"A multi-agent framework for developing adaptable and open service systems : application to web services and supply chain management"

Achbany, Youssef

Abstract
Rapid evolutions in today's businesses require new techniques and powerful tools to adapt and manage change in complex environments. Indeed, managing the complexity of systems is considered a key challenge in computing and is addressed through various approaches aimed at increased automation. One relevant response to such concerns is to rely on self-contained components, as in service-oriented computing which is a new engineering technique addressing part of those emerging issues and where a service is a self-describing, open component supporting rapid, low-cost composition of distributed applications. Adaptable and open service-oriented systems are expected to enable the provision of a large number of distinct and competing services which the service requesters will be able to choose dynamically in the aim of receiving at all times optimal offerings for their purposes. Service-oriented systems ought to be open to permit many services to participate. Openness commits such systems to ...

Document type: Thèse (Dissertation)

Référence bibliographique
Achbany, Youssef. A multi-agent framework for developing adaptable and open service systems : application to web services and supply chain management. Prom. : Saerens, Marco ; Kolp, Manuel
A Multi-Agent Framework for Developing Adaptable and Open Service Systems

Youssef Achbany

A thesis submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in
Management Sciences
of the Université catholique de Louvain

Examination committee:

Prof. Manuel Kolp (UCL), Advisor
Prof. Marco Saerens (UCL), Advisor
Prof. Stephane Faulkner (FUNDP), Examiner
Prof. Bernard Fortz (ULB), Reader
Prof. François Fouss (FUCaM), Reader

August 2009
My thesis is heartily dedicated to my beloved uncle Jilali who just passed away. You was such an inspiration to me and to many as well.
Acknowledgement

Mes remerciements vont tout d’abord à mes promoteurs de thèse; les professeurs Marco Saerens et Manuel Kolp. Lors de mon engagement, ils m’ont demandé de travailler sur un projet qui permette de fusionner leur discipline respective. Ils m’ont fait découvrir les domaines du machine learning et des systèmes multi-agents, en me supervisant et en me laissant une liberté dans mes choix de recherche. Leurs manières d’aborder leur discipline et leur démarche scientifique m’ont fortement influencées tout au long de mon processus de thèse. Sans leur aide, cette thèse n’aurait peut-être jamais vu le jour.

La troisième personne, qui m’a fortement aidé et conseillé tout au long de mon processus de thèse, est le professeur Stéphane Faulkner. Si le règlement me permettait de prendre un troisième promoteur, ce serait sans conteste lui. Il m’a pris sous son aile, comme il l’aurait fait avec son propre chercheur, et m’a guidé durant ces années. Avec sa collaboration et celle d’Ivan Jureta, que je remercie également, plusieurs papiers avec des contributions significatives ont vu le jour. Un tout grand merci donc à cette personne avec qui un grand lien d’amitié et de respect s’est installé dès le début de notre collaboration scientifique.

J’exprime également mes remerciements aux autres membres du jury, les professeurs Bernard Fortz et François Fouss, pour avoir accepté d’être les lecteurs de ma thèse et pour leurs remarques très pertinentes.

Durant les années qui m’ont conduit à l’élaboration de cette thèse, j’ai travaillé au sein de la LSM où j’ai eu beaucoup de joie et de plaisir avec mes collègues.

Je remercie bien évidemment mon éternel binôme, celui dont le nom est souvent associé au mien, celui qui a rendu mes années à la LSM inoubliable, Yves Wautelet. Essayer d’exprimer mon amitié à son égard par des mots serait
impossible, surtout en quelques lignes. Je me contente donc de dire: grand merci à toi cher ami . . .

Je tiens également à remercier mes autres collègues, en particulier Nicolas, avec qui j’ai été très proche durant mon passage à la LSM. On a partagé des moments d’anthologie. Je lui souhaite beaucoup de bonheur dans la prochaine grande étape de sa vie.

Merci à Kenisio pour les moments mémorables qu’on a partagés en seulement deux années. Je suis sûr, connaissant son tempérament proche du mien, que nos chemins se croiseront encore.

Comment parler de mes collègues sans citer ma grande amie Luh, la personification vivante d’un personnage de manga. Elle m’a souvent remonté le moral et conseillé sur divers domaines. Nos discussions vont surement me manquer. Je remercie également Adrien, Caroline, François, Didier, Jean-Christophe, Jean-Sébastien, Albert, Jérémie, Kevin, Quentin et Mathias.

Je remercie les membres non-académiques de la LSM, en particulier Lucy Cardols, Marie-Claire Wadin, Eric Nicolay et Bernard Tondeur pour leur aide et leur soutien.

Je tiens à remercier également mes amis qui m’ont soutenu tout au long de ma thèse. Je commencerai par Yousta, pour son soutien logistique durant mes années d’études et de thèse. Ils m’a souvent aidé au dernier moment et m’a sorti de pas mal de situations désespérées (même à la veille de la publication de ma thèse). Je n’oublie pas de citer Fouad, Ismael, Fikri, Bouzid, Abdel, Ahmed, Hicham, et tous mes autres amis avec qui j’ai partagé tellement de bons moments.

Enfin, mes derniers et plus grands remerciements s’adressent à ma famille, et tout particulièrement à mes parents. Ils ont montré un profond intérêt pour mes études, mes objectifs et, grâce à leur confiance et leur encouragement, j’ai pu suivre ce chemin difficile. Même si les mots ne peuvent récompenser leur longue attente, je les remercie de tout cœur d’avoir fait de moi ce que je suis aujourd’hui. Un grand merci à mes sœurs Nadia, Amal, Salwa et Shyama, Morgane, à mon petit frère Hamza, à mes deux beaux-frères Hassan et Saïd, à mon cousin Younes, à mes tantes et à mes oncles.

J’adresse en dernier lieu un petit mot à mes grands parents maternels qui m’ont gardé et élevé aux premières années de ma vie, ainsi qu’à mes grands parents paternels que je n’ai pas ou pas assez connus.

Youssef
I Introduction

1 Introduction

1.1 Context of the Research

1.1.1 Service-Oriented Computing

1.1.2 Multi-Agent System features for Service-Orientation

1.2 Principal Contributions

1.2.1 Organizational Modeling and Requirements Engineering

1.2.2 Architectural Design

1.2.3 Detailed Design

1.3 Validation

1.4 Limitations

1.5 Reading Map

II Reinforcement Learning

2 Tuning Continual Exploration in Reinforcement Learning

2.1 Related work

2.2 Basic framework

2.2.1 Computation of the total expected cost for a given policy

2.2.2 Degree of exploration as entropy
CONTENTS

2.3 Optimal policy under exploration constraints for deterministic shortest-path problems ............................................ 30
  2.3.1 Computation of the optimal policy .......................... 32
  2.3.2 Some limit cases ............................................. 34
2.4 Optimal policy for directed acyclic graphs ................................. 35
  2.4.1 Construction of a directed acyclic graph ...................... 35
  2.4.2 Computation of the optimal policy .......................... 36
2.5 Optimal policy under exploration constraints for stochastic shortest-path problems ............................................ 36
  2.5.1 Equations for optimal policy and expected cost ............ 37
  2.5.2 Online estimation of the expected cost ...................... 37
  2.5.3 Links with Q-learning and SARSA ........................... 38
2.6 Optimal policy under exploration constraints for discounted problems .................................................. 40
2.7 Experiments ....................................................... 41
  2.7.1 Preliminary experiments ...................................... 41
  2.7.2 Some experimental comparisons .............................. 51
  2.7.3 The puzzle problem ......................................... 54

3 Reinforcement Learning: Two Additional Extensions 59
  3.1 Randomized Shortest-Path Problem: Global Entropy .................. 61
    3.1.1 Introduction ................................................. 61
    3.1.2 Contributions and Application Areas ....................... 63
    3.1.3 Statement of the problem and notations ................... 65
    3.1.4 Optimal routing policy under entropy constraint .......... 69
    3.1.5 Computation of an optimal policy ........................ 74
    3.1.6 Experiments ............................................... 75
  3.2 Hierarchical reinforcement learning ................................ 81
    3.2.1 Related Work .............................................. 81
    3.2.2 Macro-actions and primitive actions ....................... 84
    3.2.3 Concurrent actions ....................................... 86
    3.2.4 AND/OR Graph ........................................... 88
    3.2.5 Macro-actions and Concurrent actions in a common hierarchical framework ................................................. 90
## CONTENTS

5.4 SM: Task Allocation Algorithm ........................................ 156
   5.4.1 Randomized RL Algorithm ..................................... 157
   5.4.2 Formulation of the Problem .................................... 159
   5.4.3 Satisfying Hard Constraints ................................. 160
   5.4.4 Computation of the Optimal TA Policy ...................... 162
   5.4.5 Illustrative Examples ......................................... 163
5.5 SR: Probabilistic Reputation Model ................................. 166
   5.5.1 Formulation of the Problem ................................. 167
   5.5.2 Computation of the TA Reputation Scores .................. 168
   5.5.3 Illustrative Examples ......................................... 170
5.6 Logical Architecture .................................................. 173

6 Service-Oriented Framework for MAS Modeling ..................... 177
   6.1 Problem Statement ................................................. 178
      6.1.1 Related Work .............................................. 179
      6.1.2 The Lack of Scope Elements .............................. 180
      6.1.3 Contributions .............................................. 180
   6.2 A Unified-Framework for Agent-Oriented Requirements Engi-
      neering ............................................................. 181
      6.2.1 The service approach ....................................... 181
      6.2.2 The Framework .............................................. 182
      6.2.3 The FaMOs Framework: a Project Management Per-
                     pective ........................................... 183
   6.3 Service Driven Agent Modeling: a Formalization ............. 184
      6.3.1 Strategic Services Model .................................. 184
      6.3.2 Dynamic Service Model ................................... 186
   6.4 Illustrative Example .............................................. 189
      6.4.1 The Strategic Services Diagram .......................... 189
      6.4.2 Strategic Dependency Diagram ............................ 190
      6.4.3 Dynamic Services Graph ................................... 190
      6.4.4 Lessons learned ............................................ 192

7 Process for Developing Adaptable and Open Service Systems ......... 195
   7.1 Open and Dynamic Architectures ................................ 196
   7.2 Process Structure ................................................ 196
   7.3 Process Life Cycle ............................................... 197
## CONTENTS

10.1 Thesis Summary ................................................. 242  
10.1.1 Analysis Level ........................................... 242  
10.1.2 Design Level ............................................. 242  
10.1.3 Thesis Organization ....................................... 243  
10.2 Further Work .................................................. 244  
10.2.1 Process ProDAOSS ....................................... 244  
10.2.2 FaMOs Framework ....................................... 245  
10.2.3 Service Center Architecture ............................ 245  
10.2.4 RL Framework and Reputation Model ............... 246  

VII Appendix ......................................................... 249  
A Determination of the optimal policy under local entropy constraints 251  
B Convergence of the iterative updating rule ...................... 254  
C Computation of the expected cost-to-go in terms of the fundamental matrix ........................................ 258  
D Determination of an optimal policy ................................ 259  
E Computation of \( \sum_{l \neq n} (\partial n_l / \partial p_{kk'}) h_i \) ............................................................... 262  
F Updating rules for the basic model PMR1 ............... 264  
F.1 A study of the likelihood function ......................... 264  
F.2 The expectation step ......................................... 266  
F.3 The maximization step ...................................... 267  
G Updating rules for the model PMR2 involving truncation 269  
G.1 The expectation step ......................................... 269  
G.2 The maximization step ...................................... 273
Part I

Introduction
Chapter 1

Introduction

Contents

1.1 Context of the Research ................................. 4
   1.1.1 Service-Oriented Computing ....................... 4
   1.1.2 Multi-Agent System features for Service-Orientation 6
1.2 Principal Contributions ................................. 7
   1.2.1 Organizational Modeling and Requirements Engineering ....................... 8
   1.2.2 Architectural Design ............................... 9
   1.2.3 Detailed Design ................................. 10
1.3 Validation ............................................. 12
1.4 Limitations ........................................... 13
1.5 Reading Map ........................................... 14
   1.5.1 Part 2: Reinforcement Learning .................... 14
   1.5.2 Part 3: Reputation Models ......................... 15
   1.5.3 Part 4: Service-Oriented Systems .................. 16
   1.5.4 Part 5: Application ............................... 18

This chapter introduces the context and the motivation for the thesis. In Section 1.1, we describe the context of the thesis and the advantages of service-oriented computing. In Section 1.2, we formulate our research proposal and principal contributions. Section 1.3 presents research validation means while Section 1.4
1.1. Context of the Research

exposes the limitations. Finally, Section 1.5 proposes a reading map for the rest of the dissertation.

1.1 Context of the Research

Rapid evolutions in today’s businesses require new techniques and powerful tools to adapt and manage change in complex environments. Indeed, managing the complexity of systems is considered a key challenge in computing and is addressed through various approaches aimed at increased automation. Actually, it is now widely acknowledged that degrees of automation needed in response cannot be achieved without open, distributed, interoperable and modular information systems capable of dynamic adaptation to changing operating conditions.

One relevant response to such concerns is to rely on self-contained components, as in service-oriented computing which is a new engineering technique addressing part of those emerging issues and where a service is a self-describing, open component supporting rapid, low-cost composition of distributed applications. Indeed, service-oriented computing is expected to facilitate the building and evolution of everything from small to large-scale enterprise and other applications, by ensuring that the systems in question are, among other, scalable, evolvable, interoperable, and adapt cost-effectively and quickly to their users’ needs. Service-orientation is a polysemous term; in standardization efforts in industry, it is broadly understood to stand for a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains [51, 52].

1.1.1 Service-Oriented Computing

Service-Oriented Computing is a new paradigm for distributed computing platform which builds upon past distributed design principles and adds new concepts as design layers, governance considerations, and a vast set of preferred implementation technologies [51, 52]. Service-Oriented Computing (SOC) is described as a set containing its own design principles, a distinct architecture model and related distributed concepts [51, 52].

Given the above definition, computing platforms designed as SOC represent by essence distributed systems. In other words, a SOC can be define as a collection of distributed components grouped together because they relate to the same business context established by the designer or the business itself.
1.1. Context of the Research

In [51, 52], the author defines the primary parts of a typical SOC platform showing its fundamental complexion,

- **Service** provides a collection of capabilities related to a functional context established by the service definition. Thus, a service is a container of related capabilities well-defined and relatively isolated from other associated services [51, 52].

- **Service-Oriented Architecture (SOA)** is an architecture model improving the efficiency, agility, and productivity of an enterprise by positioning services as the primary means [51, 52].

- **Service Compositions** is an old principle in software engineering that lies at the heart of SOA and can be seen as a coordinated aggregate of services. Indeed, the successful realization of service reusability has a great deal to do with effective and repeated aggregation, and service composition is the mechanism coordinating this service aggregation, i.e., this service reusability.

- **Service-Oriented Architecture** is an architecture model improving the efficiency, agility, and productivity of an enterprise by positioning services as the primary means [51, 52].

- **Service Discoverability** is a mechanism allowing a service to be easily identified and described when opportunities for reuse present themselves.

- **Service-Oriented Architecture** (SOA) is an architecture model improving the efficiency, agility, and productivity of an enterprise by positioning services as the primary means [51, 52].

- **Service Compositions** is an old principle in software engineering that lies at the heart of SOA and can be seen as a coordinated aggregate of services. Indeed, the successful realization of service reusability has a great deal to do with effective and repeated aggregation, and service composition is the mechanism coordinating this service aggregation, i.e., this service reusability.

- **Service-Oriented Architecture** is an architecture model improving the efficiency, agility, and productivity of an enterprise by positioning services as the primary means [51, 52].

- **Service Compositions** is an old principle in software engineering that lies at the heart of SOA and can be seen as a coordinated aggregate of services. Indeed, the successful realization of service reusability has a great deal to do with effective and repeated aggregation, and service composition is the mechanism coordinating this service aggregation, i.e., this service reusability.
1.1. Context of the Research

![Service Inventory Diagram]

**Figure 1.1:** A service inventory establishes a pool of services, many which will be deliberately designed to be reused within multiple service compositions.

- **Service Inventory** integrates a set of complementary services in an independently standardized and bounded (i.e., within a boundary representing an enterprise or a meaningful segment of an enterprise) collection as shown in Figure 1.1.

1.1.2 Multi-Agent System features for Service-Orientation

Service-orientation is expected to enable the provision of a large number of distinct and competing services which the service requesters will be able to choose dynamically in the aim of receiving at all times optimal offerings for their purposes. Service-oriented systems ought to be open to permit many services to participate. Openness commits such systems to a distributed design for entering and leaving services are likely to be decentralized. To ensure that the service requesters continually receive optimal results when using a service-oriented system, the system should be made adaptable i.e., service provision ought to be performed by selecting and composing the participating services at runtime and according to multiple criteria, including constraints and preferences set by service requesters. Efficiency and flexibility that such systems are expected to exhibit are valuable given the pressing complexity concerns.

In this dissertation, we combine the multi-agent approach with the service-orientation paradigm. Multi-Agent Systems (MAS) appear to be more flexible, modular and robust than traditional systems including object-oriented ones [38, 63, 190]. They tend to be open and adaptable in the sense that they aim to cope with a changing organizational and operational environment where new
1.2. Principal Contributions

components can be added, modified or removed at any time. MAS are based on
the concept of agent which is an entity situated in some environment and able
of flexible autonomous action in order to meet its design objective. Three key
concepts support this definition:

• Situatedness: an agent receives input from the environment in which it
operates and can perform actions, which change the environment in some
way;
• Autonomy: an agent is capable to act without direct and continuous super-
vision. It has full control over its own actions;
• Flexibility: an agent is not only reactive but also pro-active. Reactivity
means that it has perceptions of the world inside which it is acting and
reacts to change in quasi real-time fashion. Proactivity means that be-
behavior is not exclusively reactive but it is also driven by internal goals, i.e.,
it may take initiative.

Considering these aspects, a MAS can be defined as an organization com-
piled of autonomous and proactive agents interacting with each other to achieve
common or private goals.

With regard to such features, designing a MAS service composition system
by a service-orientation approach is a relevant trade-off. Indeed, the service-
orientation paradigm gives a concrete view of pervasive environments where
heterogeneous services are provided by several designers, thus enabling to eas-
ily design complex and efficient systems through service composition. The MAS
paradigm complements this modular and flexible systems by providing high level
adaptation to users’ activity. The combination of both approaches allows the con-
ception of a fully adaptive system, which takes users activities and context into
account to adapt and reconfigure themselves. We also wish to address perfor-
ance issues thanks to a flexible agent approach were trade-offs between com-
putation/communication and optimality are easier to handle than in a centralized
system.

1.2 Principal Contributions

This thesis presents multiple contributions in various fields like reinforcement
learning, reputation model, service-oriented architecture and multi-agent mod-
elling. All these fields have a common characteristic; they can all be applied in
1.2. Principal Contributions

an open and adaptable (or dynamic) environment. In the sequel, we introduce these fields and their usage in an open and dynamic context.

Reinforcement learning is described by the problem an agent faces when it tries to learn in a dynamic environment without a teacher. Most of the reinforcement learning experiments presented in this thesis has been done in a dynamic context.

It is more interesting to use a reputation model in a dynamic environment where the agents, providers, services or any entities able to be scored can have a changing behavior or not constant performance. The reputation models presented in this dissertation are able to handle a dynamic environment. Some of the simulations have been done in a dynamic context where the providers and the consumers have unstable behavior. The analysis of results proves the robustness of our models.

A service-oriented architecture is also described in this dissertation. This architecture is adapted to open, dynamic and distributed systems and is able to support an effective, flexible, reliable, easy-to-use, low-cost, dynamic, time-efficient composition of distributed business services.

In conclusion, all of these fields can be articulated around the same subject, i.e., Adaptable and Open Service System (AOSS). The main contribution is the development of a process called ProDAOSS for developing adaptable and open service systems. Through ProDAOSS, we have integrated the various fields addressed in this dissertation in a common framework. As illustrated in Figure 1.2, ProDAOSS framework includes three standard stages: (i) Organizational modeling and requirements engineering, (ii) architectural design and (ii) detailed design.

1.2.1 Organizational Modeling and Requirements Engineering

On the basis of the weaknesses identified in the i*/Tropos *[38, 194] MAS development methodology, we propose a generic framework called FAMOOS for organizational modeling and requirements engineering using higher abstraction level elements: the services. The importance of distinguishing the system behavior at such level will be characterized and a set of models allowing services representation through multiple complementary views will be introduced. This framework is composed of a static high level services view, of a static and a dynamic decomposed services views.

*Tropos [38] is a MAS software development methodology founded on intentional and social concepts, inspired by early requirements analysis.
1.2. Principal Contributions

The main contributions of FaMOs are:

• A complete multiple view analysis framework for service-oriented modeling using i*. The framework also offers the fundamental elements to develop a complete model driven software project management framework. It can be considered as a startup for a global broader software engineering method;

• A dynamic model for representing services realization at defined quality of service level;

• Enhancements into the Strategic Services Model to model system threats;

1.2.2 Architectural Design

In order to respond to the requirements implied by a dynamic and open environment, we have defined a service-oriented MAS architecture called Service Center Architecture which is adaptable, open and distributed.

Deploying open, distributed, service-oriented, and self-organizing MAS that allow unbiased service provision driven by multiple concerns and tailored to user expectations, places very specific requirements on the properties of MAS architectures that can support such systems:

1. A simple architecture would minimize the variety of interactions between the participating agents. Ensuring interoperability would thus not involve high cost on the providers.

2. Internal functions as task allocation, reputation computation, etc, ought remain outside the responsibility of the entering and leaving agents to avoid bias. The architecture must therefore integrate a special class of agents that coordinate service provision and that are internal to (i.e., do not leave) the system.

3. Since there is no guarantee that agents will execute tasks at performance levels advertised by the providers, reputation calculation and task allocation should be grounded in empirically observed agent performance and such observation be executed by internal agents.

4. Varying QoS requests and change in the availability of agents require task allocation to be automated and driven by QoS, reputation scores and other relevant considerations (e.g., deadlines).
5. To ensure continuous optimization of system operation, task allocation within the architecture should involve continuous observation of agent performance, the use of available information to account for agent reputation or behavior, and exploration of new options to avoid excessive reliance on prior information.

A MAS architecture corresponding to the requirements 1–5 above is proposed. Overall, the architecture organizes agents into groups, called service centers. Each service center specializes in the provision of a single service. Within each center, a single service mediator agent receives service requests. Upon reception, the mediator decides how to allocate individual tasks that compose the service to corresponding task specialist agents, and this depending on observed prior performance while accounting for anticipated performance of newly entering agents. As no constraints are placed beyond organizing service delivery through mediators, the architecture is adapted to open and distributed systems.

Within such an architecture, the mediator organizes work by allocating tasks, negotiates with individual agents, and observes agent performance in order to adjust task allocation in the aim of continually optimizing quality criteria. Mediator behavior is guided by a reinforcement learning algorithm which is an efficient way of dynamically allocating tasks to agents, subsequently allowing the architecture to be characterized as adaptable to changes in the environment.

Another important contribution is the integration of reputation models in the MAS architecture which are used by a service reputation agent integrated in the proposed architecture. This agent uses a model computing the reputation scores of the agents in order to use these values as a parameter of the task allocation algorithm.

The suggested architecture, the mediator behavior and the reputation agent together allow the building of open, distributed, service-oriented MAS, adaptable to changes in the environment by accounting both on experience acquired through system operation and optimal exploration of new task allocation options. Because the principal contributions at this point are the architecture and the algorithms governing behavior of mediator and reputation agent, no specific commitments are made on, e.g., task allocation or negotiation protocols, to ensure the results are generic.

1.2.3 Detailed Design

At detailed design, we present two models integrated to the Service Center Architecture. The first model is a reinforcement learning model used by the me-
1.2. Principal Contributions

diator agent in order to allocate dynamically tasks to agents. The second model
is a probabilistic reputation model used by the reputation agent in order to com-
pute the reputation scores of the agents. In the sequel, we present separately the
contributions of these models.

1.2.3.1 Reinforcement Learning Model
In this dissertation, we present an unified reinforcement learning framework in-
tegrating exploitation and exploration for continual exploration. This framework
has three main contributions:

• The model proposed integrates exploration and exploitation into a single
framework, with the purpose of tuning exploration. Specifically, we quan-
tify the exploration rate as the entropy associated to the probability distri-
bution for choosing an action in a state. This allows to quantify and control
the exploration rate in a principled way for continual online exploration.

• The standard Boltzmann exploration strategy based on the Q-value is shown
to be optimal within this framework. It corresponds to the best policy
in terms of average cost needed to route the agents to the goal states.
Stochastic shortest-path and discounted problems are also studied within
this framework and links with the SARSA algorithm are investigated.

• Simulation experiments on dynamic environments confirm that the Boltz-
mann exploration strategy outperforms the $\epsilon$-greedy strategy.

We propose also two variants to this basic framework by extending the explo-
ration strategy (i.e., introduction of a global entropy) and the learning structure
(i.e., hierarchical learning and inclusion of macro and primitive actions).

1.2.3.2 Probabilistic Reputation Model
We describe a general procedure allowing to estimate reputation scores charac-
terizing a provider’s quality of service, based on transactions between consumers
(raters) and providers. As a by-product, some essential features of the raters,
such as their reactivity and their bias, are also estimated.

The procedure is based on a probabilistic model of consumer-provider inter-
actions whose parameters are estimated by a variant of the expectation-maximization
algorithm. Computer simulations show that the model is able to accurately re-
trieve the correct parameters, much more accurately than simply taking, as a
measure of reputation, the average of the ratings for each provider.
1.3 Validation

The ProDAOOS process has been applied to case studies. The use of the process on the development of web services and supply chain applications is described in Chapter 8 and 9.

The validation of the ideas developed in this thesis must be thought at several levels. Indeed, we can distinguish two types of contribution:

- The ProDAOSS software process itself that we will refer as macro-level in the rest of this section;
- The developed artefacts constituted of the FaMOs-models, the service-center architecture, the reinforcement learning algorithm and the probabilistic reputation models that we will refer as micro-level in the rest of this section.

At macro-level, the empirical evaluation of the benefits of AOSS process as development is hard to measure. This could be done by achieving a similar case study with ProDAOSS on the one side and with other AOSS methodologies on the other using development teams with similar experience. Such an experience is practically unachievable. However, if it was, we could measure:

- The developed solution with a couple of software engineering metrics to evaluate the impact of ProDAOSS development on each aspect as well as quantitative factors as execution time, solution performance, etc. Qualitative aspects such as user satisfaction, fault tolerance, qualitative aspects and, of course in the context of service-oriented development, flexibility could also be evaluated for the developed solutions to measure the process contributions;
- The software process performance in terms of development time, costs, produced documentation; etc.

At micro-level, developed artefacts can be evaluated by a comparison with existing ones, more precisely:

- The FaMOs framework has been compared to the traditional $i^*$ approach, benefits have been highlighted;
- The service-center architecture has been validated through the case study on web services.
1.4. Limitations

- For our reinforcement learning framework, the reported experiments show our framework outperforming two standard exploration methods, namely, $\epsilon$-greedy and naive Boltzmann.

- The experiments on our probabilistic reputation models suggest that these models are able to extract useful information from the ratings. We have also compared these models with two similar approach, namely, Brockhoff and Iterative Refinement models.

1.4 Limitations

The process description proposed in this dissertation constitutes a contribution to the definition of service-oriented MAS development. This dissertation focuses on the process description by conceptualizing a framework allowing to build MAS using a services approach. It models and introspects the process along different complementary stages. We, nevertheless, point out some important limitations of our research:

- At highest level, we only consider the design of cooperative MASs. Indeed, MAS may be either cooperative or competitive. In a cooperative MAS, the agents cooperate together in order to achieve common goals. Inversely, a competitive MAS is composed of agents that pursue personal goals and defend their own interests. The design of competitive MAS at higher level is left for future developments;

- The process needs to gain experience with its use. In this dissertation, we have applied it on two case studies. By doing so, we have tailored the process to particular projects and shown how our framework can help the design of MAS. However, it should be tested on more case studies and knowledge base should evolve from project to project;

- The methodology uses a couple of well defined artefacts, evolutions in aspects as reinforcement learning should be incorporated into the process to improve its performance. The process is service driven and each new contribution should be thought of as a - traceable - part (or contribution) of this approach.

- The methodology is mostly designed for industrial software projects involving lots of agents. It is especially suited for companies IS development rather than for stand alone developments.
1.5 Reading Map

In addition to the introduction and the conclusion, this dissertation is organized in four parts. Figure 1.2 presents the main structure of the thesis in relation to the process ProDAOSS.

![Thesis Structure Diagram]

**Figure 1.2:** Illustration of the thesis structure through ProDAOSS.

1.5.1 Part 2: Reinforcement Learning

This part introduces exclusively the reinforcement learning (RL) concepts. It presents some fundamentals of RL, a basic RL framework allowing to tune continual exploration and some extensions of this basic framework. Here, we focus only on RL model, independently of the rest of the thesis. In the sequel, we will integrate this model in the Service Center Architecture, as a detailed design stage of the process ProDAOSS.
1.5.1.1 Chapter 2: Tuning Continual Exploration in Reinforcement Learning

This chapter presents a RL framework allowing to tune continual exploration in an optimal way by integrating exploration and exploitation in a common framework. It first quantifies exploration by defining the degree of exploration of a state as the entropy of the probability distribution for choosing an admissible action in that state. Then, the exploration/exploitation tradeoff is formulated as a global optimization problem: find the exploration strategy that minimizes the expected cumulated cost, while maintaining fixed degrees of exploration at the states. The theoretical results are confirmed by simple simulations showing that the proposed exploration strategy outperforms the $\epsilon$-greedy strategy.

1.5.1.2 Chapter 3: Reinforcement Learning: Two Additional Extensions

This chapter describes two extensions of the basic RL framework introduced in Chapter 3. The first extension introduces a new ways of using the entropy by computing global entropy at the level of the entire network rather than a local entropy at the level of the nodes as made in the basic framework. It focalises more on the randomized shortest-path problem (RSP) than on a simple RL problem. Indeed, RL can be a potential application (but not the only application) of RSP which is more general and somewhat more theoretical. Thus, this first extension discusses a RSP model, which allows to introduce a limited degree of randomness to the shortest-path routing policy of an agent. Experimental results on small artificial problems are presented to demonstrate the properties of the proposed algorithm. The second extension introduces a hierarchy in the learning process. It addressed the problem of learning problems too large to be solved practically at the level of the most primitive actions. One strategy for overcoming this difficulty and used in this second model is to combine smaller actions into larger, temporally-extended actions, thus reducing the effective length of the solutions.

1.5.2 Part 3: Reputation Models

This part presents various probabilistic reputation models used thereafter as a detailed design stage of the process ProDAOSS in order to compute the agent reputation score in Service Center Architecture. As for Part 2, this part focus only on the reputation model independently of the rest of the thesis.
1.5.2.1 Chapter 4: A Probabilistic Reputation Model

This chapter introduces a probabilistic model of reputation. It is based on the following simple consumer-provider interaction model. Consumers are assumed to order items to providers, who each have some internal, latent, “quality of service” score. In the basic model, the providers supply the items with a quality following a normal law, centered on their internal “quality of service”. The consumers, after the reception of the item, rate it according to a linear function of its quality (a standard regression model). This regression model accounts for the bias of the consumer in providing ratings as well as his reactivity towards changes in item quality. Moreover, the constancy of the provider in supplying an equal quality level for delivering the items is estimated by the standard deviation of his normal law of item quality generation. Symmetrically, the consistency of the consumer in providing similar ratings for a given quality is quantified by the standard deviation of his normal law of ratings generation. Two extensions of this basic model are considered as well: a model accounting for truncation of the ratings and a Bayesian model assuming a prior distribution on the parameters. Expectation-maximization algorithms allowing to provide estimations of the parameters only based on the ratings are developed for all the models. The experiments suggest that these models are able to extract useful information from the ratings.

1.5.3 Part 4: Service-Oriented Systems

This part presents the process ProDAOSS and the following stages: (i) the Service Center Architecture as the architectural design stage, and (ii) the FaMOS framework as the organizational modelling and requirements engineering stages. The models presented in Parts 2 and 3 are also integrated directly in the Service Center Architecture as the detailed design stage of ProDAOSS.

1.5.3.1 Chapter 5: Service Center Architecture

This chapter proposes a MAS architecture, consisting of as many service centers as there are requests of distinct services. Within each service center, a mediator coordinates service delivery by allocating individual tasks to corresponding task specialist agents depending on their prior performance while anticipating performance of newly entering agents. Because no constraints are placed beyond organizing service delivery through mediator agents, the architecture is adapted to open and distributed systems. By basing mediator behav-
ior on a novel multicriteria-driven reinforcement learning algorithm (introduced in Chapter 2), integrating the exploitation of acquired knowledge with optimal, undirected, continual exploration, responsiveness of the system to changes in its environment is ensured. The reported experiments indicate the algorithm behaves as expected in the context of service center and outperforms two standard approaches. The reputation model presented in Chapter 4 is also included in this MAS architecture to study and report the agent performance. Indeed, the presented architecture relies on a multicriteria oriented MAS and allows the calculation of the reputation with a probabilistic model. A reputation score of a provider agent in a MAS quantifies past favorable or defavorable feedback of its client agents. Eliciting feedback and computing reputation scores is critical for a MAS since reputation affects trust between the agents and increases the sensitiveness of the system to variations of providers’ performance.

1.5.3.2 Chapter 6: Service-Oriented Framework for MAS Modeling

This chapter proposes an analysis framework (FaMOs) providing a multiple view process for conducting a software project through a model driven approach. The aim is to allow project managers to produce an organizational model that precisely captures the services the organisation offers as well as allowing risk management, time management, quality management i.e., software project management. To this end, the framework offers a conceptual meta-model identifying the elements needed services modeling through several complementary views. The approach allows the analyst to dispose of multiple perspectives offering various levels of knowledge so that he can adopt the most accurate one for dialogue with stakeholders. The framework takes roots in both requirements analysis and project management. It offers several complementary views of the organizational modeling problem and helps bridging the gap between requirements models and risk, quality and time management constraints.

1.5.3.3 Chapter 7: Process for Developing Adaptable and Open Service Systems

This chapter presents the process ProDAOSS used to develop adaptable and open service systems. At analysis level, the framework refines the traditional i* approach by including the FaMOs framework which introduces new models and refines existing ones. At design level, it defines a custom architecture as well as task allocation and reputation algorithms for the development of the mod-
1.5. Reading Map

elled problem converted in an architectural solution. The process is generic and flexible enough to be adapted to any MAS development methodology.

1.5.4 Part 5: Application

This part presents two applications of the ProDAOSS framework in the two different contexts: web services and supply chain management.

1.5.4.1 Chapter 8: Application to Web Services

Open service-oriented systems which autonomously and continually satisfy users’ service requests to optimal levels are an appropriate response to the need for increased automation of information systems. Given a service request, an open service-oriented system interprets the functional and nonfunctional requirements laid out in the request and identifies the optimal WS composition—that is, identifies web services (WS) whose coordinated execution optimally satisfies the various requirements laid out in the service request. The pool of services in such a system has three important characteristics: first, many WS are usually capable of executing the same functional requirements, so that it is necessary to select among competing WS based on their nonfunctional (i.e., quality) characteristics; second, new services may become available and others unavailable, so that there is no guarantee that a composition optimal at some point in time subsequently remains such—some of the needed services may be unavailable, or some new services may be available and more appropriate for a given request; finally, there is no guarantee that a service will execute as its provider advertises. Consequently, when producing service compositions it is relevant to: (1) revise WS compositions as new WS appear and other WS become unavailable; (2) use multiple criteria, including nonfunctional ones to choose among competing WS; (3) base WS comparisons on observed, instead of advertised performance; and (4) allow for uncertainty in the outcome of WS executions. To address issues (1)–(4), we propose the application of the Service Center Architecture (introduced in Chapter 5 and specialized here to WS) including the Randomized Reinforcement Learning (RRL) approach to WS composition. MCRRL learns and revises WS compositions using the multicriteria-driven (including various quality of service parameters, deadline, reputation, cost, and user preferences) reinforcement learning algorithm presented in previous chapters, which integrates the exploitation of acquired data about individual services’ past performance with optimal, undirected, continual exploration of new compositions which involve services
whose behavior has not been observed. The RRL enables adaptability to actual performance of WS and changes in their availability. The reported experiments indicate the algorithm behaves as expected and outperforms two standard approaches.

