many others and it is important at this moment to "draw-the-line" on what is the focus of this research. This work is *not* concerned with: explicit coordination protocols, ontologies, social behavior dynamics (except for learning-related behaviors), concurrence, synchronization, security, non-cooperative/malicious agents, or the solution to the problems of any specific

application-domain. The experiments were designed with the sole purpose of verifying the quality of the approach we present in paradigmatic instances of problems in which several teams of autonomous learning agents may be used to solve similar problems at different locations.

1.3 Objectives and Approach

Our objective is to improve the efficiency of several teams of learning agents using communication. We aim at results in which all teams achieve better performances in scenarios where they are allowed to communicate than in those where this option is not available.

We study the effects of exchanging of information between agents of different types during the learning process. In the experiments we made, we use several different learning algorithms as well as pre-programmed agents with fixed (heuristic) behaviors. All teams will be attempting to solve similar problems and trying to use communication with their peers to improve their skills.

In this work we experiment with one well known toy-problem and two simulated application domains (load-balance and traffic-control), but the model is not restricted by any particular feature of these application-domains. Other MAS applications can also use our model, provided that: they learn from environment rewards, and there are several communicating teams of learning agents, solving similar problems, at different locations. In our experiments, we have chosen a few learning algorithms to demonstrate how the concepts could be applied. The choice of these learning algorithms is related to their adequacy in solving the type of problems we intended to address and with a concern to cover different types of algorithms, which should lead to a more general set of solutions. Even though the efficiency of some proposed solutions is algorithm and problem-dependent, the same concepts can be applied to other problems and learning algorithms.

Our approach can be used directly in a real environment as much as any learning solution that depends on learning from errors. We believe, however, that validating this model requires a cycle that includes simulation, testing and deployment phases. In the first phase agents would learn in a simulated environment, that follows, as closely as possible, the dynamics of the real system to which a solution is required. After generating a set of policies (and possibly using fixed adaptation parameters) the solutions must be tested in a real environment under strict supervision, either human or automatic or in a more detailed simulation. Finally, if tests prove the efficiency of the new solutions, these can be deployed on a real situation and the changes in the environment's dynamics propagated back to the simulation where a new learning phase would start. This type of cycle is well known in software engineering but seldom considered in the automatic employment of adaptable solutions. The evaluation cycle is hinted

by our architecture although the impossibility of testing in real systems does not allow us to complete the cycle and determine the efficiency of this solution when applied to real-life problems. In the experiments reported here all phases run on simulated environments, but a new project [1] related to traffic-control, which is currently in its startup phase, may allow us to go one step further in this ideas.

Another important question related to this approach is its communication cost. We have not been excessively concerned with this matter, although aware of it. The limits considered for communication were thought of in the perspective of having as few autonomy restrictions as possible, not in saving bandwidth. It is more important for us that an agent does not depend on information sent by another to take its decision than to reduce the total amount of information exchanged. Stone and Veloso (1997) argue that unrestricted communication between *homogeneous* agents leads to a scenario where multiagent learning is similar to centralized learning. We agree with this point of view, but it is important to make it clear that this is a different situation. The agents considered in this research are *heterogeneous* in the sense that an agents' policy is not necessarily the best for another agent. A solution where all agents learn the same behavior is usually suboptimal and often even a disastrous one. In the situations focused here each agent must preserve its autonomy and specialize in its own task, even though there are common points in the agents' tasks within the team or in different teams. This is true even if, at first, any agent can fulfill any of the roles, or if the roles are not strictly defined. In (Stone and Veloso, 1997) only the above-mentioned form of heterogeneity is considered. In this work we have heterogeneity at different levels: not only are the agents within a team required to have different policies, but also, each team may use different learning algorithms. Throughout this text the word " *heterogeneous*" will be used in the broader sense.

1.4 Contributions

The main contributions of this work are: the extension of the results on the effectiveness of communication to different environments, problems and learning algorithms; and a set of techniques that are aimed at enhancing the use of multiple sources of information during learning.

In this work we report our study of the effects of communication between agents that use different learning algorithms. Nearly all of the previous approaches to this type of problems considered solely agents that use Q-Learning and/or have expert advisors. Also, most of the related work focuses single-agent problems, as can be seen in chapter 3. We extend the study to teams that use different learning algorithms, and non-expert advisors. Another important contribution is the definition of the foundations of an ar-

[1] The GRICES/CAPES Project, approved for funding in 2005

chitecture that supports the integration of communication from different sources during learning.

Along the way several minor contributions were added, namely: the development of particular types of hybrid algorithms, specially suited for integrating different types of information; the adaptation of Strongly Typed Genetic Programming/ID3 to real-valued program trees; the use of other agent's reward information for the adjustment of learning parameters; and the exploration of the concept of learning-related trust.

During this study several variables that influence the performance of the exchange of information during learning were identified and studied, namely: use of batch or specific information; online or offline integration; number of advisors; use of heuristic advisors; heterogeneity of the environment; type of algorithm used in the integration of external information.

The initial expectations pointed towards the possibility of increasing the speed of learning and the performance by exchanging information, particularly when using heterogeneous environments. It was verified that exchanging information is beneficial, in terms of speed, performance and reliability for several types of learning agents, but, contrary to our expectations, the environments' heterogeneity and other tested techniques did not show the desired effects. This fact is due, mostly, to the near-optimal performance of agents in most environments where agents are allowed to communicate.

1.5 Reader's Guide

In this thesis we have tried to present the subject in increasingly complex layers. The reader will see the same subjects at several points of this discussion and their descriptions will become increasingly more detailed. Even though it may appear repetitive, we believe that the top-down approach is easier to follow given the amount of details present at the last level (which would, eventually, be the code itself).

Following these introductory notes, chapter 2 describes the standard versions of the learning algorithms used in our approach and reformulates the learning problem to enhance the special characteristics of learning in the type of MAS we are interested in studying. The first section of chapter 2 can be skipped by those familiarized with the algorithms described in it.

Chapter 3 contains a review of related work, both to give the necessary credits to those *"whose shoulders we stand on"* and to situate the reader in terms of the research conducted in the area in the past decade.

Chapter 4 contains an analysis of the problem, a set of proposals to deal with it along with the arguments that led us to take these options rather than others. In chapter 5 we define the architecture of the MAS and the characteristics of the agents that compose it. While doing this we present a more detailed explanation of the implementation of the concepts discussed in chapter 4.

In chapter 6 we explain what kind of experiments were conducted, their objectives and detailed descriptions. Chapter 7 presents and discusses the results achieved. Finally, in chapter 8, we present a summary of the conclusions and directions for future work.

Appendix A.1 contains a review of the abandoned tracks and unsolved problems related to this work. Appendix B contains details on the simulators' design and construction. In Appendix C we present the full set of results and parameters used in the experiments. Finally, Appendix D contains a critical review of (Whiteson and Stone, 2004) exposing a flaw that originated a correction in the above-mentioned paper.

In this chapter we introduced the area of research, the type of problems we are concerned with and provided the first hints concerning the type of solutions we propose. We have clearly stated the question we address and summarized the main contributions of this work. The following chapter is dedicated to an overview of some background concepts related to Machine Learning and Multiagent Systems.

Chapter 2

Background Concepts

In this chapter we will take a closer look at the type of learning problems we will be addressing and at some of the tools inherited from Machine Learning (ML) to deal with these problems. In the first section we will present an overview of standard versions of the learning algorithms that will be part of the integrated learning process described in chapter 4. The following section analyses in detail the type of problems we are attempting to deal with.

2.1 Overview of Learning Algorithms

The most common subdivision of learning algorithms is done according to the type of feedback they need. When adopting this division we have three main categories: *Unsupervised*, *Supervised* and *Reward-based* (a.k.a. Reinforcement). A similar division can be done in the type of learning problems (although some problems can be posed in different ways, falling in different categories). These categories are used here to describe different types of learning algorithms. We use *Reward-based* to describe algorithms that can learn from a qualitative feedback, and Reinforcement Learning to mention a specific family of these algorithms, inspired by dynamic programming (Sutton and Barto, 1987).

Unsupervised Learning: no feedback information, the learning agent's job is to divide a set of examples into a certain number of classes according to a given measure (distance between examples in most cases). The evaluation of the results depends on the application, although in most cases what is requested is that the learning agent finds a division that results in a compact and well defined partition of the example-space.

Supervised Learning: the feedback information is in the form of a desired response for each presented example. The agent's job is to emulate, generalize

(and in some cases, describe the underlying rules that govern) the behavior represented by the example-response pairs.

Reward-based Learning: the feedback information comes in the form of a quality evaluation. When given an example (usually called *state* in this case) the agent chooses an action and gets a reward (which may be delayed in time). The agent's objective is to maximize this reward, i.e. learn to choose actions that cause the environment to give high rewards as feedback.

In the the next few chapters we will mention several types of well known learning algorithms, from two of the types discussed above: Supervised and Reward-based Learning. We will describe (briefly) the standard versions of these learning algorithms, so that later on, in chapter 4 we can focus on the changes that were made to integrate these learning algorithms in our agent's architecture. Readers that are familiarized with these algorithms can skip this section.

2.1.1 Backpropagation

Backpropagation (BP) (Rumelhart *et al.*, 1986) is, perhaps, the most well known Supervised Learning algorithm. The concept is that for a given function (equation 2.2), which contains a set of real-valued parameters (usually called weights) it is possible to reduce the mean squared difference between the actual output, $F(\bar{x})$, for a given example $\bar{x}_t$, and a certain desired value, $\bar{d}$, by changing the parameters. The function, $F(\bar{x})$, must be differentiable in relation to each of these parameters. This is done as shown in equation 2.1, using, in this case, the squared difference as error function.

$$\begin{aligned} e &= \|\bar{d} - \bar{F}(\bar{x})\|/2 \\ \Delta w_j &= -\alpha \frac{\delta e}{\delta w_j}, \end{aligned} \tag{2.1}$$

In equation 2.1 α represents the learning-rate parameter and $\delta x/\delta y$ is the partial derivative of x in relation to y. Δw_j is the amount added to parameter w_j to bring $\bar{F}(\bar{x})$ closer to $\bar{d}$. The learning-rate, α, is either fixed from the beginning the training or reduced during training to ensure convergence.

Since, for most functions, it is not efficient to compute these derivatives, BP is commonly associated with a type of functions for which this computation is efficient. These functions were called Artificial Neural Networks (ANN), a name that is often confused with the BP algorithm itself and its variants. ANN are (usually) organized in layers of computational units that perform a non-linear function of the sum of the activations of all incoming connections. Given an input $\bar{x}$ the ANN calculates the

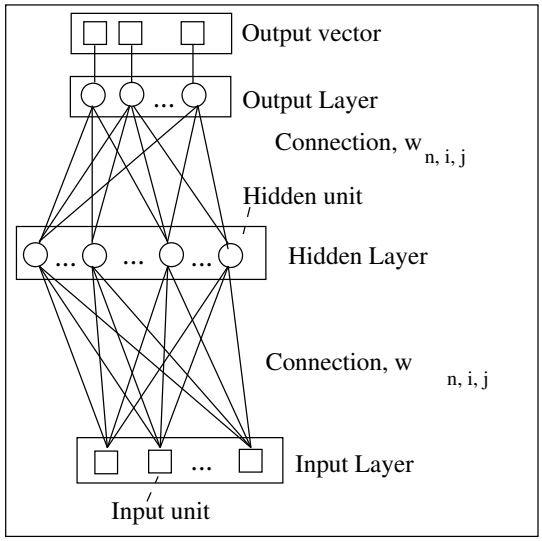

Figure 2.1: ANN with three, fully connected, layers. Circles represent nodes that sum their inputs an perform a non-linear transformation, while squares represent input and output variables. Rectangles group a set of related variables or nodes (layer). The lines represent links that carry the result of the lower nodes' computation to the upper nodes multiplying it by a weight factor.

output $\bar{F}(\bar{x})$ in the following way:

$$\begin{aligned} y_{1,j} &= f(\sum_{i=0}^{M_0} w_{1,ij} \cdot x_i) \\ y_{n,j} &= f(\sum_{i=0}^{M_n} w_{n,ij} \cdot y_{n-1,i}), 1 < n \leq N \\ F_j(\bar{x}) &= y_{N,j}, \end{aligned} \tag{2.2}$$

where n is the layer number, i and j are the indexes defining a connection between two nodes in consecutive layers and $f(x)$ is a sigmoid function, of which the most commonly used examples are: $tanh(x)$ and $1/(1+e^{-x})$. M_n is the number of nodes in layer n and N is the number of layers of the ANN. The number of nodes in the first layer (M_0) is considered to be the dimension of the input vector $\bar{x}$. For each node in all but the first layer there is also a parameter called *bias*, which is often represented as parameter 0 of that layer ($w_{n,0j}$) and it is assumed to be associated to a permanently activated node, i.e. $y_{n-1,0} = 1, \forall n \in \{1, 2, ..., N\}$).

Usually, the adaptation shown in equation 2.1 is not done for each example presented but accumulated for a set of examples before each change in the parameters. This is usually referred to as "batch" or "offline" learning, by opposition to "online" learning where each example-response pair triggers a change in the hypothesis parameters.

Although it was a breakthrough at the time of its creation, the standard implementation of BP was still a relatively slow procedure, because it required many examples to be repeatedly presented in order to achieve a reasonable approximation of the desired function. One of the (many) processes found to reduce the training time was labeled Adaptable Learning Rates (ALR) (Silva and Almeida, 1990). This method proposes an adaptation, not only of the weights, but also of the learning-rates α_j that are, in this case, different for each of the hypothesis' parameters. In summary, this technique proposes that the learning rate is increased when two consecutive adaptations of a parameter point in the same direction (i.e. the sign of Δw_j is the same) and decreased otherwise. This technique also proposes backtracking and a drastic cut in the learning-rates when the total error of the network increases during an epoch to avoid overshooting a local minimum.

We will not go into further detail on BP or ANN here, since this is consolidated work, but the interested reader is advised to lookup (Haykin, 1999) for an extensive discussion of all subjects related to this type of learning algorithms.

2.1.2 ID3

The *ID3* algorithm (Quinlan, 1986) was created having in mind the supervised classification of examples composed of discrete-valued attributes. The extension to continuous attributes was later proposed by Fayyad (1991). The information required to execute this algorithm is a set of examples, each classified in one of N classes. We will focus this explanation, for simplicity, in the example of just one class and consider as positive examples those that belong to the class and the remaining as negative. Table 2.1 summarizes the main steps of ID3.

The main problem is to find the attribute that best classifies the set of examples E. Each member of E is composed of a vector of values, one for every corresponding attribute in a. This is equivalent to finding which partition of E into subsets has the highest information gain. The information gain of partitioning E using attribute a is defined by:

$$Gain(E, a) = Entropy(E) - \sum_{\forall vi} \frac{\#E_{vi}}{\#E} Entropy(E_{vi}), \tag{2.3}$$

where $\#E$ is the number of elements in E, E_{vi} is the set of examples in which a evaluates to v_i, and the $Entropy$ function is defined as:

$$Entropy(E) = \sum_{\forall c} -p_c \, log_2(p_c), \tag{2.4}$$

where c stands for each possible classification of an example from E and p_c is the proportion of elements in E that belong to class c.

Entropy can be seen as a measure of the amount of information contained in a set. Consider a binary classification and a set in which we know the proportion of

Table 2.1: Summary of ID3 Algorithm for binary classification, adapted from (Mitchell, 1997). E stands for a set of examples classified as positive or negative and A is a set of attributes that compose each example. Node$_+$ stands for a positive classification, Node$_-$ for a negative.

ID3(E, A)
Create new node: Root;
IF all $e \in E$ positive, THEN return Root = Node$_+$;
IF all $e \in E$ negative, THEN return Root = Node$_-$;
IF $A = \emptyset$ return Root = Node$_{label}$,
where $label$ is the most common value ($+$ or $-$) for the examples in E;
LET a = Element of A that best classifies the examples in E (see equation 2.3);
Set decision attribute for the current node to a;
For each possible value (v_i) of a:
Create new branch of Root, corresponding to the test $a = v_i$;
Select E_{vi}, the subset of E for which $a = v_i$;
IF $E_{vi} = \emptyset$ THEN branch$_{vi}$ = Node$_{label}$,
(most common classification for the examples in E);
ELSE branch$_{vi}$ = ID3(E_{vi}, $A - a$);
Return Root

elements of each class it contains. If the set has elements of only one class (minimum entropy) guessing the correct classification of a random element of the set is trivial. But, if the elements of the set are evenly distributed through both classes (maximum entropy), classifying a random element without analyzing it further is a random guess, with equal chances of succeeding or failing. The objective of ID3 is to reduce the problem to subsets with the least possible degree of entropy at each stage. The gain is a measure of the reduction of entropy that can be achieved by partitioning the set according to a certain attribute. Thus, the attribute a that is best suited to partition a certain set E is the one with highest gain.

If we want to consider continuous attributes we will need to change the test $a = v_i$ into a function $f(a, v_i)$ that returns a boolean value, e.g. $f(a, v_i) = (a > v_i - K) \wedge (a < v_i + K)$ with $K > 0$. The only restriction is that in all decisions there can be only one i such that $f(a, v_i) = true$.

2.1.3 Q-Learning

Q-Learning (QL) (Watkins, 1989) is the most commonly used learning algorithm in the class of *Reinforcement Learning* which is a subclass of what we have labeled *Reward-Based learning*. Its most simple form (one-level Q-Learning), is based on a table that stores the estimated quality, $Q(\bar{s}, a)$, of performing action a at state $\bar{s}$, for

every possible state-action pair. When a reward r_t is received, at time t, the value of $Q_t(\bar{s}, a)$ is updated as follows:

$$Q_{t+1}(\bar{s}_t, a) = (1 - \alpha)Q_t(\bar{s}_t, a) + \alpha(r_t + \beta Q_{max}(\bar{s}_{t+1})), \tag{2.5}$$

where $\bar{s}_{t+1}$ is the state of the environment after performing action a at state $\bar{s}_t$, $\alpha \in]0, 1[$ is the learning rate and $\beta \in [0, 1[$ a discount factor applied to the estimated quality of the next state, that is given by:

$$Q_{max}(\bar{s}_{t+1}) = \max_a(Q(\bar{s}_{t+1}, a)), \tag{2.6}$$

for all possible actions a when the system is in state $\bar{s}_{t+1}$. The learning rate α is usually decayed during training. When prompted to choose an action for a given state the Q-Learning algorithm will fetch the values of $Q_t(\bar{s}, a)$ for all available actions a. There are several strategies to chose an action given a set of Q-values. The main dilemma is the exploration/exploitation tradeoff. One of the most used techniques to regulate this compromise is Boltzmann selection. In this case an action a is selected with probability $p(a|\bar{s})$ that is given by a Boltzmann distribution,

$$p(a|\bar{s}) = \frac{e^{\frac{Q_t(\bar{s},a)}{T}}}{\sum_{i \in A_s} e^{\frac{Q_t(\bar{s},i)}{T}}} \tag{2.7}$$

where T is a temperature parameter, decayed during training, and A_s is the set of all actions available from state $\bar{s}$. This allows an emphasis on exploratory behavior when T is high that gradually shifts to exploitation as T decreases.

It is often the case that the number of state-action pairs is too big to represent all possible combinations explicitly. One of the most popular solutions to this problem is *Connectionist Q-Learning* (ConnQL) (Lin, 1992). In this approach the table that stores $Q(\bar{s}, a)$ is replaced by an approximation function that is trained to map state-action pairs to its estimated quality. Usually an ANN encodes this mapping using standard online *Backpropagation* (section 2.1.1) as an auxiliary learning algorithm. In this case the update in equation 2.5 is replaced by the presentation of an example-response pair to the ANN in which the input is the concatenation of the state and action vectors and the desired output is set to:

$$\bar{d_t} = r_t + \beta Q_{max}(\bar{s}_{t+1}) \tag{2.8}$$

2.1.4 Evolutionary Algorithms

Evolutionary Algorithms (EA) (Holland, 1975; Koza, 1992) are a well known set of learning techniques inspired by the processes that rule the evolution of species in nature. The hypothesis' parameters are interpreted as a specimen (or a phenotype from which a specimen can be generated), and its performance in a given problem as its

fitness. After the evaluation of all the specimens, the ones with best fitness are selected for breeding. The selected specimens are then mutated and crossed-over to generate a new population. There are countless variations on the structure of the specimens the crossover and mutation procedures. The type of EA chosen for our experiments was inspired by the one used in (Glickman and Sycara, 1999). According to the definitions in (Kantrowitz, 1997) this approach can fall in two different sub-categories, labelled *Evolutionary Programming* and *Evolutionary Strategies*. Even though Glickman and Sycara (1999) refer to this technique as a Genetic Algorithm (GA), the use of this term to define the technique is debatable according to (Kantrowitz, 1997). We have adopted the broadest category, *Evolutionary Algorithms*, to avoid possible misinterpretations.

An EA keeps a set of specimens (called population). In each step of its life cycle, it evaluates all specimens. At the end of each cycle a given number of specimens is selected for breeding. Glickman and Sycara (1999) use tournament selection. Tournament selection consists in testing a small subset of the population in parallel and selecting the one with highest fitness (highest accumulated reward) to pass to the next generation. Tournaments are repeated until the required number of selected specimens is achieved.

Breeding consists in applying mutation and/or crossover operators to one, or a pair of, selected specimen(s) to generate a specimen for a new population. The specimens in this case contain the weights of an ANN with fixed dimensions.

Mutation is done by adding a certain amount to the value of a parameter with a certain probability (*mutation probability*). The amount to be added is randomly generated with normal distribution and zero average. The variance of the distribution will depend on a learning parameter labelled *mutation rate*, that is, usually, decayed during training to ensure convergence.

Crossover was not used by Glickman and Sycara (1999), but there are many strategies to select the partitioning of the parameters in subsets for ANN. For a review on this please refer to (Salustowicz, 1995; Yao, 1999). The description of our own approach to crossover as well as other changes made to this approach will be detailed in section 5.2.3.

2.1.5 Strongly Typed Genetic Programming

Strongly Typed Genetic Programming (STGP) (Haynes *et al.*, 1995a) relies on a process that is very similar to the above description for EA. All steps down to the mutation and crossover details are the same, but the underlying evaluation function is different: a program tree. A program tree consists of a number of nodes, which contain instructions (inner nodes) or class labels (leafs). When evaluated, the control flow will follow an evaluation path consisting of selection instructions, gated by boolean instructions leading to a leaf. The label of this leaf will determine to which class the state belongs to.

What is particular to STGP, when compared to other Genetic Programming (GP) approaches, is the fact that it takes into account the input and output types of each node of the program tree when mutation and crossover are applied. By enforcing these restrictions it will always generate trees that can be evaluated for all possible input patterns and reduce the search-space eliminating programs that may result either in invalid or useless evaluations. Another interesting characteristic is that problem-specific functions can be inserted into the instruction nodes. For example, if the designer thinks that a certain combination of values of the state can be useful, he can code it and make it available to the STGP algorithm. The algorithm will use it as it sees fit in the generation of new programs. It is far easier to program, for example, the norm of a state vector, than to expect a search mechanism to "discover" it on its own by random combination of sums and multiplications.

The initial trees are randomly generated, respecting the restrictions posed by the types and with a given (pre-set) probability of generation for each node type. The root-node is always a selection instruction to avoid trivial trees (e.g. with a leaf as a root node). The maximum depth can also be pre-set to restrict the complexity of the generated trees. When maximum depth is reached the generation of a leaf node is forced.

Trees are mutated by applying a mutation operator to each node. If a given random number (uniform in [0,1]) is smaller than the current *mutation probability* the node will suffer a mutation. The type of mutation will depend on the node it is applied to. Nodes with subtrees can delete and regenerate them (using the same procedure as in random initialization); Nodes containing data can change it, e.g. by disturbing a constant value with a certain amount of noise; A node can also change the type of function it performs, as long as the input and output types are maintained.

Crossover is usually done by selecting subtrees from different parents and pasting them into a new tree, always respecting the nodes' input and output types.

We have reviewed in this section some specific instances of algorithms used in the past to deal with learning problems in different ways. Now we will focus our attention in the type of problems we will study.

2.2 Learning in Multiagent Systems

The most concise, and broadly accepted, definition of learning in ML is:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*
>
> in Machine Learning, by Tom M. Mitchell (1997)

This definition, due to its generality, does not describe well enough the characteristics of the learning problems we will be dealing with. We will now detail this definition to highlight these characteristics and achieve a better understanding of the problem we are facing.

The learning problem faced by an agent i is to maximize the average rewards achieved in given time periods, by learning to map each observed *state*, $\bar{s}_{i,t}$, to an *action*, $a_{i,t}$. Time is measured in discrete units, which we will call *turns*. An *epoch* contains a fixed number of turns. This can be generalizable to other types of learning if we think of the state as an example, the action as a desired response and the reward as a quantity that is inversely proportional to the error.

The mapping of states to actions is done according to a *policy* (π_i). The snapshot of a policy at a given time t, is a function:

$$a_{i,t} = \pi_{i,t}(\bar{s}_{i,t}), \tag{2.9}$$

where $a_{i,t}$ is an element of $\mathcal{A}_{i,t}$, the set of available actions at time t that may depend on the current state.

But the function $\pi_{i,t}$ hides a few details. Each agent i, at a given time t contains a set of *adaptable parameters*, $\mathcal{H}_{i,t}$, called *hypothesis* and an *evaluation function* $\mathcal{F}_i()$ that performs the mapping of states and hypotheses to actions:

$$\pi_{i,t}(\bar{s}_{i,t}) = \mathcal{F}_{i,t}(\mathcal{H}_{i,t}, \bar{s}_{i,t}). \tag{2.10}$$

$\mathcal{F}_{i,t}$ itself contains a set of parameters that are usually fixed throughout the learning process, such as the size and structure of an ANN, or the step used to discretize the state-space when building a Q-table. Some of these parameters may be interpreted as a part of the hypothesis if the learning algorithm is allowed to alter its value. $\mathcal{F}_{i,t}$ can also be a composition of several functions and it is not necessarily deterministic. Many evaluation-functions return a vector ($\bar{a}_{i,t}$) that characterizes the *adequacy* of each possible action as a response to state, $\bar{s}_{i,t}$. In general, an action is more adequate than another if it can generate higher rewards or contribute to its generation in the future. The choice of a particular action, $a_{i,t}$, based on the information contained in $\bar{a}_{i,t}$ can range from simply selecting the action with maximum adequacy, making it a deterministic choice, to considering the choice of all actions with a probability that is proportional to their adequacy, as exemplified for Q-Learning in section 2.1.3.

The state, $\bar{s}_{i,t}$, is a subset of the *global state*, $\mathcal{S}_{l,t}$, on a certain *location*, l. An *environment* contains several locations. The subset of $\mathcal{S}_{l,t}$ that is taken into consideration when calculating the observable state for agent i may be related to its role in the team, its current geographical position, etc. Each location contains a team of agents, each with its own view of the problem. We will refer to the agents in the same location as *partners* and the agents in different locations as *peers*.

$$\mathcal{S}_{l,t+1} = \mathcal{T}(\mathcal{S}_{l,t}, A_{l,t}, \mathcal{U}_{l,t}). \tag{2.11}$$

The global state's transition to $\mathcal{S}_{l,t+1}$ depends on: the previous state, $\mathcal{S}_{l,t}$; the actions of all agents, $A_{l,t} = \cup a_{j,t}, \forall j \in \Phi_l$, where Φ_l is the set of all agents acting in location l; a set of *external events*, $\mathcal{U}_{l,t}$. The external events are, for example the number of data packets generated by the users of a system in a given time interval or the number of cars entering a certain location in a given period of time.

The only restriction we pose to the structure of the state is that its features have the same meaning for all agents, for example: the first feature represents the distance to the closest partner in the upper-left quadrant of the agents' sensing field. This is essential to allow communication of state vectors between agents. This restriction can be relaxed when using the concept of *roles*, defined in section 5.2.6. In this case, only agents with the same role are required to have the same state-structure. The agent itself can pre-process the state to adapt the data to its own learning algorithm, although when communicating with others it must always respect the environment's format.

Based on an evaluation of a subset of state $\mathcal{S}_{l,t+1}$ (mainly, but not only, in the subset $\bar{s}_{i,t+1}$) the environment will calculate the agents' *reward* for each action.

The environment can also evaluate a team's policy calculating the *team reward* achieved by the team controlling location l. This is calculated based on a subset of $\mathcal{S}_{l,t}$. The subset of features used to calculate the team's reward can differ from the union of the subsets used to calculate the agents' rewards, although they should be as highly correlated as possible.

It is fundamental that the environment supports these two types of rewards so that different types of learning algorithms can be used. While some algorithms use rewards that are associated with a certain state (or state-action pair) and rely on their internal structure to measure the long term effects of their actions, others measure only the performance of full policies. The relative effectiveness of both techniques depends on the problem, but since one of our main goals is to assert the possible benefits of different approaches it is imperative that both types of rewards coexist. Panait and Luke (2003) provide an interesting discussion of the consequences of using each of these types of rewards.

The *combined reward*, at the end of epoch n, can be seen as a combination of several components:

$$R_{i,n} = \alpha_1 \sum_{t=t_0}^{t_f} r_{i,t}/(t_f - t_0) + \alpha_2 \sum_{t=t_0}^{t_f} r_{l,t}/(t_f - t_0) + \alpha_3 r_{l,n} + \alpha_4 r_{i,n}, \quad (2.12)$$

where t_0 is the start-time of epoch n and t_f its ending, and the reward components are: the immediate-individual reward, $r_{i,t}$; the immediate-team reward, $r_{l,t}$; the long-term individual reward, $r_{i,n}$; the long-term team reward, $r_{l,n}$. The α_j parameters are the weights of each component. But it is not always possible, or useful, to define all the components of the reward. These components should be highly correlated and in most cases this may allow a simplification of equation 2.12 loosing as little information as possible.

The easiest component to determine is, usually, $r_{l,n}$ because the designer knows what is the long-term objective of the team, but this information alone often poses difficulties to learning, specially for algorithms such as QL. It is important to have some immediate information. If this information was also team-related we would have a system in which agents were strictly bound to maximizing the teams' reward, i.e. with no possibility of acting in their own best interest even if that did not hurt the team's performance. We have adopted a balanced simplification of equation 2.12:

$$R_{i,n} = \alpha \sum_{t=t_0}^{t_f} r_{i,t}/(t_f - t_0) + \beta r_{l,n}, \tag{2.13}$$

where the weights α and β, whose sum is 1.0 if the magnitude of the components is equal, control the relative importance of teamwork and long-term evaluation versus immediate and individualistic behavior.

It may be important for the agent to know the value of each component of the reward. When both terms of equation 2.13 increase consistently due to a change in policy, the agent is definitely going in the right direction. Conversely if both terms decrease the new policy is clearly worse. When the first term increases while the second decreases, the agent is being greedy, either in terms of seeking immediate payoff or by exploring its partners behavior. It is increasing its own immediate reward at the expense of long-term rewards or its teams' performance. In the opposite situation (first term decreases and second term increases) the agent is sacrificing its individual reward for the common good and/or to obtain better long-term performance. By ignoring or changing the weights of the several reward components the systems' designer can decide what type of behaviors are more adequate.

In real systems it is common that a long-term evaluation of a team's performance is easier (and more precise) than the evaluation of each agent's individual response to every state. This is specially true when actions may have long term consequences. It is, for example, difficult to evaluate the impact of switching a traffic light to green for a 20s period. It is easier to evaluate the performance of a control policy in a bounded area over a fixed period of time, knowing that, during that period, the system was under a certain load. In this problem there is a balance to be attained. Even though the global reward is a more exact (less noisy) evaluation, the local reward is easier to learn from, because it is more closely related to the state observed by the agent and its last action(s).

Considering unobserved elements of the global state in the computation of the reward and state transition causes a problem known as *partial observability*, which is a common feature of most real MAS. The main causes for this are:

- The need to keep some information private;
- The impossibility of dealing with all the variables that are relevant for state transition;

- The impossibility to measure some variables directly, or even to know which of them are relevant;

One of the main differences between this formulation and most others is that the reward is calculated externally and not by a critic module inside the agent. This was a natural consequence of calculating the rewards based on more than just the information explicitly available to any given agent. The presence of an external evaluator and team rewards may be considered a partial centralization but there may be good reasons to keep the critic outside the agent's scope, such as: ensuring the privacy of some information used to calculate the reward or limiting the access to certain sensors. In situations where no environment infrastructures are present, the individual reward evaluation would necessarily be done by the agent itself and the team reward could be estimated by communication with immediate neighbors. In this case, the need to synchronize the gathering of rewards could cause the same autonomy problems that are often criticized in semi-centralized solutions.

The problems caused by partial observability are similar to the consequences of external events. These factors, from the agent's point-of-view, cause the reward and state transitions to be stochastic, turning their environment into a non-static world. In other words, there is no guaranty that choosing a certain action in the presence of a certain state will lead to the same reward as it did in a previous case. The agent can only learn the reward distribution achievable for a certain observed state and not the exact reward that it will achieve.

In non-static environments, agents, when starting from a certain state $\bar{s}_{i,t_0}$, will have a certain probability p of getting each possible reward, by applying a given policy π_i, so the best policy π_i^* is the one that maximizes an estimated reward $\hat{R}_{i,n}$ in the future:

$$\hat{R}_{i,n}(\bar{s}_{i,t0}, \pi_i) = \sum_k (\hat{R}_{i,n,k} \cdot p(\hat{R}_{i,n,k} | \bar{s}_{i,t0}, \pi_i)) \tag{2.14}$$

$$\pi_i^* = arg\max_{\pi_i}(\sum_{n \geq n0} (\lambda_n \hat{R}_{i,n}(\bar{s}_{i,t0}, \pi_i))) \tag{2.15}$$

where k takes a different value for all possible rewards and λ_n is the discount applied to future rewards.

When the action choice is deterministic π is a mapping of states to actions. In cases where stochastic action selection is required π is a mapping of states to probability distributions over possible actions.

The direct determination of $p()$ (from equation 2.14), considering all these variable conditions and the number of possibilities, is unthinkable, except for very simple problems. It becomes specially difficult when $\hat{R}_{i,n}$ is not discrete, which turns the sum in equation 2.14 into an integral. Most learning algorithms abstract away many of these components and concentrate on finding the direct mapping of state-action (or state-policy) pairs to rewards.

The learning algorithm changes the parameters ($\mathcal{H}_{i,t}$) (Eq. 2.16), to find the set of parameters ($\mathcal{H}_i^*$) which lead to a behavior that collects the highest possible reward.

$$\mathcal{H}_{i,t+1} = \mathcal{L}_i(\mathcal{B}_i, \mathcal{H}_{i,t}, \mathcal{E}_{i,t}). \tag{2.16}$$

We will refer to the current best estimate of $\mathcal{H}_i^*$ as $\hat{\mathcal{H}}_i^*$. The learning function ($\mathcal{L}_i$) changes the hypothesis, based on the current value of $\mathcal{H}_{i,t}$, a set of learning parameters ($\mathcal{B}_i$), such as learning-rates, and on previous experience ($\mathcal{E}_{i,t}$). Each element of $\mathcal{E}_{i,t}$ includes the state, action and reward experienced at a given time in the past, i.e. $e_{i,t} = \{\bar{s}_{i,t}, a_{i,t}, r_{i,t}\}$. An experience tuple can be extended with further information, such as: its source (because it could have been generated by a different agent than the one that is using it), the epoch reward achieved when this action was used, the state after a the action was completed, etc. We will go deeper into the possibilities of extending this information in chapter 4. The function $\mathcal{L}_i$ needs not to be unique for each agent, to be exact it should be represented as $\mathcal{L}_{K,i}$, where K represents the learning algorithm used. Different learning functions can be used to change the same hypothesis parameters, depending, for example, on the source or type of information they are using. The interaction between different learning functions over the same hypothesis requires some care, but there are many reports of successful integration of learning functions in the literature, some of which are mentioned in chapter 3.

In order for cooperation to be possible it is important that all teams of agents are facing *similar problems*. In this case we assume that the state-space, the possible actions and the reward functions are the same in all locations. What differs from one location to the other are: initial conditions, state transitions and external events. These variations, along with different partner behaviors, are the causes for the different dynamics of each location.

Notice that we have not made any assumptions regarding the convergence of agent's objectives. All we assume is that all agents that can be contacted are willing to share their knowledge. In adversarial environments it would be likely that agents in the same location would withhold information. This is only a problem if there are no other teams in other locations, a scenario that contradicts our initial assumptions. In our environments there is a mix of cooperation and competition. The tasks require that all agents work together and cooperate, but there is also a certain measure of competition for limited resources. An agent must make the best of the available resources to maximize its performance, without causing its exhaustion, which usually leads to high penalties in the team rewards.

We have not considered the possibility of malicious intents of some agents, although the use of *trust* (see chapter 4 for details) can overcome this problem.

The elements of the language, used in agent's communication, may be a simple set of keywords that refer to the type of information that follows, for example, the meanings of: *Individual Average Reward* and *Team Reward* must be understood by

all agents in order to question the environment or other peers about $r_{i,t}$ and $r_{l,t}$. The matters related to the types of data to be exchanged will be discussed in chapter 4. We will not, however, define an ontology or specific protocol since the requirements in these aspects are very simple and most existing communication protocols meet these requirements.

In this section we described the type of learning problems we are interested in, their main difficulties and the assumptions we make regarding environments and agents. We will proceed by analyzing in more detail the main difficulties this problem presents.

2.3 The Problems of Learning in MAS

It is apparent from the description in section 2.2 that learning in these environments presents several problems. From the agents' point-of-view the environment is non-static. This is caused by: partial observability; changes in partners' policies; external events. These factors were examined in the previous section. In addition, there are difficulties in policy coordination and transferring knowledge between different learning algorithms. In this section we will discuss these problems in more detail.

2.3.1 Continuous Policy Changes

Learning requires continuous change in search of better policies. When this is done by all agents in a team, simultaneously, it is defined as *co-learning* (Panait and Luke, 2003). In MAS this causes the environment's dynamics to change from each agent's point-of-view. If we assume that observability is limited and agents are autonomous, it is not possible for an agent to be aware of the state of its partners or of the changes in their policies. For this to be true, the information necessary to make a decision would be far too large and all agents would be trying to solve the whole problem, instead of solving only their part.

The main consequence for learning is that the optimal policy is a moving target. This can be dealt with in two ways, either restricting the possibility of changing the policy in some agents, during certain times, to make the system more stable, or designing systems that can quickly react to environment changes. Both solutions have disadvantages, the first will limit the agent's learning capabilities, its autonomy, and will require a certain degree of synchronization, the second generates a very difficult tradeoff. An agent should be stable enough to avoid changing its policy due to abnormal sequences of events that have a low probability of happening again, but still react promptly to lasting changes in the environment or new opportunities to increase the reward. It is, however, difficult to differentiate, in the short-term, situations where agents will have to deal with new environment's dynamics and those that are temporary effects. The reaction time must be carefully chosen.

2.3.2 Policy Coordination

Policy coordination is highly related to the problem we examined in the last section and also to the type of MAS we are dealing with. MAS learning is often divided into competitive and cooperative. If agents' have conflicting interests and can only increase performance by the loss of performance of others (as in zero-sum games) the task is competitive. When the increase in one agent's performance will lead to an increase in the team's performance the task is cooperative. However, in most situations (the ones we consider here) there is a mix of interests. Without coordination the performance of all agents will drop considerably but, to a certain extent, agents need to maximize their use of the resources.

Tragedy Of the Commons

Let us see how this works in a practical situation. Consider an instance of a load-balancing problem where we have two servers and two load-balancers. One of the servers is relatively faster than the other but not fast enough to serve all requests made by the two load-balancers. If both clients use the best server they will have lower performance than if one was using the slower server. It is obvious that one of the agents must sacrifice its performance or that both will need to balance the number of requests they make to the faster server to reach a good performance. The key factor, in global terms, is to use the full capacity of the faster server, but if the systems' designer also aims at fairness, this capacity must be used by both agents (either equally or proportionally to their respective loads). This problem is known as the "Tragedy of the Commons" (TOC) (Wiering *et al.*, 1999). The TOC occurs when all agents are competing for a limited resource and the optimal level of usage of this resource is lower than the sum of their work. An individual policy that chooses to use only the best resource is only good as long as the number of agents using the same policy is below the resource's capacity. If all agents choose the same "good" policy, they may overuse the resource and be penalized. Still, self-interested agents should strive to acquire as much of the resource as possible.

We may consider the problem of policy coordination similar to learning in a very large state-space, if we see the team as a whole, but with an added difficulty: different parts of the system may be pushing in different directions to optimize the individual component of their rewards.

The solutions to this problem are mostly related to reward design. It is important in these situations either, to share a global reward, or that the reward has global component, but it is difficult to learn from this type of rewards because of its low correlation with each individuals' states and actions. So a balance must be struck between the learnability of local rewards and the ease of coordination that can be achieved with global rewards.

As we can see from the TOC example, it may be important that agents in a team have different policies, i.e. fill different *roles* in the team. The use of roles is another tool to deal with the problems of policy coordination.

2.3.3 Knowledge Transfer

The problem of transferring knowledge from one agent to the other, regardless of the learning algorithm used, has several difficulties. On the basis of these is the different representation used for knowledge. Different types of representations (and evaluation functions) have different capabilities and each type of learning algorithm is more adapted to deal with certain types of data or rewards.

Evaluation Function Bias

In some situations, it may be nearly impossible to transfer knowledge from one agent to the other. For a trivial, but elucidating, example consider the case of the XOR problem. A table-based Q-Learning algorithm will easily learn to map pairs of binary digits to the result of a XOR operation between them. An ANN may, depending on its inner structure, not be able to represent this knowledge at all. This is a very simple example that would easily be overcome by restructuring the ANN, but we have found more complicated versions of this type of behavior. Some policies learned by QL-agents, where neighboring states often require different actions, are very difficult to model by knowledge structures that assume a certain degree of smoothness in the mapping they are required to do. Conversely, in problems where there are vast regions of the state-space where the best action is the same, ANN-based algorithms quickly find good solutions, due to their generalization capabilities, while Q-Learning must explore each and every possible state-action pair before a good mapping is learned. This is both a problem and an opportunity.

Even though most learning algorithms use functions that are, in some way or another, universal approximators (i.e. can approximate any function with an unlimited degree of precision), this guaranty usually comes with a set of restrictions, such as an unknown number of parameters, or the foreseen smoothness of the target function. The more a function is biased to solve a particular type of problem, the less it will be adaptable enough to approximate different target functions. Given a limited (finite) number of parameters there are always limits to the representational capabilities of any evaluation function. Again, having more information can be used to counter this effect. If an agent has enough information and autonomy to change its learning parameters, it can reduce some of these problems, for example, by changing the structure of its ANN during training, or by replaying information from other sources to increase the number of visited states (in the case of QL). Ultimately, an agent could even decide what learning algorithm it must use to deal with each problem based on information from other sources, although we do not explore this path.

On the one hand we see that some agents could, in some cases, help others in their exploration by transferring their generalizations but, on the other hand, we may find situations where knowledge is not transferable at all.

Common Format

Storing knowledge in a common format, readable by all, is an interesting research subject, but it is doubtful that a format exists that is easy to produce and integrate by all learning algorithms. Even if it did exist its generation would introduce another bias in the hypothesis. Nevertheless, if the agents have the appropriate structure, they can send the best hypothesis' parameters along with the information necessary to build an evaluation function similar to that of the advisor. Then, they can use this "copy" of the advisor's function to generate the advice information and integrate it in their own knowledge structure. This would significantly reduce communication and eliminate the need for synchronization. Ultimately, an agent could switch between different evaluation functions using the one best suited for each state, in a similar way to the approach defined in (Powers and Shoham, 2005), and with similar disadvantages (see section 3.2.2 for details).

Specialization

Another problem in transferring knowledge is that the specialization acquired for a certain location may not be adequate for another, i.e. the same policy can have different consequences when used at different locations. The selection of the source of information is the key to overcome this problem. Another technique to minimize this problem is to synchronize information requests between members of the same team. This will limit the problems of learning a policy that does not match the other agents' in this area, but it may also limit the emergence of new global solutions that combine useful aspects of different advised policies.

As we can see there are many difficulties. Some are well known from previous work in single-agent problems, others appear only when agents must work in a team, or when they are endowed with the ability to communicate, others yet are a consequence of the complexity of the environment they are facing. For every expansion of the complexity of a learning system, new difficulties appear, but also more tools are available to deal with these difficulties. Our purpose is to study how the use of some of these tools can improve the ability of agents to cope with the new problems. This will be the subject of chapter 4. But before that, in the next chapter, we will see how other authors dealt with some of these problems.

Chapter 3

Related Work

In this chapter we summarize the research that focuses on learning using both, reward and other sources of information. We also browse through some of the most interesting concepts related to hybrid learning algorithms, team-learning, trust and online-adaptation of learning parameters. This chapter is divided in two main sections, one that covers closely related work in Q-Learning, dating from the early nineties, and another that browses through current related work. Our work touches many points that were previously subject of research on their own. A full review of each of these subjects would be too long to include here. Therefore, in this chapter, we make a brief summary and comment of the main related references, going a little deeper in the analysis of all work where teams of agents learn from sources other than the environment's reward. Given the vast amount of related work the choice of what to focus on was a difficult one. The main *criteria*, apart from its relation to our work, was the visibility (for the most recent works) and its relevance (for the older works), which was estimated by the number of citations seen in related papers.

The research subtopics that support this work range from hybrid learning to the development of trust relationships between agents. Some of these subjects were addressed from different perspectives since the early days of ML research. Approaches such as *Bagging*, (Breiman, 1996), *Boosting*, (Schapire, 1990; Freund and Schapire, 1996) and *Cascade Generalization* (Gama and Brazdil, 2000) tried to capture the advantages of having several different learning systems, usually called *experts* in this context, dealing with the same problem (or with different parts of it). These techniques proved often to be more successful than using a single expert. They were, however, designed for supervised learning, which is not the case of the type of problems we are facing here. Furthermore, the components are regarded as a single learning system, not as a cooperating group of autonomous entities.

Hybrid approaches that mix different learning algorithms were also subject of research: *Counterpropagation Networks* (Hecht-Nielsen, 1987) and *Radial Basis Func-*

tions (Moody and Darken, 1988) mix supervised and unsupervised learning; Hybrids of *Backpropagation* (BP) (Rumelhart *et al.*, 1986) with *Genetic Algorithms* (GA) (Holland, 1975; Koza, 1992) are also reported in many research works. For a review on these the reader can refer to (Salustowicz, 1995; Yao, 1999). The great majority of this work was, however, aimed at solving Supervised Learning problems.

The above mentioned techniques are related to parts of our work, but the first research that clearly focus on using communication between autonomous learning agents in reward-based scenarios appeared in close relation to *Reinforcement Learning* (RL) (Sutton and Barto, 1987) and its most popular variant: *Q-Learning* (QL) (Watkins, 1989) (defined in section 2.1.3). In the early nineties several researchers made important contributions in this direction. This will be the focus of the next section.

3.1 Early Related Work (1990-94)

The work on information exchange between QL-agents (agents that use QL as a basis for their learning skills) started with the contributions of (Whitehead, 1991; Clouse and Utgoff, 1991, 1992; Lin, 1992) and (Tan, 1993).

Whitehead created two cooperative learning architectures labelled *Learning with an External Critic*, (LEC) and *Learning By Watching* (LBW). In the first (LEC) the learner uses the help of an expert agent to bias the search procedure. In normal Q-Learning the values considered when deciding which action to take in a certain state $\bar{s}$ are the estimated qualities for all possible actions a_i, $Q(\bar{s}, a_i)$. In this case the agent considers a sum of these values with a quantity, $B(\bar{s}, a_i)$, called bias. This quantity is updated using information sent by an expert teacher. When a certain action a_i is advised by the teacher, $B(\bar{s}, a_j)$ is added a certain quantity $+K$ if $i = j$, or $-K$ otherwise, where K is a fixed positive value. In the second architecture the agent learns by watching its peers' behavior (which is equivalent to sharing series of state, action, quality triplets). In LBW an agent will use other agents' episodes as if they were its own. This work proves that the complexity of the search mechanisms of LEC and LBW is inferior to that of standard Q-Learning (drops from exponential to linear) for an important class of state-spaces. The state-spaces considered are required to be one-step-invertible (each action has a reverse) and uniformly k-bounded (all states can be reached in, at most, k steps). The main problem with LEC is that the expertise of the teacher agent is critical. A suboptimal teacher may lead to worse learning performance than when no advice is used. In the LBW architecture there are two major drawbacks: it requires that all agents are engaged in exactly the same task and that there is full observability, or unrestricted communication between teacher and student.

The results presented in (Clouse and Utgoff, 1991, 1992) are reviewed and expanded in Clouse's Ph.D. thesis (Clouse, 1997). This important contribution reports the results of a strategy labeled *Ask for Help* (AH), in which QL-agents learn by asking

their peers for suggestions and imitating the suggested actions. Clouse proposes two different approaches for the problem of deciding when to ask for help: in the first approach the advisee asks for help, randomly, for a given percentage of the actions it has to take; in a second approach the agent asks for help only when it is "confused" about what action to take next. Confusion is defined as having similar quality estimates for all possible actions that the agent can perform at a given stage. Several variants on the AH architecture are tested in two types of problems: mazes and the "race track problem". Clouse concludes that the integration of QL and selective imitation achieves better results than each of its constituents separately. However, this solution uses only direct imitation, and the advisee has no choice over different advisors. In the experiments it was proved that AH is not sensitive to random suboptimal advice, up to a certain level, but when getting advice from other agents, specially if advice is coming from a single source, suboptimal advice is not randomly distributed. Consistent errors in a part of the policy are likely to lead the system into a suboptimal response early in the training and hinder exploration. There are two other important differences between AH and the solutions proposed in this work: AH was employed in problems that can be solved by a single agent and not by teams; AH can only be used by QL-agents, the extension of this method to other learning algorithms is not foreseen in Clouse's work.

The work presented by Lin (1992) uses an expert trainer to teach lessons to a QL-agent that is starting its own training. Lin experimented with several architectures, the most interesting for the current subject is QCON-T (*Connectionist Q-Learning with Teaching*). This architecture records and replays both "taught" and "learned lessons", i.e. sequences of actions that led to a success. The "taught lessons" are given by an expert trainer (human or automatic), while "learned lessons" are recorded by the student agent as it explores the problem. The student will replay the actions backwards to reduce the number of iterations necessary to propagate the influence of discounted rewards. Among other things, this work reports that the *"advantages of teaching should become more significant as the learning task gets more difficult"* (Lin, 1992, section 6.4). Results in variants of the maze problem show that replay of taught and learned lessons does improve learning performance in the harder task, although it seems to have no effect on the performance of the agents on the easier task. As in the previous case these methods were tested in single-agent problems and focus exclusively on Q-Learners.

Tan (1993) addressed the problem of exchanging information between QL-agents engaged in teamwork. This work reports the results of sharing several types of information among agents that are working in the pursuit (a.k.a. predator-prey) problem. In these experiments QL-agents shared policies (internal solution parameters), episodes (series of state, action, quality triplets), and sensations (observed states). The conclusions are: *"a) additional sensation from another agent is beneficial if it can be used efficiently, b) sharing learned policies or episodes among agents speeds up learning at the cost of communication, and c) for joint tasks, agents engaging in partnership can*

significantly outperform independent agents, although they may learn slowly in the beginning" (Tan, 1993, in Abstract). The research directions followed in Tan's work are very similar to the ones in which this research is based, although Tan uses only QL-agents with a comparatively simple architecture and it does not focus the problems selecting the information to be used in each case.

The work on exchange of information between QL-agents continued in several fronts. The following section will, briefly, point to some of these approaches.

3.2 Recent Contributions

3.2.1 Transfer of Knowledge from Different Problems

Thrun and Mitchell (1995) approached the problem of exchanging information during learning in a general way. Their work, in the context of *lifelong robot learning*, aims at transferring knowledge about generic problem solving strategies between learners. The rationale is based on the fact that humans do at times generalize the solution to a problem from just a few examples (or even from a single example), often based on previous experience with *other problems*. This meta-generalization is extremely interesting when considering an agent that has to learn a great variety of concepts during its "lifetime". Thrun uses Connectionist Reinforcement Learning (ConnRL), mapping state-action pairs to quality with Explanation-Based Neural Networks (EBNN). Previous knowledge is used to bias the generalization in the training of EBNN and it effectively reduces training times. The authors consider only single-agent learning and the problem of choosing the most appropriate source for this knowledge, when several possibilities are available, was not considered.

Taylor and Stone (2004) follow the same path. In their work they effectively transfer knowledge between two reinforcement learners, reducing the training time when compared to agents that do not use knowledge transfer. One agent (the teacher) has learned to perform a certain task while the other (the student) is learning to perform a similar task. In this case the state and action sets of teacher and student need not be the same. Transfer is accomplished by defining a *transfer function* that was specifically designed for a particular application (labelled Keep-away, a sub-task of Robotic Soccer). The main drawback of this technique is the specificity of the transfer function, which is relatively easy to design for the tasks used in this experiment, but may not be in other situations.

3.2.2 Multiagent Reinforcement Learning

Multiagent Reinforcement Learning (MARL) has been a very active research area in the recent past. Shoham *et al.* (2003) present an interesting discussion and critic

view on some of these techniques. We will look into their opinions after summarizing the main approaches to this problem.

A very broad review of Cooperative Multiagent Learning can be found in (Panait and Luke, 2003). This survey is a good aid in identifying starting points for bibliographical research in the several sub-areas covered.

The first reference to the concept of MARL is in the work of Littman (1994) where QL-agents are used to solve (i.e. attain optimal *equilibrium* in) two-player, zero-sum, games. This approach was labeled *Minimax-Q* and, even though it is still a reference, it was soon challenged by other authors due to its restricted applicability and slow convergence in empiric tests.

In (Claus and Boutilier, 1998) the authors define the concept of *Joint-Action Learners* (JAL). JAL are QL-agents that learn, each on its own, the quality of joint actions. A joint action is composed of an agents' own action plus the actions chosen by all its partners at a given time. The main drawback of this approach is that it presupposes full observability of the actions done by all agents in the team or the construction of a model of other agents' behavior. Also, as pointed out by Bowling and Veloso (2001), the application of this technique in homogeneous MAS results in all agents multiplying efforts to learn the same mapping.

Hu and Wellman (1998) proposed a different update rule for Multiagent Q-learners labeled *Nash Q-Learning* in which the update of the quality values is made under the assumption that all other agents will adopt an *equilibrium* strategy in all future actions.

Bowling and Veloso (2001) proposed an algorithm, *Win or Learn Fast - Policy Hill Climbing* (WoLF-PHC), based on two characteristics that are, in the authors' view, desirable for a learning agent when in the presence of others, namely:

Rationality: Convergence to a policy that is the best response to other agents' policies;

Convergence: All agents must necessarily converge to a stationary policy.

An interesting characteristic of WoLF-PHC is that agents are not required to observe the behavior of their counterparts as in JAL. WoLF-PHC attempts to learn fast when loosing and slowly when winning, accomplishing this by changing the learning-rates appropriately.

Also related to this research, is the work of Kapetanakis and Kudenko (2002) that proposes an optimistic heuristic *Frequency Maximum Q-Value* (FMQ) that uses an extra term in the action selection, along with $Q(\bar{s}, a)$, that benefits actions that produce a maximum reward with high frequency. As in (Bowling and Veloso, 2001) this algorithm does not require full observability of other agents' actions.

The term *Joint Learning* was used by Berenji and Vengerov (2000) when referring to agents that update concurrently the same quality values, in a similar way to the

LBW approach referred above. These authors presented a study of a "fuzzy" variant of multi-agent Q-Learning in which agents cooperate during learning by updating a common quality table. Their work provides theoretical and experimental evidence that, for a wide class of problems, a group of cooperative learners will outperform independent learners, i.e. N communicating agents achieve better results in M steps than one agent will achieve with the same amount of experience ($N \times M$ steps). Even though these are interesting results, the update of a common quality table will prevent local specialization and severely hinder agent's autonomy, not to mention the possibility of causing a communication bottleneck.

Shoham *et al.* (2003) provide us with a good discussion of the directions taken in the recent developments of Multiagent Q-Learning. The main argument is that, in the authors' opinion, too much attention is being dedicated to *Nash equilibrium*. Nash equilibrium consists on having a set of policies, one for each agent in a team, such that: if any agent would decide, individually, to change its own policy, while all other agents maintain theirs, this decision would inevitably reduce its future reward. In this paper, Shohoam *et. al.* advocate that the focus of the work in Multiagent Reinforcement Learning should change to match the four agendas they propose, namely:

Human Agenda: How do humans learn in face of other learners?

DAI Agenda: What procedures should be used by a central designer when defining the (possibly adaptive) behavior of a group of agents?

Equilibrium Agenda: When does a vector of learning strategies form an *equilibrium*?

AI Agenda: What is the best strategy an agent can play for a fixed class of the other agents in the game?

In the sequence of this work Powers and Shoham (2005) define a specific set of *criteria*, which all learning systems should meet in order to be useful, and propose a *Meta Strategy* approach. This approach consists (putting it simply) in switching between different known strategies according to a number of pre-specified conditions. The authors not only prove theoretically that the conditions they have put forth are met by this *Meta Strategy* but also run an extensive set of tests on fully observable, two-player, repeated games, matching their strategy against most of the other known fixed strategies and learning algorithms. The results are surprisingly good in all respects (even more so than the theoretical guarantees) overpowering all other strategies in a vast majority of the cases. Future work points to the extension of this approach to n-player games and eventually to any type of stochastic games. In our opinion this extension will raise a question that was not approached in this work, and is possibly its main flaw: the difficulty in creating a fixed set of rules that is efficient in a larger set of problems.

3.2.3 Advice by Humans and Automated Experts

Many researchers have focused on the use of human advice, or pre-programmed teachers, to help QL-agents.

Gordon and Subramanian (1993) provide high-level, human, advice in the form of rules that are later optimized by a Genetic Algorithm (GA). The technique was tested with experiments in navigation problems. They concluded that, although the combination of procedures did improve performance in some tests, its success was highly dependent on how the advice biased the GA search.

Following a similar path, the work of Maclin and Shavlik (1996) uses connectionist QL with Knowledge Based Artificial Neural Networks (KBANN), (Fu, 1989). In this case human advice is provided by the user, in the form of rules in a structured high-level language. Rules are inserted into the KBANN by creating new nodes that encode the knowledge. Experiments in game domain prove that advice does improve the performance. An important conclusion is that an agent must be able to refine advice so that general instructions can be assimilated and also to overcome the effects of bad advice. Again, the tests were performed in single-agent tasks and group dynamics are not considered.

Matarić (1996) shares sensory data and rewards between teams of robots that use RL in a gathering task. This leads to effective learning of social behaviors such as avoiding collisions by yielding to others when necessary (without causing deadlocks) and following each others' tips on the location of resources to be gathered. This work focused on sharing raw information and using it, mostly, for coordination purposes. Its most interesting characteristic is that data is exchanged locally (between robots that are temporarily close to each other). In subsequent work (Nicolescu and Matarić, 2001) a robot is taught how to perform a task by demonstration, either by a human or by another robot. 9 out of 10 of these experiments resulted in learning a structurally correct set of rules and in an appropriate test behavior. The knowledge structure in this case is based on rules that are triggered by state conditions. Atkeson and Schaal (1997) follow a similar path in trying to teach a robot to swing-up a pendulum, but in this case knowledge is encoded in the parameters of a control function and in a Q-table. In their conclusions, they point out that mimicking is not enough to accomplish this task and that learning is required. Model-based learning achieves a relative good performance fast, but the parallel use of a non model-based approach compensates for modeling errors and slow model learning. This is one of the few cases where two different approaches are used in parallel, even if both approaches are related to the RL family.

Still in this line, the work, reported by Sen and Kar (2002) is one of the few that considers a form of heterogeneity. The *Agent Teaching Agent* (ATA) framework relies on an expert to teach a student agent a certain concept by selecting the training examples to use on each epoch, based on the errors the student made in the previous epochs.

Contrary to most work done in this area the task is supervised, i.e. information on the correct classification of each example is available. The two most interesting issues of this work are: the refinement of the choice, made by the teacher, of which examples to present to the student, and the fact that both agents are using different learning algorithms. In this case the teacher uses instance-based learning (IB2), while the student uses decision-tree learning (C4.5). Initial results are encouraging and further experiments are undergoing. It is important to notice, however, that the teacher is not learning to do the same task as its student, it already posses perfect knowledge on this, and its focus is simply in helping the student. Also, the task is supervised and done by a single agent.

One of the most interesting recent works is (Price and Boutilier, 1999, 2000; Price, 2003) in which novice QL-agents learn by *implicit imitation* of pre-trained expert agents. This work has some very interesting characteristics: the student agent has no knowledge of the actions done by the expert, it can only observe its state transitions; there is no explicit communication between the expert and the student; the goals and actions of both agents can be different. The student agent uses an augmented Bellman equation to calculate the utility of each state that incorporates the information regarding the state transitions made by the expert. This technique was tested in several maze problems with good results. Again, in this work the problems of heterogeneity and teamwork are not focused.

3.2.4 Trust

Trust is an important concept to our work because agents need to acquire information regarding the competence of others. This view of trust is slightly different from most views found in the literature in which trust is associated with security or with exchange of services between agents. There are however some definitions of trust that point-out a competence-related component. When we speak of trust in this work, we are in fact considering only this component that measures the competence of other agents to provide help in a given situation. The same type of technique could be used to disregard the information coming from agents with malicious purposes, but this is not our main goal.

The research on *trust* relationships between agents that is most related to our own, was developed by Sen's research group (Sen, 1996; Biswas *et al.*, 2000; Banerjee *et al.*, 2000). This series of works, and other related papers, focus on several aspects of learning to (dis)trust other agents, and dealing with agents that will not cooperate. The above-mentioned papers are mostly concerned with the effects of different policies in cooperation between agents. These policies range from malicious agents that try to get others to perform their job without, in turn, giving them any assistance, to philanthropic agents that always help their peers. The results indicate that agents that take decisions based on trust are able to cope with malicious agents. The development of trust relationships improves the overall performance by enabling cooperation between

the non-malicious agents and shutting out the others from the society by not cooperating with them. The use of trust in this series of works was never related to learning. The focus is in the design of agent societies rather in how each agent can protect itself against other agents' malicious intentions or lack of expertise.

3.2.5 Adaptation of Learning Parameters

Apart from the, previously cited, work of Bowling and Veloso (2001) not many authors used the available information to change the way the agent learns or automate the evolution of the learning parameters.

One exception is the work of Weibull (1995) in evolutionary game theory, where individuals growth-rate is set to be inversely proportional to the overall success of the population. This allows the agents to change more quickly when performance is low and explore in more detail a given area of the search-space when performance is high. This rationale is similar to the one used in Bowling and Veloso (2001) to justify the adaptation of learning rates.

Dorigo and Colombetti (1994b) use a trainer (that can be a human or another program) to help a QL-agent in learning a certain task by giving it extra reinforcement. They propose three development phases (labeled *baby*, *young* and *adult*) in which the learning agent changes the way it relates to the trainer and its environment. In the baby phase it gets a positive/negative feedback from the trainer for every action. This strategy is highly related to the concept of *shaping* Dorigo and Colombetti (1994a). In the young phase it learns solely from the environments' reinforcement. When the agent reaches the adult phase, the training is assumed to be complete and learning is switched off.

In summary, the work reviewed in the previous sections is mostly devoted to Q-Learning and there are relatively few examples that consider the effects of co-learning within a team. None of these works addresses the effects of exchanging information between different learning algorithms in team scenarios. Most approaches that involve exchange of information use only expert advisors and not other agents that are also in the process of learning. Although several proposals were made for integrating information from other agents and users, these are all specific to a problem or algorithm and no general procedure or architecture was ever proposed. None of these works uses external information, both to tune the learning process and as extra information to improve the current hypothesis. These are the gaps we intend to explore in our work.

3.3 Application-Domains

Although it is not the objective of our work to provide a specific solution to any particular application, reference must be made to previous results on the application-

domains we have selected to test our ideas. That will be the focus of this section. In chapter 6 we will explain the reasons for choosing these problems and the variants we have implemented.

3.3.1 Predator-Prey

Many researchers have used variations of the predator-prey problem (a.k.a. pursuit problem). We will focus our attention in two of these references — (Tan, 1993) and (Haynes *et al.*, 1995b) — due to the close relation between the research presented in these papers and our own. The full description of the scenario used in our experiments can be found in section 6.1. Haynes *et al.* (1995b) present an interesting review of the work on this problem to that date.

As mentioned in Stone and Veloso (1997) this problem has several variable parameters that change its type and difficulty level. According to these authors, the variable parameters are:

- Definition of capture
- Size and shape of the world
- Legal moves
- Simultaneous or sequential movement
- Visible objects
- Sensing range
- Predator communication
- Prey movement

Using these parameters (and a few others that were found useful in this context), we can summarize Tan's scenario by table 3.1. In Tan's experiments the state consists of the coordinates of the prey's position relative to the predator (2 integer inputs), plus the partner's relative position on cooperative trials. When the prey is not visible a "unique sensation" is used. The reward structure is +1 on capture and -0.1 otherwise. Trials are finished when a capture occurs or at the end of a maximum number of turns. The agents that achieved a capture are randomly repositioned at the beginning of a new trial. The conclusions of these experiments, as well as the learning approach, were already mentioned in section 3.1. Tan's results on sharing policies or episodes are based on a 10×10 arena with 2 predators and one prey and a visual range of 4. From the graphics in section 6 we can conclude that the best performance in training was an average of 10 steps per catch, using expert advice, and around 12 when using peer advice.

In Haynes' scenarios (summarized in table 3.3) the state consists only of the prey's position. Capture is achieved by surrounding the prey from all sides. When this is achieved the prey is unable to move and if agents maintain their positions it will remain

Table 3.1: Parameters for Tan's predator-prey experiments.

Parameter	Value
Definition of capture	Individual: same position as prey Cooperative: #predator in vicinity > 1
Size and shape of the world	10x10, bounded
Legal moves	4 (N, S, E, W)
Simultaneous/sequential moves	simultaneous
Visible objects	closest prey and predator
Visible range	2 - 4
Predator communication	yes (of different types)
Prey movement	random
Overlapping objects	yes
Initial placement	random
Number of predators/prey	2/1-4/2

Table 3.2: Average number of steps to individual capture in Tan's Predator-Prey Experiments (for 2 predators and 1 prey). In "mutual scouting" mode the agents have extra information on the prey's position sent by their partner.

Predator type	Visual Range	Avg. steps
Independent	2	24.0
Mutual scouting	2	24.5
Independent	3	16.0
Mutual scouting	3	12.9
Independent	4	11.5
Mutual scouting	4	8.8

locked-in for the remainder of the trial. When two predators try to move to the same cell one is bounced off. The reward for epoch n is:

$$R_{i,n} = \sum_{t \in n} g/d_t \tag{3.1}$$

where g is the grid width and d_t is the distance between predator i and the prey at time t. At the end of the trial predators that are close to the prey receive a bonus of $m \cdot g$ (where m is the number of moves allowed) and when the prey is surrounded (i.e. captured) all predators receive a bonus of $4mg$. This is one of the few approaches where a mixture of local/immediate and global/delayed rewards is used. The results achieved are presented in table 3.4.

Table 3.3: Parameters for Haynes' predator-prey experiments.

Parameter	Value
Definition of capture	surround prey
Size and shape of the world	30x30, unbounded
Legal moves	4 (N, S, E, W)
Simultaneous or sequential movement	simultaneous
Visible objects	prey
Visible range	unlimited
Predator communication	no
Prey movement	away from closest (90%), random (10%)
Overlapping objects	no
Initial placement	prey on center
Number of predators/prey	4/1

Table 3.4: Average number of captures in test in 1000 random scenarios x 200 steps in Haynes' predator-prey experiments, using 4 predators and 1 prey.

Type of Prey	Movement	Average	t-test 90%
Random prey	moving first	100.4	+/- 13.23
Random prey	simultaneous w/ predators	332	+/- 18.71
Escaping prey	moving first	74.1	+/- 9.79
Escaping prey	simultaneous w/ predators	162	+/- 10

3.3.2 Traffic-Control

Traffic-control has been approached by several researchers in different ways. Goldman and Rosenschein (1995) use the example of controlling the traffic-lights at an intersection to prove that learning can replace explicit coordination. In this case each traffic-light controls one of the two roads in orthogonal directions. The purpose is learning to synchronize red/green periods without explicit coordination. This is achieved by having each agent in turn trying to teach the other agent when to set its traffic-light to a given color (opposite to its own).

Thorpe (1997) performed an interesting set of experiments using SARSA (Singh and Sutton, 1996) for traffic-lights control. The objective of these experiments was twofold: establish that learned control is more effective than fixed control and study how the use of different types of state structures can affect learning. The objectives were partially achieved. It was verified that the learning performance is very dependent on the type of information used and that, in some cases, the system is unable to learn.