1.5.4.2 Chapter 9: Application to Supply Chain Management

The process presented in this dissertation allows designing flexible and adaptable software systems that can be easily adopted on demand by software customers. Those benefits are of primary importance in the context of supply chain management, that is why this chapter proposes to apply ProDAOSS, a process for developing adaptable and open service systems to an industrial case study in outbound logistics. ProDAOSS covers the whole software development life cycle. At analysis level, flexible business processes were generically modelled with different complementary views. First of all, an aggregate services view of the whole applicative package is offered; then services are split using an agent ontology - through the i* framework - to design a flexible multi-agent system. A dynamic view completes the documentation by offering the service realization paths. At design stage, the service center architecture proposes a reference architectural pattern for services realization in an adaptable and open manner.
Part II

Reinforcement Learning
Chapter 2

Tuning Continual Exploration in Reinforcement Learning

Contents

2.1 Related work ........................................ 26
2.2 Basic framework ...................................... 27
  2.2.1 Computation of the total expected cost for a given policy ........................................ 29
  2.2.2 Degree of exploration as entropy ............... 30
2.3 Optimal policy under exploration constraints for deterministic shortest-path problems .......... 30
  2.3.1 Computation of the optimal policy .............. 32
  2.3.2 Some limit cases ................................. 34
2.4 Optimal policy for directed acyclic graphs .......... 35
  2.4.1 Construction of a directed acyclic graph ....... 35
  2.4.2 Computation of the optimal policy .............. 36
2.5 Optimal policy under exploration constraints for stochastic shortest-path problems ............ 36
  2.5.1 Equations for optimal policy and expected cost ........................................ 37
  2.5.2 Online estimation of the expected cost ........ 37
  2.5.3 Links with Q-learning and SARSA .............. 38
2.6 Optimal policy under exploration constraints for discounted problems ........................ 40

23
CHAPTER 2. TUNING CONTINUAL EXPLORATION IN REINFORCEMENT LEARNING

2.7 Experiments ........................................... 41
   2.7.1 Preliminary experiments ....................... 41
   2.7.2 Some experimental comparisons ............. 51
   2.7.3 The puzzle problem ............................ 54

Motivations

Reinforcement learning is described by the problem an agent faces when it tries to learn in a dynamic environment without a teacher [171]. The agent can only learn by systematic trial-and-error interactions with the environment of which it receives reinforcement signals (i.e., reward or cost). Intuitively, it can be compared with primitive learning, as animal learning for example. Indeed, animals acquire complex behaviors by learning how to maximize rewards and to minimize punishments, and reinforcement learning theory is a formal computational model of this type of learning. More generally, the objective of a reinforcement learning learner is to discover a policy, i.e., a mapping from situations to actions, so as to maximise the reward (or similarly, minimize the cost) it receives. The reinforcement signal can be delayed or only become apparent after a number of actions are performed [171]. This means that the agent faces a credit-assignment problem, that is, deciding which action or sequence of actions should get credit for the received reward.

An issue central to reinforcement learning is the tradeoff between exploration and exploitation [90, 171]. Exploration aims to search for new ways of solving the problem while exploitation aims to capitalize on already well-established solutions. Exploration is especially relevant when the environment is changing. Then good solutions can deteriorate and better solutions can appear over time. Without exploration, the system sends agents only along the currently best paths without exploring alternative paths. It therefore remains unaware of the changes and its performance inevitably deteriorates with time. A key feature of reinforcement learning is that it explicitly addresses the exploration/exploitation issue as well as the online estimation of the associated probability distributions in an integrated way [171].

This chapter presents a unified framework integrating exploitation and exploration for continual exploration. Technically speaking, exploration is defined as the association of a probability distribution to the set of available control actions in each state (choice randomization or stochastic policy). We propose to quantify the degree of exploration in a state as the (Shannon) entropy [40, 93] of this probability distribution. If no exploration is performed, the agents are routed on the best path with probability one – they just exploit the solution. With explo-
CHAPTER 2. TUNING CONTINUAL EXPLORATION IN REINFORCEMENT LEARNING

ration, the agents continually explore a possibly changing environment to keep current with it. When the entropy is zero in a state, no exploration is performed from that state, while a maximal entropy corresponds to a blind exploration with equal probability of choosing any available action.

The online exploration/exploitation issue is then stated as a global optimization problem: learn the exploration strategy that minimizes the expected cumulated cost from the initial state to a goal (the destination state) while maintaining a fixed degree of exploration at each state. In other words, maximize exploitation for constant exploration. This approach leads to a set of nonlinear equations defining the optimal solution. These equations can be solved by iterating them until convergence, which is proved for a stationary deterministic environment and a particular initialization strategy. Their solution provides the action policy (the probability distribution for choosing an action in each state) that minimizes the expected cost from the initial state to a destination state, given the degree of exploration in each state. Interestingly, the optimal action policy satisfying entropy constraints at each state is simply a Boltzmann distribution involving the Q-value, one of the standard exploration strategies [171]. Moreover, when the degree of exploration is zero in all states (no exploration), the nonlinear equations reduce to the Bellman equations for finding the shortest path from the initial state to a destination state. The main drawback of the method is that it is computationally demanding since it relies on iterative algorithms like the value-iteration algorithm.

The present chapter is organized as follows. Section 2.1 presents the related work. Section 2.2 introduces the notation, the standard deterministic shortest-path problem, and how we manage continual exploration. Section 2.3 describes our algorithm for dealing with exploration in deterministic shortest-path problems, as defined for instance in [21], where control actions deterministically drive agents to a single successor state. The special case where the graph of states is directed and acyclic is treated in Section 2.4, where it is shown that the nonlinear equations can easily be solved by performing a single backward pass from the destination state. Section 2.5 discusses stochastic shortest-path problems, where actions have uncertain effects. A short discussion of discounted problems follows in Section 2.6. Some numerical examples and comparisons with an $\epsilon$-greedy exploration strategy are provided in Section 2.7. Part of this work is a joint work together with Fouss, Pirotte and Saerens and has been published in [4, 5].
2.1 Related work

This chapter is devoted to continual, on-line, exploration. Indeed, *preliminary* or *initial exploration* must be distinguished from *continual online exploration*. The objective of *preliminary exploration* is to discover relevant goals, or destination states, and to estimate a first, possibly optimal, policy for reaching them using search methods developed in artificial intelligence [92]. *Continual online exploration* aims to keep exploring the environment after the preliminary exploration stage, in order to adjust the policy to changes in the environment.

*Preliminary exploration* can be conducted in two ways [177, 178, 179, 183]. A first group of strategies, often referred to as *undirected exploration*, explore at random; control actions are selected with a probability distribution, taking the expected cost into account. The second group, referred to as *directed exploration*, use domain-specific knowledge for guiding exploration. *Directed exploration* usually provides better results in terms of learning time and cost. The main idea behind directed methods of exploration is to encourage the agent to explore either parts of the state space that have not been sampled often or parts that have not been sampled recently [135]. Sophisticated directed exploration strategies have been developed by using, for instance, importance sampling [118], back-propagation of uncertainty [117], or the Metropolis algorithm popular in combinatorial optimization [68].

*Online exploration* is performed by re-exploring the environment periodically or continually [26, 160], usually with an $\epsilon$-greedy or a Boltzmann exploration strategy [119, 171]. For instance, a joint estimation of the exploration strategy and the state-transition probabilities for continual online exploration can be performed within the SARSA framework [147, 166, 171].

Yet another attempt to integrate exploration and exploitation, this time in a temporal-difference algorithm, is presented in [43], where the existence of at least one fixed point is proven. In [138], Peshkin & Savova estimate a randomized policy for packet routing in network communication by policy search via gradient ascent. The algorithm is able to adapt the policy to changing environments and extends the algorithm of Boyan & Littman [26]. Policy-search algorithms directly work in the policy space, trying to maximize the expected reward directly in the policy space.

Our work can also be viewed as addressing *randomized shortest-path problems* (RSP). We define a RSP as a shortest-path problem to which exploration constraints (here, specifically, entropy constraints) are added in order to obtain optimal randomized policies (also called stochastic policies), and therefore cont-
tinual exploration. The entropy constraint can be set at the node level, as proposed in this chapter, or at the network level, as proposed in the next chapter. In the latter case, the total entropy spread in the network is fixed and an optimal policy is sought, minimizing the expected cost for routing the agents. To our knowledge, optimal randomized strategies, while popular, for instance, in game theory (see for instance [128]) and Markov games [105], have not yet been exploited in the context of shortest-path problems. One noticeable exception is the work of Akamatsu [10] which proposes a randomized policy in the context of transportation networks. In transportation studies, randomized strategies are called stochastic traffic assignments and, within this context, the model of [10] is the reference. It provides a probability distribution for following an edge at any node of the network. More precisely, it proposes to design the transition probability matrix in such a way that long paths to the destination are penalized.

Also related to this chapter, the work of Tahbaz-Salehi & Jadbabaie [173] introduces a one-parameter family of algorithms that, like our algorithm, reduces to both the Bellman-Ford procedure for finding shortest paths and the iterative algorithm for computing the average first-passage time. However, it is based on heuristic grounds and not on a well-defined cost function to be optimized. Moreover, it does not provide a randomized policy. Todorov [181] studies a family of Markov decision problems that are linearly solvable, that is, for which a solution can be computed by solving a matrix eigenvector problem. To make this possible, a special form is assumed for the control of the transition probabilities, which transforms the problem of finding the policy into an eigenvector problem. In [27], a Markov chain is designed that has fastest mixing properties; in [169] the same group of researchers discuss its continuous-time counterpart. In a completely different framework, uninformed random walks, based on maximizing the long-term entropy [44, 182], have recently been proposed as an alternative to the standard PageRank algorithm.

### 2.2 Basic framework

We consider that agents are sent from an initial state $k_0$ and that, at time $t$, they choose a control action $u$ in state $s_t = k$ ($n$ states in total) with a probability distribution $\pi_k(u)$, $k \in \{1, 2, \ldots, n\}$. Agents then perform the chosen action (say, $i$), observe and incur a finite cost $c(k, i)$ (here, we are optimizing costs instead of rewards) associated with executing action $i$, and jump to the next state $s_{t+1} = k'$. In this framework, we consider costs instead of re-
2.2. Basic framework

wards, depending only on the current state and the performed action. The **policy** \( \Pi \equiv \{ \pi_k(u), k = 1, 2, \ldots, n \} \) defines for each state \( k \) a probability distribution \( \pi \) on the set \( U(k) \) of actions available in this state \( k \). For instance, if the available actions in some state \( k \) are \( U(k) = \{1, 2, 3\} \), the distribution \( \pi_k(u) \) specifies three probabilities \( \pi_k(u = 1), \pi_k(u = 2) \), and \( \pi_k(u = 3) \) summing to one. Such random choices are common in a variety of fields, for instance decision sciences [142] or game theory, where they are called mixed strategies (see, e.g., [128]). Thus, the problem is a **randomized shortest-path problem** (RSP) as characterized above.

The cost \( c(k, i) \) can be positive (penalty), negative (reward), or null. In a nonstationary environment, it may vary over time. We assume that no cycle exists whose total cost is negative. This is a standard requirement in shortest-path problems [39]; indeed, if such a cycle existed, then traversing it an arbitrary large number of times would result in a path with an arbitrary small cost so that a best path could not be defined. In particular, this implies that, if the graph of states is nondirected, all costs are nonnegative. We also assume that the costs are additive, that is, they are summed along a path followed by an agent. We will use path cost and path length interchangeably, and shortest-path for lowest-cost path.

We also assume that, once action \( i \) has been chosen in state \( k \), the next state \( k' = f_k(i) \) is known deterministically and that different actions lead to different states. This framework corresponds to a **deterministic shortest-path problem** [21]. A simpler model of this problem does without actions and directly defines state-transition probabilities. We chose a general formalism that also fits **stochastic shortest-path problems**, the full stochastic problems where both the choice of actions and the state transitions are governed by probability distributions (they are studied in Section 2.5). We assume, as in [21], that there is a special cost-free destination or goal state \( d \); once the system has reached that state, it remains there at no further cost (discounted problems are studied in Section 2.6).

The goal is to minimize the **total expected cost** \( V_{\pi}(k_0) \) (Equation (2.1)) accumulated along a path \( s_0, s_1, \ldots \) in the graph starting from an initial (or source) state \( s_0 = k_0 \) and ending in the goal state:

\[
V_{\pi}(k_0) = E_{\pi} \left[ \sum_{t=0}^{\infty} c(s_t, u_t)|s_0 = k_0 \right] \tag{2.1}
\]

The expectation \( E_{\pi} \) is taken on the policy \( \Pi \), that is, on all the random choices of action \( u_t \) in states \( s_t \).
2.2. Basic framework

We consider a problem structure such that termination is guaranteed, at least under an optimal policy. Thus, the horizon is finite, but its length is random and it depends on the policy. The conditions for which termination holds are equivalent to requiring that the destination state be reachable in a finite number of steps from any possible initial state (for a rigorous treatment, see [21, 23]).

2.2.1 Computation of the total expected cost for a given policy

The problem can be represented as a Markov chain (see, e.g., [21]) where actions correspond to transitions between states. Then, the destination state is an absorbing state, i.e., a state with no outgoing link, and the computation of the expected cost (2.1) from any state \( k \) is closely related to the computation of the average first-passage time in the associated Markov chain [96]. The average first-passage time is the average number of steps that a random walker, starting from the initial state \( k_0 \), will take in order to reach destination state \( d \). It can easily be generalized to take the cost of the transitions into account [57]. By first-step analysis (see for instance [174]), it can be shown that, if the policy is fixed, \( V_\pi(k) \) can be computed through the following equations

\[
\begin{cases}
V_\pi(k) &= \sum_{i \in U(k)} \pi_k(i) [c(k, i) + V_\pi(k'_i)], \text{ with } k \neq d \\
V_\pi(d) &= 0, \text{ where } d \text{ is the destination state}
\end{cases}
\] (2.2)

where \( k'_i \) is the state resulting from applying control action \( i \) in state \( k \), i.e., \( k'_i = f_k(i) \). Equations (2.2) can be solved by iteration or by inverting the so-called fundamental matrix [96]. They are analogous to the Bellman equations in Markov decision processes (see, e.g., [15]).

The same framework easily handles multiple destinations by defining one absorbing state for each destination. If the destination states are \( d_1, d_2, \ldots, d_m \), then

\[
\begin{cases}
V_\pi(k) &= \sum_{i \in U(k)} \pi_k(i) [c(k, i) + V_\pi(k'_i)], \text{ where } k'_i = f_k(i) \text{ and } k \neq d_1, d_2, \ldots, d_m \\
V_\pi(d_j) &= 0 \text{ for all destination states } d_j, j = 1, \ldots, m
\end{cases}
\] (2.3)

We now address the following questions: (i) how to quantify exploration, (ii) how to control the random choices (i.e., exploration), and (iii) how to compute the optimal policy for a given degree of exploration.
2.2.2 Degree of exploration as entropy

We define the degree of exploration \( E_k \) at each state \( k \) as

\[
E_k = -\sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) \tag{2.4}
\]

This is the Shannon entropy of the probability distribution of choosing control actions in state \( k \) [40, 93]. \( E_k \) is equal to zero when there is no uncertainty \( (\pi_k(i) = \text{then reduces to a Kronecker delta); } E_k \) is equal to \( \log(n_k) \), where \( n_k \) is the number of available choices in state \( k \), in the case of maximum uncertainty, where \( \pi_k(i) = 1/n_k \) (a uniform distribution). The Shannon entropy is a natural choice for characterizing exploration since it quantifies the uncertainty associated with the probability of outcomes (control actions in the present case). Other choices are conceivable (see, e.g., [40, 93]). Of course, the optimal policy depends on the definition of this exploration measure; other choices will lead to other optimal policies. The exploration rate \( E_k^r = E_k / \log(n_k) \), with values comprised between 0 and 1, is the ratio between \( E_k \) and its maximum value.

Fixing the entropy at a state sets the exploration level out of the state; increasing the entropy increases exploration up to the maximal value, in which case there is no exploitation since the next action is chosen completely at random, with a uniform distribution.

Another choice would be to fix the global entropy \( E \) spread in the network instead of at the node level:

\[
E = -\sum_k u_k \sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) \tag{2.5}
\]

where \( u_k \) is some weighting factor. This alternative randomized shortest-path model leads to an optimal exploration strategy different from the one obtained in this chapter (the standard Boltzmann exploration) by using constraint (2.4) and will be studied in the next chapter.

2.3 Optimal policy under exploration constraints for deterministic shortest-path problems

We seek a policy \( \Pi \equiv \{ \pi_k(u); k = 1, 2, \ldots, n \} \) for which the expected cost \( V_n(k_0) \) from the initial state \( k_0 \) is minimal when the degree of exploration in each state \( k \) is maintained constant. The problem is thus to find the policy \( \Pi \) leading
2.3. Optimal policy under exploration constraints for deterministic shortest-path problems

ten to the minimal expected cost \( \Pi^* = \arg \min_{\Pi} \{ V(k_0) \} \). It can be formulated as a constrained optimization problem. Equivalently, it amounts to construct the optimal transition-probability matrix \( P^* \) of the associated Markov chain.

The optimal probability distribution \( \pi_k^*(i) \) for choosing control actions \( i \) in state \( k \) (which is a multinomial logistic function) is derived in Appendix A:

\[
\pi_k^*(i) = \frac{\exp \left[ -\theta_k (c(k, i) + V^*(k'_i)) \right]}{\sum_{j \in U(k)} \exp \left[ -\theta_k (c(k, j) + V^*(k'_j)) \right]},
\]

(2.6)

where \( k'_i = f_k(i) \) is the successor state of state \( k \) and \( V^* \) is the optimal (minimum) expected cost given by

\[
\begin{cases}
V^*(k) = \sum_{i \in U(k)} \pi_k^*(i) [c(k, i) + V^*(k'_i)], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\
V^*(d) = 0, \text{ for the destination state } d
\end{cases}
\]

(2.7)

This random choice strategy (2.6) is often called Boltzmann distributed exploration [171] and it will be referred to as Boltzmann distribution in the sequel. In Equation (2.6), the parameter \( \theta_k \), corresponding to the inverse of the temperature, must be chosen in order to satisfy

\[
-\sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) = E_k
\]

for each state \( k \) and predefined \( E_k \), where \( \pi_k(i) \) depends on \( \theta_k \) through Equation (2.6). \( E_k \) takes its values in \([0, \infty]\). Of course, if, for some state, the number of possible control actions reduces to one (i.e., no choice), no entropy constraint is introduced. Since Equation (2.8) has no analytical solution, \( \theta_k \) must be computed numerically from \( E_k \). It can be shown that the function \( \theta_k(E_k) \) is strictly monotonic decreasing, so that a line search algorithm (such as the bisection method; see for instance [16]) can efficiently find the value of \( \theta_k \) corresponding to a given value of \( E_k \).

Equation (2.6) has a simple appealing interpretation: in state \( k \), choose preferably (with highest probability) action \( i \) leading to state \( k'_i \) of lowest expected cost \( c(k, i) + V^*(k'_i) \). Thus, the agent is routed preferably to the state that is closest (on average) to the destination state.

The same necessary optimality conditions can also be expressed in terms of the \( Q \)-values of the popular \( Q \)-learning framework [171, 186, 187]. Indeed,
2.3. Optimal policy under exploration constraints for deterministic shortest-path problems

In the deterministic case, the $Q$-value represents the expected cost from state $k$ when choosing action $i$: $Q(k, i) = c(k, i) + V(k')$. Thus $V(k) = \sum_{i \in U(k)} \pi_k(i) Q(k, i)$ and

$$
\begin{align*}
Q^*(k, i) &= c(k, i) + \sum_{j \in U(k')} \pi^*_k(j) Q^*(k', j), \\
Q^*(d, i) &= 0, \text{ for the destination state } d
\end{align*}
$$

(2.9)

and the $\pi^*_k(i)$ are given by

$$
\pi^*_k(i) = \frac{\exp[-\theta_k Q^*(k, i)]}{\sum_{j \in U(k)} \exp[-\theta_k Q^*(k, j)]}
$$

(2.10)

which corresponds to a Boltzmann exploration involving the $Q$-value. Thus, a Boltzmann exploration may be considered optimal, since it provides the best expected performance when degrees of exploration, quantified as the Shannon entropy, are given.

2.3.1 Computation of the optimal policy

Equations (2.6) and (2.7) suggest an iterative procedure, very similar to the well-known value-iteration algorithm in Markov decision processes [15, 180], for computing both the expected cost and the policy.

Specifically, when agents perform an action $i$, they observe and incur the associated cost $c(k, i)$ (which, in a nonstationary environment, may vary over time) and they jump to the next state $k'$. Agents then update the estimates of the policy and of the average cost until destination, noted as $\hat{\pi}_k(i)$ and $\hat{V}(k)$, respectively. The iterative updating procedure can be summarized as follows.

1. Initialization phase:

   - Choose an initial policy $\hat{\pi}_k(i)$, for all $i$ and $k$, satisfying the exploration constraints (2.8).
   - Compute the expected cost until destination $\hat{V}(k)$ with any procedure for solving the linear equations (2.7) where $V^*(k)$ and $\pi^*_k(i)$ are substituted by $\hat{V}(k)$ and $\hat{\pi}_k(i)$, respectively. The $\hat{\pi}_k(i)$ are kept fixed in the initialization phase. Any standard iterative procedure (for instance, a Gauss-Seidel-like algorithm) for computing the expected cost until absorption in a Markov chain can be used (see [96]).
2.3. Optimal policy under exploration constraints for deterministic shortest-path problems

2. Iteration phase:

For each visited state \( k \), do:

- Choose an action \( i \in U(k) \) with current probability estimate \( \hat{\pi}_k(i) \), incur the current cost \( c(k, i) \), and jump to the next state \( k' \).
- Update the probability distribution for state \( k \) as:
  \[
  \hat{\pi}_k(i) \leftarrow \frac{\exp\left[-\hat{\theta}_k \left(c(k, i) + \hat{V}(k'_i)\right)\right]}{\sum_{j \in U(k)} \exp\left[-\hat{\theta}_k \left(c(k, j) + \hat{V}(k'_j)\right)\right]},
  \]
  (2.11)
  where \( k'_i = f_k(i) \) and \( \hat{\theta}_k \) is set in order to satisfy the predefined degree of entropy (for instance a line search; see Equation (2.8) and the discussion following this equation) for this state.
- Update the expected cost from state \( k \):
  \[
  \begin{cases}
  \hat{V}(k) &\leftarrow \sum_{i \in U(k)} \hat{\pi}_k(i) \left[c(k, i) + \hat{V}(k'_i)\right], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\
  \hat{V}(d) &\leftarrow 0, \text{ where } d \text{ is the destination state }
  \end{cases}
  \]
  (2.12)

Thus, the procedure allows to optimize the expected cost \( V(k_0) \) and to obtain a local minimum for it. It is not guaranteed to converge to a global minimum, however. Whether \( V(k_0) \) has only one global minimum or many local minima remains an open question in general. Convergence of these equations is proved in Appendix B for a stationary environment.

Simpler initialization schemes can also be applied. For instance, initially set \( c(k, i) = 0 \), \( \hat{\pi}_k(i) = 1/n_k \), \( \hat{V}(k) = 0 \), where \( n_k \) is the number of available actions in state \( k \); then proceed with updating rules (2.11) and (2.12). This simple scheme has been used in all our experiments. Although convergence has not been proved with it (our convergence proof uses the initialization phase shown in the iteration algorithm above), we did not observe any convergence problem.

Performing a line search for computing the value of \( \theta \) is computationally expensive, as will be shown in the experimental sections 2.7.1.3 and 2.7.2. Indeed, at each iteration step of the algorithm, a line search for computing \( \theta_k \) is performed for each node \( k \) of the graph. Thus, instead of fixing entropy \( E_k \) at each state, one could simply fix \( \theta_k \), therefore avoiding the need for computing \( \theta_k \) in
2.3. Optimal policy under exploration constraints for deterministic shortest-path problems

terms of $E_k$. This, of course, corresponds to the standard Boltzmann strategy. This choice is still optimal in our sense since temperature and entropy are related by a one-to-one mapping (constant entropy means constant temperature). Still, it could be argued that entropy is a more sensible measure of the exploration performed from a node.

Concerning the algorithm complexity presented in this section, it is quite similar to the complexity of a Markov decision process (MDP) which is not simple to characterize but seems to be polynomial (for instance, see [131] for more details on the MDP complexity).

2.3.2 Some limit cases

We now show that, when the degree of exploration is zero for all states, the nonlinear equations reduce to the Bellman equations for finding the shortest path from the initial state to the destination state.

Indeed, from Equations (2.11)-(2.12), if $\hat{\theta}_k$ is large, corresponding to a near-zero entropy, the probability of choosing the action with the lowest value of $(c(k, i) + \hat{V}(k'))$ dominates all the other probabilities. In other words, $\hat{\pi}_k(j) \simeq 1$ for action $j$ corresponding to the lowest average cost (including the action cost), while $\hat{\pi}_k(i) \simeq 0$ for the other alternatives $i \neq j$. Equations (2.12) can therefore be rewritten as

\[
\begin{align*}
\hat{V}(k) &\leftarrow \min_{i \in U(k)} [c(k, i) + \hat{V}(k')] \text{, with } k' = f_k(i) \text{ and } k \neq d \\
\hat{V}(d) &\leftarrow 0, \text{ where } d \text{ is the destination state}
\end{align*}
\]

which are the Bellman equations for finding the shortest path to the destination state [19, 21]. In terms of $Q$-values, the optimality conditions reduce to

\[
\begin{align*}
Q^*(k, i) &= c(k, i) + \min_{j \in U(k')} Q^*(k', j), \text{ with } k' = f_k(i) \text{ and } k \neq d \\
Q^*(d, i) &= 0, \text{ for the destination state } d
\end{align*}
\]

which corresponds to the standard $Q$-learning for the deterministic case.

On the other hand, when $\hat{\theta}_k = 0$, the probability distribution reduces to $\hat{\pi}_k(i) = 1/n_k$ and the degree of exploration is maximum for all states. The nonlinear equations then reduce to linear equations allowing to compute the average cost for reaching the destination state from the initial state in a Markov chain with transition probabilities equal to $1/n_k$. In other words, a “blind” random exploration is then performed that does not take costs into account.
2.4. Optimal policy for directed acyclic graphs

Any intermediate setting $0 < E_k < \log(n_k)$ leads to an optimal exploration vs. exploitation strategy minimizing the expected cost and favoring short paths to the solution. In the next section, we show that, if the graph of states is directed and acyclic, the nonlinear equations can easily be solved by performing a single backward pass from the destination state.

2.4 Optimal policy for directed acyclic graphs

This section shows that the computation of the optimal policy is greatly simplified with acyclic graphs. We first describe an interesting procedure, proposed by Dial [45], which simplifies the graphical model in order to obtain an acyclic graph. An acyclic graph is a graph for which there is no cycle, that is, one can never return to the same state.

2.4.1 Construction of a directed acyclic graph

Dial [45] proposed the following procedure to define a meaningful acyclic graph from the original one:

- Compute the minimum cost $m(k_0, k)$ from the initial state $k_0$ to any other state $k$, by using for instance the Bellman-Ford algorithm [39].

- Suppress all actions associated with state transitions $k \rightarrow k'$ for which $m(k_0, k') \leq m(k_0, k)$; the remaining actions are “efficient” and are retained. Indeed, it is natural to consider that “efficient paths” should always lead away from the origin $k_0$.

This procedure results in an acyclic graph since states can be ordered and relabeled with increasing values of $m(k_0, k)$ from 0 (initial state) to $n$, the number of nodes. Ties are ordered arbitrarily. The destination state is noted $d$. Of course, this procedure can be adapted to the problem at hand. For instance, instead of computing the minimum cost from the initial state, one could compute the cost to the destination state $m(k, d)$ and eliminate actions according to this criterion. Other measures than the minimum cost could be used as well.

One drawback of this procedure is that the environment (the topology and the cost) must be known beforehand.
2.4.2 Computation of the optimal policy

When the states form a directed acyclic graph, the procedure described in Section 2.3.1 simplifies greatly. Indeed, if $U(k)$ now represents the set of “efficient actions” for each state, the algorithm (2.11)-(2.12) reduces to

1. **Initialization phase:** $\hat{V}(d) = 0$, for the destination state $d$.

2. **Computation of the policy and the expected cost under exploration constraints:**

   For $k = (d - 1)$ to initial state $0$, compute:

   $$
   \begin{align*}
   \hat{\pi}_k(i) &\leftarrow \exp \left[ -\hat{\theta}_k \left( c(k, i) + \hat{V}(k'_i) \right) \right] \\
   \hat{V}(k) &\leftarrow \sum_{i \in U(k)} \hat{\pi}_k(i) [c(k, i) + \hat{V}(k'_i)], \text{ with } k \neq d
   \end{align*}
   $$

   where $k'_i = f_k(i)$ and $\hat{\theta}_k$ is set in order to satisfy the prescribed degree of entropy in each state (see Equation (2.8)).

   Since states have been relabeled with increasing values of $d(k_0, k)$ and as efficient actions always lead to a state with higher label, the values of the expected cost at states $k'_i$ can be computed with equations (2.15).

2.5 Optimal policy under exploration constraints for stochastic shortest-path problems

With stochastic shortest-path problems, when an action is performed, the transition to the next state is no longer deterministic but stochastic [21]. More precisely, when agents choose action $i$ in state $k$, they jump to state $k'$ with a transition probability $P(s_{t+1} = k' | u = i, s_t = k) = p_{kk'}(i)$. Thus there are now two distinct probability distributions:

- $\pi_k(i)$: the probability of choosing action $i$ in state $k$;
- $p_{kk'}(i)$: the probability of jumping to state $k'$ after having chosen action $i$ in state $k$.

We now extend our randomized shortest-path model to stochastic shortest-path problems.
2.5. Optimal policy under exploration constraints for stochastic shortest-path problems

### 2.5.1 Equations for optimal policy and expected cost

In [21, 139], the authors show how to obtain recurrence relations for the expected cost $V_\pi(k)$ given a policy $\Pi$ in a heuristic way by first-step analysis:

\[
\begin{aligned}
V_\pi(k) &= \sum_{i \in U(k)} \pi_k(i) \left[ c(k, i) + \sum_{k' = 1}^{n} p_{kk'}(i) V_\pi(k') \right], \\
V_\pi(d) &= 0, \text{ where } d \text{ is the destination state}
\end{aligned}
\]  

(2.16)

By defining the average cost after choosing control action $i$ in state $k$ as $\overline{V}_\pi(k, i) = \sum_{k'} p_{kk'}(i) V_\pi(k')$, Equation (2.16) can be rewritten as

\[
\begin{aligned}
V_\pi(k) &= \sum_{i \in U(k)} \pi_k(i) \left[ c(k, i) + \overline{V}_\pi(k, i) \right], \\
V_\pi(d) &= 0, \text{ where } d \text{ is the destination state}
\end{aligned}
\]  

(2.17)

By proceeding as described in Appendix A, the optimal policy is obtained by substituting $V^*(k')$ by $\overline{V}^*(k, i)$ in both Equation (2.6) and (2.7):

\[
\pi^*_k(i) = \frac{\exp \left[ -\theta_k \left( c(k, i) + \overline{V}^*(k, i) \right) \right]}{\sum_{j \in U(k)} \exp \left[ -\theta_k \left( c(k, j) + \overline{V}^*(k, j) \right) \right]}
\]  

(2.18)

The additional difficulty here, in comparison with a deterministic problem, is that, if unknown, the probability distributions $p_{kk'}(i)$ have to be estimated online, together with the costs and the distribution of the randomized control actions [171].

### 2.5.2 Online estimation of the expected cost

An alternative solution is to directly estimate the expected cost $\overline{V}_\pi(k, i) = \sum_{k'} p_{kk'}(i) V_\pi(k')$ from the observation of the value of $V_\pi(k')$ in the next state $k'$. There is a large range of potential techniques for doing that, depending on the problem at hand (see for example [30]). One could simply use exponential smoothing, leading to

\[
\hat{V}(k, i) \leftarrow \alpha \hat{V}(k') + (1 - \alpha) \overline{V}(k, i)
\]  

(2.19)
2.5. Optimal policy under exploration constraints for stochastic shortest-path problems

with $\alpha$ fixed. Alternatively, one can rely on a stochastic approximation scheme by letting $\alpha(t)$ decrease over time $t$:

$$
\hat{V}(k, i) \leftarrow \hat{V}(k, i) + \alpha(t) \left[ \hat{V}(k') - \hat{V}(k, i) \right]
$$

(2.20)

which converges for a suitable decreasing policy of $\alpha(t)$ [168].

This leads to the following updating rules. For each visited state $k$, do:

- Choose an action $i$ with current probability estimate $\hat{\pi}_k(i)$ and observe the current cost $c(k, i)$ for performing this action.

- Perform action $i$ and observe the value $\hat{V}(k')$ for the next state $k'$. Update $\hat{V}(k, i)$ accordingly (here, we choose the stochastic approximation scheme)

$$
\hat{V}(k, i) \leftarrow \hat{V}(k, i) + \alpha(t) \left[ \hat{V}(k') - \hat{V}(k, i) \right]
$$

(2.21)

- Update the probability distribution for state $k$ as:

$$
\hat{\pi}_k(i) \leftarrow \frac{\exp \left[ -\hat{\theta}_k \left( c(k, i) + \hat{V}(k, i) \right) \right]}{\sum_{j \in U(k)} \exp \left[ -\hat{\theta}_k \left( c(k, j) + \hat{V}(k, j) \right) \right]}
$$

(2.22)

where $\hat{\theta}_k$ is set in order to respect the prescribed degree of entropy (see Equation (2.8)).

- Update the expected cost of state $k$:

$$
\begin{cases}
\hat{V}_\pi(k) = \sum_{i \in U(k)} \pi_k(i) [c(k, i) + \hat{V}_\pi(k, i)], \\
\hat{V}_\pi(d) = 0, \text{ where } d \text{ is the destination state}
\end{cases}
$$

(2.23)

2.5.3 Links with Q-learning and SARSA

We now restate the optimality relations in terms of the $Q$-values. In the stochastic case, the $Q$-value represents the expected cost from state $k$ when choosing action $i$, $Q(k, i) = c(k, i) + \hat{V}(k, i)$. As before, the relationship between $Q$ and $V$ is
2.5. Optimal policy under exploration constraints for stochastic shortest-path problems

\( V(k) = \sum_{i \in U(k)} \pi_k(i) Q(k, i) \) and the optimality conditions are

\[
\begin{align*}
Q^*(k, i) &= c(k, i) + \sum_{k'} p_{kk'}(i) \sum_{j \in U(k')} \pi_{k'}(j) Q^*(k', j), \text{ with } k \neq d \\
Q^*(d, i) &= 0, \text{ for the destination state } d
\end{align*}
\]

(2.24)

The \( \pi_k^*(i) \) are provided by

\[
\pi_k^*(i) = \frac{\exp[-\theta_k Q^*(k, i)]}{\sum_{j \in U(k)} \exp[-\theta_k Q^*(k, j)]}
\]

(2.25)

which, as before, corresponds to a Boltzmann strategy involving the \( Q \)-value.

Now, if the entropy is close to zero (no exploration), the updating rule reduces to

\[
\begin{align*}
\hat{Q}(k, i) &\leftarrow c(k, i) + \sum_{k'} p_{kk'}(i) \min_{j \in U(k')} \left[ \hat{Q}(k', j) \right], \text{ with } k \neq d \\
\hat{Q}(d, i) &= 0, \text{ for the destination state }
\end{align*}
\]

(2.26)

which can be approximated by the following stochastic approximation scheme

\[
\hat{Q}(k, i) \leftarrow \hat{Q}(k, i) + \alpha(t) \left[ c(k, i) + \min_{j \in U(k')} \left[ \hat{Q}(k', j) \right] - \hat{Q}(k, i) \right], \text{ with } k \neq d
\]

(2.27)

and this corresponds to the standard Q-learning algorithm [171, 186, 187]. In this equation, \( k' \) is the next observed state.

The SARSA algorithm [147, 166, 171] considers both exploration and an unknown environment. In substance, SARSA approximates the \( Q \)-learning values by averaging out empirically both the uncertainty about the action to choose and the uncertainty about the next state in (2.24) by using stochastic approximation. Thus, from Equation (2.24), the \( Q \)-values are updated through

\[
\hat{Q}(k, i) \leftarrow \hat{Q}(k, i) + \alpha(t) \left[ c(k, i) + \min_{j \in U(k')} \left[ \hat{Q}(k', j) \right] - \hat{Q}(k, i) \right], \text{ with } k \neq d
\]

(2.28)

when choosing action \( i \) with probability

\[
\hat{\pi}_k(i) = \frac{\exp[-\theta_k \hat{Q}(k, i)]}{\sum_{j \in U(k)} \exp[-\theta_k \hat{Q}(k, j)]}
\]

(2.29)
2.6. Optimal policy under exploration constraints for discounted problems

However, since we are controlling the probability distributions for choosing an action (2.25), it is more sensible to rely on an updating rule like (2.21) instead of (2.28), leading to

\[
\hat{Q}(k, i) \leftarrow \hat{Q}(k, i) + \alpha(t) \left( c(k, i) + \sum_{j \in U(k')} \hat{\pi}_{k'}(j) \hat{Q}(k', j) \right) - \hat{Q}(k, i)
\]

(2.30)

while the \( \hat{\pi}_k(i) \) are updated in the usual way

\[
\hat{\pi}_k(i) \leftarrow \frac{\exp \left[ -\theta_k \hat{Q}(k, i) \right]}{\sum_{j \in U(k)} \exp \left[ -\theta_k \hat{Q}(k, j) \right]}.
\]

(2.31)

which means that only the uncertainty about the next state is estimated online.

2.6 Optimal policy under exploration constraints for discounted problems

In this section, instead of Equation (2.1), we consider discounted problems, for which there is a discount factor \( 0 < \gamma < 1 \),

\[
V_\pi(k_0) = E_\pi \left[ \sum_{t=0}^{\infty} \gamma^t c(s_t, u_t) | s_0 = k_0 \right]
\]

(2.32)

The meaning of \( \gamma \) is that future costs matter less than costs incurred at the present time. In this situation, as will be seen, there is no need to assume the existence of a destination state \( d \).

The problem can be converted into a stochastic shortest-path problem for which the analysis of the previous sections holds (see [21]). Let there be an extra termination (absorbing) state \( t \) in addition to the original states \( s = 1, \ldots, n \). State transitions and costs are obtained as follows: from state \( k \), when control action \( i \in U(k) \) is chosen with probability \( \pi_k(i) \), a cost \( c(k, i) \) is incurred and the next state is \( k'_i = f_k(i) \neq t \) with a fixed probability \( \gamma \), or \( t \) with probability \( (1 - \gamma) \). Once the agent has reached state \( t \), it remains there forever with no additional cost. Here, absorbing state \( t \) plays the same role as destination state \( d \) in our previous analysis, and the deterministic shortest-path model can easily be
2.7. Experiments

adapted to this framework, leading to the following optimality condition

$$V^*(k) = \sum_{i \in U(k)} \pi^*_k(i) [c(k, i) + \gamma V^*(k'_i)], \text{ with } k'_i = f_k(i), \ 0 < \gamma < 1 \ (2.33)$$

where, as before, $\pi_k(i)$ is given by

$$\pi^*_k(i) = \frac{\exp \left[ -\theta_k \left( c(k, i) + \gamma V^*(k'_i) \right) \right]}{\sum_{j \in U(k)} \exp \left[ -\theta_k \left( c(k, j) + \gamma V^*(k'_j) \right) \right]} \ (2.34)$$

which are the corresponding optimality conditions for deterministic discounted problems.

On the other hand, the optimality conditions for stochastic discounted problems are

$$V^*(k) = \sum_{i \in U(k)} \pi^*_k(i) [c(k, i) + \gamma V^*(k, i)] \ (2.35)$$

$$\pi^*_k(i) = \frac{\exp \left[ -\theta_k \left( c(k, i) + \gamma V^*(k, i) \right) \right]}{\sum_{j \in U(k)} \exp \left[ -\theta_k \left( c(k, j) + \gamma V^*(k, j) \right) \right]} \ (2.36)$$

with $V^*(k, i) = \sum_{k'} p_{kk'}(i) V^*(k')$.

2.7 Experiments

This section reports on experiments performed to study the behavior of the proposed algorithm.

2.7.1 Preliminary experiments

Preliminary experiments were performed on the graph shown in Figure 2.1, containing 14 numbered nodes connected by edges whose weights represent costs. The algorithm described in Section 2.3.1 was used in each experiment. It searches an optimal path from an initial node to a destination node, for a fixed exploration degree. We investigated how the algorithm reacts to changes of the exploration degree and the impact of these changes on the total expected cost.

Simple experiments were performed for evaluating the algorithm in three different settings: (i) a static environment (i.e., where the weight of the edges
2.7. Experiments

Figure 2.1: The graph used in our preliminary experiments. Node 1 is the initial state and node 13 is the destination state. Initial costs are indicated on the edges.

does not change over time), (ii) a dynamic (nonstationary) environment, and (iii) a simple load-balancing experiment.

In each experiment, the expected costs and transition probabilities are updated at each time step according to Equations (2.11) and (2.12). In the sequel, one run represents the complete routing of an agent from the source (initial node) to the destination. At the beginning of each simulation, \( \hat{V}(k) \) and \( \hat{\pi}_k(i) \) are given initial values according to the procedure described at the end of Section 2.3.1. Exploration rates are set to a common value for all states.

2.7.1.1 Static environment

This experiment sends 15,000 agents (15,000 runs) through the network shown in Figure 2.1. The cost of the external edges (i.e., the edges on paths \([1,2,6,12,13]\) and \([1,5,9,10,14,13]\)) is set to 1, while the cost of all the other edges (internal edges) is set to 2.

The goal of the simulation is to observe the paths followed by the routed agents (i.e., the traffic) for various values of the entropy. The experiment was performed with exploration rates of 0%, 30%, 60%, and 90%.

Figure 2.2 shows the behavior of the algorithm on the network of Figure 2.1 for the four exploration rates, with the following drawing convention: the more
2.7. Experiments

an edge is crossed in the routing, the more important are its grey level and width.

<table>
<thead>
<tr>
<th>0% exploration rate</th>
<th>30% exploration rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="0% exploration rate" /></td>
<td><img src="image2" alt="30% exploration rate" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>60% exploration rate</th>
<th>90% exploration rate</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image3" alt="60% exploration rate" /></td>
<td><img src="image4" alt="90% exploration rate" /></td>
</tr>
</tbody>
</table>

**Figure 2.2:** Traffic through the edges for four different exploration rates. We used the following drawing convention: the more an edge is crossed in the routing, the more important are its grey level and width.

For an exploration rate of 0%, the algorithm finds the shortest path from node 1 to node 13 (path [1,2,6,12,13]), and no exploration is carried out: [1,2,6,12,13] is the only path traversed. For an exploration rate of 30%, the shortest path [1,2,6,12,13] is still the most used, but other edges not on the shortest path are now also investigated. The second shortest path [1,5,9,10,14,13] now also routes a significant traffic. Exploitation thus remains a priority over exploration. For an exploration rate of 60%, the algorithm uses more edges that are not on the shortest path, but still prefers the shortest path. Finally, for an exploration rate of 90%, the exploitation of the shortest path is no longer the priority: exploration is now clearly favored over exploitation.
2.7. Experiments

2.7.1.2 Static environment: Convergence to the same minimum

This experiment investigates the convergence of our algorithm for estimating the transition probabilities. We will experimentally test the algorithm with different initial conditions (i.e., one for each simulation) on the graph of Figure 2.1 to study the convergence towards the same solution (the algorithm only finds a local optimum). These initial conditions involve the initialization of the two parameters of our model within a range of numbers generated stochastically. Taking these initial conditions as parameters, a simulation is the execution of the algorithm until convergence of the total expected cost \( \hat{V} \) to a local optimal solution. Thus, for each simulation (50 simulations in total), we initialise the total expected cost \( \hat{V} \) of each node and the parameter \( \theta \) (using to compute the initial transition probabilities matrix) with numbers generated uniformly in \([0, 5]\).

Figure 2.3 shows the estimated density of the average cost on the 50 simulations for three exploration rates, i.e., 20%, 50% and 70%. It can be observed that the different simulations converge to the same minimum, i.e., 4.4635, 5.9440 and 7.9094 respectively for an exploration rate of 20%, 50% and 70%. The standard deviations computed for the different exploration rates are 0.0006 (20%), 0.0011 (50%) and 0.0018 (70%), and these small values confirm the convergence of our simulations to the same minimum.

<table>
<thead>
<tr>
<th>Entropy</th>
<th>20%</th>
<th>50%</th>
<th>70%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average value</td>
<td>0.0743</td>
<td>0.2218</td>
<td>0.4471</td>
</tr>
</tbody>
</table>

Table 2.1: Average values (the numbers have to be multiplied by \(1.0e - 005\)) of the Frobenius norm computed on the transition probabilities matrix of the 50 simulations for three exploration rates, i.e., 20%, 50% and 70%.

Table 2.1 presents the average Frobenius norm computed on the transition probabilities matrix for three exploration rates, i.e., 20%, 50% and 70%. We can see that these values are very small showing a large similarity between the computed transition probabilities of each simulation. In other words, the transition probabilities of each simulation converge to the same values; that is, the policy is similar.

2.7.1.3 Static environment: Convergence time and line search impact

This section analyzes experimentally on 30 simulations the convergence time of our algorithm with respect to the network size (number of states and edges)
2.7. Experiments

Figure 2.3: Estimated density of the average costs (on 50 simulations) needed to reach the destination node (13) from the initial node (1) of Figure 2.1 for three exploration rates: 20%, 50% and 70%.

and the level of entropy specified, in a static environment. As explained in the previous section, a simulation executes the algorithm until convergence of the total expected cost $\hat{V}$. Here, we will study the impact of the network size and the exploration rate on the convergence rate. The impact of a line search on the convergence time is also studied in this experiment.

Concerning the experimental settings, we run the algorithm on four simple grids of different size (i.e., 5x5, 10x10, 20x20, 30x30 and 50x50 states) and for three different levels of entropy (i.e., 25%, 50% and 75%). Thus, for both exploration strategies (i.e., Boltzmann with and without line search), we obtain a simulation for each couple of grid size and exploration rate (i.e., 5 grid sizes multiplied by 3 exploration rates, with 15 simulations for each exploration strategies,
2.7. Experiments

Figure 2.4: A simple grid of 5x5 states.

thus 30 simulations in total).

Figure 2.4 represents a simple grid of 5x5 states. The agent is initially located in cell (1,1) (i.e., top left) of the grid and has to reach cell in the bottom right representing the goal state. The agent has the choice among eight possible actions in each state: move one cell forward in one of eight directions (i.e., north, south, west, east, north-east, north-west, south-east, and south-west). Each action has a cost of 1 and the experiment is carried out until convergence.

Table 2.2 reports, for each setting, the convergence time of the algorithm in seconds. Figure 2.5 shows the logarithm of convergence time evolution for an exploration rate of 50% and various grid sizes. As expected, the convergence time increases with the grid size for both exploration strategies. The convergence time seems to be polynomial in terms of the number of nodes. We can also observe that the Boltzmann strategy with line search is computationally more expensive than without line search. This difference is significant when the exploration rate is close to 100%, corresponding to a small \( \frac{1}{\text{temperature}} \) close to 0 (i.e., \( \theta \to 0 \)).

2.7.1.4 Dynamic environment

The second experiment setting introduces changes in the environment by abruptly varying the cost of some edges.
2.7. Experiments

Table 2.2: Comparison of two exploration strategies in term of convergence time (in seconds), Boltzmann with line search and Boltzmann without line search for various exploration rates and grid sizes.

The basic setting is the same as in the first experiment, namely, convey 15,000 agents (15,000 runs) from initial node 1 to destination node 13. The cost of the external edges (i.e., the edges belonging to paths \([1,2,6,12,13]\) and \([1,5,9,10,14,13]\)) is initialized to 3, while the cost of other edges (internal edges) is initialized to 6. In this configuration, the shortest path from node 1 to node 13 is the path \([1,2,6,12,13]\) with a total cost of 12.

After having sent 7,500 agents (i.e., in the middle of the simulation, at run number 7,501), the cost of all internal edges is set to 1, all other things remaining equal. This change creates new shortest paths, all of them with a total cost of 4, passing through internal nodes \([3,4,7,8,11]\).

Figure 2.6 shows the total cost of transferring agents from source to destination, in terms of numbers of runs, for five exploration rate: 0\%, 10\%, 20\%, 30\% and 50\%.

The bottom graph shows the total cost for an exploration rate of 0\%. Despite the change, the cost remains equal to 12 throughout the simulation. Indeed, once
2.7. Experiments

Figure 2.5: Logarithm of convergence time (in seconds) of the Boltzmann strategy with and without line search for a constant exploration rate of 50% and various grid sizes.

The algorithm has discovered the shortest path, it does not explore any more. Consequently, the system misses a path that has become (at run number 7,501) much cheaper than the previously optimal path.

The four other graphs show the results for exploration rates of 10%, 20%, 30%, and 50%. Each of these settings is able to find the new optimal path after the change of environment. The total cost is updated from about 12 to 4. The higher the exploration rate, the faster the new option is discovered. Note also the slight rise in average total cost as the value of the exploration rate increases.

Figure 2.7 shows the evolution of the average cost, the maximal cost, and the minimal cost (computed on the first 7,500 runs) in terms of the exploration rate. Increasing the entropy induces a growth in the average total cost, which is natural since the rise in entropy causes an increase in exploration, therefore producing a reduction of exploitation. We also notice that the difference between the minimum and the maximum total costs grows with the increase in entropy.
## 2.7. Experiments

<table>
<thead>
<tr>
<th>Run number/100</th>
<th>0% exploration rate</th>
<th>10% exploration rate</th>
<th>20% exploration rate</th>
<th>30% exploration rate</th>
<th>50% exploration rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>20</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>30</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>40</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>50</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>60</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>70</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>80</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>90</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
<tr>
<td>100</td>
<td>5.0</td>
<td>10.0</td>
<td>15.0</td>
<td>5.0</td>
<td>10.0</td>
</tr>
</tbody>
</table>

**Figure 2.6:** Average cost needed to reach destination node (13) from the initial node (1) for various exploration rates, with a change in the environment at run number 7,501.

but remains reasonably small.

### 2.7.1.5 Load-balancing experiment

This experiment simulates a simple load-balancing problem. Load balancing is used to spread work between processes, computers, disks or other resources. The specification of the load-balancing problem for this experiment is as follows. We use the graph defined previously, with the same cost on the edges (see Figure 2.1). Two destination nodes are now present (node 11 and 14), but the approach easily generalizes to more than two destination nodes. Each destination node is a special node able to provide some resource (such as food) desired by the agents crossing the network. We focus on the average minimal cost of the paths followed by agents interested in consuming the resource.

A component of that cost takes into consideration the amount of resource remaining at each destination node. Indeed, the amount of resource at each destination node being limited, we introduce a “warning threshold” indicating
2.7. Experiments

![Cost vs Exploration Rate Graph](image)

**Figure 2.7:** Average, maximal and minimal costs needed to reach the destination node (13) from the initial node (1) in terms of exploration rate.

When the resource is becoming scarce, translated into a decrease in reward. For simplicity, this critical threshold is the same for each destination node and is set to 10% of the starting amount. When an agent reaches a destination node, the amount of resource at that node is decreased by one unit. If the remaining amount is higher or equal to the warning threshold, the agent receives a positive reward (a negative cost). On the other hand, if the amount of resource is lower than the warning threshold, the agent receives a lower reward, proportional to the remaining amount: the reward is then weighted by the ratio of the remaining amount to the threshold value. The introduction of this threshold results in a gradual decrease of the reward of a destination node.

Values of the simulation parameters were defined in the following way. Destinations nodes chosen for the experiment are nodes 11 and 14 with an initial amount of resource of 5,000 and a threshold of 500 (10% of 5,000). 10,000 agents are sent from node 1 (10,000 runs) with the objective of gathering some amount of resource with minimal average cost.

For this experiment, we use four exploration rates, 0%, 40%, 60 and 80%; Figure 2.8 shows the resulting graphs for each rate. Each graph reports the quan-
2.7. Experiments

tities of the resource remaining at nodes 11 and 14 in terms of run number. The 
resource at node 14 is consumed first because its path from node 1 is shorter.

For an exploration rate of 0%, no exploration takes place and the algorithm 
proposes only destination node 14 (i.e., the shortest path). Node 11 starts being 
exploited only when the amount remaining at node 14 has reached the threshold 
and when the cost of exploiting node 11 becomes less than that of exploiting 
node 14 (see Figure 2.8 at about run 4700). Node 14 stops being exploited until 
the remaining amount of resource at node 11 has reached the threshold and the 
cost of exploiting node 14 has become again less than that of exploiting node 11 
(see Figure 2.8 at about run 9300). Notice that in that case of a zero-entropy, the 
two destination nodes are never exploited simultaneously, so that no real “load 
balancing” occurs.

The other graphs show the results of simulation with exploration rates of 40, 
60, and 80%. For non-null exploration rate, both destination nodes are exploited 
from the beginning. The higher the exploration rate, the more important is the 
simultaneous exploitation of the destination nodes. Thus, as expected, the degree 
of load balancing increases with the exploration rate.

2.7.2 Some experimental comparisons

This experiment focuses on comparisons between the optimal method of explo-
ration, which will be referred to as Boltzmann technique (the Boltzmann explo-
ration strategy based on the $Q$-value of Equations (2.11) and (2.12)), and the 
$\epsilon$-greedy strategy. The popular $\epsilon$-greedy exploration strategy [171] selects the 
currently best action (the best according to the current estimate of the total ex-
pected cost $\hat{V}_{\pi}$), in any given state, with probability $(1 - \epsilon)$, and selects another 
random action with a uniform probability $\epsilon / (m - 1)$ where $m$ is the number of 
available actions. A constant exploration rate ($\epsilon$ in the $\epsilon$-greedy exploration) 
is often used in practice. This exploration strategy appears, for instance, in a 
special class of Markov decision process, introduced in [181].

In order to compare the results obtained by the two different methods, the 
parameter $\epsilon$ ($\epsilon$-greedy) was tuned to achieve the same exploration rate as for the 
Boltzmann method.

To carry out these comparisons, we simulate the problem of load balancing 
described previously. We use the graph of Figure 2.1 with the same weights 
on the edges, node 1 being the source node. We assign the same amount of 
resource to each destination node and we introduce a threshold which influences 
the costs in the same way as before (see Section 2.7.1.5). However, compared
2.7. Experiments

Figure 2.8: Load-balancing experiment: Evolution of the remaining amount of resource in terms of the run number, for four different exploration rates.

...to the previous simulation, the destination nodes change over time. Specifically, node 15 is chosen as a permanent destination node and another destination node is chosen among nodes 7, 8, and 11, with a probability of 1/3 each. Changes of destination nodes are performed at regular intervals (e.g., every 500 runs). Nodes 7, 8, and 11 are chosen as potential destination nodes because they do not belong to the shortest path from source node 1 to destination node 15.

As before, agents are sent through the network from the source node until they reach a destination node, and this operation is referred to as a run. We execute four simulations for this experiment. Each corresponds to a period for the environment changes, called an epoch, respectively of 500, 250, 100 and 50 runs. Destination nodes change at the beginning of an epoch they do not change within an epoch. Thus, a short epoch corresponds to a fast-changing environment while a large epoch corresponds to a slowly-changing environment. Each simulation is carried out by sending 5,000 agents through the network (5,000 runs) and the corresponding average cost per run (averaged on 5,000 runs) is recorded.

For illustration, Figure 2.9 presents the results (the cost averaged on 100...
2.7. Experiments

runs) for the simulation with an epoch of 500 runs. The Boltzmann method outperforms the $\epsilon$-greedy method. The oscillations reflect the succession of epochs and the progressive reduction of the amount of resource available at the destination nodes.

![Exploration rate of 5%](image)

**Figure 2.9:** Results of one simulation when the environment changes every 500 runs (one epoch = 500 runs), for an exploration rate of 5%. The total duration of the simulation is 5,000 runs. The average cost is displayed in terms of percentage of simulation progression.

Table 2.3 represents the results of the simulations for each method, five exploration rates (ranging from 1% to 30%), and four epochs. We report the total cost averaged on 5,000 runs. The 95% confidence interval computed on the 5,000 runs is also displayed. Again, the Boltzmann method clearly outperforms the $\epsilon$-greedy method.

Finally, Table 2.4 reports the average computing time (in seconds) for 10 experiments of 1,000 runs for (i) the standard $\epsilon$-greedy strategy without line search ($\epsilon = 0.5$), (ii) the Boltzmann strategy without line search (fixing $\theta = 0.5$), and (iii) the Boltzmann strategy with a line search ($E' = 10\%$). The line search algorithm that was implemented is a simple bisection method; no attempt has been made in order to optimize it. We clearly observe that fixing the entropy and performing a line search significantly increases the computing time (a factor of about $7 \times$ is typically observed for the Boltzmann strategy with a line search versus the standard $\epsilon$-greedy strategy).
2.7. Experiments

<table>
<thead>
<tr>
<th>Environment change every 500 runs (epoch=500)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploration rate</td>
</tr>
<tr>
<td>Boltzmann</td>
</tr>
<tr>
<td>(\epsilon)-greedy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment change every 250 runs (epoch=250)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploration rate</td>
</tr>
<tr>
<td>Boltzmann</td>
</tr>
<tr>
<td>(\epsilon)-greedy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment change every 100 runs (epoch=100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploration rate</td>
</tr>
<tr>
<td>Boltzmann</td>
</tr>
<tr>
<td>(\epsilon)-greedy</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Environment change every 50 runs (epoch=50)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exploration rate</td>
</tr>
<tr>
<td>Boltzmann</td>
</tr>
<tr>
<td>(\epsilon)-greedy</td>
</tr>
</tbody>
</table>

Table 2.3: Comparison of two exploration strategies in term of average cost, Boltzmann and \(\epsilon\)-greedy for various exploration rates and epochs, corresponding to various environment change rates.

2.7.3 The puzzle problem

This section reports on a more realistic experiment on a 21 x 21 puzzle problem inspired from [68] and compares the results obtained by the Boltzmann and the \(\epsilon\)-greedy strategies.

In Figure 2.10, black cells are inaccessible; blocks represent walls; the agent is initially located in cell (1,1) (i.e., top left) of the grid and has to reach cell (21,21) (i.e., bottom right) representing the goal state. The agent has the choice among eight possible actions in each state: move one cell forward in one of eight directions (i.e., north, south, west, east, north-east, north-west, south-east, and south-west). When encountering a wall (an inaccessible cell), the agent has no possibility of moving in the intended direction and it stays in place. At the beginning of the simulation, each action has a cost of 10 if no wall is encountered while actions hitting a wall have a cost of 110. The simulation is carried out by sending 1,000 agents through the grid (i.e., 1,000 runs), and the corresponding average cost per run (averaged on 1,000 runs) is recorded. At the middle of the simulation (i.e., after 500 runs), the environment is changed by dividing by 10
2.7. Experiments

<table>
<thead>
<tr>
<th>Exploration Strategy</th>
<th>Computing Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$-greedy (no line search)</td>
<td>2.6</td>
</tr>
<tr>
<td>Boltzmann (no line search)</td>
<td>2.1</td>
</tr>
<tr>
<td>Boltzmann (with line search)</td>
<td>18.6</td>
</tr>
</tbody>
</table>

Table 2.4: Comparison of the computing time (averaged on 10 repetitions) needed for performing 1,000 runs, for three exploration strategies, $\epsilon$-greedy without line search and Boltzmann with and without line search.

The cost of each action available from the cells located above the main obstacle, i.e. the cells in the x-range [5,21] and the y-range [1,4]. In other words, after the environment change, each action from these cells has a cost of 1 if no wall is encountered; otherwise it has a cost of 11. This change creates new shortest paths situated above the main obstacle.

Figure 2.10 shows the paths used by the agents with the Boltzmann strategy and two different exploration rates: 0% and 60%. The left side of Figure 2.10 presents the results before the environment change, and the right side shows results after the change. The intensity of colors of the figures correspond to the average number of passages through the cell. When the exploration rate is at 0% (no exploration), we observe that the agents follow the same path in the left and right side: they did not notice the environment change, and therefore did not discover the new shortest path. On the other hand, when the exploration rate is 60%, the agents exploit their knowledge and are able to find the shortest path while continuing to explore the environment.

The graph in Figure 2.11 presents the evolution of the average cost of each agent in terms of percentage of simulation progress (1,000 runs are performed) with an exploration rate of 60%, for the Boltzmann and $\epsilon$-greedy strategies. Both strategies improve their average cost but Boltzmann still obtains better results.
Figure 2.10: The puzzle problem: 1,000 agents (1,000 runs) are sent through the grid and the Boltzmann strategy is used with two different exploration rates, 0% (top) and 60% (bottom). The left side of the figure represents the paths used during the first 500 runs while the right side shows the paths used after the change of the environment (run > 500). The walls are displayed in black blocks while, in each cell, the intensity of the colors is proportional to the average passage through it.
Figure 2.11: Comparison of \( \epsilon \)-greedy and Boltzmann strategies for the puzzle problem when the environment changes at the middle of the simulation, for an exploration rate of 60%. The total duration of the simulation is 1,000 runs. The average cost is displayed in terms of percentage of simulation progress.
Chapter 3

Reinforcement Learning: Two Additional Extensions

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 Randomized Shortest-Path Problem: Global Entropy</td>
<td>61</td>
</tr>
<tr>
<td>3.1.1 Introduction</td>
<td>61</td>
</tr>
<tr>
<td>3.1.2 Contributions and Application Areas</td>
<td>63</td>
</tr>
<tr>
<td>3.1.3 Statement of the problem and notations</td>
<td>65</td>
</tr>
<tr>
<td>3.1.4 Optimal routing policy under entropy constraint</td>
<td>69</td>
</tr>
<tr>
<td>3.1.5 Computation of an optimal policy</td>
<td>74</td>
</tr>
<tr>
<td>3.1.6 Experiments</td>
<td>75</td>
</tr>
<tr>
<td>3.2 Hierarchical reinforcement learning</td>
<td>81</td>
</tr>
<tr>
<td>3.2.1 Related Work</td>
<td>81</td>
</tr>
<tr>
<td>3.2.2 Macro-actions and primitive actions</td>
<td>84</td>
</tr>
<tr>
<td>3.2.3 Concurrent actions</td>
<td>86</td>
</tr>
<tr>
<td>3.2.4 AND/OR Graph</td>
<td>88</td>
</tr>
<tr>
<td>3.2.5 Macro-actions and Concurrent actions in a common hierarchical framework</td>
<td>90</td>
</tr>
</tbody>
</table>

Motivations From the basic reinforcement learning framework introduced in Chapter 2, we can extend this framework from two points of view: the exploration strategy and the learning structure. This Chapter presents these extensions in two models and described them independently.
The first extension, as in previous chapter, addresses the problem of designing the transition probabilities of a finite Markov chain (the policy) in order to minimize the expected cost for reaching a destination node from a source node while maintaining a fixed level of entropy spread in the network (the exploration). This time, however, the entropy is fixed at the global level. It is motivated by the following scenario. Suppose you have to route agents through a network in some optimal way; for instance by minimizing the total travel cost. Nothing particular up to now – you could use a standard shortest-path algorithm. Suppose, however, that you want to avoid pure deterministic routing policies in order, for instance, to allow some continual exploration of the network, to avoid congestion, or to avoid complete predictability of your routing strategy. In other words, you want to introduce some randomness/unpredictability in the routing policy, i.e., the routing policy is randomized. This problem, which will be called the randomized shortest-path problem (RSP), is investigated in this extension. The global level of randomness of the routing policy will be quantified by the expected Shannon entropy spread into the network, and is provided a priori by the designer. Then, necessary conditions allowing to compute the optimal randomized policy – minimizing the expected routing cost – are derived. Iterating these necessary conditions, reminiscent of Bellman’s value iteration equations, allows to compute an optimal policy, that is, the set of transition probabilities in each node, but no convergence proof is provided. It is shown that an optimal policy can be obtained by solving a simple linear system of equations. Finally, simulation results obtained on simple, illustrative, examples show that the model behaves as expected. Part of this work is a joint work together with Fouss and Saerens, and has been published in [152].

The second extension presented in this chapter introduces a hierarchy in the learning process. Indeed, another issue central to reinforcement learning is how to deal with large state spaces and complex actions [90]. A strategy is to treat them as a hierarchy of learning problems including special actions [90]. Hierarchical decision-making models has many benefit: (a) by reusing subtasks learned on previous problems, we can learn faster new problems; (b) because fewer parameters must be learned and because subtasks can ignore irrelevant features of the full task, we can learn from fewer trials; and (c) we improve exploration because exploration can take “big step” at high levels of abstraction [46, 61, 60, 109]. Often, hierarchical strategies introduce insignificant sub-optimality in performance, but potentially gain a good deal of efficiency in execution time, learning time, and space [90].

Concerning the structure of the chapter, it is organized as follows. Section
3.1. Randomized Shortest-Path Problem: Global Entropy

3.1 presents the first extension RSP with a global entropy. Section 3.2 describes the second extension based on a hierarchical reinforcement learning framework.

3.1 Randomized Shortest-Path Problem: Global Entropy

3.1.1 Introduction

As introduced in Chapter 2, algorithms for finding the shortest path in a network are currently used in a wide range of application areas including transportation networks, medical imaging, wide-area network routing, artificial intelligence, to name a few. Many extensions of the basic shortest-path algorithms have been proposed, still extending their application range (see for instance [19, 21, 35, 39, 85]). In particular, many important applications developed in artificial intelligence, machine learning, pattern recognition, bioinformatics and data mining, such as speech recognition (dynamic time warping or the Viterbi algorithm ([79, 141]), sequence alignment ([49, 69]), Markov decision processes ([15, 139]), routing in data networks ([22]), social network analysis (betweenness centrality; see [185]), game playing (minimax algorithms; see, e.g., [7]), planning ([59, 103]), neurocomputing ([23]), rely on one way or another on variants of shortest-path algorithms.

This section aims to introduce a related problem which will be called the randomized shortest-path problem (RSP), in the framework of a single source and a single destination. It can be described informally as follows. Suppose we have to find the path of minimum length from a source node to a destination node in a network, where the length of a path is just the sum of the costs of the arcs on the path. Usually, shortest-path algorithms provide pure deterministic routing policies: when standing in a given node \( k \), we just follow the arc adjacent to \( k \) on the shortest path. In the present section, we investigate yet another global possibility of randomizing the routing policy: the agents could follow different paths, according to some probability distribution. As remind, there are some circumstances where randomization could eventually prove useful. Let’s briefly remember some of these circumstances:

- If the environment is changing over time (non-stationary), the system could benefit from randomization or continual exploration. Indeed, without exploration, the agents are routed exclusively along the best path – without exploring alternative paths. They would therefore not be aware of the changes occurring in the network, for instance some alternative path be-
3.1. Randomized Shortest-Path Problem: Global Entropy

coming shorter. For the model introduced in the first extension, the structure of the network is supposed to be known while the costs may change over time.

- Introducing randomness could be beneficial per se. Consider for instance the situation where an agent has to reach a given goal without being intercepted by some opponents. A deterministic shortest-path policy would make its behaviour totally predictable; on the contrary, randomness introduces unpredictability and therefore renders interception more difficult. Randomization has proven useful for exactly this reason in game theory.

- In the case where there are multiple destination nodes (or goals), introducing randomness allows to perform some load balancing, by exploiting the goal nodes in parallel.

- Randomization also allows to spread the traffic on multiple paths, therefore reducing the danger of congestion. Indeed, by following a randomized strategy, the goods are routed along multiple different paths and are therefore spread over the network.

- One may want to use a dissimilarity measure between two nodes that does not only account for the shortest path, but also for all the other paths, with longer paths being penalized with respect to short ones; therefore considering that nodes connected by many, short, paths are closer than nodes connected by, for instance, only one short path (as in [57]).

- In some applications areas, such as sequence alignment, computing a similarity measure accounting for all paths could eventually provide better results than relying on the best path.

For all these reasons, we decided to investigate randomized shortest-path problems. We thus define a randomized shortest-path problem as a shortest-path problem to which exploration constraints (for instance entropy constraints) are added in order to obtain optimal randomized policies (also called stochastic policies), and therefore continual exploration.

Mathematically speaking, randomization corresponds to the association of a probability distribution on the set of admissible arcs to follow in each node (choice randomization). If no randomization is present, the agents are routed on the shortest path (the optimal choice) with probability one – only the best policy is exploited. Randomization appears when this probability distribution is
3.1. Randomized Shortest-Path Problem: Global Entropy

no more peaked on the best choice: the agent is willing to sacrifice efficiency for
exploration.

In this framework, we propose to measure the randomness associated to a
given node by the (Shannon) entropy (see for instance [40, 93, 161]) of the prob-
ability distribution on the set of admissible arcs to follow (transition probabili-
ties) in this node. This entropy value captures the degree of randomness linked
to the node. When the entropy is zero, there is no uncertainty, while when the
entropy is maximal, a blind choice, with equal probability of following any arc,
is performed.

Then, in our model, we restate the randomized shortest-path problem as a
global optimization problem: define the best randomized policy (the set of transi-
tion probabilities in each node) that minimizes the expected cumulated cost from
the source node to the destination node while maintaining a fixed degree of ran-
doness. This problem leads to a set of nonlinear equations defining necessary
conditions of optimality. These equations, reminiscent of Bellman’s equations,
can be solved by iterating them until convergence. They provide the policy (the
transition probabilities) that minimizes the expected cost from the initial node to
the destination node, for a given degree of randomness. Interestingly enough,
when the global degree of randomness is zero, the nonlinear equations reduce
to Bellman’s equations for finding the shortest path from the initial node to the
destination node.

What concerns the organization of this section, Section 3.1.2 presents the
contribution and possible application areas of the model presented. Section 3.1.3
introduces the notations, the randomized shortest-path problem and the way we
manage randomness. In Section 3.1.4, the necessary conditions of optimality
are derived. Section 3.1.5 describes an iterative procedure for computing the
randomized policy. Finally, Section 3.1.6 shows some simulation examples.

3.1.2 Contributions and Application Areas

The idea of quantifying the uncertainty linked to each node by entropy was intro-
duced in Chapter 2 in the context of reinforcement learning. The main difference
between this previous work is the fact that, in this work, we fix the global en-
tropy spread in the network, instead of fixing the entropy defined at each node
in Chapter 2. While this difference seems a priori insignificant, it appears that
constraining the global entropy spread into the network is more natural and more
involved. Clearly, the nodes that need a large spread are difficult to determine
in advance, and the model has to distribute the entropy by optimizing it globally
over the network. More precisely, in this work, the \textit{global degree of randomness} associated to the whole network is quantified by a weighted sum of the individual entropies associated to each node.