It was also reported that the differences between the best adaptive strategies and the best results of tests with fixed-duration lights were relatively small. Thorpe performs its experiments in a 4x4 grid of roads where intersections are set approximately 145m apart. Each intersection contains 4 incoming roads, labeled North, South, East and West according to their direction.

Several different state representations are tested, namely:

Count representation: Number of incoming vehicles each of the two approaching directions (north-south and east-west) plus the elapsed time since the last light change. The number of vehicles incoming from the north is summed with those from the south, a similar operation is done for the east and west vehicles;

Fixed-distance representation: Registers the occupation (or not) of each 40m partition of the incoming lanes. North and South information is combined using a boolean "or" for corresponding partitions, as is East and West. The elapsed time since the last light change is also present in this representation;

Variable-distance representation: Similar to previous except in the size of the partitions, which in this case is variable. The size of partitions increases with the distance to the crossing;

Count-duration representation: Equal to the count representation plus the current north-south light color and minimum light duration.

Each agent controls a crossing and its action is to set the color of the north-south lights (and indirectly the east-west, which are automatically set to the opposing color). The learning algorithm (SARSA) has differentiated eligibility traces for each crossing but a common table of state-action values for all traffic-lights.

Car trajectories are pre-set before they enter the scenario by randomly selecting their points of entry and exit. Tests consist in inserting 100, 500 or 1000 cars in the system, with a uniform distribution over time, and allowing 1200 (first two cases) to 2400 time-steps (last case) for the system to clear the scenario.

The performance is measured along four different dimensions:

- Total number of steps taken for all cars to reach their destination.
- Average travel time.
- Total waiting time for all cars.
- Average waiting time.

The most important conclusions, from our point-of-view, were:

- Count-duration representation provides best results in terms of learnability.

- Load-based control (attributing the green light to the road with more traffic) achieved very poor results due to quick oscillation of the traffic lights when the traffic in both directions is similar.

These conclusions were taken into account when deciding on the representation and heuristics used in our own traffic simulation (described in section 6.3).

(Bazzan, 1997) proposes the use of Game Theory and Evolutionary Algorithms for coordination in signal-plan selection. Results indicate that the approach is feasible and learning can reduce or even eliminate the need for communication and negotiation, providing coordinated choice of signal-plans for groups of individually motivated agents. The experiments were run on a traffic-control simulation with computer-generated traffic-flows. One common differentiation between the types of traffic simulators is in the detail level. When the simulation considers the movements of each individual car separately it is labeled a *micro-level simulation*. Contrary to all other works referenced in this section, the simulator used in these experiments does not use micro-level traffic simulation. The simulation is based on flow density which is considered a *meso* or *macro-level simulation*.

The work of Fernandes and Oliveira (1999) consists on the development of a multi-layered architecture in which the first layer controls a single traffic-light, the second layer (local decision-maker) a set of connected traffic-lights, and a third layer, labelled cooperative-layer, was in charge of ensuring that different locations cooperated in the management of traffic. This work proves that agent-based strategies are more successful than fixed-time firing plans in maintaining high traffic flows and that coordinated action plans can emerge from the interaction of independent decision-makers.

Brockfeld *et al.* (2001) evaluates several fixed strategies for synchronized traffic-light control. One of the models for traffic flow used is the same as in our experiments (Nagel and Shreckenberg, 1992), but contrary to our approach they keep the number of cars in the system fixed by re-entering the vehicles in the beginning of the same lane they exit. None of the approaches tested involves learning. Their findings were the following:

- The efficiency of fixed-time cycles depends heavily on the chosen cycle time.
- Results indicate that two different models tested for vehicle movement cause only slight differences in the results, thus this choice does not play an important role in the simulators' design.
- Two dimensional green-waves show better performance than fixed, synchronized, cycles, although still dependent on the choice of the cycle time.
- The use of a random offset along with fixed-time cycles also shows better performance than synchronized traffic-lights.
- Results with "inhomogeneous traffic" suggest that local decision taking at every crossing should be able to outperform any of the global techniques tested in these experiments.

Montana and Czerwinski (1996) use Genetic Algorithms (GA) and Strongly Typed Genetic Programming (STGP) (described in section 2.1.5) to control the timings of traffic-lights. This is similar to (Bazzan, 1997), but over a micro-level simulation. Montana considers that the information comes from two sensors, one that counts cars leaving an intersection and the other that has a twofold purpose: counting inbound cars and verifying if an incoming lane is full. Performance is inversely proportional to the delay, which is defined as the time lost due to stopping at an intersection. Only fixed traffic flows are considered, i.e. the probability of insertion of new cars is fixed in each experiment. In a first experiment GA are used to evolve specimens that consist simply on choosing the cycle time, split and offset for each intersection. The split is the division of time between the N phases of a traffic-light set. Phases must follow a certain sequence (e.g. phase one: green for north-south lanes, phase two: green for right turn in north-south lanes, phase three: green for east-west lanes). In these experiments all intersections have four phases. In subsequent experiments STGP evolves program trees that take into consideration the sensor data in the decision of when to end the current phase. Their findings are summarized by the following statements:

- The learning approach is superior to fixed-cycle control;
- It also generalizes well to situations with similar statistics.
- Performance is expected to drop if variations of the traffic flows are considered.
- Coordination was achieved both with and without communication.

The main problems with this approach, reported by the authors themselves, are: the fact that traffic-flows were fixed for each experiment and that nothing can be said in what regards its scalability.

Wiering *et al.* (2004) compare several different approaches in their "*Green Light District*" simulator. Their original approach is based on a combination of the predictions made by each car. Cars try to predict what their waiting time will be if the traffic-light ahead of them turns green or red and use these estimates to vote on their preferred color for a given traffic-light. They argue against the use of fixed firing sequences based mostly on the fact that choosing which phase is used next is a more flexible procedure. In one of the approaches tested cars also learn to choose their path (this was labelled the co-learning approach). Their conclusions were that RL is more efficient in controlling traffic than fixed controllers (even a simplified version of RL can have good performances in certain situations); and co-learning can, in some situations, avoid overcrowding intersections, although in others it causes no difference in performance.

The use of learning methods in traffic-control, as in other realistic application-domains, is a delicate matter. All systems that must learn from their own mistakes must be projected to learn offline, and its employment should be controlled by another layer of software that must ensure that they operate within reasonable bounds. There

may be, however, much to learn from traffic-control simulation, even though some of the authors mentioned above report only residual improvements of using learning strategies as opposed to fixed ones.

As we mentioned previously we do not aim at solving the problem of traffic-light control in this work, we merely use it as a test case. Nevertheless, we believe that our approach may point in some directions that can be useful when adaptive applications are considered in this, as well as in other, real-life problems.

3.3.3 Load-Balancing

In (Schaerf *et al.*, 1995) the authors present a very broad framework for the study of computing load-balancing problems and an adaptive solution to this problem. This solution is based on a very simple adaptation scheme, labeled: Best Choice Selection Rule. In the first set of experiments agents use only local information. These experiments prove that it is possible to design an adaptable system capable of executing the task, based only on local information. The effect of varying the adaptable parameters of their solution is also studied in these experiments. The second set of experiments is related to heterogeneous systems (here heterogeneity means that different sets of agents may have different policies). Some results show that, under certain circumstances, "parasitic" agents can take advantage of their peers' flexibility and capacity for adaptation. The third set of experiments regards sharing of historical information between agents. It is proven that naive communication may have no effect, and can even deteriorate, the overall systems' performance.

The load-balance scenario used in the experiments – described in section 6.2 – was inspired in (Whiteson and Stone, 2004). This work defines a simulation environment for job routing and scheduling and compares a variation of Q-Routing (Boyan and Littman, 1994) coupled with different types of schedulers, to other routing methods. In the conclusions, the authors claim that: "*The results presented here (...) provide evidence of the value of combining intelligent, adaptive agents at more than one level of the system*".

This is a typical scenario where communication between teams can be used, and it was selected due to our interest in comparing the results of our approach to those presented in the paper. When attempting to replicate the experiment, the discrepancy between the results of our baseline experiments and those presented in (Whiteson and Stone, 2004) led us to make a deeper analysis of their results. From this analysis we concluded that either the results or the experiment description presented in the above-mentioned paper were faulty [1]. Appendix D presents the description of the experiments in the initial version of the paper and our analysis that led to a revision of this work.

[1] It was later established that there was indeed a fault in the experiment's description.

3.3.4 Summary on related applications

We reviewed, in this last section, the main contributions related to the applications we use for our tests. Several possibilities were never addressed in any of this work, namely: None of these applications uses communication between teams of agents with different learning algorithms; In traffic-control scenarios no learning approach ever essayed the use of communication for other purposes than extending the state information; In the load-balance scenarios the only form of cooperation attempted is by sharing all past information. Our work will test new solutions to these problems by using different learning algorithms and selective communication. The following chapter is dedicated to an analysis of the problem and the explanation of our proposed solutions.

Chapter 4

Communication During Learning

In this chapter we analyze the problem, propose solutions and justify the options taken during this work. Some of these concepts were already mentioned in chapter 2 as possible solutions to the problems we described. Some details are left out to maintain a clear line of reasoning. These details will be fully covered in the following chapters.

4.1 Useful Information

The first question is: what useful information can be supplied by other agents facing similar problems? Usually a learning agent selects an action by evaluating its own evaluation function, (as seen in equation 2.10) but if we consider that there may be other agents with whom it is possible to communicate and that they also have some experience in the problem, there are other options. An agent can request information concerning, for example: the best action for the current state; the policy other agents are using; the results of choosing different actions at this state; the magnitude of rewards achievable in the current situation.

If all agents used the same learning algorithm an advisee could simply request the best solution's parameters ($\hat{\mathcal{H}}^{*}_{k,t}$) to an advisor ($k$) that reports a good performance. But we are mainly interested in cases where agents use different learning algorithms, in which case this solution is infeasible. It is also undesirable if we want our system to search for solutions in different ways. Since using the hypothesis' parameters acquired by other agents is not an option, what other information can be useful?

As explained previously condensing the hypothesis' information into a format that can easily be produced and read by all learning algorithms is a difficult task, if at all possible. This solution was not considered in this work, except in the sense that a set of

examples can define a policy. Some efforts were made to aggregate subsets of examples in order to achieve a more condensed set of information that would represent a given policy. These efforts lead to a degradation of the performance and were abandoned during the research (see chapter 8).

Since we are dealing with similar problems, in which the state and actions have the same structure for all agents in all locations, it is possible to exchange the historical information ($\mathcal{E}_{i,t}$ in equation 2.16) on which the current hypothesis is based. But the repeated exchange of all the experiences by all agents in a, possibly large, group can lead to vast amounts of communication and has obvious scalability problems. This is not the only problem: the use of outdated information, its storage, and the different types of information each agent needs, make it clear that we must focus on an *interesting* part of the information produced. We must choose only a limited number of sources and store only relevant parts of the information. The two most obvious *criteria* for storing examples are: efficiency and recency. We have focused on storing and exchanging examples of policies that have recently proved to be efficient.

The information that contains examples experienced by other agents is what we refer to as *advice*. Advice is a tuple consisting of two mandatory elements: a state ($\bar{s}_{i,t}$) and an advised action ($a_{i,t}$) – or a vector defining the adequacy of each possible action to that state ($\bar{a}_{i,t}$). Optionally it may include the source i, the time when was issued t, the reward obtained when the situation occurred ($r_{i,t}$), the state after the action was performed and the reward obtained in the epoch where the action was used ($r_{l,n}$).

$$e_{i,t} = \{\bar{s}_{i,t}, \bar{a}_{i,t}, [i, t, r_{i,t}, \bar{s}_{i,t+1}, r_{l,n}]\} \tag{4.1}$$

Asking a reputedly experienced agent k for the best action for a certain state was the approach used in previous experiments (Nunes and Oliveira, 2003c, 2004, 2005a). We have labeled this type of interaction *specific-advice* as opposed to requesting sub-sets of historical information, *batch-advice*.

Specific advice has the advantage of not wasting resources on information requests for situations the agent may never experience and also does not require the advisor to store its previous experience, but it also has disadvantages:

- The advisee will not have information on the whole policy that led agent k to a good reward, it will learn based on fragments of a policy;
- The advisor will be unable to inform on what was the result of using its policy in that situation at a previous time because it is producing the advice on request, based on its current best hypothesis, and may have never experienced the same state itself;
- It may be difficult to evaluate the confidence in the information sent or the experience an agent has in a certain situation, due to the reasons stated above;
- When this information is intended for immediate use the advisee will have to wait for the advisors' reply before taking an action.

When learning from a sequential set of examples, experienced by the advisor (*batch advice*), the advisee will know exactly what was the result achieved previously with that behavior. It will also have the advantage of training with a full sequence of states that took the advisor from a given state to a goal. Still, the use of batch advice will tend to generate more communication and may fail to respond to the special needs of the advisee in particular situations. The tradeoff between using these two types of information may also depend on the learning algorithm used, or the learning agent's expertise.

As mentioned above, the information may be requested to an agent that "reports a good performance". The knowledge of other agents' performance will also involve communication. Agents may request their potential advisors information regarding their performance statistics in the form of a standing-request that will trigger an answer when a certain situation occurs, e.g. surpassing the best score achieved in previous epochs or completing a given number of validation epochs.

The statistical information on the performance of other agents can also be used for other purposes than just finding the best advisor. An agent is able to know, for example, how far it is from the best solution achieved this far to its problem and use this knowledge to tune its learning parameters.

In summary, in this section we argue that:

- An agent may find it useful to know:
  - Examples of others' policies (batch);
  - Examples of others' response for a given situation (specific);
  - The results achieved by choosing a certain action for a certain state;
  - The rewards achieved during certain periods;
  - The average performance of the best policies.
- The problem of having a large body of available information must be dealt with by filtering this information according to certain *criteria*. These *criteria* can moderate the amount of information exchanged keeping the solutions' scalability.

4.2 When and Where to Collect Information?

Humans ask for information in one of two situations: when they learn that someone else seems able to solve the problem they are facing more efficiently, or when they cannot tell which is the best course of action for a certain situation. This procedure can be replicated by automated agents if they are aware of other agent's performance and if they can measure how certain they are of the action to take next. In the first case all an agent will need is to know its peers' average performance. For the second case an

agent must be able to determine the confidence in each action or the level of experience for a given state. The latter is easier for some agents than for others.

As noted by Dorigo and Colombetti (1994b) (see section 3.2), it may be useful to divide the learning process in stages. Although we agree with the principle we disagree with the proposed division. We believe that the initial phase (where most learning takes place) can be subdivided in several different stages. The *learning stage* of an agent can be used to change the way it learns and also to decide whether or not to ask for advice, and, if so, in which form. Humans, when learning about a subject, go through different stages:

1. First they need to acquire some basic knowledge on the subject;
2. After this phase they require a stream of detailed, sequential, information about a solution. This is usually given by a single teacher;
3. Then, as their skills in solving the problem increase, the information needs decrease and become more flexible. Useful information can be collected from several sources and may be specific for a certain sub-problem.
4. The ability to deal with conflicting information, to experiment new solutions, and to pick up the best of several different ideas, arises when the solver is well acquainted with the variants of the problem and with several different solutions.

In an attempt to mimic this behavior we have defined four learning stages:

Exploration: Initially the agent will not use external information and will explore using only the environment's feedback, to acquire some experience on the problem and assess its own capabilities in learning a solution;

Novice: After the exploration stage if an agent has a low performance, much worse than the best performance in other teams, it will require detailed, and coherent, information about the solution to its problem.

Intermediate: When the agent has a reasonable performance, although below the level of the best agents, it may benefit from being exposed to different solutions. This may lead it to combine parts of different solutions or use policies that are more suited to its own problem, even though they may be reported as suboptimal by other agents.

Expert: When the agent has learned a policy that is on a par with the best know so far, it can devote more resources to exploration and to advise its less successful peers.

These will be our *learning stages* and they will determine how an agent decides what type of advice to request and how to parameterize its learning algorithms.

In what regards the "where" (to collect information), the obvious answer is: the agent with best performance, – or even simpler, ask everyone and then decide – but the similarity of the problem being solved, the capability of an agent to duplicate the teacher's solution and the advisors under use by its partners, can also be taken into consideration. This may allow better decisions and limit communication.

An agent should seek information that is:

Efficient: Proved to have a good performance in a related situation;

Learnable: Can be integrated in the hypothesis using the tools available to this particular agent;

Adequate for its problem: Matches the policies of the agents' partners.

If an agent's partners behave quite differently from its advisor's partners then it is likely that the advice is useless (or even harmful). An agent must choose its advisors not only based on their performance but also on how useful is their advice.

Advisees must learn which of their peers provide more valuable information from their own point-of-view. If a whole team asks for information to another team, in synchrony, the problems related to inadequate advice may be mitigated. However, synchronous advice from one team to another requires either dynamic or pre-established agreements. The easiest form of establishing an agreement for synchronized information requests is to trigger this advice based on the comparison of teams' scores. One of the solutions to prevent all agents from taking advice from the same peer is to define *roles*. This can be done simply by attributing a different role to each team member and establishing that advice can only be requested to agents that share the same roles.

After trying several types of strategies to distribute advisors, avoiding that two members of a team ask advice to the same member of an expert team, we have settled in a simple form of *role* assignment. This simple procedure consists of assigning sequential numbers to the agents in a team by order of creation and restricting advice requests to agents with the same team number (i.e. role). This method is simple and proved to be as effective as more elaborate solutions, referred in (Nunes and Oliveira, 2003b) and in chapter 8. These solutions involved exchanging information between members of the same team to minimize the probability of sharing advisors. Another approach, that also aims at solving the problem of different team dynamics, is the use of unsupervised learning to create a set of representatives of the most common states for each agent. Matching these representatives for a pair of agents would provide a measure of how closely related were the problems they were attempting to solve. This way we could group related agents allowing them to choose an advisor that experienced states that were more closely related to its own experience. Neither of these methods obtained better performance than the simple sequential numbering of agents. Nevertheless, these methods may be useful when the problems each team is solving differ in more than just the partners' behavior and initial conditions.

One possible use of external information, that we have not mentioned previously, is in the parameterization of the learning process. Most learning parameters control how much an agent learns from each example (or set of examples) and how much of the available time is it willing to use for exploration of new policies. In this case, since validation occurs in parallel (see chapter 5 for details) one could be tempted to say that exploration should always be the most important factor, but there are drawbacks to this. An excessive emphasis on exploration may prevent the test of coherent policies, since an agents' options may be constantly changing for the same state. Exploitation, in a team scenario, is important, not only to take advantage of what was learned but also to provide coherence in the choices and enable the fine-tuning of partners' policies when a low rate of exploration is used.

When an agent knows that there are better solutions (by getting performance information from its peers) it is natural that it will maximize the learning-rates used in the acquisition of knowledge from peers, while controlling its exploratory behavior (to test the advised policies appropriately). After this stage there are two possible situations: either the agent reaches a similar performance level to that of its advisor, or not. In the first case, it is likely that the agent became an expert, having a score that is on a par with the best. The acquired policy will be in continuous validation and the agent can increase the exploration levels and stop information exchange (or reduce it significantly). If the advisee did not reach the advisors' performance levels, this may have occurred for two types reasons:

Internal: The advisee is unable to mimic the advisors' solution due to limitations of its learning algorithm or evaluation function, or

External: The advisors' policy is not an adequate solution for its own problem, either due to differences in the locations' current dynamics, or because its partners are not performing matching policies.

In both cases the agent should either try a new advisor, or increase the exploration levels and learn more from its own experience than from advice. In case of internal difficulties an agent could also increase the representational capabilities of its own evaluation function, although we have not explored this path in this work.

These two types of failure in integrating the advisors' policy are related to two concepts that can also be estimated.

- The first (internal causes) is related to what we have labeled as *learnability*. This quantity represents the capacity to learn an advisors' policy. It can be measured as the inverse of the classification error that the advisee shows when comparing its own options with those of its advisor.
- The second case (external causes) is related to the adequacy of the policy being taught to the advisees situation. We have labeled this quantity *trust* and measure

it as the quotient between the reward achieved with the advised actions and the reward that the advisor claims to achieve with the same actions. Trust is used here as a measure of the competence of a certain advisor.

In the previous paragraphs we have defined (informally) a set of rules to control several learning parameters. These rules will be formally stated below in section 5.2.3.

As hinted during the introduction of learning stages, another important matter is the the stability on the choice of an advisor. On the one hand it may be important to have just one advisor for a long period so that its policy can be acquired without interference. On the other hand, getting advice from several advisors enables the emergence of policies that merge different aspects of advisors' policies. Using too many advisors, can, in some cases, prevent the agent from learning any of the policies if advisors give conflicting advice. So, in certain situations, it may be important to select a *standing-advisor* that will be the sole source of information for a specific period. This has two advantages: it is easier to attribute the responsibility for the long-term result of using advice and it is less likely to have inconsistent advice (it is not impossible because the advisors' policy may be stochastic). However, the use of *multiple-advisors* may contribute to the emergence of policies that result from merging several different solutions. It is important to establish in which situations is the use of only one advisor preferable to using information from several available sources.

Even though some care is taken to restrict communication at all levels it should be noted that the cost of communication is often non-linear (except when it is associated with energy consumption and/or wireless access). The cost, in most cases, is virtually null up to the band-width limit and very high when over that limit (if it can be surpassed at all). This is similar to establishing a maximum band-width, which was our approach. We have defined maximum numbers of examples to be exchanged per epoch, thus simulating a limit band-width.

In summary, we can say that:

- An agent should ask for information either when it knows there is a better solution than the one it is currently using, or when doubts arise in choosing an action.
- Advisors should fulfill three requirements:
  - Have a good performance;
  - Their policy must be learnable by the advisee;
  - The policy must be adequate for the advisees' problem.
- Trying to acquire a teams' policy as a whole may be adequate in certain stages and this can be enforced by simple agreements that do not require agents to (explicitly) synchronize or exchange any further information.

- The failure to acquire a solution may be caused by internal or external factors. This can be estimated by measuring how much was learned from an advisor and how well does its policy adapt to the advisees' current problem.
- The tradeoffs resulting from using single or multiple advisors must be considered.

4.3 Integrating Information

The way in which information is integrated is highly dependent on the learning algorithm(s) used by an agent. The exact process of integration used for each type of agent will be detailed in section 5.2.3. Here we will review the basic guidelines on how to use information.

The main concept is to have two (or more) different learning functions working on the same hypothesis. When reinforcement information arrives the hypothesis will be changed by a learning function that uses the reward information. When advice information arrives, another function can be used taking advantage of the extra information in the advice. If an advisor is well chosen its advice can be interpreted as supervision information.

Two difficulties are immediately apparent:

1. The interplay between two different learning methods may fail to respect the particular convergence conditions of each method;
2. When the limitations are at the level of the evaluation function neither of the methods will be able to solve the problem.

The first problem must be dealt with in the design process. In ML literature there are quite a few successful works in the integration of different learning functions working on the same hypothesis (see chapter 3 for examples). In some of these, the problems related to the convergence conditions are circumvented by alternating the use of both learning algorithms, others simply prove empirically that the process does converge and is effective on a number of practical cases. Considering that one of the characteristics of the systems we are studying is the continuous change, it is appropriate that all agents maintain a certain degree of adaptability which, in most cases, precludes convergence to a static hypothesis. Given these facts, we have taken a pragmatic approach to this first problem by choosing combinations of learning algorithms and evaluation functions that proved to be applicable in the situations we have studied.

The problem of learnability can be attacked in two ways: by changing the parameters of the evaluation function, or by simply ignoring parts of the state where the function does not perform well and keeping the information (advice or its source) necessary to overcome these situations.

There are several ways to integrate advice information in an hypothesis, depending on the structure of the evaluation function. Advice can be used in one, or more, of the following ways:

Imitation: The advisee performs the advised action and learns from the consequences of this action. In this case, advice must be specific and used online. It also requires that the advisee can learn from actions that it did not generate itself. The advisee must send its current state to the advisor and wait for its reply.

Virtual Experience: Advice is interpreted as if the agent performed the action itself but the reward information is the one received by the advisor when the advice was recorded. In this case, advice must be collected in batch and the advisee must be able to integrate information about others' experiences. This type of advice must contain information on the reward obtained by the advisor, and possibly also the state of the environment after the action was performed.

Supervision: In this case, advice is interpreted as examples of the correct mapping of states to actions. Only these two pieces of information must be transmitted. The advisee must be able to perform supervised learning. This type of advice is usually best suited for batch-offline learning, because a large set of examples is usually required to learn a reasonable mapping of the advisors' policy.

The information requested to an advisor can be used *online* (i.e. trigger a change in the current hypothesis immediately upon arrival), or *offline* (i.e. be stored on arrival and processed in batch later). Both methods have advantages and disadvantages, many of which are related to the learning algorithm used and will be discussed in section 5.2. In general, only specific-advice can be used online. The only exception is when the advisee finds an answer to its current problem in previously stored advice that may have been received in batch.

The integration of information about others' performance is done at a different level. Reward information is mostly used for control purposes. In our experiments we have used this information to: determine an agents' learning stage, enable and disable advice, select an advisor, decide which type of advice to use and change learning parameters.

In summary:

- The integration of information requires that an agent is able to learn from reinforcement as well as from supervision information.
- Advice can be imitated, used as virtual-experience or as an example given by a teacher.
- Information can be used immediately (i.e. online) or stored and used in offline.
- Information on peers' performance can be used to control the learning process.

As we can see from the previous sections, there are several ways in which information from other peers can be useful, but there are also several tradeoffs that require our study. Up to this point we identified the following options/tradeoffs: specific vs. batch, online vs. offline and using all advisors vs. using a standing advisor. We also have several possibilities regarding the way to use advice: imitation, virtual-experience or supervised training.

Sections 5.2.1 and B.3 show how the several concepts are related. The first presents an explanation of the response to all the events, the second presents an explanation in pseudo-code of the reactions to the main events.

In this chapter we have analyzed the problem, explained the reasons that lead to our main options and we have defined, informally, the main components of our approach. We have also mentioned several directions that may prove interesting, but that we have decided not to pursue to avoid overextending the focus of our research. In the following chapters we will detail the design of the simulator, the specific implementation of these ideas and the experiments we have devised to study the effects of these techniques.

Chapter 5

Simulator Architecture and Agent's Structure

We have discussed, in general terms, the possibilities and pitfalls of using information provided by other agents. In this chapter we look at the structure of the agents and environment. We will define the requirements they must fulfill to have the capabilities discussed in the previous chapter.

In the following discussion we will assume that the agents are part of a software environment. This implies the presence of environment Infrastructures (IS), such as Directory Facilitators (DF), sensors, and reward evaluation modules. The application of these methods in environments without these infrastructures would eliminate some requests but would also require a more elaborate way to collect information on partners, peers and rewards.

The objective of this work is to evaluate the impact of communication during learning. To do this we must simulate MAS problems that require agents with learning skills. Initially the choice was either to implement a MAS using parallel processes, which could run in different physical sites, or to simulate such a system. We chose the simulation of this system in a single-threaded program for the following reasons:

- Learning is a processor and memory-intensive activity. The availability of the required number of machines to deploy a true MAS in realistic conditions, during long periods of time required to perform the experiments (several consecutive weeks at times), was out of the question. It may be important at this point to say that, despite the number of publications and the interest this work generated in some members of the research community, the funding for a project related to this work was only successful in its last few months.
- The simulator is required to run 100 to 1000 times faster than real-time (this is specially significant in the traffic-control problem). The adaptations required

to achieve this speed would necessarily lead to minimizing delays, including communication and synchronization. This would require either the distribution of the processes over a relatively large number of computers, or the concentration of processes of the main parts of the system reducing communication delays. These restrictions are mostly related to the simulation rather than the advice-exchange or learning procedures.

- The pre-developed agent platforms, existent when development started, were relatively slow and, in some cases, had limitations to agents' behavior, linkage with external code, or top-level language used. Most MAS simulators were closely related to Java, a language that has gone through an enormous evolution during these few years. Its efficiency, even though still not par with C/C++ related languages, has improved greatly since this project started.
- The design and implementation, from scratch, of an agent platform was out of the scope of our research and would require too many resources that were necessary for other activities.
- Since synchronization of processes between learning agents is necessary only in a very limited number of situations the problems that can derive from parallel implementation are minimal.
- A distributed MAS that uses this architecture would require distribution only at the validation level. Most learning could still be done offline in processes where efficiency would be the most important issue and these would not require any type of synchronization.

We have implemented a multi-level simulation with three different levels. The first level (*learning*) quickly sorts out the bad policies from the promising candidates. At a second level (*test*) the promising candidates are thoroughly tested in terms of long-term adequacy and coordination with other agent's policies. At a third level (*validation*) the best policies are under continuous validation and will only be replaced when a test-policy proves to have better performance. The existence of the validation-level can also allow us to determine when there are sudden changes in the environment's dynamics. In a real situation the first too levels would run offline in a simulated environment while the validation would be online acting on the problem. This type of system requires continuous adaptation of the simulation to mimic as close as possible the real environment. In our simulation agents switch at fixed intervals between different modes (learning, test and validation) to simulate the parallel execution of these different levels. The predator-prey and load-balance problems have cycles of 200 epochs (100+80+20 for learning, test and validation, respectively). The Traffic problem has a cycle of 540 epochs (180+180+180). The difference in the partition and sizes of the cycles is related to finding a comparable interval for each problem. In the traffic-control problem 180 epochs correspond to a week and, given the daily oscillations, is the minimum comparable interval.

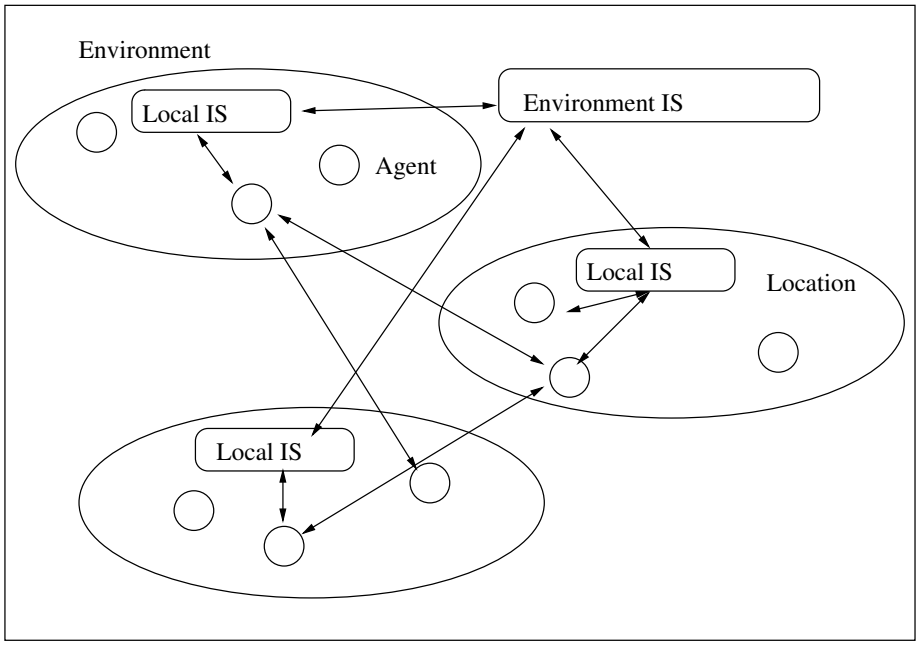

Figure 5.1: Environment's structure. The Infrastructure (IS) modules may contain Directory Facilitators (DF), as well as other services necessary to gather and distribute information on a particular environment/location. Lines represent some of the possible communications between elements in the environment.

5.1 Environment's Structure

The main structural units of simulation software are: Environment, Location and Agent, organized as depicted in figure 5.1. The Environment IS provides contact between different Locations. Each location provides agents with the following services: addresses of other agents solving similar problems at different locations, state observations and rewards. State transitions are calculated at the location level.

There are several time scales in this environment. In the location-time an increment corresponds to one state transition. Several local state transitions may occur between two agent's observations. All transitions occurred in the environment between two observations will be perceived by the agent as a single transition.

Different environments may require a different number of actions per turn. In the predator-prey problem we have one action per turn. In the traffic problem agents are only required to act once every 12 turns (12s). The remaining turns are used to calculate the evolution of the cars in the environment on a second by second basis. In the load-balance problem each agent may route several jobs in each turn, so, in this case, several actions take place in each turn.

At the agent level a time-step (or turn) is the interval between two state observa-

tions. In real environments there are two types of critical timings, the response-time (i.e. the time elapsed between the observation of a certain state and the production of an action) and the adaptation-time (the number of observations an agent needs to adapt to new environment conditions). The first is easily met, because the evaluation process is computationally simple for the algorithms under study. The remaining tasks required for learning could be performed in the idle-time between two observations, or in an offline server. The second critical timing is the most important for our study, therefore measurements are always taken in terms of the number of validation epochs (which represent a fixed number of turns). The points where synchronization between learning agents is necessary in a real system would be:

- During the initialization procedures.
- Waiting for online advice to produce an action.

The first is not critical since it happens only once at startup. The second is necessary only in a few of the studied scenarios and could be surpassed by local implementation of the advisors' evaluation function, as described in section 2.3.3. We stress this point because autonomy is a very important characteristic and it may be severely limited by the need to synchronize.

It can be argued that using communication and receiving external data increases the amount of information and thus it cannot be directly comparable to solutions where there is no communication. This view is correct if we assume that an increase in the amount of data leads to a proportional increase in the critical response times (as was common in ML approaches). In MAS, agents often have idle time while waiting for the state transitions to occur in the environment. Thus, using this idle time to acquire and process external information will incur in no penalty, except for the communication cost. For this reason we have concerned ourselves mostly with running-times and amount of data exchanged and less with comparing performances for similar amounts of information. Nevertheless we have made an analysis of these costs in the discussion (chapter 7).

5.2 Agent's Structure

From an agent's point-of-view each turn is composed of four phases:

1. **Sensing:** In this phase all agents receive their own view of the environment's state. All sensing actions are considered to be simultaneous. This emulates a standing request, from each agent to a location IS, for state information. This information is broadcasted to all members of the team simultaneously.

2. **Acting:** Agents perform the action selected for the current state.

3. **State Transition:** In this phase the location calculates the evolution of its state, based on the previous state, the external events and the actions of all agents (as explained in section 2.2, equation 2.11). During this time agents may be idle or occupied with internal tasks.

4. **Reward distribution:** After the state transition each agent receives its own reward ($r_{i,t}$ in equation 2.13). At fixed intervals the location IS sends a team-reward ($r_{l,n}$ in equation 2.13) to all agents in its location.

An agent contains the following components (depicted in figure 5.2 and detailed in sections 5.2.2 through 5.2.8):

Evaluation: This module contains the evaluation function ($\mathcal{F}_i$ in equation 2.10) that maps states to actions according to the current policy or a previously saved one, depending on the active set of parameters. Evaluation functions may be required to keep certain temporary values used in some learning algorithms, e.g. the derivative of the error used in BP.

Learning: The learning algorithm contains the $\mathcal{L}_{K,i}$ functions that use rewards and advice information to change the current hypothesis (equation 2.16, detailed in section 5.2.3 for each type of algorithm used). Heuristic agents do not have this module.

Control: The Control module takes decisions regarding: when to request advice, how to filter and relay it to its Learning Algorithm. The calculations of *trust*, *learning stages* and other concepts used in the decision of when and to whom should information requests be made are also calculated here.

Parameter Stores: These stores contain several labeled hypotheses (i.e. parameter sets) that can be saved and retrieved by the Learning Algorithm and Control modules. The labels are: learning, test-candidate and validation.

Example Stores: This set of stores contains labeled sets of examples or advice divided in epoch-size blocks. The stores have the following labels: current epoch, last epoch, online advice$_N$, advisor-name validation$_N$, where N stands for the epoch number. In each case where a store keeps information regarding more than one epoch, old information is discarded after a certain time, or the store is reset when a certain event happens.

Communication: This module receives, interprets, and relays information. It also keeps track of the peers addresses received from the Location-DF. In the case of our simulation it merely keeps pointers to peers that enable an agent to call another's methods.

Reward Statistics: Receives reward information and keeps several sets of reward statistics.

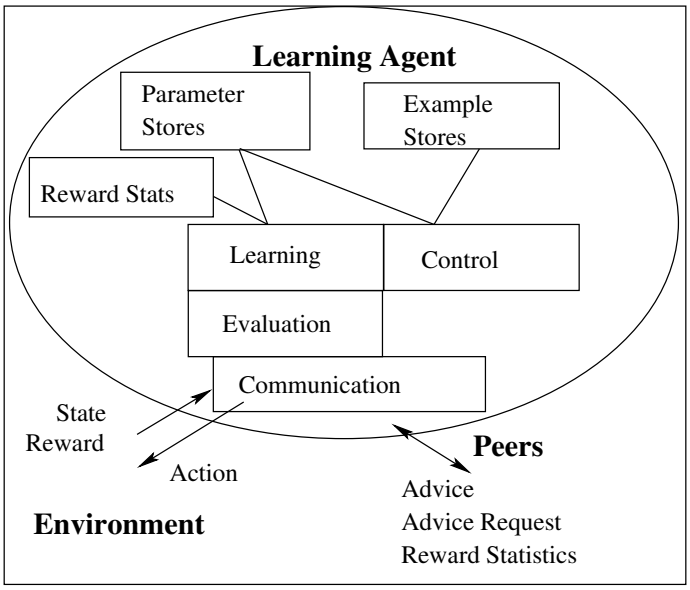

Figure 5.2: Agent components.

In the current implementation the parameter stores, reward statistics and learning algorithms are associated to the Algorithm class hierarchy, and all remaining modules are associated to the Agent class hierarchy. Further details on the simulator design can be found in Appendix B.

In the following sections we will describe each of the components of our learning agents in more detail and give more precise definitions for some of the concepts debated in sections 2.2, 2.3 and chapter 4.

5.2.1 Response to External Events

The agents' parameters, on creation, will define what type of learning algorithms it will use as well as other characteristics that will be mentioned in the following sections. A summary of these parameters can be found in appendix B. On creation an agent will issue a standing request for the addresses and identifications of other agents possibly parameterized by its own role. On arrival of a peer's address and identification it will issue a standing request for validation-reward statistics, whenever these are updated.

Agents' functions, except for initialization procedures, are triggered by the arrival of new information. The following list describes the actions triggered by each of the events. Section B.3 contains the pseudo-code for the main events described here.

Observed state: Evaluate the state, with the suitable parameters for the current mode (i.e. learning, test or validation), and execute the selected action. If using

specific advice, request advice. When imitating this process is suspended while waiting for the advice reply.

Advice arrival: When new advice arrives the agent will process it, or store it for offline usage. When an evaluation is suspended, waiting for advice information, it will be resumed.

Standing-request for advice: Send all advice (i.e. examples of validation epochs stored) that is currently available and matches the request; save the standing request for future reference.

Advice request: Process the request using the current validation hypothesis and send the requester the advised action.

Peer's reward update: Update peer's information. The arrival of this information triggers a re-evaluation of all parameters that are related to peer's rewards. When the number of peers increases the information requests are restricted to the most successful advisors.

End of validation epoch: Triggers the reply to all standing requests for rewards and validation examples. When the validation policy changes it will signal this event to all advisees with standing requests.

Peer's reward request: These are usually standing requests, only issued during initialization. They will be stored and re-evaluated at the end of validation epochs.

Advisor's change of policy: Triggers the process that clears the advice stores that belong to that advisor.

Reward: Triggers the learning procedure, parameterized by the state and action that correspond to this reward. Some learning algorithms may delay the learning procedure until the arrival of the next observed state, whenever this information is necessary for learning.

Epoch reward: Calculate combined reward for epoch n. Calculate each advisor's responsibility in this result (depending on the percentage of imitated actions or on the classification accuracy of offline training) and update the trust coefficient of each advisor. When offline advice is used, replay stored advice (EA and GP-agents only use offline advice when a new-generation is created).

5.2.2 Reward Statistics

As discussed in section 2.2 an agent receives an individual reward for each action issued and a team-reward for the performance in a given epoch. These are combined, as

shown in equation 2.13, to produce a combined reward. The *Reward Statistics* module calculates several indicators based on each type of reward. The following description focuses (in terms of notation) on the combined reward ($R_{i,n}$), but can easily be extrapolated for other cases. In the following equations, i stands for the agent's identification and n is the current epoch.

- Infinite discounted reward ($R^{id}_{i,n}$) provides a weighted reward in which the importance of older rewards decays in time:

$$R^{id}_{i,0} = R_{i,0}, \tag{5.1}$$

$$R^{id}_{i,n+1} = \alpha R^{id}_{i,n} + (1-\alpha) R_{i,n+1}, \tag{5.2}$$

  with $\alpha \in [0,1]$;
- Absolute best reward ($R^{max}_{i,n}$) is the highest reward achieved this far:

$$R^{max}_{i,n} = \max_{m<n}(R_{i,m}); \tag{5.3}$$

- Decayed best reward ($R^{best}_{i,n}$) is an adaptive estimate of the maximum reward achieved in the recent past:

$$R^{best}_{i,0} = 0, \tag{5.4}$$

$$R^{best}_{i,n+1} = \max(\beta R^{best}_{i,n}, R_{i,n}), \tag{5.5}$$

  with $\beta \in [0,1]$;
- Average reward ($R^{avg}_i(n_0, d)$) over a period d staring at epoch n_0:

$$R^{avg}_i(n_0, d) = 1/d \sum_{n=n_0}^{n_0+d-1} R_{i,n}; \tag{5.6}$$

- Average reward evolution ($R^{ev}_i(k, n_0, d)$) estimates the variation of the average reward in $k-1$ periods ($k >= 2$) of $d \neq 0$ epochs starting at epoch n_0:

$$\Delta R^{avg}_i(n_0, d) = R^{avg}_i(n_0 + d, d) - R^{avg}_i(n_0, d) \tag{5.7}$$

$$R^{ev}_i(k, n_0, d) = 1/(k-1) \sum_{n=0}^{k-2} \Delta R^{avg}_i(n_0 + dn, d) \tag{5.8}$$

  Replacing d with suitable values we can have estimates of the short, mid and long term evolution of the average reward.

All these statistics, are permanently updated by all agents and may be sent to any other agent upon request. Several of these were used in previous experiments to decide when to get advice or select an advisor although, currently, only R^{avg}_i and R^{ev}_i are used.

5.2.3 Learning Algorithms and Evaluation Functions

The choice of the learning algorithms was based on two *criteria*: their applicability to the type of problems we wanted to study and the differences in the way they explore search-space. One of the main questions of our study is related to the effects of cooperation between agents using different learning algorithms, therefore we made all efforts to use learning algorithms with very different characteristics and exploration strategies, to enforce heterogeneity. This allowed also a study of the common points in the architecture that were useful to a broad spectrum of, reward-based, learning techniques. In the first experiments (Nunes and Oliveira, 2002b) we used Q-Learning (QL), Evolutionary Algorithms (EA) and Simulated Annealing (SA). SA was abandoned due to the poor results registered in the initial trials. QL and EA showed interesting (and different) behaviors in the initial experiments and were kept for that reason. In the final stretch of this research it was found useful to have another algorithm with a radically different knowledge structure from the other two. Strongly Typed Genetic Programming (STGP) was chosen for this reason and also because it can produce knowledge in a human-friendly format and incorporate domain knowledge, which is not the case of EA or QL.

In the following subsections we will explain how each of these algorithms is used. This explanation is based on the definitions presented in section 2.2 and will focus on the differences between the standard implementations and our own.

QL-Agents

Upon observation of a new state ($\bar{s}_t$) a QL-Agent will evaluate it by calculating $Q_j(\bar{s}_t, a_{t,j})$ for available actions $a_{t,j}$ (as explained in section 2.1.3) producing a vector of Q-values ($\bar{Q}_j(\bar{s}_t, \bar{a}_{t,j})$). This evaluation can be a simple extraction of a row of values from a matrix, indexed by state-action pairs (in table-based QL), or the evaluation of an ANN that is trained to perform this mapping (ConnQL). The majority of the QL-agents used in our experiments use ANN to map state-action pairs to Q-values due to the high dimensionality of the state-spaces. It was found useful to divide the mappings of the different actions. We have one, independent, ANN for each action, which can be interpreted as an ANN array or a single ANN with independent hidden layers for each output, as depicted in figure 5.3. In this case the units in the last layer are linear, i.e. do not use a sigmoid function to restrict the output to $[0, 1]$.

The vector of Q-values $\bar{Q}(\bar{s}_t, \bar{a}_t)$ contains the estimated quality of performing each possible action. The choice of which action will be executed may be done in one of two ways: using the action with the highest Q-value, or, choosing a random action with probability $p(a_j|\bar{s})$ calculated by Boltzmann selection as shown in equation 2.7. The former is used in test or validation modes, while the latter is used learning mode. When the choice of an action is stochastic the Q-values are normalized to unit sum and $p(a_j|s)$ is the normalized Q-value.

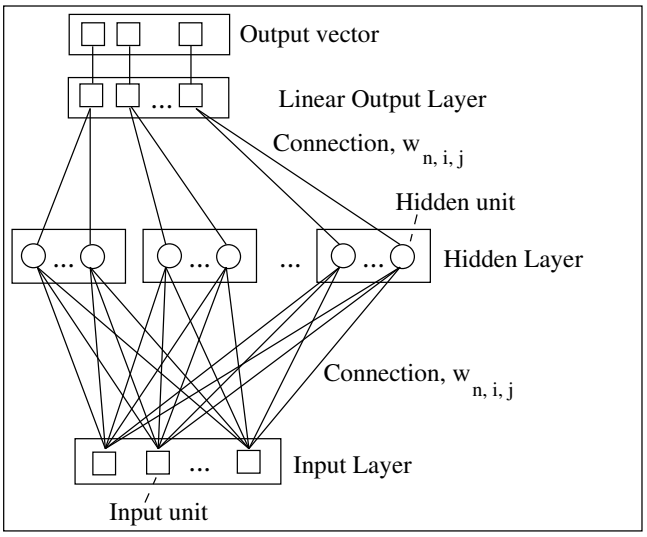

Figure 5.3: ANN for Connectionist Q-Learning with separate hidden layers for each output and a linear output layer. Circles represent ANN nodes, rectangles enclose associated nodes and lines represent weighted links between nodes.

We can define the output of $\pi_{i,t}(\bar{s}_{i,t})$ (equation 2.10) as any of the following: the vector of Q-values, the probability of using each action, or a binary vector with 1 representing the final choice and 0 for the remaining actions. The first two provide more information than the latter, so, we keep the probability of choice as the action vector to be stored and delay the choice of the actual action until its execution. It is also this action-vector that is used as advice. The hypothesis' parameters are the Q-values themselves in table-based Q-Learning or the weights of the ANN in ConnQL.

When a reward, r_t, arrives, the ANN that corresponds to the action that originated this reward, is trained using standard backpropagation, with $\bar{s}_t$ as input value and $r_t + \beta Q_{max}(\bar{s}_{t+1})$ as the desired response. QL-Agents store these examples and perform a replay (presenting the examples in reverse order) at the end of each training epoch, as advised in (Lin, 1992) to improve the speed of knowledge acquisition by the ANN. When using table-based QL instead of ConnQL the update is done as shown in equation 2.5. The epoch-reward is registered but not used directly in the learning process. This may be a problem when the combined reward has a strong component of the global reward and it conflicts with the individual reward (i.e. increase in the individual reward may cause a degradation of global rewards). The main parameters of QL-agents and the effects of changing them are described in table 5.1.

Advice can be used in one of three ways:

Imitation: In this case advice replaces the evaluation process and the agent uses the advised action and observes the result, as in the work of Clouse (1997), described in section 3.1.

Table 5.1: Main parameters for a QL-Agent. The second column contains the type of change these parameters undergo during training in standard implementations. The third column is a summary of the expected effects of changing each parameter.

Parameter	Standard	Effects of change
ANN structure	None	Insertion of nodes in the hidden layer(s): Possible increase in the ability to map other functions. Higher probability of over-training/specialization; Removal of nodes: Decrease in the detailed mapping capabilities. Possible better generalization.
BP learning-rate α (Eq. 2.1) for ConnQL only	Decayed	Increase: Faster convergence, but also higher instability and possibly jumps to regions outside the local minimum's attraction basin; Decrease: Slower convergence, better exploration of the current local minimum area.
QL discount β (Eq. 2.5, 2.8)	None	Increase: Emphasize long-term results; Decrease: Emphasize immediate results.
Biasing weight γ (Eq. 5.9) Used in advice only	None	Increase (over 1.0): Give more importance to advice than own experience; Decrease (below 1.0): Give more importance to own experience than advice.
QL learning-rate α (Eq. 2.5) Used in table-based QL only	Decayed	Increase: Focus on last experiments; Decrease: More emphasis on previously acquired knowledge.
Temperature T (Eq. 2.7)	Decayed	Increase: Increases exploration and makes the response more stochastic in learning mode; Decrease: Increases exploitation of previously acquired knowledge and determinism in the choice of actions.

Virtual-Experience: In this case, the agent will learn from another agent's examples as if they were its own, as in LBW (Whitehead, 1991), described in section 3.1. This requires the existence of information on the reward obtained and the state observed after acting, thus it cannot be used with specific advice.

Biasing: In the third case the Q-value that corresponds to the advised action is updated as defined in Eq. 5.9. In ConnQL the ANN is trained using this value

as desired response, while in table-based QL this will correspond to the above-mentioned assignment. This process is based on the same rationale as the one used by Whitehead (1991), but differs in two important details. Whitehead uses a fixed bonus (positive for the chosen action and negative for all other available actions) and separates the biasing information from the Q-values. A fixed bonus will quickly freeze the action choices by setting a large gap between advised actions and others. Setting the value of the advised action equal or slightly higher than the best Q-value for that state will enable the agent to explore the advised option and, if the advice proves to be inadequate, quickly return to the previous policy.

$$Q(\bar{s}_t, a_{t,k}) = \begin{cases} \gamma \cdot \max\limits_j(Q_j(\bar{s}_t, a_{t,j})) & : \quad Q(\bar{s}_t, a_{t,j}) < \max\limits_j(Q_j(\bar{s}_t, a_{t,j})) \\ Q(\bar{s}_t, a_{t,k}) & : \quad \text{otherwise} \end{cases} \tag{5.9}$$

The variable j is the index that differentiates the several possible actions, $\gamma \in]0, 2[$, $a_{t,k}$ is the advised action, and $a_{t,j}$ runs through all actions available at the current state.

It is important to notice that imitation, is the only advice process that requires synchronization, i.e. advice must arrive before a decision is taken. This requirement can be relaxed as explained above (in section 2.3.3) or by using stored advice.

It is also important to notice that these processes differ in how new knowledge is acquired. When imitating the agent learns from its own reward. If the advice is not appropriate there is no change in the policy, i.e. the action with maximum Q-value will be the same after the update. The advisee can also attribute direct responsibility to the advisor for the outcome of the action. When biasing or virtual-experience is used the Q-values are changed, possibly assuming a value that is not adequate for the advised agent. This value may have impact on future choices. It also is not guaranteed that the advised action is selected. If this is the case, the advisor cannot be held directly responsible for the result.

Virtual-Experience and Biasing can be used offline, which will mean that information is stored and the actual update procedure will be done only in the beginning of a learning phase. In QL-agents, when using offline advice, the examples collected from all the available advisors are used only at this moment, changing the current learning hypothesis.

EA-Agents

The evaluation function for EA-Agents is similar to the one described for ConnQL in the previous section, i.e. an ANN array where the input layer has the same dimension as the state. In this case, however, the output layer is composed of sigmoid (softmax) functions ($tanh()$ in this case). The output vector will contain values in [-1,1] and

the desired values for training must be within that range. Another difference is in the interpretation of the result. The output of this network is not an estimate of the reward as in QL. All we can say is that the evolution mechanism should tend to select mappings in which the highest activation corresponds to the best action. The action that corresponds to the highest output is selected and ties are broken randomly. When trained from advice it will generate maps of the "relative adequacy" of each action for a given state, from the advisors' point-of-view.

The algorithm used differs in the following points from the one presented in (Glickman and Sycara, 1999), summarized in section 2.1.4:

- Each specimen is evaluated during an epoch and its fitness is the combined reward for that epoch. When all specimens are evaluated the selected specimens will be the ones with highest fitness. Contrary to (Glickman and Sycara, 1999) we do not use tournament selection;
- The selection strategy is elitist (i.e. keeps a number of the best specimens in the next generation without any changes);
- The crossover strategy consists on choosing two parents from the selected pool and copying subsets of the hypothesis from each of the parents to form a new individual. Notice that the restriction that forces all ANN to have the same structure makes it possible to copy subsets from one parent for the same position (i.e. node) in the offspring, thus, leading to the same type of feature-detection behavior in the output of that layer. The number of specimens generated by crossover is minimal when compared to the ones generated by mutation, but contrary to (Glickman and Sycara, 1999) we have found this procedure useful.

At the end of each generation a certain number of specimens is selected and the remaining members of the population are destroyed. The best specimens in the selected pool are copied to the next generation unchanged. The remainder of the population is filled with specimens resulting from mutation and crossover of the selected specimens. The number of specimens that are kept unchanged, generated by mutation and crossover is determined by the algorithms' parameters, detailed in appendix B. The main parameters of EA-agents and the effects of changing them are described in table 5.2.

Mutation is done by copying a selected specimen and changing its hypothesis' parameters ($h_{i,t} \in \mathcal{H}_t$), which in this case are the ANN's weights, according to:

$$h_{i,t+1} = \begin{cases} h_{i,t}, & : \quad r_1 < mutation_{prob} \\ h_{i,t} + r_2, & : \quad \text{otherwise} \end{cases} \tag{5.10}$$

where $r_1 \in [0, 1]$ is a randomly generated number with uniform probability and r_2 is randomly generated by a normal distribution with a variance equal to the $mutation_{rate}$. $mutation_{prob}$ is the probability of mutating a given hypothesis' parameter.

Table 5.2: Main parameters for an EA-Agent. The second column contains the type of change these parameters undergo during training in normal circumstances. The third column is a summary of the expected effects of changing each parameter.

Parameter	Type of change	Effects of change
Population size	Fixed	Increase: More exploration; slower training; Decrease: Less exploration; faster fine-tuning of the best specimens found.
BP learning rate α (Eq. 2.1)	Decayed	Increase: Faster convergence, but also higher instability and possible jumps to regions outside the local minimum's attraction basin; Decrease: Slower convergence, better exploration of the current local minimum area.
Mutation-rate (Eq. 5.10)	Decayed	Increase: Higher exploration, higher probability of generating bad specimens; Decrease: Focus on a few good areas of the search space. Danger of loss of diversity.
Mutation probability (Eq. 5.10)	Decayed	Similar to mutation-rate. The relative impact of these two parameters depends on the structure of the hypothesis (specially the number of hypothesis' parameters).
Kept specimens (%)	Fixed	Increase: More elitist. Less exploration, more exploitation. Danger of loosing diversity Decrease: Higher exploration
Crossover (%)	Fixed	The effects of crossover are highly dependent on the hypothesis structure, but, generally, the number of crossover specimens is positively correlated with exploration
Mutated (%)	Fixed	Similar to the above.
ANN structure	Fixed	Similar in all respects to the situation described in table 5.1. Difficult to implement because all specimens must change simultaneously to keep the crossover mechanism. Some proposed solutions to mix EA and BP may overcome this problem elegantly, see (Salustowicz, 1995; Yao, 1999) for details

Crossover is done by generating a new specimen from copies of parameter sub-sets from the parents. The subsets are defined *a priori*. In this case we use subset-masks that

group the incoming weights of each node. Since units (specially those in the hidden layer) work as feature detectors this seemed an appropriate partitioning. The same view is defended in (Salustowicz, 1995).

Learning from advice can be done online or offline using standard or ALR-BP. Online BP was abandoned early during this research due to its poor results when compared to offline ALR-BP. EA-agents cannot use imitation or virtual-experience because they cannot learn from actions that were not generated by a specimens' policy. The desired value ($\bar{d}$) for a certain advised action $a_{k,t}$ is defined as $d_j = C$ when $j = k$ and $d_j = -C$ otherwise. In most examples of BP algorithms C is set to 1, but this target-value can never reached by the sigmoid ($tanh()$) which may cause weights to grow indefinitely in an attempt to reach the target. To solve this problem some authors suggest re-scaling the sigmoid to $[-1.2, 1.2]$ others the use values of C lower than one. We have chosen the latter, using $C = 0.97$. Using a value of C lower than one has yet another advantage. By keeping the output sigmoids in the verge of saturation it is more likely that small changes in the weights can have large effects on the outputs, this allows low mutation-rates to continue being effective in changing the network's mapping.

The integration of advice is done at the beginning of each new generation. Part of the new specimens are trained using the examples saved in the advisor's stores. When standing advisors are used only one specimen is trained. When using multiple advisors, one specimen is trained for each of the advisors that exhibits a better validation performance. Each specimen is trained with the advise store of only one advisor. The training of the new specimen continues until the number of epochs without a decrease in the error is higher than $K \times E_n$, where $K = 150$ is a constant and E_n is the classification error in epoch n.

GP-Agents

The third type of agents use Strongly Typed Genetic Programming (STGP). The major differences between our approach and the ones currently in use (explained in section 2.1.5) are:

- Each action has a different program tree that performs a fuzzy classification of the pattern;
- The leaf nodes are scalar values instead of labels;

A Program Tree (PT) has one root-node for each possible action and evaluates to a real number having as argument a state vector. Nodes have two possible base-types according to their output (boolean or real). A summary of the types of nodes available can be found in table 5.3.

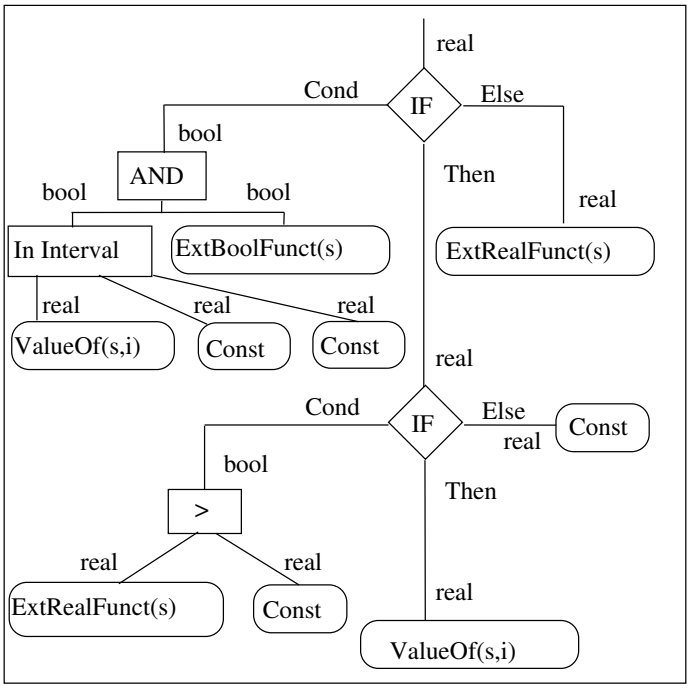

Figure 5.4: An example of a Program Tree. Diamonds represent conditional instructions. Rectangles represent functions. Round-edged rectangles represent leaf-nodes. The result types appear above each node.

External Function nodes point to environment-specific functions, of the appropriate type that each environment must supply. All External Functions (boolean and real-valued) take the state as input and calculate a certain value based on that state, e.g. an environment-specific distance between two states, an evaluation of the equality between two elements of the state-vector, etc. This allows the introduction of domain-specific knowledge by making available to the program a set of functions that are meaningful in the context and would be too complex to generate by random assembly of nodes. This will obviously introduce a bias, but it is a common procedure in STGP and has proved to achieve interesting results (Haynes *et al.*, 1995b). The functions used for each of the problems can be found in sections 6.1 through 6.3.

The initial trees are randomly generated within the restrictions posed by the types and with a given (pre-set) probability of generation for each node type. The root-node is always an If-Then-Else to avoid trivial trees. The maximum depth is also pre-set. When maximum depth is reached the generation of a terminal node (one without subtrees) is forced.

Evaluation is done recursively always carrying the state as a parameter. The set of program trees generates a vector of real numbers as in the case of EA and QL-agents. The selected action is the one with highest activation and ties are broken randomly.

Table 5.3: Description of the main types of Nodes in a Program Tree. The subtree types are: (b)oolean and (r)eal.

Name	Type	Subtrees	Data	Evaluates to
If-Then-Else	real	1b+2r	-	The value of the true branch if the condition is true and the false branch otherwise
Index Value	real	0	index	state[index]
Real Const Value	real	0	real	A constant real value in [-1,1]
Arithmetic	real	2r		The result of an arithmetic operation between two real numbers (sum, multiplication or subtraction). The result is bounded to the [-1,1] interval
Ext. Real Func.	real	0	function name	A real number, f(s), where f() is a given external real-valued function
Logical	bool	2b	-	The result of a logical operation between two booleans (AND, OR, XOR). The negation operator is also available having a single branch.
Ext. Bool Func.	bool	0	function name	True/False, f(s), where f() is a given external boolean function
In Interval (and Outside Interval)	bool	3r	-	True if the evaluation of the first sub-tree is in the interval defined by the evaluation of the two remaining subtrees, false otherwise
Greater Than (and Smaller Than)	bool	2r	-	True if the evaluation of the first sub-tree is greater than the second, false otherwise

All processes related to specimen evaluation and selection as well as the generation of a new population are similar to the ones described for EA except for mutation and crossover. The main parameters of GP-agents and the effects of changing them are described in table 5.4.

Table 5.4: Main parameters for a GP-Agent. The second column contains the type of change these parameters undergo during training in normal circumstances. The third column is a summary of the expected effects of changing this parameter. Most parameters were omitted because they are similar to those referred in table 5.2 and changes have similar effects.

Parameter	Type of change	Effects of change
Max. tree depth	Fixed or undefined	Increase: Possible increase in the ability to map other functions; slower training. Higher probability of over-training/specialization Decrease: Less detailed mapping capabilities. Possible better generalization; faster training

Trees are mutated by recursively applying a mutation operator to each node. If a given random number (uniform in [0,1]) is smaller than the current mutation probability the current node will mutate. The type of mutation will depend on the node it is applied to. Nodes with subtrees can delete a sub-tree and generate a new one. Nodes containing data can change it by disturbing the constant value stored in a Real Const Value node with random noise, or by changing the External Function to be applied, as long as the return type of the function is maintained.

Crossover is done simply by selecting action-subtrees randomly from one of the parents, to form a new set of trees for the offspring. The probability of choosing a subtree from a certain parent is proportional to the parents' fitness.

Most of the algorithm parameters are similar to those used EA, except for BP-related parameters that, in this case, are replaced by the PT-specific parameter: maximum tree depth.

When a random tree is generated (or a mutation occurs) a restriction vector may be passed as a parameter during the recursive generation procedure. In some nodes restrictions will be added to this vector before each branch generation. For example, when generating an *In Interval* node we will need three branches, the first is the variable and the remaining two define an interval. It is pointless to have three *Constant Real* nodes as branches since the evaluation is always the same regardless of the state value. We restrict at least one of these values (the first) to being a non-constant. We also restrict the evaluation of all arithmetic functions to the [-1,1] interval by cutting-off the results that exceed these limits.

Advice is integrated in the knowledge structure using ID3 based on a batch of examples collected from other agents. Each set of examples is processed once for each action. Before processing, the examples are divided into positive (states for which that

action was advised) and negative ones. Each action is trained separately generating its own program tree. An action k is considered to be advised for a certain state if k is the index of the maximum value in the action-adequacy vector returned by the advisor.

As for EA-agents the training of new specimens with advice occurs only when a new generation is created and each specimen integrates the advice from a single advisor.

There is a predefined set of boolean functions that partitions any set of states in two subsets. This set contains the following functions:

- Greater Than
- Smaller Than
- In Interval
- Outside Interval
- External Bool Functions

The first four functions are tried for different combinations of *Index Value* and *Constant Value* nodes (e.g. *Greater Than(Index Value: 1, Constant Value 0.2)* corresponds to the test $s_{i,t,1} > 0.2$, for $s_{i,t,1} \in \bar{s}_{i,t}$). The maximum gain partition is chosen at each step. When a subset contains only positive or only negative nodes a Constant Real Value node is generated with 1.0 or -1.0, respectively. When maximum depth is reached a node is generated with a constant value that represents the percentage of positive examples in the current subset. This variant of ID3 enables the representation of fuzzy solutions instead of limiting the leaf nodes to a definite class and, although it was autonomously developed by the author, it may be considered a simplified version of the algorithms referred in (Ichihashi *et al.*, 1996; Chiang and Hsu, 2002).

5.2.4 Information Storage

Two main types of learning-related information are stored by the agents: hypothesis and examples.

Hypothesis

The stored hypothesis are labelled: learning, test-candidate and validation. EA and GP agents also store the hypothesis' parameters for each specimen of their populations.

The learning hypothesis contains the parameters currently in use. It is usually a copy of the parameters in memory that is updated at fixed intervals. The test hypothesis contains the parameters that are under evaluation when the system is in test-mode. These parameters may become the new validation hypothesis if they prove to achieve a better performance. The validation hypothesis is the one that is estimated to provide the best performance. These are the parameters used to provide advice to other peers.

All hypotheses require a certain evaluation-time before any changes are allowed. This time, the mid-term reward evolution interval, is defined in Eq. 5.7. In implementations where the learning, test and validation environments run in parallel these stores would be split through different levels.

Examples

The example stores are labeled: last epoch, current epoch, validation and advice. In addition to these there is one store for each advisor. The last and current epoch stores keep the examples experienced in the last two epochs. The validation stores are used to cache the examples that will be sent to peers that have a standing-request for validation-examples. The number of validation-stores that an agent is allowed to keep at each time is a parameter defined during initialization. Validation stores are destroyed whenever the validation hypothesis is updated. Advice stores keep the advice that was provided in the past for possible future replay. The advice is split by several advisor-related stores.

Example stores are cleared/replaced when the following events take place:

- End of epoch: last epoch's stored examples are replaced by the ones in the current epoch store. The latter is cleared.
- Change of standing advisor: Advisor's store is cleared.
- Change in an advisors' best policy: Advisor store is cleared on its signal.
- Learning stage of advisee changes (Eq. 5.12 through 5.14) or advice is started or stopped (Eq. 5.18). All advisors' stores are cleared.

5.2.5 Learning Stages

The evolution of an agents' learning process is mimicked by the definition of four *Learning Stages*: Exploration, Novice, Intermediate and Expert. Each learning-stage has specific characteristics. Some learning parameters and advice integration modes are more suitable for each case. By comparing table 5.5 with tables 5.1, 5.2 and 5.4 we can start to draw a picture of the changes in parameters that are appropriate in each situation.

Agents evaluate the conditions to transit from one stage to the other at the end of a validation epoch. An agent can transit only once from Exploration to any of the other stages and will not return to Exploration stage. Transitions between all other stages can happen any number of times.

An important condition for all transitions is the stabilization of the learning process (C_s), defined as:

$$C_s = R_i^{ev}(K, n-K, d) < \eta_d, \tag{5.11}$$

Table 5.5: Learning Stages, their characteristics and impact on learning parameters.

Stage	Characteristics
Exploration	No advisor Decaying exploration: QL: decaying temperature, EA/GP: decaying mutation-rate and mutation probability Decaying learning-rate (QL/EA)
Novice	Single advisor, batch advice Low exploration-rate: QL: low temperature or fast decay, EA/GP: low mutation-rate and probability High advice integration-rate: QL: high biasing-rate QL/EA: high learning-rate for advice Medium/Low learning-rate for own experience: QL/EA: medium/low learning-rate
Intermediate	Multiple advisors Medium exploration-rate: QL: medium temperature, EA/GP: medium mutation-rate and probability Medium/Low advice integration rate: QL/EA: medium/low learning-rate for advice Medium/High learning-rates for own experience: QL/EA: medium/high learning-rate
Expert	No advisors High exploration-rate: QL: high temperature (decaying) EA/GP: high mutation-rate and probability (decaying) Medium/High learning-rate for own experience: QL/EA: medium/high learning-rate

where K is the number of time intervals considered, $d \in \{10, 20, 100\}$ is number of epochs and $\eta_d \in \{0.01, 0.01, 0.05\}$ is the minimum evolution required in the short, mid and long-term, respectively. The traffic-control problem uses slightly different values to synchronize the defined intervals with full days or weeks. When C_s is true for all values of d learning is said to have stabilized. Whenever information is not available for a long-term evaluation it is replaced by mid-term information. If mid-term information is not available C_s is false.