Let us mention some interesting papers that are related to the present work. [173] introduced a one-parameter family of algorithms that, as our algorithm, recover both the Bellman-Ford procedure for finding shortest paths as well as the iterative algorithm for computing the average first-passage time. However, it was based on heuristic grounds and not on a well-defined cost function to optimize. Moreover, it does not provide a randomized policy. On the other hand, [181] studies a family of Markov decision problems that are linearly solvable, that is, for which a solution can be computed by solving a matrix eigenvector problem. In order to make this possible, Todorov assumes a special form for the control of the transition probabilities, which transforms the problem of finding the policy into an eigenvector problem. In [27] design a Markov chain that has fastest mixing properties and in [169] they discuss its continuous-time counterpart. In a completely different framework, uninformed random walks, based on maximizing the long-term entropy ([44, 182]), have recently been proposed as an alternative of the standard PageRank algorithm.

Since this work is somewhat theoretical, we also mention a few potential applications of randomized shortest paths (RSP) in artificial intelligence, machine learning, pattern recognition, bioinformatics or data mining:

- **Routing and planning when the environment is changing.** Reinforcement learning and Markov decision algorithms based on the RSP could be designed and studied.

- **Using mixed, randomized, strategies based on the RSP instead of the usual minimax in game playing.** In a two-persons game, it is unrealistic to assume that the opponent is completely rational, as minimax does. Therefore, it could be interesting to model the opponent’s behavior by a RSP strategy instead of a pure minimax one, which leads to mixed minimax strategies.

- **Computing dissimilarities between nodes of a weighted, directed, graph.** The expected cost for reaching one node from another node defines a dissimilarity measure between nodes of the graph, ranging from the shortest-path to the average first-passage time distance, depending on the amount of entropy spread in the network. This fact could be exploited in data-mining applications such as recommender systems (see for instance [153, 57]). In-
3.1. Randomized Shortest-Path Problem: Global Entropy

Indeed, random-walk or electrical based proximity measures between nodes of a graph are becoming popular alternatives to the standard shortest-path distance (see [57, 97, 125, 140, 153]).

- **Computing dissimilarities between strings or sequences.** Instead of using the Viterbi algorithm on a lattice for computing the dissimilarity between two sequences, one could use the RSP algorithm which will account for all the alignments between the two sequences instead of the single shortest one.

Of course, we do not know a priori if any of these application of the RSP would be beneficial; this should essentially be considered as propositions for further work. Moreover, we have to stress the fact that the present work is not focused on any specific application area; rather, its main purpose is to study RSP problems per se and to propose some general techniques to tackle them.

### 3.1.3 Statement of the problem and notations

For the sake of simplicity, this section will be focused on what is called “**deterministic shortest-path problems**” as defined, for instance by [21]. Consider a weighted directed graph or network, $G$, with a set of $n$ nodes $V$ (or vertices) and a set of arcs $E$ (or edges). To each arc linking node $k$ and node $k'$, we associate a number $c(k, k')$ representing the immediate cost of following this arc. This cost can be positive (penalty), negative (reward), or zero, provided that no cycle exists whose total cost is negative ([39]). In particular, this implies that if the graph is undirected, all costs are nonnegative.

The choice to follow an arc from node $k$ will be made according to a **probability distribution** (transition probabilities) defined on the set $S(k)$ of neighbouring nodes (successors $S$) that can be reached from node $k$. These transition probabilities, defined on each node $k$, will be denoted as $p(k'|k) = p_{kk'}$ with $k' \in S(k)$. Furthermore, $P$ will be the matrix containing the transition probabilities $p_{kk'}$ as elements. If there is no arc between $k$ and $k'$, we simply consider that $c(k, k')$ takes a large value, denoted by $\infty$; in this case, the corresponding transition probability is set to zero, $p_{kk'} = 0$. The main difference between randomized and standard deterministic shortest-paths problems resides in the fact that we will impose randomized choices. Randomization is introduced in order to guarantee a predefined degree of randomness that will be quantified by the

---

*Notice that in this work paths may contain cycles. Paths containing cycles are also commonly called walks in the literature.*
3.1. Randomized Shortest-Path Problem: Global Entropy

Shannon entropy of the probability distributions. Randomized choices are common in a variety of fields; for instance game theory (called mixed strategies in this context; see for instance [128]), computer sciences ([120]), Markov games ([105]) or decision sciences ([142]).

Moreover, as [21], we assume that there is a special cost-free destination or goal node \( d \); once the system has reached that node, it remains there at no further cost. Thus, node \( d \) has no outgoing link except eventually \( d \) itself. In order to simplify the notations, we further assume that we cannot return to the initial node \( k_0 \); that is, node \( k_0 \) has no incoming link (no predecessor). If this is not the case, just add a new initial node pointing to the previous one with zero cost. Finally, we will consider a problem structure such that termination is inevitable. Thus, the horizon is in effect finite, but its length is random and it depends on the policy being used. The conditions for which this is true are, basically, related to the fact that the destination node can be reached in a finite number of steps from any potential initial node; for a rigorous treatment, see ([21, 23]).

The overall goal will be to determine the set of probability distributions \( \pi \equiv \{p(k'|k); k, k' = 1, 2, \ldots, n\} \), contained in the transition-probabilities matrix \( P \), and called the choice probabilities or the policy, that minimizes

\[
V_\pi(k_0) = E_\pi \left\{ \sum_{t=0}^\infty c(s_t, s_{t+1}) \mid s_0 = k_0 \right\}
\]

where \( s_t \) is a random variable containing the label of the node reached at time step \( t \) and \( V_\pi(k_0) \) is the total expected cost accumulated over an infinite horizon, when starting from the initial (or source) node \( k_0 \), and following policy \( \pi \). The expectation is taken on the transition probabilities associated to the nodes.

Thus, the main objective is to design a randomized policy minimizing the expected cost-to-go (Equation (3.1)) subject to an entropy constraint controlling the total randomness spread in the network, and therefore the exploration rate. In other words, we are looking for an optimal policy, \( \pi^* = \arg\min_\pi [V_\pi(k_0)] \) or, equivalently, the optimal transition-probabilities matrix \( P^* \) subject to entropy constraints. Stated differently, the problem is to design an optimal finite Markov chain minimizing the expected cost needed to reach a destination state from an initial state, while fixing the entropy spread in the chain.

3.1.3.1 Total expected cost and Markov chains

Thus, the essence of the problem is to reach the destination node \( s = d \) with minimal expected cost. Once the set of transition probabilities, \( \pi \), is fixed, this
3.1. Randomized Shortest-Path Problem: Global Entropy

problem can be represented as a Markov chain where each node is a state. The destination state is then considered as absorbing with no outgoing link. In this framework, the problem of computing the expected cost (3.1) from any state \( k \) is closely related to the computation of the “average first-passage time” in the associated Markov chain ([96, 127]). The average first-passage time is the average number of steps a random walker starting from the initial state \( k_0 \) will take in order to reach destination state \( d \). By first-step analysis (see for instance [174]), one can easily show that, once the transition probabilities are fixed, the total expected cost \( V_\pi(k) \) can be computed through the following equations

\[
\begin{align*}
V_\pi(k) &= \sum_{k' \in S(k)} p_{kk'} [c(k, k') + V_\pi(k')] \quad \text{for } k \neq d \\
V_\pi(d) &= 0 \quad \text{for destination state } d
\end{align*}
\]

These equations can be iterated in order to find the expected costs; a closed-form solution will be derived later, and necessitates the introduction of the fundamental matrix of the Markov chain.

Let us now renumber the states in order to have state \( k_0 \) in the first position (index 1) and state \( d \) in the last position (index \( n \)). After this reordering, the formula can be put in matrix form:

\[
v = \text{diag}(PC^T) + Pv
\]

where \( \text{diag}(M) \) is a column vector containing the elements on the diagonal of matrix \( M \), \( v \) is a \( n \)-dimensional column vector containing the expected costs-to-go, \( V_\pi(k) \), for each of the \( (n - 1) \) first states and a 0 as last \( (n \text{th}) \) element, while \( C \) is the matrix containing the immediate costs \( c(k, k') \).

From this Markov chain, one can also compute the expected number of visits to each state in the following way ([96, 127]). First, observe that the states of the Markov chain can be divided into two families: the transient states, \( k \in \{1, \ldots, n-1\} \), and one absorbing state, \( n \). The transition-probabilities matrix \( P \) can therefore be rewritten in partitioned form,

\[
P = \begin{bmatrix} Q & r \\ 0^T & 1 \end{bmatrix}
\]

where \( Q \) is the \((n-1) \times (n-1)\) substochastic matrix of transition probabilities among the transient states, \( r \) is an \((n-1) \times 1\) column vector representing the transition probabilities from transient states to the absorbing state \( n \) and \( 0 \) is an \( 1 \times (n-1) \) row vector full of 0’s. Now, it is well-known that the probability
of finding a random walker, starting from state 1 at \( t = 0 \), in any state at time step \( t \) is provided by \( x(t) = (P^T)^t e_1 \) where \( x(t) \) is an \( n \times 1 \) column vector with components \( x_i(t) \) corresponding to the probability that the random walker will be in state \( s = i \) at time step \( t \) and \( e_1 \) is a \( n \times 1 \) column vector containing a 1 as first element and 0’s otherwise. Focusing the analysis on the transient states only, the corresponding probability of finding the random walker in any transient state at time step \( t \) is \( \tilde{x}(t) = (Q^T)^t \tilde{e}_1 \), where \( \tilde{x}(t) \) and \( \tilde{e}_1 \) are \( (n-1) \times 1 \) column vectors obtained from \( x(t) \) and \( e_1 \) by removing the last \((n)\)th element.

Therefore, the expected number of visits to each transient state, when starting from state 1 at time \( t = 0 \), is provided by

\[
N = \sum_{t=0}^{\infty} \tilde{x}(t) = \sum_{t=0}^{\infty} (Q^T)^t \tilde{e}_1 = (I - Q^T)^{-1} \tilde{e}_1 = N^T \tilde{e}_1 \quad (3.5)
\]

The matrix \( N = (I - Q)^{-1} \) is usually called the fundamental matrix of the Markov chain ([96, 127]). Its elements \( n_{ij} = [N]_{ij} \) correspond to the expected number of times the process is in transient state \( j \) if it is initiated in state \( i \). Thus, the column vector \( n \) contains the expected number of visits to each transient state, when starting from state 1.

The fundamental matrix can also be computed thanks to a Jacobi-like iterative procedure, \( N \leftarrow I + QN \), which converges since \( Q \) is substochastic.

In Appendix C, we show that the expected costs \( v \) can be expressed in closed form in terms of the fundamental matrix \( N \). Indeed, by partitioning the cost matrix \( C \), containing the immediate costs \( c(k, k') \), as

\[
C = \begin{bmatrix} D & s \\ \infty^T & 0 \end{bmatrix},
\]

Equation (3.3) can be solved in terms of \( \tilde{v} \) and re-expressed as (see Appendix C)

\[
\tilde{v} = N \text{diag}(QD^T + rs^T) \quad (3.7)
\]

where \( \tilde{v} \) is a \((n-1) \times 1\) column vector containing the \((n-1)\) first elements of \( v \) (the \(n\)th element being trivially equal to 0). These relationships will be useful later, when deriving the necessary conditions of optimality.

### 3.1.3.2 Controlling randomness by fixing the entropy spread in the network

Now that we have introduced the problem, we will explain how we manage the exploration. At each state \( k \), we compute the Shannon entropy ([40, 93, 161]) of
3.1. Randomized Shortest-Path Problem: Global Entropy

the transition probabilities:

\[ h_k = - \sum_{k' \in S(k)} p_{kk'} \log p_{kk'} \] (3.8)

In matrix form, Equation (3.8) becomes

\[ \mathbf{h} = -\text{diag}(\mathbf{P}(\log \mathbf{P})^T) \] (3.9)

where the logarithm is taken elementwise. \( \mathbf{h} \) is a \((n - 1) \times 1\) column vector containing the \( h_k \) for each transient state (for the last, absorbing, state \( n \), the entropy is trivially equal to 0).

Thus, \( h_k \) measures the uncertainty about the choice in state \( k \). It is equal to zero when there is no uncertainty at all (\( p_{kk'} \) reduces to a Kronecker delta, \( \delta_{kk'} \)); it is equal to \( \log(m_k) \), where \( m_k \) is the number of admissible choices (outdegree) at node \( k \), in the case of maximum uncertainty (\( p_{kk'} = 1/m_k \); a uniform distribution). Increasing the entropy increases randomness; a maximum entropy aims to a completely random choice since the next state is chosen completely at random, with a uniform distribution, without taking the costs into account.

The **global (weighted) entropy** \( H \), measuring the degree of randomness spread in the whole network is simply defined as the sum of the individual entropy values of each state, weighted by some factor \( u_k \):

\[ H = \sum_{k=1}^{n-1} u_k h_k = -\sum_{k=1}^{n-1} u_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'} \] (3.10)

The factors \( u_k \) weigh the contribution of each node to the global entropy and could depend on the transition probabilities. The next step will be to set the global entropy \( H \) to a predefined value, therefore setting the global randomness occurring in the network, and to compute the optimal transition probabilities minimizing the total expected cost, \( V_\pi(1) \).

3.1.4 Optimal routing policy under entropy constraint

We now turn to the problem of determining an optimal policy under a global entropy constraint. More precisely, we will seek the set of transition probabilities, \( \pi \equiv \{p_{kk'}\} \), for which the expected cost \( V_\pi(1) \) from initial state 1 to destination state \( n \) is minimal while fixing the global entropy in the network, \( H \), to a constant value. The problem is thus to find the transition probabilities \( p_{kk'} \) leading
3.1. Randomized Shortest-Path Problem: Global Entropy

to the minimal expected cost. It can be formulated as a constrained optimization problem involving a Lagrange function: minimize $V_\pi(1)$ subject to the constraint (3.10). In other words, we seek $P^* = \arg \min_P [V_\pi(1)]$ subject to the constraint (3.10).

In Appendix D, we derive the form of the optimal transition probability distribution within state $k$, which appears to be a multinomial logit, or Boltzmann, distribution:

$$p_{kk'} = \frac{\exp \left[ -n_k \eta u_k \left( c(k, k') + V(k') \right) + \frac{1}{u_k} \sum_{l \neq n} \left( \partial u_l / \partial p_{kl} \right) h_l \right]}{\sum_{l' \in S(k)} \exp \left[ -n_k \eta u_k \left( c(k, l') + v(l') \right) + \frac{1}{u_k} \sum_{l \neq n} \left( \partial u_l / \partial p_{kl} \right) h_l \right]}, \text{ for } k \neq n$$

(3.11)

where $n_k$ is the $k$th element of $n$ (containing as entries the first row of the fundamental matrix $N$) and $V(k)$ is the minimum expected cost computed thanks to Equation (3.7). The parameter $\eta > 0$ controls the entropy: the larger $\eta$, the larger the entropy. Notice that $\eta$ could be found from the relationship $H = -\sum_{k=1}^n u_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'}$ by using for instance a bisection algorithm, but this is not needed.

3.1.4.1 Fixing the expected entropy spread in the network

By looking to Equation (3.11), we observe that the entropy is preferably spread on the states that are seldom visited (having a small $n_k$). This behaviour is intuitively not very appealing since the global entropy does not properly reflect the amount of randomness in this case. Therefore, a convenient choice would be to weight the entropy related to each node, $h_k$, by the expected number of visits to this node, that is, to set $u_k = n_k$.

$$H = \sum_{k=1}^n n_k h_k = -\sum_{k=1}^n n_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'}$$

(3.12)

The global entropy $H$ can be interpreted as the expected total entropy spread in the network, when traveling from node 1 to node $n$. We finally have to compute the second term $\sum_{l \neq n} (\partial u_l / \partial p_{kl}) h_l = \sum_{l \neq n} (\partial n_l / \partial p_{kl}) h_l$ in (3.11), which is done in Appendix E:

$$\sum_{l \neq n} (\partial n_l / \partial p_{kl}) h_l = n_k e_k^T N h = n_k \sum_{l \neq n} n_{kl} h_l$$

(3.13)
3.1. Randomized Shortest-Path Problem: Global Entropy

where \( n_{k,l} \) is element \((k', l)\) of matrix \( N \) and \( \tilde{e}_k \) is a \((n-1) \times 1\) column vector full of 0’s except its \( k \)th entry containing a 1. Let us rewrite Equation (3.11) by setting \( u_k = n_k \), and using Equation (3.13),

\[
p_{kk'} = \frac{\exp \left[ -\frac{1}{\eta} \left( c(k, k') + V(k') \right) + \kappa_{k'} \right]}{\sum_{l' \in S(k)} \exp \left[ -\frac{1}{\eta} \left( c(k, l') + V(l') \right) + \kappa_{l'} \right]} \text{, with } k \neq n
\]

where

\[
\begin{aligned}
\kappa_{k'} &= \sum_{l \neq n} n_{k'l} h_l, \text{ for } k' \neq n \\
\kappa_n &= 0, \text{ for destination state } n
\end{aligned}
\]

Notice that the second term in the exponential of the numerator of Equation (3.14), \( \kappa_{k'} \), can be interpreted as the expected entropy when starting from state \( k' \). When the global entropy \( H \) is small, that is, \( \eta \) is small as well, the first term in the exponential dominates the second one, which can be neglected in this special case.

Moreover, when \( \eta \to 0 \), the largest \( p_{kk'} \) will dominate the other terms in the exponential of (3.14) and \( \kappa_{k'} \to 0 \), with the result that the rule (3.2) reduces to Bellman’s equations for finding the shortest path from the initial state to the destination state.

Equations (3.7) and (3.14) are thus the necessary optimality conditions. The parameter \( \eta \) is supposed to be provided a priori by the user, according to the desired degree of randomness he is willing to concede: in this work, no attempt has been made to optimize or estimate it. This would, however, be an interesting research topic for further work.

3.1.4.2 Fixing the expected entropy per visit

Yet another sensible choice would be to fix the expected entropy per visit,

\[
H = \frac{\sum_{k=1}^{n} n_k h_k}{\sum_{l=1}^{n} n_l} = -\sum_{k=1}^{n} \pi_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'}
\]

\[
(3.15)
\]
3.1. Randomized Shortest-Path Problem: Global Entropy

where \( \pi_k = \frac{n_k}{\sum_{l=1}^{n} u_l} = n_k/n \). A few calculus, similar to the previous computation, shows that the resulting necessary conditions are

\[
p_{kk'} = \exp \left[ -\frac{1}{\eta} (c(k, k') + V(k')) + \kappa_{k'} \right] \sum_{l' \in S(k)} \exp \left[ -\frac{1}{\eta} (c(k, l') + V(l')) + \kappa_{l'} \right], \text{ with } k \neq n
\]

where \( \kappa_{k'} = \sum_{l \neq n} n_{k'l}h_l - \left( \sum_{l \neq n} n_{k'l} \right) \left( \sum_{l' \neq n} \pi_{l'h_{l'}} \right) \), for \( k' \neq n \)

\[
\kappa_n = 0, \text{ for destination state } n
\]

Other choices are, of course, possible, depending on the problem at hand.

3.1.4.3 Using a reference a priori policy by fixing the Kullback divergence

In this section, we show that we could start from a reference, a priori, policy, denoted by \( p_{kk'}^{ref} \), and fix the Kullback divergence \( J \) between this reference policy and the policy we are seeking. Thus, we define \( j_k \) as

\[
j_k = \sum_{k' \in S(k)} p_{kk'} \log \frac{p_{kk'}^{ref}}{p_{kk'}}
\]

and \( J \) as

\[
J = \sum_{k=1}^{n} u_k j_k = \sum_{k=1}^{n} u_k \sum_{k' \in S(k)} p_{kk'} \log \frac{p_{kk'}^{ref}}{p_{kk'}}
\]

By proceeding as in previous section, we easily obtain

\[
p_{kk'} = \frac{p_{kk'}^{ref} \exp \left[ -\frac{n_k}{\eta u_k} (c(k, k') + V(k')) + \frac{1}{u_k} \sum_{l \neq n} (\partial u_l / \partial p_{kk'}) j_l \right]} {\sum_{l' \in S(k)} p_{kl'}^{ref} \exp \left[ -\frac{n_k}{\eta u_k} (c(k, l') + V(l')) + \frac{1}{u_k} \sum_{l' \neq n} (\partial u_l / \partial p_{kl'}) j_l \right]}, \text{ for } k \neq n
\]

This time, a sensible choice could be to weight the entropy related to each node, \( h_k \), by the expected number of visits to this node for the reference policy,
3.1. Randomized Shortest-Path Problem: Global Entropy

$n_k^\text{ref}$, that is, to set $u_k = n_k^\text{ref}$. This leads to the following necessary conditions

$$p_{kk'} = \frac{p_{kk'}^\text{ref} \exp \left[ -\frac{n_k}{\eta n_k^\text{ref}} (c(k, k') + V(k')) \right]}{\sum_{l' \in S(k)} p_{kl'}^\text{ref} \exp \left[ -\frac{n_k}{\eta n_k^\text{ref}} (c(k, l') + V(l')) \right]}, \text{ for } k \neq n$$

(3.20)

**Algorithm 1: Computation of an optimal policy**

1. **begin**
   
   **Input**
   
   1. Node 1 is the initial node while node $n$ is the destination node.
   
   2. The network has one single connected component.
   
   3. $\eta > 0$: the parameter controlling the degree of randomness.
   
   4. $C = \begin{bmatrix} D & \mathbf{s} \\ \infty^\top & 0 \end{bmatrix}$: the $n \times n$ cost matrix; node $n$ is the destination node.
   
   **Initialisation**
   
   5. $P \leftarrow \begin{bmatrix} Q & \mathbf{r} \\ \mathbf{0}^\top & 1 \end{bmatrix}$, for instance by setting $p_{kk'} = \frac{1}{m_k} (k \neq n)$ where $m_k$ is the outdegree of node $k$.
   
   **Repeat**
   
   6. if $(I - Q)$ isn’t of full rank then
   
   7. return Error: the fundamental matrix is not invertible
   
   8. end if
   
   9. $N \leftarrow (I - Q)^{-1}$
   
   10. $\mathbf{v} \leftarrow N \text{ diag}(QD^\top + r\mathbf{s})$
   
   11. $h \leftarrow -\text{diag}(P \text{ log} P)^\top$
   
   12. 
   
   13. $p_{kk'} = \frac{1}{\eta} \left( e^{\frac{1}{\eta} (c(k, k') + V(k')) + h_{kk'}} \right)$, for all $k \neq n, k'$
   
   14. *Until* convergence of $P$
   
   **Return** The policy $P = \begin{bmatrix} Q & \mathbf{r} \\ \mathbf{0}^\top & 1 \end{bmatrix}$: the transition-probabilities matrix.
   
   **end**

**Table 3.1:** Computation of an optimal policy while maintaining the expected entropy spread in the network: An iterative algorithm.
3.1. Randomized Shortest-Path Problem: Global Entropy

### 3.1.5 Computation of an optimal policy

Equations (3.7) and (3.14) suggest a simple iterative procedure, similar to the well-known value-iteration algorithm, for the computation of the policy. This algorithm is detailed in Table 3.1 and will be used in all our experiments. Notice that, instead of computing the fundamental matrix, we prefer to solve two linear systems of equations at each iteration.

The algorithm is obtained by performing a block coordinate descent on the following Lagrange function:

\[
\mathcal{L} = v(1) + \sum_{k \neq n} \lambda_k \left[ v(k) - \sum_{k' \in S(k)} p_{kk'} (c_{kk'} + v(k')) \right] \\
+ \lambda_n [v(n) - 0] + \sum_{k \neq n} \mu_k \left[ \sum_{k' \in S(k)} p_{kk'} - 1 \right] \\
+ \eta \left[ \sum_{k \neq n} n_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'} + H_0 \right] \\
+ \sum_{k' \neq n} \phi_{k'} \left[ n_{k'} - \sum_{k \neq n} p_{kk'} n_k - \delta_{1k'} \right],
\]

which is identical to the Lagrange function defined in Appendix D, Equation (E1), except that the formula for computing the expected number of passages, \(n_k\), is now made explicit, so that the \(n_k\) can now be considered as independent variables. Indeed, \(n = N^T e_1\), from which we deduce \(n = Q^T n + e_1\) and thus \(n_{k'} = \sum_{k \neq n} p_{kk'} n_k - \delta_{1k'}\). Notice also that the \(\eta\) parameter related to the temperature is fixed a priori (it is not considered as a variable) and is provided by the user. The necessary conditions of optimality are the same as for the previous formulation detailed in Appendix D and displayed in Equation (3.14).

Setting the partial derivative of \(\mathcal{L}\) in terms of \(\mu_k\) and \(p_{kk'}\) equal to zero while fixing all the other parameters provides the update equations for the transition probabilities matrix \(P\), which is uniquely attained. On the other hand, setting the partial derivative in terms of the other parameters (except \(\eta\), considered as fixed) equal to zero provides the re-estimation equations for the expected costs \(v\) and the expected entropies \(\kappa\) (we easily find \(\phi_k = \eta \kappa_k\)), which are both uniquely attained provided the matrix \((I - Q)\) is invertible. These two steps are iterated until convergence (see Table 3.1). It is known that a block coordinate descent
3.1. Randomized Shortest-Path Problem: Global Entropy

Algorithm converges to a local minimum for continuously differentiable objective functions provided the individual minimum for each block of variables is uniquely attained (see, e.g., [20], p. 268), which is indeed the case. Moreover, it is shown in [152] that a second model called Akamatsu’s model solves the same optimization problem whose solution is unique in this new formalism. Based on this fact, we provide in [152] an informal proof showing that the stationary point obtained by the algorithm in Table 3.1 must be a global minimum.

Computationally speaking, it is very demanding since it requires, in addition to be iterative, either the computation of the fundamental matrix, \( N = (I - Q)^{-1} \), or the solution of two linear systems of equations at each iteration. In [152] we introduce a much more efficient procedure to determine the optimal policy, which is therefore recommended instead of algorithm of Table 3.1. However, in changing environments, one could initialize the policy thanks to the procedure introduced in [152] and then rely on local iterative updating rules such as algorithm of Table 3.1.

On the other hand, the decision of quantifying randomness by the expected entropy spread through the network is also questionable; alternative solutions are also conceivable, such as other measures of entropy (see [93]) or simply taking the sum of the variances of the transition probabilities over all the nodes.

3.1.6 Experiments

We illustrate the procedures for solving the randomized shortest-path problem on a simple network and two maze problems. For all experiments, the algorithm detailed in Section 3.1.5 (see algorithm of Table 3.1) was iterated until convergence of the transition probabilities.

3.1.6.1 First experiment

The first experiment analyses the performances of the algorithm in term of average cost and transition probabilities for different global entropies. This experiment is performed on the network shown in Figure 3.1 which is composed of 8 nodes connected by edges of different weights, representing costs. The algorithm searches an optimal path from an initial node to a destination node, for fixed global entropy. We investigated how the algorithm reacts to changes of the global entropy and the impact of these changes on the total expected cost and transition probabilities.

Figure 3.2 displays the average cost to reach destination node 8 when starting
Figure 3.1: A simple network used in our simulations. The immediate costs are indicated on the arcs.

from initial node 1, in function of global entropy $H$ spread in the network. We clearly observe the increase in average cost when the entropy is increased.

Figure 3.3 shows the resulting Markov chain after convergence of the algorithm, for four different values of the global entropy. When global entropy increases, we observe a raise in transition probabilities of the less optimal paths, resulting in the decrease of the optimal-paths probabilities. This confirms that the algorithm explores the path least used (less optimal) with the growth of global entropy.

3.1.6.2 Second experiment

The second experiment is performed on a $40 \times 30$ grid. It aims to reach a goal state, located in the lower right corner, from an initial state, located in the upper left corner. An obstacle is placed on the grid so that the agents have to walk round in order to avoid it. The agent is allowed to move to a neighboring state and a cost of 1 unit is incurred at each move. The goal of this experiment is to observe the paths followed by the routed agents (i.e., the traffic) for various values of the parameter $\theta$ controlling the global entropy. The experiment was performed with three values of $\theta$, i.e., 0.001, 1.0 and 10.0.
3.1. Randomized Shortest-Path Problem: Global Entropy

Figure 3.2: Average cost, $V(1)$, to reach destination node 8 when starting from initial node 1, in terms of global entropy $H$ spread in the network.

The resulting expected number of passages through every cell of the grid for three values of $\theta$ is shown in Figure 3.4. For a value of $\theta$ equal to 10.0 (representing a small global entropy), the algorithm finds the shortest path from the upper left corner to the lower right corner, and no exploration is carried out: the path bypassing the left of the obstacle is the only path traversed. For a $\theta$ of 1.0, the shortest path is still the most used, but another path not on the shortest path are now also investigated. The second path now also routes a significant traffic. Exploitation thus remains a priority over exploration. Finally, for a $\theta$ of 0.001 (representing a large global entropy), the exploitation of the shortest path is no longer the priority; exploration is now clearly favored over exploitation.

3.1.6.3 Third experiment

The third experiment is performed on the network shown in Figure 3.1. This experiment compares three exploration strategies: RSP using global entropy, $\epsilon$-greedy and Boltzmann using local entropy.

Table 3.2 and Figure 3.5 display the average cost to reach destination node 8 when starting from initial node 1, for the RSP, $\epsilon$-greedy and Boltzmann strategies. We observe that our RSP strategy using a global entropy gives better results.
3.1. Randomized Shortest-Path Problem: Global Entropy

Figure 3.3: Resulting Markov chain together with the transition probabilities and the average cost, for four different values of the global entropy.

in term of average cost than $\epsilon$-greedy and Boltzmann which uses a local entropy.

The $\epsilon$-greedy strategy is much worse with equal entropy than RSP and Boltzmann strategies. The RSP strategy with global entropy is much better, but only slightly better than the Boltzmann strategy.

3.1.6.4 Fourth experiment

The last experiment is performed on a $21 \times 21$ puzzle problem shown in Figure 3.6 and already used in Section 2.7.3. It shows that the randomized algorithm considering a fixed global entropy performs better than the algorithm considering fixed local entropies when the global entropy is indeed fixed to a predefined value in the simulation, and vice-versa. This experiment fixes the global entropy in a first simulation and the local entropy in a second, and compares for each simulation the results obtained for both global and local (see, Chapter 2) entropy models. We have performed the first simulation on four global entropies (i.e., 10, 15, 20 and 25), and the second on four local entropies (i.e., 0.100, 0.125, 0.150 and 0.175).
### 3.1. Randomized Shortest-Path Problem: Global Entropy

#### Figure 3.4:
Expected number of passages through every cell of the grid, for three values of the parameter $\theta$. The agents, starting from the upper left corner, have to reach the lower right corner. An obstacle is placed in the center of the grid so that the agents have to walk round in order to avoid it.

#### Table 3.2:
Average cost, $V(1)$, to reach destination node 8 when starting from initial node 1, in terms of RSP (using global entropy), $\epsilon$-greedy and Boltzmann (using local entropy strategies), for four different values of the entropy.

<table>
<thead>
<tr>
<th>Entropy</th>
<th>RSP (global entropy)</th>
<th>$\epsilon$-greedy (local entropy)</th>
<th>Boltzmann (local entropy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>0.1192</td>
<td>0.1364</td>
<td>0.1201</td>
</tr>
<tr>
<td>1.2</td>
<td>0.4263</td>
<td>0.4691</td>
<td>0.4278</td>
</tr>
<tr>
<td>2</td>
<td>2.0174</td>
<td>1.0634</td>
<td>1.0369</td>
</tr>
<tr>
<td>3</td>
<td>2.4051</td>
<td>2.4814</td>
<td>2.4618</td>
</tr>
</tbody>
</table>

In Figure 3.6, black cells are inaccessible; blocks represent walls; the agent has to reach the goal state, located in the lower right corner, from the initial state located in the upper left corner. The agent has the choice among eight possible actions in each state: move on cell forward in one of eight directions (i.e., north, south, west, east, north-east, north-west, south-east and south-west). When encountering a wall (an inaccessible cell), the agent has no possibility of moving in the intended direction. The actions north, south, west and east have a cost of 1, while the actions north-east, north-west, south-east and south-west have a cost of 1.5.

Figure 3.7 presents the first simulation, and shows the average cost to reach destination state when starting from the initial state, for both local and global entropy models. We observe by fixing the global entropy that our global entropy-based model obtains better results in term of average cost than our local entropy-based model. This difference is emphasized when we increase the global entropy.
3.1. Randomized Shortest-Path Problem: Global Entropy

Figure 3.5: Average cost, $V(1)$, to reach destination node 8 when starting from initial node 1, in terms of RSP (using global entropy), $\epsilon$-greedy and Boltzmann (using local entropy strategies), for four different values of the global entropy, i.e., 0.5, 1.2, 2 and 3.

Thus, as expected, the global entropy-based model outperforms the local entropy-based model when fixing the global entropy.

Figure 3.8 illustrates the results of the second simulation, and presents the average cost to reach destination state when starting from the initial state, for both local and global entropy models. By fixing the local entropy, as opposed to the first simulation, we observe that our local entropy-based model now obtains better results in term of average cost than our global entropy-based model. By increasing the local entropy, this difference is emphasized.

Thus, this experiment shows that, indeed, the global entropy-based strategy minimizes expected cost for a fixed global entropy while the local entropy-based strategy minimizes expected cost for a fixed local entropy.
3.2 Hierarchical reinforcement learning

In this section, we provide a sketch of how to extend our basic flat decision-making framework (introduced in Chapter 2) to Hierarchical framework including concurrent actions.

3.2.1 Related Work

This related work presents several hierarchical learning methods studied (largely inspired by [90]). Feudal Q-learning [42, 187] involves a hierarchy of learning modules. Roughly speaking, there is a master (i.e., high-level) and a slave (i.e., low-level). The first receives reinforcements from the external environment. Its actions consist of commands that are provided to the slave (low-level learner). The master, when it generates a particular command to the slave, must reward it for taking actions that satisfy the command. So, the master learns a policy mapping states to commands, and the slave learns a policy mapping commands and states to external actions. The set of “commands” and their associated reinforcement functions are established in advance of the learning.

In [164, 165], the authors present a compositional Q-learning (C-QL) consisting of a hierarchy based on the temporal sequencing of subgoals. The high-level goal of the system is to achieve some set of conditions in sequential order. By achieving the conditions, C-QL provides reinforcement for the elemental
3.2. Hierarchical reinforcement learning

Figure 3.7: First simulation: average cost, $V(1)$, to reach destination state when starting from initial state, for the global entropy-based model and the local entropy-based model, while fixing the global entropy spread in the puzzle. We set the global entropy for four different values, i.e., 10, 15, 20 and 25.

tasks (i.e., behaviors achieving some recognizable condition), which are trained first to achieve individual subgoals.

Kaelbling’s Hierarchical Distance to Goal (HDG) algorithm [89] uses a hierarchical approach to solving problems when goals of achievement† are given to an agent dynamically. Roughly speaking, it considers problems in which goals are dynamically input to the learner. The HDG algorithm decomposes the environment into a set of regions whose centers are known as “landmarks”‡. Two cases can arise: (1) the agent is currently in the same region as the goal, then it uses low-level actions to move to the goal; (2) else high-level information is used to determine the next landmark on the shortest path from the agent’s closest landmark to the goal’s closest landmark. Then, the agent uses low-level information to aim toward that next landmark.

Other approaches developed by Dietterich [46] and similar to Singh and Kaelbling methods, rely on a programmer to design a hierarchy including a def-

†The agent should get to a particular state as quickly as possible
‡By analogy with navigation in a harbour
3.2. Hierarchical reinforcement learning

Figure 3.8: Second simulation: average cost, \( V(1) \), to reach destination state when starting from initial state, for the global entropy-based model and the local entropy-based model, while fixing the local entropy. We set the local entropy for four different values, i.e., 0.100, 0.125, 0.150 and 0.175.

In this section, we propose a framework decomposing the given reinforcement learning problem into a set of subproblems. This hierarchical decomposition provides also a splitting of the value function for the given problem into a set of value functions for the subproblems.