Several sets of conditions were tested to define the transitions between neighboring

states. The ones that were used in the experiments reported here are:

- Novice:

$$C_s \wedge R_i^{avg}(n_0, d) < R_k^{avg}(n_0, d) - 2\gamma \cdot std\_dev(R_k(n_0, d)) \tag{5.12}$$

- Intermediate:

$$C_s \wedge R_k^{avg}(n_0, d) - 2\gamma \cdot std\_dev(R_k(n_0, d)) \geq R_i^{avg}(n_0, d) \wedge \\ R_i^{avg}(n_0, d) < R_k^{avg}(n_0, d) - \gamma std\_dev(R_k(n_0, d)) \tag{5.13}$$

- Expert:

$$C_s \wedge R_i^{avg}(n_0, d) \leq R_k^{avg}(n_0, d) - \gamma std\_dev(R_k(n_0, d)) \tag{5.14}$$

where, R_i^{ev}, is the performance evolution of agent i (equation 5.7), k is the index of the agent with best performance in epoch n, or the standing advisor when one exists, and γ is a weight, set to 1.0 whenever a state transition occurs, and multiplied by 0.9 whenever the agents' performance is stabilized. γ forces agents with suboptimal strategies to ask for advice if their situation is stable for a long time.

5.2.6 Roles

In (Stone and Veloso, 1997, section 5.2) MAS are defined as homogeneous or heterogeneous depending on having "different goals, possible actions and/or domain knowledge". Although this definition is useful it does not (explicitly) address the possibility of developing heterogeneous behaviors as a competitive advantage in problems where the use of the same policy by all agents leads to low performances. In certain MAS problems the agents may be homogeneous in the sense that they have the same goals, possible actions and domain knowledge, but they must learn to behave differently from their peers to have a good performance. In this case agent's must assume certain *roles*. A role, in this case, is simply a differentiated policy that complements those of other partners.

There are two main types of role definitions: static and dynamic. Static roles are those that are pre-defined by the user for MAS where the agents have different functions to perform, as defined in (Stone and Veloso, 1997). It is common in these cases that the actions available to each role are different and even the state-structure may be different depending on the agent's role. In these cases learning information can only be exchanged between agents with similar roles. Since, in terms of information exchange, these MAS are simply a restriction of those where roles change dynamically we have not considered them in our study. Problems where strictly homogeneous agents are useful, are also a subset of the case considered.

When roles are dynamic all agents have the same capabilities but the best solution to the problem requires that they use different policies. In this case it is not important which agent assumes a certain role, as long as all critical roles are simultaneously taken. There can also be different combinations of roles that perform equally well.

In respect to information exchange, dynamic roles pose a problem of synchronism. If a certain team has a good performance, learning to emulate their behavior may depend on all agents of an advisee team choosing a *different member* of the successful team as advisor. There are different ways of achieving this result, the simplest of which is to attribute, randomly, a virtual role to each agent and to prevent agents (at least in some stages) from requesting information to advisors with different roles. When members of a team ask for advice synchronously to the corresponding members of a successful team we can be assured that all are learning different policies an that these policies match. We do not limit the choice of advisors to a certain team, although it is likely that advisees will chose members of the most successful team as advisors.

This type of strategy does not eliminate the problem of learning mismatching strategies, but it will effectively minimize it. Still, in some cases, specially in QL-Agents due to the algorithm's characteristics, this may partially destroy the acquired knowledge and a good process to find the role each agent is playing and choosing the "right" advisor could accelerate the acquisition of a new policy, provided that all critical roles of the advisor team were learned.

One such process was attempted in previous experiments (described in appendix A.1).

5.2.7 Learnability and Trust

Learnability, in this context, is the capacity to integrate the knowledge transmitted by the advisor and reproduce its behavior. The Learnability of the advice currently stored from a given advisor k, from the point-of-view of an agent i, is calculated as:

$$l_{ik,n} = m_{match}/m_{total}, \tag{5.15}$$

where m_{match} is the number of actions in which the choice made by agent i is equal to the action advised by agent k, and m_{total} is the total number of advice analyzed. When advice is taken offline, using ID3 or BP, the final classification performance (percentage of correctly classified examples) is used as the policies' learnability.

As mentioned above there are three components that must be considered when choosing an advisor. The advisors' efficiency in its own problem (its rewards), the learnability of the advisors' policies (how well can an advisee learn what is being taught) and the adequacy of the policy for the advisor's problem. Trust, in this context, is a relation between an advisee and an advisor. It measures the usefulness of the information from a certain source. To measure this we chose the quotient between the

result achieved by the advisee when using a certain policy and the result announced by the advisor for that same policy. To put it more precisely, the trust of agent i in agent j is initialized to 1 and updated by:

$$trust_{i,j,t+1} = \alpha trust_{i,j,t} + (1-\alpha)(R_{j,t_0}/R_{i,t}), \tag{5.16}$$

where $\alpha = min(0.7, l_{ik,t})$ is the trust update-rate, R_{j,t_0} is the score announced by the advisor and $R_{i,t}$ is the actual score received when applying the policy. Using learnability as an adaptation-rate for trust will prevent drastic changes in the trust when the policy of the advisor is not being correctly mimicked.

This quotient, when multiplied by an advisor's announced reward, should provide an agent with an estimate of the reward that will be obtained by using the advised policy based on previous experience with that source. If locations have very different dynamics it would be more appropriate that the adaptation of the trust values includes a softmax function.

When using online imitation all examples can be used in the update of trust. When the advisee is not imitating the advisor only those examples in which their policies agree (m_{match} in equation 5.15) can be used in the update. In cases where learnability is low, or still unknown, and it is imperative that an agent estimates the trustworthiness of a certain advisor it can simply imitate the advisor for a certain period of time to acquire this estimate and resume the normal learning process afterwards. The drawback is, again, the synchronism requirement that imitation carries. In our experiments we did not use imitation to aquire an estimate of the advisors adequacy.

Trust is reset to 1.0 when the agent changes its learning stage to allow a fresh start that may be usefull if the policies of the advisors' partners changed recently.

5.2.8 Advice

An advice ($\bar{e}_{i,t}$) a is tuple with the following components:

$$\bar{e}_{i,t} = \{\bar{s}_{i,t}, \bar{a}_{i,t}, [i, t, r_{i,t}, \bar{s}_{i,t+1}, r_{l,n}]\}, \tag{5.17}$$

as defined in section 4.1. When learning stages are not used advice is collected when:

$$C_s \wedge R_i^{avg}(n_0, d) < R_k^{avg}(n_0, d) - \gamma \cdot std\_dev(R_k(n_0, d)) \tag{5.18}$$

where i is the advisee and k the peer with best average score. This would correspond to all learning stages, except Expert and Exploration (Eq. 5.11 through 5.14).

As mentioned in section 4.1, two types of advice can be requested, *specific* or *batch*. Specific advice is requested for a given state experienced by the advisee and contains only the mandatory components of vector $\bar{e}$. Batch-advice is a standing-request made to a given advisor. This advisor will send examples of the application of its best policy, the one used in validation, whenever a validation epoch ends.

Before requesting specific advice an agent will try to find in its stores an example that matches the current state. When using multiple advisors the advice that contains the best estimated reward is chosen. In trials where roles are used only agents with the same roles in different teams are considered as potential advisors.

Specific advice can be used online (requiring synchronization) or stored for offline learning. When used online it can be integrated before the action is selected or simply imitated. Online integration of specific advice is only possible in EA and QL-agents, although for EA the initial results proved that it was not a good option. The first use online BP while the second use biasing or virtual-experience (see section 5.2.3). Imitation is only possible for QL-Agents since EA and GP agents are unable to learn from the rewards of actions they did not generate. Batch advice can only be used for offline learning. The offline learning procedures for all types of agents are described in section 5.2.3. Previously saved advice can be used offline even when the agent is not requesting further advice.

The selection of the standing advisor is reviewed at fixed time-intervals. The standing advisor will be the agent with highest average reward in recent validation trials.

In this chapter we have explained in detail the environment's structure and the components that constitute our learning agents. In the following chapter we will describe the problems used for testing our approach.

Chapter 6

Experimental Framework

This chapter is dedicated to the explanation of the experimental framework. The implementation details can be found in appendix B.

The first problem (predator-prey) was chosen because it is one of the most used in MAS and it is easy to change its difficulty level by using different formulations. It was also our initial intention to compare our results with those obtained by others in this problem using the exact same formulations of the problem, but it was soon obvious that the formulations used by other authors either did not adapt to some of the learning algorithms we used, or were easily solved with the standard versions of the algorithms, not allowing us to demonstrate the utility of exchanging information.

The second problem to be approached was traffic-control. This problem had all the required characteristics and the fact that real-data was available was an important factor that led to its choice as a case-study.

The third problem we selected, in the final phase of this work, was load-balance. It was an important goal for us to find a problem in which we could compare our results to those of other researchers, and this problem seemed to offer just such an opportunity. Even though there were some difficulties, for the reasons explained below and detailed in appendix D, we have achieved a reasonable basis for comparison. Still, the most important comparison to be taken in consideration is between advice-exchanging agents and our own implementations of standard learning algorithms.

One of the main difficulties, specially in the last two scenarios, was in defining a state structure and problem parameterization which made the problem learnable, but not trivial. It was noticed that very slight variations in some parameters of the problem could quickly shift these problems from trivial to unsolvable. On the one hand we have seen that the techniques presented here have little influence in the results for problems that are easily solvable by standard methods. On the other hand it is encouraging to see that the best results were in problems that are extremely difficult to solve with standard

learning algorithms. This seems to indicate that in harder problems there is more to gain by exchanging information during learning.

6.1 Predator-Prey

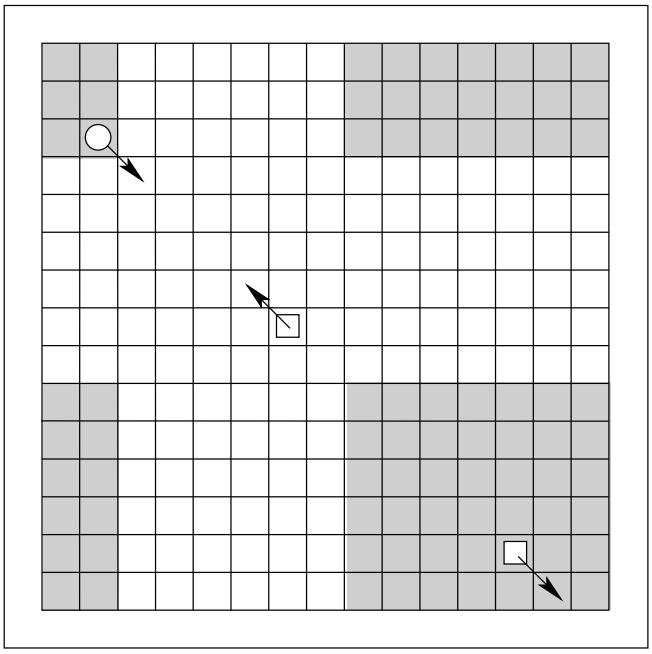

Figure 6.1: An arena of the predator-prey problem with 15×15 positions, 2 predators and 1 prey. Predators have a visual-range of 4. The white squares represent predators; the circle represents the prey; The arrows indicate the movement of the objects and the grey area defines the visual-field of the agent in the lower right corner.

This problem was first introduced in (Benda *et al.*, 1985), although the versions presented here were inspired on the variants presented in (Tan, 1993) and (Haynes *et al.*, 1995b). Each arena is a square lattice of a given dimension. An arena of 15x15 squares, is depicted in figure 6.1. The arena, in our implementation, is unbounded, i.e. an attempt to move past the bounds will cause an object to appear in the opposite end of the arena as if moving in the surface of a sphere.

In each arena there are two types of moving objects: prey and predators. Any number of objects can occupy the same position. In each cycle, all the objects are allowed to move one square in any of the eight directions available or to stay the same square, i.e. nine possible actions to choose from in each turn. The directions will be referred to according to the geographical convention: (N) North , (NE) North-East, (E) East, (SE) South-East, (S) South, (SW) South-West, (W) West, (NW) North-West. The prey moves before the predators' sensing phase.

Table 6.1: Parameters for predator-prey experiments. Comparable to tables 3.1 and 3.3.

Parameter	Value
Definition of capture	Individual: same position as prey Cooperative: #predators in vicinity > 1
Size and shape of the world	15x15, unbounded
Legal moves	All squares in vicinity (9)
Simultaneous/sequential moves	Simultaneous, except prey
Visible objects	All
Visible range	4
Predator communication	No, except for learning information
Prey movement	Heuristic (escaping-prey)
Overlapping objects	Yes
Initial placement	Random
Number of predators/prey	2/1

The predators' objective is to capture the prey. A prey is captured if at least one predator is in the same square as the prey (individual capture), or if two or more predators are in squares adjacent to the prey (cooperative capture). The overall objective of a team is to catch the prey as many times as possible.

The prey will move away from the closest predator. When several predators are equally close one is picked randomly and the prey moves away from it even if this means approaching another predator. An escaping-prey will not be allowed to move once every 10 turns to limit its speed relative to the predators.

The state observed by a predator consists of 2×9 elements. The first nine sense the prey's position, the last nine sense other predators. Each state element corresponds to one of the possible 9 directions of movement. The intensity of activation of each state element is inversely proportional to the distance between the predator and the sensed object. If (x, y) is the position of the prey relative to the predator, then the activation of the state element in the prey's direction will be $1 - 0.1 \ \max(|x|, |y|)$. The minimum activation is 0.1 when the prey is outside the visual-range. There are 1681 possible states and 15129 state-action pairs (for a visual-range of 4). When two objects of the same type are in the same direction only the closest is sensed.

Predators can sense the exact position of the closest prey and partner provided they are within the observer's visual-range. A predator's visual-range is a square of side 2 $\times$ visual-range + 1, centered at the predator.

The predators can be learning agents or H-agents (heuristic-agents). The first will choose one of the nine possible actions based on their evaluation function and current parameters. H-agents always move directly towards the prey. This is the optimal be-

havior when a single predator is considered if no assumptions are made regarding prey and partner's behavior. This policy has two drawbacks in a team scenario. First it does not account for the positions of other predators to anticipate the prey's movement (enabling the predator to move to a future position of the prey instead of the current one) and in a situation where all predators are pursuing the prey from the same side the catch will take more moves than if predators surround the prey by moving around the world in opposite directions. This second situation is increasingly less likely to happen as the number of predators in play increases. Having only two predators allowed us to judge the effect of suboptimal advice.

The individual reward for a given predator is: 1.0 if the last action has resulted in a successful catch and the predator took part in it (i.e. is in the same or an adjacent square to the prey), and $0.01 \max_j(s_j)$ otherwise, where s_j are the elements of the state that sense the prey (i.e. $j \in [0, 9]$). The balance in the reward is tied to the random relocation after a catch. The reward when a predator moves closer to the prey must indicate that this is a move in the right direction, but it must not be too high otherwise it might compensate not catching the prey. In some scenarios, with different reward structures, it was noticed that staying close to the prey would prove more efficient than actually catching it, because the random relocation of the predator might cause a series of low reward movements immediately after the catch.

The epoch-reward is n/m, where n is the number of successful catches during that epoch in a given arena, and m is the number of turns of each epoch. The number of catches is incremented once for every predator that participates in a catch. If the reward was proportional only to the number of catches, regardless of how many predators took part in it, individual catches would produce higher global rewards than cooperative catches. If $n > m$ the reward is 1.0.

In this problem the weights of combined reward are both equal to 0.5 giving equal weight to individual and global rewards. Each epoch consists of 60 turns, i.e. each agent makes 60 moves.

The functions provided by the environment to be used by STGP are the following:

- Real Functions
  - Distance to Prey
  - Distance to Partner
- Boolean Functions
  - Prey: Right, Left, Up, Down, Here, Up and Right, Up and Left, Down and Right, Down and Left, Straight Right, Straight Left, Straight Up, Straight Down
  - Partner: Right, Left, Up, Down, Up and Right, Down and Left, Up and Left, Down and Right
  - Prey Close

- Partner Close
- Partner Close to Prey

6.2 Load-Balance

The load-balance environment is similar to the one presented in (Whiteson and Stone, 2004). Each area will contain three types of objects: users, load-balancers and servers.

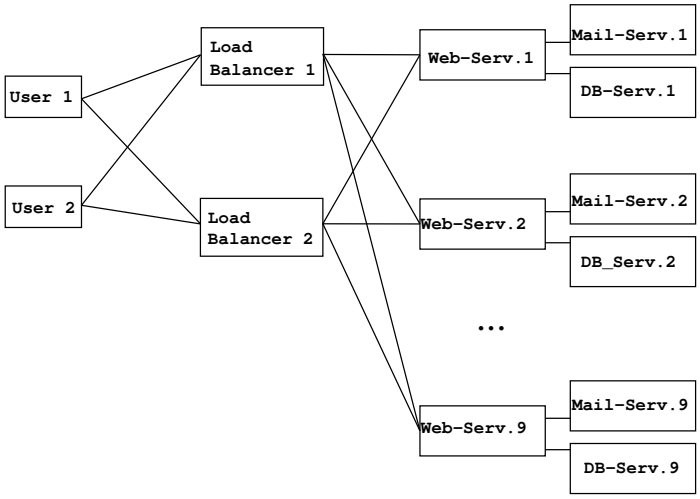

Figure 6.2: Load-balance scenario.

The two users of the system will generate jobs until a total of 500 jobs is generated. Each job will be sent to a randomly chosen router. The routers will select a server to complete the job and send it. Upon completion of each job the user will score it based on its own evaluation function and the job's completion-time. Each job will have three job-steps, each with a corresponding type. The two possible sequences of job-steps are: Web-Mail-Web and Web-Database-Web. A step of a given type can only be accomplished in a server of the same type and the steps must be performed sequentially. We chose to test our approach in one of the topologies used by Whiteson and Stone (2004), depicted in figure 6.2. In this topology servers are divided in clusters and all clusters have one server of each type. All available information concerns only the server that is directly connected to the load-balancer (i.e. the Mail-servers). The server types (and step types) considered in this case are: Web, Mail and Database.

Each server has a certain speed that defines what percentage of a job-step of that type it can complete in one cycle, e.g. a Mail Server with speed 0.5 will complete a mail-step every two turns. Our formulation of the problem differs from the one presented by Whiteson and Stone (2004) in terms of server speeds and work-load. In

the previous formulation each job-step has a certain work (number of cycles required to complete the step), and processor speeds are represented as integers. Since all job-steps of a certain type require the same amount of work, re-formulating the speed as a real number, equal to the quotient between the server's step-work and speed on the previous formulation and assuming unitary work for all job-steps, is equivalent.

Every five cycles the servers that are directly linked to load-balancers send a packet with their current load (number of pending jobs/speed). Time is discrete and communication between any two linked nodes takes one time-step.

The objective of the load-balancers is to route the jobs sent by the users as efficiently as possible according to the measure of efficiency provided by the users (all monotonically decreasing in time).

The load-balancer will route the job according to its current policy to one of the front-line servers. The action will be a vector containing the probability of choosing each server. The decision of the actual server the job will be sent to is stochastic, based on the probability-vector issued by the agent. The number of possible actions is equal to the number of available servers. Jobs are stored in a FIFO queue on arrival at a server. In each turn all servers will check the queue of pending jobs and start processing the first job in the queue.

When a server starts to process a job it will check if the processor is free. If so, it will start processing the first job. If a job is completed and there are still resources to spend, the server will pop another job out of the pending jobs queue and process it. For example, if a processor has a speed of 1.2, there are three jobs in queue and the processor is free at the beginning of the cycle, then it will process one job-step (and route the job to a server in its cluster that can complete the next step), complete 20% of the second job-step, which will remain active until the next cycle, and the third job will remain in queue. The servers' speeds in our experiments were those selected for the catastrophe scenario in Whiteson and Stone (2004), i.e. the speed of Mail Servers 1 through 9 is 0.2, 0.4, 0.6, 0.8, 1, 0.6, 0.7, 0.8, 0.9, respectively, and the speed of the Database Servers 1 through 9 is 0.9, 0.8, 0.7, 0.6, 1.0, 0.8, 0.6, 0.4, 0.2, respectively.

When a job returns to the user that generated it, the user will compare its generation-time-stamp with the current time, calculate its completion-time (i.e. the difference between the current time and generation-time) and reward the load-balancer accordingly. The user sends, along with the reward, the identifier of the completed job, so that the load-balancer can associate the reward with the state observed at the time of decision and the action taken for this situation. Each user may have a different reward function. The reward functions used in Whiteson and Stone (2004) were:

$$r_1(t_c) = \begin{cases} 1 - t_c 10, & : \quad t_c < 50 \\ 1 + t_c/10 - 495, & : \quad \text{otherwise} \end{cases} \tag{6.1}$$

$$r_2(t_c) = \begin{cases} -t_c/10, & : \quad t_c < 50 \\ -10t_c + 495, & : \quad \text{otherwise}, \end{cases} \tag{6.2}$$

where t_c is the job's completion-time. We have rescaled these functions to the interval [0,1] using a minimum of -1000:

$$r_j(t_c) = \begin{cases} 0.0, & : \quad r_{user(j)}(t_c) <= -1000 \\ 1 + r_{user(j)}(t_c)/1000, & : \quad \text{otherwise,} \end{cases} \tag{6.3}$$

where r_j is the reward for job j whose completion-time is t_c.

In this problem, the weights of combined reward are both equal to 0.5 giving equal weight to individual and global rewards. Each epoch consists of 100 turns, in each turn all users are randomly selected to generate jobs until a maximum of 500 pending jobs is reached. The number of actions each agent executes per epoch is variable.

The state observed by a load-balancer contains information about the job-type (0 for Mail-jobs and 1 for Database jobs), the user that generated it (0 for user-1 and 1 for user-2) and an *action-dependent feature* (Stone and Veloso, 1997) which encodes the last information sent by the candidate server concerning its load (in [0,1], divided into 0.01 intervals). When servers have more than 100 jobs in queue the value of the third component would be 1.0. The total number of states is $\approx 4e18$, reduced to 404 per action by the use of action dependent features. The number of state-action combinations is 3636.

An *action-dependent feature* is one that changes depending on the action the agent is considering at a given moment. In this case, if a load-balancer is considering sending the packet to server number n the third component of the state will be the load of the n-th server. This requires each component of the action vector to be evaluated separately, using a different state vector. This was one of the main reasons that led us to adopt evaluation functions that are able to process each action with an independent set of parameters.

In this problem we can consider that the state is observed several times each turn, to include the new information about the next job to be processed, or that the external state (i.e. the server loads) is only observed once per turn and the first two components are added to the state-observation every time a new job is considered. We will choose the second interpretation because it respects the definition of turn as the interval between two state observations and also because it points the way to an advisable optimization.

The heuristic strategy used was random routing. Each job was sent to a randomly selected server with an equal probability. The first experiments with this heuristic led to the discovery of an error in the initial description of the experiments, as explained in detail in appendix D.

The functions provided by the environment to be used by STGP are the following:

- Real Functions
  - Load, Low Load, Medium Load, High Load

Table 6.2: Summary of the parameters for the Load-Balance experiments.

Parameter	Value
Number of steps per job	3
Maximum pending jobs	500
Number of users	2
Number of load-balancers	2
Number of server clusters	9

- Boolean Functions
  - user1, user2
  - Mail Job, Database Job
  - Low Load, Medium Load, High Load

The real functions *Low Load()*, *Medium Load()* and *High Load()* are evaluated as follows:

$$LowLoad(j) = \begin{cases} 0 & : \quad l_j > 0.5 \\ 1 - 2l_j & : \quad \text{otherwise} \end{cases} \tag{6.4}$$

$$MediumLoad(j) = \begin{cases} 0 & : \quad 0.8 > l_j \vee l_j < 0.2 \\ 1 - 0.33|l_j - 0.5| & : \quad \text{otherwise} \end{cases} \tag{6.5}$$

$$HighLoad(j) = \begin{cases} 0 & : \quad l_j < 0.5 \\ 2(l_j - 0.5) & : \quad \text{otherwise} \end{cases} \tag{6.6}$$

6.3 Traffic-Control

The traffic-control problem used in these experiments bears some similarities to several other research simulations, such as the ones presented in (Thorpe, 1997) (Brockfeld *et al.*, 2001) and (Dresner and Stone, 2004). There is however one important difference: the traffic-flows are generated based on real data. This data was graciously made available by the Traffic-Control Department of Lisbon's City Hall. An example of a traffic scenario can be seen figure 6.3 and a sample of the used data is depicted in 6.4.

The data available was collected by magnetic sensors and gathered by Lisbon's automated traffic-control system (GERTRUDE) between 15/07/01 and 31/07/01 in five of the most busy avenues of Lisbon. It indicates the number of cars that passed through a given sensor every 5 minutes. This data was pre-processed by dividing the total number of cars passing in each period by the number of lanes in each of the streets observed.

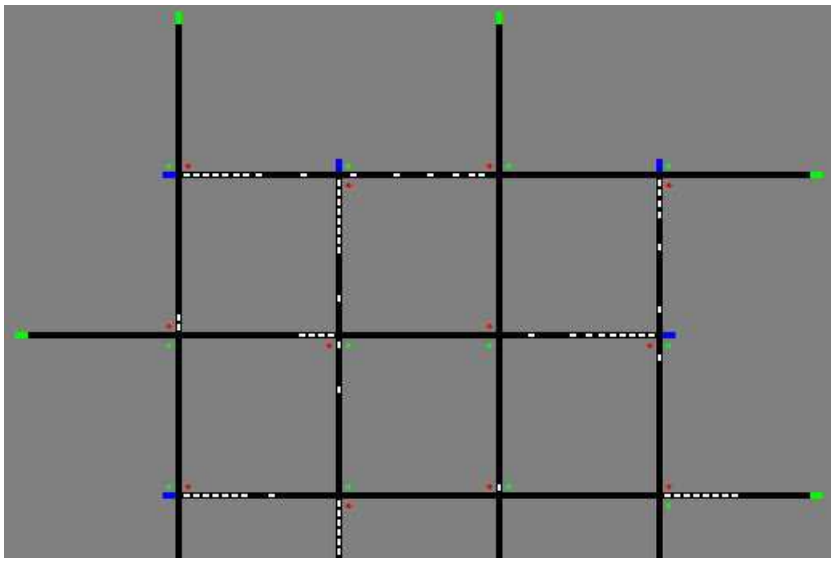

Figure 6.3: Partial view of a traffic-control area with 4 $\times$ 4 crossings (16 traffic controllers).

Other measurements were taken at the site to fine-tune the simulation, such as: typical green-time of traffic-lights (35 to 50s), average yellow-time (3s), number of cars passing a green traffic-light in one lane, after 10s, 20s, 30s, 40s and 50s ($\approx t/2$). All these measurements were done at rush-hour, when queues exceeded the traffic-lights' throughput causing traffic jams.

Each area in a traffic-control environment contains a given number of lanes arranged in a pre-established topology. Within each lane we have several sections, namely: start-of-lane, road-stretch, crossing and end-of-lane. All lanes have only one start and end-of-lane, but they may have any number of crossings, and road-stretches between them. The spatial resolution of the simulation is 1 meter and temporal resolution is 1 second, i.e the positions of all cars are recalculated for every simulated second and its positions are discrete with a minimum step of 1m.

The scenarios with 4 traffic-controllers are the ones that more closely follow the topology of the streets where data was collected. This is the main reason why we have focused on these scenarios in the experiments we are describing.

Each crossing is controlled by a different Traffic-Control Agent (TCA). TCA observe the state of the world every 12s and may choose one of two actions: set the North-South lanes traffic-light to green or red. The opposing traffic-lights will change automatically. Yellow times are automatically introduced whenever there is a change from green to red. Starvation is controlled by automatically changing the traffic-light color when it stays the same color for more than 90s, regardless of the TCA instructions. The first experiments, using an interval of 20s between two consecutive actions,

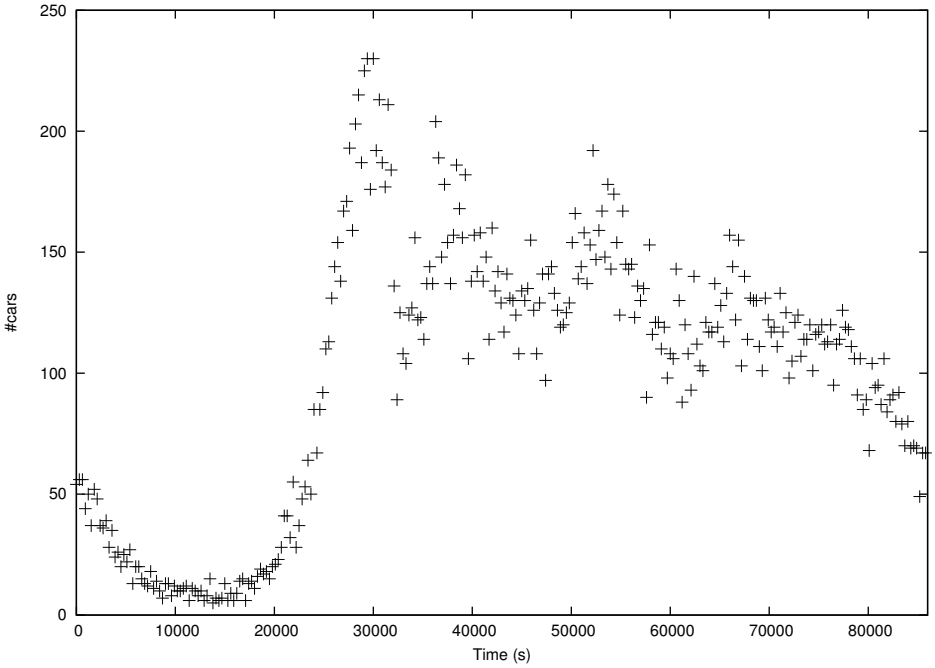

Figure 6.4: Example of the traffic flow in the 3 side-lanes of Av. Republica (North-South direction). The vertical axis represents the number of cars that passed the sensor in each 5 minutes (300s) period. Data collected on 19/07/2001, between 0h and 24h.

had much lower scores than the ones presented here, which leads us to the conclusion that quick reaction-times are essential to achieve a good performance.

The objective of each TCA is to minimize the average stopped time of the vehicles in its incoming lane-stretches and in the entire location. The overall objective of a team is to minimize the average time cars take to cross the controlled area.

The generation of cars is done as follows: at the beginning of each 5 minutes period of simulation (t_0) each lane randomly generates a delay (t_d) and the number of insertions (n_i) to be done in the 5 minutes period (i). The number of cars that must enter each lane in that period (c_i, taken from real data) is evenly distributed between the insertions (c_i/n_i), the remainder is attributed to the last insertion. Insertions are done at time $t_0 + t_d$ and every $300s/n_i$ after that. When lanes are full cars are placed on an entrance queue and the generation-time of each car is registered. Since the real traffic-flows presented a relatively easy problem we used a *flow-multiplier* (γ) to increase these flows. The actual number of cars inserted every 5 minutes is γc_i. In the experiments presented here the value of γ is 1.2.

Since the available data concerns only the number of cars that pass in a given street, all cars move in a straight line. Information on the number of cars changing direction was not made available. Car movement is calculated in a similar way to the method

described in (Nagel and Shreckenberg, 1992), which can be summarized as follows:

$$speed_t = \min(speed_{max}, speed_{t-1} + a), speed_t < speed_d \tag{6.7}$$
$$speed_t = \min(speed_{max}, speed_{t-1} + a\, r_1 - round(r_2)), speed_t \geq speed_d$$
$$d_t = \min(speed_t, dst_{max}), \tag{6.8}$$

where d_t is the distance to move, $speed_t$ is the speed at instant t, $speed_d$ is the desired speed for each vehicle, dst_{max} is the maximum distance that can be overcame without finding an obstacle (another car or a crossing with red or yellow light), $speed_{max}$ is the maximum speed allowed, a_t is the maximum acceleration, r_1 and r_2 are random numbers taken from $[0, 1]$ with uniform distribution. These random numbers account for each drivers' decision of breaking or accelerating at any given moment. All cars calculate their movement before any actual movement takes place. This ensures that all cars keep a safety-distance between them that is proportional to the speed of the vehicle directly ahead. It will also cause the acceleration of cars to be delayed, i.e. if two cars are sufficiently close and a given car moves at a certain speed s at time t, the speed of the next car at $t + 1$ is always smaller or equal to s.

To constrain the simulation-time it was necessary to simplify car movements as much as possible, thus cars have infinite breaking, i.e. they reduce speed to zero instantly. To avoid deadlocks it is pre-defined that cars will never block crossings and movement of cars in an orthogonal direction is always allowed.

A reward is issued from the environment for each agent every 12s, immediately before a new state observation. This reward r_i is calculated as follows:

$$\begin{aligned} q_i = 1 - (E(t_{stop}) - t_p)/t_p \qquad & t_p < E(t_{stop}) < 2t_p \\ q_i = 1 \qquad & E(t_{stop}) \leq t_p \\ q_i = 0 \qquad & E(t_{stop}) \geq 2t_p \\ r_i = q_i * p_{i,full} * p_{i,waiting} & \\ p_{i,full} = p^n & \\ p_{i,waiting} = p^m & \end{aligned} \tag{6.9}$$

where $E(t_{stop})$ is the average stopped time for cars in the incoming stretches, cut-off at $2t_p$, $p \in [0, 1[$ is a penalty coefficient, n is the number of incoming stretches that have an occupation rate higher than 96% (which in this case corresponds to 16 cars per lane), m is the number of stretches in which cars are waiting for more than the local patience-threshold ($t_p = 3s$) in average.

The epoch reward is

$$R_n = \begin{cases} 1 - E(t_{stop})/(g_p \cdot dif_n) & : \quad E(t_{stop}) < g_p \cdot dif_n \\ 0 & : \quad \text{otherwise} \end{cases} \tag{6.10}$$

$$dif_n = \begin{cases} c_i/K & : \quad c_i > K \\ 1 & : \quad \text{otherwise} \end{cases} \tag{6.11}$$

where c_i is the number of cars inserted in every 5 minutes period (indexed by i), $g_p = 10s$ is the global patience-threshold, dif_n is the difficulty level and $K = 40$ is an heuristic threshold that compensates for periods of heavy traffic that may, temporarily, exceed the lanes' throughput. A lane's maximum throughput, under optimal conditions, should be of approximately 67 cars in every 5 minutes.

In this problem the weights of combined reward are both equal to 0.5 giving equal weight to individual and global rewards. Each epoch consists of 1200 cycles (100 actions, in effect for 12s each).

The state is a vector of 6 real-valued attributes. The first two components represent the maximum occupation-rate of incoming stretches. The first is the maximum occupation-rate of North and South-bound stretches, while the second contains the same information for East and West-bound stretches. The third and fourth components are proportional to the maximum number of cars that entered in the last five seconds, again the former pertains to North and South-bound stretches, and the latter to East and West-bound stretches; The fifth element represents the current color of the controlled traffic-light (1 represents green in the North and South directions and 0 zero the opposite). The last component is proportional to the time since the traffic-light last changed. All values are scaled to [0,1]. Occupation-rates are calculated by dividing the number of cars present in a stretch by the maximum number of cars that can fit in the smallest stretch in the simulation (16 in this case), the result is cut-off at 1.0. The maximum number of cars entering a stretch in the past 10 seconds is also normalized to the [0,1] interval using a maximum number of cars of 5. The time since last change is normalized using a maximum of 90s. The total number of states depends on the step used for the first four elements. Using 0.1 steps we have 175,692 possible states. The number of state-action combinations would be 351,384.

The heuristic strategy consists simply in switching the color at fixed periods. All traffic-lights switch at pre-defined intervals (48s). This interval was based both in the values measured in the real scenario for the average green-time at rush-hour and in the results of experiments with different fixed timings.

The functions provided by the environment to be used by STGP are the following:

- Real Functions
  - Maximum Occupation Difference
  - Average Occupation NS, Average Occupation EW
- Boolean Functions
  - Full NS, Full EW, Busy NS, Busy EW, Very Busy NS, Very Busy EW, Incoming NS, Incoming EW

Table 6.3: Summary of the parameters for the traffic environment.

Parameter	Value
Green-time at rush-hour	35s-50s
Yellow-time	3
Number of cars passing from 10s to 50s	t/2
Car's dimensions	4x2m
Desired speed	[18,27]m/s, [64,97]Km/h
Maximum speed	30m/s, 108 Km/h
Acceleration	$6m/s^{-2}$
Max. cars entering	5s = 5 (1/sec.)
Max. time without change	90s
Heuristic fixed length	48s
Heuristic occupation tolerance	30%

Table 6.4: Summary of the parameters for the traffic experiments.

Parameter	Value
Number of lanes	16
Number of crossings	4
Number of insertions	[6,9]
Local patience threshold	3s
Global patience threshold	10s
Difficulty coefficient used	true
Flow multiplier	1.2
Max. cars in stretch	16

- Green Long NS, Red Long NS, Green Too Long NS, Red Too Long NS

A lane is considered busy or very busy if its capacity is over 50 or 80% used, respectively. A traffic-light is considered to be open/closed for long/too-long if it remained the same color for more than 40/60% of the maximum time (90s), i.e 36/54s, respectively.

In this chapter we have detailed the description of our test problems. The following chapter presents the objectives, results and discussion of each experiment.

Chapter 7

Results and Discussion

The objective of the experiments is to test how each of the components of our proposed solution affects learning. We will progress incrementally, using the predator-prey problem, from simple communication of examples using a standing advisor to the full system using the concepts of learning stages, roles and trust. The most interesting scenarios will be tested in the traffic and load-balancing problems. The ease of parameterization and control of the difficulty level in the predator-prey problem, when compared to others, led us to choose the former as the ground for a more extensive set of experiments.

The tests are divided into Experiment Sets, which are composed of several Experiments (or scenarios) that were grouped to answer a specific question. Each scenario is composed of several trials. The definitions we use for these terms are the following:

Experiment Set A logical organization of several experiments, each using different parameters

Experiment/Scenario A set of trials, with different starting conditions, but using the same number and type of agents and the same parameters

Trial A run with a fixed number of epochs at the end of which the agents' performance is verified in a series of validation epochs.

Epoch Period of time, within a trial, consisting on a fixed number of turns at the end of which a global reward is issued to each member of a team. An epoch can run in learning, test or validation modes.

Turn Period of time between two observations of the environment, in which the agent may issue one or more actions and receive the corresponding rewards.

The results presented in this section were calculated as follows. After 19980 epochs (19800 in the case of traffic-control) the performance was measured and averaged in

100 consecutive validation epochs for the predator prey problem, 160 for load-balance, and 180 for traffic-control. The results, presented in the third and fourth columns of the tables in this section, are the average and standard deviation of this result for all agents of each type in the 6 trials made for each scenario. In trials with 2 agents per area, each value results from a minimum of 12 (heterogeneous trials) and a maximum of 36 points (homogeneous trials). In the scenarios using 4 agents per area the minimum is 24 values and maximum is 72. The graphics present the average combined reward per epoch of all agents of a given type in a specific scenario. The x-axis represents the number of validation epochs while the y-axis represents the average combined-reward for the same epochs. All measurements are taken at the end of validation epochs in which the current best hypothesis is used. Each point is the result of averaging a given number of validation results (shown on the y-axis label). The difference in the number of validation epochs for different problems is related to the learning-test-validation cycles, explained in the introduction of chapter 5. The last point in each graphic includes the final validation results. When appropriate the baseline and/or heuristic results are also shown for comparison.

The Reference (Ref.) code summarizes the conditions of each scenario. The first character represents Social (S) or Individual (I) experiments, i.e. with or without communication, respectively. The second character encodes the type of learning agents present: EA-agents (E), GP-agents (G), QL-agents (Q) or Mixed (M) (i.e. all previous types). The third character (first digit) represents the number of agents per area. The last digit represents the number of heuristic areas. The number of the experiment-set and a summary of the parameters may follow the reference when necessary. All H0 scenarios have 3 areas. The homogeneous scenarios have 6 $\times$ EA, GP or QL-agents, while the heterogeneous scenarios (M) have 2 agents of each of the previous types. H1 environments have an extra area with a team of H-agents (heuristic agents, represented in the tables by the mnemonic HEU). The results for the H-agents are summarized in the last line of each table, where appropriate, with minimum and maximum scores and maximum standard deviation. The last column in each table represents the median run-time in minutes. We have chosen median times rather than other statistics because part of the experiments were made on stand-alone mode while others were not. The use of a median provides a better measure for comparison by filtering-out the high running-times achieved when running in parallel with other applications. In heterogeneous scenarios the results were averaged for each algorithm and the code of the algorithm (EA, GP, QL or HEU) is appended to the reference.

The second column contains information on the type of advice parameters used in the experiment. Only the most relevant parameters for each experiment are presented. The full description of the parameters for each case is in the text that accompanies the presentation of the results.

Appendix C presents the parameters used in each experiment, the average evolution graphics, more detailed tables for each experiment and brief comments on all experi-

ments. The following sections present a summary of the final results, the comparison of several approaches and a discussion of the results of each experiment.

7.1 Predator-Prey Baseline

The baseline experiments, as the name indicates, are a reference. These results will be compared to the following to determine the added value of the techniques used. In each of the 6 trials the simulation was composed of 3 locations, each with a team of 2 agents (Fig. 7.1). In each area agents use different learning algorithms. No communication was allowed between the agents in the baseline trials. The objective of this experiment is to set a reference performance for the predator-prey problem for each type of agent.

Given the structure of the reward (presented for the general case in Eq. 2.13 and discussed in Sect. 6.1 for the case of the predator-prey scenario) the number of turns per catch is approximately $1/R_{i,n}$.

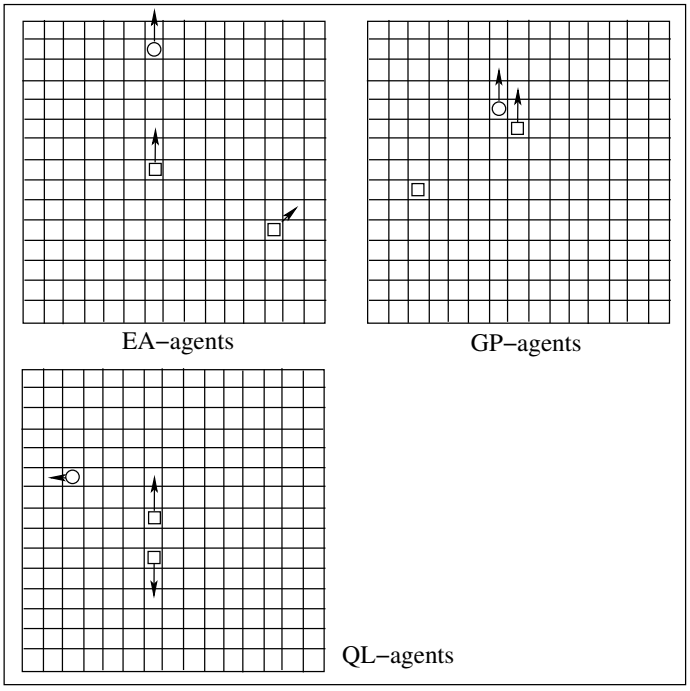

Figure 7.1: Example of three locations in a mixed scenario (S2MH0), each 15×15 positions, 2 predators and 1 prey, per location. The squares represent predators; the circle represents the prey.

Table 7.1: Summary of the final results for the predator-prey problem in the baseline experiments (without communication).

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
IM2H0-GP	No Advice	0.067	0.007	582
IM2H0-QL	No Advice	0.016	0.010	550

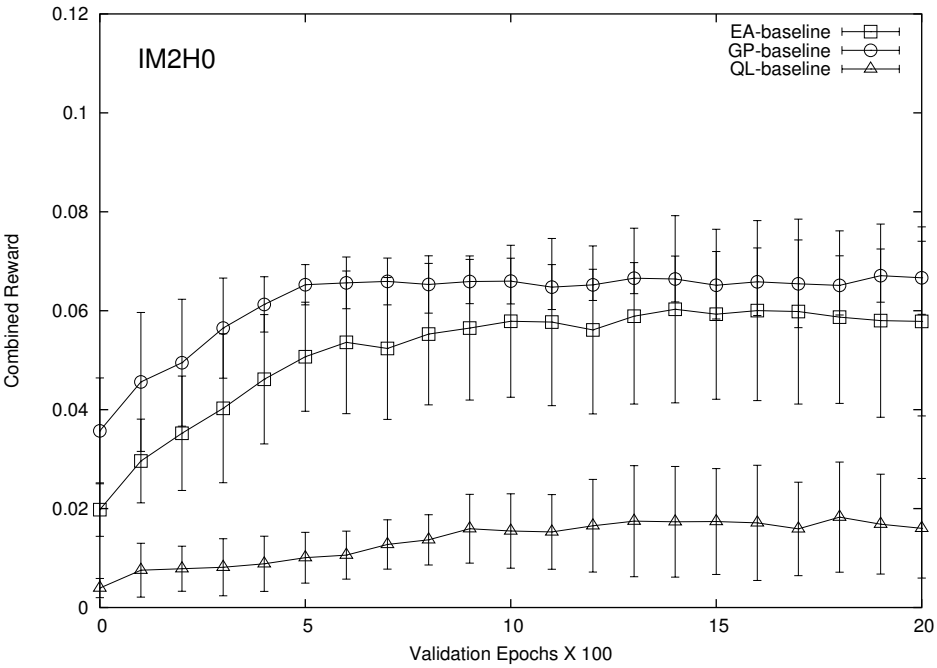

Figure 7.2: Average evolution of the combined reward in the baseline experiment (IM2H0) for the Predator-Prey problem.

Results

Observations

Performance GP shows the best performance, very close to EA. QL has a very low relative score in all trials.

The relatively low performance of QL is mainly due to the size of the arena as was concluded in (Nunes and Oliveira, 2003b). These experiments prove that 10x10 arenas, under similar conditions, are much more favorable to QL-agents. The need to persist in the same action while pursuing a prey for long periods is a handicap for QL. This is due to the use of immediate rewards, the constant changes in the Q-values and the need to explore different actions.

The score of GP-agents is mainly due to the background knowledge inserted in the external functions. It is relatively easy to learn a policy that is very similar to the one H-agents execute. i.e. always moving towards the prey. Even though this seems an advantage we can see from the small differences in behavior in all experiments that this represents a very strong local *optimum* and that GP-agents seldom escape this "trap", even when using advice.

EA-agents have a performance that was expectable, given previous experiments. They achieve a slightly lower performance than GP-agents (and H-agents, as can be seen in the following experiments) and have a slower climb during the initial phase of the training. The advantage of EA-agents when compared to QL-agents is mainly due to exploring policy-spaces rather than action-spaces. The maintenance of a coherent policy during a full epoch is advantageous in scenarios where persistence in the response to a given state is required to achieve good performances. The handicap when compared to GP-agents is related to the lack of background knowledge in EA-agents.

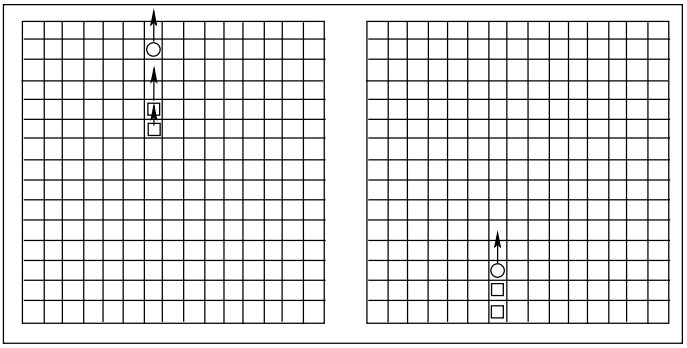

Figure 7.3: Example of pursuit by two predators in the same direction. Situation described in section 6.1, par. 7, that was common in baseline experiments. The situation on the left-side depicts the state for a time $t = 0$, while the right-side depicts the situation approximately 20 turns later, after the predator have circled around the arena twice chasing the prey.

Figure 7.3 shows a common situation in scenarios where team-work was not adequately learned (mostly for QL-agents). In this case both predators pursue the prey in the same direction taking more turns to capture the prey than if they would surround it. Starting from an initial position where both agents are close and aligned with the prey (left-side) the pursuit will continue until the prey is over-run by the predators (right-side). Given that the speed of the prey is $9/10$ of the predators this can capture can take approximately $d \times 10$ turns, where d is the initial distance between the prey and the closest predator.

7.2 Experiment Set 1: Heuristic Advisors

In this experiment we will see how much can be learned from an heuristic advisor using different types of advice. Since the advisor is suboptimal it is also interesting to see if advisees can surpass the advisors' performance. To do this we will run each scenario with 3 locations where all agents use the same learning algorithm and one location using H-agents. In this, as in all following experiments for this problem, the maximum number of examples collected from an advisor per epoch is equal to the epoch-size and the maximum number of stored examples is 10 times the epoch size. Agents request advice when the condition in Eq. 5.18 holds true. All agents can communicate with any other agent in any location. The following combinations of advice modes will be tried:

- EA/GP: Offline, Specific, Standing advisor
- EA/GP: Offline, Batch, Standing advisor
- QL: Offline, Batch, Standing advisor, Biasing
- QL: Offline, Batch, Standing advisor, Virtual-Experience
- QL: Offline, Specific, Standing advisor, Biasing
- QL: Online, Specific, Standing advisor, Biasing
- QL: Online, Specific, Standing advisor, Imitation

Online advice for EA-agents, although possible, was not tested because previous experiments proved that it did not result in any improvement in performance.

In this experiment-set the information referred in Eq. 4.1 is exchanged between advisors and advisees whenever condition 5.18 holds. In experiments where specific advice is used the advisee will request the action proposed for a the state it is experiencing ($\bar{s}_{i,t}$), and use the short-version of the advice structure:

$$e_{i,t} = \{\bar{s}_{i,t}, \bar{a}_t\}, \tag{7.1}$$

where a_t is the advised action.

When using batch-advice the advisee will receive a set of examples $e_{i,t}$ of the long-version of the advice structure (Eq. 4.1) assembled by the advisor, during its validation epochs. The state could be for example $\{0, 0.3, 0, 0, 0, 0, 0, 0.8, 0\}$ (prey far-North, partner close-South), and the advised action may be crisp $\{0, 1, 0, 0, 0, 0, 0, 0, 0\}$ (move North) or stochastic $\{0.1, 0.8, 0.02, 0.02, 0, 0.02, 0, 0.04, 0\}$ (the probability of moving in each direction clearly favoring a move to the North). The type of information sent/received depends on the type of advisor/advisee used.

EA and GP-agents will learn from each set of examples by training one specimen in each generation with previously collected advice-sets. EA-agents will use Backpropagation for this purpose, while GP-agents will use ID3.

QL-agents have several ways of integrating the knowledge, as explained in section 4.3. For example in Virtual Experience they will replay the collected advice just as if they had experienced it themselves, i.e. adapting the Q-values as shown in 2.1.3.

QL-agents are the only ones that integrate advice online. In the case of Biasing the change in Q-values is done as shown in Eq. 5.9 and occurs before they decide which action to take. The decision is taken in the normal way (using Boltzmann selection, as seen in Eq. 2.7) after the biasing of the Q-values. In the case of imitation the QL-agent simply imitates the advice it was given (i.e. chooses the advised action), and learns from the consequences (reward obtained).

The questions to answer with this experiment are:

1. Can learning agents adopt the advisors' behavior?
2. How efficient is each method/algorithm in integrating the advice?
3. Can they overcome the local *optimum* and achieve better performances than their advisors?
4. How much information do they need and for how long?
5. What is the delay induced by processing external information?

Results

Results of the baseline tests (Tab. 7.1) are included in all tables for comparison. All experiments use standing advisors.

Table 7.2: Summary of the final results for EA-agents in Experiment 1.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
SE2H1.1-EA	Offline Batch	0.082	0.012	582
SE2H1.1-EA	Offline Specific	0.077	0.008	550
SE2H1.1-HEU	–	0.067-0.068	0.003	–

Table 7.3: Summary of the final results for GP-agents in Experiment 1.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-GP	No Advice	0.067	0.007	178
SG2H1.1-GP	Offline Batch	0.069	0.007	162
SG2H1.1-GP	Offline Specific	0.068	0.009	204
SG2H1.1-HEU	–	0.068-0.070	0.003	–

Table 7.4: Summary of the final results for QL-agents in Experiment 1.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-QL	No Advice	0.016	0.010	178
SQ2H1.1-QL	Offline Batch Biasing	0.069	0.012	331
SQ2H1.1-QL	Offline Batch Virt. Exp.	0.079	0.010	260
SQ2H1.1-QL	Offline Specific Biasing	0.073	0.014	362
SQ2H1.1-QL	Online Specific Biasing	0.017	0.010	424
SQ2H1.1-QL	Online Specific Imitate	0.076	0.007	245
SQ2H1-HEU	–	0.068-0.070	0.003	–

Table 7.5: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 1. All values in average number of examples per agent $\times$ epoch. Example-size is 232 bytes for Virtual-Experience integration while all others only require 144 bytes per example.

	EA	GP	QL	HEU	Type
EA	2.63	0	0	1.39	Offline-Batch
EA	0.82	0	0	0.51	Offline-Specific
GP	0	1.75	0	1.22	Offline-Batch
GP	0	1.02	0	0.44	Offline-Specific
QL	0	0	1.99	12.61	Offline-Batch-Biasing
QL	0	0	1.08	3.01	Offline-Batch-Virtual Experience
QL	0	0	0.34	0.54	Offline-Specific-Biasing
QL	0	0	0	45.69	Online-Specific-Biasing
QL	0	0	0.88	5.99	Online-Specific-Imitation

Observations

Performance EA and QL-agents reached a performance level greater or equal to their advisors. GP-agents show slightly lower average performance than H-agents at the end of the trial but with higher standard deviations. The comparison of the best results using communication and the baseline results shows improvements of 2% for GP, 41% for EA and 495% for QL-agents.

Standard Deviations Standard deviations are at the same level as in the baseline trials and still much higher ($\approx 3\times$) than for H-agents.

GP-agents As can be observed in Fig. 7.5 the learning curves for all GP-agents experiments are similar, although we can see that GP-agents converge faster to a good performance when using advice.

QL-agents In QL-agents experiments (Fig. 7.6) Offline-Specific advice with Imitation and Offline-Batch with Virtual-Experience show better initial response and all,

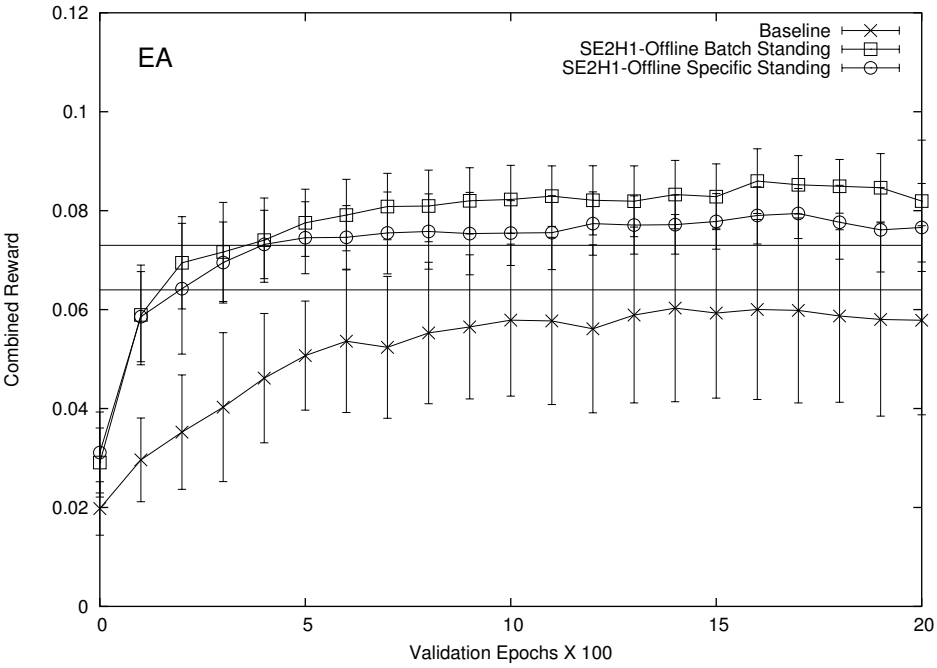

Figure 7.4: Evolution of the average combined-reward for EA-agents in Experiment Set 1.

except Offline-Specific-Biasing, have significantly better results than the baseline approach.

EA-agents EA-agents show significant improvements over the individual approach (Fig. 7.4) with an advantage for Offline-Batch advice-mode.

Batch-advice In EA-agents batch advice shows a slight advantage all throughout the training (Fig. 7.4) although final results are within the standard deviation of Specific trials.

Performance with same amount of data Considering that agents using standing advisors receive, at best, twice as much information as non-communicating agents, the comparison of performances for equal amounts of data corresponds to comparing baseline final scores with the mid-training scores for advice-taking agents. This comparison is clearly favorable to advice-taking agents for EA and QL.

Run-times The run-times increase by a factor of 3 (EA) and 2 (QL). In GP agents the run-times are similar to the baseline trials.

Communication In terms of communication (table 7.5) we can see that Batch scenarios involve (in general) more communication than Specific ones, regardless of the learning algorithm used. The exceptions are QL Online-Specific-Biasing and Imitation. The first, due to its low performance, requires advice during the whole training (45, in a maximum of 60 examples per epoch from each advisor). It is interesting to notice that in the two scenarios with best performances for QL-agents, Virtual-Experience and Imitation, the former has roughly half

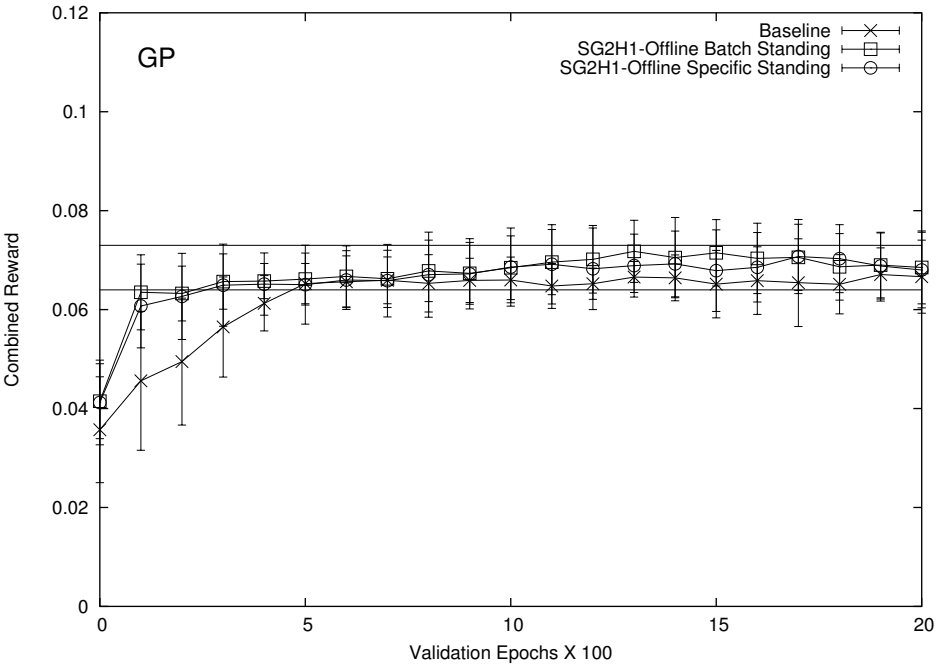

Figure 7.5: Evolution of the average combined-reward for GP-agents in Experiment Set 1.

the communication cost of the latter. In most cases communication costs are proportional to the time agents take to reach a good performance. QL Offline-Specific-Biasing has relatively low communication costs compared to other QL scenarios that reach similar final results.

Advisor switch EA and QL agents stop getting advice from H-agents during the training and chose other learning agents as advisors. The switch-over happens near epoch 500 in EA-agents (when its results surpass those of the best H-agent) and later for QL-agents (as can be observed in Fig. 7.4 and 7.6).

Figure 7.7 shows a common situation in this experiment where agents learn to surround the prey and achieve a capture in less time (7 turns) than in the situation described in Fig. 7.3 (approximately 20 turns). Strategies like this are developed more often when using advice because the communication allows a quick sharing of experiences. Communication accelerates the adoption of basic predator skills (approaching the prey) very early in the training, by learning from H-agents. This allows more time to explore different strategies and match the strategies of both partners, enabling learning agents to out-perform the individualistic H-agents at the end of the training.

Reviewing the questions posed above we can conclude that:

1. Learning agents can improve their capabilities by exchanging information.

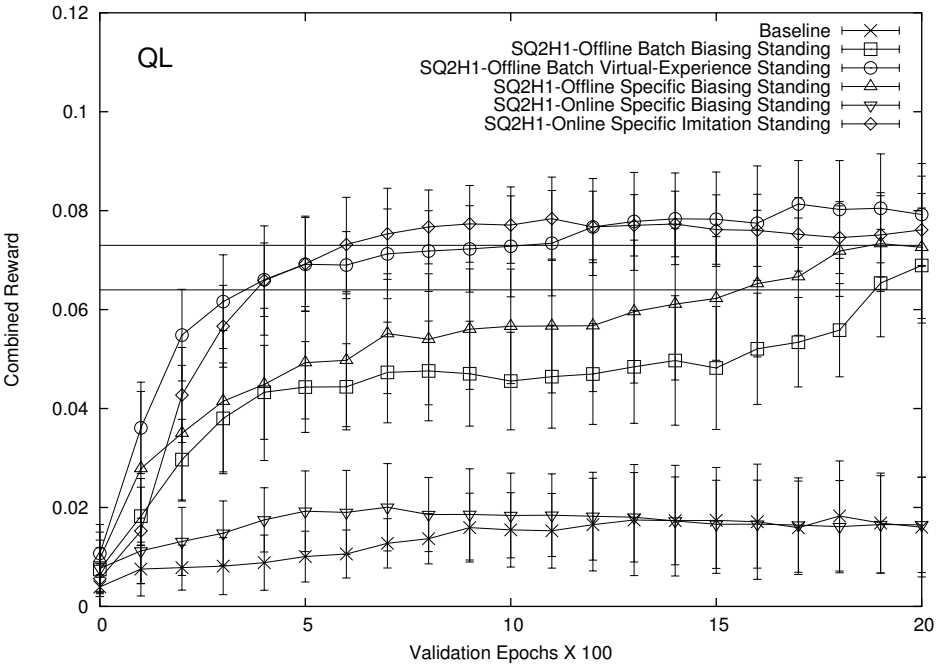

Figure 7.6: Evolution of the average combined-reward for QL-agents in Experiment Set 1.

2. They can overcome suboptimal advice and learn better policies than, both, their advisors and their non-communicating peers (with the exception of GP-agents).
3. The amount of information required depends heavily on the type of integration and its efficiency.
4. The maximum delay induced by processing the extra information is a factor of 3 for EA agents, 2 for QL-agents and lower than 1.5 for GP-agents.

Given the communication costs for EA-Batch (4.02) and for EA-Specific (1.33), and their rewards (0.0819 and 0.0766, respectively), as well as the reward for EA-agents without communication (0.0578) we can calculate a measure of the efficiency of communication as $(0.0819-0.0578)/4.02 = 0.005995$ and $(0.0766-0.0578)/1.33 = 0.014135$. So, in this case, even though the final results for EA-Batch are better, EA-Specific agents make a more efficient use of communication.

The same reasoning for GP agents produces a factor of 0.00083 for GP-Batch and 0.00088 for GP-Specific, revealing a low efficiency in the use of communication when compared to other learning agents, as expectable given the low performance increase.

For QL-agents, the factors are the following: QL-Offline-Batch-Biasing: 0.003648; QL-Offline-Batch-Virtual Experience: 0.01545; QL-Offline-Specific-Biasing: 0.06420; and QL-Online-Specific Imitation: 0.00872. Offline-Specific-Biasing stands-out for its efficiency in using communication, and Virtual Experience shows an efficiency comparable to EA-Batch.

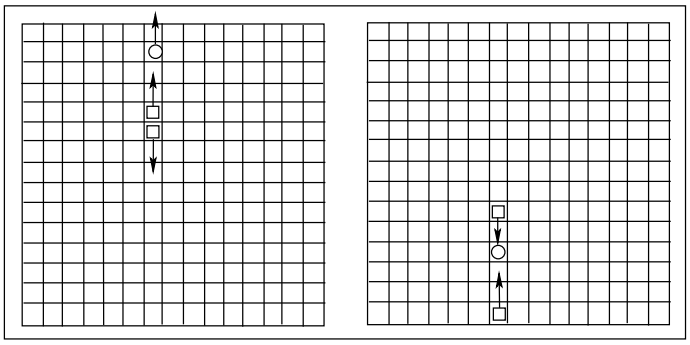

Figure 7.7: Example of pursuit by two predators surrounding the prey. Situation described in section 6.1, par. 7. The situation on the left-side depicts the state for a time $t = 0$, while the right-side depicts the situation 5 turns later.

The bad results for Online-Specific-Biasing can be attributed to an early freezing of a single hypothesis over others, which prevents exploration. This is due to the repetition of the same advice for specific patterns and the generalizations made by the underlying ANN. This problem has less impact on table-based QL and on Offline Biasing. In the first case because there is no generalization and in the second because the effect of similar advice is reduced by training with large sets of examples.

GP-agents' lower performance is caused by the difficulty in overcoming a local *optimum*. This local *optimum* is similar for H-agents and GP-agents due to the structure of the background knowledge used by both types of agents. It is admissible that different background knowledge could overcome this problem, although these experiments do not prove this fact.

7.3 Experiment Set 2: Homogeneous and Heterogeneous Advisors

In this experiment set we use five scenarios without heuristic agents. In the first three (SE2H0, SG2H0 and SQ2H0) there are 3 locations (homogeneous) and all agents are of the same type in each of the scenarios. In the remaining 2 scenarios (SM2H0-2 Standing and SM2H0-2 Multiple) each of the 3 locations has a different type of learning agents (heterogeneous). In the first we use standing advisors, in the second we use multiple advisors. All EA and GP-agents use Offline-Batch advice. QL-agents use Offline-Batch advice and Virtual-Experience. The objectives of this experiment are:

1. To compare the results of homogeneous with heterogeneous teams.
2. Verify how the use of multiple advisors influences learning.
3. Verify if agents with low performance can improve their results by collaborating with other low performing agents.

Results

Results of the baseline tests are included in the tables for comparison as well as the best result obtained this far for each type of agent.

Table 7.6: Summary of the final results for EA-agents in Experiment 2.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
SE2H1.1-EA	Offline Batch	0.082	0.012	582
SE2H0.2-EA	Offline-Batch	0.078	0.004	435
SM2H0.2-EA	Standing Advisor	0.078	0.004	464
SM2H0.2-EA	Multiple Advisors	0.077	0.008	630

Table 7.7: Summary of the final results for GP-agents in Experiment 2.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-GP	No Advice	0.067	0.007	178
SG2H1.1-GP	Offline Batch	0.069	0.007	162
SG2H0.2-GP	Offline Batch	0.071	0.006	209
SM2H0.2-GP	Standing Advisor	0.071	0.009	464
SM2H0.2-GP	Multiple Advisors	0.072	0.008	630

Table 7.8: Summary of the final results for QL-agents in Experiment 2.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-QL	No Advice	0.016	0.010	178
SQ2H1.1-QL	Offline Batch Virt. Exp.	0.079	0.010	260
SQ2H0.2-QL	Offline Batch Virt. Exp.	0.079	0.008	265
SM2H0.2-QL	Standing Advisor	0.078	0.006	464
SM2H0.2-QL	Multiple Advisors	0.077	0.008	630

Observations

Heterogeneity Contrary to our initial expectations neither heterogeneous, nor multiple-advisor scenarios show significant improvements over the best final results achieved in homogeneous and standing advisor scenarios, respectively. There is a slight improvement in GP-agents' average performance, although within the standard deviation of previous results.

Table 7.9: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 2. All values in average number of examples per agent $\times$ epoch. Example-size is 232 bytes for Virtual-Experience integration while all others only require 144 bytes per example.