By splitting the target MDP into a hierarchy of subtasks (or subproblems), an issue is how subtasks should be specified. There are three general approaches to defining these subtasks [46]. The first approach is to define each subtask in terms of a fixed policy that is provided by the programmer. The “option” method of Sutton, Precup, and Singh [172] takes this approach. The second is to define each subtask in terms of a non-deterministic finite-state controller. The Hierarchy of Abstract Machines (HAM) method of Parr and Russell [133] takes this approach. This method permits the programmer to provide a “partial policy” that constrains the set of permitted actions at each point, but does not specify a com-
3.2. Hierarchical reinforcement learning

plete policy for each subtask. Finally, the last approach describes what it means for the subtask to be completed and what the final reward should be for completing the subtask by defining each subtask in terms of a termination predicate and a local reward function. The method described in this section follows this approach, building upon previous work by Singh, Kaelbling, Dayan and Hinton, Dean and Lin, and Dietterich.

This section thus extends the current flat decision-making model (introduced in Chapter 2) to Hierarchical model including concurrent actions, and is organised as follow. Section 3.2.2 introduces the concept of macro-action. This concept makes a distinction between primitive actions and macro-actions. We can see a macro-action like a sequence of actions chosen from the primitive actions of the problem. Joining actions together as macros can be of great help for solving large problems and can sometimes greatly speed up learning [61, 114]. Macro-actions and hierarchical model might be essential for scaling up reinforcement learning to very large problems. Section 3.2.3 presents the concept of concurrent actions. Concurrency clearly allows agents to execute actions in parallel and to achieve goals more quickly. Section 3.2.4 describes concurrent actions by using the AND/OR graph representation. Finally, Section 3.2.5 presents the unified extended framework with macro-actions and concurrent actions.

3.2.2 Macro-actions and primitive actions

Several problems in RL are too large to be solved practically by searching the state-space using available primitive actions. By using only primitive actions to seek for the goal, the RL system is limited by the depth and the breadth of the search tree.

One solution to handle this issue is through macro-actions. By putting primitive actions together into macro-actions, the solution’s length is shortened. Another advantage of using macro-actions in searching solutions is their capability to find faster solutions in cases where the system was unable to find answers by searching in primitive state-space [75, 99].

Two kinds of action are included for the extension of our RL framework: primitive action and macro-actions. Macro-actions are policies with terminations conditions [114]. On each time step, the agent has the choice between a macro or a primitive action. Once the agent has chosen a macro-action, it selects the primitive actions in accordance with the macro-action’s policy until the macro-action’s termination condition is satisfied. For example, driving from a city A to city B could be a macro-action. This macro-action enables the agent to skip
thinking or planning at the level of street moving. If an accident is encountered along the way, the agent can safely change the primitive actions of walking to bypass the accident while executing the macro-action of going to the city B.

Figure 3.9: An example where the action 3 (in the top level graph) is a macro-action composed of n+1 states. The initial state of the macro-action is state $k_0$ and the terminal state is state $k_n$.

To include macro-actions in the framework in term of Q-value, we must extend the notion of the optimal action-value function, $Q^*$. That is, for each state $s_t$ and macro-action $m$ we define a macro value $Q^*(s_t, m)$, as the maximal expected return given that we start in state $s_t$ and take macro-action $m$ [114]. Thus when a macro-action terminates, we update its value by using the cumulative discounted cost received while executing the macro-action and the expected cost at the resulting state [114]. More precisely, after a multi-step transition from state $s_t$ to state $s_{t+n}$ using macro-action $m$, the approximate action value $Q(s_t, m)$ is updated by:
3.2. Hierarchical reinforcement learning

\[ \hat{Q}(s_t, m) \leftarrow \hat{Q}(s_t, m) + \alpha \left( c_{\Pi_m} + \sum_{i \in U(s_{t+n})} \hat{\pi}_{s_{t+n}}(i) \hat{Q}(s_{t+n}, i) \right) - \hat{Q}(s_t, m), \]

(3.22)

where \( \hat{Q}(s_{t+n}, i) \) is taken over both actions and macro-actions, \( s_{t+n} = f_{s_t}(m) \) and \( s_t \neq d \), and

\[ c_{\Pi_m}(s_t) = E_{\Pi_m}(s_t) \left[ \sum_{l=0}^n c(s_{t+l}, u_{t+l}) \right] \]

(3.23)

is the total expected cost accumulated over a path \( s_t, s_{t+1}, \ldots, s_{t+n} \) in the graph starting from the initial state \( s_t \). The expectation \( E_{\Pi_m} \) is taken on the macro-action policy \( \Pi_m \) that is, on all the random choices of action \( u_{t+l} \), in state \( s_{t+l} \).

As in our local entropy exploration framework (see Chapter 2), the probability distribution of choosing macro-action \( m \) in state \( s_t \) is given by

\[ \hat{\pi}_{s_t}(m) = \frac{\exp \left[ -\theta_{s_t} \hat{Q}(s_t, m) \right]}{\sum_{i \in U(s_t)} \exp \left[ -\theta_{s_t} \hat{Q}(s_t, i) \right]} \]

(3.24)

This is a discrete-time version of the Semi-MDP Q-learning method studied by Bradtke & Duff [28] and proven to converge by Parr [132].

### 3.2.3 Concurrent actions

Now, we investigate a way to model concurrent actions allowing clearly the agents to achieve goals more quickly. In a general way, we formalize the concept of concurrent action to include two situations: (1) a single agent executes multiple parallel processes, and (2) a MAS where many agents act in parallel [146].

To include concurrent actions in our framework, we consider the Concurrent Action Model (CAM) [145, 146] allowing concurrent actions with different durations. In the deterministic case, the action value equation of the original MDP model is almost identical to that of CAM

\[ Q(k, \tilde{i}) = c(k, \tilde{i}) + \sum_{i' \in U(k_i')} \pi_{k_i'}(i') Q(k_i', i'), \text{ with } k_i' = f_k(\tilde{i}) \text{ and } k \neq d \]

(3.25)
3.2. Hierarchical reinforcement learning

which represents the total expected cost from state \( k \) when choosing action \( \hat{i} \). The only difference is that \( \hat{i} \) here is a set of concurrent actions (or joint actions) instead of a single action. The cost \( c(k, \hat{i}) \), incurred when concurrent actions of \( \hat{i} \) are executed in state \( k \), is computed according to the representation of this cost in the learning problem to be solved. For example, if \( c(k, \hat{i}) \) represents the execution time of \( \hat{i} \), then \( c(k, \hat{i}) = \max_{i \in \hat{i}} c(k, i) \), but if \( c(k, \hat{i}) \) represents the use of resource by the concurrent actions of \( \hat{i} \), then \( c(k, \hat{i}) = \sum_{i \in \hat{i}} c(k, i) \). Thus using this model, we can execute actions in parallel, and this is considered as a single joint action.

![Diagram](image)

**Figure 3.10:** An example of concurrent actions where action 3 is a set of concurrent actions, says action 1, 2, and 3.

A specific issue to be addressed by concurrent actions is the determination
3.2. Hierarchical reinforcement learning

of the termination schemes which determines when the next decision epoch occurs. Because multiple concurrent actions may not terminate synchronously, the notion of a decision epoch needs to be generalized. The main contribution of [146] is the introduction of termination schemes. They defined three termination schemes: \( \tau_{\text{any}} \), \( \tau_{\text{all}} \) and \( \tau_{\text{continue}} \). The termination scheme \( \tau_{\text{any}} \) means that an agent makes a new decision if any concurrent action finishes, and the rest of the actions that did not terminate naturally are interrupted. Alternatively, in \( \tau_{\text{all}} \) the agent waits until all concurrent actions that started together to finish before making a new decision. Finally, we can design other termination schemes by combining \( \tau_{\text{any}} \) and \( \tau_{\text{all}} \), like the termination scheme \( \tau_{\text{continue}} \) which is one that always terminates based on the \( \tau_{\text{any}} \) termination scheme, but lets those primary actions that did not terminate naturally continue running, while initiating new primary actions if they are going to be useful. It was proved that optimal policies learned using \( \tau_{\text{any}} \) dominates both \( \tau_{\text{all}} \), and \( \tau_{\text{continue}} \) [145, 146].

3.2.4 AND/OR Graph

We now presents another way to represent concurrent actions by using AND/OR graph. The AND/OR search space is a well known problem solving approach developed in the area of heuristic search [110], that accommodates problem decomposition. Each search algorithm can be understand as a crossing of a directed graph. Standard literature usually distinguishes between two basic types of search graphs [92, 110, 148]. In the simpler case to be considered, the graph is a so-called OR graph. A node in the OR graph represents a given problem to be solved and each arc emerging from this node represents a possible move or decision that can be made at the current state of the search process. A solution is found by crossing the graph following a certain strategy (as depth-first search, breadth-first search or best-first search) and being guided by some heuristics exploiting problem-specific knowledge [110].

For some problems however, it is useful to allow graphs with two kinds of nodes: AND and OR (each node representing a different kind of search algorithms \(^5\)). If at a given search state a certain move is made this may lead to several new problems that all have to be solved. We will next formally define the AND/OR graph.

---

\(^5\)Such AND/OR graphs are the basis of many searching methods employed in different fields.
3.2. Hierarchical reinforcement learning

3.2.4.1 Concurrent actions

AND/OR graphs can be viewed as generalizations of directed graphs [110]. An AND/OR graph $G$ is a directed graph with a special node $s$, called the initial (or root) node, and a nonempty set of terminal nodes. The start node $s$ represents the given problem to be solved, while the terminal nodes correspond to possible solutions. The nonterminal nodes of $G$ are of two types: OR and AND. An OR node can be solved in any one of a number of alternate ways, but an AND node is solved only when every one of its immediate subproblems is solved.

The use of a graph including two types of nodes, OR and AND, allows the representation of concurrent actions. As remind, we have introduce the concept of concurrency as being an action representing a set of concurrent actions. This kind of representation uses a graph including only one type of node: OR node. Graphs AND/OR allow the representation of the concurrent actions in a more explicit way thanks to AND nodes.

Figure 3.11: A representation of state $k$ as OR and AND nodes. AND node is represented by a circle.

Figure 3.11 shows an example of AND/OR graph. In the left side, $k$ is a typical OR node. The admissible actions in state $k$ are $U(k) = \{1, 2, 3\}$, the distribution $\pi_k(u)$ specifies three probabilities $\pi_k(u = 1)$, $\pi_k(u = 2)$, and $\pi_k(u = 3)$. In the right side, $k$ is an AND node. The admissible actions in state $k$ are $U(k) = \{1, 2, 3\}$, but there is no distribution probabilities because the agent has to execute all available actions in $k$. Indeed, when an agent arrives on a node AND, it has to execute all actions emerging from this node to jump to the next node.

In the deterministic case, the state value equation of a AND state $k$ is given

89
3.2. Hierarchical reinforcement learning

by

\[ V(k) = c(k, \tilde{i}) + V(k'_{\tilde{i}}), \text{ with } \tilde{i} = U(k) \text{ and } k \neq d \]  

(3.26)

where \( k'_{\tilde{i}} = f_k(\tilde{i}) \) is a following state and \( c(k, \tilde{i}) \) is the cost incurred when concurrent actions of \( \tilde{i} \) is executed in state \( k \).

Equations (3.25) and (3.26) are similar and represent the special case where all the concurrent actions lead to the same state (see Figure 3.10), i.e \( \forall i \in U(k) \), \( f_k(i) = k'_{\tilde{i}} \) for all AND node \( k \). To extend our model to the most general case (see Figure 3.11), i.e \( \exists (i_1 \in U(k) \text{ and } i_2 \in U(k)) \), \( f_k(i_1) \neq f_k(i_2) \) where \( k \) is an AND node, we take as assumption that the costs corresponding to the execution of actions are independent (i.e., execution time, the use of independent resources, \ldots). In this case, the extension of Equation (3.26) to concurrent actions leading to different states is given by

\[ V(k) = \max_{i \in U(k)} \left( c(k, i) + V(k'_{i}) \right), \text{ with } k'_{i} = f_k(i) \text{ and } k \neq d \]  

(3.27)

For a problem introducing costs which does not respect the assumption of dependence, Equation (3.26) will have a form which depends on the relation existing between these costs.

3.2.5 Macro-actions and Concurrent actions in a common hierarchical framework

We have presented the different elements used in the construction of a hierarchical model allowing the execution of concurrent actions. Indeed, the parallel execution of a set of actions is defined by the concept of concurrent actions. The concurrent actions in this framework are a set of actions containing primitive actions and/or macro actions. As for the concept of macro-actions, it allows the development of a hierarchical learning with several levels. The top level of this hierarchy is a graph, say \( G \), made up of primitive actions, macro-actions and concurrent actions. For each macro-action \( i \) of \( G \), we define, on a lower level in the hierarchy, a subgraph \( G_i \) representing the states of the macro action \( i \). Subgraph \( G_i \) has an initial state \( k_{i_{0}} \), final states \( k_{i_{d}} \), its own policy \( \Pi_i \), and its own value function \( V_i \). Subgraph \( G_i \) can itself have a set of primitive actions, macro-actions and concurrent actions. Its macro actions are also represented by a subgraph defined in a level even lower, and so on.
3.2. Hierarchical reinforcement learning

3.2.5.1 Computation of the Optimal Policy

We consider that agents are sent from the initial state and that they choose an action $i$ in each state $k$ with probability distribution $\pi_k(u = i)$. When an agent chooses an action in a state, say action $i$, three cases can arise: the chosen action is either a primitive action, a macro action or a set of concurrent actions. In the case of primitive action, the agent then performs the action $i$ and incurs the associated cost, $c(k, i)$, and jumps to the new state, $k'$. In the case of macro action, the agent is routed in the subgraph describing the states of action $i$ from the initial state $k'_0$ to a final state $k'_d$. When a final state is reached, the agent updates (in the higher graph) the cost associated to the action $i$, $c(k, i)$, with the total expected cost $\hat{V}(k'_0)$ of the initial state $k'_0$, and the agent jumps to the next state, $k_t$, in the original graph. Finally, in the case of concurrent actions, all actions of $i$ are executed in parallel. Depending to the termination scheme (i.e., $\tau_{\text{any}}, \tau_{\text{all}}$ or $\tau_{\text{continue}}$) and the meaning of the action cost for the learning problem, the agent incurs the associated cost, $c(k, i)$, together with the new state, $k'$.

This allows the agent to update the estimates of the cost, of the policy, and of the average cost until destination; these estimates will be denoted by $\hat{c}(k, i)$, $\hat{\pi}_k(i)$, and $\hat{V}(k)$ and are known (shared) by all the agents.

1. Initialization phase:

- Choose an initial policy, $\hat{\pi}_k(i)$, $\forall i, k$, satisfying the exploration rate constraints and
- Compute the corresponding expected cost until destination $\hat{V}(k)$ by using any procedure for solving the set of linear equations. The $\hat{\pi}_k(i)$ are kept fixed in the initialization phase.

2. Computation of the policy and the expected cost under exploration constraints:

For each visited state $k$, do until convergence:

- Choose an action $i$ with current probability estimate $\hat{\pi}_k(i)$,
  - if $i$ is a primitive action: observe the current cost $c(k, i)$ for performing this action, update its estimate $\hat{c}(k, i)$, and jump to the next state, $k'_i$
    $$\hat{c}(k, i) \leftarrow c(k, i)$$ (3.28)
3.2. Hierarchical reinforcement learning

- if $i$ is a macro action: the agent is routed in the subgraph of the macro action $i$ from the initial state $k^0_i$ to a final state $k^d_i$, update its estimate $\hat{c}(k, i)$, and jump to the next state, $k'_i$

$$\hat{c}(k, i) \leftarrow V(k^0_i)$$ (3.29)

where $V(k^0_i)$ is the total expected cost accumulated over all paths from the initial $k^0_i$ to the final state $k^d_i$ within macro-action $i$.

- if $i$ is a set of concurrent actions: the agent executes in parallel all actions of $i$. The next decision epoch occurs (i.e., the moment when the agent can jump to the next state $k'_i$) depending on the termination scheme (i.e., $\tau_{any}$, $\tau_{all}$ or $\tau_{continue}$) chosen. The updating of the $\hat{c}(k, i)$ depends on the problem to solve. If the cost represents the execution time of an action, $\hat{c}(k, i)$ is updated by

$$\hat{c}(k, i) \leftarrow \max_{j \in i} c(k, j)$$ (3.30)

- Update the probability distribution for state $k$ as:

$$\hat{\pi}_k(i) \leftarrow \exp \left[ -\hat{\theta}_k \left( \hat{c}(k, i) + \hat{V}(k'_i) \right) \right] \sum_{j \in U(k)} \exp \left[ -\hat{\theta}_k \left( \hat{c}(k, j) + \hat{V}(k'_j) \right) \right],$$ (3.31)

where $k'_i = f_k(i)$ and $\hat{\theta}_k$ is set in order to respect the given degree of entropy.

- Update the expected cost of state $k$:
  - if $k$ is an OR state:

$$\begin{cases} 
  \hat{V}(k) \leftarrow \sum_{i \in U(k)} \hat{\pi}_k(i) \left[ \hat{c}(k, i) + \hat{V}(k'_i) \right], \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\
  \hat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state}
\end{cases}$$ (3.32)

  - if $k$ is an AND state:

$$\begin{cases} 
  \hat{V}(k) \leftarrow \max_{i \in U(k)} \left( \hat{c}(k, i) + \hat{V}(k'_i) \right), \text{ with } k'_i = f_k(i) \text{ and } k \neq d \\
  \hat{V}(d) \leftarrow 0, \text{ where } d \text{ is the destination state}
\end{cases}$$ (3.33)
3.2. Hierarchical reinforcement learning

### 3.2.5.2 Learning the hierarchical decision process

Algorithms of tables 3.3 and 3.4 illustrates how agents make decisions and learn the value iteration of their actions. The main variables and functions are:

- $G$: the graph where the agents are routed. Each graph represents a learning problem and has its own policy, value function and cost matrix.
- $\text{entropy}$: represents the exploration rate fixed by the user.
- $s$: the current state observed from the environment
- $\hat{\pi}$: the policy of the current graph learned by the agents.
- $\hat{V}$: the value function of the current graph evaluated by the agents.
- $\hat{c}$: represents the execution cost of each state and each available action in this state.
- $\text{terminalState}(\text{graph } G)$: return the set of terminal states in graph $G$.
- $\text{chooseAction}(\text{policy } \pi_s, \text{state } s)$: return an action available at state $s$ and chosen by following the policy $\pi_s$.

Step 1.2 corresponds to the initialisation stage. We get the initial state of the graph, its policy, its value function and its cost matrix. Step 1.3-1.8 represents the routage of an agent from the initial state to a destination (or termination) state by executing primitive actions, macro-actions and concurrent actions. Step 1.9 correspond to the updating stage of the value function, policy and current cost for each visited state. Step 1.13 returns the total expected cost from the initial state and it is primarily interesting for macro actions.

Algorithm 2 of table 3.4 executes a set ($m$) of concurrent actions. Steps 2.4-2.10 execute the set of concurrent actions in parallel according to the termination scheme chosen. They take into account the macro-actions (Steps 2.5-2.6) and primitive actions (Steps 2.7-2.8). Steps 2.11-2.17 wait the end of actions following the termination scheme. Step 2.18 computes the total cost of executing the set of concurrent actions. As remind, this total cost depends of the learning problem to solve.
3.2. Hierarchical reinforcement learning

Algorithm 1: cost $\leftarrow$ function learning($G, \text{entropy}$)

1.1 begin

1.2 Initialisation

1.3 if ($s \notin \text{terminalState}(G)$)

1.4 $a \leftarrow \text{chooseAction}(\pi(s), s)$

1.5 if (isMacroAction($a$) = true)

1.6 cost $\leftarrow \text{learning} \left( \text{getGraph}(a), \text{entropy} \right)$

1.7 else

1.8 if (isConcurrentActions($a$) = true)

1.9 cost $\leftarrow \text{ConcurrentActions}(a, \text{entropy})$

1.10 else

1.11 cost $\leftarrow \text{primitiveAction}(a)$

1.12 end if

1.13 end if

1.14 Updating Rules

1.15 $\tilde{c}(s, a) \leftarrow \text{cost}$

1.16 if (isORSState($s$))

1.17 $\tilde{V}(s) \leftarrow \sum_{i \in U(s)} \tilde{\pi}_s(i) \tilde{c}(s, i) + \tilde{V}(s'_i)$, with $s'_i = f_x(i)$

1.18 else (isANDState($s$))

1.19 $\tilde{V}(s) \leftarrow \max_{i \in U(s)} \left( \tilde{c}(s, i) + \tilde{V}(s'_i) \right)$, with $s'_i = f_x(i)$

1.20 end if

1.21 $\tilde{\theta}_s \leftarrow \text{computeTheta(entropy, } \tilde{\pi}_s)$

1.22 for each action $i \in U(s)$

1.23 $\tilde{\pi}_s(i) \leftarrow \frac{\exp[-\tilde{\theta}_s(\tilde{c}(s, i) + \tilde{V}(s'_i))]}{\sum_{j \in U(s)} \exp[-\tilde{\theta}_s(\tilde{c}(s, j) + \tilde{V}(s'_j))]}$, with $s'_i = f_x(i)$

1.24 end for

1.25 $s \leftarrow \text{observeCurrentState}(G)$

1.26 Goto 1.3

1.27 end if

1.28 return $\tilde{V}(\text{initialState}(G))$

1.29 end

Table 3.3: The learning algorithm used to route the agent through the network. It includes the macro-actions and concurrent actions in a common hierarchical model.
3.2. Hierarchical reinforcement learning

Algorithm 2: cost ← function ConcurrentActions(m,entropie)

```plaintext
begin
arrayCost ← new Array(1,size(a))
index ← 1
for each action iεa, executes in parallel with the chosen termination scheme
if ( isMacroAction(i) = true)
arrayCost[index] ← learning(getGraph(i),entropie)
else
arrayCost[index] ← primitiveAction(i)
end if
end for
if (terminationScheme(τ_all) = true)
wait("all actions of m to finish")
else
if (terminationScheme(τ_any) = true)
wait("any action of m to finish")
end if
end if
cost ← compute ConcurrentCost(arrayCost)
return cost
end
```

Table 3.4: The algorithm executing a set of concurrent actions with a determined termination scheme. It includes primitive actions and macro-actions used to route the agent through the network. It includes the macro-actions and concurrent actions.
Part III

Reputation Models
Chapter 4

A Probabilistic Reputation Model

Contents

4.1 Related work .............................................................. 102
4.2 Probabilistic models of reputation (PMR) ........................ 105
  4.2.1 The basic model: PMR1 ......................................... 107
  4.2.2 A more sophisticated model accounting for truncation: PMR2 ........................................ 111
  4.2.3 A simplified model with truncation: PMR3 ............... 114
  4.2.4 A further simplified model with truncation: PMR4 .... 115
  4.2.5 Introducing a Bayesian prior on the parameters: PMR5 and simple PMR5 (sPMR5) .................. 117
  4.2.6 Some guidelines on the use of the probabilistic reputation models ....................................... 119
4.3 Brockhoff and Iterative Refinement Models ......................... 120
  4.3.1 Brockhoff Model .................................................. 120
  4.3.2 Iterative Refinement Models .................................. 121
4.4 Experiments ............................................................. 122
  4.4.1 Experimental settings .......................................... 122
  4.4.2 First preliminary, experiment: Comparing PMR1, PMR2 and PMR3 .................................. 124
  4.4.3 Second experiment: Varying the stability $\sigma_y$ of the consumers ....................................... 126
Motivations

The Internet has created a lot of new opportunities to interact with strangers. These interactions involve various kinds of applications, such as online markets. After just three years of business, eBay had already conducted over one billion dollars in transactions in 1998, and by 2007 [50] this figure had climbed to $59.4 billion. A total of 276.3 million users either bid or listed an item on eBay during 2007, and according to a July 2005 survey conducted for eBay by ACNielsen International Research, an estimated 724,000 Americans and 170,000 Europeans report eBay as a primary or secondary source of income [107].

With the growth of online markets comes an increasing need for bidders and sellers to engage in transactions with counterparts with whom they have had little or no previous interaction. This new type of markets has introduced a new risk dimension to traders: the winner of the auction might not deliver payment, the seller might not deliver the item, or the delivered item might not be as the seller described [73]. These risks are an important restraint to the growth of online markets. One of the principal means by which online auction sites try to mitigate these risks associated with exchange among strangers is to use electronic reputation or feedback mechanisms. Such mechanisms aim at providing the type of information available in more traditional close-knit groups, where members are frequently involved in one another's dealings. Bolton et al compare in [24] trading in a market with online feedback to a market without feedback, as well as to a market in which the same people interact with one another repeatedly. They found that the feedback mechanism induces quite a substantial improvement in transaction efficiency. Using such mechanisms is a means to distinguish honest sellers from dishonest ones. Without a mechanism for sellers to develop a reputation, dishonest sellers might drive out honest ones, leading to a kind of market failure. The idea is that even if the consumer can not try the product or service in advance [84], he can be confident that it will be what he expects as long as he trusts the seller. A seller with a good reputation score therefore has a significant advantage in case the product quality can not be verified in advance. Moreover, as shown in several works [106, 107, 112], the reputation of the sellers (and consequently, the method used to compute the reputation scores) has much influence on auctions (for example, on the decision
CHAPTER 4. A PROBABILISTIC REPUTATION MODEL

of a bidder to participate in an auction or on the price of an auction. Notice then that Livingston et al enumerated in [106] various reasons to doubt that online market’s (such as eBay) reputation mechanism should work (e.g., sellers could build a reputation by selling relatively inexpensive items and then cheat in auctions of more expensive items).

Before being studied in such on-line markets, reputation and trust have become important topics of research in many fields, as shown in this section.

Reputation has long been of interest to economists. Kreps et al use reputation to explain the cooperation observed in experimental studies of Prisoners’ dilemma game [100]. Economic theory indicates that there is a balance between the cost of establishing a good reputation and the financial benefit of having a good reputation, leading to an equilibrium [84]. Variations in the quality of services or goods can be a result of deliberate management decisions or uncontrollable factors, and whatever the cause, the changes in quality will necessarily lead to variations in reputation. Although a theoretic equilibrium exists, there will always be fluctuations, and it is possible to characterize the conditions under which oscillations can be avoided or converge towards the equilibrium [162].

Scientometrics [124], referring to the study of measuring research outputs such as journal impact factors, also used the notion of reputation. In this context, reputation refers to the number of cross citations that a given author or journal has accumulated over a period of time.

In the field of social science, reputation as a quantitative concept is often studied as a network parameter associated with a society of agents [36, 185]. Reputation or prestige is often measured by various centrality measures. An example of such a measure is provided by Katz in [94], taking into account not only the number of direct links between agents but, also, the number of indirect links (going through intermediaries) between agents.

Another consideration, closely related to the work done in social sciences, about reputation systems is its collaborative aspect. Indeed, reputation systems could also be called collaborative filtering systems [84] to reflect their collaborative nature (notice that Katz’ measure has recently been rediscovered in the context of collaborative recommendation [74] and kernel methods where it is known as the von Neumann kernel [163]). Collaborative filtering systems (also called recommender systems) try to provide people with recommendations of items they will appreciate, based on their past preferences (evaluations), history of purchase, and demographic information. (see, for example, [8, 57, 72]). As stated by Josang in [84], collaborative filtering systems have similarities with reputation systems (in that both collect ratings from members in a community)
4.1. Related work

as well as fundamental differences: (1) in recommender systems different people have different tastes, and rate things differently according to subjective taste while in reputation systems, all members in a community should judge the performance of a transaction partner or the quality of a product or service consistently; (2) recommender systems take ratings subject to taste as input, whereas reputation systems take ratings assumed insensitive to taste as input; and (3) recommender systems and reputation systems assume an optimistic and a pessimistic world view respectively (recommender systems assume all participants to be trustworthy and sincere while reputation systems, on the other hand, assume that some participants try to misrepresent the quality of services in order to make more profit).

This section has highlighted the two fundamental aspects of reputation systems: the presence of communication protocols (allowing participants to provide ratings about transaction partners as well as to obtain reputation scores of potential transaction partners) and a reputation computation method to derive aggregated scores for each participant, based on received ratings, and possibly also on other information. Notice that, in his survey of trust and reputation systems [84], Josang also distinguishes centralised from distributed reputation systems.

This chapter is devoted to the definition of new reputation computation models. Part of this work is a joint work together with Fouss and Saerens, and has been published in [56].

The present chapter is organized as follows. Section 4.2 introduces a probabilistic model of reputation (based on a simple consumer-provider interaction model where consumers order items to providers and rate them while providers supply the items with a certain quality) as well as two extensions of this basic model. Expectation-maximization (EM) algorithms allowing to provide estimations of the parameters only based on the ratings are developed for all the models. Section 4.3 presents the Brockhoff and Iterative Refinement models adapted here to our reputation context. Section 4.4 describes the experimental settings as well as the four experiments which are conducted. The results of all these experiments are also part of Section 4.4.

4.1 Related work

Reputation is indicative of the confidence and trust placed in a system or entity’s ability to deliver desired results. It also represents the user perception and a reputation system is intended to aggregate and disseminate feedback about partici-
4.1. Related work

pants’ past behavior. These systems encourage trustworthiness and help people choose the right system for service request. This section describes various principles for computing reputation and trust measures, some of them are used in commercial applications. According to a recent survey [84], trust and reputation systems can be classified into several categories, as described now (and inspired by [102]).

The simplest form of computing reputation scores is simply to sum the number of positive ratings and negative ratings separately, and to keep a total score as the positive score minus the negative score (used in eBay’s reputation forum as described in [144]). The advantage is that anyone can understand the principle behind the reputation score, the disadvantage is that it is primitive and therefore gives a poor picture on participants’ reputation score although this is also due to the way rating is provided. Computing the average of all ratings as reputation score is also used in the reputation systems of numerous commercial web sites (see, for example, [159]). Advanced models in this category compute a weighted average of all the ratings, where the rating weight can be determined by factors such as rater trustworthiness/reputation, age of the rating, distance between rating and current score, etc.

Discrete trust models have also been suggested in several works [3, 32, 34]. Humans are often better able to rate performance in the form of discrete verbal statements, than continuous measures. In the model of Abdul-Rahman et al [3], the trust concept is divided into direct and recommender trust. The agent’s belief in another agent’s trustworthiness (direct trust) is represented within a certain context to a certain degree (“very trustworthy”, “trustworthy”, “untrustworthy”, or “very untrustworthy”). Recommender trust can be derived from word-of-mouth recommendations, which they consider as reputation. In [32], Cahill et al investigate how entities that encounter each other in unfamiliar, pervasive computing environments can overcome initial suspicion to allow secure collaboration to take place. More precisely, their model focuses on a set of trust values (whose elements represent degrees of trust) with two orderings: the first one reflecting the fact a particular trust value may represent a higher level of trust than another whereas the second one reflects that a particular trust value may be more informative than another.

Probabilistic models [29, 102] directly model the statistical interaction between the consumers and the providers. For instance, in the context of food quality assessment, Brockhoff & Skovgaard [29] analyze sensory panel data where individuals (raters) evaluate different items. In their model, they assume that each rater evaluates each item through a linear regression of its quality. The
4.1. Related work

models proposed in this work belong to probabilistic models category and can be considered as an extension of the Brockhoff & Skovgaard model, as detailed later. On the other hand, Laureti et al [102] propose a model for raters evaluating objects. Each object has a quality that is estimated by a weighted average of the ratings provided for this object. The weighting factor depends on the rater and is proportional to a reliability score of the rater, defined as the inverse of his variance.

Bayesian systems (see, for example, [83, 123, 124, 189]) are based on computing reputation scores by statistical updating of probability density functions. The a posteriori (i.e., the updated) reputation score is computed by combining the a priori (i.e., the previous) reputation score with the new rating. The advantage of Bayesian systems is that they provide a theoretically sound basis for computing reputation scores, whereas the main disadvantage is that it might be too complex and difficult to interpret.

Belief theory is a framework related to probability theory, but where the sum of probabilities over all possible outcomes not necessarily add up to 1, and the remaining probability is interpreted as uncertainty. Josang [81, 82] has proposed a belief/trust metric called opinion as well as a set of logical operators that can be used for logical reasoning with uncertain propositions. Yu and Singh [193] have proposed to use belief theory to represent reputation scores.

Trust and reputation can also be represented as linguistically fuzzy concepts, where membership functions describe to what degree an agent can be described as, for example, trustworthy or not trustworthy. Fuzzy logic provides rules for reasoning with fuzzy measures of this type. The methodology proposed by Manchala [111] as well as the REGRET reputation system proposed by Sabater and Sierra [149, 150, 151] fall in this category (see also [143]). In Sabater and Sierra’s scheme, individual reputation is derived from private information about a given member, social reputation is derived from public information about an agent, whereas context dependent reputation is derived from contextual information.

Flow models represent systems that compute trust or reputation by transitive iteration through looped or arbitrarily long chains. Some flow models assume a constant trust/reputation weight for the whole community, and this weight can be distributed among the members of the community. Participants can only increase their trust/reputation at the cost of others. Google’s PageRank [130], the Appleseed algorithm [197], and Advogato’s reputation scheme [104] belong to this category. In general, a participant’s reputation increases as a function of incoming flow, and decreases as a function of outgoing flow. In the case of
4.2 Probabilistic models of reputation (PMR)

Google, many hyperlinks to a web page contributes to an increased PageRank score for that web page. Flow models do not always require the sum of the reputation/trust scores to be constant. One such example is the EigenTrust model [91] which computes agent trust scores in P2P networks through repeated, iterative, multiplication, and aggregation of trust scores along transitive chains until the trust score of each member of the P2P community converge to a stable value.

4.2 Probabilistic models of reputation (PMR)

The proposed reputation models are based on the following simple consumer-provider interaction model. First, we assume a consumer (or buyer) is ordering an item to a provider (or seller), which has some internal “quality of service” score. He will supply the item with a quality following a normal law, centered on his internal “quality of service”. The consumer, after the reception of the item, rate it according to a linear function of its quality (a standard regression model). This regression model accounts for the bias of the consumer in providing ratings as well as its reactivity towards changes in quality. Moreover, the constancy of the provider in supplying a constant quality level for delivering the items is estimated by the standard deviation of the normal law. Symmetrically, the consistancy of the consumer in providing similar ratings for a constant quality is quantified by the standard deviation of the normal law of ratings generation. This is the framework for the basic model of consumer-provider interaction, called Probabilistic Model of Reputation (PMR1).

A second more sophisticated model is also investigated. It accounts for the fact that ratings are often constrained to a specific range of values. Consequently, the rating provided by the consumer is truncated in order to scale within a limited interval, for instance $[-3, +3]$. This second model, PMR2, leads to a more complex parameters updating scheme. Therefore, a third model, which is an approximation of the second one, is also introduced. It corresponds to a simplification of the second model that takes truncation into account while keeping the simplicity of the first model. It will be called PMR3. Experiments show that this model behaves well while keeping the implementation as simple as possible. Then, a fourth, further simplified, model with truncation (PMR4), reducing dramatically the number of variables to be estimated is studied. It assumes that the providers always supply the same quality for the items (the quality of the item is deterministic and no more a random variable). This model greatly simplifies the previous model; in particular it reduces significantly the number of variables to
be estimated. We expect this model to be particularly useful when the number of providers is greater than the number of consumers, in which case the number of parameters to estimate would be too high.

The last extension (PMR5) consists in introducing a prior probability distribution on the reputation parameter as well as the parameters characterizing the consumer (a Bayesian framework). This allows to regularize the estimate and to take the number of ratings into account when computing the reputation score. Indeed, the uncertainty about the reputation estimate is certainly larger for a provider having received very few ratings than for a provider having a large number of ratings. Introducing a prior distribution on the reputation scores allows to balance the a priori, subjective, opinion about the provider of interest and the evidence provided by the ratings.

![Figure 4.1: Summary of the various models: 1 for truncated ratings, 2 for dropping the variance term, 3 for fixing the internal quality score deterministically, and 4 for introducing prior distributions on the parameters.](image)

After having developed the different PMR models, we realized that the PMR models can be viewed as extensions of the Brockhoff & Skovgaard model [29] which was developed in the context of food quality and preference assessment. Brockhoff & Skovgaard’s model is in fact similar to a simplified PMR1 model. Indeed, it assumes deterministic providers that always provide the same quality level, \( q_k \), for the supplied items (while the PMR1 model assumes a stochastic provider). It further assumes that each consumer rates once

---

*We thank Professor Ritter for pointing us the relationships between the PMR models and Brockhoff & Skovgaard’s model.
4.2. Probabilistic models of reputation (PMR)

and only once each provider, as often considered in ANOVA models. Moreover, the estimation procedure proposed in [29] is different and is not based on the expectation-maximization algorithm.

A wide range of probabilistic reputation models has been investigated; the one that should be used depends, of course, of the problem at hand. Therefore, the main considerations that have to be taken into account for choosing the most relevant model are detailed in Section 4.2.6.

For all the models, the only observed data are the ratings; the other quantities being unobserved. A variant of the well-known Expectation-Maximization (EM) algorithm is used in order to estimate both the quality of service of each provider as well as the bias and the reactivity of each consumer. The estimated quality of service of the providers will be the suggested reputation score for the provider.

4.2.1 The basic model: PMR1

4.2.1.1 Description of the basic model

Assume we have a set of $n_x$ providers and $n_y$ consumers. Each provider (say provider number $k$) has a latent internal quality score, $q_k$, that is hidden to the external world. We define the reputation score associated to provider $k$ as the estimate of $q_k$ based on empirical data. Indeed, each time the provider $k$ sells an item (referred to transaction $i$; each sold item corresponds to a transaction) the internal quality of this item $x_{ki}$ is a random variable following a normal law centered on $q_k$. Thus, the quality of transaction $i$ generated by provider $k$ is given by

$$x_{ki} = q_k + \varepsilon_{x_{ki}}$$

(4.1)

where $x_{ki}$ is a realization of the random variable $x_k$, the noise random variable $\varepsilon_{x_k}$ (the superscript $x$ means that the noise model involves the provider) is normal centered and $\varepsilon_{x_{ki}}$ is a realization of this random variable appearing in transaction number $i$, $\varepsilon_{x_{ki}} \sim N(0, \sigma_{x_{ki}}^2)$. The total number of transactions is denoted by $N$. Therefore, each provider is characterized by two features, (i) his internal quality score $x_k$, and (ii) his stability in providing a constant quality $\sigma_{x_k}^2$.

On the other hand, the consumer $l$ who ordered the item rate it based on the inspection of its quality $x_{ki}$. Here, we assume that the consumer can be characterized by three different features: (i) his reactivity with respect to the quality of the provided item $a_l$, (ii) his bias $b_l$, and (iii) his stability in providing constant ratings for a fixed observed quality $\sigma_{x_{ki}}^2$. A linear regression model taking all these features into account is assumed. The rating provided by consumer $l$
4.2. Probabilistic models of reputation (PMR)

for transaction $i$ with provider $k$ is given by

$$y_{kli} = a_l x_{ki} + b_l + \varepsilon_{li}^y$$  \hspace{1cm} (4.2)

where the random variable $\varepsilon_{li}^y$ (the superscript $y$ means that the noise model involves the consumer) is normal centered and $\varepsilon_{li}^y$ is a realization of this random variable appearing in transaction number $i$, $\varepsilon_{li}^y \sim N(0, \sigma_{yli}^2)$. Since, as for a one-way analysis of variance, the $q_k$ are only defined up to an additive constant, we constrain the $q_k$ parameters to sum to zero, $\sum_{k=1}^{nx} q_k = 0$ [29]. The $q_k$ are therefore normalized.

The quantity $e_l = -b_l/a_l$ is often called the expectation of the consumer in marketing research [25]. The model described by Equation (4.2) can be re-expressed in terms of this expectation as

$$y_{kli} = a_l(x_{ki} - e_l) + \varepsilon_{li}^y$$  \hspace{1cm} (4.3)

which indicates that the rating provided by the consumer is directly proportional to the difference between the observed quality of the item $x_{ki}$ and his expectation $e_l$. The expectation of each consumer is an interesting feature that is provided as a by-product of the model.

Yet another interesting feature concerns the consumers, i.e., the raters who can be evaluated as well, as already proposed in [29]. A rater (or consumer) is considered as highly reliable if (i) his reactivity $a_l$ is close to 1 (he is fair), (ii) his bias $b_l$ is close to 0 (he is unbiased), and (iii) his standard deviation $\sigma_{yli}$ is close to 0 (he is consistent). Therefore, the reliability $r_l$ of a consumer/rater $l$ could, for instance, be evaluated by $r_l = [(a_l - 1)^2 + (b_l)^2 + (\sigma_{yli}^2)^2]^{1/2}$, corresponding to the expectation of the squared error of the provided ratings when the input is a iid standardized signal (independent, zero-mean and unit variance), but other choices are, of course, possible.

One can easily show that the joint probability of $[x_{ki}, y_{kli}]$ is also normal with mean and variance-covariance matrix

$$m = [q_k, a_l q_k + b_l] \text{ and } S = \begin{bmatrix} (\sigma_k^x)^2 & a_l (\sigma_k^y)^2 \\ a_l (\sigma_k^x)^2 & a_l^2 (\sigma_k^y)^2 + (\sigma_{yli}^2)^2 \end{bmatrix}$$  \hspace{1cm} (4.4)

Notice that this implies that the random variable $y_{kli}$ relative to provider $k$ is normally distributed $y_{kli} \sim N\left(a_l x_{ki} + b_l, \sqrt{a_l^2 (\sigma_k^x)^2 + (\sigma_{yli}^2)^2}\right)$, given that provider is $k$. In the first model, we suppose that only the ratings $y_{kli}$ are observed while the $x_{ki}$ are unobserved.
4.2. Probabilistic models of reputation (PMR)

4.2.1.2 The likelihood function of the model

We now consider the problem of estimating the different parameters of the model. For this purpose, the set of values is considered as incomplete. The complete set of variables is \( \{ x_{ki}, y_{kli} \}, k = 1 \ldots n_x, l = 1 \ldots n_y, i = 1 \ldots N \), and since only the \( \{ y_{kli} \}, k = 1 \ldots n_x, l = 1 \ldots n_y, i = 1 \ldots N \) are observed, all the other variables are considered as unobserved. Assuming independence between the observations, the likelihood for the complete (observed and unobserved) data is

\[
L(\Theta) = \prod_{k=1}^{n_x} \prod_{l=1}^{n_y} \prod_{i \in (k,l)} P(x_{ki}, y_{kli}) = \prod_{k=1}^{n_x} \prod_{l=1}^{n_y} \prod_{i \in (k,l)} P(y_{kli} \mid x_{ki})P(x_{ki})
\]

where \( \Theta \) is the vector containing all the parameters of the model, \( \Theta = \{ q_k, a_l, b_l, \sigma^x_k, \sigma^y_l \} \), and \((k, l)\) denotes the set of transactions involving provider \( k \) and consumer \( l \), with \( n_{kl} \) being the total number of transactions \( \in (k, l) \). Thus, in the previous equation, the product on \( i \) is taken on the set of \( n_{kl} \) transactions belonging to \((k, l)\). Taking the logarithm of both sides and denoting the log-likelihood by \( l = \log L \) provides

\[
l(\Theta) = \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \log(P(x_{ki})) + \log(P(y_{kli} \mid x_{ki})) \right\}
\]

\[
= -\sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \frac{1}{2} \left[ \frac{x_{ki} - q_k}{\sigma^x_k} \right]^2 + \frac{1}{2} \left[ \frac{y_{kli} - (a_l x_{ki} + b_l)}{\sigma^y_l} \right]^2 + \log(\sigma^x_k \sigma^y_l) + \log(2\pi) \right\}
\]

This likelihood function serves as basis for the parameters estimation algorithm.

4.2.1.3 Estimating the reputation scores

This section is based on Appendix F where we develop an EM-based algorithm (the so-called “one-step-late” algorithm ([65]; see also [115]) for the estimation of both the parameters of the model and the unobserved variables. We suppose that we observe the rating \( y_{kli} \) for each transaction. Notice that the estimates of the parameters of interest are denoted by a hat.

The estimate of the unobserved variable \( x_k \) depends on the consumer \( l \) (see Appendix F) and is therefore denoted by \( \hat{x}_{kl} = E[x_k \mid y_{kli}, \Theta] \), including a subscript \( l \) stressing the dependence on the consumer. The obtained update of the
estimate of the unobserved variable \( x_{kli} \) for transaction \( i \) is

\[
\hat{x}_{kli} \leftarrow \hat{q}_k + \frac{\hat{a}_l (\hat{\sigma}_k^x)^2}{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2} \left[ y_{kli} - (\hat{a}_l \hat{q}_k + \hat{b}_l) \right]
\]

(4.9)

\[
\hat{x}_{kli} \leftarrow 1 \frac{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2}{\hat{s}_{kl} \left[ (\hat{\sigma}_l^y)^2 \hat{q}_k + (\hat{\sigma}_k^x)^2 \hat{a}_l \left( y_{kli} - \hat{b}_l \right) \right]}
\]

(4.10)

where we define \( \hat{s}_{kl} = \hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2 \). The meaning of this updating formula is standard: for computing the update of \( \hat{x}_{kli} \) at time \( t \) (left-hand side), use the estimates at time \( t - 1 \) (right-hand side).

On the other hand, we obtain the following reestimation formulas for the parameters of the model. The total number of transactions involving provider \( k \) is denoted by \( n_{k} \bullet \) while \( n_{\bullet l} \) denotes the total number of transactions involving consumer \( l \). In order to simplify the notations, we define the following intermediate variable, also updated at each iteration,

\[
(\hat{\sigma}_{kl}^{xy})^2 = \frac{(\hat{\sigma}_k^x)^2 (\hat{\sigma}_l^y)^2}{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2} = \frac{(\hat{\sigma}_k^x)^2 (\hat{\sigma}_l^y)^2}{\hat{s}_{kl}}
\]

(4.12)

Let us first consider the initialization of the parameters. At the first iteration step \(( t = 0)\), the quality levels are initialized to the average score and normalized \((\sum_{k=1}^{n_p} \hat{q}_k = 0)\):

\[
\begin{align*}
\hat{q}_k &\leftarrow \frac{1}{n_{k} \bullet} \sum_{l=1}^{n_{\bullet}} \sum_{i \in (k,l)} y_{kli}, \text{ and normalize the } \hat{q}_k \\
\hat{\sigma}_k^x &\leftarrow 1, \hat{a}_l \leftarrow 1, \hat{b}_l \leftarrow 0, \hat{\sigma}_l^y \leftarrow 1
\end{align*}
\]

(4.13)

The parameters associated to the providers are then updated at each iteration \( t \):

\[
\hat{q}_k \leftarrow \frac{1}{n_{k} \bullet} \sum_{l=1}^{n_{\bullet}} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } \hat{q}_k
\]

(4.14)

\[
(\hat{\sigma}_k^x)^2 \leftarrow \frac{1}{n_{k} \bullet} \sum_{l=1}^{n_{\bullet}} \sum_{i \in (k,l)} \left[ (\hat{q}_k - \hat{x}_{kli})^2 + (\hat{\sigma}_{kl}^{xy})^2 \right]
\]

(4.15)
4.2. Probabilistic models of reputation (PMR)

For the parameters associated to the consumers, we obtain

$$\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} x_{kli} (y_{kli} - \hat{b}_l)}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left( \hat{x}_{kli}^2 + (\hat{\sigma}_{x|y}^{|y}_{kli})^2 \right)}$$

(4.16)

$$\hat{b}_l \leftarrow \frac{1}{n_x} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (y_{kli} - \hat{a}_l \hat{x}_{kli})$$

(4.17)

$$\hat{\sigma}_{y}^{|y} \leftarrow \frac{1}{n_x} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (y_{kli} - (\hat{a}_l \hat{x}_{kli} + \hat{b}_l))^2 + \hat{\sigma}_{x|y}^{|y}_{kli}^2 \right]$$

(4.18)

Iterating these equations until convergence provides an estimation of the quality of each provider as well as of the other parameters.

4.2.2 A more sophisticated model accounting for truncation: PMR2

4.2.2.1 Description of the model

As already stated before, ratings are often expressed on a limited scale. This model assume that the ratings are truncated in order to obtain a final rating $$z_{kli}$$ in the interval $$[-c, +c]$$. We assume, without lack of generality, that the ratings are normalized in the interval $$[-1, +1]$$, and thus $$0 \leq c \leq 1$$. In other words, only the truncated ratings $$z_{kli}$$ are observed while the $$y_{kli}$$ are unobserved. Therefore,

$$z_{kli} = \text{trunc}(y_{kli}, c)$$

(4.19)

where trunc is the truncation operator defined as

$$\text{trunc}(y, c) = \delta(y \geq 0) \min(y, c) + \delta(y < 0) \max(y, c)$$

(4.20)

The function $$\delta(y \geq 0)$$ is equal to 1 if the condition $$y \geq 0$$ is true and 0 otherwise. Thus, the truncation operator saturates the variable $$y$$ by constraining its range in the interval $$[-c, +c]$$.

This model thus considers that we directly observe the truncated ratings for the $$N$$ transactions, $$\{z_{kli}\}$$, and the objective is to estimate the quality of the providers based on these ratings. As before, this estimate, $$\hat{q}_k$$, will be the reputation score for provider $$k$$.  

111
4.2. Probabilistic models of reputation (PMR)

4.2.2.2 The likelihood function of the model

This section considers the problem of estimating the different parameters of the model. In this case, the complete set of variables is \( \{ x_{ki}, y_{kli}, z_{kli}; k = 1 \ldots n_x, l = 1 \ldots n_y, i = 1 \ldots N \} \), and since only the \( \{ z_{kli}; k = 1 \ldots n_x, l = 1 \ldots n_y, i = 1 \ldots N \} \) are observed, all the other variables are considered as unobserved. Assuming independence between the observations, the likelihood of the observations for the complete data is

\[
L(\Theta) = \prod_{k=1}^{n_x} \prod_{l=1}^{n_y} \prod_{i \in (k,l)} P(x_{ki}, y_{kli}, z_{kli})
\]

(4.21)

and the likelihood of the complete data has quite the same form as for the PMR1 model.

4.2.2.3 Estimating the reputation scores

A few notations are needed before stating the reestimation formulas. The standard normal distribution and the standard normal cumulative distribution function are denoted by

\[
\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \quad \text{and} \quad \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du
\]

(4.23)

For the update of the unobserved variable \( x_{ki} \), we obtain the same form as for model PMR1,

\[
\hat{x}_{kli} \leftarrow \frac{1}{\hat{s}_{kl}^2} \left[ (\hat{\sigma}_k^y)^2 \hat{q}_k + (\hat{\sigma}_l^x)^2 \hat{a}_l \left( \hat{y}_{kli} - \hat{b}_l \right) \right]
\]

(4.24)

with \( \hat{s}_{kl}^2 \) defined as

\[
\hat{s}_{kl}^2 = \hat{a}_l^2 (\hat{\sigma}_k^y)^2 + (\hat{\sigma}_l^x)^2
\]

(4.25)

The update of the unobserved variable \( y_{kli} \) depends on the observed value of the corresponding rating, \( z_{kli} \). Three cases have to be considered: \( z_{kli} = -c \), \( -c < z_{kli} < +c \) and \( z_{kli} = +c \) (for more details, see G).
4.2. Probabilistic models of reputation (PMR)

First case: $z_{kli} = -c$:

$$
\hat{y}_{kli} \leftarrow (\hat{a}_i \hat{q}_k + \hat{b}_l) + \hat{\sigma}_{kl} \lambda(\hat{\gamma}_{kl})
$$

$$
\hat{V}_{kl}^- \leftarrow \hat{\sigma}_{kl}^2 [1 + \hat{\gamma}_{kl} \lambda(\hat{\gamma}_{kl}) - \lambda^2(\hat{\gamma}_{kl})]
$$

with

$$
\begin{cases}
\hat{\gamma}_{kl} = -c - (\hat{a}_i \hat{q}_k + \hat{b}_l) \\
\lambda(\hat{\gamma}_{kl}) = -\frac{\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
\end{cases}
$$

Second case: $-c < z_{kli} < +c$:

$$
\hat{y}_{kli} \leftarrow z_{kli}
$$

Third case: $z_{kli} = +c$:

$$
\hat{y}_{kli} \leftarrow (\hat{a}_i \hat{q}_k + \hat{b}_l) + \hat{\sigma}_{kl} \lambda(\hat{\gamma}_{kl})
$$

$$
\hat{V}_{kl}^+ \leftarrow \hat{\sigma}_{kl}^2 [1 + \hat{\gamma}_{kl} \lambda(\hat{\gamma}_{kl}) - \lambda^2(\hat{\gamma}_{kl})]
$$

with

$$
\begin{cases}
\hat{\gamma}_{kl} = c - (\hat{a}_i \hat{q}_k + \hat{b}_l) \\
\lambda(\hat{\gamma}_{kl}) = \frac{\varphi(\hat{\gamma}_{kl})}{1 - \phi(\hat{\gamma}_{kl})}
\end{cases}
$$

and

$$
\hat{V}_{kli} = \delta(z_{kli} = -c) \hat{V}_{kl}^- + \delta(z_{kli} = +c) \hat{V}_{kl}^+
$$

At the first iteration step ($t = 0$), we initialize the parameters as before (see Equation (4.13)). For the parameters associated to the providers, we have

$$
\hat{\gamma}_k \leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } \hat{\gamma}_k
$$

$$
(\hat{\sigma}_k^x)^2 \leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left[ (\hat{\gamma}_k - \hat{x}_{kli})^2 + (\hat{\sigma}_{kl}^x)^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 \hat{V}_{kli}}{\hat{s}_{kl}^2} \right]
$$
And for the parameters associated to the consumers,

\[
\hat{a}_l \leftarrow -\frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli} \left( \hat{y}_{kli} - \hat{b}_l \right) + \frac{\hat{a}_l (\hat{\sigma}_x^2)^2 \hat{V}_{kli}}{\hat{s}_{kl}^2} \right]}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli}^2 + (\hat{\sigma}_x^2)^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_x^2)^4 \hat{V}_{kli}}{\hat{s}_{kl}^4} \right]}, \tag{4.36}
\]

\[
\hat{b}_l \leftarrow \frac{1}{n \cdot l} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{x}_{kli}) \tag{4.37}
\]

\[
(\hat{\sigma}_l^y)^2 \leftarrow -\frac{1}{n \cdot l} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (\hat{y}_{kli} - (\hat{a}_l \hat{x}_{kli} + \hat{b}_l))^2 + \hat{a}_l^2 (\hat{\sigma}_x^2)^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_x^2)^4 \hat{V}_{kli}}{\hat{s}_{kl}^4} \right] \tag{4.38}
\]

with \(\hat{\sigma}_{kl}^{x|y}\) defined as

\[
(\hat{\sigma}_{kl}^{x|y})^2 = \frac{(\hat{\sigma}_x^2)(\hat{\sigma}_y^2)^2}{\hat{\sigma}_x^2 + (\hat{\sigma}_y^2)^2} \tag{4.40}
\]

### 4.2.3 A simplified model with truncation: PMR3

For this simplified model, called **PMR3**, we drop the variance term, \(V\). In this case, the model corresponds to a simple two-stage procedure: (i) compute the conditional expectations of the \(y_{kl}\) given the observed \(z_{kl}\) and (ii) compute the estimates of the \(x_k\) as well as the parameters by considering that the conditional expectations of the \(y_{kl}\) are the real observed values as in PRM1. Thus, this simplified model is equivalent to PMR2 where we drop the variance term, \(V\) (i.e., we use the update equations of PMR2 with \(V = 0\)).

What concerns the unobserved variable \(x_{kli}\), we obtain the same updating rules as for models PMR1 and PMR2. For the unobserved \(y_{kli}\), we still have to consider three cases: \(z_{kli} = -c, -c < z_{kli} < +c\) and \(z_{kli} = +c\).

**First case**: \(z_{kli} = -c\):

\[
\hat{y}_{kli} \leftarrow (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{s}_{kl} \lambda(\hat{\gamma}_{kl})
\]

\[
\begin{align*}
\hat{\gamma}_{kl} & = -c - (\hat{a}_l \hat{q}_k + \hat{b}_l) \\
\lambda(\hat{\gamma}_{kl}) & = -\frac{\hat{s}_{kl}}{\hat{\phi}(\hat{\gamma}_{kl})}
\end{align*}
\tag{4.41}
\]
4.2. Probabilistic models of reputation (PMR)

Second case: \(-c < z_{kli} < +c:\)

\[ \hat{y}_{kli} \leftarrow z_{kli} \]  \hspace{1cm} (4.42)

Third case: \(z_{kli} = +c:\)

\begin{align*}
\hat{y}_{kli} & \leftarrow (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{s}_{kl} \lambda(\hat{\gamma}_{kl}) \\
\hat{\gamma}_{kl} & = \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{s}_{kl} \lambda(\hat{\gamma}_{kl})} \\
\hat{\lambda}(\hat{\gamma}_{kl}) & = \frac{\varphi(\hat{\gamma}_{kl})}{1 - \varphi(\hat{\gamma}_{kl})}
\end{align*}

\hspace{1cm} (4.43)

At the first iteration step \((t = 0)\), we initialize the parameters as before. Then, for the parameters associated to the providers, we have

\[ \hat{q}_k \leftarrow \frac{1}{n_k \bullet} \sum_{l=1}^{n_x} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } \hat{q}_k \]  \hspace{1cm} (4.44)

\[ (\hat{\sigma}_{x_k}^2)^2 \leftarrow \frac{1}{n_k \bullet} \sum_{l=1}^{n_x} \sum_{i \in (k,l)} \left[ (\hat{q}_k - \hat{x}_{kli})^2 + (\hat{\sigma}_{x_{kli}}^y)^2 \right] \]  \hspace{1cm} (4.45)

For the parameters associated to the consumers, we have

\[ \hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli} \left( \hat{y}_{kli} - \hat{b}_l \right) \right]}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \hat{x}_{kli}^2 + (\hat{\sigma}_{x_{kli}}^y)^2} \]  \hspace{1cm} (4.46)

\[ \hat{b}_l \leftarrow \frac{1}{n \bullet} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{x}_{kli}) \]  \hspace{1cm} (4.47)

\[ (\hat{\sigma}_{l}^y)^2 \leftarrow \frac{1}{n \bullet} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (\hat{y}_{kli} - (\hat{a}_l \hat{x}_{kli} + \hat{b}_l))^2 + \hat{a}_l^2 (\hat{\sigma}_{x_{kli}}^y)^2 \right] \]  \hspace{1cm} (4.48)

Notice finally that the model without truncation is simply obtained by replacing the unobserved variables \(\hat{y}_{kli}\) by the observed ones \(y_{kli}\).

4.2.4 A further simplified model with truncation: PMR4

A further simplified model with truncation, PMR4, can be obtained by assuming that the provider \(k\) always supplies the same quality \(q_k\) for the items (the quality
of the item is no more a random variable). This model simplifies greatly the previous model; in particular it reduces significantly the number of variables to be estimated since there is no need to introduce unobserved variables, \( \hat{x}_{kli} \). We expect this model to be particularly useful when the number of providers is greater than the number of consumers \((n_x > n_y)\), in which case the number of parameters to estimate would be too high.

There are essentially two main differences with the previous model: (1) the quality of the provided item is now deterministic and is equal to the latent quality score of the provider (in particular, this implies that the variable \( y_{kl} \) is now normally distributed with mean \( q_k \) and standard deviation \( \sigma_y^l \) (instead of \( s_{kl} \)), \( y_{kl} \sim N(a_lq_k + b_l, \sigma_y^l) \), which results in a different update of the estimated \( \hat{y}_{kli} \), and (2) the simplified expected log-likelihood function, after the expectation step and after neglecting the extra variance term coming from the truncation in PMR3, is now

\[
E_{y|z}[l | z, \hat{\Theta}] = -\frac{1}{2} \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in \{k,l\}} \left\{ \left[ \frac{\hat{y}_{kli} - (a_lq_k + b_l)}{\sigma_y^l} \right]^2 + \log(\sigma_y^l)^2 \right\}
\]

instead of Equation (H14).

It can be easily shown that the resulting update rules are the following. Let us first consider the three truncation cases: \( z_{kli} = -c \), \(-c < z_{kli} < +c \) and \( z_{kli} = +c \). \( s_{kl} \) is simply replaced by \( \sigma_y^l \) in the updating formulas of PMR3 (Equations (4.41)-(4.43)):

**First case**: \( z_{kli} = -c \):

\[
\hat{y}_{kli} \leftarrow (a_l\hat{q}_k + \hat{b}_l) + \frac{\sigma_y^l}{\sigma_y^l} \lambda(\hat{\gamma}_{kl})
\]

with

\[
\hat{\gamma}_{kl} = \frac{-c - (a_l\hat{q}_k + \hat{b}_l)}{\sigma_y^l} \quad \text{and} \quad \lambda(\hat{\gamma}_{kl}) = \frac{-\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
\]

**Second case**: \(-c < z_{kli} < +c \):

\[
\hat{y}_{kli} \leftarrow z_{kli}
\]
4.2. Probabilistic models of reputation (PMR)

Third case: $z_{kli} = +c$:

$$\hat{y}_{kli} \leftarrow (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{\sigma}_y \lambda(\hat{\gamma}_{kl})$$

with

$$\hat{\gamma}_{kl} = \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{\sigma}_y}$$

(4.52)

For the reputation parameters associated to the providers, when minimizing Equation (4.49) (M-step), we easily obtain

$$\hat{q}_k \leftarrow \frac{\sum_{i=1}^{n_y} \sum_{i \in (k,l)} (\hat{a}_l (\hat{y}_{kli} - \hat{b}_l)) / (\hat{\sigma}_y^2)}{\sum_{i=1}^{n_y} (n_{kl} \hat{a}_l^2) / (\hat{\sigma}_y^2)}$$, and normalize the $\hat{q}_k$ (4.53)

And for the parameters associated to the consumers,

$$\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{q}_k (\hat{y}_{kli} - \hat{b}_l))}{\sum_{k=1}^{n_x} n_{kl} \hat{q}_k^2}$$ (4.54)

$$\hat{b}_l \leftarrow \frac{1}{n_x} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{q}_k)$$ (4.55)

$$\hat{\sigma}_y^2 \leftarrow \frac{1}{n_x} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} [(\hat{y}_{kli} - (\hat{a}_l \hat{q}_k + \hat{b}_l))^2]$$ (4.56)

4.2.5 Introducing a Bayesian prior on the parameters: PMR5 and simple PMR5 (sPMR5)

Yet another extension consists in introducing a prior probability distribution on the reputation parameter $q_k$. This allows to regularize the estimate and to take the number of ratings into account when computing the reputation score. Indeed, the uncertainty about the reputation estimate is certainly larger for a provider having very few ratings than for a provider having a large number of ratings. Introducing a prior distribution on the $q_k$ allows to balance the a priori, subjective, opinion about the provider of interest and the evidence provided by the ratings.

We consider that the a priori reputation score is zero (a neutral rating), but this can be easily modified if some a priori information concerning the consumer is provided. In this case, the reputation score will be around zero at the beginning
4.2. Probabilistic models of reputation (PMR)

(no rating yet recorded) and will progressively deviate from zero when the number of ratings for this customer becomes more significant. The same approach is applied in order to regularize the parameters $a_k$ and $b_l$.

Concretely, we introduce a normal prior on $q_k$, $q_k \sim N(0, \sigma^q_0)$ where $\sigma^q_0$ is typically set in order to obtain a 0.99 probability of observing $q_k \in [-c, +c]$, in which case $\sigma^q_0 = c/2.57$. The normal distribution is the natural conjugate prior for the location parameter of a normal distribution [58]. This extension aims to maximize the a posteriori distribution of $q_k$, and this can be done by observing (see [115]) that the maximum a posteriori estimate of $\Theta$ maximizes $\log(P(\Theta|z)) = \log(P(z|\Theta)) + \log(P(\Theta)) + \log(P(z))$ where $z$ is the set of observed data. Since $P(z)$ does not depend on the parameter, this is equivalent to maximize $\log(P(\Theta|z)) = \log(P(z|\Theta)) + \log(P(\Theta))$.

The PMR5 model: PMR3 with priors. It can easily be shown that the expectation step remains the same as for the computation of the maximum likelihood [115]. On the other hand, the maximization step differs in that the objective function for the maximization process is augmented by the log prior density, $\log(P(\Theta))$. A few calculus shows that the update formula for $q_k$ in PMR3 becomes

$$\hat{q}_k \leftarrow \frac{\sum_{l=1}^{n_y} \sum_{i \in (k,l)} \hat{x}_{kli}}{n_{k•} + (\hat{\sigma}_k^x / \sigma^q_0)^2}, \text{ and normalize } \hat{q}_k \quad (4.57)$$

Of course, prior distributions could be assigned to the other parameters $a_l$ and $b_l$ as well by following the same procedure. Here are the resulting update rules for $a_l, b_l$ extending the PMR3 model,

$$\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli} \left( \hat{y}_{kli} - \hat{b}_l \right) \right] + (\hat{\sigma}_l^y / \sigma^a_0)^2}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \hat{x}_{kli}^2 + (\hat{\sigma}_l^y)^2} \quad (4.58)$$

$$\hat{b}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{x}_{kli})}{n_{l•} + (\hat{\sigma}_l^y / \sigma^a_0)^2} \quad (4.59)$$

where $\sigma^a_0$ and $\sigma^b_0$ are the prior standard deviations, assuming a normal distribution.

The other update equations are not modified. The PMR3 model extended with this Bayesian framework is called PMR5 model. It consists of the update rules of PMR3 with the update rule for $q_k$ being replaced by (4.57).
4.2. Probabilistic models of reputation (PMR)

The simple PMR5 model (sPMR5): PMR4 with priors. In exactly the same way, here is the adaptation of the simple PMR4 model:

\[
\hat{q}_k \leftarrow \frac{\sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left[ \hat{a}_l \left( \hat{y}_{kli} - \hat{b}_l \right) \right]}{\sum_{l=1}^{n_y} \left[ \frac{n_{kl} \hat{a}_l^2}{\hat{\sigma}_y^2 / \hat{\sigma}_0^2} \right] + 1}, \text{ and normalize the } \hat{q}_k \quad (4.60)
\]

\[
\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{q}_k \left( \hat{y}_{kli} - \hat{b}_l \right) \right] + \left( \hat{\sigma}_y / \hat{\sigma}_0 \right)^2}{\sum_{k=1}^{n_x} \left( n_{kl} \hat{a}_l^2 \right) + \left( \hat{\sigma}_y / \hat{\sigma}_0 \right)^2} \quad (4.61)
\]

\[
\hat{b}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{q}_k)}{n_{l*}} + \left( \hat{\sigma}_y / \hat{\sigma}_0^b \right)^2 \quad (4.62)
\]

4.2.6 Some guidelines on the use of the probabilistic reputation models

A wide range of probabilistic reputation models has been investigated in the previous sections. The one that should be used depends of course of the problem at hand. There are three main considerations that have to be taken into account for choosing the most relevant model:

- Should we consider a deterministic generation of the quality of the items?
- Are the ratings truncated or not?
- Does it make sense to regularize the parameters of interest, namely the internal quality, the bias, and the reactivity?

Concerning these alternatives, the guidelines are the following:

- Are there more providers than customers? If this is the case, a simplified model considering a deterministic generation of the quality of the items (such as PMR4 or sPMR5) should be used.
- Do the ratings involve truncation? If yes, a model accounting for truncation should be used (PMR2, PMR3, or PMR4).
4.3. Brockhoff and Iterative Refinement Models

- Are there very few ratings? In this case, a Bayesian prior on the parameters should be considered (PMR5 or sPMR5).

In any case, when dealing with truncation, we recommend the use of the simplified model which avoids the complexity of the full model without a significant drop in performance.

4.3 Brockhoff and Iterative Refinement Models

In this section, we present two models which are close to our probabilistic reputation models: Brockhoff and Iterative Refinement models. These models are also probabilistic models which directly model the statistical interaction between the consumers and the providers. Section 4.3.1 and 4.3.2 describes respectively the Brockhoff and Iterative Refinement models.

4.3.1 Brockhoff Model

In the context of food quality assessment, Brockhoff & Skovgaard [29] present a parametric model for sensory panel data where individuals (raters) evaluate different items. In their model, they assume that each rater evaluates each item through a linear regression of its quality. The models proposed in this chapter can be considered as an extension of the Brockhoff & Skovgaard model.

4.3.1.1 Description of the Model

Here, we present the adaptation of the Brockhoff model to our statement by considering \( n_y \) consumers rating \( n_x \) providers. As for our models, each provider \( k \) has a latent quality \( q_k \) and each consumer \( l \) can be characterized by three different features: (i) his reactivity with respect to the quality of the provided item \( a_l \), (ii) his bias \( b_l \), and (iii) his stability in providing constant ratings for a fixed observed quality \( \sigma_{y,l}^2 \). For each transaction \( i \), the consumer \( l \) provides a rating \( y_{kli} \). The parameters associated to the providers are then updated at each iteration \( t \):

\[
\hat{q}_k \leftarrow \frac{\sum_{l=1}^{n_y} \sum_{i \in (k,l)} \frac{\hat{a}_l (y_{kli} - \hat{b}_l)}{\sigma_{y,l}^2}}{\sum_{l=1}^{n_y} \frac{n_{kl}}{\sigma_{y,l}^2}}, \text{ and normalize the } \hat{q}_k \tag{4.63}
\]
4.3. Brockhoff and Iterative Refinement Models

For the parameters associated to the consumers, we obtain

$$\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_e} \sum_{i \in (k,l)} \left( \hat{q}_k \left( \hat{y}_{kli} - \hat{b}_l \right) \right)}{\sum_{k=1}^{n_e} \left( n_{kl} \hat{q}_k^2 \right)}$$  \hspace{1cm} (4.64)

$$\hat{b}_l \leftarrow \frac{1}{n_{xl}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left( \hat{y}_{kli} - \hat{a}_l \hat{q}_k \right)$$  \hspace{1cm} (4.65)

$$\left( \hat{\sigma}_y^2 \right) \left( l \right) \leftarrow \frac{1}{n_{xl}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{y}_{kli} - \left( \hat{a}_l \hat{q}_k + \hat{b}_l \right) \right]^2$$  \hspace{1cm} (4.66)

Iterating these equations until convergence provides an estimation of the quality of each provider as well as of the other parameters.

4.3.2 Iterative Refinement Models

Laureti et al. [102] propose a model for raters evaluating objects. Each object has a quality that is estimated by a weighted average of the ratings provided for this object. The weighting factor depends on the rater and is proportional to a reliability score of the rater, defined as the inverse of his variance.

4.3.2.1 Description of the Model

The Iterative Refinement model (IR) considers \( n_y \) consumers rating \( n_x \) providers. As for our models, each provider \( k \) has a latent quality \( q_k \) and each consumer \( l \) has a latent judging power \( \frac{1}{\hat{V}_l} \). For each transaction \( i \), the consumer \( l \) provides a rating \( y_{kli} \). The latent quality \( q_k \) of provider \( k \) is estimated by a weighted average of the received ratings

$$\hat{q}_k \leftarrow \frac{\sum_{l=0}^{n_y} \sum_{i \in (k,l)} \hat{w}_l \hat{y}_{kli}}{\sum_{l=0}^{n_y} n_{kl} \hat{w}_l}$$  \hspace{1cm} (4.67)

where the \( \hat{z}_{kli} \) are the observed ratings.