	EA	GP	QL	Type
EA	5.66	0	0	SE2H0.Offline-Batch-Standing
GP	0	5.04	0	SG2H0.Offline-Batch-Standing
QL	0	0	7.81	SQ2H0.Offline-Batch-Standing-Virt. Exp.
EA	0	3.99	2.48	SM2H0-EA-Standing
GP	9.96	0	3.86	SM2H0-GP-Standing
QL	7.26	4.35	0	SM2H0-QL-Standing
EA	8.88	17.76	17.76	SM2H0-EA-Multiple
GP	19.72	9.86	19.72	SM2H0-GP-Multiple
QL	24.18	24.18	12.09	SM2H0-QL-Multiple

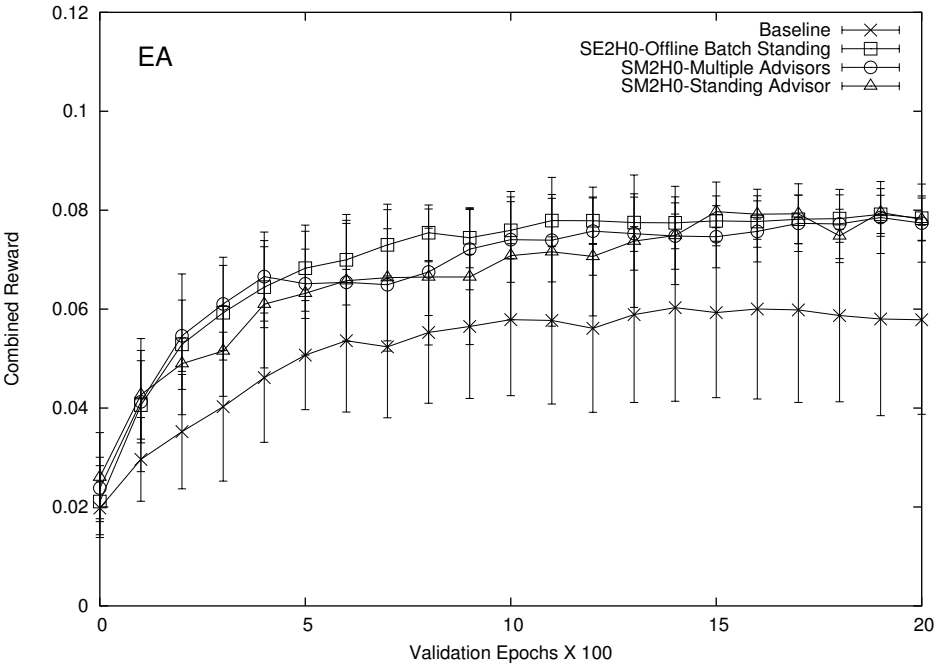

Figure 7.8: Evolution of the average combined-reward for EA-agents in Experiment Set 2.

Smaller Standard Deviations It is interesting to observe that, in general, there is a reduction in the standard deviations when compared to the results of previous experiments.

QL-agents In Fig. 7.10 we can detect a better initial climb in QL-agents when in

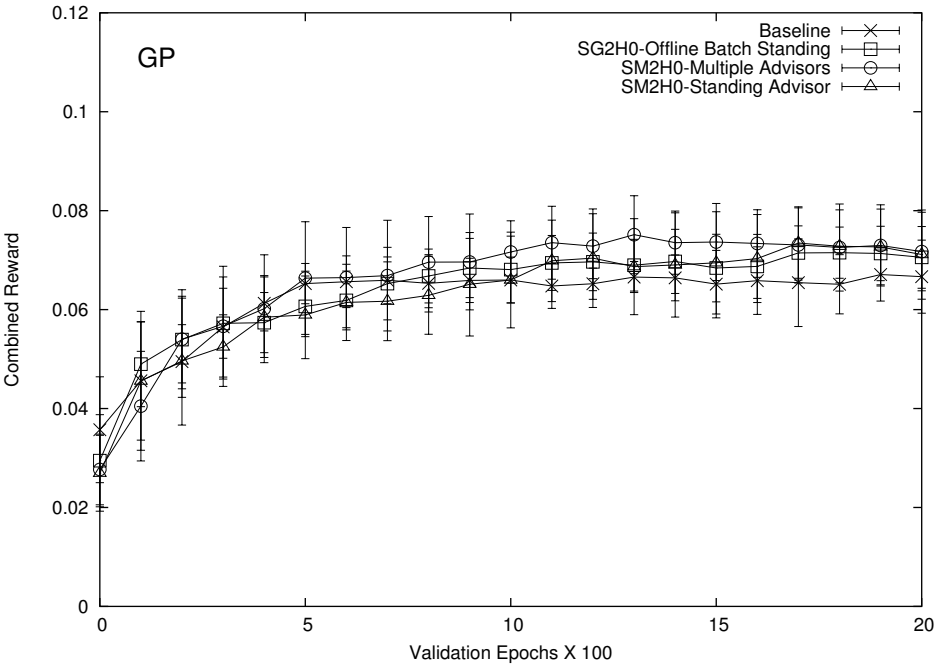

Figure 7.9: Evolution of the average combined-reward for GP-agents in Experiment Set 2.

heterogeneous scenarios, which is understandable given their low scores in the baseline trials. The final results of exchanging information for QL-agents prove to be more efficient than the baseline trials in all scenarios.

Multiple Advisors It can also be observed that the use of multiple advisors leads to faster convergence for QL-agents despite the concurrent use of information from several sources and its integration in the same hypothesis. All other curves are very similar. In all scenarios there is a slight difference throughout the training between using multiple or standing advisors in heterogeneous environments (SM2H0) which is, generally, favorable to the former.

Performance with same amount of data Considering that agents using standing advisors receive a maximum of twice, and agents that use multiple advisors five times, as much information as non-communicating agents, the comparison of performances for equal amounts of data corresponds to comparing baseline final scores with the mid-training scores for agents using standing advisors and validation scores for point 4×100 for multiple advisors. This comparison is clearly favorable to advice-taking agents for EA and QL. The same comparison between the use of standing advisors and multiple advisors is, in general, favorable to the use of standing advisors.

Advice All scenarios that use advice-exchange, for EA and QL-agents, produce better results than in individual trials. GP-agents seem unable to escape the local

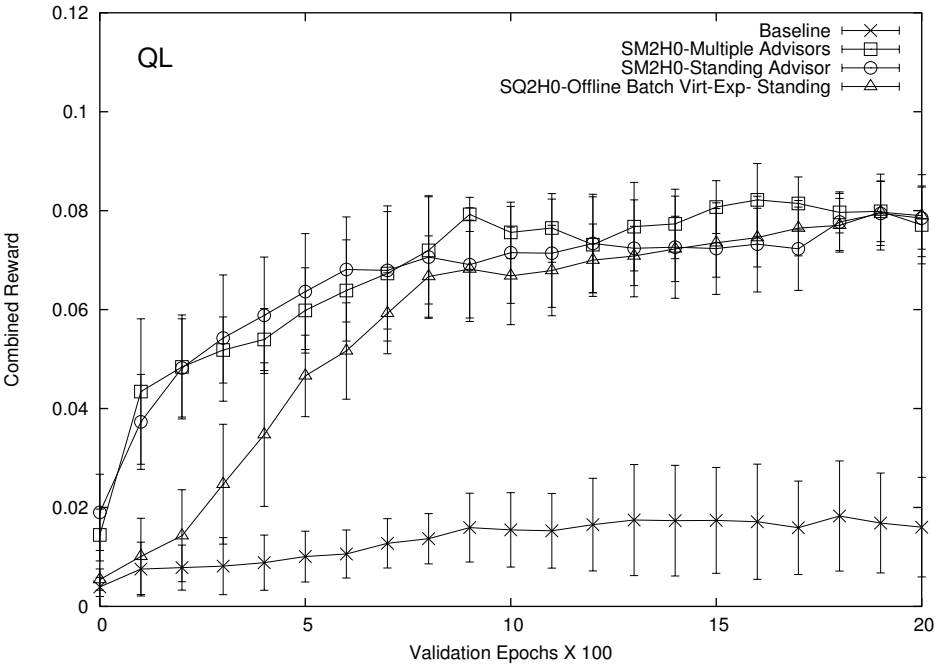

Figure 7.10: Evolution of the average combined-reward for QL-agents in Experiment Set 2.

optimum even when communicating with peers that achieve better results.

Run-times Median run-times increase up to 2.5 times in homogeneous trials, and up to 3.5 when using multiple advisors, relative to the baseline trials.

Communication In terms of communication, there is a slight increase from homogeneous scenarios to heterogeneous ones when using Standing advisors, and a relatively large increase when Multiple advisors are used. The amount of communication in Multiple advisors' trials is much higher because agents always request all their peers to send advice when their score is not near the best score (caching). Still, not all this advice will actually be used. If only the used advice would be accounted for the values would be much closer to those of Specific advice. The advantage of caching advice avoiding communication bursts may balance the disadvantage of requiring more information.

Regarding the questions posed in this experiment we conclude that:

1. Heterogeneity may cause a faster adaptation for low-performance agents when high-performance agents are present. This was already verified in previous experiments (Nunes and Oliveira, 2005a). However, the final results are similar to those obtained in homogeneous scenarios.
2. The use of multiple advisors shows a slight advantage over standing advisors during training, specially in the initial phase for low performing agents, although

always within the same standard deviation interval.

3. Communication between agents with low performance can improve their results. All scenarios in which communication is used have better training and test results than baseline trials, although for GP the increase in average performance is not significant, given the standard deviation of the results.

7.4 Experiment Set 3: Specific Advice, Roles and Trust

The objective of this experiment is to verify the effects of fixed role assignment (interdicting the exchange of information between agents with different roles, SM2H0.3 Roles), and adding trust and learnability to select advisors (SM2H0.3 Roles + Trust).

Using Roles should allow a better synchronization of advice requests between various team members, while Trust should allow a better selection of advisors based only on the past history of interactions.

Results

Experiments in this set have the same parameters used experiment SM2H0.2 with Multiple advisors, except for the use of Offline Specific (OffS) advice, Roles and Trust. EA and GP-agents use Offline Specific (OffS) advice and Multiple advisors (Mult) and QL-agents use Offline Batch (OffB) advice, Multiple advisors and Virtual Experience (MVExp). Results of the baseline tests (first three lines) and the best results for each type of agent (lines 3 to 6) are included in table 7.10 for comparison.

Table 7.10: Summary of the final results for Experiment 3.

Ref.	Advisor Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
IM2H0-GP	No Advice	0.067	0.007	178
IM2H0-QL	No Advice	0.016	0.010	178
SE2H1.1-EA	Off. Batch	0.082	0.012	582
SM2H0.2-GP	Off. Batch Mult.	0.072	0.008	630
SM2H0.2-QL	Off. Batch Stand.	0.078	0.006	464
SM2H0.3-EA	OffS Mult. + Roles	0.081	0.011	506
SM2H0.3-GP	OffS Mult. + Roles	0.066	0.005	506
SM2H0.3-QL	OffB MVExp.+ Roles	0.084	0.002	506
SM2H0.3-EA	OffS Mult. + Roles+Trust	0.072	0.006	236
SM2H0.3-GP	OffS Mult. + Roles+Trust	0.065	0.005	236
SM2H0.3-QL	OffB MVExp. + Roles+Trust	0.071	0.010	236

Table 7.11: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 3. All values in average number of examples per agent × epoch. Example-size is 144 bytes.

	EA	GP	QL	Type
EA	0	1.41	1.41	SM2H0.3 + Roles
GP	7.88	0	7.81	SM2H0.3 + Roles
QL	23.64	23.64	0	SM2H0.3 + Roles
EA	0	1.54	1.52	SM2H0.3 + Roles + Trust
GP	9.59	0	9.54	SM2H0.3 + Roles + Trust
QL	90.89	90.89	0	SM2H0.3 + Roles + Trust

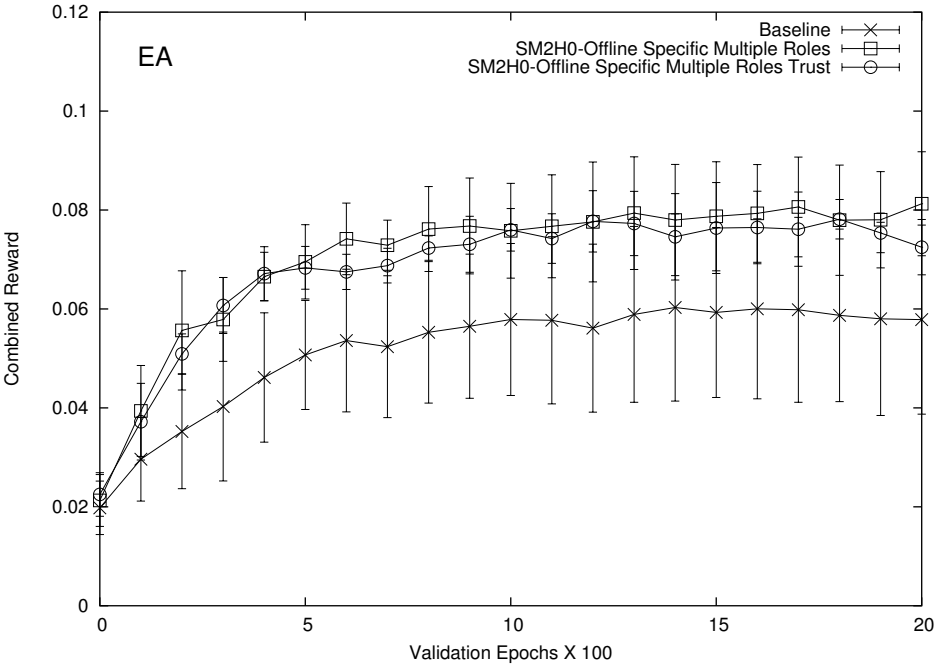

Figure 7.11: Evolution of the average combined-reward for EA-agents in Experiment Set 3.

Observations

Roles The use of Roles produced the best overall results for QL-agents. EA-agents are also at their best level. It is interesting to notice also the significant reduction of the standard deviation in QL-agents' results when compared to previous experiments.

Performance QL-agents, the ones with lowest performance in individual trials, improved their performance approximately 5.2 times, and EA-agents 1.4 times.

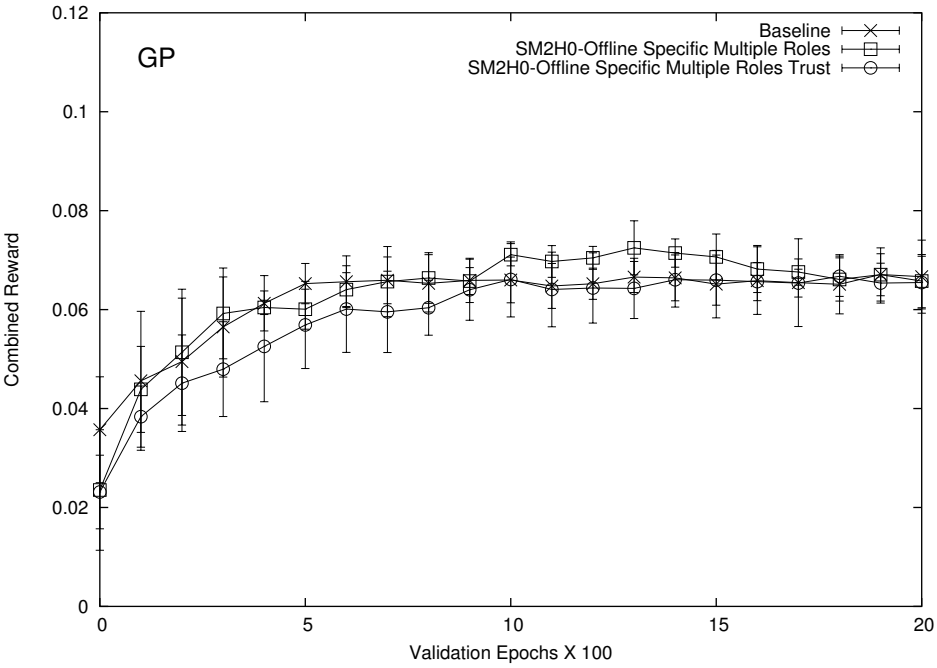

Figure 7.12: Evolution of the average combined-reward for GP-agents in Experiment Set 3.

GP-agents' performance is in the same range as in previous tests.

Trust The use of this particular implementation of Trust leads, in general, to worst results than using Roles, with similar communication costs.

Run-times The run-times, when using Roles, are higher by a factor of 2 than when using Trust, even though the communication is nearly the same (higher even in the case of QL-agents). This indicates that including Trust in the selection of the advisors causes less specimens to be trained.

Communication In terms of communication, there is a reduction in communication costs when using Trust in all cases except QL-agents + Roles + Trust.

The two main conclusions that can be drawn from these experiments are that restricting advice to agents with the same role is effective for EA and QL-agents, limiting communication and producing better overall results. Trust produces very poor results, which leads us to conclude that persistence in getting advice from a given peer may pay-off, even when the first experiments prove to be unsuccessful. The most common situation we have identified in the experiment logs is when an agent starts to request advice before its partner. This will cause an initial drop in performance due to non-matching policies. By the time the second agent starts to request advice, the first stops trusting its current advisor and will switch to a different one, leading, once more, to the adoption of non-matching policies. This phenomenon, along with the use of less information, seem to be the main causes for the poor results achieved when using Trust.

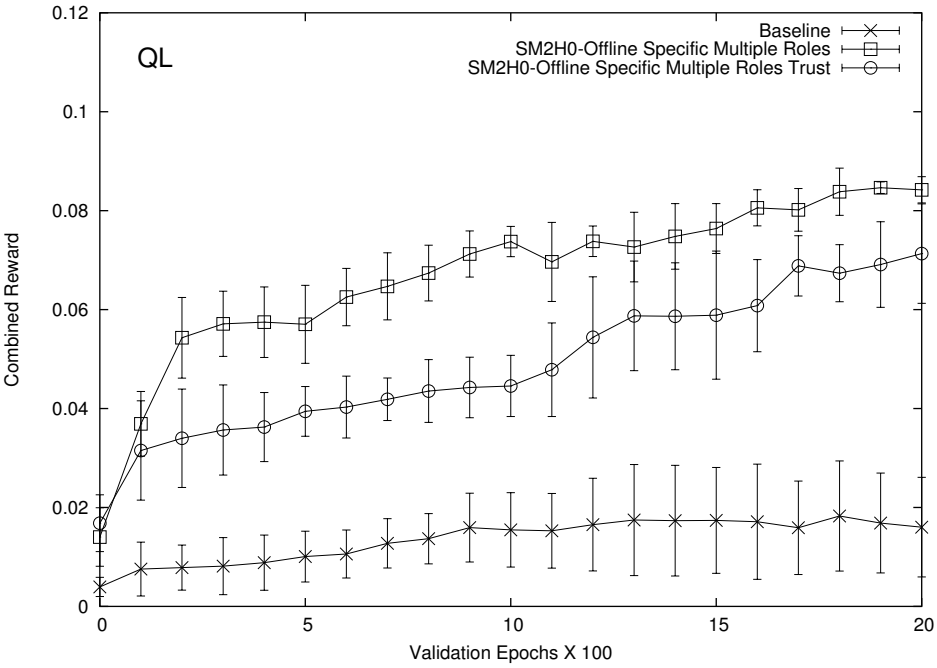

Figure 7.13: Evolution of the average combined-reward for QL-agents in Experiment Set 3.

7.5 Experiment Set 4: Batch Advice, Roles and Trust

This experiment is complementary to the previous one using Batch advice in EA and GP-agents and experimenting with Roles and Trust separately.

Results

Observations

Performance Final results for EA and QL-agents are relatively low when compared to other approaches where communication is used. Most agents take longer to reach reasonable performance levels.

Communication QL-agents' communication dropped significantly when compared to Experiment Set 3, while EA-agents' increased.

Roles Interestingly, the use of Roles for GP-agents leads to a slight rise in performance, that was never achieved in previous experiments. Still, the performance collapses to the values found in other experiments (with lower variance). This seems to indicate that GP-agents can encode a policy that results in better average behavior, but this policy is unstable (has higher standard deviation) and easily collapses to a more stable local *optimum*.

Table 7.12: Summary of the final results for Experiment 4. Results of Baseline (lines 1 to 3), Experiment Set 2 (lines 4 to 5), and best performances (lines 7 to 9) introduced for comparison.

Ref.	Advisor Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
IM2H0-GP	No Advice	0.067	0.007	178
IM2H0-QL	No Advice	0.016	0.010	178
SM2H0.2-EA	Off. Batch Mult.	0.077	0.008	630
SM2H0.2-GP	Off. Batch Mult.	0.072	0.008	630
SM2H0.2-QL	OffB MVExp.	0.077	0.008	630
SE2H1.1-EA	Off. Batch	0.082	0.012	582
SM2H0.2-GP	Off. Batch Mult.	0.072	0.008	630
SM2H0.3-QL	OffB MVExp.+ Roles	0.084	0.002	506
SM2H0.4-EA	Off. Batch Mult. + Roles	0.070	0.008	623
SM2H0.4-GP	Off. Batch Mult. + Roles	0.069	0.008	623
SM2H0.4-QL	OffB MVExp + Roles	0.078	0.006	623
SM2H0.4-EA	Off. Batch Mult. + Trust	0.076	0.006	268
SM2H0.4-GP	Off. Batch Mult. + Trust	0.066	0.005	268
SM2H0.4-QL	OffB MVExp + Trust	0.075	0.007	268

Table 7.13: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 4. All values in average number of examples per agent $\times$ epoch. Example-size is 144 bytes.

	EA	GP	QL	Type
EA	0	11.10	10.68	SM2H0-Roles
GP	9.65	0	8.39	SM2H0-Roles
QL	13.08	13.08	0	SM2H0-Roles
EA	2.73	5.28	5.04	SM2H0-Trust
GP	7.08	3.36	7.20	SM2H0-Trust
QL	12.84	11.22	6.03	SM2H0-Trust

The impact of EA and GP-agents' lower scores, and the reduction of communication, on QL-agents' performance seems to indicate that QL-agents' previous good scores (in Experiment Set 3) are dependent on the performance of its advisors. Contrary to the other two types of agents, EA-agents show a better performance in the Trust-scenario than when using Roles. Although this is an isolated phenomenon from which it is difficult to draw conclusions we hypothesize that EA-agents are more dependent on the quantity of information than on its quality, due to the specific character-

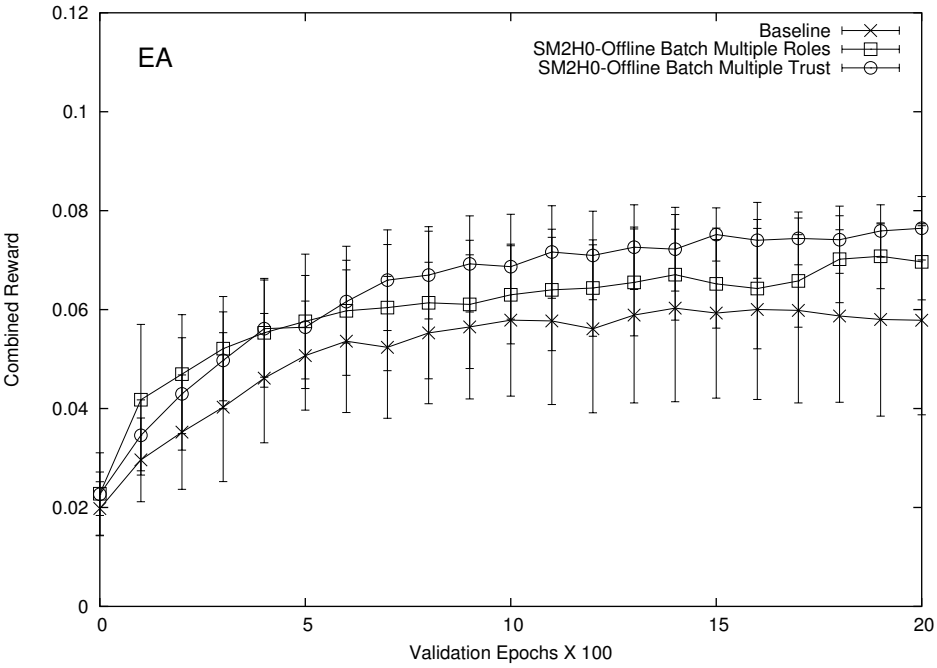

Figure 7.14: Evolution of the average combined-reward for EA-agents in Experiment Set 4.

istics of the learning algorithms used and the fact that multiple advisors' information is integrated in parallel in different specimens.

7.6 Experiment Set 5: Learning Stages and Adaptation of Learning Parameters

In this experiment we introduce:

- Learning-stages;
- Adaptive advice modes (SM2H0.5 ChgAdvModes);
- Adaption of learning parameters (SM2H0.5 ChgLearnParams);
- Learning-stages using both of the above (SM2H0.5 ChgModesAndParams).

The objective of this experiment is to verify the effects of learning-stages with dynamic advice modes (SM2H0.5 ChgAdvModes) and dynamic learning parameters (SM2H0.5 ChgLearnParams) and the mixture of the two methods.

The number of advisors and type of advice in experiments SM2H0.5 ChgAdvModes and SM2H0.5 ChgModesAndParams is decided online according to the agents' learning-stage. Experiment SM2H0.5 ChgLearnParams use the same parameters as

SM2H0.2. In Novice stage EA and GP agents use Offline, Batch, and Standing Advisors, QL agents use Offline, Batch, Virtual-Experience and Standing Advisors. On Intermediate stage EA and GP agents use Offline, Specific and Multiple advisors, QL agents use Offline, Batch, Virtual-Experience and Multiple Advisors. When changing only learning parameters EA and GP agents use Offline, Batch, and Multiple Advisors, QL agents use Offline, Batch, Virtual-Experience and Multiple Advisors.

Results

Results of the baseline tests and best results are included in the tables for comparison.

Observations

Performance With the exception of SM2H0.5-QL+ChgAdvModes, all scores are relatively low when compared to the best results of previous experiments although still higher than baseline results for all but GP-agents.

Adaptive Learning Parameters The decrease in performance is specially significant for scenarios in which learning parameters are dynamic.

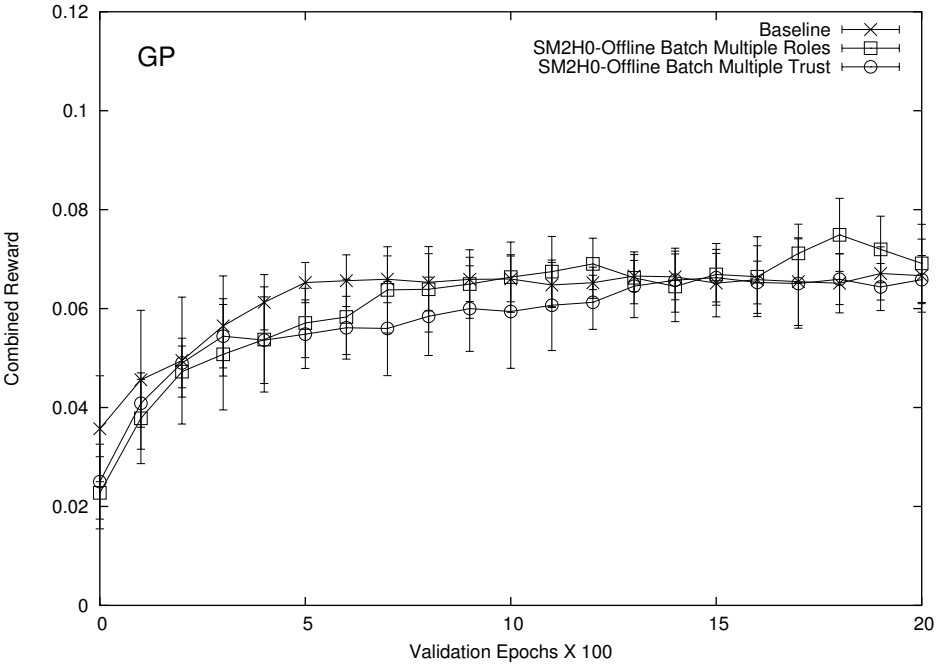

Figure 7.15: Evolution of the average combined-reward for GP-agents in Experiment Set 4.

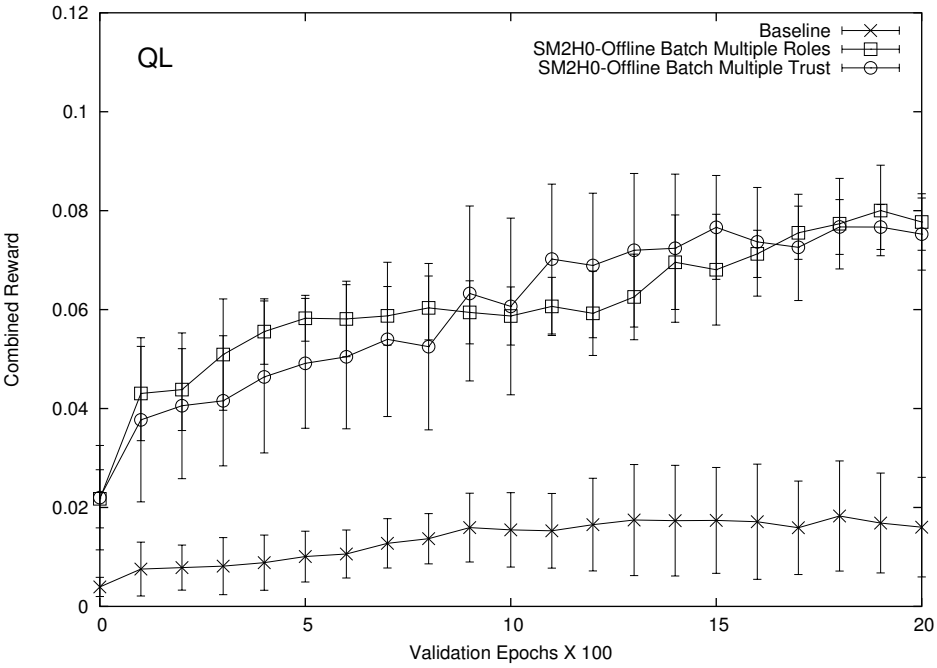

Figure 7.16: Evolution of the average combined-reward for QL-agents in Experiment Set 4.

Table 7.14: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 5. All values in average number of examples per agent $\times$ epoch. Example-size is 144 bytes.

	EA	GP	QL	Type
EA	1.08	7.62	7.98	ChgModes
GP	19.46	2.82	17.60	ChgModes
QL	15.27	13.95	1.86	ChgModes
EA	0	5.07	1.71	ChgParams
GP	2.24	0	.72	ChgParams
QL	47.16	46.92	23.46	ChgParams
EA	2.16	11.43	12.78	ChgModesAndParams
GP	23.45	2.85	23.87	ChgModesAndParams
QL	10.53	9.27	1.53	ChgModesAndParams

Contrary to initial expectations the experimented techniques cause disturbance and delay in the agents' learning process, with the possible exception of Changing-Advice-Modes in QL-agents. The conclusions we can draw from these results are that the processes used to change the type of advice and the learning parameters were inadequate. We have experimented with different conditions for triggering the changes in

Table 7.15: Summary of the final results for Experiment 5.

Ref.	Advisor Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
IM2H0-GP	No Advice	0.067	0.007	178
IM2H0-QL	No Advice	0.016	0.010	178
SE2H1.1-EA	Off. Batch	0.082	0.012	582
SM2H0.2-GP	Off. Batch Mult.	0.072	0.008	630
SM2H0.3-QL	OffB MVExp.+ Roles	0.084	0.002	506
SM2H0.5-EA	ChgAdvModes	0.070	0.016	370
SM2H0.5-GP	ChgAdvModes	0.067	0.011	370
SM2H0.5-QL	ChgAdvModes	0.080	0.005	370
SM2H0.5-EA	ChgLearnParams	0.060	0.005	404
SM2H0.5-GP	ChgLearnParams	0.059	0.004	404
SM2H0.5-QL	ChgLearnParams	0.075	0.007	404
SM2H0.5-EA	ChgModesAndParams	0.060	0.004	448
SM2H0.5-GP	ChgModesAndParams	0.059	0.005	448
SM2H0.5-QL	ChgModesAndParams	0.075	0.007	448

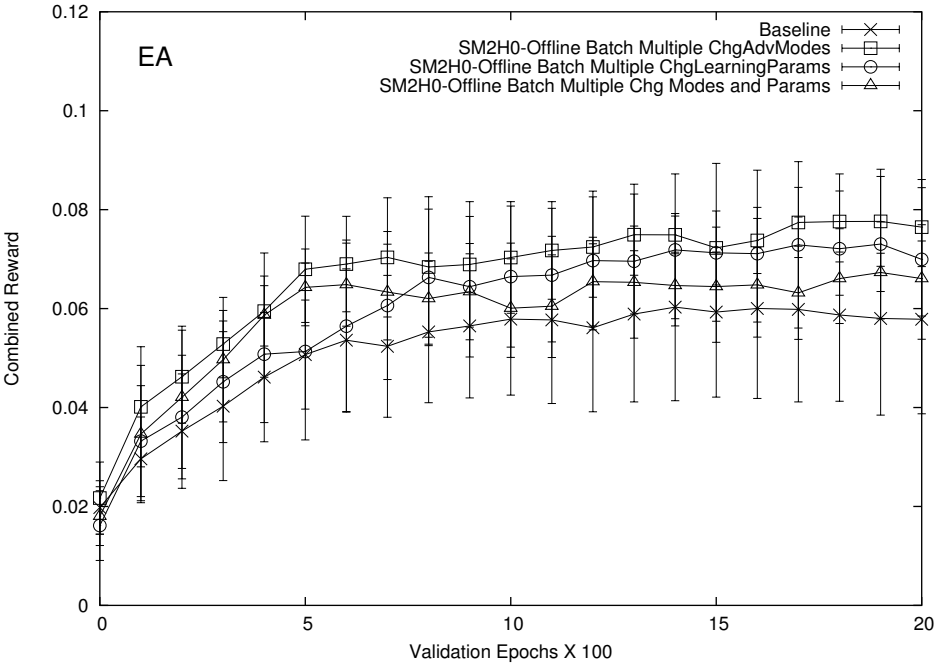

Figure 7.17: Evolution of the average combined-reward for EA-agents in Experiment Set 5.

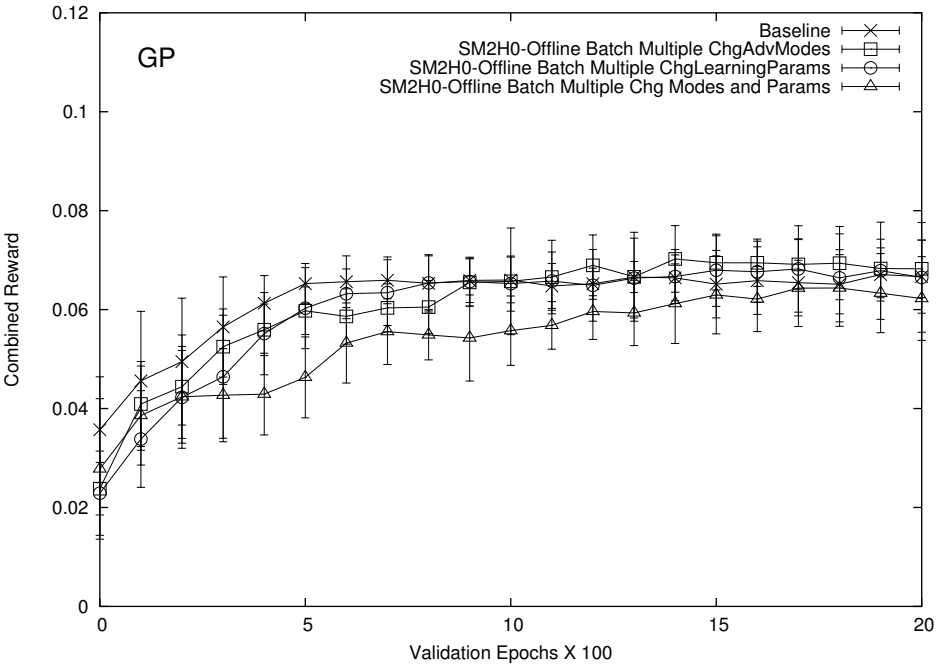

Figure 7.18: Evolution of the average combined-reward for GP-agents in Experiment Set 5.

advice modes and learning parameters, all with similar results. In all these cases we used a fixed set of conditions and applied it to all types of agents, adapting only at the lowest possible level. For example, when the performance was above a certain threshold all agents should increase their exploration. This was done in QL-agents by raising the temperature and in EA and GP-agents by increasing the population and/or raising the mutation-rates. It is possible that there is no set of rules which can be applied to all learning algorithms, or that we have not found the correct rules, despite the arguments put forth in chapter 4 that support our reasoning. One interesting possibility would be to let agents decide, but the time necessary to test the several combinations may be prohibitive, unless this can be done in parallel.

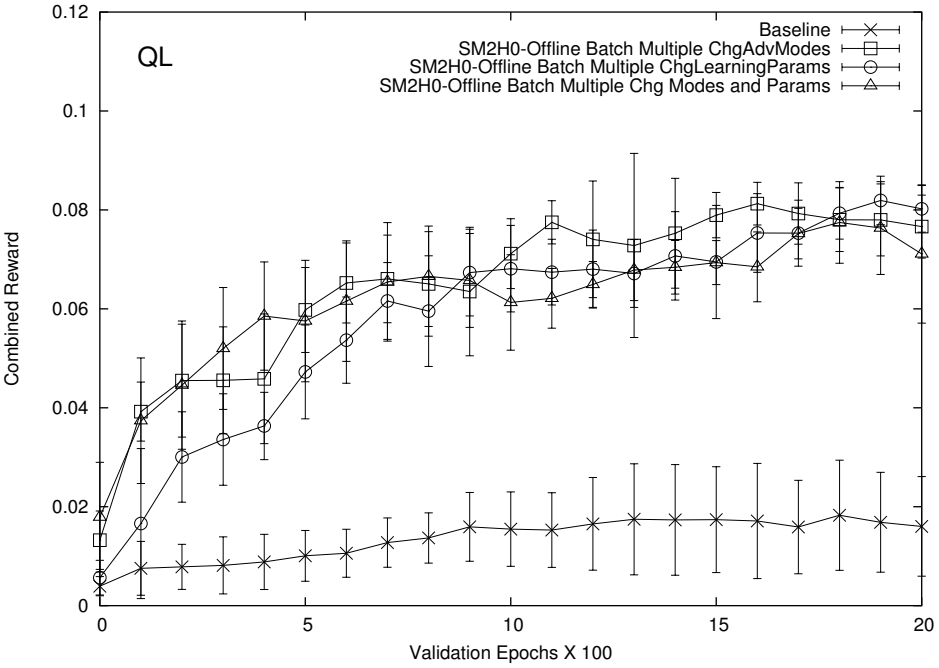

Figure 7.19: Evolution of the average combined-reward for QL-agents in Experiment Set 5.

7.7 Experiment Set 6: Combining Roles, Trust and Adaptation of Learning Parameters

This experiment tests the combined use of: heuristic advisors, changing advice modes, changing learning parameters, roles and trust. The number of advisors and type of advice in experiments SM2H0.6 is decided online according to the agents' learning-stage, as in the previous experiment.

Results

Results of the baseline tests and best results are included in the tables for comparison.

Table 7.16: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in Experiment Set 6. All values in average number of examples per agent × epoch. Example-size is 144 bytes.

	EA	GP	QL	Heu	Type
EA	26.71	0	0	15.82	SE2H1
GP	0	8.25	0	4.67	SG2H1
QL	0	0	23.00	12.55	SQ2H1
EA	3.60	23.78	25.60	24.92	SM2H1
GP	10.02	2.40	11.85	12.06	SM2H1
QL	2.28	3.51	.60	4.02	SM2H1

Observations

Performance As in the previous experiment QL-agents are the exception, showing relatively good performances, even though they have higher standard deviations than the previous best scores (Experiment Set 3).

QL-agents The introduction of H-agents as a good effect on QL-agents' performances.

GP-agents GP-agents show a fast initial climb.

EA-agents EA-agents appear to become trapped in a local *optimum* with a performance similar to GP and H-agents, although they improve over the previous experiments' results.

Run-times The run-times for EA-agents are greater than for most other cases, which causes a delay in heterogeneous (SM) tests.

It is interesting to notice the effects of introducing H-agents at this level. On QL-agents the simple introduction of an agent with a good initial performance (although

Table 7.17: Summary of the final results for Experiment 6.

Ref.	Advisor Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.058	0.019	178
IM2H0-GP	No Advice	0.067	0.007	178
IM2H0-QL	No Advice	0.016	0.010	178
SE2H1.1-EA	Off. Batch	0.082	0.012	582
SM2H0.2-GP	Off. Batch Mult.	0.072	0.008	630
SM2H0.3-QL	OffB MVExp.+ Roles	0.084	0.002	506
SE2H1.6-EA	ChgModesAndParams	0.067	0.010	841
SG2H1.6-GP	ChgModesAndParams	0.066	0.006	257
SQ2H1.6-QL	ChgModesAndParams	0.082	0.010	222
SM2H1.6-EA	ChgModesAndParams	0.065	0.014	701
SM2H1.6-GP	ChgModesAndParams	0.071	0.006	701
SM2H1.6-QL	ChgModesAndParams	0.083	0.009	701

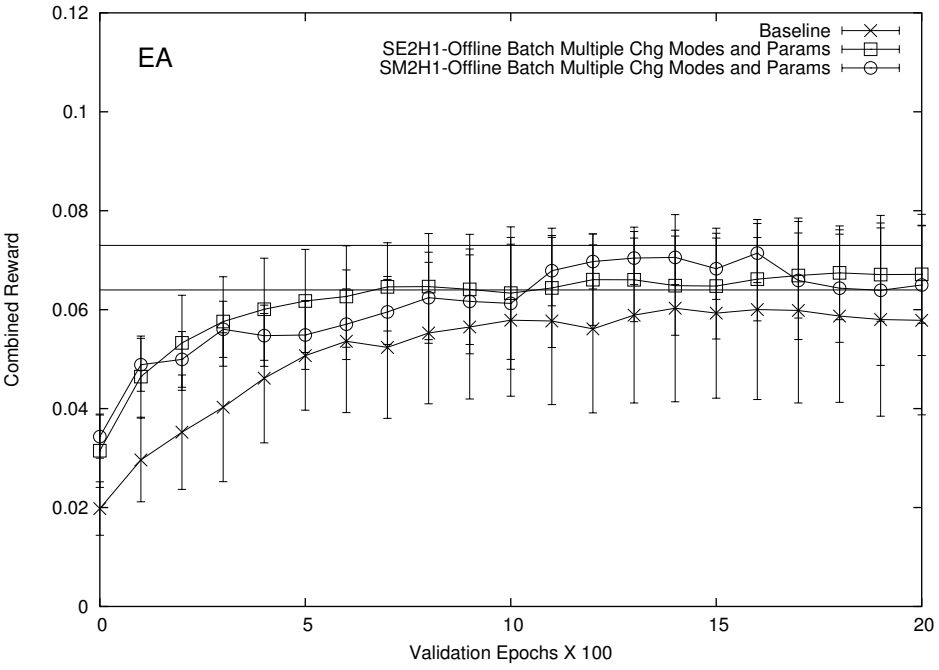

Figure 7.20: Summary of the final results for EA-agents in Experiment Set 6.

suboptimal) and a coherent strategy, marks the difference between the relatively low performance shown in the previous experiment (SM2H0.5-QL-ChgModesAndParams) to a relatively high performance achieved in this experiment set. We have noticed in several experiments that QL-agents appear to be dependent on the appearance of a peer

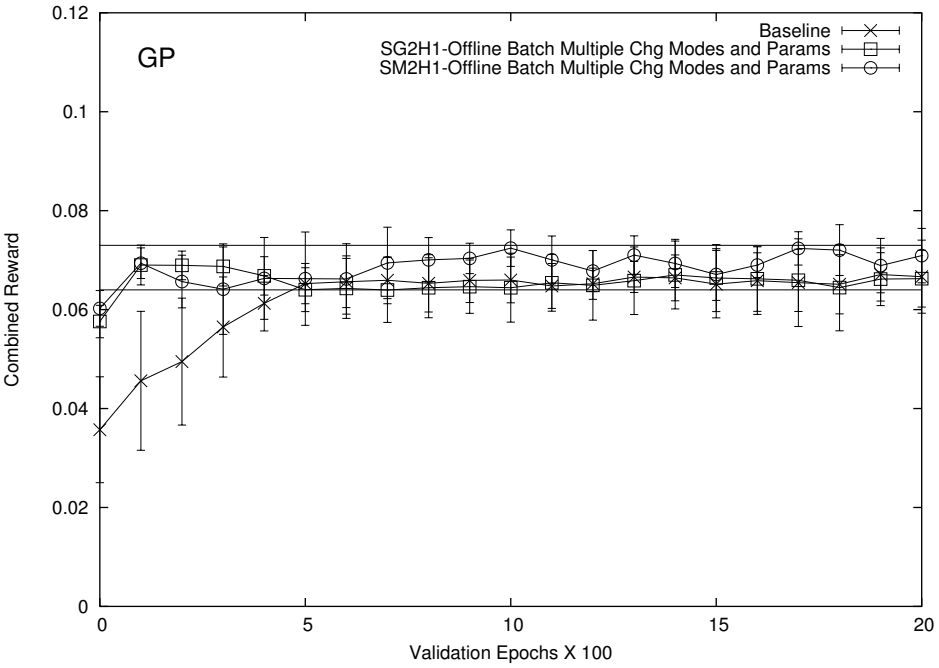

Figure 7.21: Summary of the final results for GP-agents in Experiment Set 6.

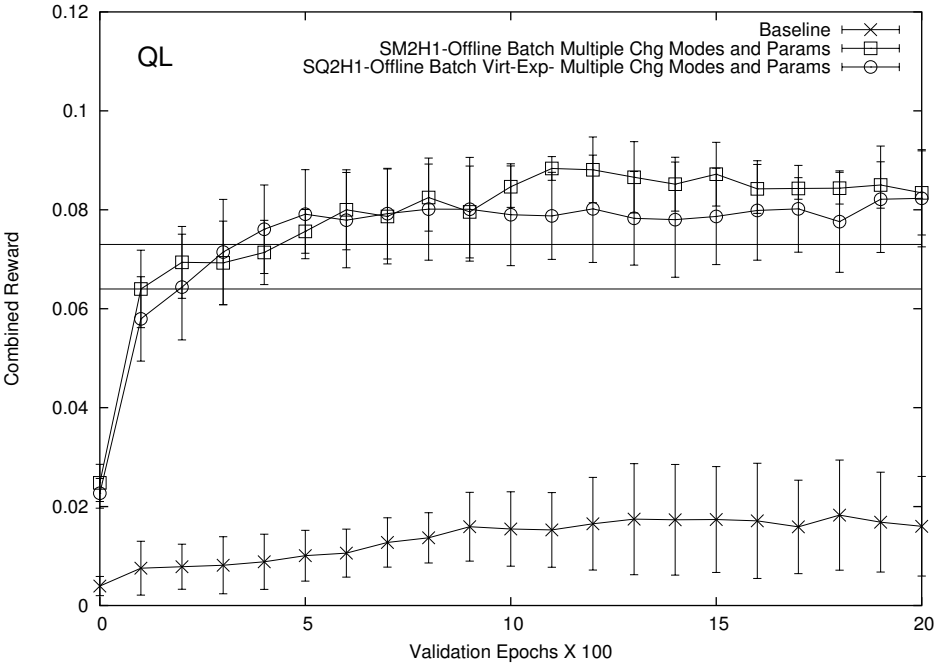

Figure 7.22: Summary of the final results for QL-agents in Experiment Set 6.

with a reasonable solution early in the training. When this happens, QL-agents can rise quickly to good solutions and, having more time to explore, reach the best scores in the tests.

In EA-agents, the introduction of H-agents causes an increase over the very low performances observed in (SM2H0.5-EA-ChgModesAndParams), although still far from the best results achieved.

For GP-agents the results are very low for (SM2H0.5-GP-ChgModesAndParams), similar to baseline in the homogeneous scenario (SG2H1.6-GP-ChgModesAndParams) and very close to the best in SM2H1.6-GP-ChgModesAndParams.

7.8 Traffic Baseline

In the baseline experiments for the traffic-control problem each of the 6 trials runs 3 locations, each with a team of 4 agents. In each area, agents use different learning algorithms. No communication is allowed between the agents. The objective of this experiment is to set a reference performance for each type of agent in the traffic-control problem.

Results

Table 7.18: Summary of the final results for the traffic-control problem in baseline trials.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM4H0-EA	No Advice	0.607	0.030	297
IM4H0-GP	No Advice	0.573	0.027	297
IM4H0-QL	No Advice	0.615	0.030	297

Observations

Performance QL-agents reach a near-optimal performance in the first few epochs (less than 180), and no learning takes place after that period.

Emergent behavior The emergence of green-wave patterns and synchronization is noticeable, contrary to previous experiments with traffic-control with different decision-times, flow-multipliers and topology (Nunes and Oliveira, 2004).

GP-agents GP-agents are, in this case, the ones with the lowest performances in the baseline trials. It is interesting to see that H-agents also have a relatively low performance in traffic scenarios (Experiment Set 8), which leads us to the conclusion that the background knowledge used in this case has a low potential.

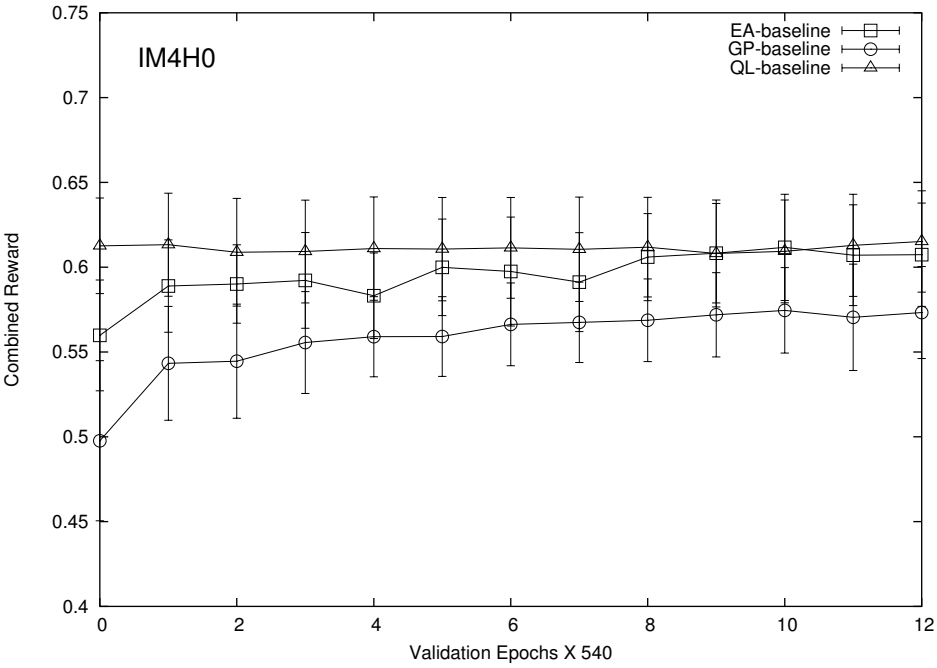

Figure 7.23: Average evolution of the combined reward in the baseline experiment (IM4H0) for the Traffic-control problem.

7.9 Experiment Set 7: Homogeneous and Heterogeneous Advisors

The objectives of this experiment are: to verify the effects of communication and the use of Roles in the traffic environment; to compare the results obtained in homogeneous and heterogeneous environments in this problem.

The conditions for this experiment are the same as for Experiment Set 4, using Roles.

Observations

Performance QL and GP-agents show a small improvement in their final performances, with average results near the upper limit of the baseline standard deviation.

GP-agents GP-agents climb faster to a reasonable reward in Multiple advisors scenarios although final results are very similar to situations with no communication. This indicates, once more, that agents with low performance benefit from communication with expert peers.

QL-agents Contrary to the baseline experiment QL-agents continue learning after the initial trials, although slowly. It is interesting to notice that QL-agents ask for their peers advice, although in a small scale.

Results

Table 7.19: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in each epoch in Experiment Set 7. Example-size is 64 bytes.

	EA	GP	QL	Type
EA	301	0	0	SE4H0
GP	0	204	0	SG4H0
QL	0	0	156	SQ4H0
EA	0	164	204	SM4H0
GP	302	0	310	SM4H0
QL	22	21	0	SM4H0

Table 7.20: Summary of the final results for the traffic-control problem in Experiment Set 7.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM4H0-EA	No Advice	0.607	0.030	297
IM4H0-GP	No Advice	0.573	0.027	297
IM4H0-QL	No Advice	0.615	0.030	297
SE4H0.7	OffB Mult. + Roles	0.587	0.035	308
SG4H0.7	OffB Mult. + Roles	0.591	0.033	538
SQ4H0.7	OffB MVExp + Roles	0.622	0.024	664
SM4H0.7-EA	OffB Mult. + Roles	0.600	0.027	669
SM4H0.7-GP	OffB Mult. + Roles	0.580	0.041	669
SM4H0.7-QL	OffB MVExp + Roles	0.636	0.022	669

7.10 Experiment Set 8: Learning Stages and Adaptation of Learning Parameters

The objectives of this experiment are: to verify the effects of introducing learning stages (along with changing advice modes and parameters) in the traffic environment; to compare the results obtained in homogeneous and heterogeneous environments in this problem.

The conditions for this experiment are the same as for Experiment Set 6. All agents in the Novice stage will request batch-advice to a standing advisor with the same role. All agents in Intermediate stage will request batch advice to multiple advisors. Ex-

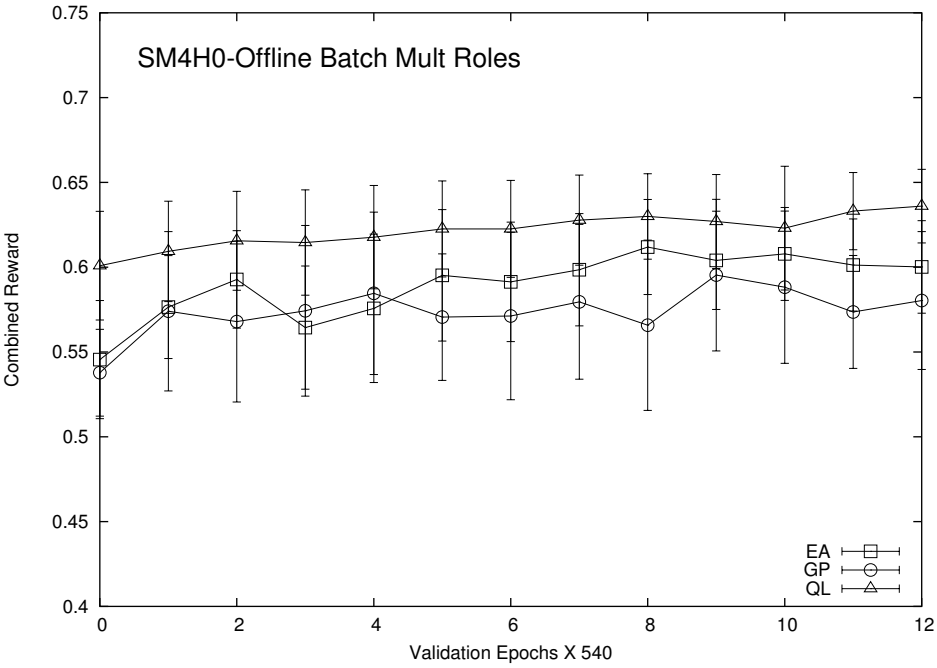

Figure 7.24: Average evolution of the combined reward in Experiment Set 7 for the Traffic-control problem.

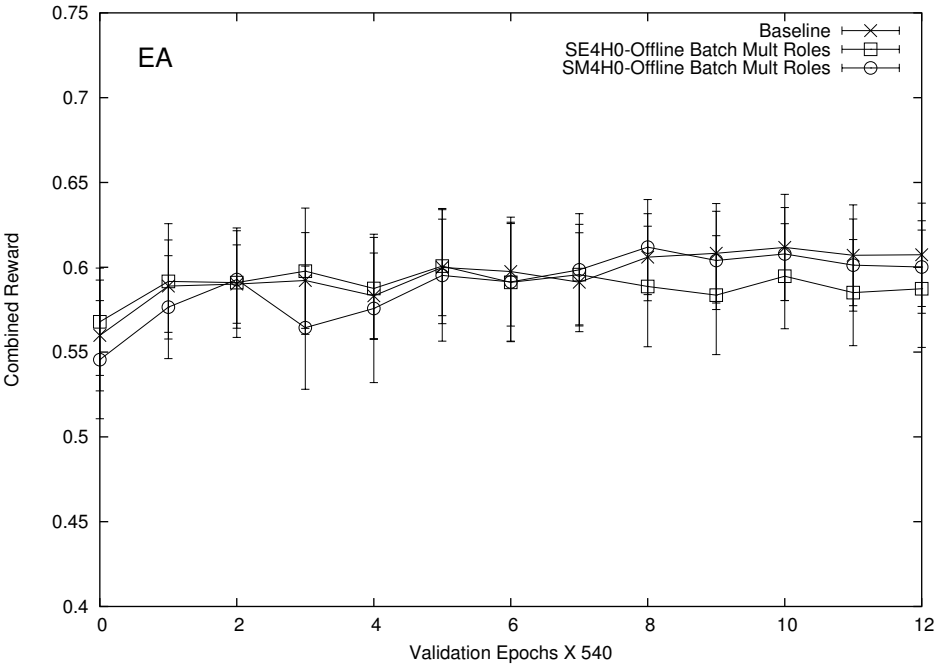

Figure 7.25: Summary of the final results for EA-agents in Experiment Set 7.

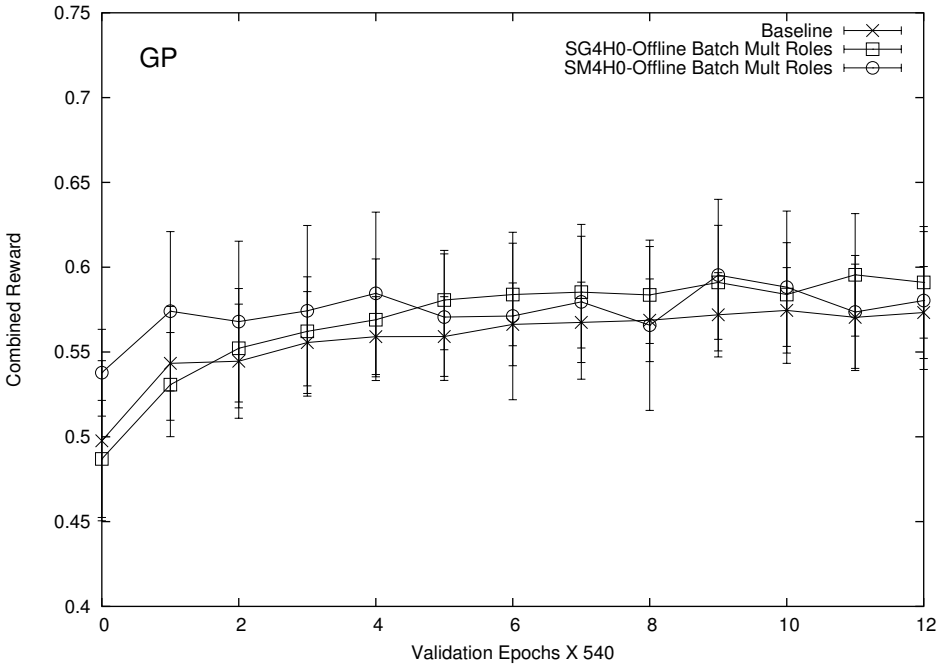

Figure 7.26: Summary of the final results for GP-agents in Experiment Set 7.

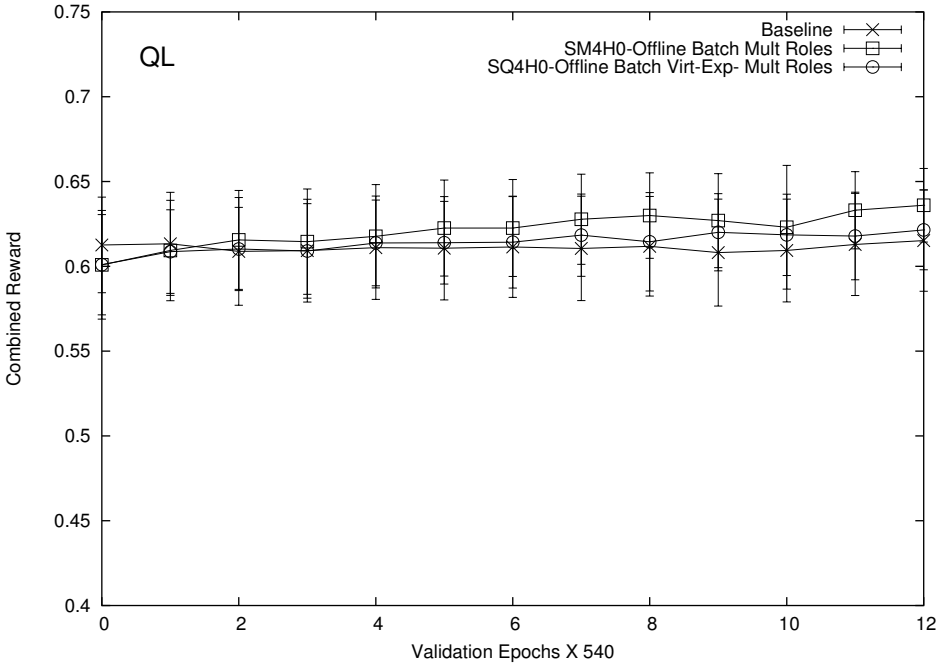

Figure 7.27: Summary of the final results for QL-agents in Experiment Set 7.

pert agents do not request advice. Stage transitions occur in the conditions defined in equations 5.11 through 5.14.

Results

Results of the baseline trials are included in table 7.21 for comparison. The acronym (ChgMAndP stands for Changing advice Modes And learning Parameters).

Table 7.21: Summary of the final results for the traffic-control problem in Experiment Set 8.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM4H0-EA	No Advice	0.607	0.030	297
IM4H0-GP	No Advice	0.573	0.027	297
IM4H0-QL	No Advice	0.615	0.030	297
SE4H1.8	ChgMAndP + Roles + Trust	0.608	0.030	367
SG4H1.8	ChgMAndP + Roles + Trust	0.579	0.026	353
SQ4H1.8	ChgMAndP + Roles + Trust	0.596	0.033	513
SM4H1.8-EA	ChgMAndP + Roles + Trust	0.612	0.020	431
SM4H1.8-GP	ChgMAndP + Roles + Trust	0.599	0.036	431
SM4H1.8-QL	ChgMAndP + Roles + Trust	0.581	0.021	431
SM4H1.8-HEU	-	0.205	0.012	431

Table 7.22: Average number of bytes exchanged between an advisee of a given type (lines) and an advisor (columns) in each epoch in Experiment Set 8. Example-size is 64 bytes.

	EA	GP	QL	HEU	Type
EA	57.55	0	0	0.8	SE4H0
GP	0	21.58	0	0.2	SG4H0
QL	0	0	29.10	0.2	SQ4H0
EA	0	0	0	0	SM4H0
GP	1.15	0	1.15	0.2	SM4H0
QL	0	0	0	0	SM4H0

Observations

Performance No significant differences in final results when compared to previous experiments, with the exception of a slight reduction of QL-agents' performance, not significant given the standard deviations.

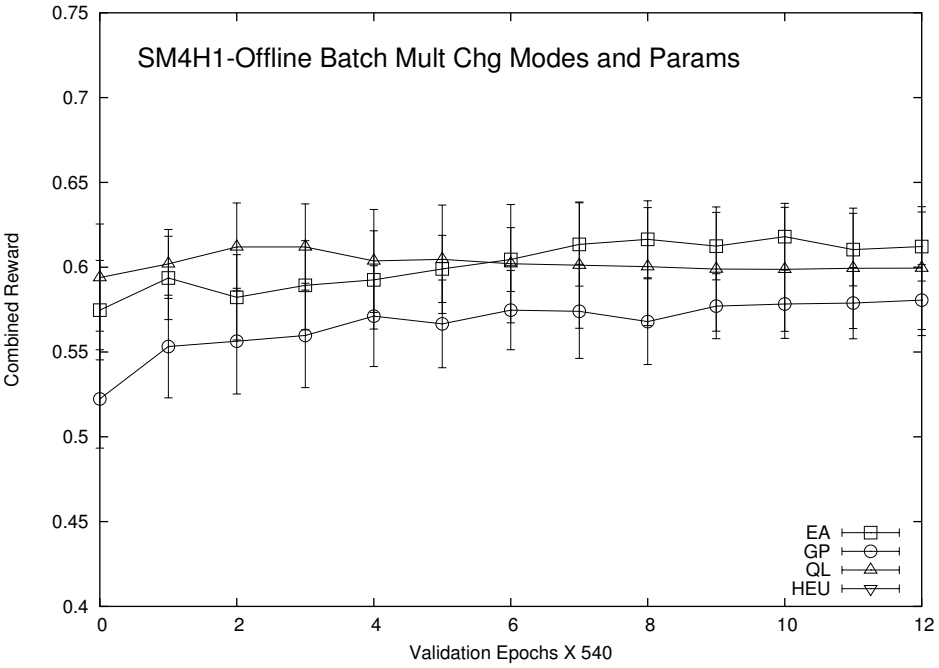

Figure 7.28: Average evolution of the combined reward in Experiment Set 8 for the Traffic-control problem.

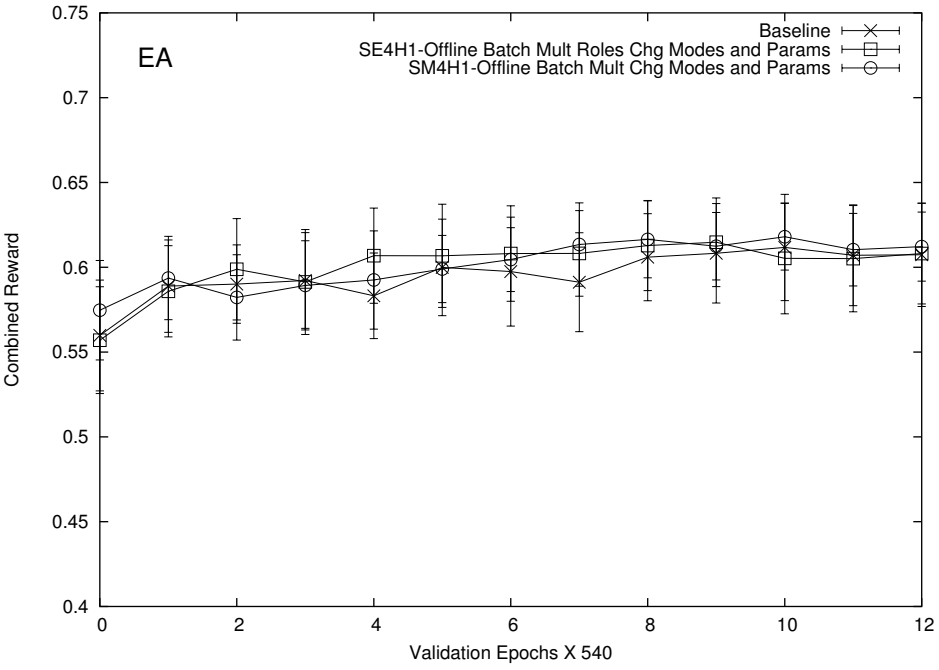

Figure 7.29: Summary of the final results for EA-agents in Experiment Set 8.

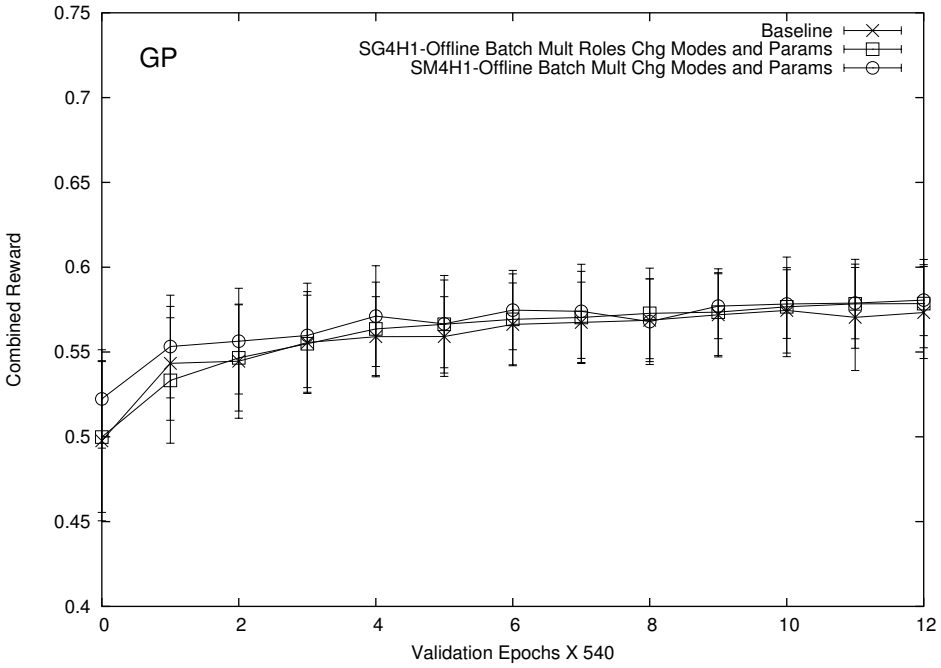

Figure 7.30: Summary of the final results for GP-agents in Experiment Set 8.

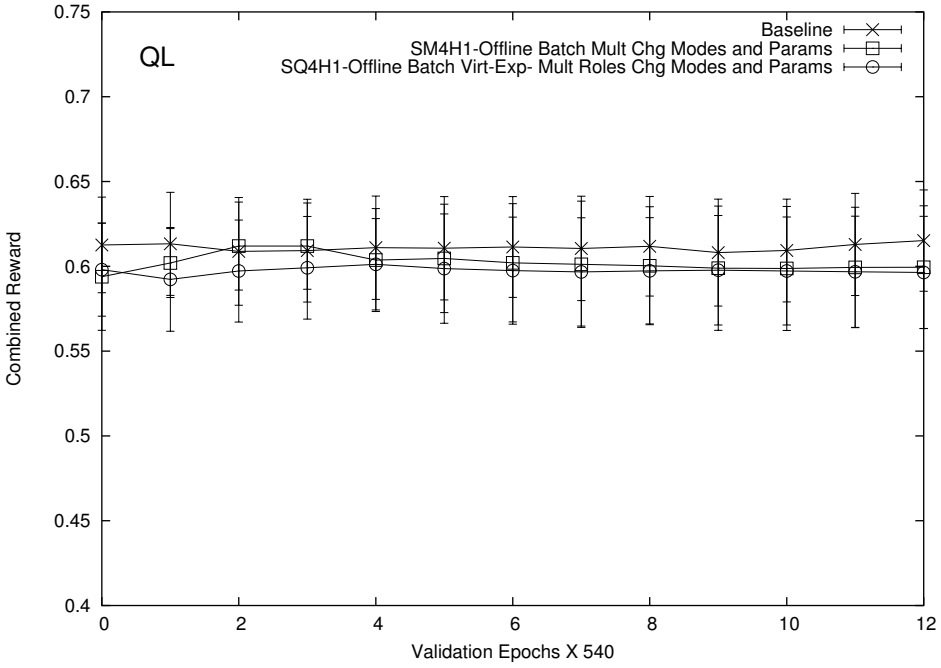

Figure 7.31: Summary of the final results for QL-agents in Experiment Set 8.

Communication Very low communication in heterogeneous scenarios (negligible for EA-agents and QL-agents) due to the very similar performance of all agents during the trials, contrary to previous experiment where QL-agents' better performance caused other agents to engage in communication.

H-agents H-agents' results do not appear in the figures because they are relatively low in comparison to the results presented, as shown in table 7.21.

As in Experiment Set 6, the results of using the full set of techniques are not the best results achieved. The fact that no significant differences were achieved from the baseline results (with the exception of SM4H0.7-QL) leads us to conclude that the problem is easily solved by all types of learning agents with very similar results with or without communication. Still the fact that the best performance (along with one of the lowest standard deviations) occurred in a trial with communication must be taken into account.

7.11 Load-Balance Baseline

In the baseline experiments for the traffic-control problem each of the 6 trials runs 3 locations, each with a team of 2 agents. In each area, agents use different learning algorithms. No communication is allowed between the agents. The objective of this experiment is to set a reference performance for each type of agent in the load-balance problem. No communication is allowed between the agents.

Results

Table 7.23: Summary of the final results for the load-balance problems in baseline trials.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.912	0.010	319
IM2H0-GP	No Advice	0.898	0.013	319
IM2H0-QL	No Advice	0.910	0.009	319

Observations

Performance (EA and QL) EA and QL-agents show better performance and relatively low standard deviations. QL climbs faster to a good performance.

Performance (GP) GP-agents' performance is slightly below the lower bound of other agents' results.

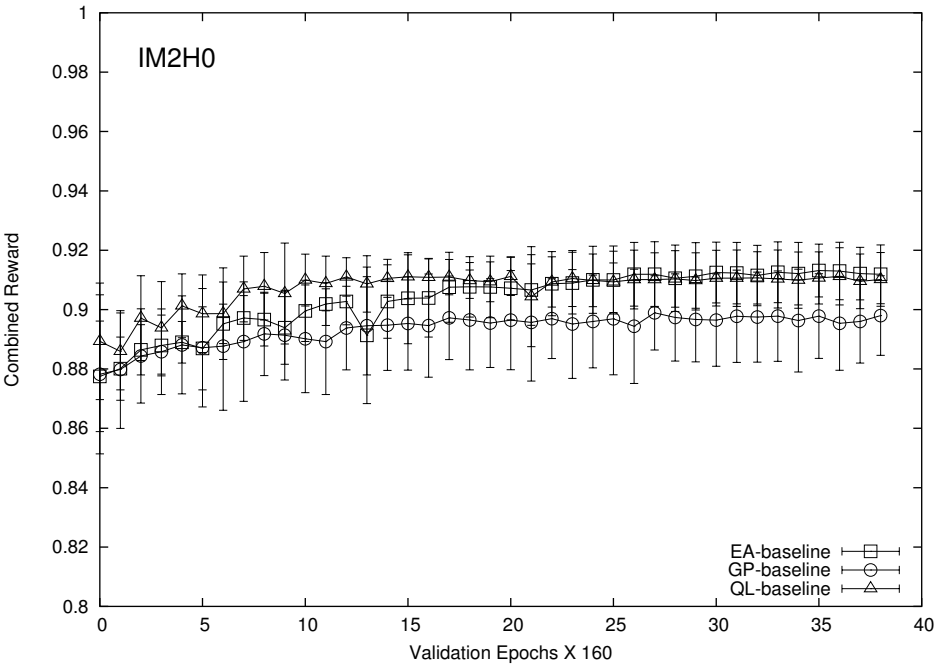

Figure 7.32: Average evolution of the combined reward in the baseline experiment (IM2H0) for the load-balancing problem.

7.12 Experiment Set 9: Learning versus Advice

The objectives of this experiment are: to verify the effects of communication in the same circumstances as Experiment Set 8, except for the use of Trust and the comparison between heterogeneous and homogeneous environments.

Results

Results of the baseline trials are included in table 7.24 for comparison.

Observations

Performance (EA) There is a slight decay in EA-agents' final results, although not significant.

Performance (GP and QL) GP and QL agents show a better initial climb to their best results. There is a slight improvement in performance for QL-agents, although within the baselines' standard-deviation interval.

Standard Deviations All results present lower standard deviations than baseline trials

Table 7.24: Summary of the final results for the load-balance problems in Experiment Set 9.

Ref.	Advice Type	Avg.	Std. Dev.	Time(m)
IM2H0-EA	No Advice	0.912	0.010	319
IM2H0-GP	No Advice	0.898	0.013	319
IM2H0-QL	No Advice	0.910	0.009	319
SM2H0.9-EA	ChgMAndP + Roles	0.909	0.004	615
SM2H0.9-GP	ChgMAndP + Roles	0.900	0.011	615
SM2H0.9-QL	ChgMAndP + Roles	0.915	0.006	615
HEU	-	0.867	0.001	-

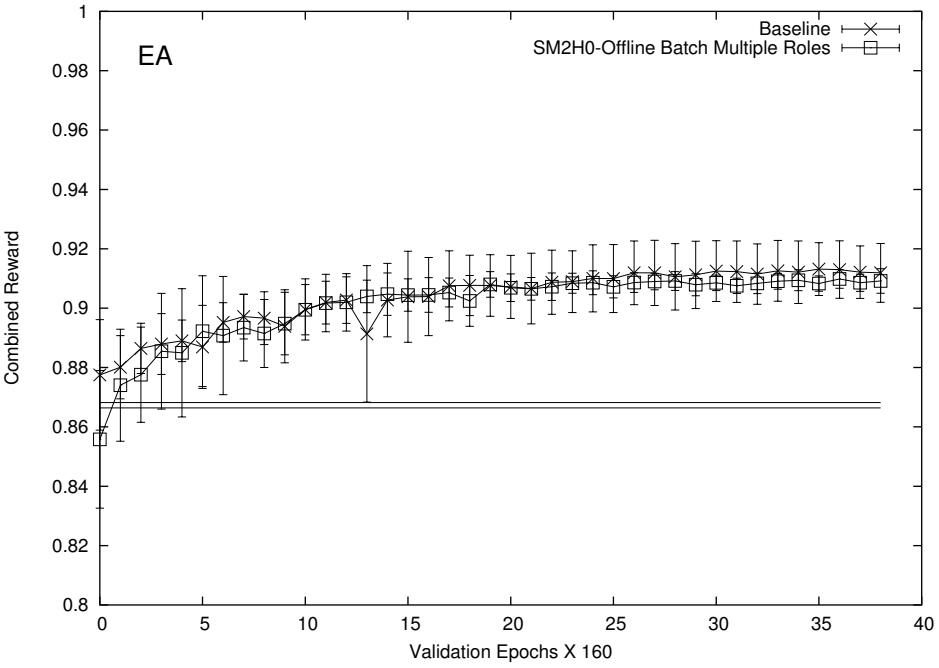

Figure 7.33: Summary of the final results for EA-agents in Experiment Set 9.

As in previous experiments, where the problem could be solved with relative ease by learning agents, even without communication, we can detect only small improvements, lower standard deviations and better initial results. The fact that the agents that achieve these improvements are again QL-agents, leads us to conclude that in all the problems studied QL-agents are the ones with more potential to improve when communication is available, even when the baseline results are below others'.

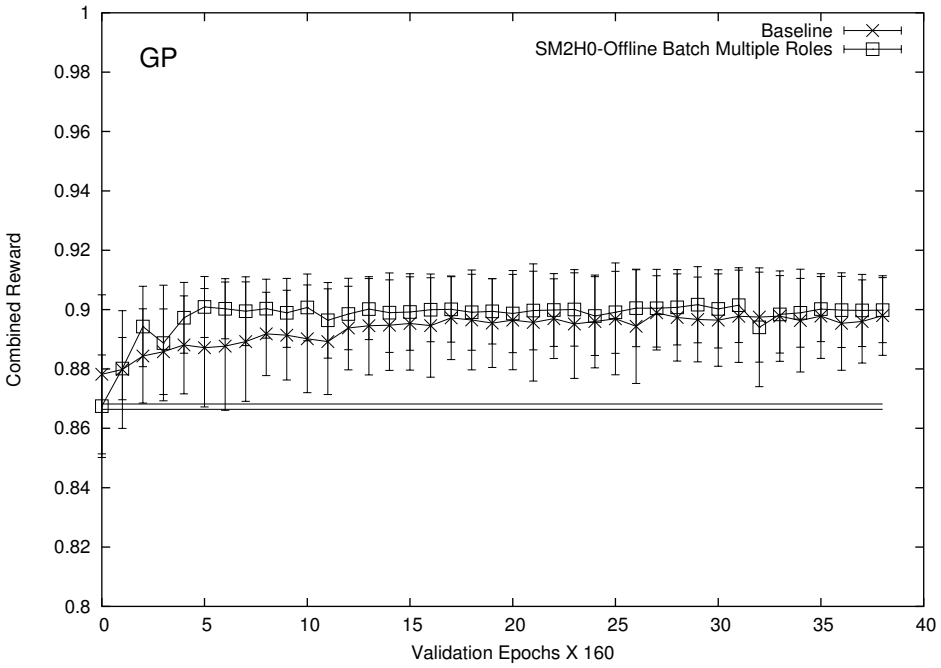

Figure 7.34: Summary of the final results for GP-agents in Experiment Set 9.

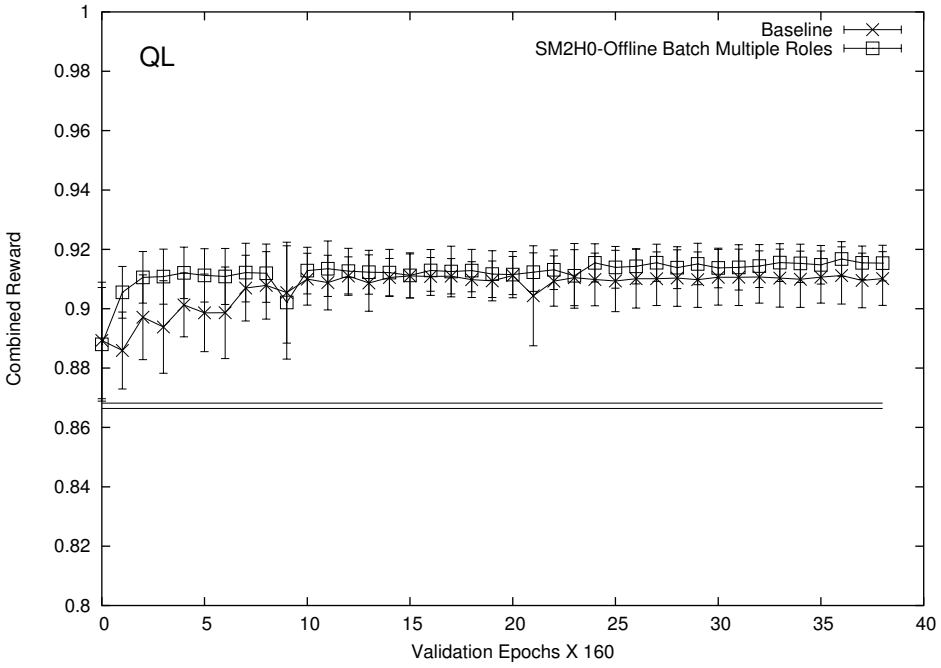

Figure 7.35: Summary of the final results for QL-agents in Experiment Set 9.

7.13 Comparison with results of other authors

Our observations support the conclusions of Tan and Lin, that were, at the time, applicable only to the exchange of information in QL-agents:

- *"sharing learned policies or episodes among agents speeds up learning at the cost of communication"* (Tan, 1993, in Abstract)
- *"advantages of teaching should become more significant as the learning task gets more difficult"* (Lin, 1992, section 6.4)

Our results are also coherent with the observations reported in (Berenji and Vengerov, 2000) for QL-agents: that, under certain conditions, N communicating agents can achieve better results in M steps than one agent will achieve with the same amount of experience ($N \times M$ steps) as shown by the analysis in sections 7.2 and 7.3. We have seen also that the same conclusion is true for EA-agents in the predator-prey problem.

The apparent contradictions with Clouse's results are discussed and explained in section A.1.8.

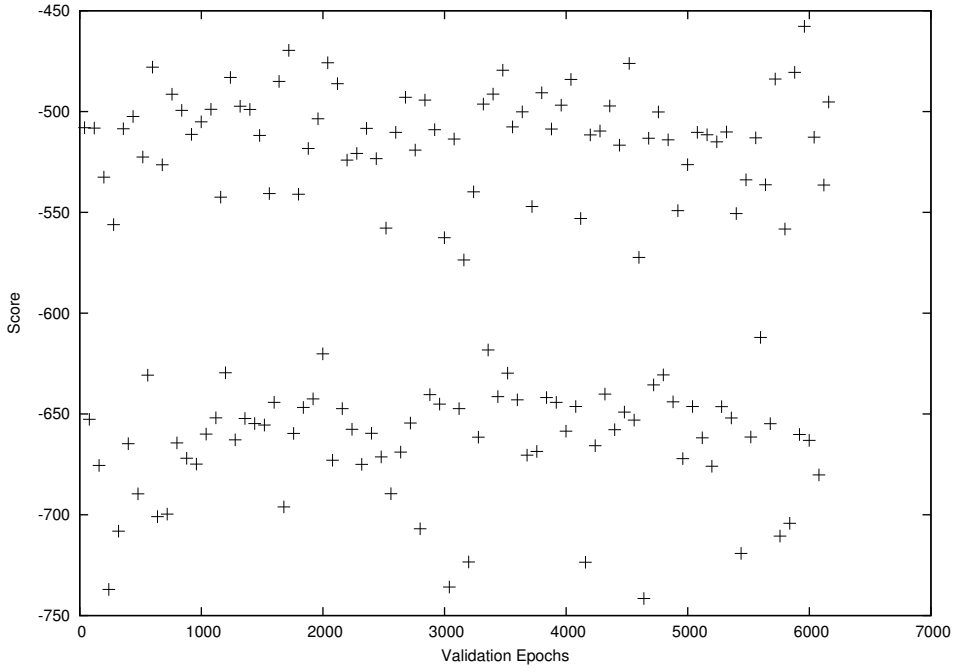

Figure 7.36: Score for H-agents (random-routing) in the load-balancing problem (Experiment Set 9).

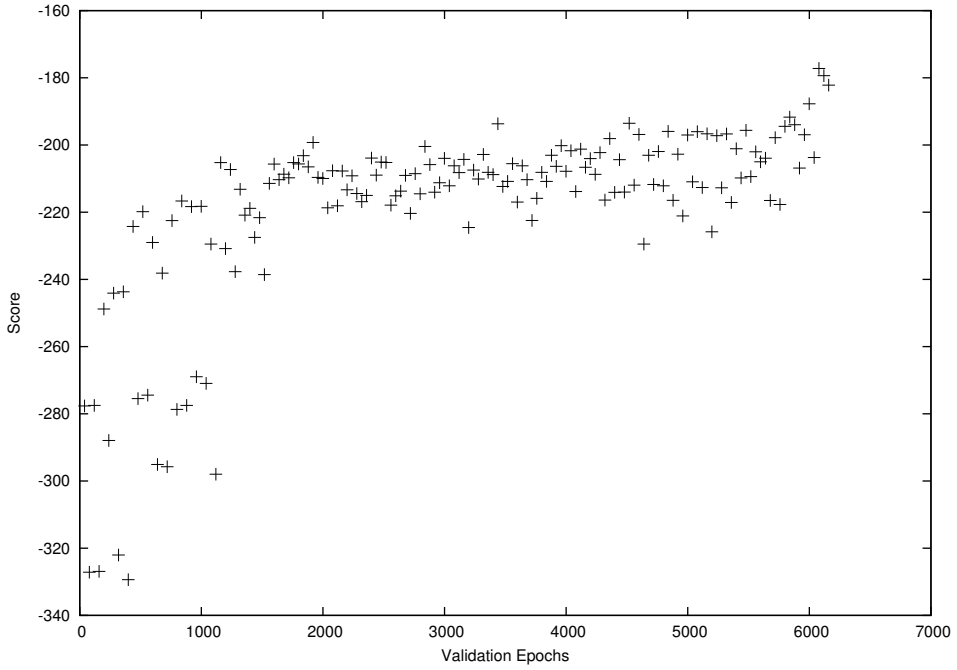

Figure 7.37: Score for QL-agents in the load-balancing problem (Experiment Set 9).

The results of choosing different learning modes (Dorigo and Colombetti, 1994b) and using different learning parameters depending on external information (Bowling and Veloso, 2001), were achieved for QL-agents and under quite different conditions from those we have tested. Nevertheless, our attempts to enhance the results achieved by exchanging advice using similar techniques were unsuccessful.

The results achieved by Tan and Haynes (Tan, 1993; Haynes *et al.*, 1995b) in the predator-prey domain are not comparable since these authors use only movement in 4 directions and different state representations. This movement makes it harder to catch a prey but also restricts the dimension of the search-space greatly, which makes the problem easier to solve. The state representations are inadequate for some learning algorithms. These difficulties prevented the direct comparison of results with these authors.

The results presented in (Whiteson and Stone, 2004) for FIFO schedulers range from -300 to -250 in the regular scenario and -400 to -350 under catastrophe conditions (presented in (Whiteson and Stone, 2004, figure 8.a)). These results represent a continuous moving average, during training, of the last 100 packets delivered by all load-balancers (equations 6.1 and 6.2). The baseline results (random routing) have a score between -750 to -650.

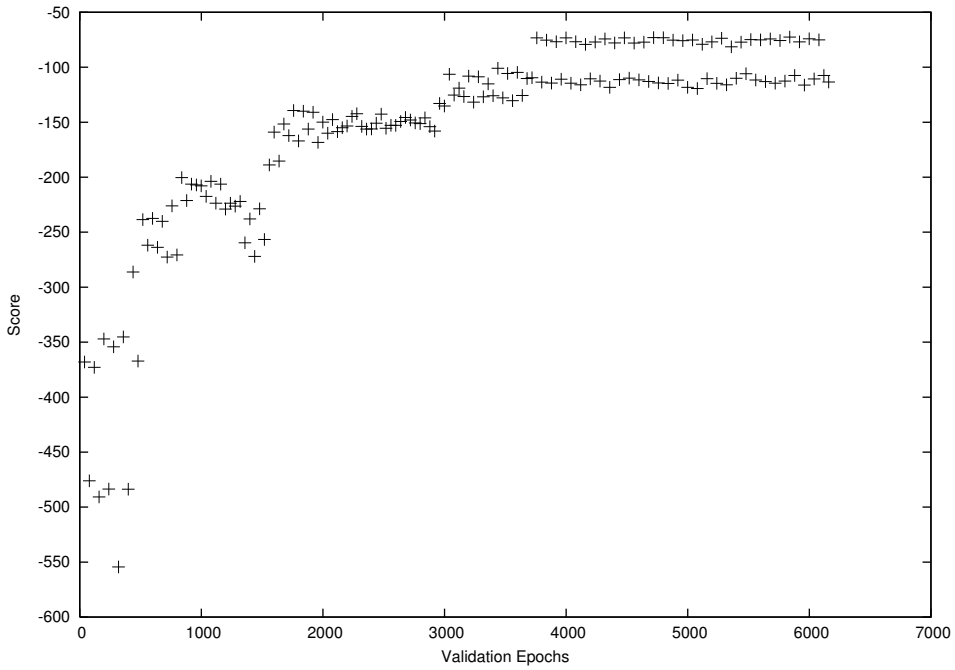

Figure 7.38: Score for EA-agents in the load-balancing problem (Experiment Set 9).

Due to our epoch-based evaluation and the fact that EA and GP specimens must be evaluated on equal conditions we cannot compare results during training. In our simulations the network is flushed (runs without any more packets being generated until it is empty) at end of each learning epoch and the load-balancers are evaluated for all packets they have routed in that epoch. Even though the network is not flushed during test or validation the effect of starting-fresh at the beginning of a new cycle produces an average score in validation mode of -515 for H-agents (i.e. random routing). This is still far from the value reported by Whiteson and Stone (2004), that we have verified in our own experiments (see Appendix D). Averaging our results over 40 epochs (4000 steps), instead of the full 80 epochs that compose a validation phase, we can clearly see a separation between the initial 40 epochs and the final 40 (figure 7.36)where the H-agents (random-routing) results are within the range of those reported by Whiteson and Stone (2004). Figures 7.37, 7.38, 7.39 represent the same measurements taken for QL, EA and GP-agents, respectively, and apart from Fig. 7.37 the difference between the first and the last 40 epochs is clearly visible.

Having this difference under consideration we can say that, using the same metrics for the last 8000 steps (80 epochs of 100 steps) of our final validation run (under catastrophe conditions and without flushing the network), we achieve results of -110 for EA-agents, -228 for QL-agents and -180 for GP-agents.

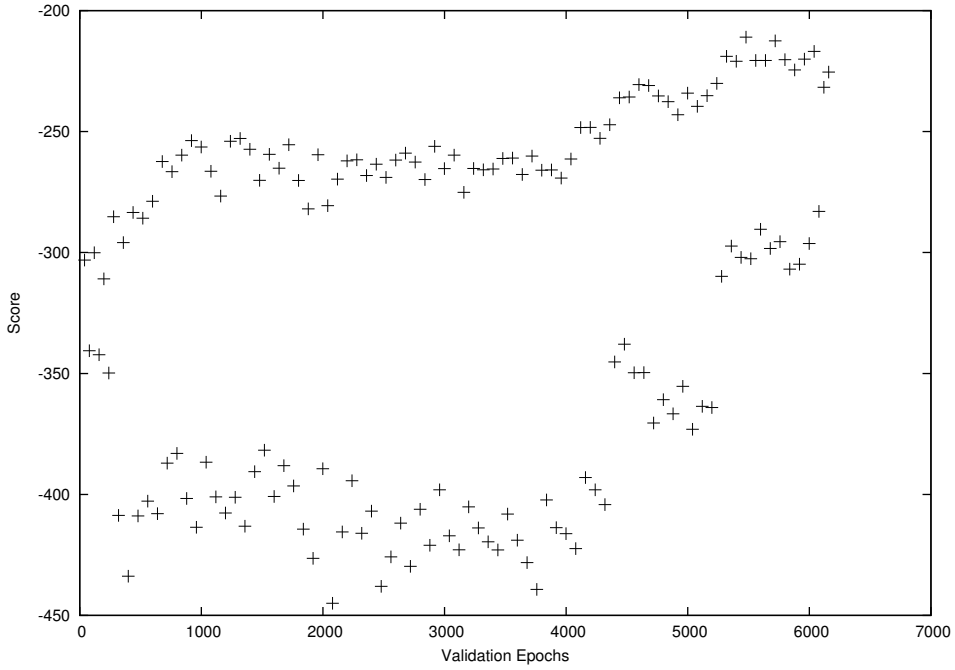

Figure 7.39: Score for GP-agents in the load-balancing problem (Experiment Set 9).

These results are not directly comparable, because ours were not achieved on a continuous run, the policies are static, but most of all due to the difference in the initial 40 epochs, caused by starting with an empty set of servers. However, if we analyze the lowest average scores on the final trials of figures 7.37, 7.38 and 7.39, we have results that are not influenced by initial conditions.

Even though some of the conditions that prevent direct comparison still hold, it seems clear that our approach is, at least, on a similar level as state-of-the-art results, with results that are approximately -300 for GP-agents, -180 for QL-agents and -100 for EA-agents, versus -400 to -350 for (Whiteson and Stone, 2004). It is also interesting to point out that in (Whiteson and Stone, 2004) there is learning both on servers and routers (load-balancers), while in our case only the latter have learning capabilities. We must say in all fairness that our approach uses much more information and learning-time than the ones presented in (Whiteson and Stone, 2004).

7.14 Summary of the discussion

In summary, our experiments point to the following conclusions:

- Communication can improve the performance, both in terms of final results as well as quicker climb to reasonable results in difficult problems.
- In several situations, exchanging advice improved agent's performance beyond the capabilities of the best agent when measured in baseline trials.
- Also, we have observed that communicating agents (EA and QL) using the same amount of information are able to achieve better performances.
- The performance improvements can be achieved even when using suboptimal advisors, i.e. the extra information seems to be more important than the expertise of the advisors.
- The improvements achieved with communication, and other related techniques experimented here, are marginal in problems where learning agents quickly find near optimal solutions. Occasionally, the interference caused by peers' advice can slow the learning procedure or lead to local *optima*.
- Heterogeneity and the use of multiple advisors can speed-up the training for learning agents with a low relative performance. However, there is no proof that these techniques can bring any further advantages in other situations.
- Changing the way in which advice is used, or the learning parameters, during training may lead to instability and did not produce better final results or more efficient learning. These conclusions are however restricted to the cases we have studied. Different techniques, related to these approaches, may prove to be worthy.
- The use of roles to restrict communication, narrow the search for advisors and lead all members of a given team to seek different advisors, produced some of the best results in the tests.
- The use of trust relationships between the advisors leads to relatively poor results. Persistence in getting advice from a certain advisor, even in face of bad initial results, seems to pay off when the main difference between the problems is in the partners' behaviors.

Chapter 8

Conclusions and Future Work

The question we proposed to address was: '**(How) can heterogeneous teams of learning agents benefit from exchanging information during the learning process?**". Or, more extensively: Can communication improve agents' learning performance for several learning algorithms in a specific type of problem? Is it possible to enhance the benefits of communication by: using hybrid algorithms to integrate information from different sources? using heterogeneous environments and/or an adequate selection of information sources?

The main hypothesis was clearly verified in the predator-prey problem. The use of communication can indeed be advantageous for QL and EA-agents engaging in communication. This is true even if advisors are suboptimal. Performances improve significantly just by exchanging information with a single advisor (the one with the best performance). We have found several situations where the whole performs better than the best of the parts. In the case of GP-agents the increase is not significant, mostly due to the structure of the hypothesis. The close tie to the background knowledge inserted in the external functions is a double-edged sword, on the one hand it provides quick convergence to reasonable results (the best in the baseline trials for predator-prey), on the other hand it is very difficult to improve that score because better solutions require much more complex Program-Trees and are likely to be less stable, even when they have better average payoff. This seems to be a case where the representational capabilities of the evaluation function limited the type of policies it can achieve, making it virtually impossible to learn beyond a certain threshold. These difficulties may be circumvented by other implementations of GP-agents, a different set of external functions, or its application to other problems.

In the traffic-control problem there are no significant changes in performance regardless of the method used. GP and QL-agents, in Experiment Set 7, have an average final performance that is close to the upper limit of the baseline results. In other two situations SE2H0.7 and SMH0.8-QL results are slightly lower than baseline. In Ex-

periment Set 7 GP-agents show faster convergence in heterogeneous scenarios. Previous results in versions with different parameters and topology (2 traffic lights) showed small improvements in performance when exchanging advice, but were also, for the most part, inconclusive. We have observed that changes in the reaction-time have a great impact on the performance. The difference in performance achieved by reducing the time between two decisions from 20s to 12 is relatively high.

In the load-balance problem the results show only an increase in the initial performance when using advice-exchange. Final results have no significant differences.

The use of multiple advisors versus a standing advisor has no significant differences in most situations. Considering the increase in communication it can be considered more efficient to use a standing advisor. We are led to the conclusion that communication, beyond a certain threshold, brings no further advantage. This threshold is related to the size and topology of the search-space.

Heterogeneous advisors are useful only when there is a large performance gap between two different types of learning-agents and the advisee is flexible enough to learn the advisors' policy. Considering the limitations to communication imposed by heterogeneity, specially the fact that internal parameters cannot be exchanged, we conclude that the utility of heterogeneous environments is limited.

In what regards Roles we conclude that, although very simple, it is a useful way to synchronize advice and allow the acquisition of team policies. In problems where the differences in the dynamics of each location are caused by other factors than the agents' behavior this technique may need to be complemented. A good matching between the problems may become essential. The definition of a language to describe problems, in which similar problems could be easily identified, could provide the agents with the necessary information to choose their advisors wisely.

Contrary to our initial expectations the use of trust did not improve the efficiency in communication, and, in some cases even resulted in loss of performance. The reason for this is that the initial phase of advice causes a disruption of the teams' behavior, leading to a temporary drop of performance until the advisees' partners adapt to the new behavior. During the initial phase of advice the advisee is often unable to maintain the trust in its advisor. When this happens before its partners have adapted, and a higher *equilibrium* is found, the system will either return to a previous *equilibrium* or continue a run of low-performance epochs using non-matching advisors. So, in summary, the use of trust should be avoided, unless a different implementation of this concept takes these issues into consideration, allowing an initial phase of instability and rewarding persistence. Still, the purpose of quickly discovering the usefulness of a given information source will not be achieved if this tolerance for initial bad results is allowed.

Another set of concepts that did not produce the expected results was the one including learning-stages, dynamic advice-modes and changing learning parameters.

The objective of applying these techniques was to give a learning agent a greater capability to change the way in which it learns. These concepts were attempted as an initial (and possibly naive) approach to a truly autonomous learning agent. Although this implementation has failed to produce interesting results in these problems we believe that, sooner or later, this will become a central issue for learning-agents and new solutions must be found that allow the learning agent to have more control over how it learns.

8.1 Summary of contributions

In this work we have provided additional evidence that exchanging information can improve the performance of QL-agents and made a comparative study of several advice modes, concluding that Virtual Experience and Imitation are the methods that provide faster performance increases. The need to synchronize agents imposed by Imitation leads us to chose Virtual Experience as the best method for integrating external information in QL-agents.

We have proved that EA can improve their performances when communicating, and that GP-agents are often limited by the structure of the background knowledge introduced by the user. In this case only minor gains can be attained in the initial phases of training.

We have created and tested an architecture for teams of learning agents, in which the possibility of exchanging information to improve learning performances is taken into consideration.

We have concluded that the effects of heterogeneity are limited to specific situations where one type of learning agents has very low relative performance. Even in these cases the final results may be similar to homogeneous environments.

We have shown a successful way to use static role-assignment, which proved to be a simple and adequate way to synchronize information exchange.

The use of Trust was, in general, unsuccessful in limiting communication and making a more rational use of information because a certain degree of persistence in face of initial bad results is a necessary characteristic when learning from peers' advice in team scenarios.

One good sign is that advice-exchange proved to be more advantageous in difficult problems (in the limits of learnability). In these problems the best communicating-agents can achieve better final results and faster convergence to reasonable policies. In relatively easy problems the best that can be achieved is a faster convergence.

The results achieved in the traffic-control scenario, although they show no significant differences between learning approaches, provide additional evidence that learning systems can handle high traffic flows more successfully than fixed-light approaches.

It is also interesting to observe that the difficulty of the problem can be increased beyond the real traffic conditions and learning approaches can still provide a reasonable response. Of course the comparison with real traffic controllers, such as the one used in the avenues where data was collected, is not fair given the simplifications made in the simulation, but having nearly 20% more traffic than in the real-data and maintaining reasonably fluid traffic conditions is an interesting result that we will try to explore in future projects.

The emergence of synchronization in traffic-lights is another indication that adaptable approaches, using local information, can cooperate and work efficiently as a team.

The experiments with load-balance applications proved only that our approaches are comparable to the state-of-the-art. Due to the ease of resolution of this problem by standard learning algorithms no advantages could be observed in the use of communication.

This work studied several approaches to the use of communication in automated learning. This subject was, from the very beginning, deemed interesting by most of the anonymous reviewers that have judged our work throughout these years. Proof of this is the acceptance of all submitted papers in some of the most renowned refereed conferences in the area of MAS and ML (Nunes and Oliveira, 2002b, 2003a,b,c, 2004, 2005b). The interest in this work lead also to an invitation to write a book-chapter (Oliveira and Nunes, 2004), and a nomination for best student paper in one of the conferences where our work was presented (Nunes and Oliveira, 2003c) (the paper was classified as second runner-up by the jury). It is also interesting to point out that in all conferences where selected papers were published in LNCS/LNAI proceedings our work was always chosen for these publications (Nunes and Oliveira, 2002a, 2003d, 2005a).

The majority of the above-mentioned papers contains results of experiments with variations of the problems and techniques presented here, showing that we have studied other approaches during this research. The approaches that emerged from this work are either included in the body of this thesis (the most successful or demonstrative ones) or mentioned in appendix A.1 for the benefit of future researchers.

8.2 Future Work

Undergoing work is mainly related to the application of learning agents to traffic-control. One project is already starting with the objective of building simulation tools that allow us to try different types of adaptive solutions at several levels in this problem. Another project (pending approval) is specifically dedicated to learning and adaptive solutions for traffic management. This project will allows us, in cooperation with other European researchers, to test the effectiveness of exchanging information in state-of-the-art traffic simulations.

Following the line of research presented here we plan to test the impact of including communication costs in the reward functions and experimenting with exchange of information between agents that are solving different problems with a common structure. This will enable us to determine how different can problems be before communication becomes useless.

The goal we aim at is a truly autonomous learning agent that can chose the best sources of information from a vast set, select the appropriate techniques to deal with a given problem, adapt the learning parameters online based on his own results and external information, and require as little as possible human assistance and background knowledge to deal with any given problem.

Machine Learning is changing its focus, from single-agents, static environments and user-controlled systems, to multi-agent, dynamic environments and autonomous entities. These new challenges cannot be faced with the old tools. This thesis is a step (however small it may be) in the creation and study of new tools to meet these challenges. Many paths are left open, nevertheless we hope this work can be informative to those that are already studying these issues, allowing them to know the results (both positive and negative) of treading this path, and inspiring to those that are looking for a subject of study in this area of research.

Bibliography

Atkeson, C. G. and Schaal, S. (1997). Learning tasks from a single demonstration. In *Proc. of the 1997, IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 1706–1712.

Banerjee, B., Mukherjee, R., and Sen, S. (2000). Learning mutual trust. In *Working Notes of AGENTS-00 Workshop on Deception,Fraud and Trust in Agent Societies*, pages 9–14.

Bazzan, A. (1997). *An Evolutionary Game-Theoretical Approach for Coordination of Traffic Signal Agents*. Ph.D. thesis, University of Karlsruhe, Technische Hochschule.

Benda, M., Jagannathan, V., and Dodhiawalla, R. (1985). On optimal cooperation of knowledge resources. Technical Report BCS G-2012-28, Boeing AI Center, Boeing Computer Services, Bellevue, WA.

Berenji, H. R. and Vengerov, D. (2000). Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In *Proc. of the Ninth IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE '00)*.

Biswas, A., Sen, S., and Debnath, S. (2000). Limiting deception in groups of social agents. *Applied Artificial Intelligence Journal*, **14**(8), 785–797. Special Issue on Deception, Fraud and Trust in Agent's Societies.

Bowling, M. and Veloso, M. (2001). Rational and convergent learning in stochastic games. In *Proc. of the Int. Joint conference on Artificial Intelligence*, pages 1021–1026. Seattle.

Boyan, J. A. and Littman, M. L. (1994). Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in Neural Information Processing Systems*, volume 6, pages 671–678.

Breiman, L. (1996). Stacked regressions. *Machine Learning*, **24**(1), 49–64.

Brockfeld, E., Barlovic, R., Shadshneider, A., and Shreckenberg, M. (2001). Optimizing traffic lights in a cellular automaton model for city traffic. *Physical Review*, **64**.

Chiang, I.-J. and Hsu, J. Y.-J. (2002). Fuzy classification trees for data analysis. *Fuzzy Sets and Systems*, **130**, 87–99.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence*, pages 746–752.

Clouse, J. and Utgoff, P. (1992). A teaching method for reinforcement learning. In *Proc. of the Ninth Int. Conf. on Machine Learning*, pages 92–101.

Clouse, J. A. (1997). *On integrating apprentice learning and reinforcement learning*. Ph.D. thesis, University of Massachusetts, Department of Computer Science.

Clouse, J. A. and Utgoff, P. E. (1991). Two kinds of training information for evaluation function learning. In *Proc. of AAAI'91*.

Dorigo, M. and Colombetti, M. (1994a). Robot shaping: Developing autonomous agents through learning. *Artificial Intelligence*, **71**(2), 321–370.

Dorigo, M. and Colombetti, M. (1994b). The role of the trainer in reinforcement learning. In *Proc. of MLC-COLT 94, Workshop on Robot Learning*.

Dresner, K. and Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. *Third Int. Joint Conf. on Autonomous Agents and Multiagent Systems*.

Fayyad, U. M. (1991). *On the induction of decision trees for multiple concept learning*. Ph.D. thesis, University of Michigan, EECS Department.

Fernandes, J. M. and Oliveira, E. (1999). Tramas: Traffic control through behaviour-based multi-agent system. *Proc. of Practical Applications of Intelligent Agents and Multi-agent Systems (PAAM'99)*, pages 457–458.

Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proc. of the Thirteen Int. Conf. on Machine Learning*, pages 148–156.

Fu, L. M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science*, **1**, 325–340.

Gama, J. and Brazdil, P. (2000). Cascade generalization. *Machine Learning*, **41**(3).

Glickman, M. and Sycara, K. (1999). Evolution of goal-directed behavior using limited information in a complex environment. In *Proc. of the Genetic and Evolutionary Computation Conf., (GECCO-99)*.

Goldman, C. and Rosenschein, J. (1995). Mutually supervised learning in multi-agent systems. In *Proc. of the IJCAI-95 Workshop on Adaptation and Learning in Multi-Agent Systems, Montreal, CA*.

Gordon, D. and Subramanian, D. (1993). A multi-strategy approach to assimilating advice in embedded agents. In *Proc. of the Second Int. Workshop on Multi-Strategy Learning*, pages 218–233.

Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation (2nd edition)*. Prentice-Hall.

Haynes, T., Wainwright, R., Sen, S., and Schoenfeld, D. (1995a). Strongly typed genetic programming in evolving cooperation strategies. In *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 271–278.

Haynes, T., Wainwright, R., Sen, S., and Schoenefeld, D. (1995b). Strongly typed genetic programming in evolving cooperation strategies. In *Proc. of the 6th Int. Conf. on Genetic Algorithms*, pages 271–278. Pittsburgh, Pennsylvania.

Hecht-Nielsen, R. (1987). Counterpropagation networks. *Applied Optics*, **26**, 4979–4984.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hu, J. and Wellman, M. P. (1998). Multi agent reinforcement learning: A theoretical framework and an algorithm. In *Proc. of the Fifteenth Int. Conf. on Machine Learning*, pages 242–250.

Ichihashi, H., Shirai, T., Nagasaka, K., and Miyoshi, T. (1996). Neuro-fuzzy id3: a method of inducing fuzzy decision trees with linear programming for maximizing entropy and an algebraic methos for incremental learning. *Fuzzy Sets and Systems*, **81**, 157–167.

Kantrowitz, M. (1997). Genetic algorithms - frequently asked questions (f.a.q.). Technical report, AI.Repository@cs.cmu.edu.

Kapetanakis, S. and Kudenko, D. (2002). Reinforcement learning of coordination in cooperative multi-agent systems. In *Proc. of the Eighteenth National Conf. on Artificial intelligence*, pages 326–331. Edmonton.

Kazakov, D. and Kudenko, D. (2001). Machine learning and inductive logic programming for multi-agent systems. In *Multi Agents Systems and Applications: Ninth ECCAI Advanced Course, Selected Tutorial Papers, LNAI 2086*, pages 246–271.

Koza, J. R. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge MA.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, **8**, 293–321.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proc. of the Eleventh Int. Conf. on Machine Learning*, pages 157–163.

Maclin, R. and Shavlik, J. (1996). Creating advice taking reinforcement learners. *Machine Learning*, **22**, 251–281.

MarsRover (2005). marsrover.jpl.nasa.gov/home.

Matarić, M. J. (1996). Using communication to reduce locality in distributed multi-agent learning. Computer Science Technical Report CS-96-190, Brandeis University.

Mitchell, T. (1997). *Machine Learning*. McGrawHill and MIT Press.

Montana, D. J. and Czerwinski, S. (1996). Evolving control laws for a network of traffic signals. In *Proc. of the First Annual Genetic Programming Conf.*

Moody, J. and Darken, C. (1988). Learning with localized receptive fields. In G. H. D. Touretzky and T. Sejnowski, editors, *Proc. of the 1988 Connectionist Models Summer School*, pages 133–143. Morgan Kaufmann.

Nagel, K. and Shreckenberg, M. (1992). A cellular automaton model for freeway traffic. *J. Phisique I*, **2**(12), 2221–2229.

Nicolescu, M. and Matarić, M. J. (2001). Learning and interacting in human-robot domains. *IEEE Transactions on systems, Man Cybernetics, special issue on Socially Intelligent Agents - The Human In The Loop*.

Nunes, L. (1997). *Inicialização e construção de redes locais e perceptrões multicamada*. Master's thesis, Instituto Superior Técnico.

Nunes, L. and Oliveira, E. (2002a). Cooperative learning using advice exchange. *Adaptive Agents and Multi-Agent Systems, LNCS/LNAI Hot Topics, 2636*.

Nunes, L. and Oliveira, E. (2002b). On learning by exchanging advice. In *Proc. of the Artificial Intelligence and the Simulation of Behaviour Convention, Second Symposium on Adaptive Agents and Multi-Agent Systems (AISB02/AAMAS-II)*. Imperial College, London.

Nunes, L. and Oliveira, E. (2003a). Advice-exchange amongst heterogeneous learning agents: Experiments in the pursuit domain. In *(poster abstract) Autonomous Agents and Multiagent Systems (AAMAS03)*. Melbourne, Australia.

Nunes, L. and Oliveira, E. (2003b). Advice-exchange between evolutionary algorithms and reinforcement learning agents: Experiments in the pursuit domain. In *Proc. of the Artificial Intelligence and the Simulation of Behaviour Convention, Third Symposium on Adaptive Agents and Multi-Agent Systems (AISB03/AAMAS02)*. Aberystwyth, Wales.

Nunes, L. and Oliveira, E. (2003c). Exchanging advice and learning to trust. In *Seventh Int. Conf. on Cooperative Information Agents (CIA-03)*. Helsinki, Finland.

Nunes, L. and Oliveira, E. (2003d). Exchanging advice and learning to trust. *Cooperative Information Agents VII, LNCS/LNAI 2782*.

Nunes, L. and Oliveira, E. (2004). Learning from multiple sources. In *Proc. of 3th Int. Joint conf. on Autonomous Agents and Multiagent Systems (AAMAS04)*, pages 1106–1113. IEEE, Computer Society. NY USA.

Nunes, L. and Oliveira, E. (2005a). Advice-exchange between evolutionary algorithms and reinforcement learning agents: Experiments in the pursuit domain. *Adaptive Agents and MAS II, LNCS/LNAI 3394*, pages 185–201.

Nunes, L. and Oliveira, E. (2005b). Communicating during learning. In *Workshop on Cooperation and Multiagent Learning, co-located with ECML'05*.

Oliveira, E. and Nunes, L. (2004). *Design of Intelligent Multi-Agent Systems*, chapter 9, pages 279–314. Studies in Fuzziness and Soft Computing Series. Springer.

Panait, L. and Luke, S. (2003). Cooperative multi-agent learning: The state of the art. Technical Report GMU-CS-TR-2003-1, George Mason University.

Powers, R. and Shoham, Y. (2005). New criteria and a new algorithm for learning in multi-¿gent systems. In *Proc. of Neural Information Processing Systems-2005*.

Price, B. (2003). *Accelerating Reinforcement Learning with Imitation*. Ph.D. thesis, University of British Columbia.

Price, B. and Boutilier, C. (1999). Implicit imitation in multiagent reinforcement learning. In *Proc. of the Sixteenth Int. Conf. on Machine Learning*, pages 325–334.

Price, B. and Boutilier, C. (2000). Imitation and reinforcement learning in agents with heterogeneous actions. In *Seventeenth Int. Conf. on Machine Learning ICML2000*.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, **1**(1), 81–106.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, **1**, 318–362.

Salustowicz, R. (1995). *A Genetic Algorithm for the Topological Optimization of Neural Networks*. Ph.D. thesis, Tech. Univ. Berlin.

Schaerf, A., Shoham, Y., and Tennenholtz, M. (1995). Adaptive load balance: A study in multi-agent learning. In *Journal of Artificial Intelligence Research*, pages 475–500.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, **5**(2), 197–227.

Sen, S. (1996). Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. In *Proc. of the Second Int. Conf. on Multiagent Systems*, pages 322–329. AAAI Press, Menlo Park, CA.

Sen, S. (1997). Multiagent systems: Milestones and new horizons. *Trends in Cognitive Science*, **1**(9), 334–339.

Sen, S. and Kar, P. P. (2002). Sharing a concept. In *Working Notes of the AAAI-02 Spring Symposium on Collaborative Learning Agents*.

SETI (2005). setiathome.ssl.berkley.edu.

Shoham, Y., Powers, R., and Grenager, T. (2003). Multi-agent reinforcement learning: a critical survey.

Silva, F. M. and Almeida, L. B. (1990). Speeding up backpropagation. *Advanced Neural Computers*, pages 151–158.

Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. In *Machine Learning*. SARSA.

Stone, P. and Veloso, M. (1997). Multiagent systems: A survey from a machine learning perspective. Technical Report CMU-CS-97-193, Carnegie Mellon University.

Sutton, R. S. and Barto, A. G. (1987). A temporal-difference model of classical conditioning. Technical Report TR87-509.2, GTE Labs.

Tan, M. (1993). Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proc. of the Tenth Int. Conf. on Machine Learning*, pages 330–337.

Taylor, M. E. and Stone, P. (2004). Speeding up reinforcement learning with behavior transfer. Technical report, AAAI 2004, Fall Symposium on Real-Life Reinforcement Learning.

Thorpe, T. (1997). *Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents*. Master's thesis, Department of Computer Science, Colorado State University.

Thrun, S. and Mitchell, T. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, **15**, 25–46.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. thesis, University of Cambridge, U.K.

Weibull, J. W. (1995). *Evolutionary Game Theory*. MIT Press.

Weiss, G. and Dillenbourg, P. (1999). What is 'multi' in multiagent learning. *Collaborative learning. Cognitive and computational approaches*, pages 64–80.

Whitehead, S. D. (1991). A complexity analysis of cooperative mechanisms in reinforcement learning. *Proc. of the 9th National Conf. on AI (AAAI-91)*, pages 607–613.

Whiteson and Stone, P. (2004). Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence special issue on Autonomic Computing and Automation*, **17**(7).

Wiering, M., Krose, and Groen (1999). Learning in multi-agent systems. Technical report, University of Amsterdam.

Wiering, M., van Veenen, J., Vreeken, J., and Koopman, A. (2004). Intelligent traffic light control. Technical Report UU-CS-2004-029, Institute of Information and Computing Sciences, Utrech University.

Yao, X. (1999). Evolving artificial neural networks. In *Proc. of IEEE*, volume 87, pages 1423–1447.

Appendix A

Abandoned Tracks and Unsolved Problems

A.1 Discussion

During the course of this research many approaches were attempted. Several included Competitive Learning in an attempt to summarize information, either to reduce communication or to compare state-spaces. These experiments were abandoned due to one, or more, of the following reasons:

- The solutions proved inadequate, either by reducing the performance or increasing the learning-times;
- The results were not conclusive as to the impact of the approach when compared to others', i.e. there was no significant difference in results or delay;
- Better solutions were found in the mean time, rendering the previous approach obsolete;
- The full presentation of results and discussion of all the abandoned approaches would increase the size of this document dramatically.

Nevertheless, the lessons learned with these "five thousand mistakes" are summarized in the following sections for the benefit of all those that may follow similar research directions.

A.1.1 Problems with quick transitions

We have identified, specially in traffic-control and load-balance problems, a phenomenon that was not easy to deal with in the simulations. It was observed that the threshold between easy and unsolvable problems was abrupt. Small changes in the

problem representation or parameterization rendered the same problems unsolvable. When we classify a problem as ”easy” we mean that some learning agents can, in a very small number of epochs and even without using communication, reach a reward that is near the maximum experienced for that problem and the reward remains at the same level until the end of the trial. The problems we have labeled “unsolvable” are those where none of solutions we have tried could find a reasonable policy until the end of the trial. The failure in learning possible good policies could be verified either by hand-coding an agent that would surpass this performance, or by observation of the agents' behavior using the graphical interface, for e.g.: in the predator-prey problem agents would simply not learn to get close to the prey in big arenas due to the vastness of the search-space and the low probability of catching an escaping prey with random movements; In the traffic problem agents would not be able to dispatch enough vehicles, avoiding the cluttering of the initial stretches of each lane, when the flow-multiplier was taken above a certain threshold (from 1.4 to 1.8, depending on the scenarios).

This problem was observed during the formulation of the problems (choice of state representation, actions, etc.) as well as in the choice of parameters (number of vehicles inserted, server speeds, etc.). As demonstrated by some of the results this problem was not completely eradicated, but the results of numerous experiments led us to sets of parameters (and problem structures) in the borderline of solvability.

A.1.2 Common storage format

One of the challenges in using the knowledge collected by agents with different learning structures is the possibility of translating all the hypotheses to a common format. Given the variety of hypothesis' structures used and the, even greater, number of possible structures available in different ML paradigms this goal was considered far to ambitious. After considering the problems in the translation of knowledge between Q-tables, ANN and program trees we have abandoned this goal and turned to the use of examples as representatives of an hypothesis. This lead us to the possibility of generalizing example representations, discussed in the next section.

A.1.3 Generalization of examples

The generalization of examples was attempted in two different ways. The first was to store the knowledge acquired during validation in the form of classes of examples, using Competitive Learning. Each example would be stored in the most similar class and change the representative of the class slightly, reducing the distance between itself and the class representative. This distance was a weighted least-mean-square, where the difference between actions had a higher weight to allow grouping of states that required the same action.

The second attempt was more akin to case-based reasoning, each example would search a list of pre-stored generalized-examples trying to find the best fit. A generalized-example would contain a state, action and reward, but instead of a definite value for each element of the action and state vectors the generalized-example contained an interval of values. When the best match was found one of three situations occurred: the new example fitted within the ranges of the closest generalized-example; the distance to the closest generalized-example was below a certain threshold; the distance to the closest generalized-example was above a certain threshold. In the first case there was nothing to be done since the example was already well represented. In the second case the ranges of the closest generalized-example would be expanded to include the new example, when too many examples were represented by one class this class was split in two. In the third case the new example was used to create a new generalized-example where ranges would be based on a small disturbance of the actual values in the example.

Both techniques were implemented with the intention of reducing communication and taking the place of a common storage format. When using these techniques we could communicate examples of an hypothesis by sending class-representatives or generalized-examples instead of the full set of validation examples. The results obtained were not encouraging. Often, the use of these techniques lead to a degradation of the performance and the cost in computation time was too high. This is mainly due to the exponential increase in the number of classes necessary to obtain a good representation of the example-space when it has many real-valued components. Another problem was in the generation of training examples from class representatives, which is a non-trivial matter and, in some cases, caused severe performance drops. These techniques can be considered for cases when communication costs are very high, but otherwise sending the full set of validation examples should produce better results.

A.1.4 Evaluation of advisor $\times$ situation

Another idea that we have explored was an attempt to relate advisors and their expertise. All agents used Competitive Learning (in a similar way to what was explained in the previous section) to keep a summarized representation of the state-space. Each class stored also the average reward obtained by the advisor for the correspondent examples. When using specific advice the advisee could question each advisor on its skill for a particular state. This question could always be answered, contrary to the situation presented in the thesis where the information is available only when an exact match is found in the validation stores, but the result would be an average score for the corresponding class of states. The information on which advisor performed best in the particular case the advisee was faced with was then used in the selection of the best advisor for that situation.

This technique was abandoned because it induced a considerable delay and caused

disturbances in the learning process, frequently leading to worse performances. These disturbances can be attributed to two main factors: 1) the constant use of different advisors prevented the acquisition of full policies and 2) the error in the estimation of performance induced by classes in which the reward had a high variance lead, frequently, to a bad selection of advisors.

A.1.5 Dynamic role-learning

In Sect. 5.2.6 we have debated the possible interpretations for the term "role" in MAS teams. One of these interpretations, which we labelled dynamic roles, consists using it as a label for each of the complementary policies used by the team's members. In this case it might be useful to find advisors that visit similar states and and have greater success. What we have essayed was to summarize the states visited by an agent using Competitive Learning and compare the sets of class representatives, ordered by their respective number of hits, in pairs of agents. This should allow us to discover the advisors that had a most similar set of state-representatives. This information was then used in the selection of standing advisors. Again, the delay introduced and the fact that no significant differences were observed in the performance, determined that this technique was left-out of the following experiments.

A.1.6 Team supervisors

Team-supervisors were our first attempt to synchronize the advise requests, preventing or discouraging any two members of the same team from requesting advice to the same advisor. This was achieved by informing the team supervisor when a new advisor was selected. The team supervisors, would, in turn, signal the remaining partners to disable, or reduce the trust in, this advisor. The use of team supervisors was made obsolete by the introduction of roles.

A.1.7 Combining advice

When using specific advice from multiple advisors an agent can combine the information given by several agents in several ways. We have experimented simple voting and weighted voting. In the first case an agent would choose the action in which more advisors had voted. In the second case the votes were weighted by the advisors' performance. One of the main disadvantages was that this process was tied to specific, online, imitation. Some agents (EA and GP-agents) could not learn by imitating the advisors' action, thus could not use this process. Experiments using QL-agents resulted in no significant difference from the approaches presented in this thesis. The advantages of choosing a better advice were, at times, shadowed by the constant change of advisors that made it difficult to learn coherent policies.

A.1.8 Confidence

When advice is retrieved from a store the advisee can calculate a state-confidence ($c_s(.)$) based on the difference between its own observed state and the $\bar{s_t}$ component of the advice from agent j, represented as $\bar{s_{j,t}}$ below:

$$c_{s(}\bar{s}_{i,t}, \bar{s}_{j,t}) = 1 - |\bar{s}_{i,t} - \bar{s}_{j,t}|/\sqrt{n}\ , \tag{A.1}$$

where n is the dimension of the state vectors. Here it is assumed that state vectors contain values in [0,1]. This may be useful when the agent gets (from its own stores or from its advisors') advice that was previously used in states that, although similar, do not match exactly the advisees' current state.

The state-confidence was used in the selection of specific advice from multiple advisors as a weight applied to the estimated reward. It was abandoned when we restricted the retrieval of advice to exact matches. Even though an exact match must use thresholds when the state components are real-valued, the use of non-exact matches beyond these thresholds lead to bad advice even when confidence thresholds were used to filter these matches.

Since the action is real vector representing the adequacy of each possible choice the advisee can also evaluate an action-confidence. This quantity can be defined as:

$$c_{a(}\bar{a}_{i,t}) = 1/\sum_{n \neq m} |a_{i,t,n} - a_{i,t,m}|, \tag{A.2}$$

where m is the index of the highest value in $\bar{a}_{i,t}$.

A similar type of action-confidence is proposed by Clouse (1997), although it takes the form of a decision *criterium*. An agent is said to be confident in a certain action if the difference between the maximum value and all the other components of $\bar{a}_{i,t}$ is greater than a certain threshold. This definition could be extended to problems in which there are several actions that produce similar results by defining the maximum number of components whose distance to the maximum is smaller than the threshold. When this limit was exceeded the agent would not be confident in the proposed action.

Action-confidence was used to decide whether or not to request advice based on the result of the advisees' evaluation-function. Results were inconclusive as to the effectiveness of this approach. Although it did reduce communication it also affected learning due to the mix between the actions selected by the advisee and those selected by the advisor, when the former was less confident. Apparently this result contradicts Clouse's conclusions on the effectiveness of action-confidence as a mechanism to select when to choose an advisor. This contradiction can be explained by the fact that Clouse uses a single expert-advisor. This reduces significantly the impact of bad advice. Even when noise is introduced, to determine the effects of bad advice, we have a significant difference between Clouse's experiments and ours: the introduction of random white noise can be effectively dealt with by most learning algorithms, while in

our experiments bad advice cannot be considered white noise because it is the result of a specific policy and will, for long periods, produce the same action in response to the same state.

A.1.9 Influence-Exchange

At one point in our investigation we have considered the possibility of exchanging suggestions with partners. An agent tried to influence its partners to perform the actions that were in its own best interest. Its partner would accept the influence if the expected performance drop would be less than a certain percentage of the best expected reward for its current situation. The agent that issued the influence would try to learn how to obtain best results from influencing its partners by correlating good results with influences whenever these were accepted. This (possibly naive) approach had the following consequences:

- Severe performance drops;
- Making policy evaluations much harder;
- Spending a lot of resources in learning the effects of different combinations of neighbors' actions, which also leads to a form of centralization.

It was found that this technique does not lead to better global policies in the tested cases. When an agent accepts influences it is also harder to evaluate the performance of a certain policy. Even if the amount of influence is low it may have occurred at crucial moments and changed dramatically the results of applying the current policy. Since the effects of communication of learning-related data was our main concern we have soon abandoned "influence-exchange". This does not mean that all influence exchange processes are doomed to fail, but that the coordination of influences and learning is very difficult and we have found no way to overcome the problems of this combination.

Appendix B

Simulator Design and Parameters

The experimental framework was coded in C++, on a *Linux* operating system (Mandrake Linux 9.0 3.2-1mdk). It was compiled with gcc (version 2.9 and later 3.2) and debugged with gdb (version 5.2). The only external code used as source was the boost library (version 1.29). Graphic displays were coded in Java (2SE, Java Sun 1.3 and later 1.4.1). The C++ code, made specifically for this project, contains, roughly, 230 classes, comprising in total more than $90,000$ lines of code (blanks and comments included). The majority of the tests run in an Intel P3 at 1.2GHz (256Mb memory).

In this appendix we will present a brief historical summary of the development of this project and a summary of the main classes used in the simulations. The actual code can be found in the attached CD. This code will also be publicly available at http://home.iscte.pt/ lmmn/www.

B.1 Design

The maintenance of a flexible and extensible code was one of main concerns throughout the project. The core of the base-classes has suffered little change since the beginning of the project, which is a strong indication of good initial design.

Scientific software has special characteristics that are seldom mentioned in Software Engineering textbooks, mainly the fact that requirements are constantly changing which prevents the application of development strategies designed for commercial software.

Our approach, although partly *ad-hoc* followed a development strategy closely related to the spiral model (or prototyping-model). We are convinced that only a good

usage of the Object Oriented paradigm made it possible to cope, single-handedly, with a project of this magnitude.

B.1.1 Algorithms, Parameters and Evaluation Functions

The first part of the system to be implemented was a first version of the Algorithm (Fig. B.1), Evaluation Functions (Fig. B.3) and Parameters (Fig. B.2) hierarchies. The Neural-Networks' classes were imported from a previous project (Nunes, 1997). These modules, that at the time did not include STGP and Program-Trees, were thoroughly tested in a series of ML problems.

Later the Program-Trees and Genetic-Programming modules were added. The introduction of GP and integration with EA classes caused a few minor structural changes with the distinction between EA-Parameters and ConnEA-Parameters. The first capture the common features of ConnEA-Parameters and GP-Parameters, while the second contain the features that are specific of ANN-based EA.

The Algorithms hierarchy contains Supervised, Reward-Based and Unsupervised sub-trees (the latter not shown in Fig. B.1). Each class contains the sequence of steps for a given type of learning-algorithm.

Parameters contain many basic functions such as those related to storing and retrieving hypotheses. At the leaves of this tree we have the actual update formulas for each type of learning parameters.

The Evaluation Functions hierarchy contains the several types of evaluation-functions used (and a few others not displayed here due to their irrelevance for the current work).

The most interesting part of the interplay between these modules is in the the initialization procedures. Since there may be several combinations of these three types of structures we have chosen to keep this as dynamic as possible. The sequence of construction is: Algorithm, Evaluation Function, Parameters. After the Algorithm is built (without any internal parameters) the construction of the Evaluation Function is triggered having the Algorithm as a parameter, during this construction the Evaluation Function asks the Algorithm for sets of parameters with a specific set of restrictions and maps these parameters into its own evaluation structure. The parameters, built by the Algorithm on request, are shared between these two structures. The Algorithm sends update messages when the learning process so requires, while the Evaluation Function sends evaluate messages.

B.1.2 Statistics

Statistics classes were first included in the Algorithms' classes and later isolated as an independent module, used by the Algorithms and the Agents to process different views of the information. These classes can be parameterized to save and recover data

in several different ways releasing memory resources or diminishing the input-output-time, depending on the users' needs.

B.1.3 Environments and Agents

The Environment and Agents' base-classes were implemented and tested in parallel on the Traffic and Predator-Prey environments. This was the time for one of the most difficult decisions: whether or not to support concurrent processes or detached processes. The reasons for choosing to simulate parallelism were already presented in section 5, but, in summary, we can say that the main justification was that concurrence-related problems were outside our intended scope.

The Load-balance module was later built upon this hierarchy with minor adjustments to the running cycle to allow for multiple actions in one turn.

The Environments and Agents' classes are coupled by multi-level registration. Each level of the Environments' hierarchy knows the instances of Agents at the same level. For example an agent related a specific problem X that is, by inheritance a Learning-Agent and an Environment-Agent, is registered in the XEnvironment, in the Learning-Environment and in the MASL-Environment. Each environment is responsible for one view of the system. The X-Environment is in charge of problem-specific messages, the Learning-Environment triggers reward calculations and end-of-epoch messages and the MASL-Environment contains structures that are common to all MASL-Environments, such as a Directory facility.

B.1.4 Java Interface Server

The Java interface-server is graphical information display server, used mostly for the predator-prey and traffic-control modules. It is completely independent of the rest of the structures and runs as an independent process. The interface can be switched on/off at any time. When launched the interface server will open a TCP connection at a specific port and wait for a connections to be established. The environment will, at specific times defined in the configuration file `graph.ptab`, attempt to connect to the servers' port. When connected the environment will send redraw messages to all the objects that can be displayed (`GraphObject`) at the end of each turn. To ensure that the interface-server is not flooded the environment will sleep for a given time (configurable for each environment). The type of messages (in text) understood by the server are:

- Set world: Sets the representable area. Arguments: upper left corner x-coordinate, upper left corner y-coordinate, length, height;
- Set view-port: Sets the viewed area. Arguments: upper left corner x-coordinate, upper left corner y-coordinate, length, height;

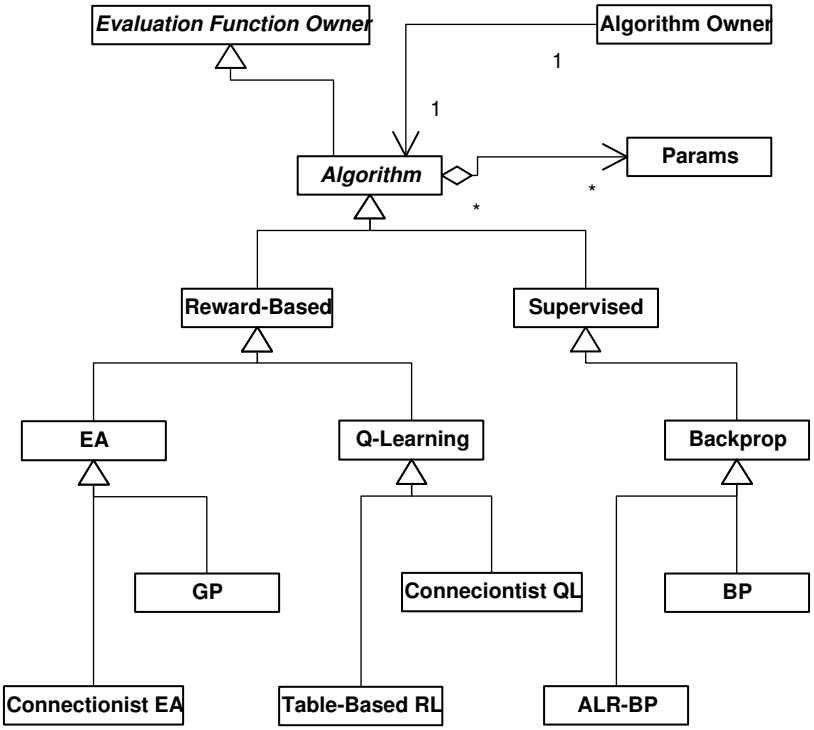

Figure B.1: Summary of the Algorithm hierarchy.

- Set zoom: Sets the zoom. Arguments: increase/decrease.
- Scroll: Scrolls the view-port in the world area. Arguments: Direction.
- Add: Adds a new graphical object. Arguments: object id, shape, position, color, fill-type. Where shape is one of the following: circle, rectangle.
- Remove: Removes an object from the graphical interface. Arguments: Object-id.
- Change color: Changes the color of an object: Arguments: Object-id, new-color.
- Display: Orders an object to be displayed. Arguments: Object-id.
- Undisplay: Orders an object to be hidden from view. Arguments: Object-id.
- Refresh: Orders an update of the displayed image.

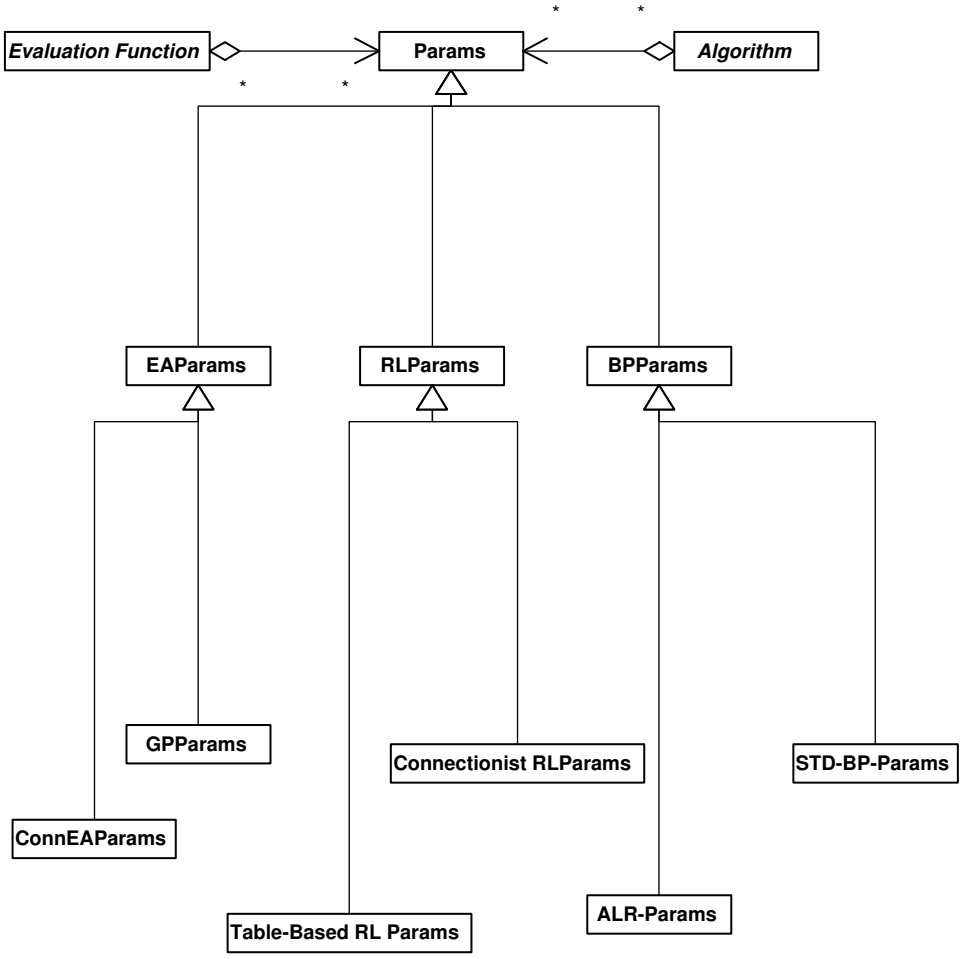

Figure B.2: Summary of the Parameters hierarchy.