The so-called inverse judging power \( \hat{V}_l \) of consumer \( l \) is estimated by

$$\hat{V}_l \leftarrow \frac{\sum_{k=0}^{n_x} \sum_{i \in (k,l)} \left( \hat{z}_{kli} - \hat{q}_k \right)^2}{n_{xl}}$$  \hspace{1cm} (4.68)
4.4. Experiments

The unnormalized weights $\hat{w}_l$ take the general form

$$\hat{w}_l \leftarrow \frac{1}{\hat{V}_l^\beta}$$

(4.69)

with $\beta = 1$ corresponding to optimal weights, as explained in [102].

The IR algorithm solves Equations (4.67-4.69) through an iterative procedure:

1. Initialize $\hat{w}_l = 1/n_y$ for each consumer $l$;
2. Estimate $\hat{q}_k$ by Equation (4.67);
3. Estimate $\hat{V}_l$ by Equation (4.68);
4. Plug the estimated values in Equation (4.69) to compute the weights;
5. Repeat steps 2 to 4 until convergence.

4.4 Experiments

The experimental section aims to answer four important research questions: (1) Are the PMR models able to estimate the parameters of interest, namely the provider’s quality, the customer’s bias, and the customer’s reactivity, in an accurate way; (2) Do the suggested models compare favorably with respect to a simple average; (3) Do the suggested models provide better results than the Brockhoff Model (BKF) and Iterative Refinement (IR); (4) Which suggested model (PMR1, PMR2, PMR3, PMR4, PMR5, or sPMR5) provides the best results overall. In other words, does the PMR models show some added value in tasks involving reputation estimation. In order to investigate these questions, we performed four experiments that are now described.

4.4.1 Experimental settings

The experiments simulate a simple consumer-provider interaction model: a consumer is requesting a service to a provider while the provider has some score representing his internal quality of service. A transaction is the execution of a requested service by a provider for a consumer. The achievement of a transaction between a provider and a consumer brings the consumer to give a rating representing the quality of the provider in executing the service. Thus, the reputation
score of a provider for a transaction depends on one hand, on the service quality and on the other hand, on the behavior of the consumer.

In accordance with previous notations, a provider \( k \) is characterized by his internal quality score \( q_k \), and his stability in providing a constant quality \( \sigma_x^k \). A consumer \( l \) is characterized by his reactivity \( a_l \), his bias \( b_l \), and his stability in providing constant rates for a fixed observed quality \( \sigma_y^l \). These values are referred to as the parameters of the providers/consumers.

The experiments are performed on an artificial data set generated in order to simulate the interactions between \( n_x \) consumers and \( n_y \) providers. Each consumer-provider pair is connected through \( n_t \) links; each link representing a transaction \( i \) which is characterized by its quality \( x_{ki} \) depending on provider \( k \) and a rating \( y_{kli} \) provided by consumer \( l \). Each investigated model estimates the reputation score \( q_k \) and the stability \( \sigma_x^k \) of each provider \( k \), as well as the behavior \((a_l, b_l, \sigma_y^l)\) of each consumer \( l \) from a data set containing \( n_k, n_l, n_i \) transactions. The parameters are estimated on the available ratings only.

The first step aims to generate a set of consumers and providers having each their own parameters. The way the parameters are generated and the number of providers and consumers differ from one experiment to another and are explained later.

The second step consists of generating a set of providers-consumers transactions characterized by their quality \( x_{ki} \) and their rating \( y_{kli} \) for each triplet \((k, l, i)\). The quality and the ratings are generated from a normal distribution:

\[
\begin{align*}
  x_{ki} & = N(q_k, \sigma_x^k) \\
  y_{kli} & = N(a_l x_{ki} + b_l, \sigma_y^l)
\end{align*}
\]  
\(4.70\)

The ratings are then truncated in order to belong to the \([-1, 1]\) interval.

The aim of each model applied in the various experiments is of course to estimate at best the values of the parameters \{\( \hat{q}_k \), \( \hat{\sigma}_k \), \( \hat{\sigma}_y^l \), \( \hat{\sigma}_x^k \), and \( \hat{q}_k \)\} from the ratings only. The estimated values are compared to the real generated values in order to evaluate the ability of the model to retrieve the real, generated parameters’ value (which are hidden to the model). Two performance indicators within this context are reported: the average absolute error between real and predicted values and the linear correlation between real and predicted values.

The estimated parameters are initialized as follows:

\[
\hat{q}_k \leftarrow 0, \hat{\sigma}_k \leftarrow 1, \hat{a}_l \leftarrow 1, \hat{b}_l \leftarrow 0, \hat{\sigma}_y^l \leftarrow 1
\]  
\(4.71\)

Moreover, each experiment averaged the results on 10 runs, each run consisting of (1) generating actual parameters values and (2) estimating these values
4.4. Experiments

by using specific models.

Three experiments, obeying these settings, were conducted. The first experiment (Section 4.4.2) compares the three main models: the basic model PMR1, a more sophisticated model PMR2 introducing truncation and a simplified model with truncation, PMR3 (as an intermediary model between PMR1 and PMR2). The second experiment (Section 4.4.3) analyzes the behavior and the robustness of these models (PMR1, PMR2, and PMR3) with respect to noise. The third experiment (Section 4.4.4) compares the performance of our best models (according to the first experiment) with three other models of reputation, namely, a simple average, Brockhoff et al’s model (BKF), and the Iterative Refinement model (IR) which is described in this section. Finally, the last experiment (Section 4.4.5) executes and analyzes our models (i.e., PMR4 and sPMR5 providing the best results in the previous experiment) and BKF on a real data set.

4.4.2 First preliminary, experiment: Comparing PMR1, PMR2 and PMR3

4.4.2.1 Description

In this experiment, values for all parameters (related to provider $k$ and consumer $l$) have been uniformly generated within an interval given by:

$$
\begin{align*}
    a_l &\in [-0.15, 0.15] \\
    b_l &\in [-0.25, 0.25] \\
    \sigma^y_l, \sigma^x_k &\in [0.025, 0.25] \\
    q_k &\in [-0.5, 0.5]
\end{align*}
$$

Notice that the values of $a_l$ could be negative, corresponding to extreme conditions where the consumers are cheating about their ratings (i.e., allowing such behavior as providing a bad rating for a good item).

For this experiment, (1) the data set contains 50 consumers, 50 providers, and 20 transactions for each couple of provider-consumer (i.e., 50 $\times$ 50 $\times$ 20 transactions), and (2) a simple average (SA) as well as PMR1, PMR2, and PMR3 have been compared. Each model updates the parameters associated to the providers and the consumers, at each iteration, until convergence of $\hat{q}_k$. For SA, we simply average the ratings provided by the consumers as follows:

$$
\hat{q}_k \leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} y_{kli}
$$

124
4.4. Experiments

4.4.2. Results

<table>
<thead>
<tr>
<th>Simple average (SA)</th>
<th>PMR1</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Graph" /></td>
<td><img src="image2" alt="Graph" /></td>
</tr>
<tr>
<td><img src="image3" alt="Graph" /></td>
<td><img src="image4" alt="Graph" /></td>
</tr>
</tbody>
</table>

Figure 4.2: Comparison between the real (generated) and the predicted (estimated) reputation score represented by a dotted line, as computed by PMR1, PMR2, PMR3, and a simple average on the rating provided by the consumers.

Figure 4.2 compares the real (generated, but hidden to the algorithm) and the predicted (estimated) reputation score for SA, PMR1, PMR2, and PMR3. The closer the dotted line to the continuous line \( \hat{q} = q \) (predicted = generated), the best the prediction is (i.e., the model predicts well the actual value of the parameters). Figures 4.3 and 4.4 show the real and predicted values of the reactivity \( a_l \) and the bias \( b_l \) for PMR1, PMR2, and PMR3. The performance of the PMR models are very similar and the estimated values of the parameters are close to the actual values. Moreover, we clearly observe that PMR1-3 outperform SA.

The average absolute error as well as the linear correlation between the real and the predicted reputation scores \( (q_k) \), averaged on 10 runs are provided in Table 4.1 for PMR1, PMR2, PMR3, and SA, confirming that PMR models provide better estimation than SA. The best model is PMR2 followed by PMR3 and PMR1. In order to test the difference between these models, a \( t \)-test has been
4.4. Experiments

<table>
<thead>
<tr>
<th></th>
<th>PMR1</th>
<th>PMR2</th>
<th>PMR3</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average absolute error</td>
<td>0.085</td>
<td>0.082</td>
<td>0.084</td>
<td>0.276</td>
</tr>
<tr>
<td>Linear correlation</td>
<td>0.991</td>
<td>0.998</td>
<td>0.996</td>
<td>0.442</td>
</tr>
</tbody>
</table>

Table 4.1: Comparison of PMR1, PMR2, PMR3, and SA in terms of the average absolute error and the linear correlation.

performed, for the average absolute error, on the 10 runs, showing that the results of PMR2 are significantly \((p < 10^{-2})\) better than those provided by the other models.

Figure 4.3: Comparison between the given and the predicted consumer reactivity \(a_t\) represented by a dotted line, and computed by PMR1, PMR2, and PMR3.

4.4.3 Second experiment: Varying the stability \(\sigma^H_t\) of the consumers

4.4.3.1 Description

The second experiment analyzes the behavior of SA, PMR1, PMR2, and PMR3 when increasing the noise of the ratings, i.e., \(\sigma^H_t \in [0.25, 0.5]\). Thus, this experi-
4.4. Experiments

![Figure 4.4: Comparison between the given and the predicted consumer bias \( b_l \) represented by a dotted line, and computed by PMR1, PMR2, and PMR3.]

As in the first experiment, the linear correlation and the average absolute error between the real and the predicted scores have been averaged on 10 runs. These values are reported in Table 4.2. The best model is PMR2 followed by PMR3, PMR1, and finally by SA. A t-test confirms, for the average absolute error, the significant \( (p < 10^{-2}) \) differences between PMR2 and the other models (i.e., PMR2 outperforms the other models).

Since there is no substantial difference between PMR2 and PMR3, and since PMR3 is much simpler and less time-consuming than PMR2, we decided to only keep PMR3 for further investigations.
4.4. Experiments

<table>
<thead>
<tr>
<th></th>
<th>PMR1</th>
<th>PMR2</th>
<th>PMR3</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average absolute error</td>
<td>0.223</td>
<td>0.134</td>
<td>0.136</td>
<td>0.264</td>
</tr>
<tr>
<td>Linear correlation</td>
<td>0.585</td>
<td>0.974</td>
<td>0.957</td>
<td>0.465</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of PMR1, PMR2, PMR3, and SA in terms of the average absolute error and the linear correlation.

4.4.4 Third experiment: Comparisons with the Brockhoff and the Iterative Refinement models

4.4.4.1 Description

This experiment compares the behavior of SA, PMR3, PMR4, PMR5, and sPMR5 to two other probabilistic reputation models, the Brockhoff et al’s model [29] (BKF, as described in Section 4.3.1) and the Iterative Refinement model (IR) [102] (describes in Section 4.3.2).

For this experiment, 50 runs have been performed and, for each run, the settings, i.e., the number of providers \( n_x \), consumers \( n_y \) and transactions \( n_t \), are modified in the following way: Firstly, (i) we generate an original data set containing 20 providers, 20 consumers, and 20 transactions for each pair. Secondly, (ii) we extract 9 different data sets by sampling this original data set, selecting some of the consumers, providers, and transactions: 20 providers-20 consumers, 10 providers-10 consumers, 5 providers-5 consumers, and for each couple of provider-consumer a number of transactions equal to 20, 10, or 5. The results for each of these 9 settings are then averaged on 50 runs.

In order to compare the robustness of the models and to analyze their behavior in extreme environments, the values are uniformly generated with respect to the following conditions:

\[
\begin{align*}
    a_l & \in [-1, 2] \\
    b_l, q_k & \in [-1, 1] \\
    \sigma_l^y & \in [0.1, 1] \\
    \sigma_k^x & \in [0.05, 0.5]
\end{align*}
\]  

(4.74)

4.4.4.2 Results

Table 4.3 and Figures 4.5-4.7 display the average absolute errors between the actual and estimated values of the reputation score \( q_k \), the reactivity \( a_l \), and the bias \( b_l \) (remember that IR and SA models do not use any parameter representing
4.4. Experiments

<table>
<thead>
<tr>
<th></th>
<th>5 - 5</th>
<th>10 - 10</th>
<th>20 - 20</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMR3</td>
<td>0.50</td>
<td>0.28</td>
<td>0.15</td>
</tr>
<tr>
<td>PMR4</td>
<td>0.49</td>
<td>0.27</td>
<td>0.13</td>
</tr>
<tr>
<td>PMR5</td>
<td>0.50</td>
<td>0.28</td>
<td>0.15</td>
</tr>
<tr>
<td>sPMR5</td>
<td>0.49</td>
<td>0.27</td>
<td>0.13</td>
</tr>
<tr>
<td>BKF</td>
<td>0.51</td>
<td>0.30</td>
<td>0.18</td>
</tr>
<tr>
<td>IR</td>
<td>0.52</td>
<td>0.31</td>
<td>0.19</td>
</tr>
<tr>
<td>SA</td>
<td>0.57</td>
<td>0.36</td>
<td>0.23</td>
</tr>
</tbody>
</table>

Table 4.3: Absolute errors of $q_k$ by PMR3, PMR4, PMR5, sPMR5, BKF, IR, and SA when we increase the number of providers and consumers from 5-5, 10-10 until 20-20, the transactions number remaining at 20.

The reactivity and the bias, and are therefore not included in Figures 4.6 and 4.7). We only show the results for 20 transactions, the other settings leading to the same behaviors. On the bar plots, the black bar represents the 5 providers-5 consumers setting, the dark gray the 10 providers-10 consumers setting, and the light gray the 20 providers-20 consumers setting.

Figure 4.5 shows the absolute error for $q_k$. We clearly observe an influence of the provider and the consumer numbers on the absolute-error value: the more providers / consumers, the best estimations. Notice that this observation is also valid on the $a_l$ and $b_l$ results (see Figures 4.6 and 4.7).

Moreover, the smallest absolute-error values (i.e., the best estimations) for $q_k$ are provided, for the various settings, by PMR4 (0.49, 0.27, and 0.13) and sPMR5 (0.49, 0.27, and 0.13). The worse estimations of the score, are given by SA (0.57, 0.36, and 0.23), IR (0.52, 0.31, and 0.19), and BKF (0.51, 0.30, and 0.18), other PMR models providing in-between results.

The results for the absolute error for $a_l$ are shown in Figure 4.6. The smallest absolute-error values are, again, provided by PMR4 (0.32, 0.24, and 0.21) and sPMR5 (0.33, 0.25, and 0.22) for all the settings. The biggest errors are provided by BKF (0.49, 0.44, and 0.43).

When analyzing the absolute errors for $b_l$ provided by each model (see Figure 4.7), we observe that BKF (0.21, 0.18, and 0.17) provides the worst results.
4.4. Experiments

Figure 4.5: Absolute error of $q_k$ by PMR3, PMR4, PMR5, sPMR5, BKF, IR, and SA when the number of providers-consumers increase respectively from 5-5 (the left bar), 10-10 (the middle bar) until 20-20 (the right bar), with a fixed number of transactions (20) per consumer / provider pair.

while PMR models provide very close results (0.20 – 0.21, 0.11 – 0.12, and 0.05 – 0.06).

4.4.5 Fourth experiment: Application to a real data set

4.4.5.1 Description

This experiment compares the models PMR4, sPMR5, and BKF on a real data set containing the ratings of 12 professors on 99 students. Indeed, as an application of the proposed techniques, we decided to analyze the behavior of a set of professors teaching at the university of Louvain. For this purpose, the grades of 99 students (second-year students at the University of Louvain) were collected for their 12 courses.

More precisely, this data set includes, for each student, his grade for each of the 12 courses (with a total of 99×12 grades). In this framework, the students represent the suppliers of a service during course examination. On the other hand, the professor can be viewed as a consumer rating the student during the
4.4. Experiments

Figure 4.6: Absolute error of $a_l$ by PMR3, PMR4, PMR5, sPMR5, BKF, IR, and SA when the number of providers-consumers increase respectively from 5-5 (the left bar), 10-10 (the middle bar) until 20-20 (the right bar), with a fixed number of transactions (20) per consumer / provider pair.

examination. The aim of this experiment is to compute the expectations of each professor, as defined in Equation (4.3), and compare his expectation to the average grade and the average number of failure of each course. Remember that the expectations $E R_l$ of consumer (or professor) $l$ are given by (Section 2.1):

$$E R_l \leftarrow \frac{\hat{b}_l}{\hat{a}_l}.$$  \hfill (4.75)

We also decided to estimate the reliability of the professors. A professor is considered as highly reliable if (i) he is fair ($\hat{a}_l$ is close to 1), (ii) he is unbiased ($\hat{b}_l$ is close to 0), and (iii) he is consistent (his standard deviation $\hat{\sigma}_l$ is close to 0). Therefore, as already stated in Section 2.1, the reliability $r_l$ of a professor $l$ can be evaluated by:

$$r_l = [(\hat{a}_l - 1)^2 + (\hat{b}_l)^2 + (\hat{\sigma}_l)^2]^{1/2}. \hfill (4.76)$$

Both the expectation and the reliability will be reported.
4.4. Experiments

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Absolute error of $b_l$ by PMR3, PMR4, PMR5, sPMR5, BKF, IR, and SA when the number of providers-consumers increase respectively from 5-5 (the left bar), 10-10 (the middle bar) until 20-20 (the right bar), with a fixed number of transactions (20) per consumer / provider pair.}
\end{figure}

4.4.5.2 Results

The results show that the expectations of each professor, $\widehat{E}_l$, are indeed highly correlated with the course average: we obtain a linear correlation of $-0.930$, $-0.931$, and $-0.930$ for PMR4, sPMR5, and BKF respectively. The linear correlation coefficients between $\widehat{E}_l$ and the number of failures are 0.851, 0.778, and 0.776 for PMR4, sPMR5, and BKF respectively. Moreover, the results are also consistent with the “common knowledge” about the “difficulty” of each course. For instance, the fifth course (c5) is known to be a difficult one.

Figure 4.8 shows the reliability and the standardized expectation of each professor for the taught course. A professor with a high expectation means that he expects a lot from his students as a standard, i.e., a good exam. On the contrary, a professor with a low expectation means that his standards are lower.

What concerns the reliability, a professor with a reliability score close to 0 is highly reliable, meaning that his rating for a student examination is close to the intrinsic value of the student. It can be observed that the minimal reliability is
4.4. Experiments

quite high; this is due to the high variance of the students’ capabilities who show very different behaviors in function of the courses.

This analysis confirms that interesting information about the raters can be extracted from the ratings.

4.4.6 Discussion of the results

Let us now come back to our research questions. The experiments clearly show that (1) the PMR models provide a good approximation of the parameters of interest, even in adverse conditions; (2-3) the PMR models outperform IR, BKF, and SA; (4) the PMR models providing the best results overall are PMR4 and sPMR5, whatever the number of providers and consumers (i.e., augmenting the number of providers and consumers clearly decrease the absolute errors but does not change the ranking of the models). Moreover, we observe that PMR3, which is an easy-to-implement simplified version of PMR2, provides results very close to the ones of PMR2, therefore showing that the simplified version provides a good approximation of the original model. Thus, assuming that the provider $k$ always supplies the same quality $q_k$ for the items (remember Figure 4.1) leads to the two best models overall: PMR4 and sPMR5.
4.4. Experiments

Figure 4.8: Reliability and expectation for the courses (c1 to c12) taught by a professor.
Part IV
Service-Oriented Systems
Chapter 5

Service Center Architecture

Contents

5.1 Related Work ............................................. 139
5.2 Service Center Architecture ......................... 141
5.3 SPM: Process Model Generation ...................... 150
  5.3.1 From First-Order Logic to Disjunctive Normal Form 150
  5.3.2 Directed Acyclic Graph Generation ................. 151
  5.3.3 Illustrative Example .......................... 155
5.4 SM: Task Allocation Algorithm ....................... 156
  5.4.1 Randomized RL Algorithm ..................... 157
  5.4.2 Formulation of the Problem ................... 159
  5.4.3 Satisfying Hard Constraints .................. 160
  5.4.4 Computation of the Optimal TA Policy .......... 162
  5.4.5 Illustrative Examples ......................... 163
5.5 SR: Probabilistic Reputation Model ................... 166
  5.5.1 Formulation of the Problem ................... 167
  5.5.2 Computation of the TA Reputation Scores ....... 168
  5.5.3 Illustrative Examples ........................ 170
5.6 Logical Architecture .................................. 173
Motivations Managing the complexity of systems is currently considered a key challenge in computing (e.g., [175, 176]) and is being addressed through various approaches aimed at increased automation of system operation. Relying on agents as basic building blocks of complex systems seems appropriate, for their ideal characteristics involve autonomy, situatedness, and proactivity [80]. At a higher level of abstraction, service-oriented architectures (SOA) are expected to enable the provision of a large number of both distinct and competing services which the prospective users will be able to choose dynamically in the aim of receiving at all times optimal offerings for their purposes. While standardized interfaces facilitate economically efficient service provision, SOA must also be open, as both standardization and openness to various service providers are needed to avoid biased selection. Openness commits SOA to a distributed architecture for entering and leaving resources are bound to be decentralized. Finally, continuous optimization of service provision requires the monitoring of many criteria (including, e.g., Quality of Service (QoS) considerations) and, if automated, leads the system to exhibit the ability to self-organize and respond to changes in the environment without human involvement. Efficiency, flexibility, and adaptiveness that such systems are expected to exhibit are valuable given the pressing complexity concerns.

Constructing systems that exhibit the given characteristics involves many issues already treated to varying degrees in the literature: among them, technological infrastructure for services (e.g., [37]), description of service properties and behavior (e.g., [41]), matchmaking between service descriptions and requests (e.g., [18]), and so on. One particular question which has received limited treatment is the focus of the present chapter: Given a large number of heterogenous agents capable of executing specific tasks within (but themselves unable, or able to a limited extent to adapt to) a changing environment, what open and distributed MAS architecture can be argued to fit service-orientation while ensuring continuous optimal adaptation through dynamic task allocation and subsequent service provision that responds to QoS (and other) criteria?

Apart from the necessity to ensure appropriate description, interoperability, combination, and matchmaking, deploying open, distributed, service-oriented, and self-organizing MAS that allow unbiased service provision driven by multiple concerns and tailored to user expectations, places very specific requirements on the properties of MAS architectures that can support such systems: (1) A simple architecture would minimize the variety of interactions between the participating agents. Ensuring interoperability would thus not involve high cost on
5.1 Related Work

the providers. (2) Internal functions as task allocation, reputation computation, etc, ought remain outside the responsibility of the entering and leaving agents to avoid bias. The architecture must therefore integrate a special class of agents that coordinate service provision and that are internal to (i.e., do not leave) the system. (3) Since there is no guarantee that agents will execute tasks at performance levels advertised by the providers, reputation calculation and task allocation should be grounded in empirically observed agent performance and such observation be executed by internal agents. (4) Varying QoS requests and change in the availability of agents require task allocation to be automated and driven by QoS, reputation scores and other relevant considerations (e.g., deadlines). (5) To ensure continuous optimization of system operation, task allocation within the architecture should involve continuous observation of agent performance, the use of available information to account for agent reputation or behavior, and exploration of new options to avoid excessive reliance on prior information.

This chapter proposes a MAS architecture corresponding to the requirements 1–5 above. Part of this work is a joint work together with Jureta, Faulkner and Saerens, and has been published in [6, 86, 87].

The remainder of the chapter starts with Section 5.1 as related work and Section 5.2 as description and discussion of the architecture. Section 5.3 presents the service process modeler and the algorithm used to generate a graph representing the service process from the global goal of a service center expressed in first-order logic. Section 5.4 illustrates the role of the service mediator and details the RRL algorithm used for tasks allocation. The behavior of the service reputation and its probabilistic reputation model are discussed in Section 5.5. Finally, Section 5.6 presents the logical architecture of service center and the different layers composing this system.

5.1 Related Work

Regarding task allocation, closest to the present work is the generalization of the Semi-Markov Decision Process (SMDP) [172] model which provides a representation of the mediator decision problem. Abdallah and Lesser [1] formulate the mediator decision problem by extending the original SMDP formulation to account for randomly available actions and allow concurrent task execution. With regards to prior effort (e.g., [70]), they advance the matter by avoiding only serial task execution, homogenous agents, and deterministic action availability, while the reported experiments indicate their approach outperforms the origi-
In another paper, Abdallah and Lesser [2] suggest an algorithm for coordinating work between mediators: in a distributed architecture, mediators observe only part of what other mediators can observe, so that optimal task allocation across pooled agents can be represented as a game with incomplete information. While coordination across mediators is outside the scope of the present paper, it can be noted that the learning mechanism employed by the cited authors does not involve exploration, only exploitation. RRL can be argued to improve responsiveness of the MAS to new environment conditions because of exploration. The algorithm allows the execution of potentially complex processes while assuming that the set of available TS is changing. One distinctive characteristic the mediator’s behavior suggested in the present paper is that the algorithm accounts for a vector of criteria when allocating tasks, including QoS, service provision deadline, provision cost, explicit user preferences, and agent reputation. Maximilien and Singh [113] propose service selection driven by trust values assigned to service providing agents. Trust is extracted from user-generated reports of past service performance (as usual in reputation systems) over qualities defined by a system-specific QoS ontology. Level of trust depends on the degree to which reputation and quality levels advertised by the provider match. Similar approaches have been proposed, yet fail to address service selection in open, distributed MAS architecture, furthermore without dynamic allocation so that autonomic requirements are not fulfilled. By basing selection on trust only and generating levels of trust from advertised and user-observed behavior, Maximilien and Singh’s approach involves learning driven by exploitation of historical information, without exploration.

Tesauro and colleagues [176] present a decentralized MAS architecture for autonomic computing, in which dynamic task allocation proceeds as follows: A registry maintains the list of tasks that available agents can execute, so that when a service is requested, the registry is consulted and tasks are allocated to agents capable of fulfilling them. Allocation is utility-driven, whereby each resource has an associated and continually updated function which provides the value to the application environment of obtaining each possible level of the given resource. Information on the mediation process is very limited, seemingly indicating that no empirical data on actual agent performance is employed—i.e., it seems assumed that advertised behavior is the actual behavior, thus undermining the appropriateness of the given architecture for an open system. Shaheen Fatima and Wooldridge [54] suggest a MAS architecture in which permanent agents associated to the system are provided alongside agents that can enter and leave. Their focus is on minimizing the number of tasks that cannot be executed by the
5.2. Service Center Architecture

The Service Center Architecture (SCA) groups agents into Service Centers (SC). Each SC specializes in one particular service (e.g., finding the itinerary between two physical addresses as common in map applications, booking a flight, searching for files, and so on). The number of requests and type of services to provide is assumed unknown, so that service centers are formed at runtime. Within an SC, the request for providing the service is principally managed by a single agent, called the Service Mediator (SM). Assuming that a service is composed of tasks, the main role of the SM is to allocate tasks to agents. The SM does so by observing agent’s past behavior in task execution, then subsequently using (and updating) this information through the RRL, a reinforcement learning algorithm which combines such exploitation of experience with an optimal level of exploration to avoid excessive reliance on historical information (see Section 5.4.1). Tasks are dynamically allocated, i.e., at runtime. Agents participating in service execution can execute either a single or more than one distinct task (i.e., no particular limitation is imposed on the complexity of individual agents), while interacting with the other agents and with the mediator using a standard protocol (e.g., Contract-Net [167]). Since tasks are of interest, an agent is called a Task Specialist (TS) for each task it can execute. No constraints are placed on the origin of agents, so the architecture is neither closed nor centralized. As Figure 5.1 illustrates, several kinds of relationships (shown as links between agents and between mediators) are of particular interest within the given architecture. The free links (F-Link) represent the relationship between TS without task assignment and the service center within which their capabilities can be employed. When a task is
allocated to these TS, they enter the given SC and are considered occupied, so that the link becomes an O-Link (i.e., occupied link). The remaining two types of relationships arise from including the Support Center (SupC) within the SCA. Since a number of services involve the execution of common tasks, the presence of generic tasks (i.e., tasks whose frequency of execution is above some externally fixed threshold) makes it possible to further advance the architecture by pooling idle agents capable of executing the generic tasks into the same center. The effects sought in doing so are (i) ensuring the availability of agents which execute frequently needed tasks and (ii) having an SM dedicated to identifying and scheduling the work of TS which are most appropriate for generic tasks. Since it is unrealistic that SM in an SC perfectly observes the performance of agents in other SC, the SM in SupC can choose the most adequate such specialists from experience of SM in various SC. Therefore, whenever an SM in an SC encounters a generic task, the SM communicates the schedule for its execution to the service mediator in the SupC. The latter then determines whether a TS is available in the SupC to execute the task. If available, such TS links with an F-Link to the SC of the SM requesting the execution of the generic task. The support link (S-Link) between the SupC SM and individual TS in SupC differs from the O-Link in that occupation of the TS is governed only by SM in SC; the S-Link indicates that the TS communicate their availability to the SM in SupC. The remainder of this section revisits the SCA with more precision.

As the only property of the agent relevant for the SCA is the tasks it can execute along with its advertised quality values, the following definition will suffice. A tuple \( \langle \langle t_i, q_{at_i}^a \rangle, \ldots, \langle t_{i+m}, q_{at_{i+m}}^a \rangle \rangle \) is called an agent \( a \), where \( t_i \) is a task. The agent advertises its capability to execute the task to the QoS level and cost \( q_{at_i}^a \), this being specified by the provider of the agent. The advertised level is a vector of QoS- and cost-property and value pairs following a QoS ontology. \( Ag^a \) is assumed to contain all additional properties of the agent irrelevant for the present discussion, yet necessary when building a MAS.

Format and content of \( Ag^a \) will depend on the agent programming language being used. As it is unimportant for the service mediator to know that an agent can execute more than one task, a task specialist agent is defined over a single task.

A task specialist agent \( a^{TS}_i \) is \( \langle a, t_i, q_{at_i}^a \rangle \) associating a task \( t_i \) to a quality level \( q_{at_i}^a \) advertised by the agent \( a \). The agent must be capable of executing the task: \( \forall a^{TS}_i = \langle a, t_i, q_{at_i}^a \rangle, (t_i, q_{at_i}^a) \in a \).

Any agent \( a \) that can accomplish \( m > 1 \) tasks can be seen also as a set of task specialists: \( \{ a^{TS}_1, \ldots, a^{TS}_{1+m} \} \). Let \( C \) be the set of service centers, and \( t_A \) the
5.2. Service Center Architecture

A service mediator agent $a_{SM}^c$ in a service center $c \in C$ is an agent that can execute the RRL: $t_A \in a_{SM}^c$.

The principal difference between any $a_{SM}^c$ and the service mediator in the support center, $a_{SuppC}^SM$, is that the latter allocates generic tasks to task specialists. Providing this service is still subjected to the application of the algorithm, as such allocation can be defined as a service in which tasks are assignments of individual TS to service mediators in service centers, and this at particular times.

The service mediator agent $a_{SuppC}^SM$ in the support center SuppC is an agent that can execute the RRL: $t_A \in a_{SuppC}^SM$.

Figure 5.1: Service center MAS architecture.
5.2. Service Center Architecture

Before moving further in the definition of SCA, it is necessary to have a more precise idea of how tasks and services are conceptualized, as this influences the formulation of the algorithm in Section 5.4.1.

A task $t_i$ is $\langle t_{\text{pre}}^i, t_{\text{input}}^i, \tau_i, t_{\text{post}}^i, t_{\text{output}}^i \rangle$, where $t_{\text{pre}}^i$ describes the task precondition, $t_{\text{input}}^i$ is the set of the task input, $\tau_i$ is a specification (in some language) of how the agent is to execute the task, $t_{\text{post}}^i$ describes the conditions true after the task is executed, and $t_{\text{output}}^i$ is the set of output produces by the task. Tasks belong to the set $T$.

A service $s_j$ is defined as $\langle s_{\text{io}}^j, s_{\text{goal}}^j, s_{\text{const}}^j, pm_j \rangle$, where $s_{\text{io}}^j$ provides the details of the functional specification of the service, $s_{\text{goal}}^j$ is the global goal of the service in terms of first-order logic formulas, $s_{\text{const}}^j$ is the set of hard contraints of the service and $pm_j$ is the process model of the service.

The process model $pm_j$ of service $s_j$ is $\langle s_N^j, s_E^j, \text{servTransit}_j, \text{servState}_j \rangle$, where $(s_N^j, s_E^j)$ defines a directed acyclic graph. Nodes represent states, and edges transitions between states. The two functions label nodes and edges with task information: $\text{servTransit}_j : s_E^j \mapsto T$ is a partial function returning the task for a given edge in the graph, while $\text{servState}_j : s_N^j \mapsto \{ t_{\text{pre}}^i \}_{i \in T} \cup \{ t_{\text{post}}^i \}_{i \in T}$ maps each edge to a condition from the set of all task preconditions (i.e., $\{ t_{\text{pre}}^i \}_{i \in T}$) and postconditions (i.e., $\{ t_{\text{post}}^i \}_{i \in T}$). The task specified on an edge must have the precondition and postcondition corresponding to conditions given, respectively, on its origin and its destination node.

A service can therefore be understood as a process, composed of a set of tasks ordered over the graph representing the service. The functional specification of the service, i.e., $s_{\text{io}}^j$ is not of interest here, but involves in practice, e.g., a specification of interfaces, and other implementation considerations. Requesting a service requires the specification of expected QoS, in addition to a deadline for providing the service, minimal level of reputation for agents that are to participate in service execution, the maximal monetary cost, and explicit user preferences on agents to select (e.g., users may prefer globally the services of some providers over others, regardless of actual performance—this may occur with preferential treatment resulting from environment constraints such as, e.g., legal constrasts on cooperation between organizations and/or individuals).

A service request $\hat{s}_j$ is $\langle s_j, s_{\text{QoS}}^j, s_{\text{D}}^j, s_{\text{R}}^j, s_{\text{cost}}^j, s_{\text{pref}}^j \rangle$, where:

- $s_j$ is the service to provide.
- $s_{\text{QoS}}^j$ specifies expected qualities and their required level. Its definition follows an QoS ontology, such as, e.g., the FIPA QoS ontology specification...
5.2. Service Center Architecture

Whatever the specific QoS ontology, expected qualities are likely to be specified as (at least) \( s^QoS_j = (p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r) \), where:

- \( p_k \) is the name of the QoS parameter (e.g., connection delay, standards compliance, and so on).
- \( d_k \) gives the type of the parameter (e.g., nominal, ordinal, interval, ratio).
- \( v_k \) is the set of desired values of the parameter, or the constraint \(<, \leq, =, \geq, >\) on the value of the parameter.
- \( u_k \) is the unit of the property value.

- \( s^D_j \) is a deadline, specified as a natural.

- \( s^R_j = (\hat{q}_{a, t_i}, \ldots) \) specifies minimal levels of reputation over task quality parameters that any agent participating in the provision of the given service must satisfy. It is not necessary to specify reputation for all qualities over all tasks, selective reputation expectations are admitted.

- \( s^\text{cost}_j \) is the maximal monetary cost the user requesting the service is ready to pay to obtain the service.

- \( s^\text{pref}_j \) is a set of expressions that constrain the pool of potential agents to which the service mediator can allocate tasks.

Value \( \hat{q}_{a, t_i} \) represents the reputation of \( a^T_S \) specialized for the task \( t_i \) over the QoS parameters.