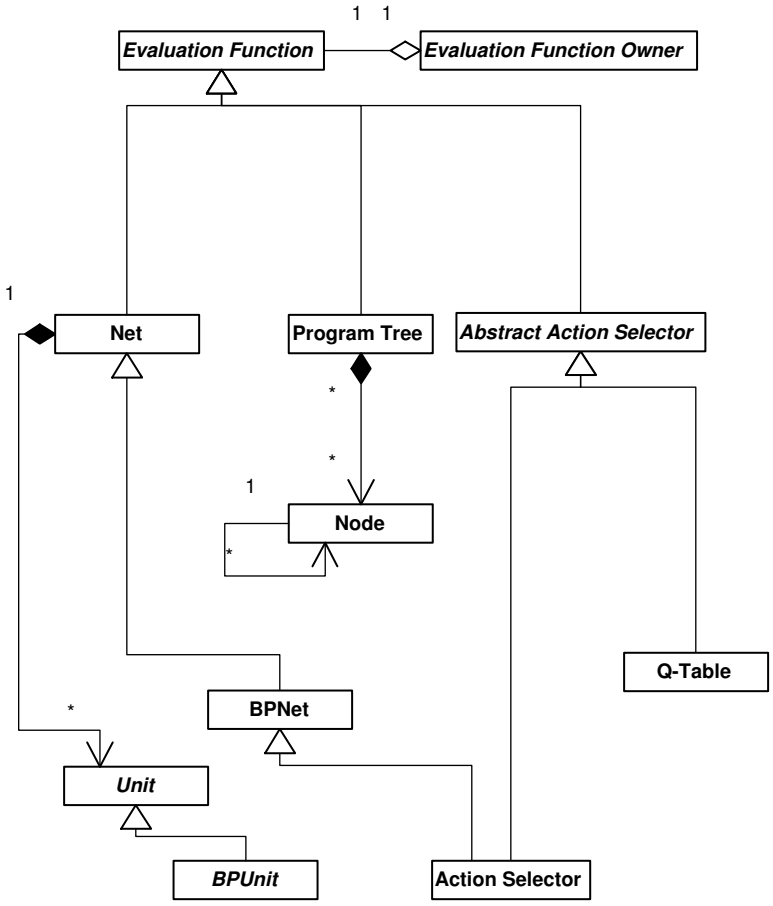

Figure B.3: Summary of the Evaluation Functions hierarchy.

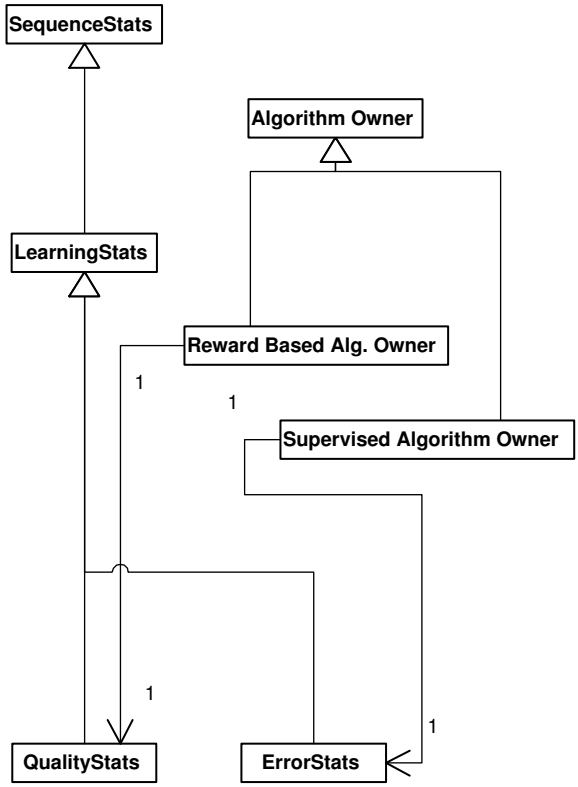

Figure B.4: Summary of the Statistics hierarchy.

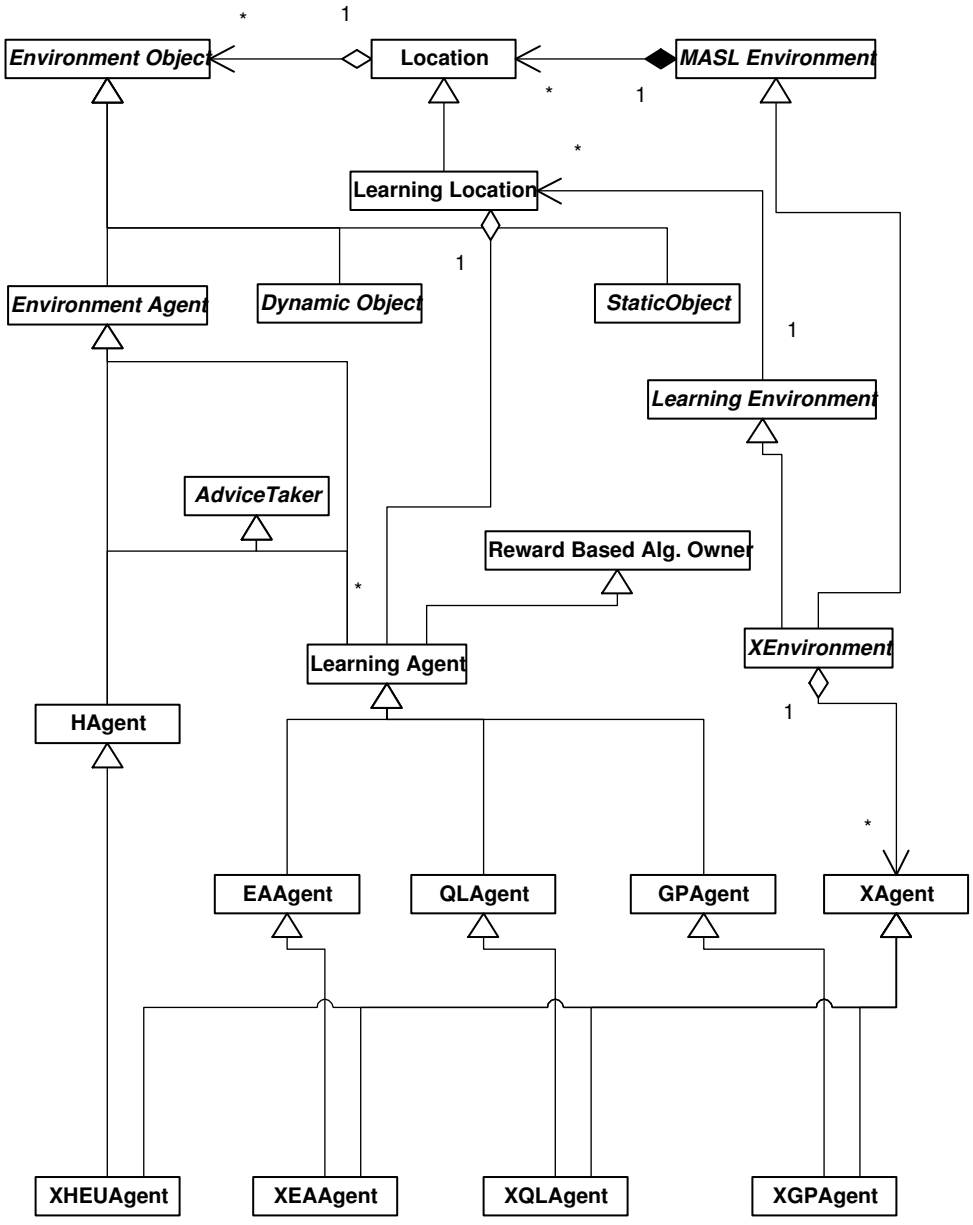

Figure B.5: Summary of the Agent's hierarchy. The X stands for each of the specialized agents and environments for each particular problem (Predator-Prey, Traffic-Control and Load-Balance).

B.2 Simulation Parameters

B.2.1 Directory organization

The common procedure is to have an output directory named *./trial/problem-name* and an input directory named *./trial/problem-name/paramFiles*. Both are automatically generated in the running scripts, based on a reference like *SM2H1*.

Input Directories

```
base-dir/
  agent1/                  # EA-agent
    AdviceTaker/
      adv.alg.ptab    # Advice parameters
    BP/
      bp.alg.ptab          # BP parameters
      bp.net_array.iobj    # ANN array size
      bp.net.iobj          # ANN structure
      stats.ptab           # Statistics parameters
    COnnEA/
      ea.alg.ptab          # EA parameters
      ea.net_array.iobj    # ANN array size
      ea.net.iobj          # ANN structure
      stats.ptab           # Statistics parameters
  agent2/                  # GP-agent
    AdviceTaker/
      adv.alg.ptab         # Advice parameters
    GeneticProgramming/
      gp.alg.ptab          # GP parameters
      gp.pt.iobj           # Program-tree parameters
      stats.ptab           # Statistics parameters
  agent3/                  # QL-agent
    AdviceTaker/
      adv.alg.ptab         # Advice parameters
    Reinforcement/
      BP/
        bp.alg.ptab        # BP parameters (ConnQL only)
        bp.net_array.iobj
        bp.net.iobj        # ANN structure
        stats.ptab         # Statistics parameters
      rl.alg.ptab          # QL parameters
      rl.sm.iobj           # Table-QL parameters
```

```
      stats.ptab          # Statistics parameters

  agent4/                 # H-agent
    AdviceTaker/
      adv.alg.ptab        # Advice parameters
    stats.ptab            # Statistics parameters

  {...}

  graph.ptab              # Graphic interface parameters
  masl.ptab               # Environment parameters
  problem-name.env        # Environment structure
```

Output Directories

This is an example of the appearance of the directories at the end of a trial. The use of the statistics scripts will add new files to these directories.

```
base-dir/
  agent1/
    BP/                             # EA-agent
      learning.history              # statistics history
                                    # (individual reward)
      test.history
      validation.history
      combined_learning.history     # statistics history
                                    # (combined reward)
      combined_test.history
      combined_validation.history
      initial.BP.params             # initial hypothesis
      current.BP.params             # last saved hypothesis
      final.BP.params               # final hypothesis
      Xagent.comms                  # communications
                                    # with agent X
    ConnEA/
      learning.history              # statistics history
      test.history                  # (individual reward)
      validation.history
      combined_learning.history
      combined_test.history
      combined_validation.history
      initial.ConnEA.params         # hypothesis files
```

```
    current.ConnEA.params
    final.ConnEA.params

{...}

seed                                # seed-file
test.log                            # Output record

{...}

area1/
  area.stats
```

B.2.2 Simulation Parameters

This section describes the simulation parameters used in our experiments. The actual or default values of these parameters are displayed in brackets, where appropriate.

Table B.1: Environment parameters for Predator Prey (file:masl.ptab).

Name	Type	Description
output_dir	string	Output directory
state_size	int	Number of state elements (18)
action_size	int	Number of action elements (9)
save_time	int	Number of epochs between saves (100)
n_turns_per_epoch	int	Number of turns per epoch (60)
advice_allowed	bool	Allows advice-exchange if true

Table B.2: Environment parameters for Load Balance (file:masl.ptab).

Name	Type	Description
output_dir	string	Output directory
state_size	int	Number of state elements (3)
action_size	int	Number of action elements (9)
save_time	int	Number of epochs between saves (100)
n_turns_per_epoch	int	Number of turns per epoch (100)
advice_allowed	bool	Allows advice-exchange if true

Table B.3: Environment parameters for Traffic Control (file:masl.ptab).

Name	Type	Description
output_dir	string	Output directory
state_size	int	Number of state elements (6)
action_size	int	Number of action elements (2)
save_time	int	Number of epochs between saves (100)
n_turns_per_epoch	int	Number of actions per epoch (100)
advice_allowed	bool	Allows advice-exchange if true
max_speed	int	Maximum car speed (30m/s)

Table B.4: Statistics-related parameters (file:stats.ptab)

Name	Type	Description
learning.dumping_stats	bool	Dumping old statistics to file?
learning.keep_n	int	Number of history values stored
learning.save_frequency	int	Number of epochs between saves
learning.short_term_interval	int	Number of epochs that define a short-term interval
learning.mid_term_interval	int	Number of epochs that define a mid-term interval
learning.long_term_interval	int	Number of epochs that define a long-term interval
learning.best_feedback_decay	real	Decay used in Eq. 5.4
learning.previous_feedback	real	Weight used in Eq. 5.1
learning.n_history_values	int	Number of history values used in Eq. 5.7

Table B.5: Advice related parameters (file:adv.alg.ptab).

Name	Type	Description
individual_reward_weight	real	Individual reward weight (0.5)
team_reward_weight	real	Global reward weight (0.5)
action_selection_type	string	Switch for stochastic/deterministic action selection (deterministic)
replay_last_trial	bool	Flag for QL replay
request_specific_advice	bool	Specific-advice flag
request_batch_advice	bool	Batch-advice flag
use_advice_online	bool	Online advice flag
use_advice_offline	bool	Offline advice flag
imitate	bool	Imitation flag
biasing	bool	Biasing flag
max_advisors	int	Maximum number of simultaneous advisors
use_any_advisor	bool	Multiple advisors flag
use_standing_advisor	bool	Standing-advisor flag
use_learning_stages	bool	Learning-stages flag
use_trust	bool	Trust flag
use_roles	bool	Roles flag
worse_scores_can_advice	bool	Agents with lower scores can advise? (true)
advisor_reevaluation	int	Number of epochs between advisor re-evaluations (10)
max_stores	int	Maximum number of epochs of advice saved (10)
minimum_increase_short_term	real	Short-term stability threshold (0.01)
minimum_increase_mid_term	real	Mid-term stability threshold (0.01)
minimum_increase_long_term	real	Long-term stability threshold (0.05)
trust_update_rate	real	Maximum update-rate for trust values (0.7)

Table B.6: Parameters for EA and GP algorithms (files: ea.al.ptab, gp.alg.ptab).

Name	Type	Description
Evaluation Function	string	Type of evaluation function used
auxiliary_alg	string	Type of auxiliary algorithm used
init_mutation_prob	real	Initial mutation probability
max_mutation_prob	real	Maximum mutation probability
min_mutation_prob	real	Minimum mutation probability
mutation_prob_decay	real	Mutation probability decay-rate
mutation_prob_decay_schedule	int	Number of epochs between updates
init_mutation_rate	real	Initial mutation rate
max_mutation_rate	real	Maximum mutation rate
min_mutation_rate	real	Minimum mutation rate
mutation_rate_decay	real	Mutation rate decay-rate
mutation_rate_decay_schedule	int	Number of epochs between updates
crossover_by_rows	bool	Flags that control crossover for EA-Agents
crossover_by_cols	bool	Flags that control crossover for EA-Agents
crossover_by_jump	bool	Flags that control crossover for EA-Agents
crossover_by_bundle	bool	Flags that control crossover for EA-Agents
special_last_crossover	bool	Flags that control crossover for EA-Agents
crossover_jump	int	Flags that control crossover for EA-Agents
bundle_size	int	Flags that control crossover for EA-Agents
initial_population_avg	real	Initial distribution average
initial_population_span	real	Initial distribution variance
population_size	int	Number of specimens in a population
selected_p	real	% of the population selected for breeding
keep_best_p	real	% of the population that passes untouched to the next generation
mutate_p	0.7	% of the population generated by mutation
crossover_p	0.2	% of the population generated by crossover
n_trials_per_specimen	1	Number of epochs per specimen

Table B.7: Parameters for QL algorithms (file:rl.alg.ptab)

Name	Type	Description
Evaluation Function	string	Type of Evaluation function used (Table-QL)
update_type	string	Type of update (QL)
StateHasher	string	Hash-table description file-name
discount	real	parameter β in Eq. 2.5
exploration_rate	real	Initial exploration-rate
min_exploration_rate	real	Minimum exploration-rate
max_exploration_rate	real	Maximum exploration-rate
exploration_rate_decay	real	Exploration-rate decay
exploration_rate_decay_schedule	int	Number of epochs between updates
learning_rate	real	Learning-rate, α in Eq. 2.5
max_learning_rate	real	Maximum learning-rate
min_learning_rate	real	Minimum learning-rate
finish_learning_rate_decay_at	int	Number of epochs to reach minimum learning-rate
learning_rate_decay_schedule	int	Number of epochs between updates

Table B.8: Program-Tree parameters (file:gp.pt.iobj)

Name	Type	Description
n_nodes	int	Number of output nodes (equal to action size)
init_max_level	int	Maximum depth of initial trees
max_level	int	Maximum depth allowed
input_size	int	State dimension

Example of an ANN used in the load-balance problem.

Definition of an ANN (file:bp.net.iobj):

```
Unit 0 Weights 3 20
Unit 1 Sigmoid 20
Unit 2 Weights 20 1
Unit 3 Sigmoid 1

# Inputs <range>
Inputs 0
InputSize 3

#Connect <out_range> - <in_range>
Connect 0 - 1
Connect 1 - 2
Connect 2 - 3

# Outputs <range>
 Outputs 3
 OutputSize 1
```

Example of an environment definition file (traffic.env). The character '#' is used as a comment marker.

```
#World: start position (x,y) and size (x,y)
World
 0 0 716 516
#StateDescription: number of elements,
#maximum value, minimum value, step.
StateDescription
 6
 1 1 1 1 1 1
 0 0 0 0 0 0
 0.2 0.2 0.2 0.2 1 0.1
#ActionDescription: number of actions,
#maximum value, minimum value.
ActionDescription
 2 0.9 0.1
#Areas: number, start position (x,y) and size (x,y)
Area 1
 10 10 348 248
Area 2
```

```
 368 10 348 248
Area 3
 10 268 348 248

#PreObjects: problem dependent,
# in this case its lane definition
#Lane: number, start position (x,y), length, area,
#direction, delay, flow database files
#(start-day, end-day)
Lane 1 108 0 132 Area 1 South 5
file: paramFiles/traffic/Flows/ct5_ 15 30
Lane 2 112 0 132 Area 1 South 5
file: paramFiles/traffic/Flows/ct5_ 15 30
Lane 3 116 100 132 Area 1 North 10
file: paramFiles/traffic/Flows/ct6_ 15 30

{...}

Lane 1 108 0 132 Area 2 South 5
file: paramFiles/traffic/Flows/ct5_ 15 30
Lane 2 112 0 132 Area 2 South 5
file: paramFiles/traffic/Flows/ct5_ 15 30
Lane 3 116 100 132 Area 2 North 10
file: paramFiles/traffic/Flows/ct6_ 15 30

{...}


#EAAgents: Area, number of traffic-lights controlled,
# definition of traffic-lights
# Algorithm parameters' directory
# Role
# Aux. Algorithm parameters' directory
EAAgent
 Area 1
 NSems 4

# Traffic-light: graphical position,
# number of lights controlled,
# lane for a specific light,
# position of stop-line in lane
```

```
 TrafficLight 104 100
  2
   1 107
   2 107
 TrafficLight 128 102
  2
   5 223
   6 223
 TrafficLight 100 124
  2
   7 107
   8 107
 TrafficLight 124 128
  2
   3 107
   4 107
 ConnEA trial/.../ConnEA
 StaticRoleWatcher Role1
 ALR_BP trial/.../BP

{...}

#GPAgents
GPAgent
 Area 2
 NSems 4

 TrafficLight 104 100
  2
   1 107
   2 107

{...}
 GeneticProgramming trial/.../GeneticProgramming
 StaticRoleWatcher Role1

{...}

#QLAgents
QLAgent
 Area 3
```

```
 NSems 4

 TrafficLight 104 100
  2
   1 107
   2 107

{...}

 Reinforcement trial/.../Reinforcement
 StaticRoleWatcher Role1
```

B.3 Response to the main events

Summary, in pseudo-code, of the core routines in classes: LearningAgent and AdviceTaker.

On reception of observed state:

```
Observed state:  s_{i,t}
IF specific-advice
     Search for advice in example stores
     (first exact match for s_{i,t},
     stores searched in random order)
     IF advice not found
         IF using standing advisor
             k = standing advisor
         ELSE
             k = advisor with best estimated reward
         ENDIF
         Request advice for (s_{i,t}) to agent k
         IF imitating
             Wait for reply
         ENDIF
         a_{i,t} = Select action for state (s_{i,t})
         Perform action(a_{i,t})
     ENDIF
ENDIF
```

On reception of reply to advice request for $s_{i,t}$:

```
Advice: e_{k,t}, Advised action: a_{k,t}
Store advice (e_{k,t})
IF online-advice
    IF imitating
        a_{i,t} ← a_{k,t}
        Perform action(a_{k,t})
    ELSE
        Process advice
        (Virt. Exp., Biasing or online BP)
    ENDIF
ENDIF
IF a_{i,t} = a_{k,t}
    action-imitation ← true
ENDIF
```

On reception of individual reward for action $a_{i,t}$:

```
Reward: r_{i,t}
IF action-imitation
    Update online-advisor information
ENDIF
Update statistics
Store example e_{i,t} = {s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1}}
Learn e_{i,t}
```

On end of learning epoch:

```
Epoch reward: r_{l,n}
Calculate combined reward (R_{i,n})
Update statistics
Retreive responsible advisors
FOR all responsible advisors
    IF using trust
        Update trust in responsible advisor
IF offline-advice
    FOR all advisors
        IF advice available
            Replay stored advice
        ENDIF
ENDIF
```

On end of validation epoch:

```
Epoch-reward: r_{l,n}
```

```
Calculate R_{i,n}
Service standing requests
 (R_{i,n} and validation examples)
```

On reception of new performance information from peer k:

```
Combined Reward for peer k:  R_{k,n}
IF learning-stages
    Update learning-stage
    IF learning-stage changed
       Update advice mode
       Update learning params
    ELSE
       start/stop advice
    ENDIF
    IF using standing advisor
       re-evalaute standing advisor
       IF advisor changed
          clear advice-stores
          renew standing requests
       ENDIF
    ENDIF
ENDIF
```

Appendix C

Detailed Results

The results presented in this section were calculated as follows. After 19980 epochs (19800 in the case of traffic-control) the performance was measured and averaged in 100 consecutive validation epochs (for the predator prey problem), 160 (for load-balance), and 180 (for traffic-control). The results, presented in the third and fourth columns of the tables in this section, are the average and standard deviation of this result for all agents of each type in the 6 trials made for each scenario. In trials with 2 agents per area each value results from a minimum of 12 (heterogeneous trials) and a maximum of 36 points (homogeneous trials). In the scenarios using 4 agents per area the minimum is 24 values and maximum is 72. The graphics present the average combined reward per epoch of all agents of a given type in a specific scenario. The x-axis represents the number of validation epochs while the y-axis represents the average combined-reward for the same epochs. All measurements are taken at the end of validation epochs in which the current best hypothesis is used. Each point is the result of averaging a given number of validation results (shown on the y-axis label). The difference in the number of validation epochs for different problems is related to the learning-test-validation cycles, explained in the introduction of chapter 5. The last point in each graphic includes the final validation results. When appropriate the baseline and/or heuristic results are also shown for comparison.

The Reference (Ref.) code summarizes the conditions of each scenario. The first character represents Social (S) or Individual (I) experiments, i.e. with or without communication, respectively. The second character encodes the type of learning agents present: EA-agents (E), GP-agents (G), QL-agents (Q) or Mixed (M) (i.e. all previous types). The third character (first digit) represents the number of agents per area. The last digit represents the number of heuristic areas. The number of the experiment-set and a summary of the parameters may follow the reference when necessary. All H0 scenarios have 3 areas. The homogeneous scenarios have 6 $\times$ EA, GP or QL-agents, while the heterogeneous scenarios (M) have 2 agents of each of the previous types. H1

environments have an extra area with H-agents. The results for the H-agents are summarized in the last line of each table, where appropriate, with minimum and maximum scores and maximum standard deviation. The last column in each table represents the median run-time in minutes. We have chosen median times rather than other statistics because part of the experiments were made on stand-alone mode while others were not. The use of a median provides a better measure for comparison by filtering-out the high running-times achieved when running in parallel with other applications. In heterogeneous scenarios the results were averaged for each algorithm and the code of the algorithm (EA, GP, QL or HEU) is appended to the reference.

The second column contains information on the type of advice parameters used in the experiment. Only the most relevant parameters for each experiment are presented.

This appendix was generated automatically with the scripts: `report_all` and `generate_appendix`.

C.1 Predator Prey Baseline Tests

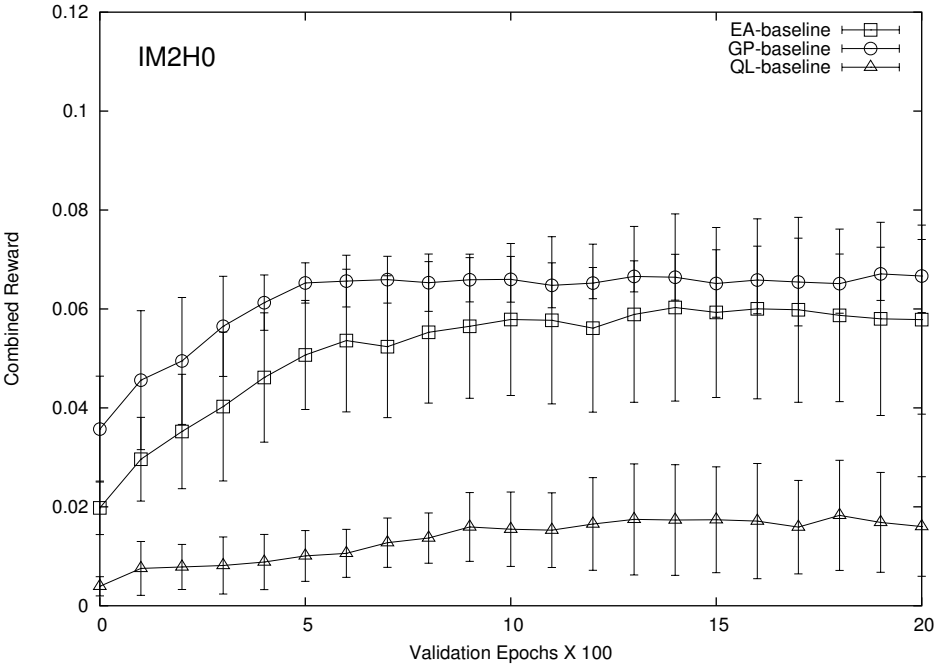

Figure C.1: Average evolution of combined reward for the Predator-Prey problem in the baseline experiments (IM2H0).

Table C.1: Results of the final validation cycle for the baseline Experiment Set on the Predator-Prey problem.

Agent	Avg	Std Dev
QL	0.016016	0.0100669
EA	0.0578403	0.0191091
GP	0.0666539	0.00739038

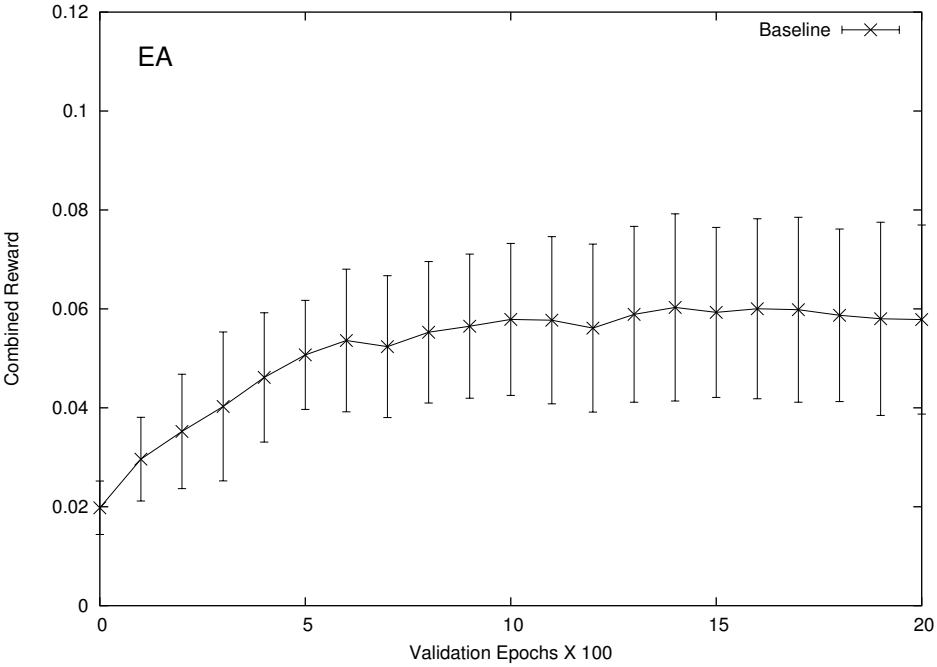

Figure C.2: Summary of results for EA-agents in the baseline Experiment Set on the Predator-Prey problem.

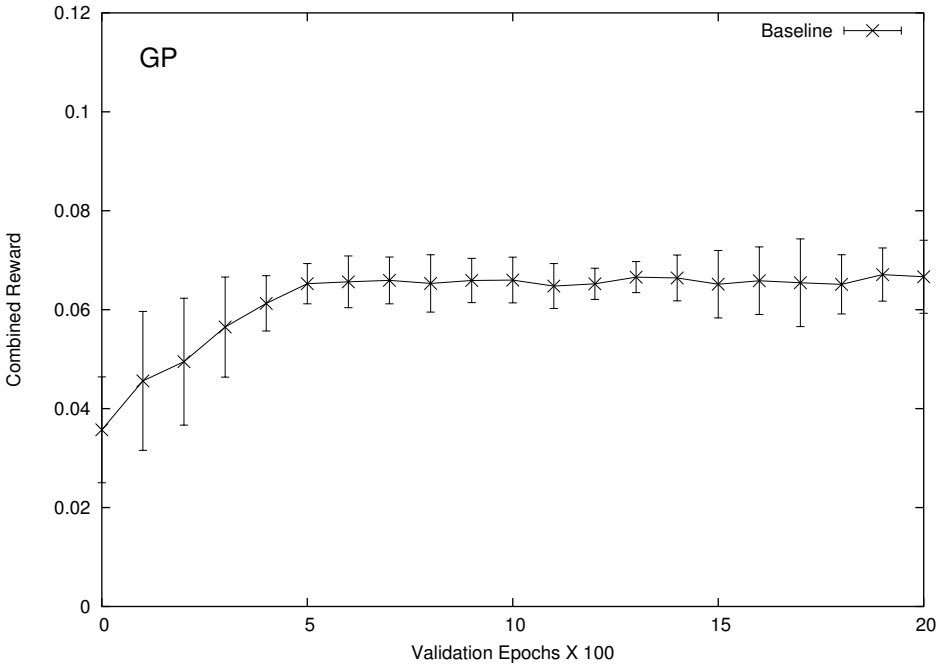

Figure C.3: Summary of results for GP-agents in baseline Experiment Set on the Predator-Prey problem.

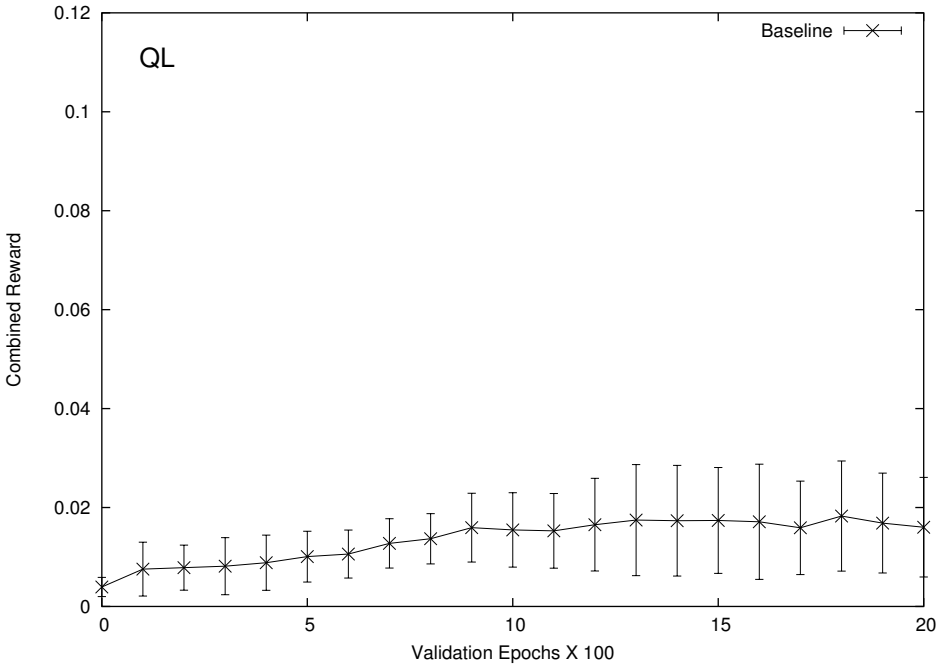

Figure C.4: Summary of results for QL-agents in the baseline Experiment Set on the Predator-Prey problem.

C.2 Experiment Set 1

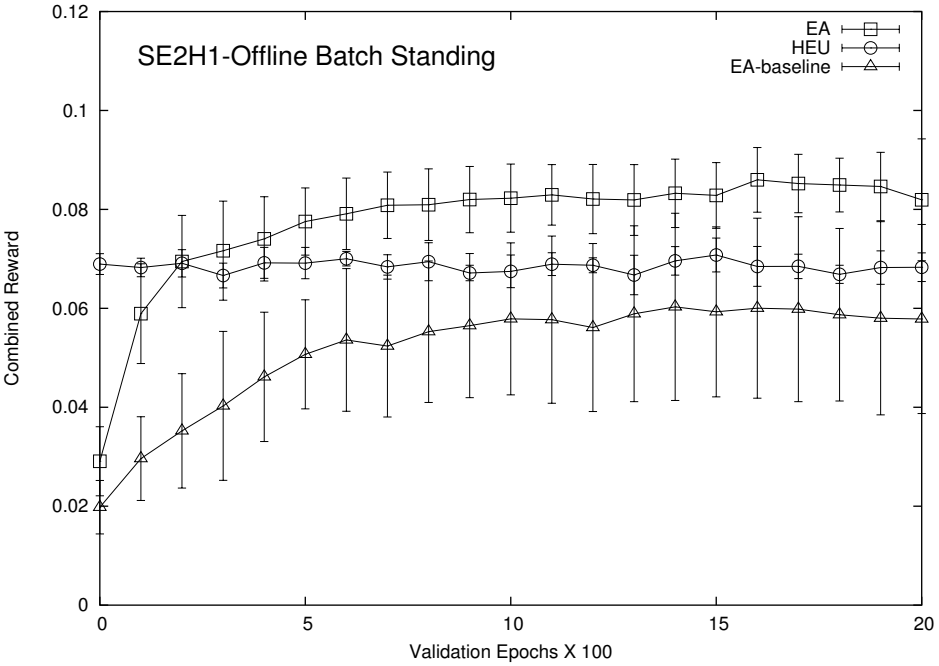

Figure C.5: Average evolution of the combined reward in experiment SE2H1-Offline Batch Standing.

Table C.2: Results of the final validation cycle for experiment SE2H1-Offline Batch Standing.

Agent	Avg	Std Dev
HEU	0.0683082	0.00290115
EA	0.0819361	0.0123265

Table C.3: Results of the final validation cycle for experiment SE2H1-Offline Specific Standing.

Agent	Avg	Std Dev
HEU	0.0674889	0.00290811
EA	0.0766049	0.00889292

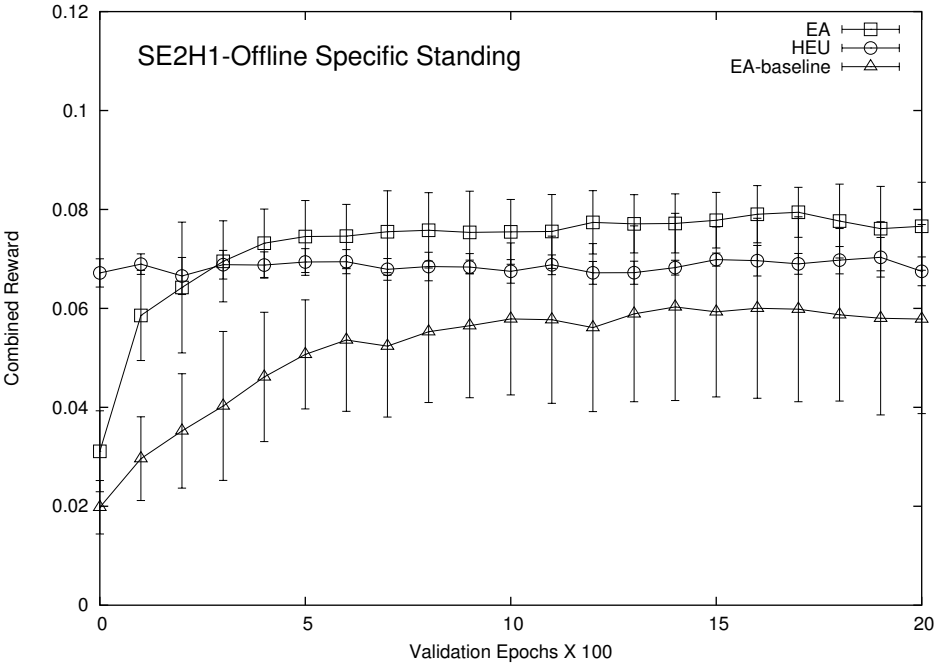

Figure C.6: Average evolution of the combined reward in experiment SE2H1-Offline Specific Standing.

Table C.4: Results of the final validation cycle for experiment SG2H1-Offline Batch Standing.

Agent	Avg	Std Dev
HEU	0.0681671	0.00237103
GP	0.0685259	0.00736909

Table C.5: Results of the final validation cycle for experiment SG2H1-Offline Specific Standing.

Agent	Avg	Std Dev
HEU	0.0706467	0.00327752
GP	0.0679962	0.0076324

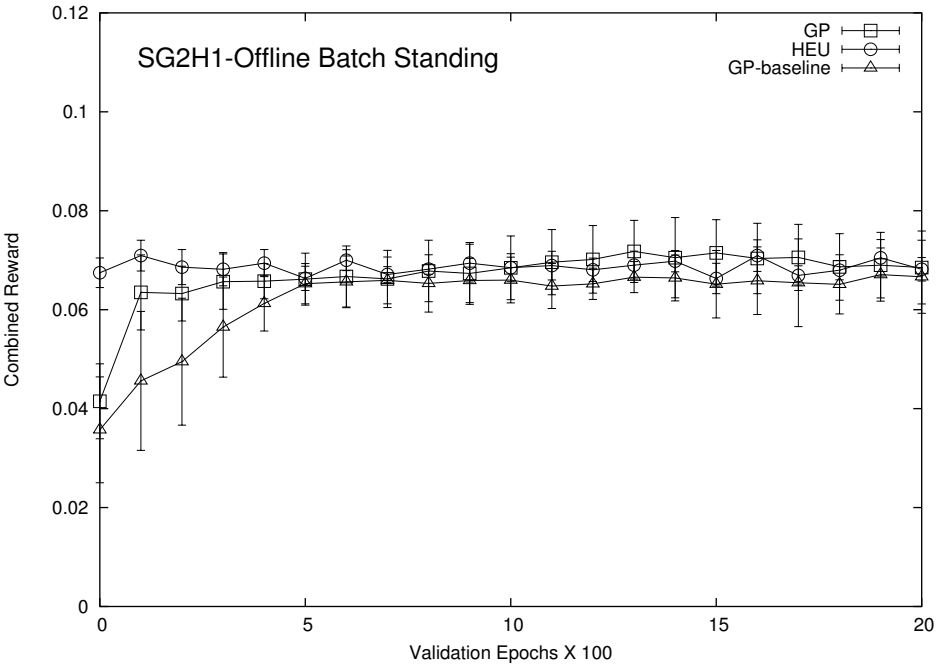

Figure C.7: Average evolution of the combined reward in experiment SG2H1-Offline Batch Standing.

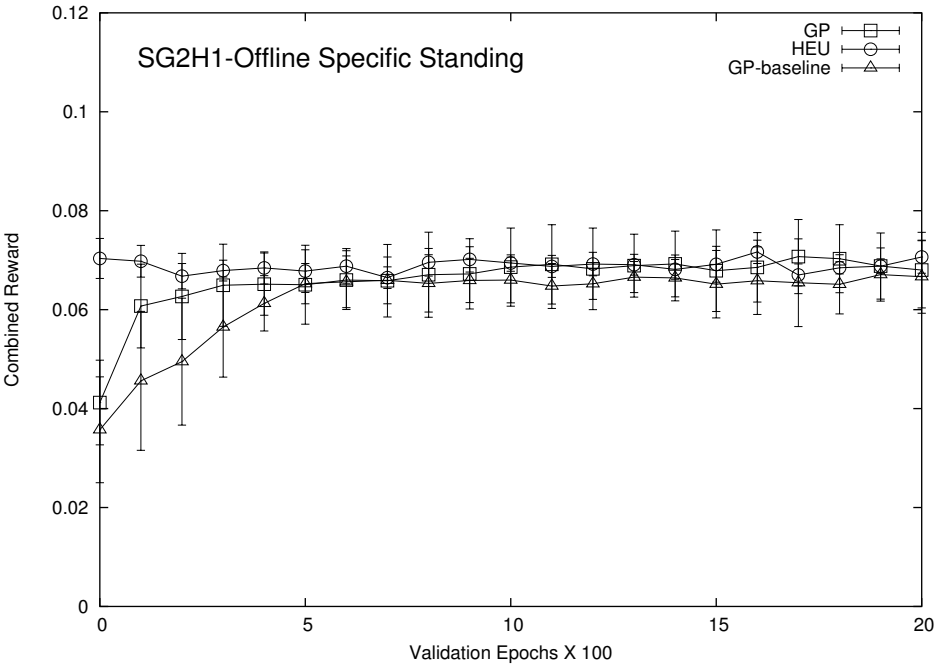

Figure C.8: Average evolution of the combined reward in experiment SG2H1-Offline Specific Standing.

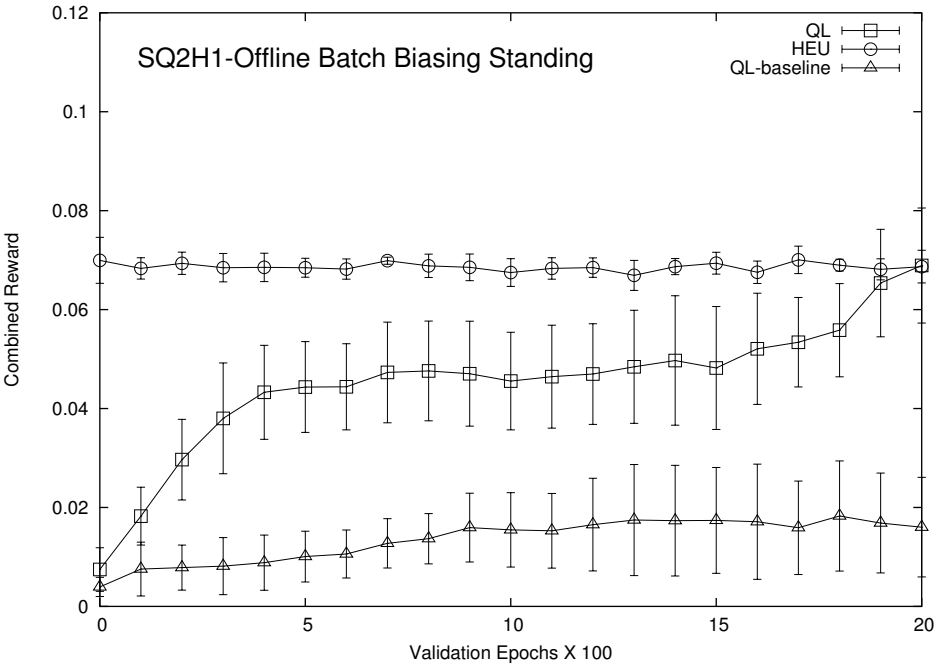

Figure C.9: Average evolution of the combined reward in experiment SQ2H1-Offline Batch Biasing Standing.

Table C.6: Results of the final validation cycle for experiment SQ2H1-Offline Batch Biasing Standing.

Agent	Avg	Std Dev
HEU	0.0686907	0.00330321
QL	0.0689109	0.0116386

Table C.7: Results of the final validation cycle for experiment SQ2H1-Offline Batch Virtual-Experience Standing.

Agent	Avg	Std Dev
HEU	0.0683005	0.00314918
QL	0.0792305	0.0103017

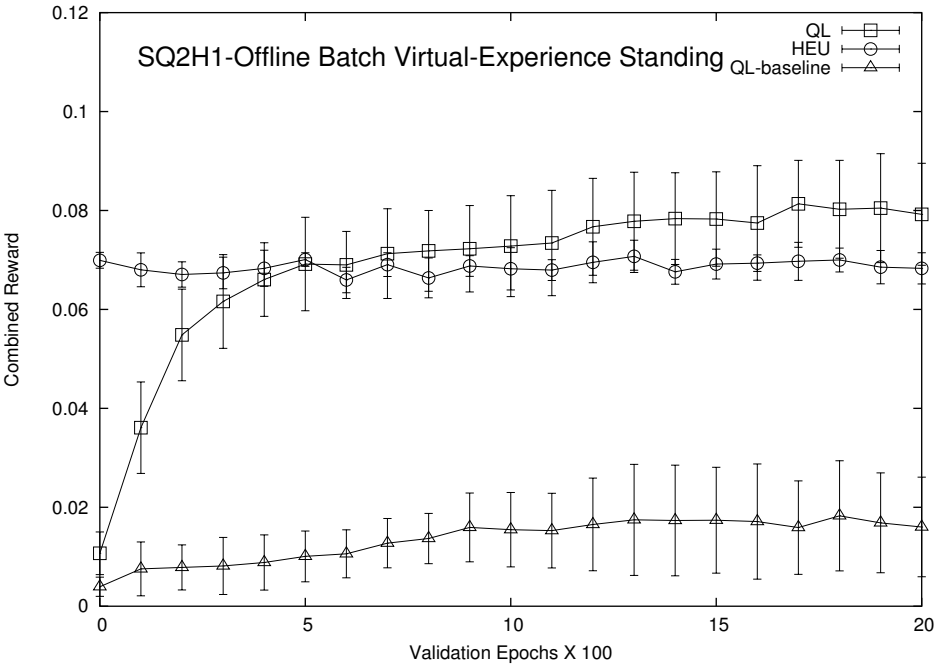

Figure C.10: Average evolution of the combined reward in experiment SQ2H1-Offline Batch Virtual-Experience Standing.

Table C.8: Results of the final validation cycle for experiment SQ2H1-Offline Specific Biasing Standing.

Agent	Avg	Std Dev
HEU	0.0685482	0.00148758
QL	0.0725953	0.0143903

Table C.9: Results of the final validation cycle for experiment SQ2H1-Online Specific Biasing Standing.

Agent	Avg	Std Dev
HEU	0.0703686	0.00312342
QL	0.0165019	0.00964853

Table C.10: Results of the final validation cycle for experiment SQ2H1-Online Specific Imitation Standing.

Agent	Avg	Std Dev
HEU	0.0701527	0.00283177
QL	0.0761233	0.00735551

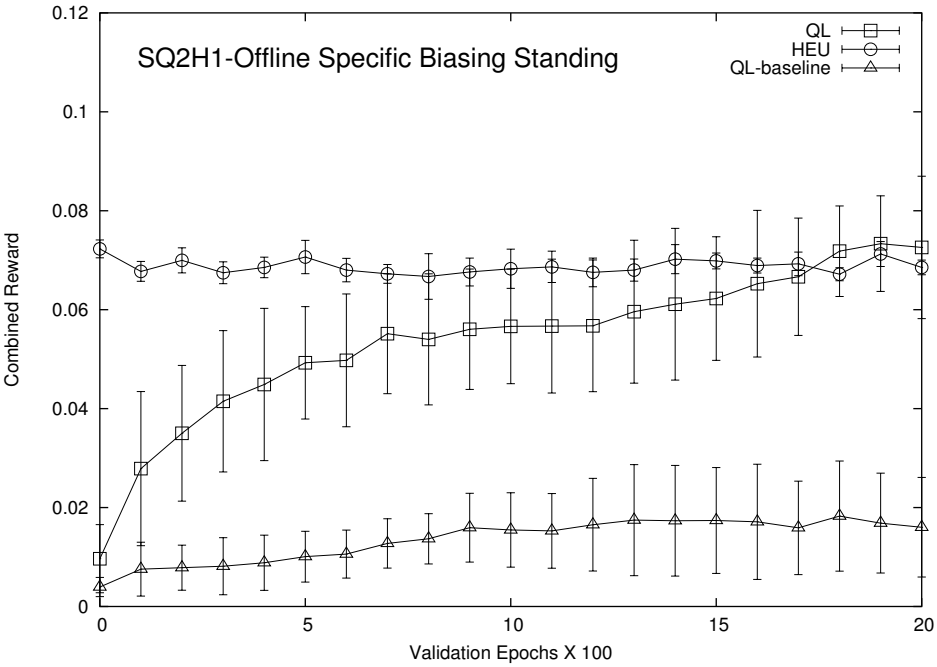

Figure C.11: Average evolution of the combined reward in experiment SQ2H1-Offline Specific Biasing Standing.

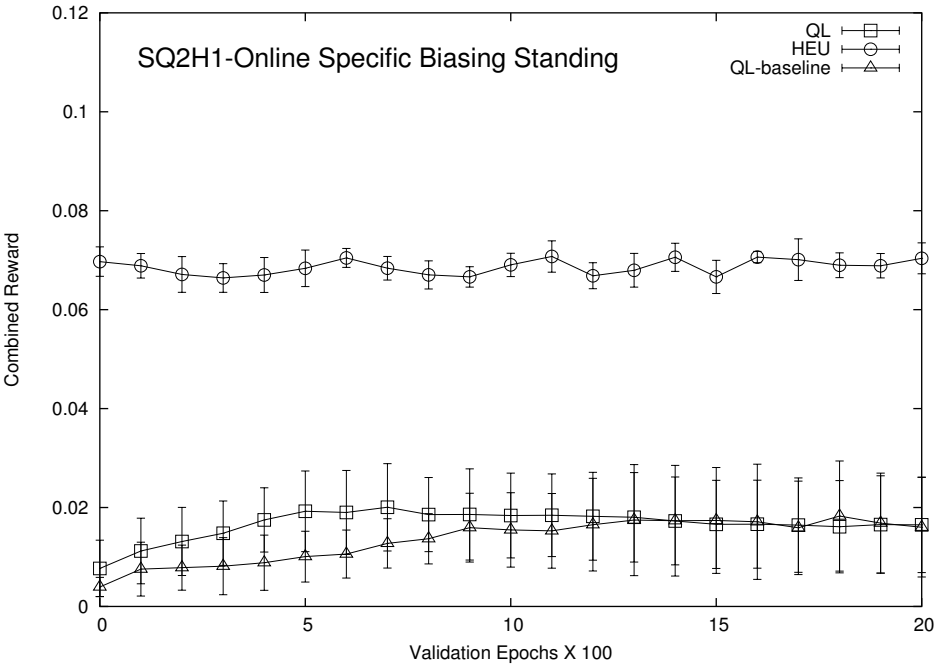

Figure C.12: Average evolution of the combined reward in experiment SQ2H1-Online Specific Biasing Standing.

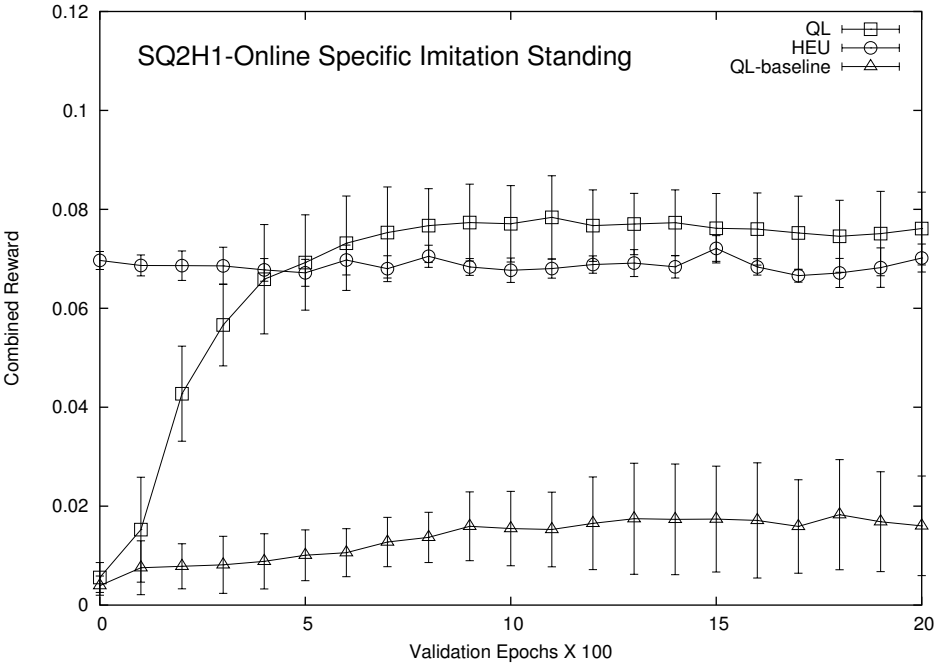

Figure C.13: Average evolution of the combined reward in experiment SQ2H1-Online Specific Imitation Standing.

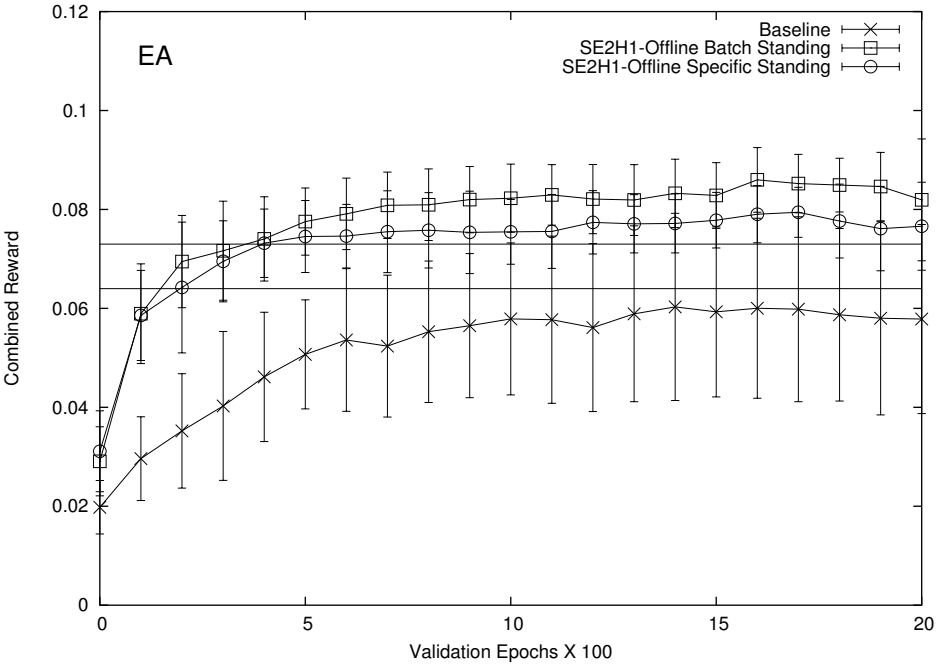

Figure C.14: Summary of results for EA-agents in Experiment Set 1.

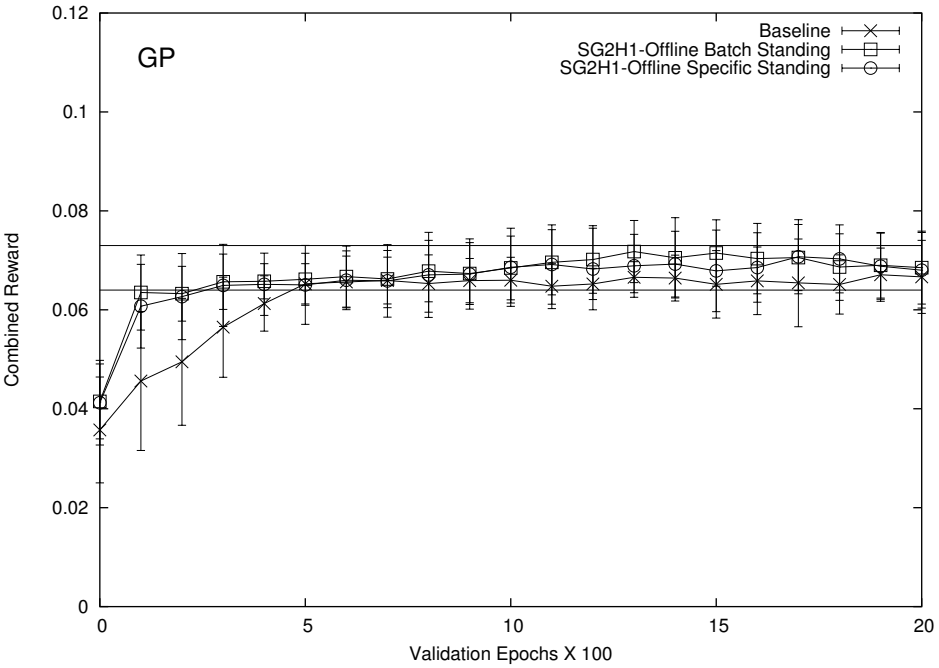

Figure C.15: Summary of results for GP-agents in Experiment Set 1.

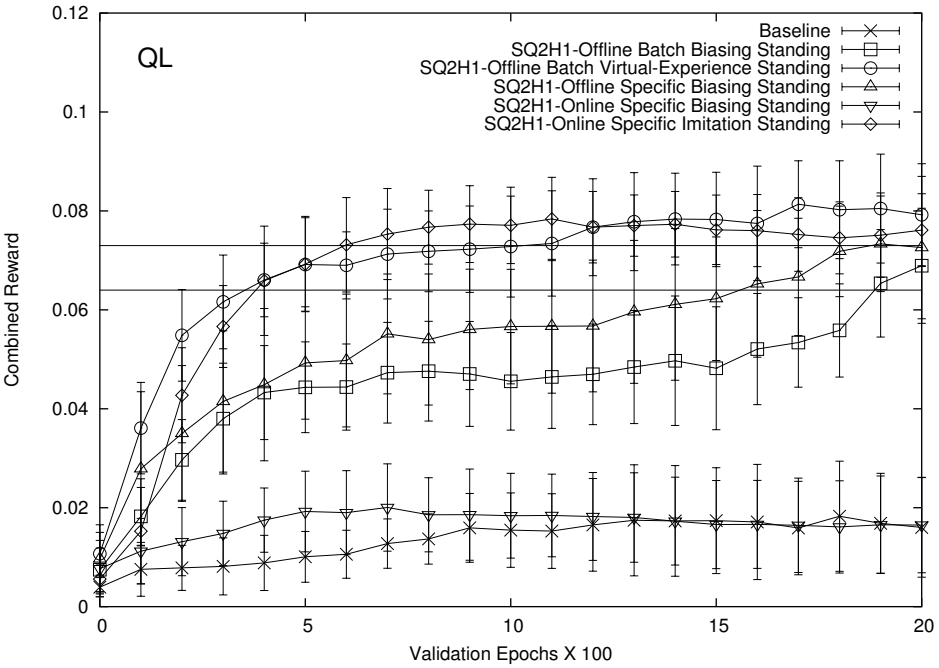

Figure C.16: Summary of results for QL-agents in Experiment Set 1.

C.3 Experiment Set 2

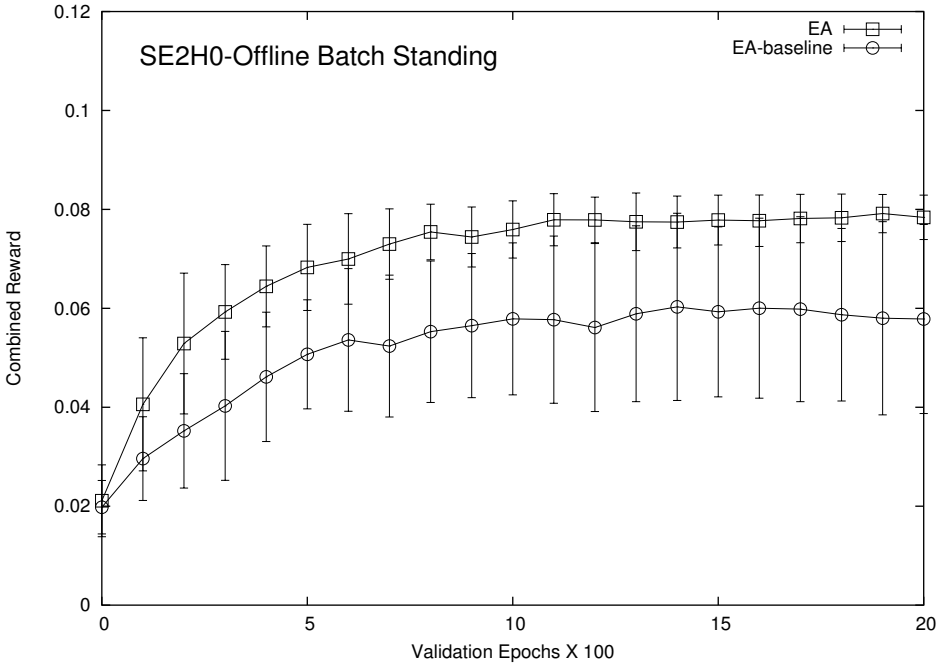

Figure C.17: Average evolution of the combined reward in experiment SE2H0-Offline Batch Standing.

Table C.11: Results of the final validation cycle for experiment SE2H0-Offline Batch Standing.

Agent	Avg	Std Dev
EA	0.0783934	0.00448353

Table C.12: Results of the final validation cycle for experiment SG2H0-Offline Batch Standing.

Agent	Avg	Std Dev
GP	0.0705565	0.00624749

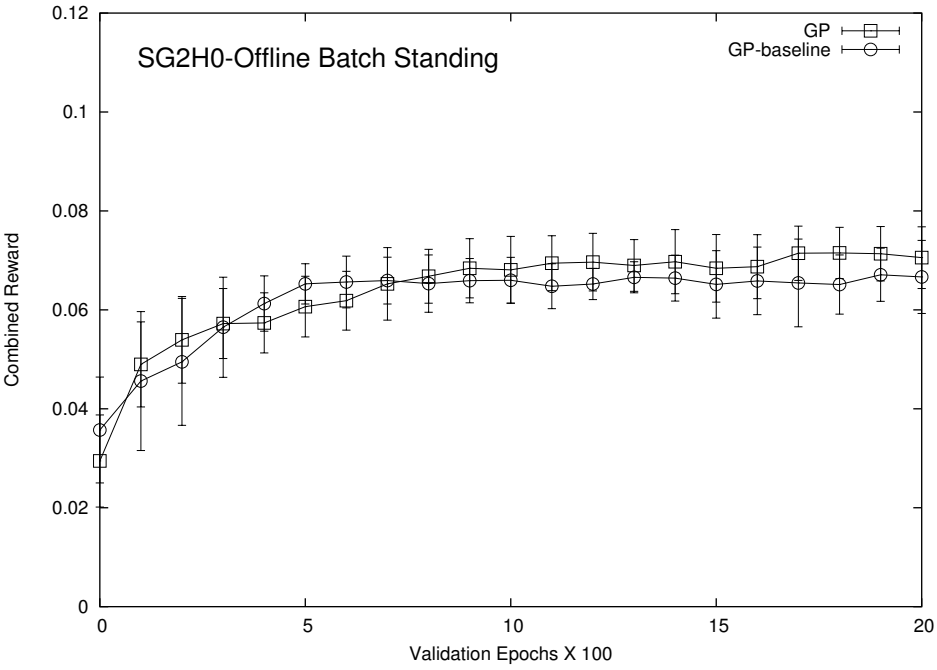

Figure C.18: Average evolution of the combined reward in experiment SG2H0-Offline Batch Standing.

Table C.13: Results of the final validation cycle for experiment SM2H0-Multiple Advisors.

Agent	Avg	Std Dev
QL	0.0771157	0.00786617
EA	0.0773864	0.00788732
GP	0.0717317	0.0079441

Table C.14: Results of the final validation cycle for experiment SM2H0-Standing Advisor.

Agent	Avg	Std Dev
QL	0.0784898	0.00627837
EA	0.0781268	0.00431552
GP	0.071124	0.0090083

Table C.15: Results of the final validation cycle for experiment SQ2H0-Offline Batch Virt-Exp- Standing.

Agent	Avg	Std Dev
QL	0.0789844	0.00828043

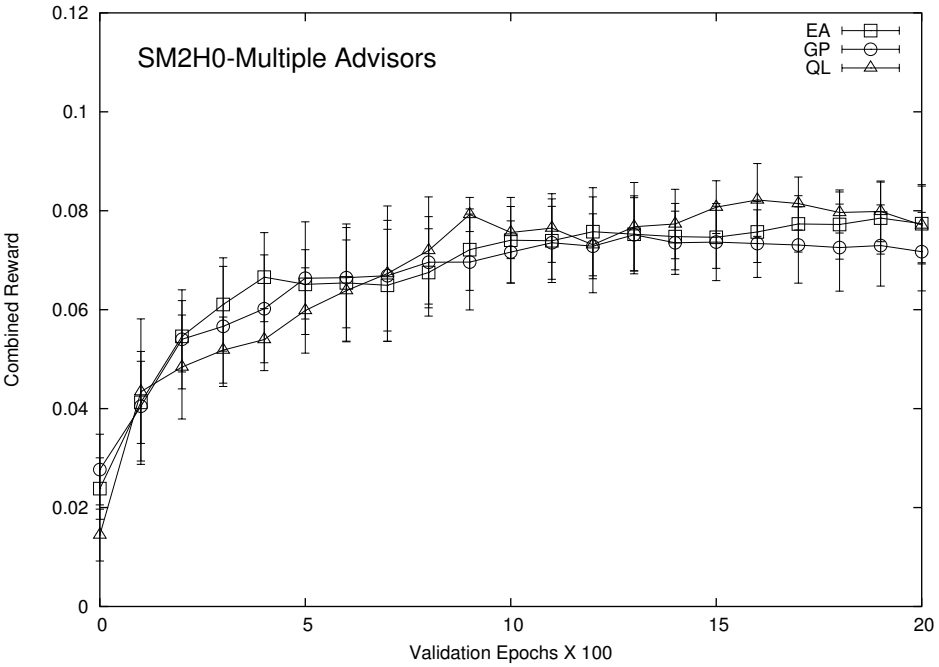

Figure C.19: Average evolution of the combined reward in experiment SM2H0-Multiple Advisors.

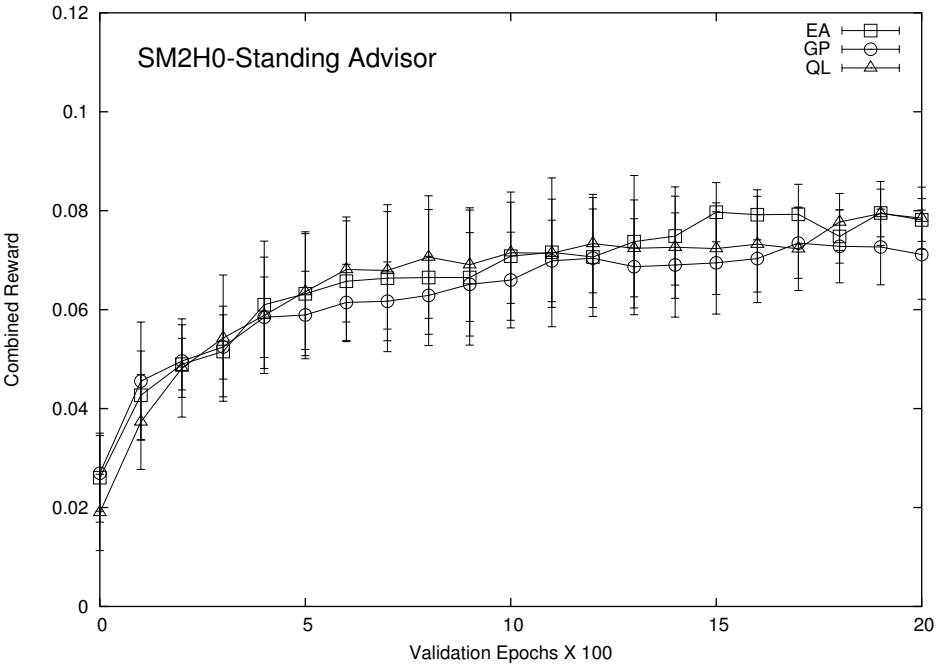

Figure C.20: Average evolution of the combined reward in experiment SM2H0-Standing Advisor.

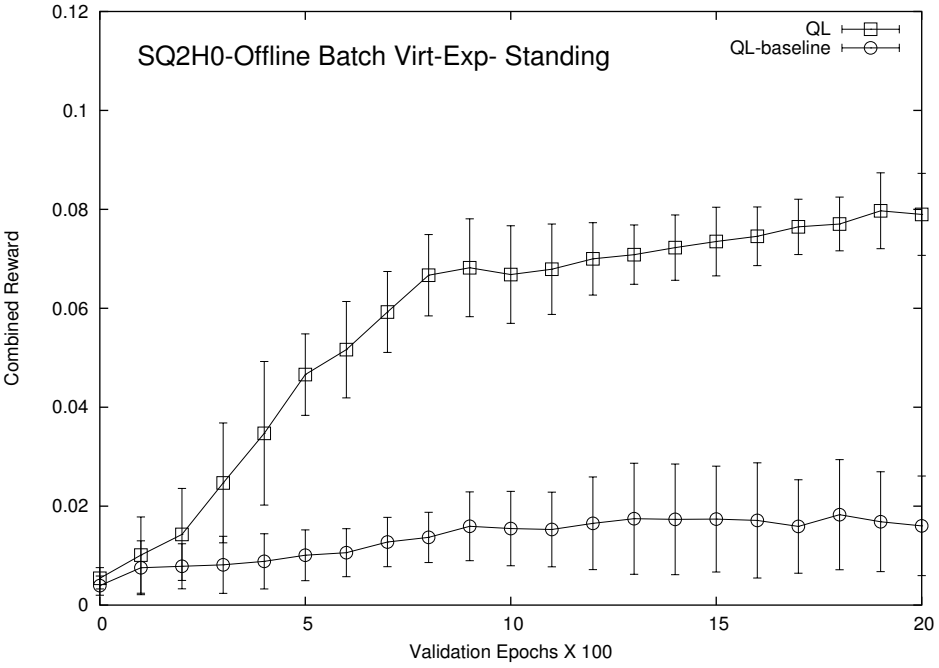

Figure C.21: Average evolution of the combined reward in experiment SQ2H0-Offline Batch Virt-Exp- Standing.

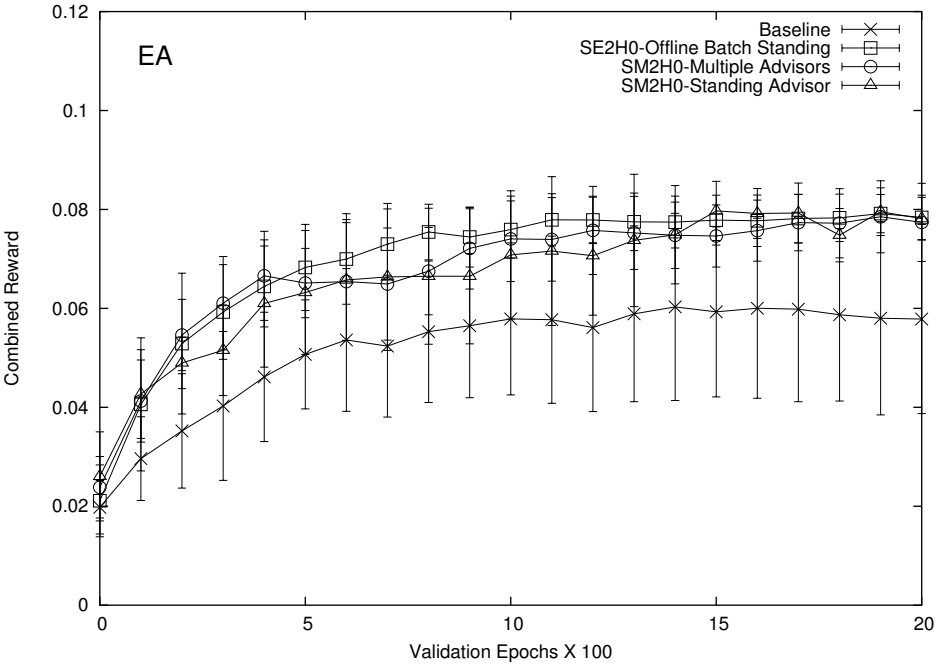

Figure C.22: Summary of results for EA-agents in Experiment Set 2.

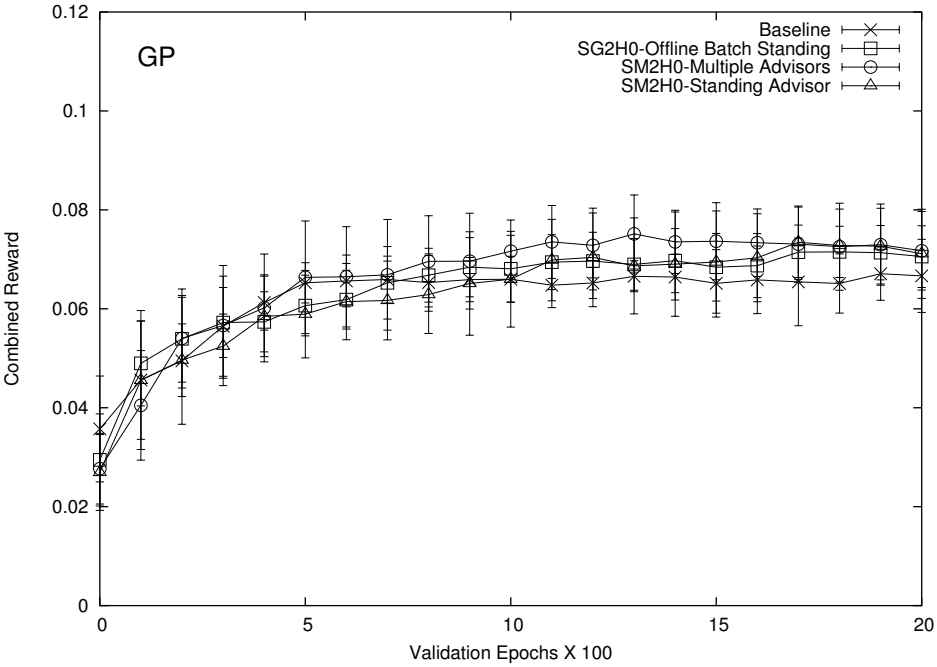

Figure C.23: Summary of results for GP-agents in Experiment Set 2.

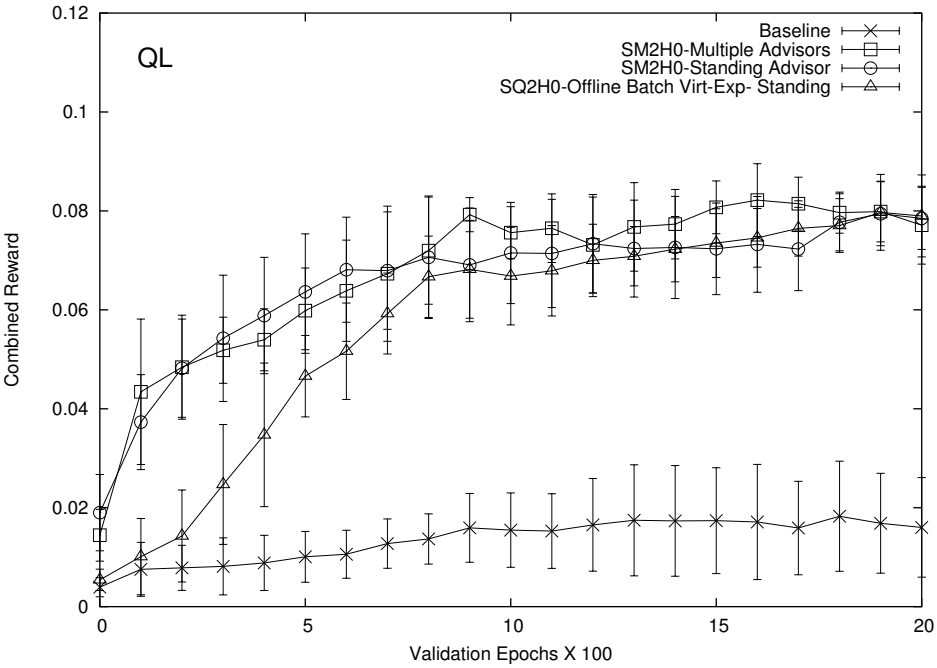

Figure C.24: Summary of results for QL-agents in Experiment Set 2.

C.4 Experiment Set 3

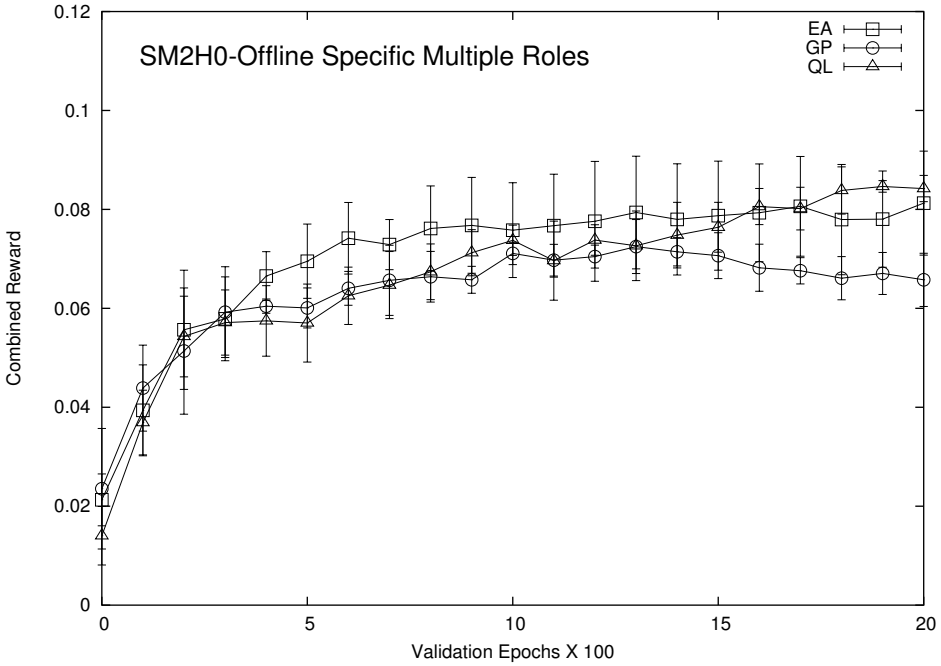

Figure C.25: Average evolution of the combined reward in experiment SM2H0-Offline Specific Multiple Roles.

Table C.16: Results of the final validation cycle for experiment SM2H0-Offline Specific Multiple Roles.

Agent	Avg	Std Dev
QL	0.0842081	0.00265709
EA	0.0812713	0.0105108
GP	0.0657592	0.00537815

Table C.17: Results of the final validation cycle for experiment SM2H0-Offline Specific Multiple Roles Trust.

Agent	Avg	Std Dev
QL	0.0713236	0.0100503
EA	0.0724886	0.00559709
GP	0.0654637	0.00531979

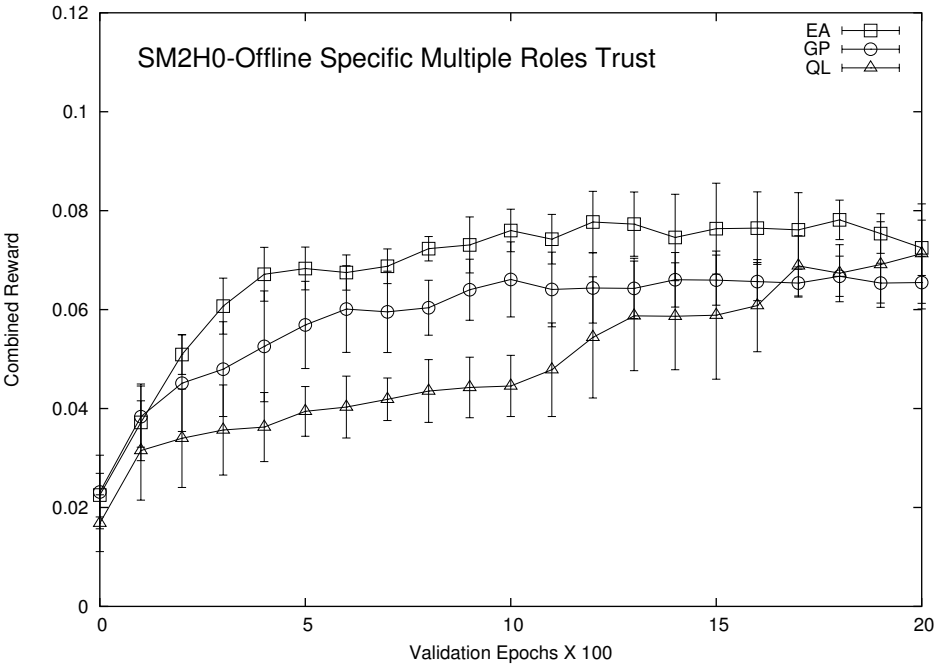

Figure C.26: Average evolution of the combined reward in experiment SM2H0-Offline Specific Multiple Roles Trust.

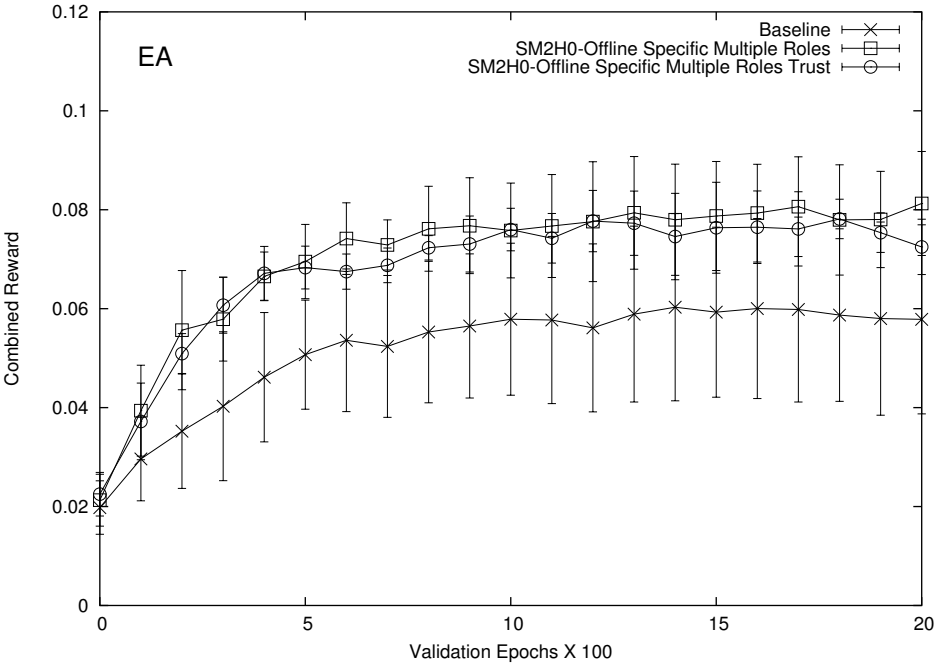

Figure C.27: Summary of results for EA-agents in Experiment Set 3.

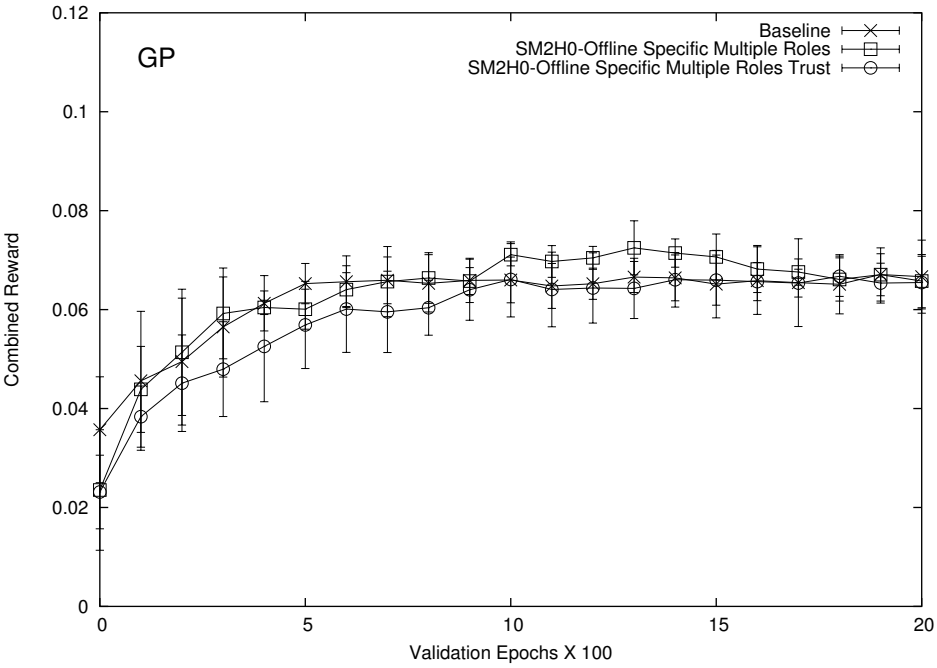

Figure C.28: Summary of results for GP-agents in Experiment Set 3.

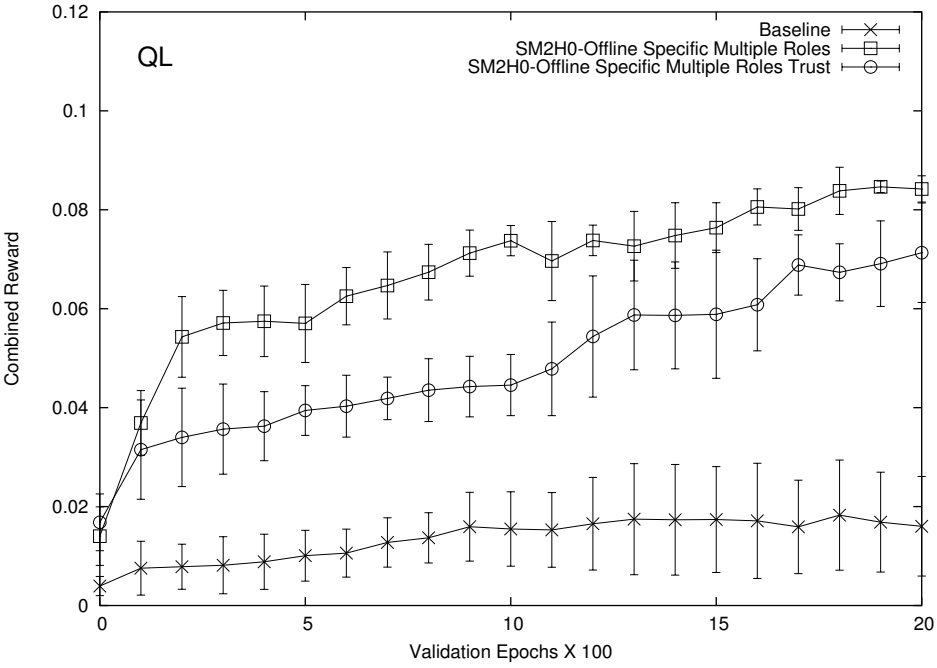

Figure C.29: Summary of results for QL-agents in Experiment Set 3.

C.5 Experiment Set 4

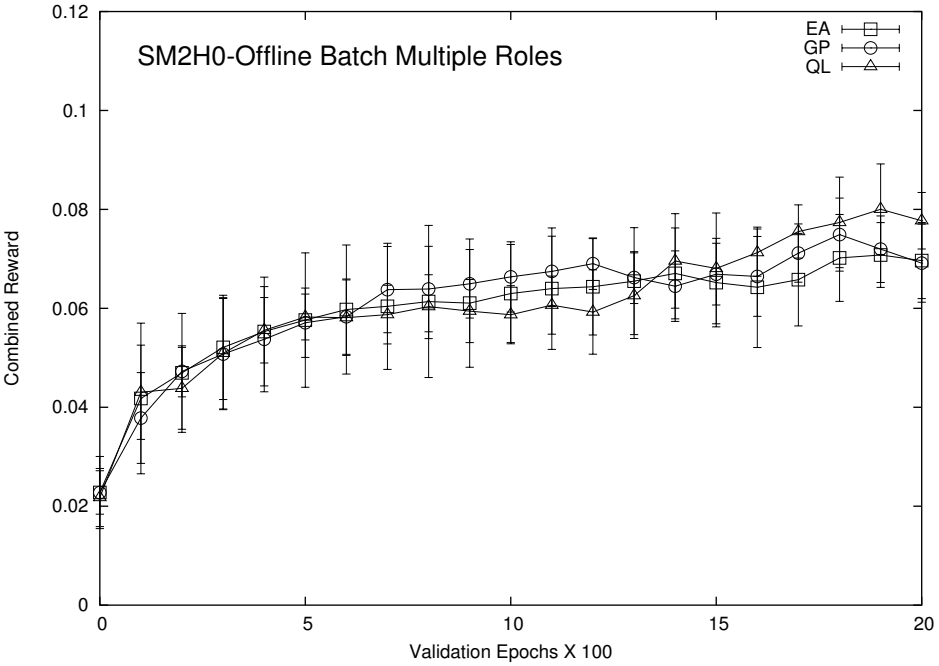

Figure C.30: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Roles.

Table C.18: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Roles.

Agent	Avg	Std Dev
QL	0.0777089	0.00571074
EA	0.0696568	0.00769363
GP	0.0691497	0.0078972

Table C.19: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Trust.

Agent	Avg	Std Dev
QL	0.0752682	0.00728397
EA	0.076441	0.00640996
GP	0.0658598	0.00486845

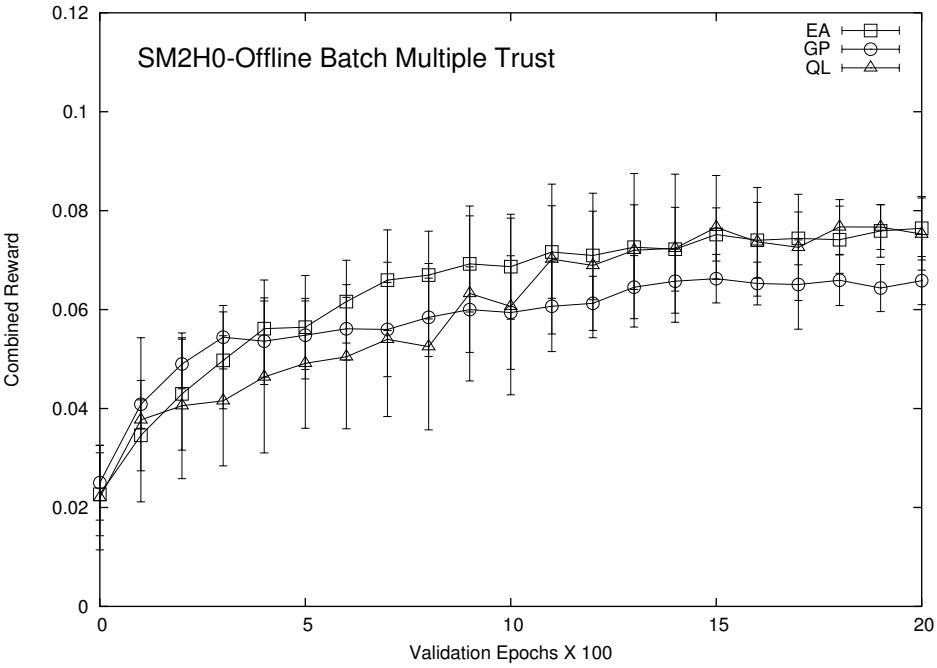

Figure C.31: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Trust.

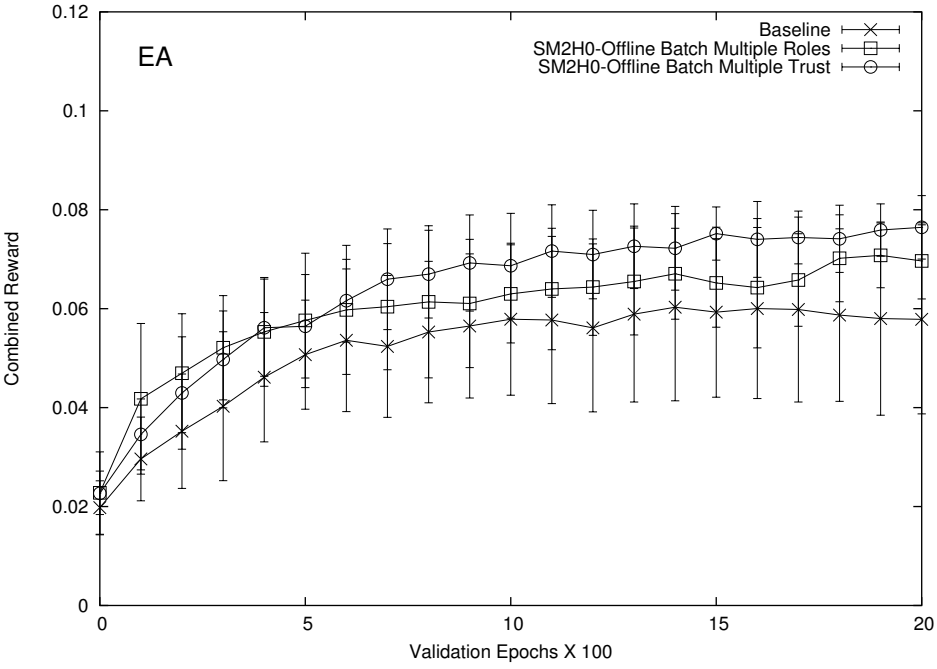

Figure C.32: Summary of results for EA-agents in Experiment Set 4.

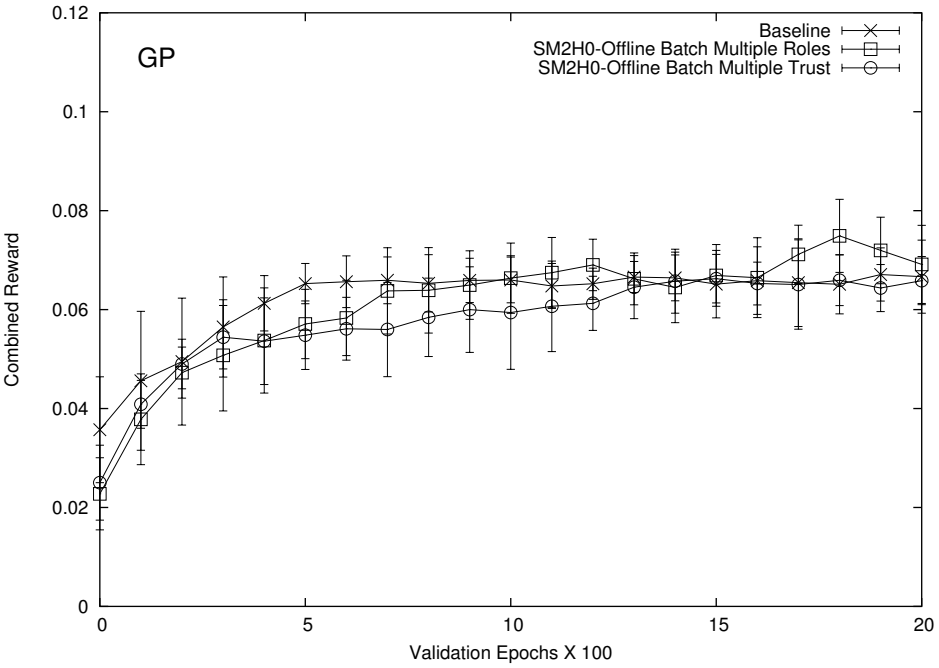

Figure C.33: Summary of results for GP-agents in Experiment Set 4.

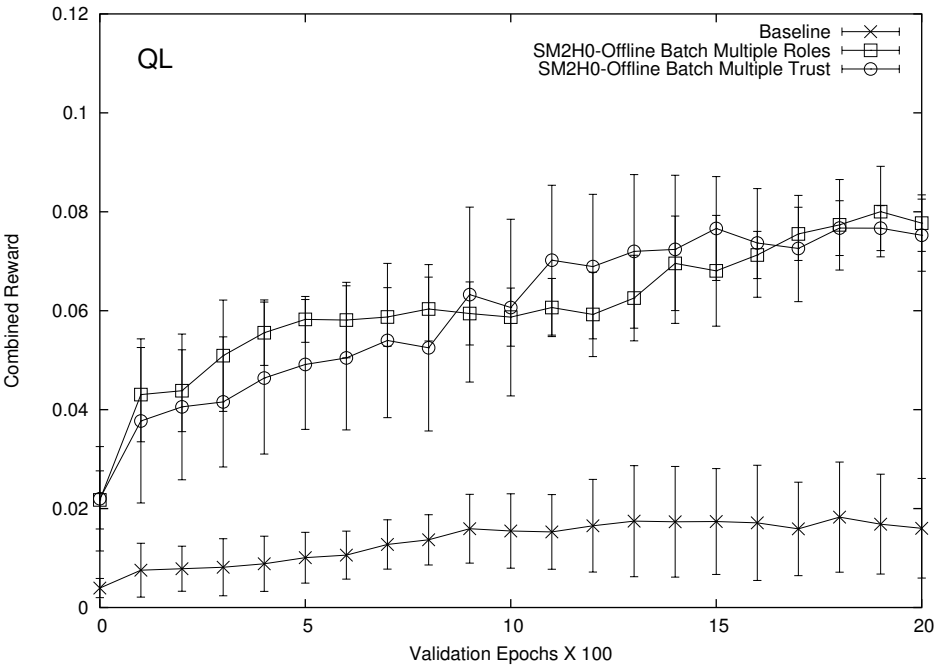

Figure C.34: Summary of results for QL-agents in Experiment Set 4.

C.6 Experiment Set 5

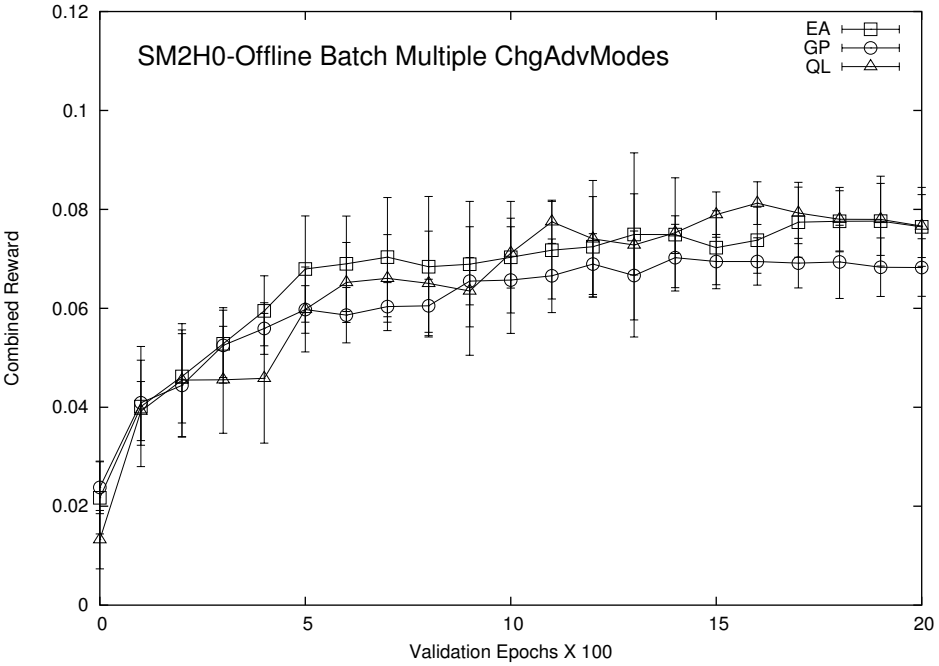

Figure C.35: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Changing Advice Modes.

Table C.20: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Changing Advice Modes.

Agent	Avg	Std Dev
QL	0.0766297	0.00635994
EA	0.0764721	0.00797502
GP	0.0682434	0.0058265

Table C.21: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Changing Learning Parameters.

Agent	Avg	Std Dev
QL	0.0802154	0.00470469
EA	0.0699507	0.0161411
GP	0.0665043	0.0110998

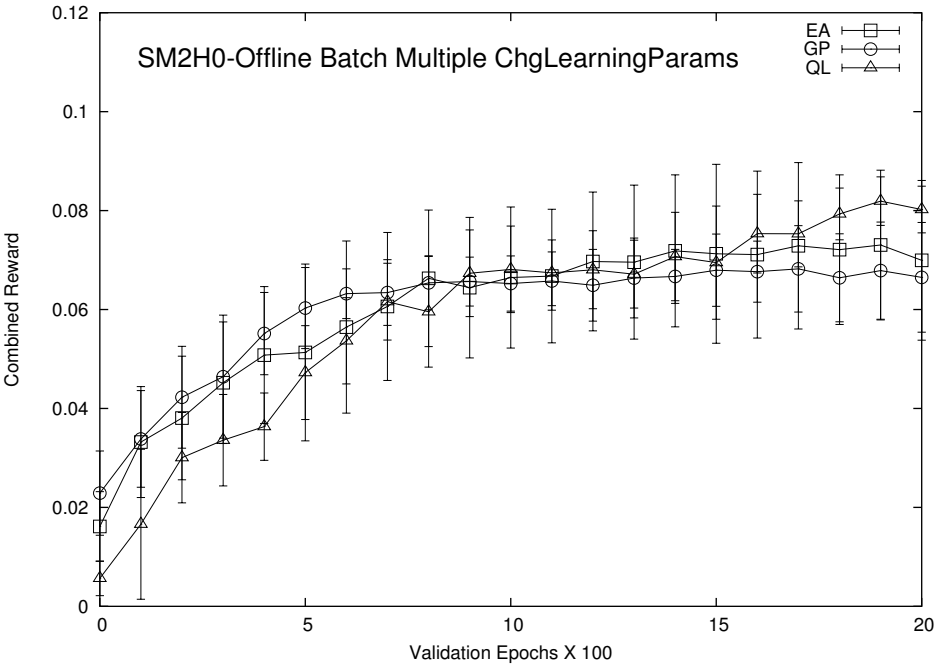

Figure C.36: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Changing Learning Parameters.

Table C.22: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
QL	0.0710992	0.0139757
EA	0.0660738	0.0075965
GP	0.0622365	0.00847292

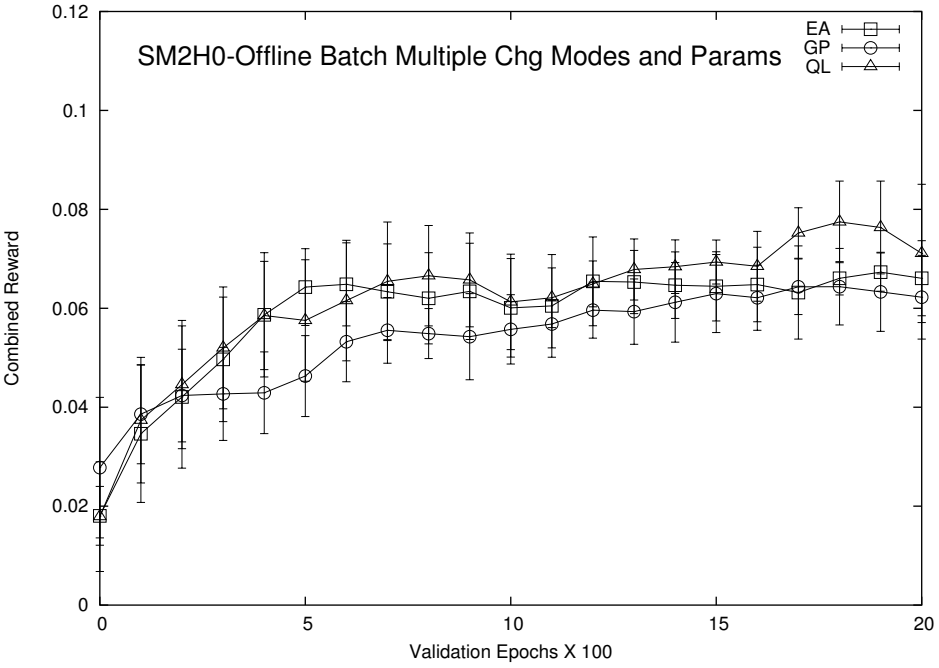

Figure C.37: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Changing Advice Modes and Parameters.

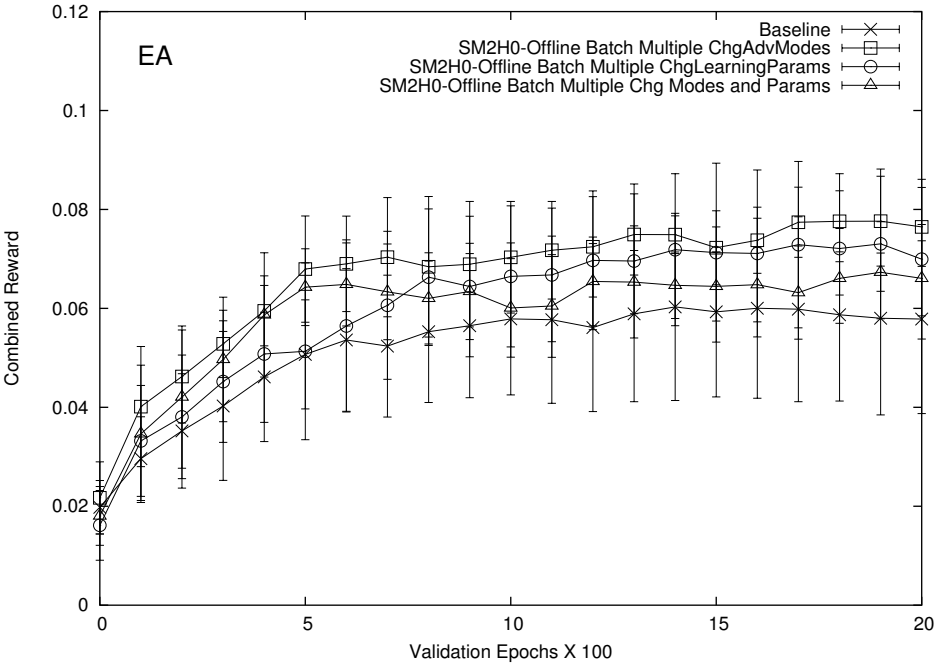

Figure C.38: Summary of results for EA-agents in Experiment Set 5.

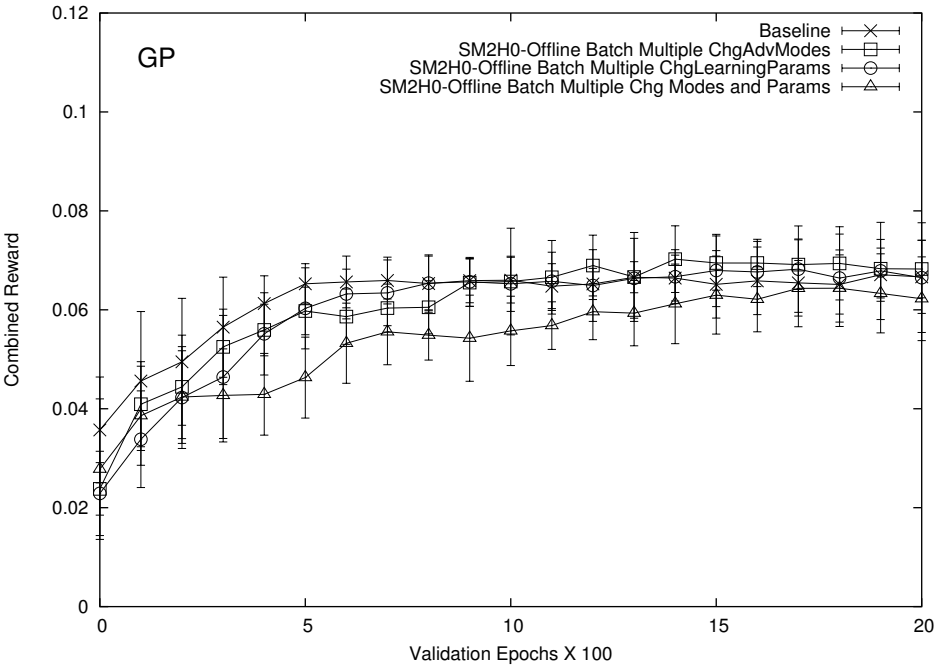

Figure C.39: Summary of results for GP-agents in Experiment Set 5.

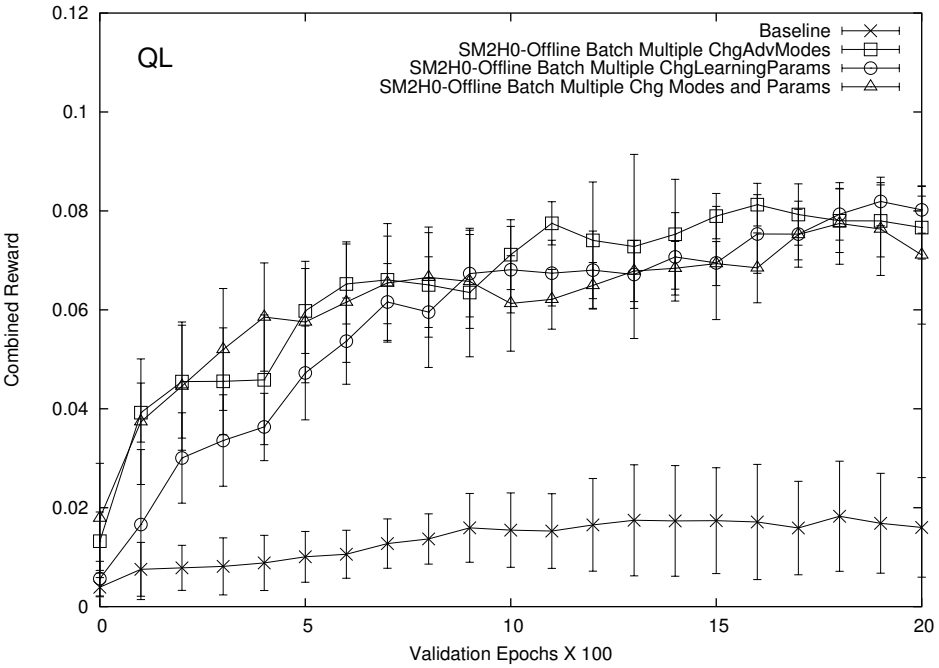

Figure C.40: Summary of results for QL-agents in Experiment Set 5.

C.7 Experiment Set 6

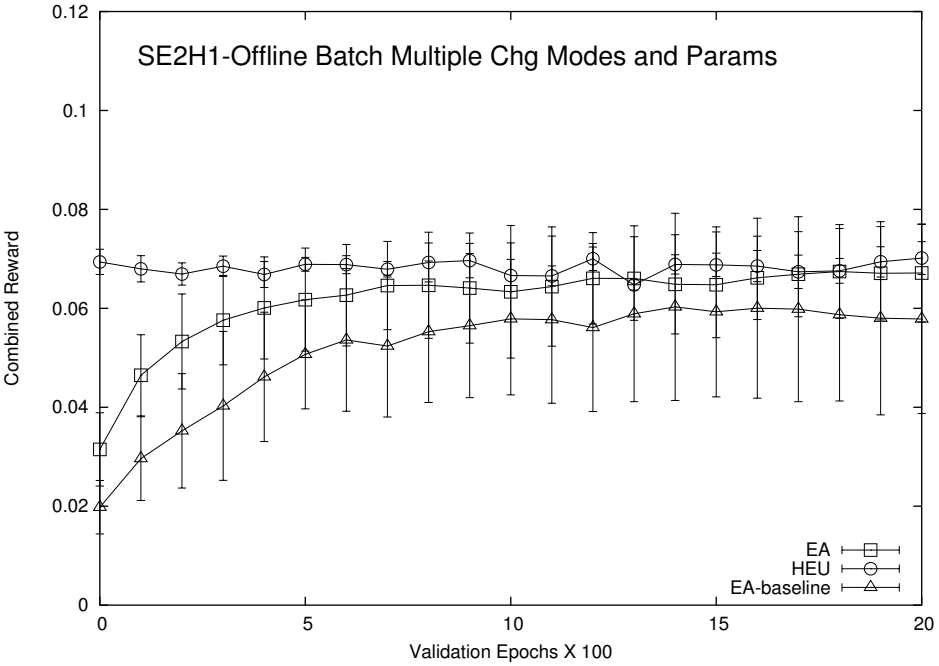

Figure C.41: Average evolution of the combined reward in experiment SE2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Table C.23: Results of the final validation cycle for experiment SE2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.0701684	0.00332053
EA	0.0671555	0.00989725

Table C.24: Results of the final validation cycle for experiment SG2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.0685155	0.00292811
GP	0.066252	0.00575137

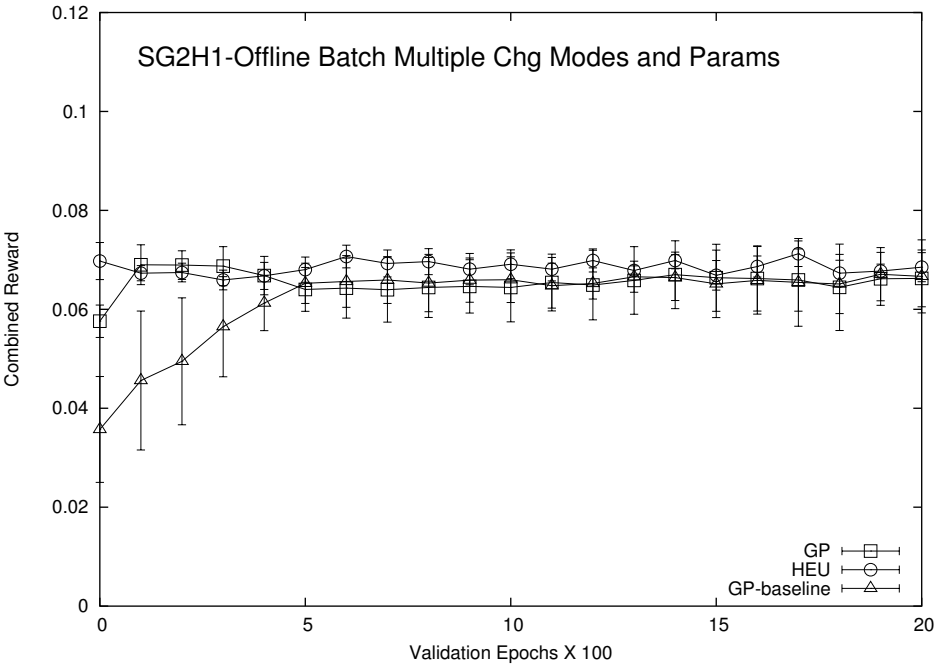

Figure C.42: Average evolution of the combined reward in experiment SG2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Table C.25: Results of the final validation cycle for experiment SM2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.0708408	0.00219108
QL	0.0834194	0.00850732
EA	0.0649986	0.0142614
GP	0.0709161	0.00550188

Table C.26: Results of the final validation cycle for experiment SQ2H1-Offline Batch Virt-Exp- Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.0682267	0.00310873
QL	0.0823297	0.00981624

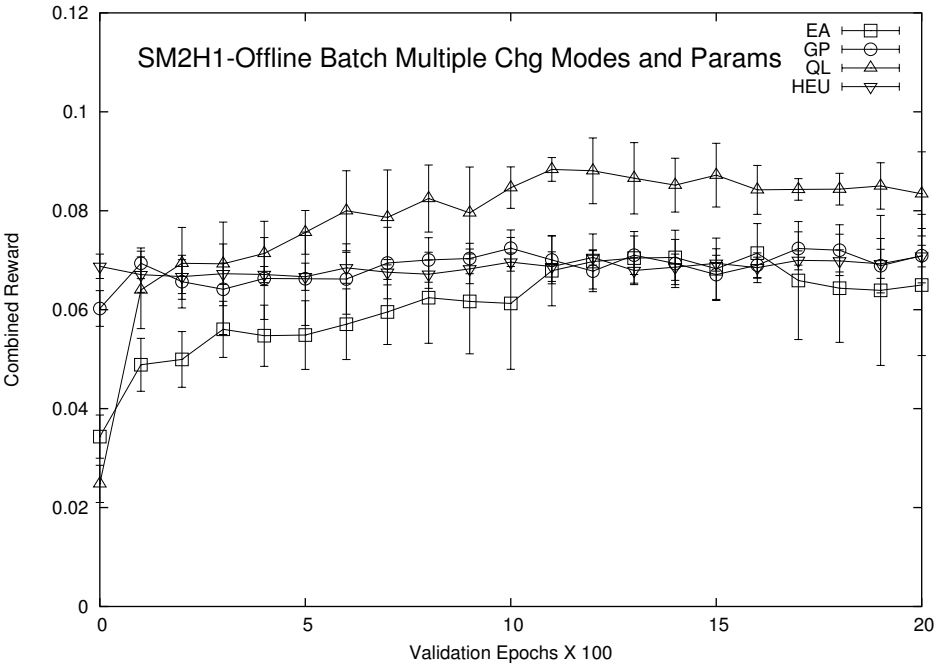

Figure C.43: Average evolution of the combined reward in experiment SM2H1-Offline Batch Multiple Changing Advice Modes and Parameters.

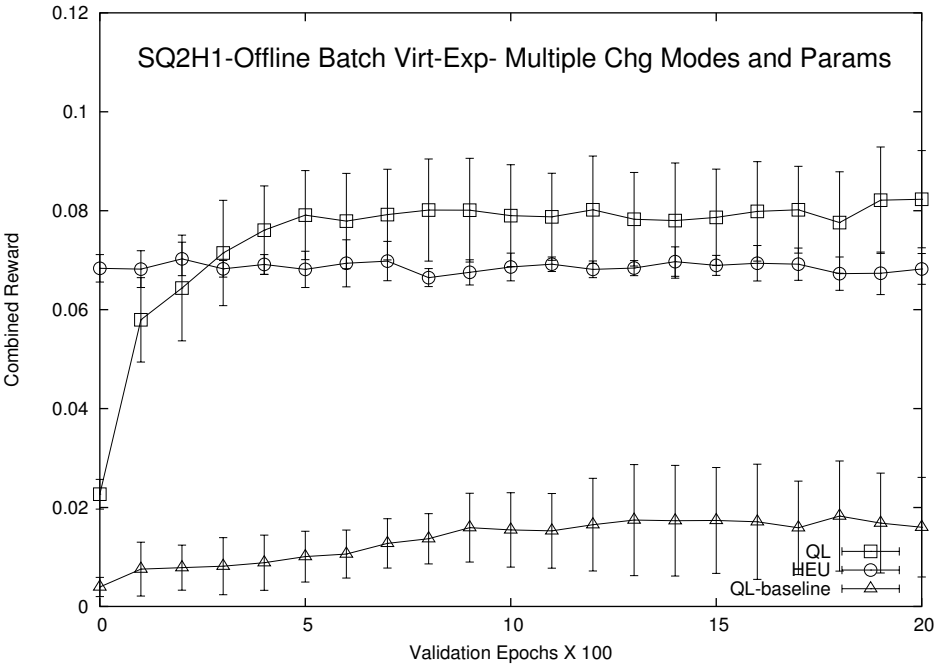

Figure C.44: Average evolution of the combined reward in experiment SQ2H1-Offline Batch Virt-Exp- Multiple Changing Advice Modes and Parameters.

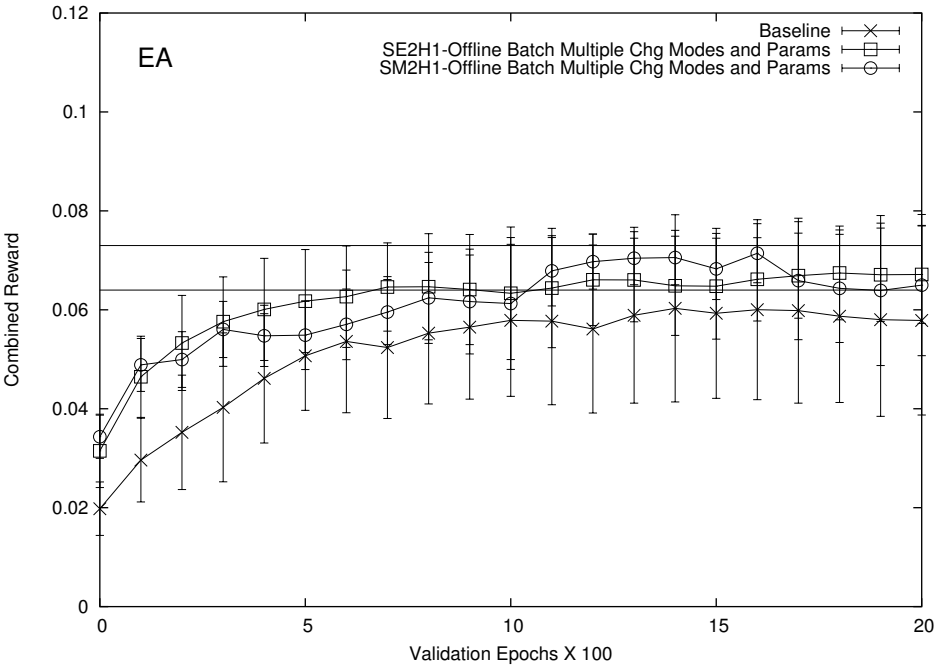

Figure C.45: Summary of results for EA-agents in Experiment Set 6.

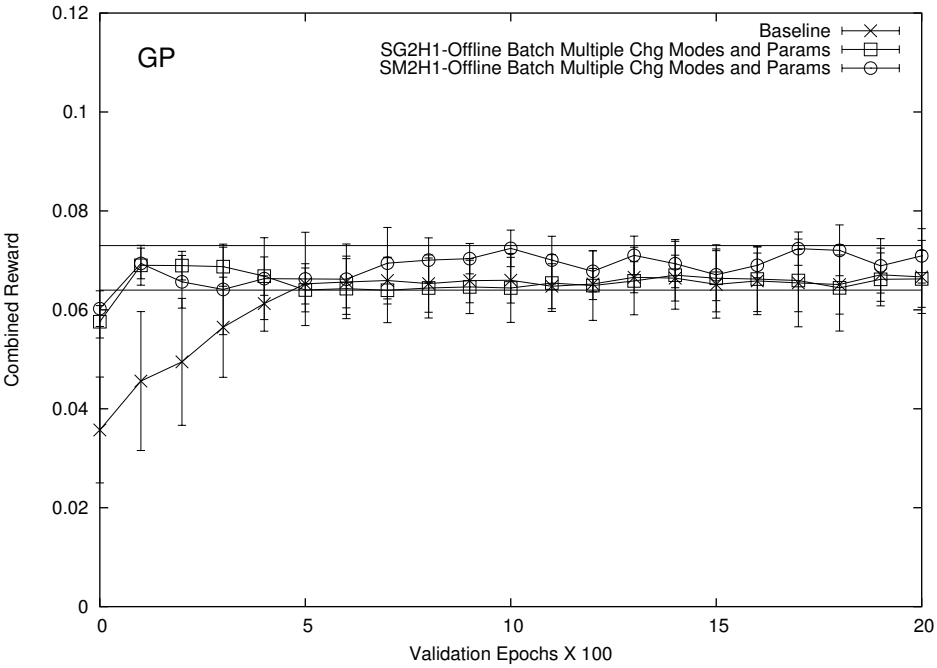

Figure C.46: Summary of results for GP-agents in Experiment Set 6.

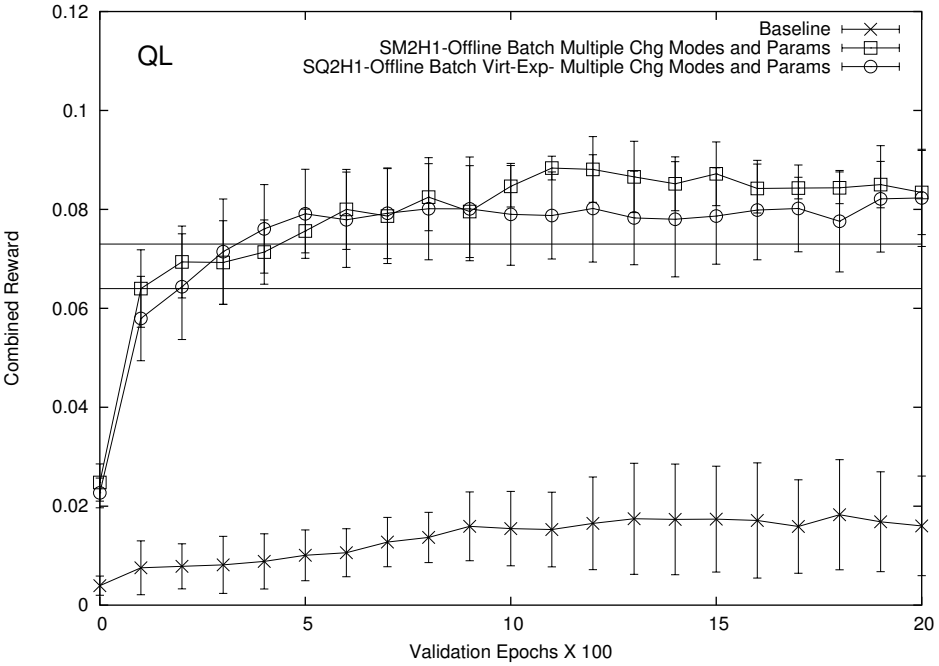

Figure C.47: Summary of results for QL-agents in Experiment Set 6.

C.8 Traffic Baseline Tests

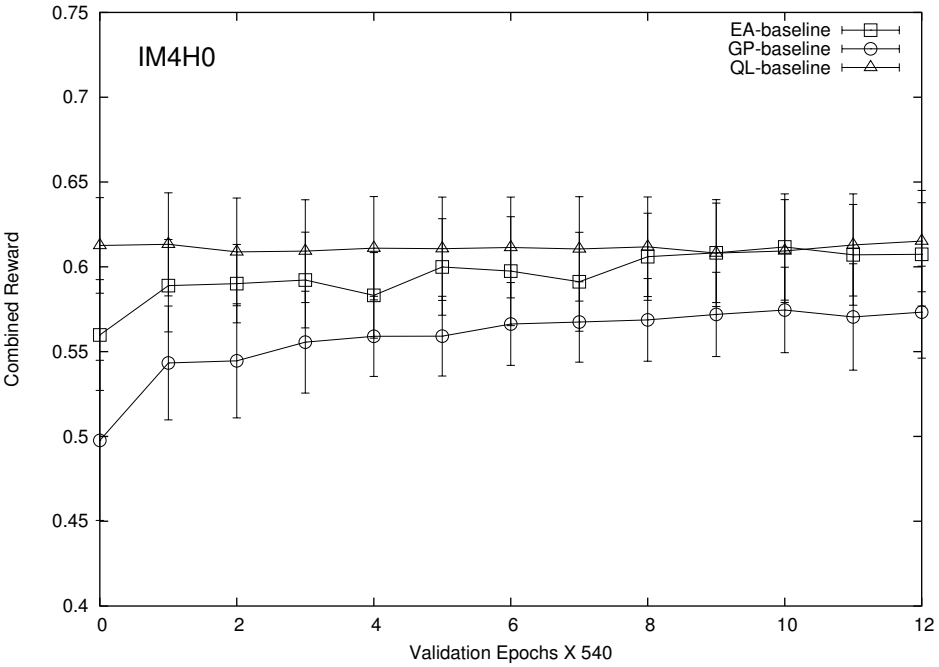

Figure C.48: Average evolution of the combined reward in the baseline Experiment Set for the Traffic-Control problem.

Table C.27: Results of the final validation cycle for the baseline Experiment Set for the Traffic-Control problem.

Agent	Avg	Std Dev
QL	0.615148	0.0298644
EA	0.607371	0.0304795
GP	0.573297	0.0271324

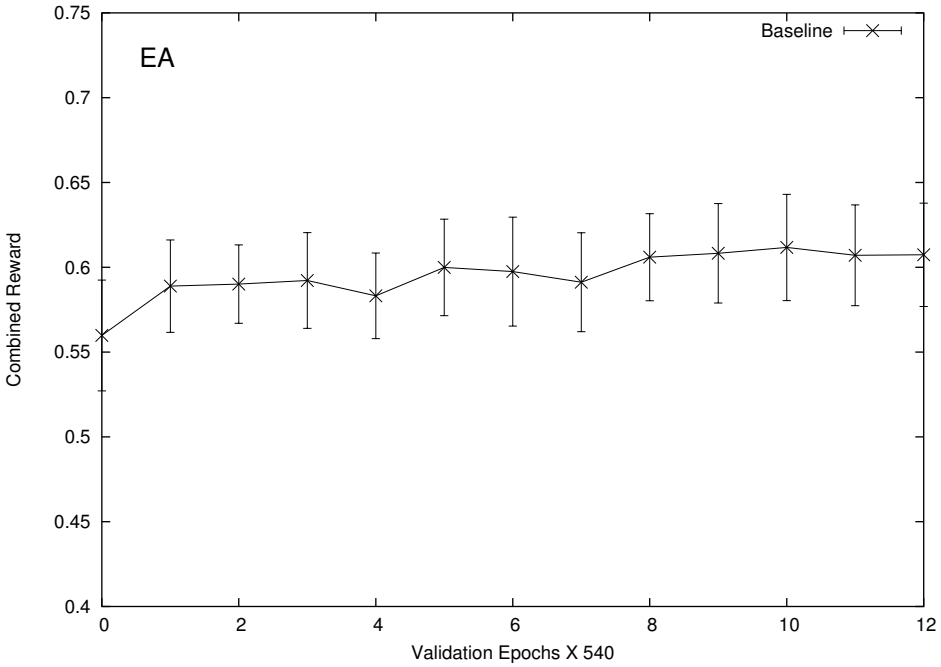

Figure C.49: Summary of results for EA-agents in the baseline Experiment Set for the Traffic-Control problem.

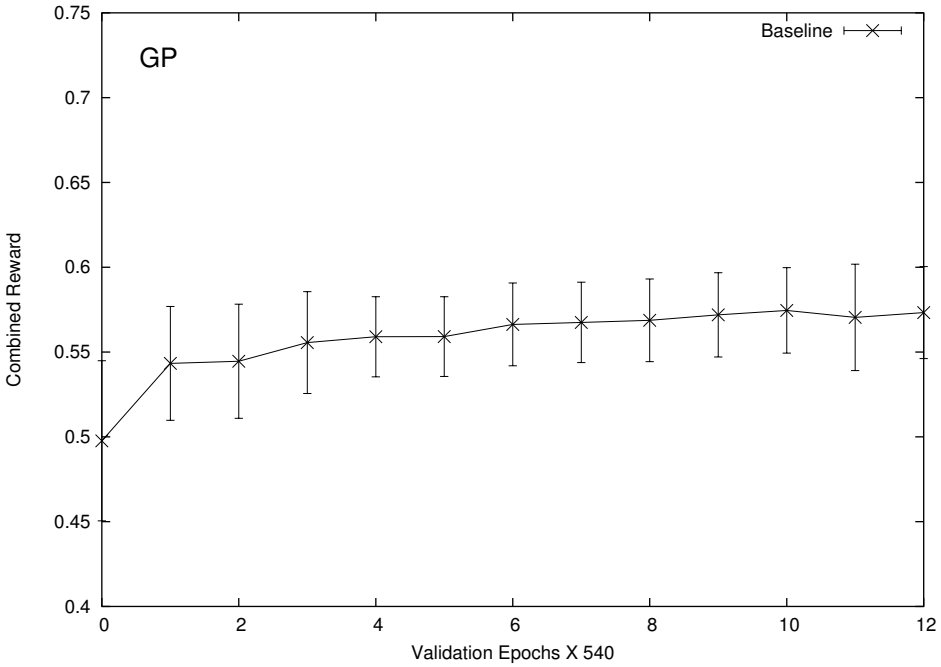

Figure C.50: Summary of results for GP-agents in the baseline Experiment Set for the Traffic-Control problem.

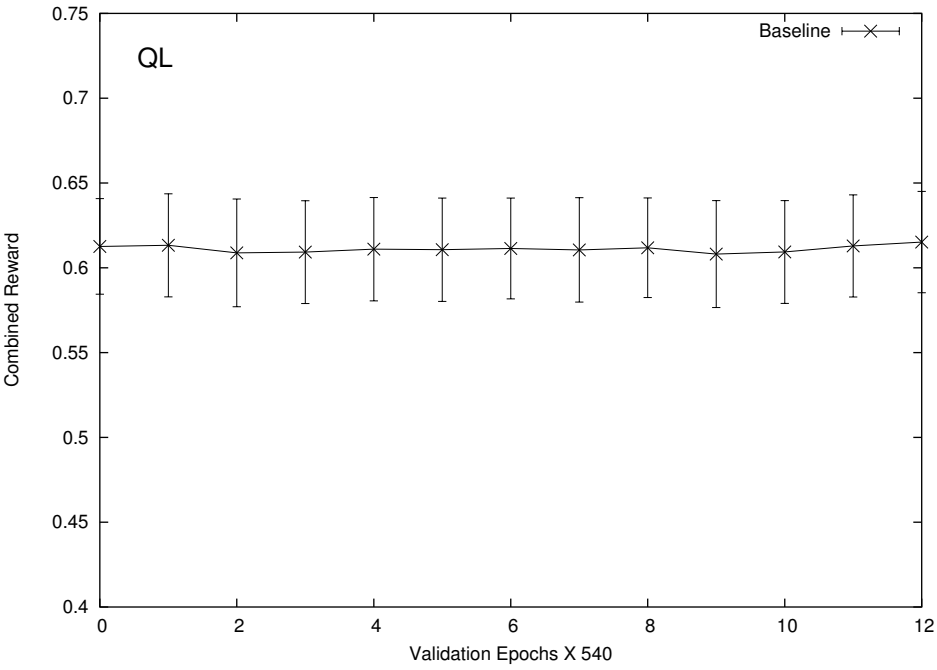

Figure C.51: Summary of results for QL-agents in the baseline Experiment Set for the Traffic-Control problem.

C.9 Experiment Set 7

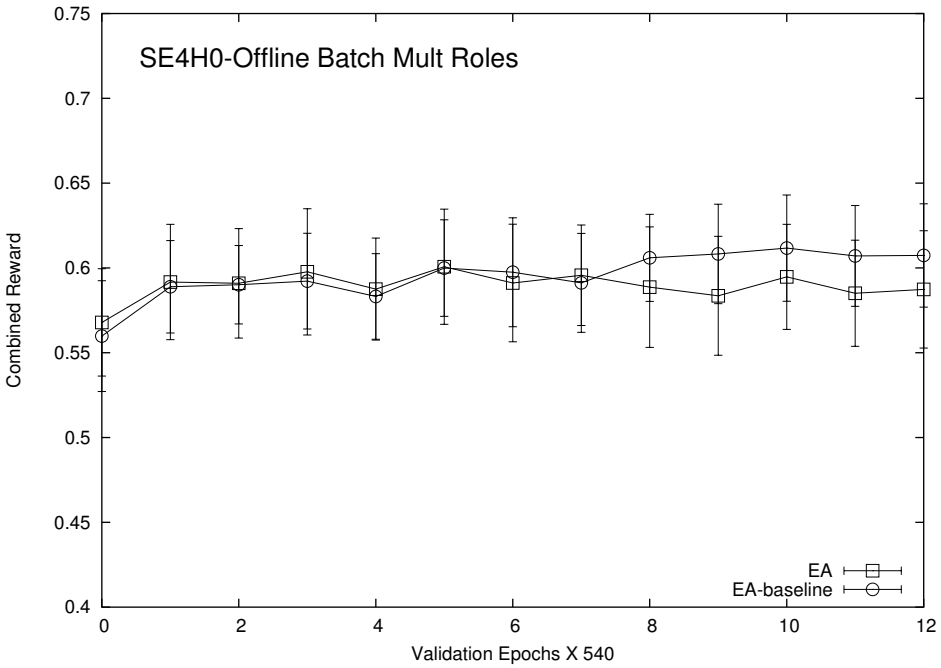

Figure C.52: Average evolution of the combined reward in experiment SE4H0-Offline Batch Multiple Roles.

Table C.28: Results of the final validation cycle for experiment SE4H0-Offline Batch Multiple Roles.

Agent	Avg	Std Dev
EA	0.587314	0.0346009

Table C.29: Results of the final validation cycle for experiment SG4H0-Offline Batch Multiple Roles.

Agent	Avg	Std Dev
GP	0.591071	0.0328854

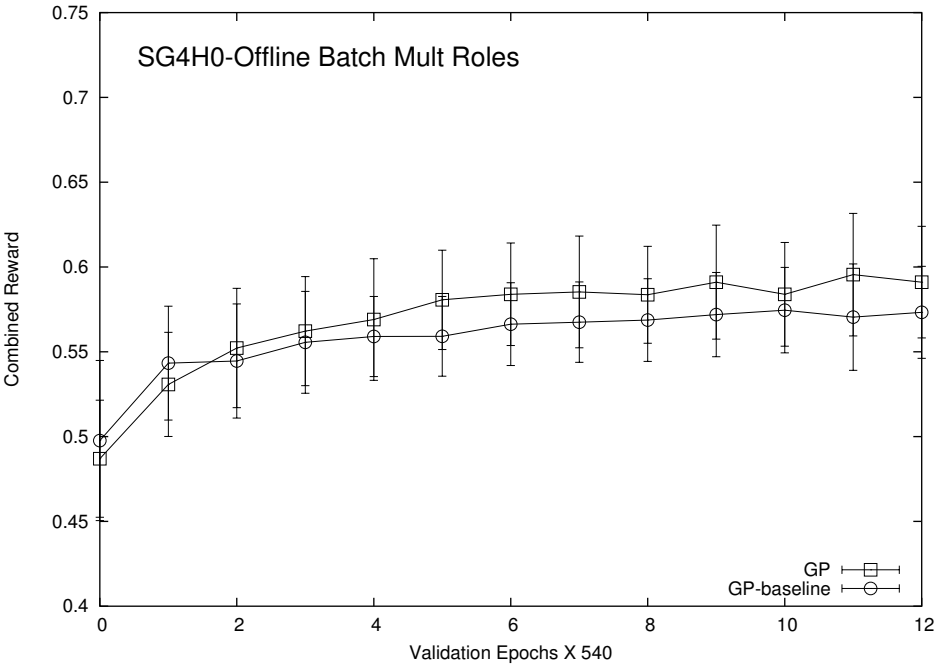

Figure C.53: Average evolution of the combined reward in experiment SG4H0-Offline Batch Multiple Roles.

Table C.30: Results of the final validation cycle for experiment SM4H0-Offline Batch Multiple Roles.

Agent	Avg	Std Dev
QL	0.636003	0.0217209
EA	0.600155	0.0272507
GP	0.580365	0.0406332

Table C.31: Results of the final validation cycle for experiment SQ4H0-Offline Batch Virtual-Experience Multiple Roles.

Agent	Avg	Std Dev
QL	0.62148	0.0235495

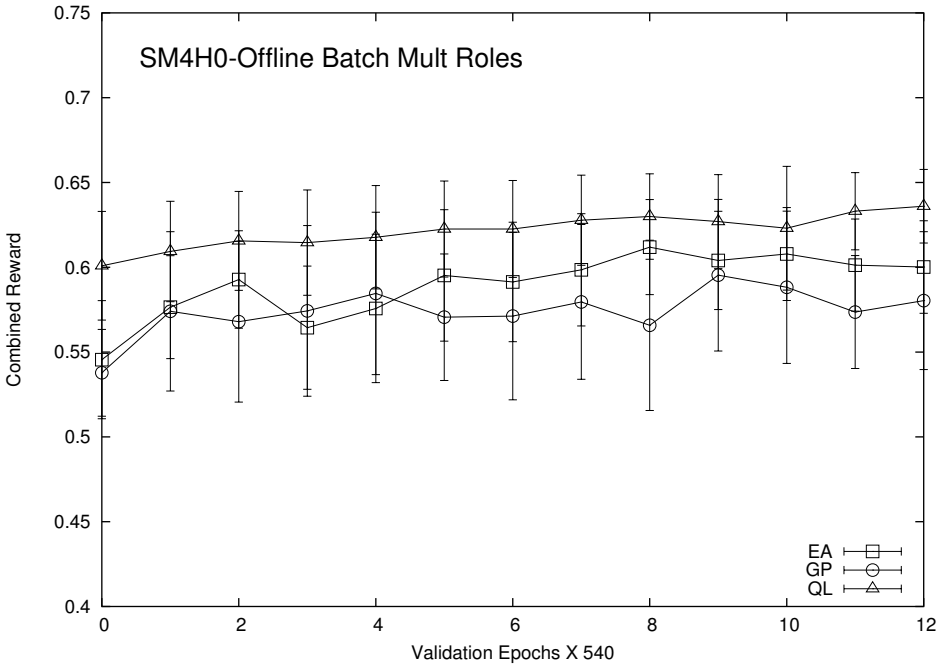

Figure C.54: Average evolution of the combined reward in experiment SM4H0-Offline Batch Multiple Roles.

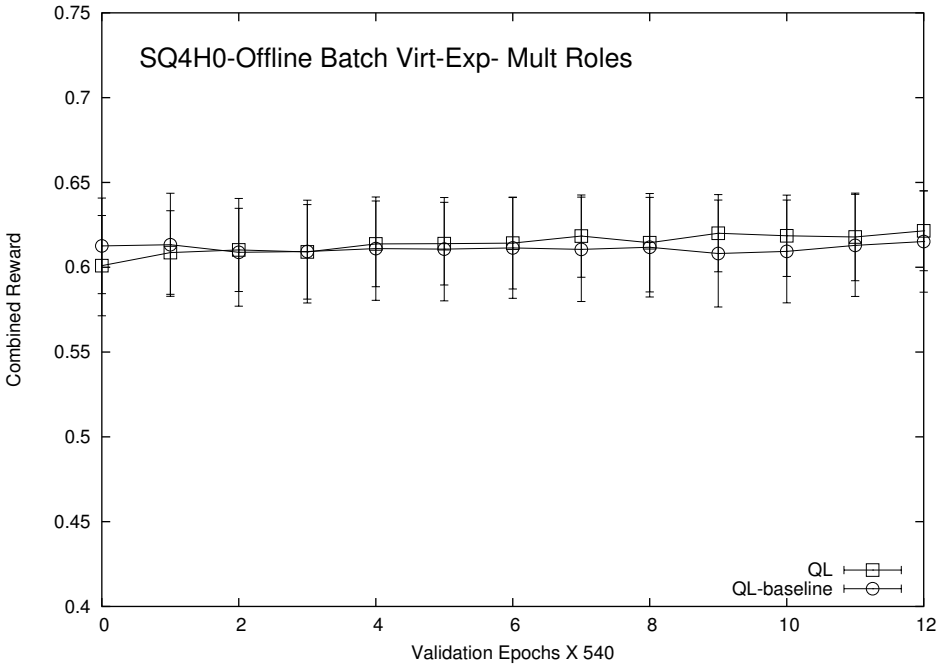

Figure C.55: Average evolution of the combined reward in experiment SQ4H0-Offline Batch Virtual-Experience Multiple Roles.

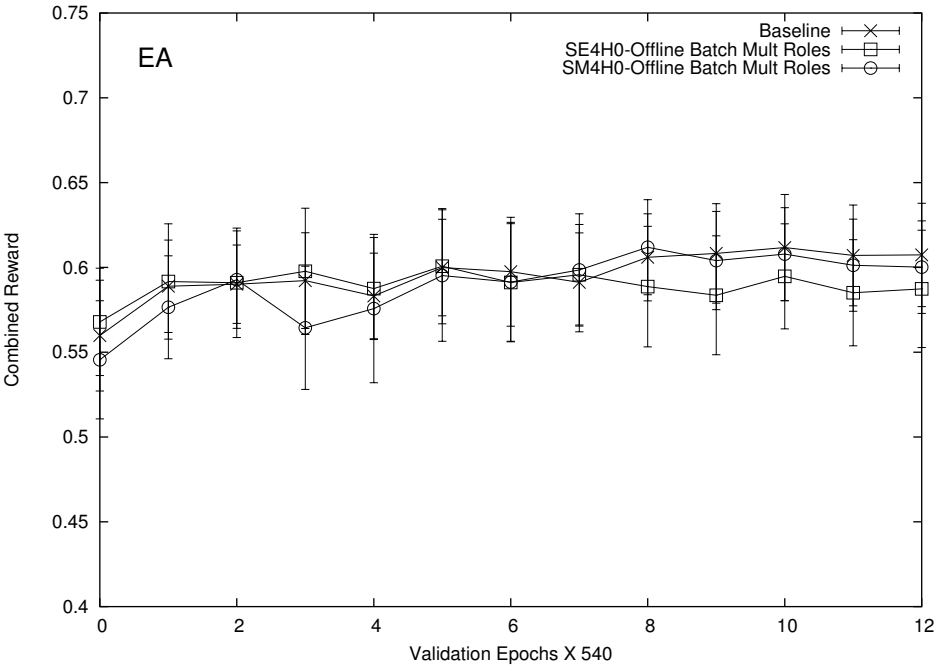

Figure C.56: Summary of results for EA-agents in Experiment Set 7.

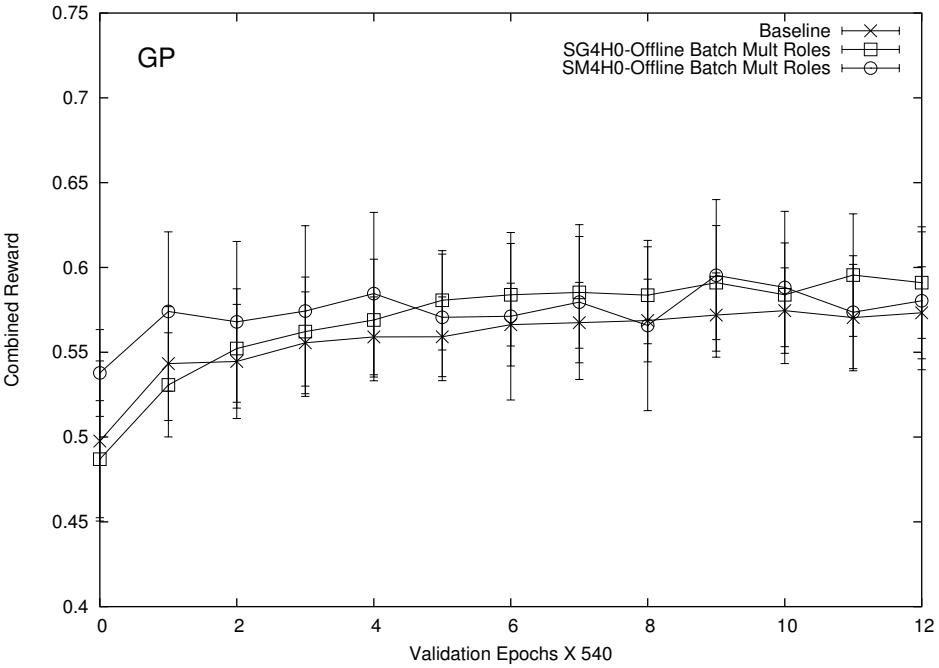

Figure C.57: Summary of results for GP-agents in Experiment Set 7.

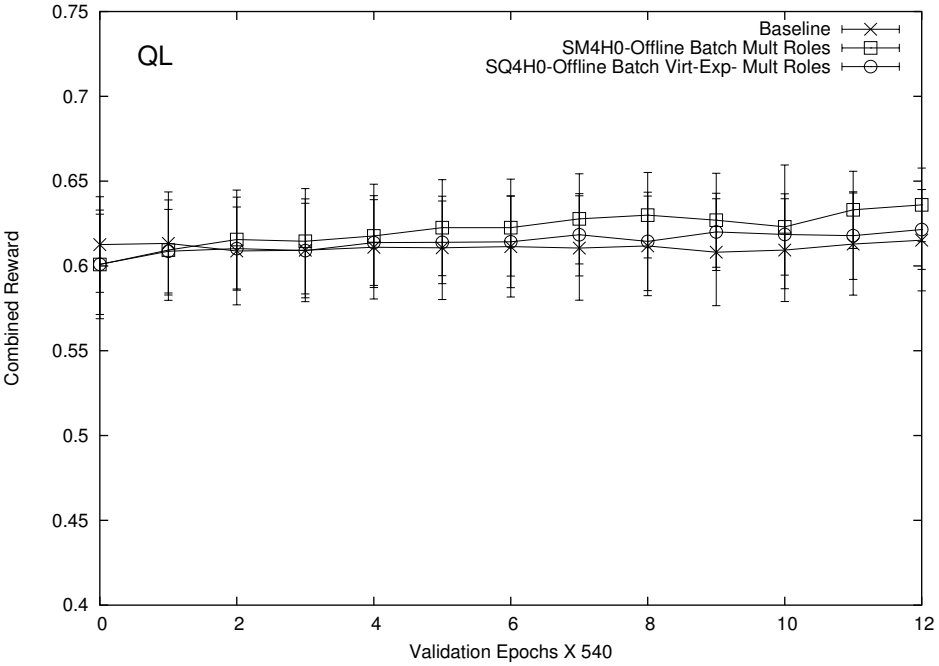

Figure C.58: Summary of results for QL-agents in Experiment Set 7.

C.10 Experiment Set 8

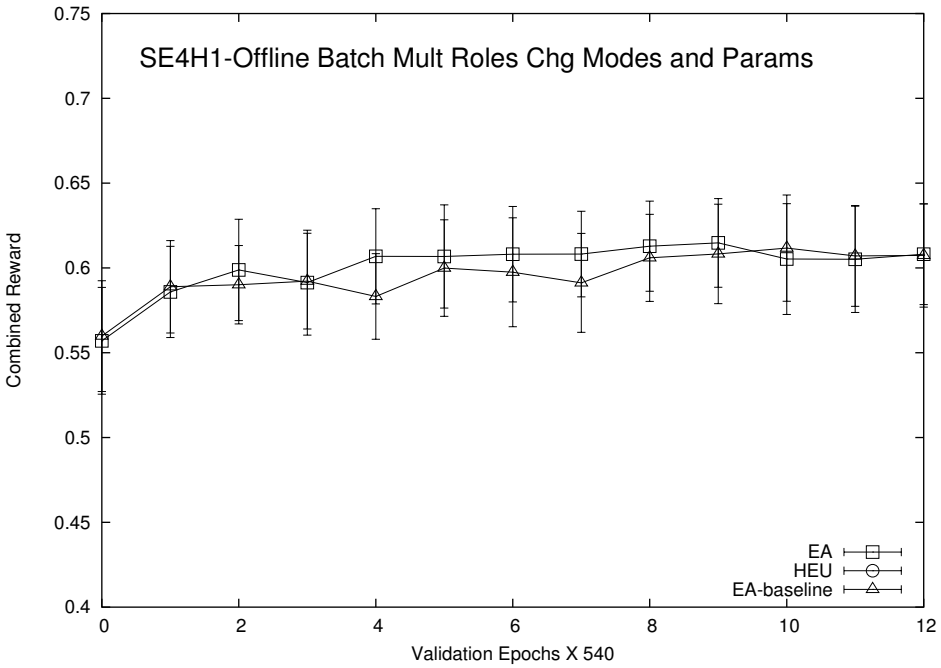

Figure C.59: Average evolution of the combined reward in experiment SE4H1-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Table C.32: Results of the final validation cycle for experiment SE4H1-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.204661	0.0120485
EA	0.608072	0.0297021

Table C.33: Results of the final validation cycle for experiment SG4H1-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.20455	0.0121521
GP	0.578557	0.0260195

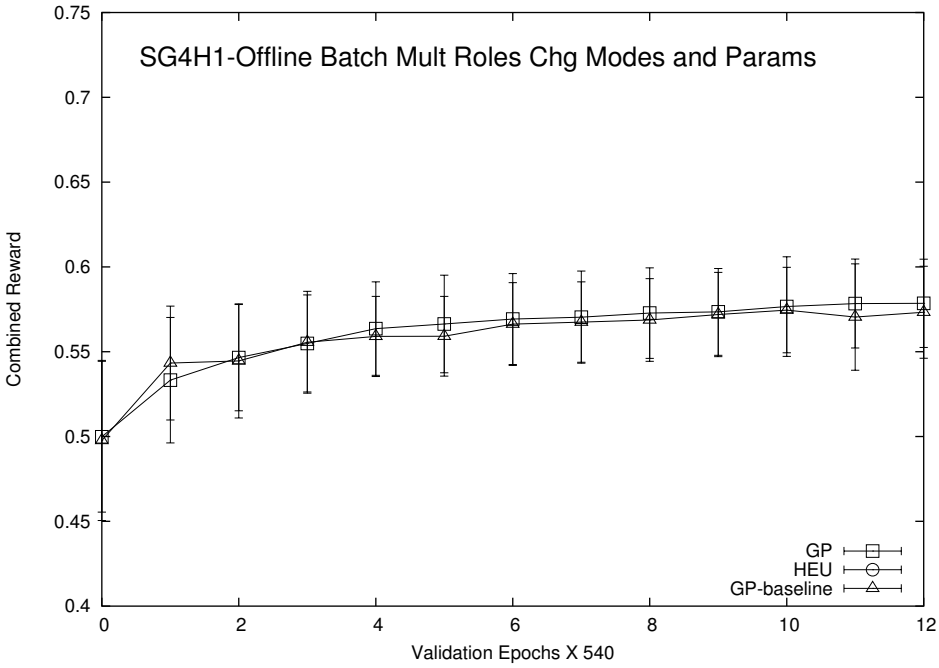

Figure C.60: Average evolution of the combined reward in experiment SG4H1-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Table C.34: Results of the final validation cycle for experiment SM4H1-Offline Batch Multiple Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.204584	0.0123701
QL	0.599468	0.0362202
EA	0.61218	0.0203334
GP	0.580606	0.0209091

Table C.35: Results of the final validation cycle for experiment SQ4H1-Offline Batch Virt-Exp- Multiple Roles Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
HEU	0.204713	0.0121507
QL	0.596397	0.0330602

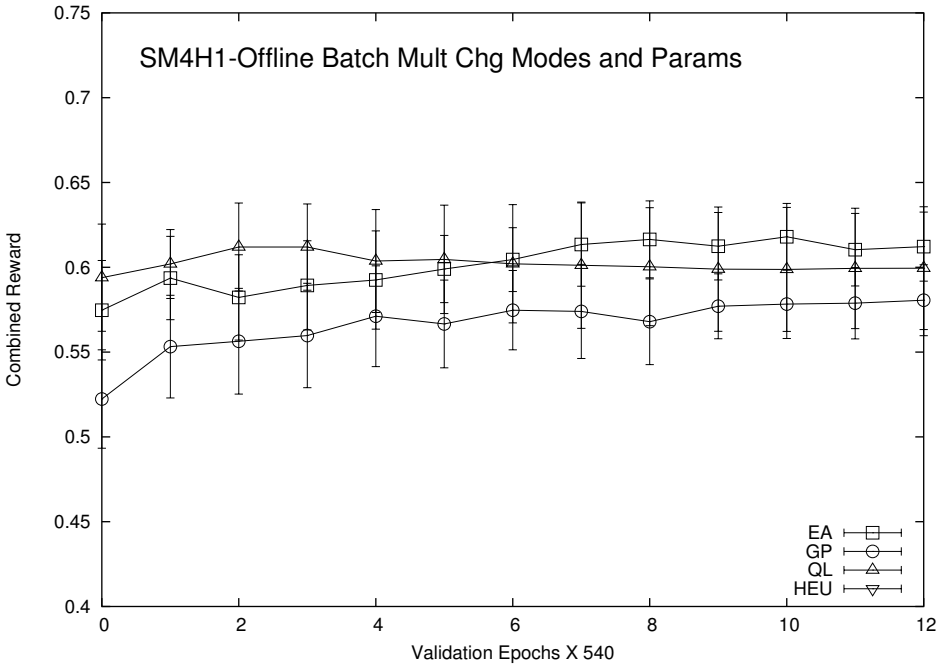

Figure C.61: Average evolution of the combined reward in experiment SM4H1-Offline Batch Multiple Changing Advice Modes and Parameters.

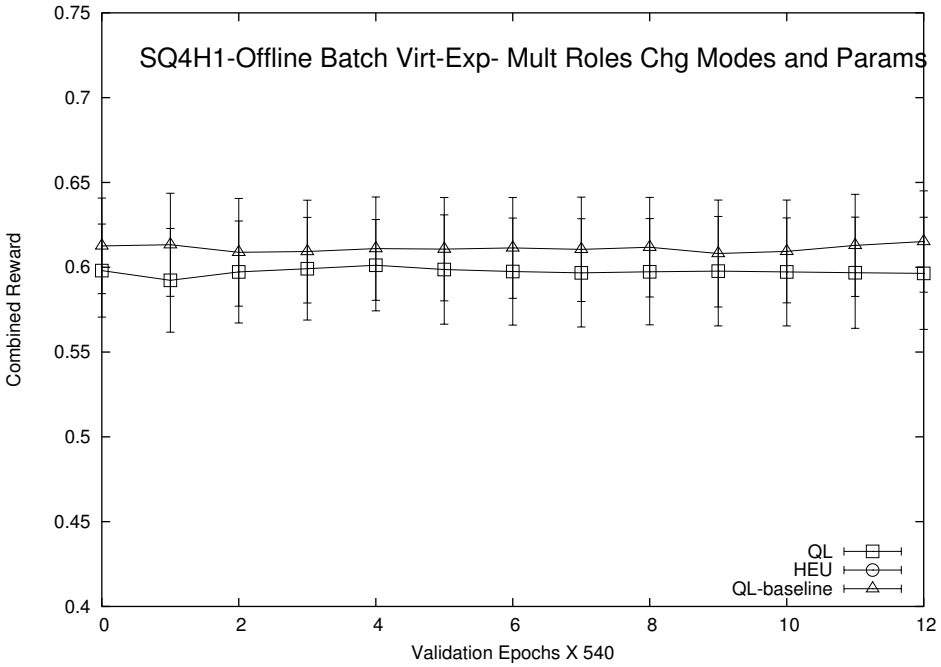

Figure C.62: Average evolution of the combined reward in experiment SQ4H1-Offline Batch Virt-Exp- Multiple Roles Changing Advice Modes and Parameters.

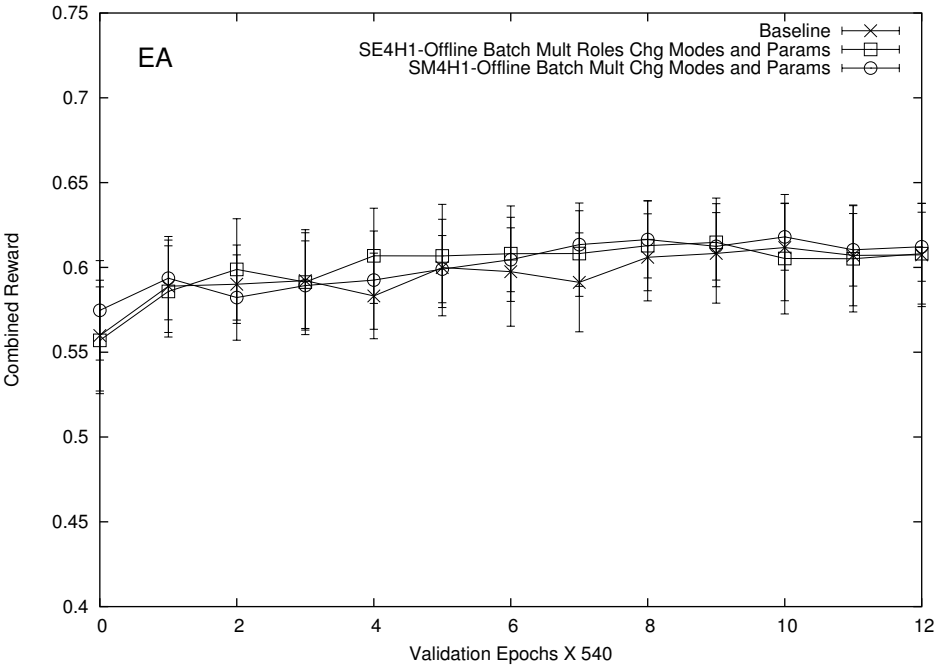

Figure C.63: Summary of results for EA-agents in Experiment Set 8.

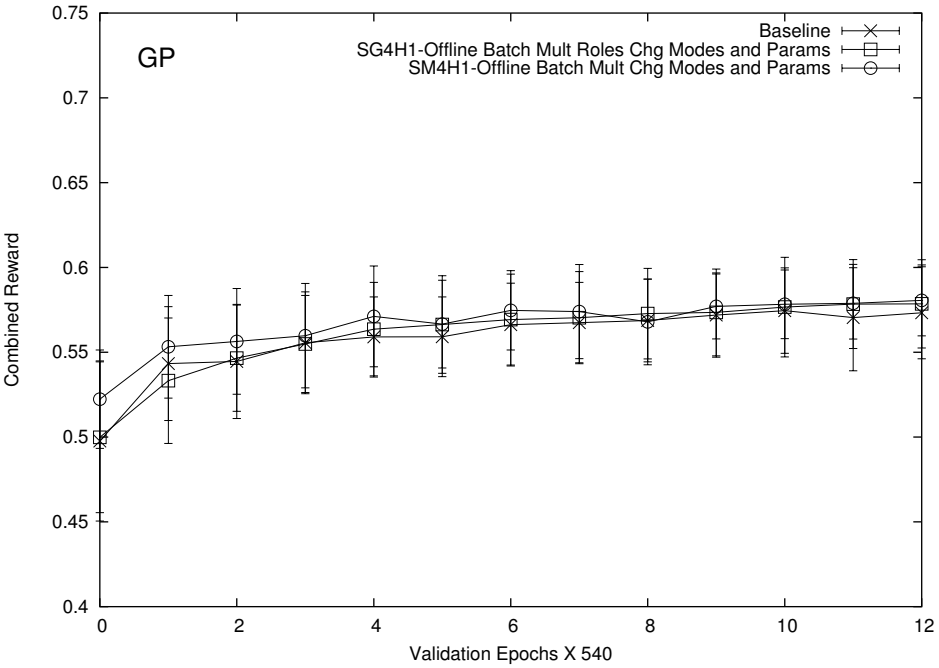

Figure C.64: Summary of results for GP-agents in Experiment Set 8.

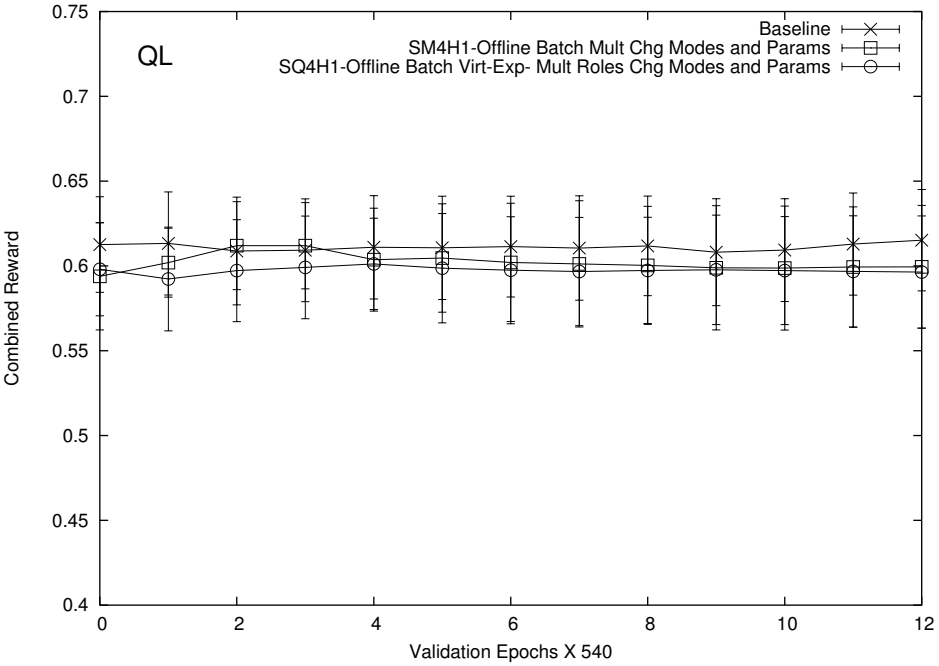

Figure C.65: Summary of results for QL-agents in Experiment Set 8.

C.11 Load Balance Baseline Tests

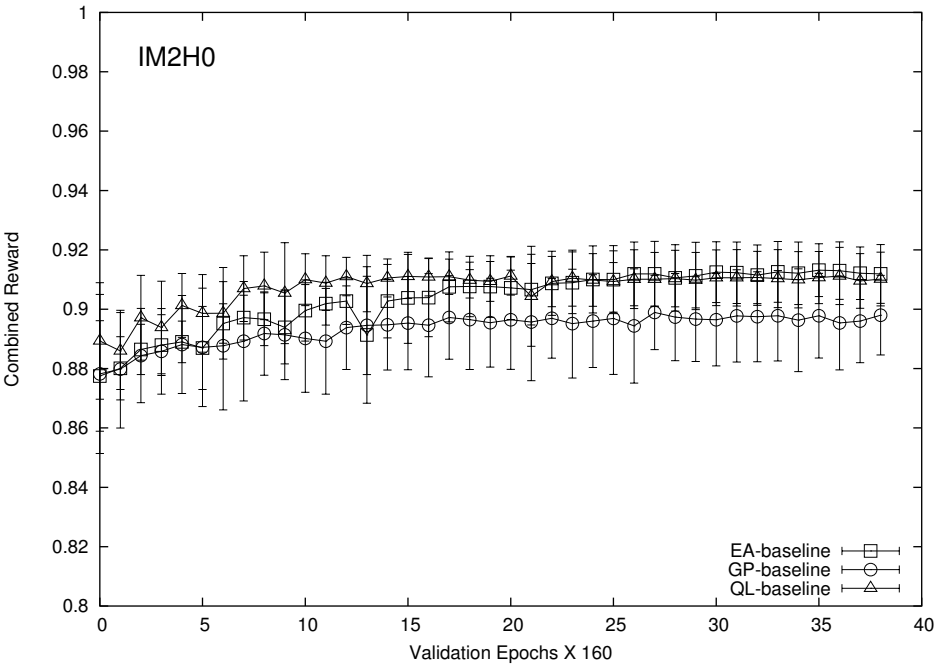

Figure C.66: Average evolution of the combined reward in the baseline experiments for the load-balance problem.

Table C.36: Results of the final validation cycle for the baseline experiment of the load-balance problem.

Agent	Avg	Std Dev
QL	0.910163	0.0090603
EA	0.911899	0.00990651
GP	0.897997	0.0134088

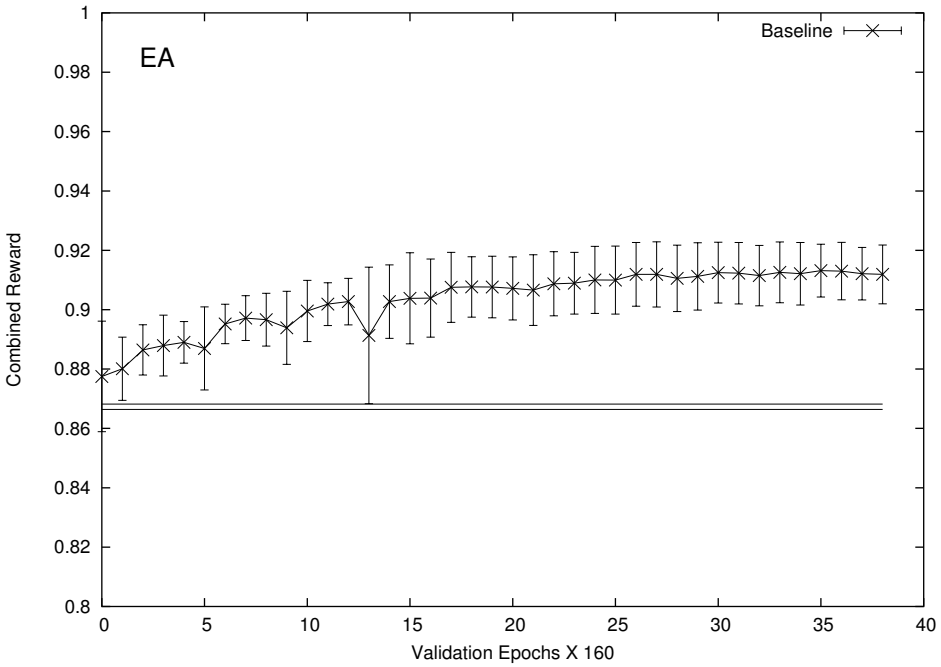

Figure C.67: Summary of results for EA-agents in the baseline Experiment Set for the load-balance problem.

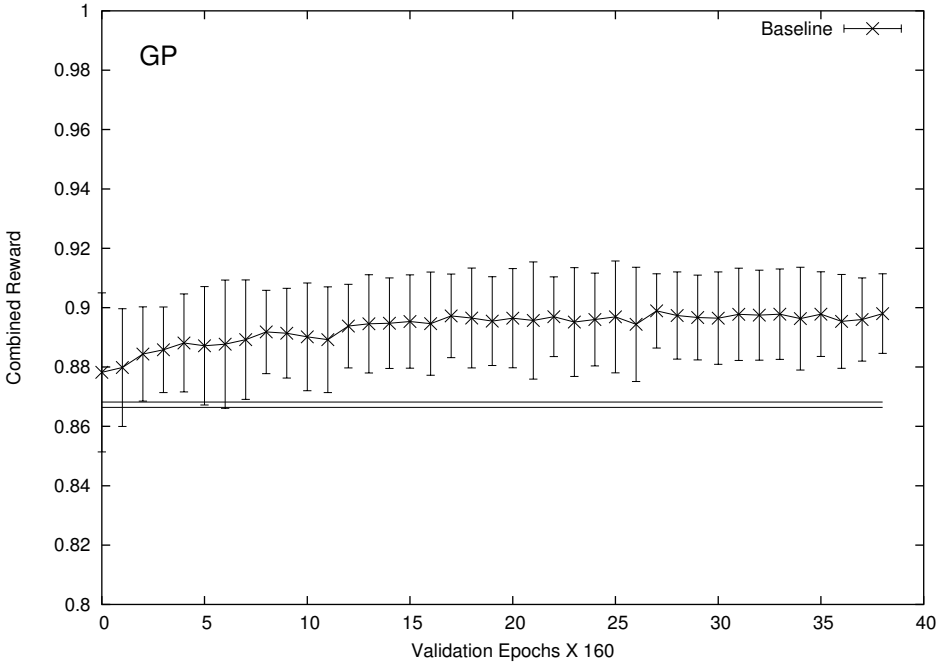

Figure C.68: Summary of results for GP-agents in the baseline Experiment Set for the load-balance problem.

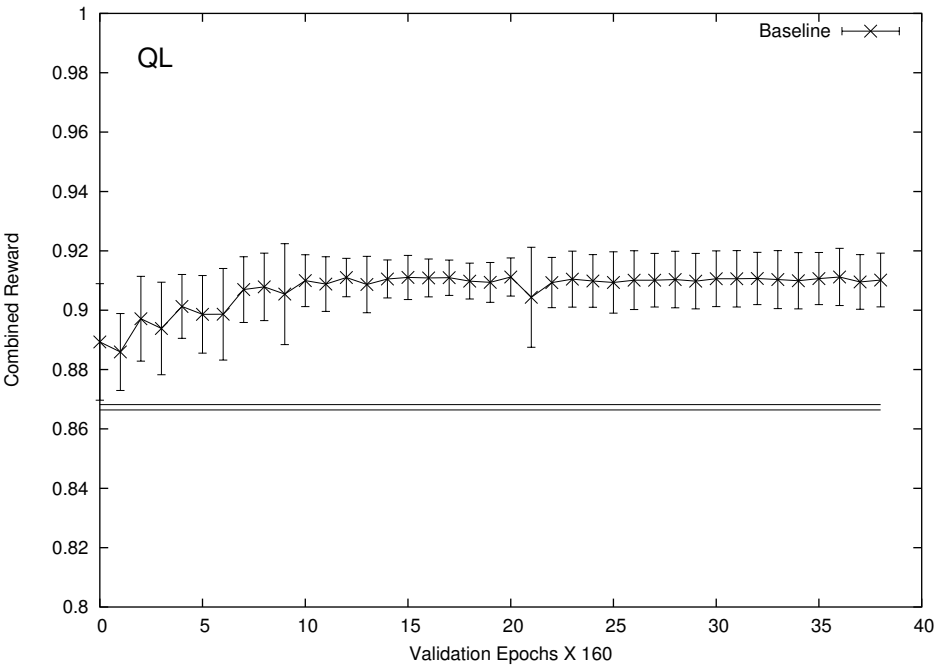

Figure C.69: Summary of results for QL-agents in the baseline Experiment Set for the load-balance problem.

C.12 Experiment Set 9

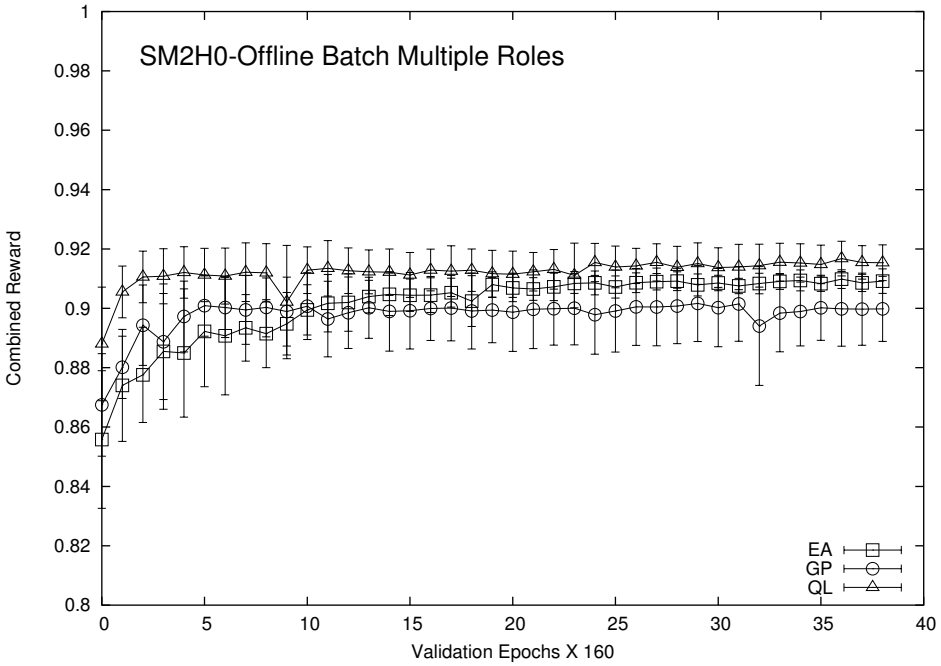

Figure C.70: Average evolution of the combined reward in experiment SM2H0-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Table C.37: Results of the final validation cycle for experiment SM2H0-Offline Batch Multiple Roles Changing Advice Modes and Parameters.

Agent	Avg	Std Dev
QL	0.915373	0.00603051
EA	0.909172	0.00413075
GP	0.899808	0.01097

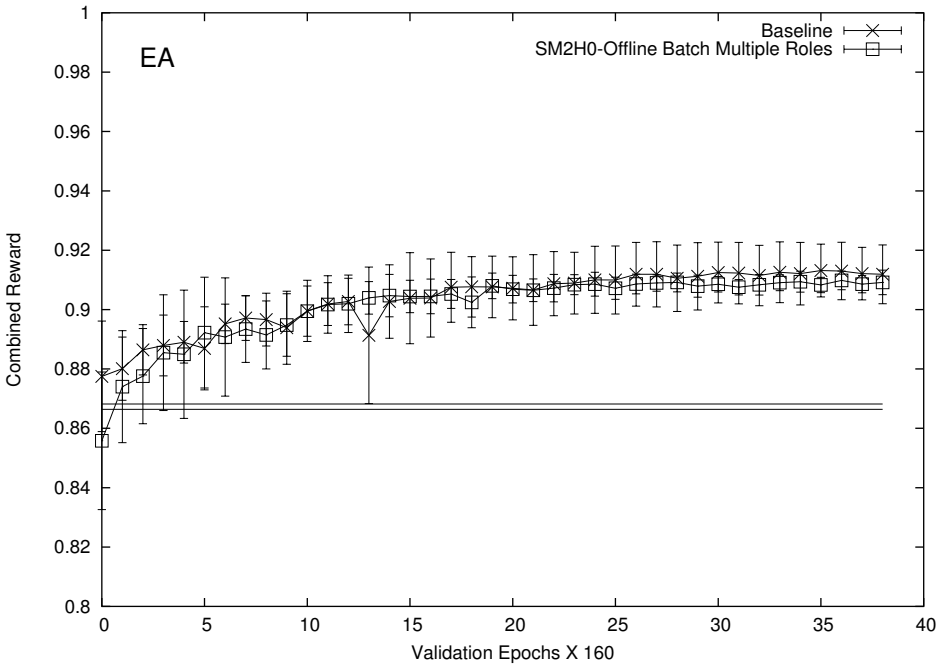

Figure C.71: Summary of results for EA-agents in Experiment Set 9.

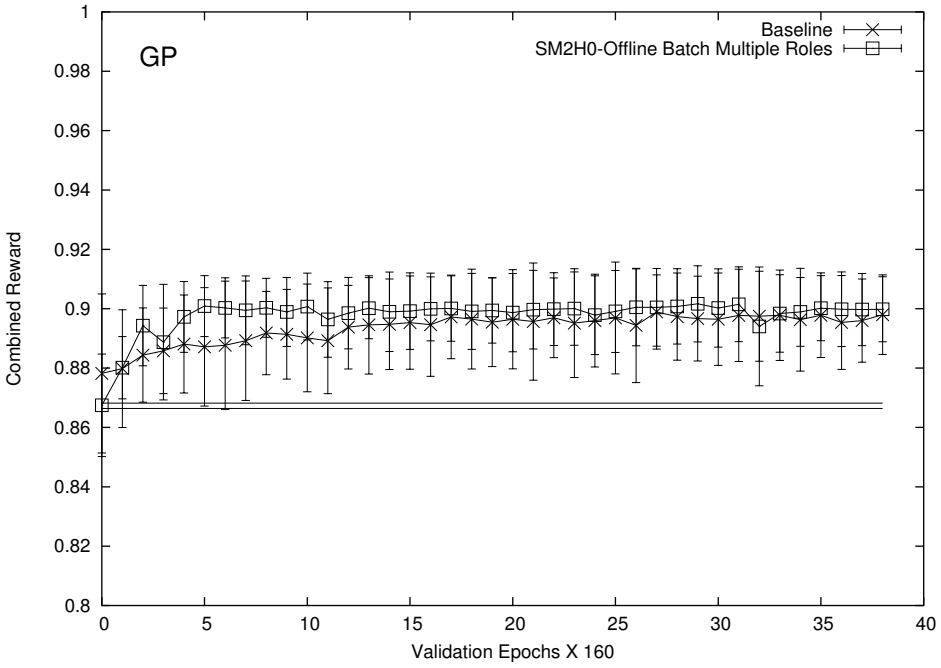

Figure C.72: Summary of results for GP-agents in Experiment Set 9.

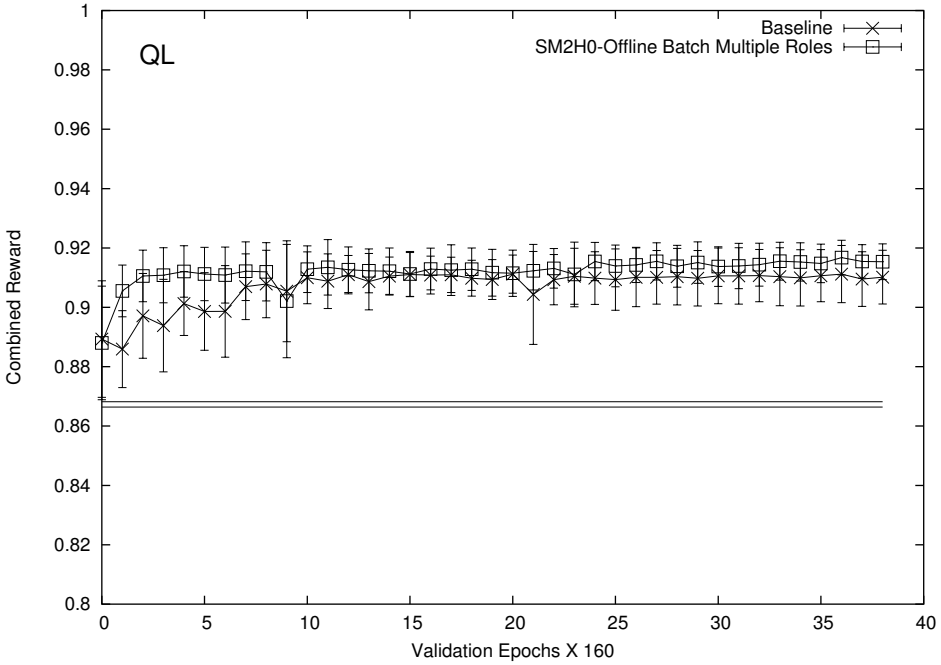

Figure C.73: Summary of results for QL-agents in Experiment Set 9.

Appendix D

Job Routing: Analysis of Baseline Tests

D.1 Introduction

During the attempt to duplicate the experiments reported in (Whiteson and Stone, 2004) we have detected differences between the results of our initial experiments and their baseline tests. We have analyzed the problem and concluded that there are errors either in the experimental results presented or the experiments' description. This report presents our experimental results and an analysis of the baseline tests modeled as a Markov chain. The consistency of the empirical and theoretical results leads us to believe that our results are accurate. These are compared with those presented by the authors of the above-mentioned paper and it is made clear that both results are inconsistent. The cause of the inconsistence was not detected, which prevents us from validating the experiments presented in (Whiteson and Stone, 2004) and comparing their results to our learning approaches.

D.2 Discussion

The baseline tests, used for comparison with the Adaptive Job Routing (AJR) approach, consist on a random router coupled with a FIFO scheduler. We will, in this section, compare an analysis of the expectable performance of the baseline tests with the actual performance reported in (Whiteson and Stone, 2004).

One of the networks studied in this work is the one depicted in figure D.1, labeled *network #3*. In this scenario each user generates one or two jobs, with 50% probability

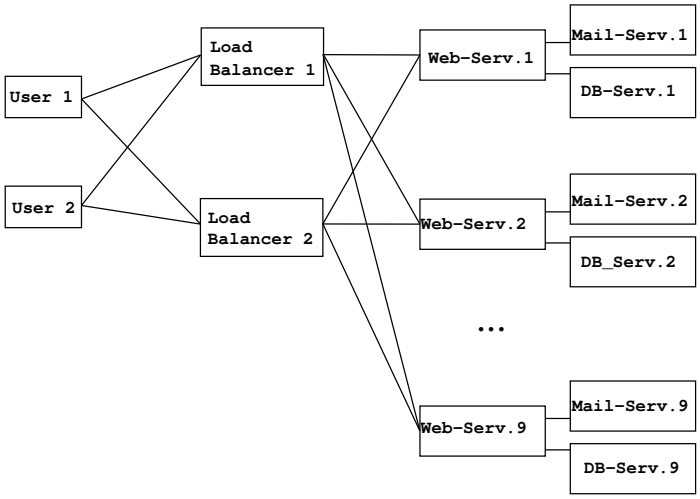

Figure D.1: Load-balance scenario (network #3).

for each case [1]. Each job has one of two possible types, also chosen randomly (with 50% probability for each case): Mail or Database. A Mail-job is composed of three, sequential, job-steps: Web-step; Mail-step; Web-step. A Data-Base job is also composed of three job-steps: Web-step; Database-step; Web-step. In average the 2 users of network #3, will generate 1.5 jobs per turn each. Both users generate a total of 3 jobs per turn, in average, which contain 6 web-steps, 1.5 Mail-steps and 1.5 Database-steps.

Let pg_n be the probability of both users generating n jobs in a given turn, then: $pg_2 = 0.25, pg_3 = 0.5, pg_4 = 0.25$. The probability of generating n jobs of a specific type is:

$$\begin{aligned} pj_0 &= 1/4pg_2 + 1/8pg_3 + 1/16pg_4 = 9/64 \\ pj_1 &= 2/4pg_2 + 3/8pg_3 + 4/16pg_4 = 24/64 \\ pj_2 &= 1/4pg_2 + 3/8pg_3 + 6/16pg_4 = 22/64 \\ pj_3 &= 1/8pg_3 + 4/16pg_4 = 8/64 \\ pj_4 &= 1/16pg_4 = 1/64 \end{aligned} \tag{D.1}$$

Users will stop generating new jobs if they have more than 100 pending jobs (this restriction was not duplicated in our experiments). The work necessary to perform a Web, Mail or Database job-step is, 50, 100 and 200, respectively. Each job is sent randomly to one load-balancer. All types of communication take one simulation step to be performed. It is not clear from the description if a job leaving at t=1 will arrive at the receiver at the beginning of time t=2, in time to be processed, or if it can only

[1] This description proved to be source of the error, in the actual implementation each user generated as many jobs as it could until the total number of jobs generated by both users was equal to 500

be processed at t=3, nor is it clear if a job can be loaded from the link, processed and put on the out-link in the same time-step. We will assume that a job is put on the link at time t, picked up for processing at time $t + 1$ and, if the queue is empty and the processor can complete the job in one time step this job can be processed and put on the out-link during time-step $t + 1$ being available for the next server at $t + 2$. As we explain below this choice has a relatively low impact in the results.

At t=2 the generated jobs arrive at the load-balancers and are queued. The jobs in queue are analyzed in order of arrival time-stamp when using a FIFO scheduler. All jobs with the same arrival time-stamp are given a random ordering. It is assumed that the load-balancing activity runs fast enough so that any number of generated jobs can be processed in one simulation step by any load-balancer [2], despite the fact that the load-balancing nodes have a Work feature that equals 200. This is consistent with the fact that there is no value for the work required to route a job. Still in the same simulation step (t=2), the jobs are sent to a randomly chosen Web-server (choice is made with a uniform probability in the case of a random router).

At t=3 each Web-server (if not completing a pending job) chooses one of the first jobs that arrived and starts processing it. In network #3 the Web-Servers have a work capacity of 400, i.e. each can process 8 Web-steps per simulation step, which gives a total processing capacity of 72 Web-steps per turn, for the 9 Web-servers. The 9 Mail Servers have the following capacities 20, 40, 60, 80, 100, 120, 140, 160 and 180 and the Data-Servers' capacities are: 40, 80, 120, 160, 200, 240, 280, 320 and 360. This, when related with the work necessities of each job-step type, leads to the conclusion that for each of the possible job-steps (Mail and Database) the 9 corresponding servers can process the following number of jobs-steps of that type per simulation step: 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4, 1.6, 1.8. So the capacity of the whole network is of 72 Web-steps, 9 Mail-steps and 9 Database-steps, per simulation step. Taking into account the number of generated job-steps (6, 1.5, 1.5) the usage is 8.3% of the Web-servers capacity and 16.6% of the Mail and Database servers capacity. Since even one Web-server can process more Web-steps than can ever be generated in one time-step no delay is incurred in the Web-server and at t=3 all Web-steps are completed and will be sent to the Mail or Database servers, which will be able to process them at t=4. The fact that there can be no congestion at the Web-servers, even in the worst case scenario, is important because it allows us to model the system as if the user is producing directly to the second-line servers, with an added fixed penalty for the time it takes to cross the links between user and Mail/Database-server. In catastrophe conditions the work/speed quotients of the faster servers are reduced to half, so the number of jobs-steps of a given type per simulation step for each of the 9 servers is, in this case: 0.2, 0.4, 0.6, 0.8, 1, 0.6, 0.7, 0.8, 0.9. This reduces the capacity of the Mail/Database servers to a total of 6 job-steps per time-step (giving a usage-rate of 25%). All our experiments run under catastrophe conditions.

[2] This was assured to us in private communication with one of the authors

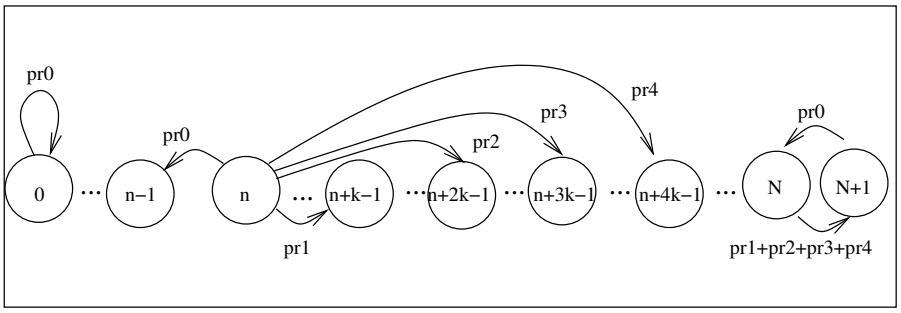

Figure D.2: Markov chain representation for the state of a given server whose speed (i.e. work capacity) is $1/k$.

The system's bottleneck in network #3 is at the Mail/Database servers. Let us then see what are the probabilities of a given Mail/Database-server receiving a given number of jobs in each turn.

Given that each job is distributed randomly and uniformly to one of the 9 possible clusters, and considering the values found in equation D.1, the probability of a server of certain type receiving n jobs (pr_n) of that same type is:

$$\begin{aligned}
pr_0 &= 1pj_0 + 8/9pj_1 + 8^2/9^2pj_2 + 8^3/9^3 + pj_3 + 8^4/9^4pj_4 = 0.843107 \\
pr_1 &= 1/9pj_1 + 16/9^2pj_2 + 3 \cdot 8^2/9^3pj_3 + 4 \cdot 8^3/9^4pj_4 = 0.147364 \\
pr_2 &= 1/9^2pj_2 + 3 \cdot 8/9^3pj_3 + 6 \cdot 8^2/9^4pj_4 = 0.009272 \\
pr_3 &= 1/9^3pj_3 + 4 \cdot 8/9^4pj_4 = 0.000247 \\
pr_4 &= 1/9^4pj_4 = 0.000002
\end{aligned} \tag{D.2}$$

So the number of jobs arriving per turn (in average) should be approximately: $0.147364 + 2 \cdot 0.009272 + 3 \cdot 0.000247 + 4 \cdot 0.000002 = 0.1(6)$. This means that, in average, a server will receive one job every 6 turns. Since the slowest server can process a job every 5 turns, in average, there should be a relatively low level of congestion, even in the slowest servers.

A more detailed analysis can be made by modeling each server as a Markov chain. If a certain server does $1/k$ jobs per time-step, then the corresponding Markov chain will be the one presented in figure D.2, where state n corresponds to having n parts of $1/k$ size of a job in queue. When m new jobs arrive and $1/k$ jobs are processed per step the state will change to $n + mk - 1$.

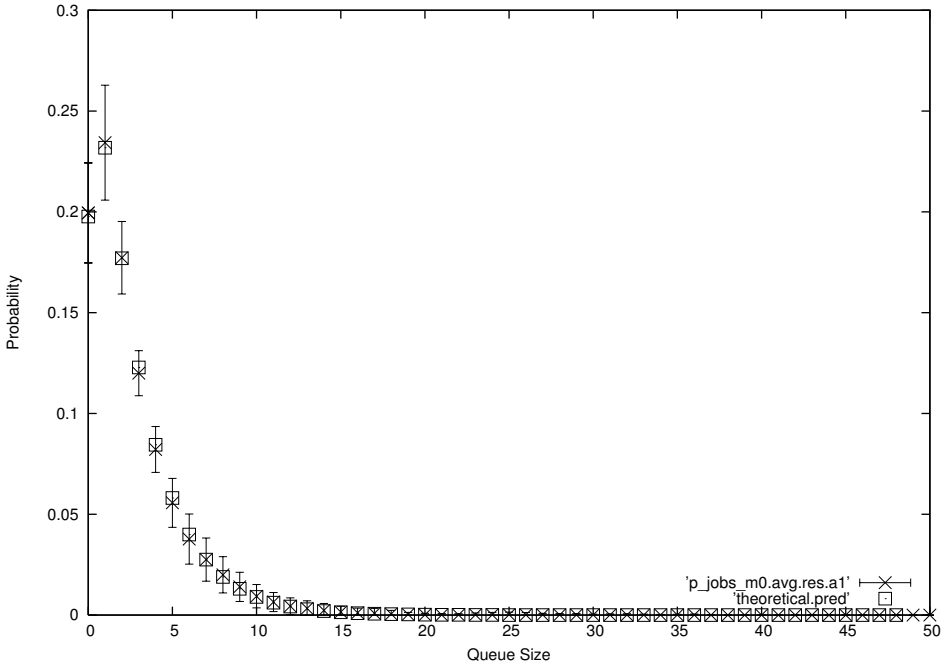

Figure D.3: Comparison of the probability of having n jobs in queue in the slowest Mail-server, (that completes only one job every 5 time-steps) obtained by equation D.3 with k=5 (represented by the squares), with the empirical results averaged over 51 experiments (represented by the cross). The 95% confidence interval for a t-test, relative to the empirical results, is also presented.

From this formulation we can extract the following equations:

$$p_0 = (p_0 + p_1)pr_0 \tag{D.3}$$

$$p_n = p_{n-4k+1}pr_4 + p_{n-3k+1}pr_3 + p_{n-2k+1}pr_2 + p_{n-k+1}pr_1 + p_{n+1}pr_0,$$

$$\sum_{n=0}^{N+1} p_n = 1,$$

where p_n is the probability of being in state n when the system is running in steady-state. This represents a system with an infinite number of equations. In order to solve D.3 numerically a Markov chain with a finite number of states is defined from the original one. In the finite Markov chain N+1 represents the state where there are N+1 or more 1/k fractions of a job in queue. On our numerical solution N is chosen to be large enough so that the probability to be is state N+1 can be considered negligible. For the server with less capacity N was set to 500 (equivalent to a queue of 100 jobs) and a value of $p_{N+1} = 1.2e - 17$ was obtained.

We can see in figures D.3 and D.4 the comparison of the probabilities of being in each state for the slowest servers. Similar results were achieved for other cases. This was chosen as an example only because it is the most critical case (along with the

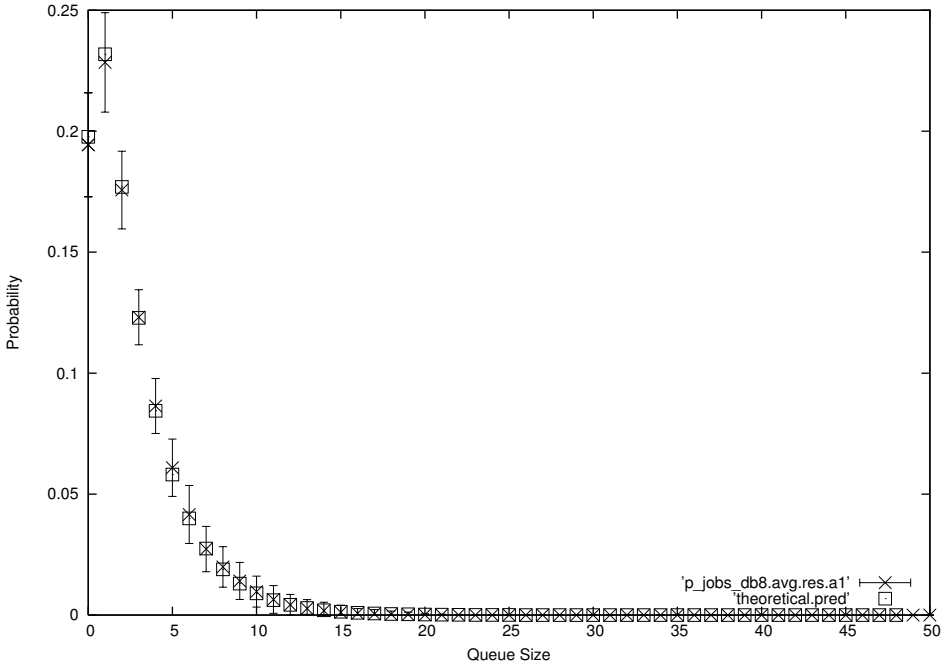

Figure D.4: Comparison of the probability of having n jobs in queue in the slowest Database-server, (that completes only one job every 5 time-steps) obtained by equation D.3 with k=5 (represented by the squares), with the empirical results averaged over 51 experiments (represented by the cross). The 95% confidence interval for a t-test, relative to the empirical results, is also presented.

slowest Database server that has an equal throughput). The fact that the theoretical predictions match very accurately our estimated probability, extracted from experimental runs, leads us to conclude that the implementation of our experiments is in strict accordance with the model we have presented and that we believe is exactly the same as the one described in (Whiteson and Stone, 2004).

Let us then continue our analysis of the path taken by a packet assuming that there is an average delay of M steps at the Mail/Database server. The job will leave the server at time $t = 4 + M$, arriving back at the Web server at $t = 5 + M$, and since from that point on there is no more cause for congestion, it will, inevitably, arrive at the user at time $t = 7 + M$ after following the remaining links: Web-server to load-balancer; load-balancer to user. If the initial assumption (that a job can be put on the outbound link in the same turn as it was finished, even if it was just taken off the inbound link in the same time-step) does not hold, we will have an added delay equal to the number of nodes the job passed in in its path, i.e. 5. So the average delay will be either $7 + M$ or $12 + M$.

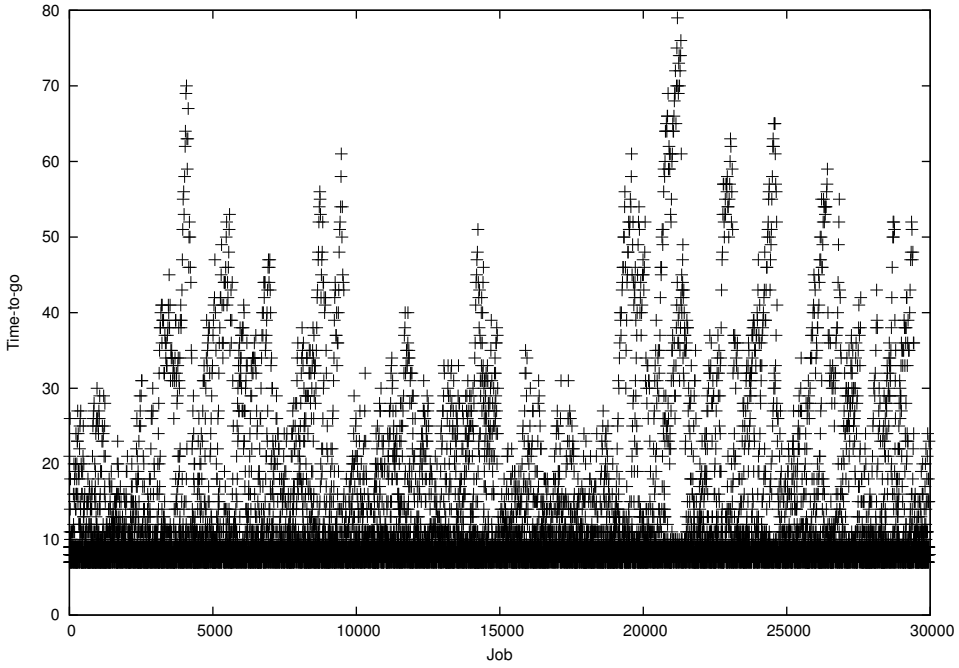

Figure D.5: Time-to-go for all jobs generated in one experiment (after warmup).

Each of the users will measure the performance in a different way

$$R_1(t) = \begin{cases} -10t, & : \quad t < 50 \\ -t/10 - 495, & : \quad \text{otherwise} \end{cases} \tag{D.4}$$

$$R_2(t) = \begin{cases} -t/10, & : \quad t < 50 \\ -10t + 495, & : \quad \text{otherwise}, \end{cases} \tag{D.5}$$

where t is the job's completion-time.

Our experimental results, based on 51 experiments indicate that the average time-to-go (averaged over all jobs in each experiment and then for all experiments) is approximately 10 (to be precise 9.8), and reach an average score of -50 (to be precise -49.7), which means that the average delay M is approximately 3. All results are acquired after 5000 warm-up steps.

Figure D.5 represents the time-to-go of all jobs in the experiment whose final score is closest to the average (-49.82). Figure D.6 represents the average score of both users for finished jobs in the same experiment.

The results are according to the expectations since the number of jobs that have a time-to-go higher than 50 is negligible and the users generate a balanced number of jobs each. Thus, the resulting score is bound to be approximately $-(10t + t/10)/2$, where t is the average time-to-go. This is confirmed by the experimental results: $-(10 \cdot 10 + 10/10)/2 \approx -50$. If an extra delay of 5 time-steps was introduced to account

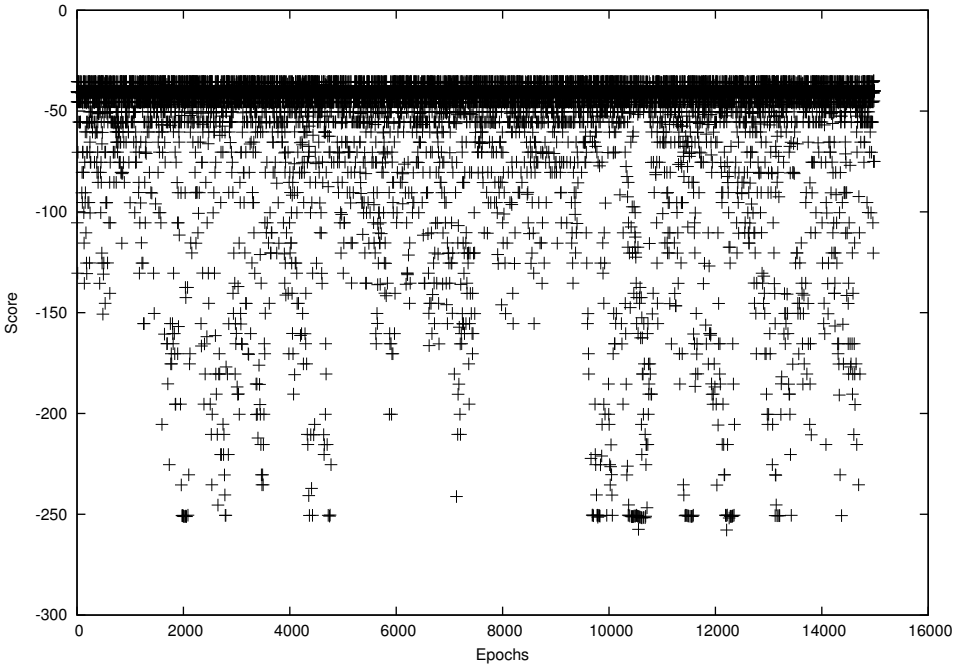

Figure D.6: Average score of both users in each epoch (after warmup).

for the above-mentioned delay, the average time-to-go would rise to 15 and the average score to -75, approximately.

To avoid testing under initial conditions the simulator is "warmed-up" for 5000 steps before measurements are taken, as in the described experiments. Since there is only temporary congestion, this is hardly relevant for our analysis. It is also worth mentioning that using the Markov chain model it is possible to prove that the time necessary to reach steady-state conditions is far less than 5000. All our measurements were taken under catastrophe conditions and without restricting the users to less than 100 pending jobs, i.e. under the worst possible scenario presented.

D.3 Conclusions

The results presented in (Whiteson and Stone, 2004) for the baseline test (random routing + FIFO scheduling) that we have attempted to replicate are in average approximately -700 (the exact number is never mentioned in the paper and we must rely on the results displayed in figure 8 a) for an estimate). This result is the same both in normal as well as in catastrophe conditions.

The above reasoning, as well as the results of replicating the experiment, led us to conclude that, either both implementations differ, or we have inconsistent results for the same experiment. Private communication with one of the authors (to whom

we thank the patience of answering our questions) led us to believe that we have accurately replicated the experiment. The theoretical analysis of steady-state behavior of the system assures us that the results of our experiment are in agreement with these predictions. Despite these facts the discussion with one of the authors was inconclusive as to the cause of the inconsistent results or the possible differences in the experiment.

These facts do not invalidate the conclusions in (Whiteson and Stone, 2004) concerning the relative increase of performance of the routing algorithms in relation to the base-test, but they make it impossible to repeat the experiments and validate these results. It is also impossible to compare results with our own learning approach, which was our main objective. We believe that an effort should be made by both teams to find the cause of the inconsistencies and that this effort may help us, and the authors of the above-mentioned paper, to clarify the description of the experiment, or, in case an error is detected, to correct it. Our only objective is to replicate the experiment and compare the adaptive approaches.

D.4 Acknowledgements

We would like to thank Shimon Whiteson for answering most of our doubts related to the implementation of these experiments. We would also like to stress that, despite the divergence in opinions concerning this particular paper, we have the utmost respect for the authors' work, that has been a source of inspiration to our own research.

We would also like to thank Rui Lopes for his collaboration in the theoretical analysis of this experiment.