Reputation and trust receive considerable attention in the literature (e.g., [195, 113]). The ideas underlying Maximilien and Singh’s approach [113] are followed, with two caveats: they use “trust” to select services from a pool of competing services and exploit user-generated opinions to calculate reputation, whereas herein it is the TS that are selected (i.e., the concerns here are of a lower level), and since they are selected automatically, reputation is generated by comparing TS behavior observed by the SM and the advertised behavior of the TS. Reputation is used here instead of trust since no user opinions are accounted for. As the concerns are of the lower level, where a great number of TS can be available, automation seems appropriate for generating reputation levels. At this point, an extensive set of criteria for use in selecting among alternative task allocations is proposed along with the generic, underlying definitions of agent, task,
5.2. Service Center Architecture

A service center (SC) \( c \in C \) is defined as \( \langle IA_c, TS_{s,c}, AT_S^{\text{occ}}, AT_S^{\text{free}} \rangle \), where:

- \( IA_c = \langle a_{c}^{SM}, a_{c}^{SCD}, a_{c}^{SPM}, a_{c}^{SRM}, a_{c}^{SR} \rangle \); \( a_{c}^{SM} \) is the service mediator operating in the given center; \( a_{c}^{SCD} \) is the service center discoverer; \( a_{c}^{SPM} \) is the service process modeler; \( a_{c}^{SRM} \) is the service request manager; \( a_{c}^{SR} \) is the service reputation agent.

- \( TS_{s,c} = \{ t_i | \exists e \in s_j^{E}, \text{servTransit}_j(e) = t_i \text{ and } \text{assigned}(s_j, c) = \text{true} \} \) is the set of tasks that the service mediator needs to allocate among agents at any given time (the function \( \text{assigned} : \{ s_j \} \times C \rightarrow \{ \text{true}, \text{false} \} \) returns \text{true} if the service and center it takes as parameters are such that the service is to be provided by the given center, \text{false} otherwise).

- \( AT_S^{\text{occ}} = \{ a_{i}^{TS} | \text{occupiedLink}(a_{i}^{TS}, c) = \text{true} \} \) is the set of TS to which the SM allocated tasks to execute.

- \( AT_S^{\text{free}} = \{ a_{i}^{TS} | \text{freeLink}(a_{i}^{TS}, c) = \text{true} \} \) is the set of potential agents to which the SM can allocate tasks.

The MAS employing the SCA involves the set \( M \) of agents, decomposed into two distinct subsets at all times: a set \( I = \{ a_{c}^{SM}, a_{c}^{SCD}, a_{c}^{SPM}, a_{c}^{SRM}, a_{c}^{SR} \} c \in C \cup \{ a_{\text{SuppC}}^{SM}, a_{\text{SuppC}}^{SCD}, a_{\text{SuppC}}^{SPM}, a_{\text{SuppC}}^{SRM}, a_{\text{SuppC}}^{SR} \} \) of internal agents which never leave the system, and a set \( E = M \setminus I \) of environment agents that can enter and exit the MAS. Service mediators, service reputations, service center discoverers, service process modelers and service request managers are in \( I \), task specialists in \( E \).

A support center (SuppC) \( c_{\text{supp}} \) is a service center in which all tasks to allocate are generic, i.e., \( \langle IA_{c}, TS_{s,c}, AT_S^{\text{occ}}, AT_S^{\text{free}} \rangle \), with the additional constraint that for all \( t_i \in TS_{s,c} \), \( \text{genericTask}(t_i, P, x) = \text{true} \), where \( \text{genericTask} : T \times \text{timePeriod} \times N \rightarrow \{ \text{true}, \text{false} \} \) returns \text{true} if a given task has been executed over a given time period for more times than the specified threshold \( x \).

SCA is a set containing one support center and \( m \geq 1 \) service centers: \( SCA = \{ c_1, \ldots, c_n, c_{\text{supp}} \} \).

Figure 5.2 introduces a UML meta-model of a SC architecture and Figure 5.3 represents the internal architecture of a SC. In this architecture, each SC can achieve a global goal which influences the kind of tasks registered in this SC. A task can be executed by a set of TS, and for each task, we know its inputs,
5.2. Service Center Architecture

(outputs, pre-conditions, and post-conditions. For all tasks, these are written in the same language and using the same domain ontology (i.e., no interoperability problems). Pre/post-conditions are first-order logic formulas. The pre-condition describes conditions that need to hold in the state in which the service can execute. The post-conditions specify conditions that hold after the task finishes executing. Each SC has a Task Directory (TD) which is a repository referencing the tasks available in this SC. Each entry of TD is a Task Directory Entry (TDE) describing a task and all information about it.

In order to apply the system requirements defined at analysis stage (see Chapter 6), we define the concept of Service Level Agreement (SLA). A SLA is a contractual obligation between the service provider and the service consumer specifying mutually agreed understandings and expectations of the provision of a service [33, 192]. A SLA covers the functional side of the service (the provided service corresponds to the requested service in terms of input, output, pre and post-conditions) and concerns also the non-functional properties of the service [95]. The mediator is responsible to apply the defined SLA when allocating tasks to TS.)
5.2. Service Center Architecture

Five special agents are present in each SC, the service mediator (SM), the service center discoverer (SCD), the service process modeler (SPM), the service request manager (SRM) and the service reputation (SR). We have already defined the role of the service mediator.

The SCD has the role to explore the network in order to discover the other SC and SupC. Discovering other SC can be interesting to find new TS able to execute task not available in the corresponding SC and necessary to attain the global goal of this SC. Another advantage in the network exploration is the composition of a more complex service by merging the global goal (or some part of it) of different SC. Let $t_{NE}$ the task corresponding to the network exploration then:

A service center discovery agent $a_c^{SCD}$ in a service center $c \in C$ is an agent that can execute the network exploration: $t_{NE} \in a_c^{SCD}$.

The SPM is responsible of the creation and the updating of the process model. To create the process model, the SPM uses the tasks registered in the

Figure 5.3: Internal architecture of service center.
5.2. Service Center Architecture

TD, the hard constraints and the global goal of the SC, and it generates the different states, the transitions and the data’s on this model. When it generates the graph corresponding to the process model, it assigns to each task the corresponding set of TS able to execute this task, adding edges to the original graph. If this set is empty or under a defined threshold, SPM sends a search request to the SCD agent in order to find more TS in other SC or SupC.

A service process modeler agent $a^{SPM}_c$ in a service center $c \in C$ is an agent that creates and updates the process model $pm_j$ of service $s_j$ on the basis of the goal service $s^\text{goa}_j$, the service specifications (i.e., $s^\text{i}_j$ and $s^\text{const}_j$) and tasks $t_i \in T$ available in $c \in C$.

Another important agent of a SC is the service request manager. SRM is responsible to receive and manage a service request. It represents the public interface of a SC. A service request gives 3 things: (i) a formula which describes the goal state or a set of goal states (that is, the important thing to do is to achieve any state in which this formula is true), (ii) a set of hard constraints on quantitative QoS criteria and (iii) a QoS criterion to optimize. When a request is received by the SRM, it dispatches the request to the service mediator which allocates the different task to the given TS by optimizing the QoS criterion and by respecting the set of hard constraints as explained in Section 5.4. In the end of the allocation, the mediator uses the allocation list and requests the different TS in this list to execute the affected task in order to respond to the initial request. For a more complex service request, involving more than one SC, the mediator asks the SCD agent to find the SC able to responds to this request. At the end, the mediator merge the results and sends an integrated response to the SRM.

A service request manager agent $a^{SRM}_c$ in a service center $c \in C$ is an agent that can handle a service request $s_j$.

The last agent of a SC is the service reputation. SR is responsible to model and compute the reputation scores of all TS. To compute these reputations, the SR uses an extension of the reputation algorithms presented in Chapter 4.

A service reputation agent $a^{SR}_c$ in a service center $c \in C$ is an agent that can model and compute the reputations $q_{a,t_i}$ of all TS $a_i^{TS}$.

A SupC has the same architecture than a SC, except it has an additional special agent, the generic task discoverer (GTD). GTD has the role to find generic tasks in other SC and to retrieve them, and the corresponding TS executing these tasks, in the SupC. As reminder, a generic task is task frequently used by most of the SC.

The Communication Layer, presented in Figure 5.3, is responsible for network topology, for network exchange between all SC and SupC, and for all im-
5.3. SPM: Process Model Generation

Applications including the usage of the network. In the network, each SC and SupC has a unique identifier. In order to communicate with a remote SC or SupC, its identifier must be known. The set of Center (SC or SupC) known to another Center are called neighbours of that Center. Note that the concept of neighbourhood does not involve any distance metrics. The collection of neighbour sets defines the Center network that might be highly dynamic. For example, the communication layer may discover a new neighbour, or it may forget about a known Center if it is considered unreachable. Both the discovery and the removal processes are mediated by SCD.

5.3 SPM: Process Model Generation

5.3.1 From First-Order Logic to Disjunctive Normal Form

As explained in Section 5.2, each service has a global goal expressed in first-order logic. The generation of the process model as a directed acyclic graph describing a service and respecting its global goal is based on this first-order logic formulation. The transformation of these formulas into disjunctive normal form (DNF) facilitates the generation of the graph. Any first-order proposition can be put in DNF [126].

In Boolean logic, a disjunctive normal form is a standardization (or normalization) of a logical formula which is in the form of a disjunction of conjunctive clauses. This normal form is useful in automated theorem proving. A logical formula is considered to be in DNF if and only if it is a disjunction of one or more conjunctions of one or more literals. As in conjunctive normal form (CNF), the only propositional operators in DNF are and, or, and not. The not operator can only be used as part of a literal, which means that it can only precede a propositional variable. For example, all of the following formulas are in DNF:

\[
\begin{align*}
& (\alpha \lor \beta) \\
& \alpha \\
& (\alpha \land \beta) \lor \gamma \\
& (\neg \alpha \land \beta \land \neg \gamma) \lor (\neg \delta \land \mu \land \varphi)
\end{align*}
\]

However, the following formulas are not in DNF:

\[
\begin{align*}
& \neg(\alpha \lor \beta) \quad \text{Operator } \neg \text{ is the outermost operator} \\
& \gamma \lor (\delta \land (\alpha \lor \beta)) \quad \text{Operator } \lor \text{ is nested within an } \land
\end{align*}
\]
5.3. SPM: Process Model Generation

Converting a formula to DNF involves using logical equivalences, such as the double negative elimination, De Morgan’s laws, and the distributive law. These rules of conversions are illustrated by Table 5.1. Note that every logical proposition can be converted into disjunctive normal form [126].

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) & \text{Commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) & \text{Commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) & \text{Associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) & \text{Associativity of } \lor \\
\neg(\neg\alpha) & \equiv \alpha & \text{Double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\beta \Rightarrow \neg\alpha) & \text{Contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg\alpha \lor \beta) & \text{Implication elimination} \\
(\alpha \iff \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) & \text{Biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg\alpha \lor \neg\beta) & \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) & \text{Distributivity } \land \text{ of over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) & \text{Distributivity } \lor \text{ of over } \land
\end{align*}
\]

Table 5.1: Standard logical equivalences. The symbols \(\alpha\), \(\beta\) and \(\gamma\) stand for arbitrary sentences of propositional logic.

The following procedure converts first-order logic formula to a DNF:

1. Eliminate implications: convert \((\alpha \Rightarrow \beta)\) to \((\neg\alpha \lor \beta)\)
2. Move operator \(\neg\) inwards
3. Drop universal quantifiers
4. Distribute \(\land\) over \(\lor\)

5.3.2 Directed Acyclic Graph Generation

From a service global-goal expressed in DNF, a process model describing this service is generated and represented as a directed acyclic graph. The procedure used for the graph generation is explained in this section.

Algorithm of Table 5.2 is the principal function of this generation and its aim is to create a graph describing the process of a service \(s\). This function has two parameters as input: (i) \(globalGoal\) describing the global goal of service \(s\) and (ii) \(T\) representing the set of available tasks. As output, the function returns the graph of service \(s\).

The function starts by initializing the variable \(Nodes\) which contains the nodes set of the graph to be generated. The first node of this set is the starting node with which the process executing the service \(s\) starts. In this chapter,
5.3. SPM: Process Model Generation

Table 5.2: Algorithm generating a directed acyclic graph from the global goal of a service.

<table>
<thead>
<tr>
<th>Alg.1: graph ← function graphGeneration(globalGoal, T)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
</tr>
<tr>
<td>1.2</td>
</tr>
<tr>
<td>1.3</td>
</tr>
<tr>
<td>1.4</td>
</tr>
<tr>
<td>1.5</td>
</tr>
<tr>
<td>1.6</td>
</tr>
<tr>
<td>1.7</td>
</tr>
<tr>
<td>1.8</td>
</tr>
<tr>
<td>1.9</td>
</tr>
<tr>
<td>1.10</td>
</tr>
<tr>
<td>1.11</td>
</tr>
<tr>
<td>1.12</td>
</tr>
<tr>
<td>1.13</td>
</tr>
<tr>
<td>1.14</td>
</tr>
<tr>
<td>1.15</td>
</tr>
<tr>
<td>1.16</td>
</tr>
<tr>
<td>1.17</td>
</tr>
<tr>
<td>1.18</td>
</tr>
<tr>
<td>1.19</td>
</tr>
</tbody>
</table>

this node is described by an empty state (i.e., containing any goal). The second node of this set is the destination node and its state is described by the global goal of service \( s \). Indeed, when the process of service \( s \) reaches this destination node, the service global goal is satisfied and executed.

The second variable initialized by this function is \( T \) describing the transitions between the nodes of the graph to build. This variable is a set of elements of the form \(<\text{predecessor, successor, tasks}>\) where the first element is the starting node, the second element is the destination node and the last element is the set of tasks allowing the transition between the nodes \( \text{predecessor} \) and \( \text{successor} \) (the set of tasks having the precondition and the postcondition satisfied respectively by the nodes \( \text{predecessor} \) and \( \text{successor} \)). This set is initialized as \( \text{empty} \) at the beginning of the function execution.

Lines 1.7 to 1.10 decompose the global goal into a set of conjunctive clause. As remind, the global goal is expressed as DNF, i.e., a disjunction of conjunctive clause. For each conjunctive clause, a node is created and a sub-graph with this node as root is generated by function \( \text{goalToGraph} \). The function \( \text{goalToGraph} \) returns the sets \( \text{Nodes} \) and \( \text{Transitions} \) completed respectively by the nodes and the transitions created by this function. When all conjunctive clauses have been handled (at the end of Line 1.10), the sets \( \text{Nodes} \) and \( \text{Transitions} \) contain all nodes and transitions of the graph representing the service process. Line 1.11 creates the graph according to the nodes and transitions already
5.3. SPM: Process Model Generation

generated.

Table 5.3: The algorithm creates a sub-graph with node root as root.

The algorithm of Table 5.3 allows the creation of a sub-graph with node root as root. This function has four parameters as input: (i) root from which sub-graph is built, (ii) the set of tasks, \( \mathbb{T} \), allowing transitions between nodes of the sub-graph, the sets of (iii) nodes \( \text{Nodes} \) and of (iv) transitions \( \text{Transitions} \) already created. As result, the function returns \( \text{Nodes} \) and \( \text{Transitions} \) increased by the new nodes and transitions created by the function.

Line 2.6 gets the attribute goal which describes the state of node root as a conjunctive clause. Lines 2.7 to 2.11 expands root by deleting each term from its clause goal. This manipulation leads to a set of smaller clauses representing possible predecessor of root. For example, let \( \text{goal} = (t_1 \land t_2 \land t_3) \). After the
expansion of goal, we obtain three smaller clauses (or sub-goals): \((t_1 \land t_2)\), 
\((t_1 \land t_3)\) and \((t_2 \land t_3)\). These clauses represent possible predecessor of node root. At the end of Line 2.11, variable childNodes contains all direct predecessor of root.

Lines 2.12 to 2.21 check the predecessor nodes (or child nodes) obtained from the expansion of root and create the transitions between these predecessors and root. Line 2.13 seeks for possible tasks allowing transition between predecessors and root. If no transition is possible between a predecessor \(p\) and root, \(p\) is not saved as predecessor of root. In the other cases, \(p\) is saved as predecessor in Nodes (Line 2.16) and their transitions are registered in Transition (Line 2.18). Line 2.19 calls recursively the function with each predecessor (or child nodes) as root.

<table>
<thead>
<tr>
<th>Alg 3: childTasks←function findApplicableTasks(parentNode, childNode, (T))</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
</tr>
<tr>
<td>3.2</td>
</tr>
<tr>
<td>3.3</td>
</tr>
<tr>
<td>3.4</td>
</tr>
<tr>
<td>3.5</td>
</tr>
<tr>
<td>3.6</td>
</tr>
<tr>
<td>3.7</td>
</tr>
<tr>
<td>3.8</td>
</tr>
<tr>
<td>3.9</td>
</tr>
<tr>
<td>3.10</td>
</tr>
<tr>
<td>3.11</td>
</tr>
<tr>
<td>3.12</td>
</tr>
<tr>
<td>3.13</td>
</tr>
<tr>
<td>3.14</td>
</tr>
<tr>
<td>3.15</td>
</tr>
<tr>
<td>3.16</td>
</tr>
<tr>
<td>3.17</td>
</tr>
</tbody>
</table>

**Table 5.4:** The algorithm seeks tasks \(t \in \mathbb{T}\) allowing the transition between nodes childNode and parentNode.

Algorithm of Table 5.4 searches transitions between two nodes: a predecessor and a successor. The function has three parameters: (i) the successor node parentNode, (ii) the predecessor node childNode and (iii) the set of task \(T\) allowing the transitions. The function generates as results a sub-set of \(T\) containing all tasks allowing the transition between parentNode and childNode.

Line 3.8 calls checkTaskCond which is a function having three parameters: (i) a task \(t\), (ii) a predecessor node \(p\) and a successor node \(s\). This function checks the pre and postcondition of \(t\) with nodes \(p\) and \(s\) in order to see if \(t\) is executable.
in $p$ and gives node $s$ as resulting node.

### 5.3.3 Illustrative Example

![Diagram](image)

**Figure 5.4:** An illustrative example representing the generation of a directed acyclic graph generation from a global goal.

Figure 5.4 shows an illustrative example of a graph generation from a goal expressed in first order logic. In this example, the goal, $(\neg \alpha \land \beta \land \neg \gamma) \lor (\neg \delta \land \mu)$, is decomposed and translated in a directed acyclic graph by our algorithm.

Level 1 of the graph contains only the destination node $s_{10}$ noted by the goal $(\neg \alpha \land \beta \land \neg \gamma) \lor (\neg \delta \land \mu)$. This logical proposition is decomposed in two conjunctive clauses: $(\neg \alpha \land \beta \land \neg \gamma)$ and $(\neg \delta \land \mu)$. The algorithm generates a sub-graph from each conjunctive clauses and merges these sub-graphs in order to obtain the global graph.

Therefore, the first clause $(\neg \alpha \land \beta \land \neg \gamma)$ is decomposed as three sub-propositions: $(\beta \land \neg \gamma)$, $(\neg \alpha \land \neg \gamma)$ and $(\neg \alpha \land \beta)$. These sub-propositions are created from the original one by deleting one of the term. We create a node for each sub-proposition. Therefore, node $s_9$, $s_8$ and $s_7$ are annotated respectively by the propositions $(\beta \land \neg \gamma)$, $(\neg \alpha \land \neg \gamma)$ and $(\neg \alpha \land \beta)$. When the nodes are generated, we search the tasks able to make the transition between them (i.e., nodes
5.4. SM: Task Allocation Algorithm

$s_9$, $s_8$ and $s_7$) and the root (i.e., node $s_{10}$). In this example, we find a transition for the nodes $s_9$ and $s_7$, but no task is available for the transition between node $s_8$ and $s_{10}$.

From now, level 2 for the clause $(\neg \alpha \land \beta \land \neg \gamma)$ is finished. The next step is to take the clauses from each node of this second level (i.e., the nodes $s_9$, $s_8$ and $s_7$) in order to generate the next level by applying the same process (i.e., delete a term from the propositions, find the tasks, etc.) and stop the process when we reach a level containing an empty node representing the start node (i.e., node $s_1$ in this example) of the graph generated.

We apply the same process for the second conjunctive clause $(\neg \delta \land \mu)$ and we create the global graph from the nodes and the transitions obtained from this process.

5.4 SM: Task Allocation Algorithm

The SCA can be argued open and distributed, for it places no constraints on agents other than that task allocation is centralized at the mediator. The SCA is further service-oriented as the work of agents is organized according to services and service requests. The SCA cannot be argued adaptable and optimally responsive to service requests without the algorithm. The RRL presented in the sequel of this section defines the behavior of SM by specifying how the mediator-specific task, $t_A$, proceeds to allocate tasks to task specialists by optimizing one or more service request criteria (referred to as $r$ in the remainder), while taking the remaining criteria (vector $s$ containing all criteria from the service request other than $r$) as hard constraints. Returning to how a service request is specified in SCA (see Section 5.2), $\hat{s}_j = (s_j, s_j^{QoS}, s_j^D, s_j^R, s_j^{cost}, s_j^{pref})$, it is apparent that many criteria can be accounted for when selecting among alternative task allocations, hence qualifying the algorithm as multicriteria-driven within the present chapter. As decision making in presence of multiple criteria permits arguing for, and accepting various decision rules (which differ on, e.g., how criteria are aggregated), the algorithm is constructed to leave much freedom to the designer in actual implementation. Moreover, it is restrictive to require full specification of all possible criteria for each service—instead, it is up to the users to choose what criteria to specify. The algorithm thus optimizes a single normalized (i.e., taking values in the interval $[0, 1]$) variable, leading to three approaches to specifying this variable and the remaining hard constraints the algorithm takes as input when running:
5.4. SM: Task Allocation Algorithm

1. If the user prefers to have one criterion optimized (this being either a ratio-type* QoS parameter in $s^q_{QoS}^j$, or $s^d_j$, or reputation from $s^r_j$, or $s^{cost}_j$), expected values over the remaining criteria will be treated by the algorithm as hard constraints, whereby task allocations which violate hard constraints will be eliminated by the algorithm.

2. If the user prefers to have several criteria optimized, it is necessary to provide an aggregation function for the relevant criteria, so that the result of the function is what the algorithm will optimize (i.e., $r$ is an aggregate). Guidelines for aggregation functions can be found in, e.g., [184]. Non-aggregated criteria are treated as hard constraints (i.e., $s$).

3. A third option is to have the mediator suggest alternative allocations and the user chooses the one to apply. Presence or absence of this approach depends entirely on the choices of the MAS designer, as it does not affect the formulation of the algorithm—it is essentially the first option above, with the nuance that the user asks the mediator to provide a list of optimal allocations for each criteria, and then selects manually.

5.4.1 Randomized RL Algorithm

Whenever the environment is changing, new agents outperforming the actual ones can appear. The exploitation of acquired knowledge about the performance of TS can therefore be usefully combined with the exploration of allocation options arising with environment change. Formally, exploration is the association of a probability distribution to the set of available task specialists in each state (i.e., choice randomization). Usually, the exploration/exploitation issue is addressed by periodically readjusting the policy for choosing actions (here, such action consists of allocating a task to a task specialist) and re-exploring up-to-now suboptimal paths [119, 171]. Such a strategy is, however, suboptimal because it does not account for exploration. The RRL algorithm introduced in Chapter 2 is adapted herein to dynamic task allocation, allowing the assignment of tasks to TS while (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the performance of new agents so as to continually adjust task allocation, and (iv) exploring new options in task allocation. The exploration rate is quantified with the Shannon entropy associated to the probability distribution

---

*Nominal or ordinal QoS parameters that cannot be converted to ratio form give rise to hard constraints.
of allocating a task to a task specialist. This permits the continual measurement and control of exploration.

Returning to the formulation of the SCA, the problem the algorithm resolves is formulated as follows. The task-allocation problem amounts to the service mediator determining the task specialists to execute the tasks in a given service. By conceptualizing the service as a labeled directed acyclic graph in Section 5.2, the task-allocation problem amounts to a deterministic shortest-path problem in a directed weighted graph. The service is thus mapped onto a directed weighted graph \( G \) where each node in \( G \) is a step in service provision and an edge in \( G \) corresponds to the allocation of a task \( t_k \) to a task specialist agent \( a_{k,u}^{TS} \), where \( u \) ranges over task specialists that can execute \( t_k \) according to the criteria set in the service request. Each individual allocation of a task to a task specialist incurs a cost \( c(t_k, a_{k,u}^{TS}) \), whereby this “cost” is a function of the criterion (or aggregated criteria, as discussed previously in this section) formulated so that the minimization of cost corresponds to the optimization of the criterion (i.e., minimization or maximization of criterion value). This criterion is the one the user chooses to optimize, whereas other criteria are treated as hard constraints.

![Figure 5.5: The service provision problem as a labeled graph.](image)

For illustration, consider the representation of a generic service as a graph in Figure 5.5 where nodes are labeled with states of the service provision problem, and edges with costs of alternative task/TS allocations (for simplicity, only some labels are shown). Nodes are connected by several edges to indicate the presence of alternative allocations of the given task to task specialists. Any path from the starting node to the destination node in the graph is a potential complete allocation of tasks to task specialists. The service provision problem is thus a global...
optimization problem: learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints (specified in the service request). At the initial node in the graph (in Figure 5.5, node s1), no tasks are allocated, whereas when reaching the destination node (s13 in the same figure), all tasks are allocated.

The remainder of this Section is organised as follows: Section 5.4.2 introduces the notations, the standard deterministic shortest-path problem, and the management of continual exploration. Section 5.4.3 presents how to satisfy hard constraints by adopting a special graph structure. Section 5.4.4 details the unified framework integrating exploitation and exploration presented in Chapter 2 to deal with task allocation problem and describes our procedure for solving the deterministic shortest-path problem with continual exploration. Finally, Section 5.4.5 reports experimental evaluation and comparison of the algorithm with standard competing solutions.

5.4.2 Formulation of the Problem

We adapt herein the reinforcement learning framework defined in Chapter 2 to a task allocation problem. At a state $k$ of the service provision problem, choosing an allocation of $t_k$ to $a^{TS}_{k,u}$ (i.e., moving from $k$ to another state) from a set of potential allocations $U(k)$ incurs a cost $c(t_k, a^{TS}_{k,u})$. Cost is an inverse function of the criterion the user wishes to optimize (see Section 5.4), say $r$. The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [39]. It is by comparing TS over estimated $\hat{r}$ values and the hard constraints to satisfy (see Section 5.4.3) that task allocation proceeds. The allocation $(t_k, a^{TS}_{k,u})$ is chosen according to a Task Allocation policy (TA) $\Pi$ that maps every state $k$ to the set $U(k)$ of admissible allocations with a certain probability distribution $\pi_k(u)$, i.e., $U(k)$: $\Pi \equiv \{\pi_k(u), k = 1, 2, \ldots, n\}$. It is assumed that: (i) once the action has been chosen, the next state $k'$ is known deterministically, $k' = f_k(u)$ where $f$ is a one-to-one mapping from states and actions to a resulting state; (ii) different actions lead to different states; and (iii) as in [21], there is a special cost-free destination state; once the service mediator has reached that state, the service provision process is complete.

As explained in Section 2.2, exploration levels of service mediators can be controlled through exploration rates. Service provision then amounts to minimizing total expected cost $V_\pi(k_0)$ accumulated over all paths from the initial $k_0$ to the final state (see Equation 2.1).

159
5.4.3 Satisfying Hard Constraints

Hard constraints are satisfied by adopting a special graph structure and task allocation process, detailed in this section and inspired by critical path analysis (see for instance [15]). As shown in Figure 5.5, each node of the graph represents the completion of a task and each edge the assignment of a TS to the specific task. Each path from the starting node (e.g., node s1 in Figure 5.5) to the destination node (node s13 in Figure 5.5) thus corresponds to a sequence of assigned tasks ensuring the completion of the service within the prescribed hard constraints. The model thus assumes that there are alternative ways for completing the service. The topology of the graph—i.e., the node structure and the tasks associated to edges between the nodes—is provided by the designer through the service definition, so that the graph is a graphical model of the different ways the service can be performed as a sequence of tasks. Each constraint will be of the form “cannot exceed a given predefined quantity” (upper bounds); for instance, the total duration along any path should not exceed some predefined duration. Extensions to interval constraints could be handled as well, but are not reported in this thesis.

To illustrate task allocation while maintaining the hard constraints satisfied, let $g_{k_i}$ be the vector containing the largest values, for each quantity subject to a constraint, along any path connecting the starting node (called $k_0$) to node $k_i$, and $h_{k_i}$ the vector containing the largest values, for each quantity subject to a constraint, along any path connecting node $k_i$ to the destination node (called $k_d$). Further, let $s^{QoS}_{ij} = (s_1^j, s_2^j)$ be the vector containing hard constraints on two QoS criteria (for the sake of simplicity, two-dimensional criteria vectors are considered; extension to $n$-dimensional vectors is straightforward). It follows that $g_{k_i}$ represents the worst $s^{QoS}_{ij}$ when reaching $k_i$, while $h_{k_i}$ is the worst $s^{QoS}_{ij}$ for moving from $k_i$ to $k_d$. Computing the two vectors is straightforward in dynamic programming (e.g., [15]):

$$\begin{align*}
    g_{k_0} &= 0 \\
    g_{k_i} &= \max_{P(k_i) \rightarrow k_i} \left\{ s^{QoS}_{P(k_i) \rightarrow k_i} + g_{P(k_i)} \right\} \quad \text{(5.1)} \\
    h_{k_d} &= 0 \\
    h_{k_i} &= \max_{S(k_i) \rightarrow k_i} \left\{ s^{QoS}_{S(k_i) \rightarrow k_i} + h_{S(k_i)} \right\} \quad \text{(5.2)}
\end{align*}$$

where $P(k_i)$ is the set of predecessor nodes of $k_i$ and $S(k_i)$ the set of successor nodes of $k_i$. When computing $g_{k_i}$, the maximum is taken on the set of edges reaching $k_i$ (i.e., $P(k_i) \rightarrow k_i$); while when computing $h_{k_i}$, the maximum is
5.4. SM: Task Allocation Algorithm

taken on edges leaving \( k_i \) (i.e., \( k_i \rightarrow S(k_i) \)). \( s_j^{QoS} \) is the QoS criteria vector \((s_j^1, s_j^2)\) for a TS \( j \) associated to an edge. Any vector \( g_{k_i} < s_{max} \) and \( h_{k_i} < s_{max} \) is acceptable since it does not violate the constraints (assuming \( s_{max} = (s_{1, max}, s_{2, max}) \) contains upper bounds on hard constraints).

A possible approach using this heuristic is described as follow:

1. We compute the values \( g_{k_i} \) and \( h_{k_i} \) for each node \( k_i \) of the graph. If any change occurs in the hard constraints associated to the edges (e.g., an inclusion of new TS in the service provision, a TS leaving the service center, constraint’s values change ...), these values, \( g_{k_i} \) and \( h_{k_i} \), have to be reevaluated. Otherwise, this step hasn’t to be done for each received service request.

2. We select a sub-graph whose edges satisfy the hard constraints by applying the heuristic described above. To create this sub-graph, we first compute, for each edge of the graph, the value \( v_{ij} \) as the number of edge between node \( k_i \) and node \( k_j \). Secondly, we renumber the edges in a vector \( v \) on the basis of two criteria, respectively their \( v_{ij} \) value and their cost in term of hard constraints. Roughly speaking, the edge having the higher \( v_{ij} \) value (first criterion) and the higher cost (second criterion) is in the first position (index 0 of \( v \)) and the edge having the lower \( v_{ij} \) value and the lower cost in the last position (index \( n \) of \( v \)). By classifying the edges in the vector \( v \) in this way, we can delete in priority the edges which are not critical for the connected nodes (by not critical, we mean that other solutions (edges) exist between the connected nodes) and the most costly in term of hard constraints. In other words, we try to satisfy the hard constraints by keeping as long as possible all the nodes connected.

(a) If \( v \) is empty (i.e., all the edges have been selected), go to the step 3. If not, select the first edge from the vector \( v \) and retrieve this edge from \( v \).

i. Say the selected edge is between nodes \( k_i \) and \( k_j \) and associated to a TS with a vector \( s_j^{QoS} \) of QoS criteria. It is clear that the TS is eligible for the given request iff \( g_{k_i} + s_j^{QoS} + h_{k_j} < s_{max} \) (the inequality is taken elementwise). TS is deleted from the sub-graph if the inequality is not verified. If the edge is eligible, go to the step 2.(a) and select the next edge. Otherwise, continue to the next step 2.(a).ii.
5.4. SM: Task Allocation Algorithm

ii. Update $g_{ki}$ and $h_{ki}$ for each node influenced by the edge deletion of step 2.(a).i. Compute $v_{ij} \leftarrow v_{ij} - 1$, and update $v_{ij}$ for each edge in $v$ between node $k_i$ and node $k_j$. Reorder vector $v$ as in step 2. Return to the step 2.(a).

3. SM executes the task allocation algorithm on the selected sub-graph as explained in Section 5.4.4.

This method ensures the constraints are always satisfied along any path, i.e., for any assignment of TS to tasks in the generated sub-graph and provides a reasonable, although not optimal, solution to the problem. As explained in [9, 14], the complexity of this method is quite similar to a constraint shortest-path problem which is NP-complete.

5.4.4 Computation of the Optimal TA Policy

The SM begins with task allocation from the initial state and chooses from state $k$ the allocation of a TS $u$ to a task $t_i$ with a probability distribution $\pi_{ki}(u)$, which aims to exploration. The associated cost $c(t_i, a^TS_u)$ is incurred and is denoted for simplicity $c(k, i)$ (cost may vary over time in a dynamic environment); the MS then moves to the new state, $k'$. This allows the SM to update the estimates of the cost, $\hat{c}(k, i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no arc going backward (i.e. there exists no arc linking a state $k'$ to a state $k$ where $k' > k$), is as follows (a detailed treatment can be found in Chapter 2):

1. Initialization phase: Set $V(k_d) = 0$, which is the expected cost at the destination state.

2. Computation of the TA policy and the expected cost under exploration constraints: For $k_i = (k_d - 1)$ to the initial state $k_0$, compute:

$$\pi_{ki}(u) = \frac{\exp[-\theta_{ki}\left(c(k_i, u) + V(k'_i, u)\right)]}{\sum_{u' \in U(k_i)} \exp[-\theta_{ki}\left(c(k_i, u') + V(k'_i, u')\right)]},$$

$$V(k_i) = \sum_{u \in U(k_i)} \pi_{ki}(u) \left[ c(k_i, u) + V(k'_i, u) \right] \text{ for } k_i \neq k_d$$

where $k'_{i,u} = f_k(u)$ and $\theta_{ki}$ is set in order to respect the prescribed degree of entropy at each state (see Equation 2.4 which can be solved by a simple bisection search).
5.4. SM: Task Allocation Algorithm

One can show that this probability distribution law for task allocation minimizes the expected cost (see Equation 2.1) from the starting to the destination node for a fixed exploration rate (see Chapter 2).

Various approaches can be applied to update the estimated TS criterion \( \hat{r}_u \); e.g., exponential smoothing leads to:

\[
\hat{r}_u \leftarrow \alpha \bar{r}_u + (1 - \alpha)\hat{r}_u
\]

(5.4)

where \( \bar{r}_u \) is the observed value of the criterion for a TS and \( \alpha \in ]0, 1[ \) is the smoothing parameter. Alternatively, various stochastic approximation updating rules could also be used. The SM updates its estimates of the criterion each time a TS performs a task and the associated cost is updated accordingly.

5.4.5 Illustrative Examples

![Exploration rate of 20%](image)

**Figure 5.6:** Success rate in terms of run number, for an exploration rate of 20%, and for the five methods (no exploration, actual r known, \( \epsilon \)-greedy, naive Boltzmann, RRL).
5.4. SM: Task Allocation Algorithm

![Graph](image.png)

**Figure 5.7:** Success rate in terms of run number, for an exploration rate of 30%, and for the five methods (no exploration, actual $r$ known, $\epsilon$-greedy, naive Boltzmann, RRL).

### 5.4.5.1 Experimental setup

Task allocation for the service displayed in Figure 5.5 was performed. A total of three distinct task specialists were made available for each distinct task. Each $a_{k,u}^{TS}$ is characterized by its actual $r_u$ which is an indicator of the TS’s performance over the optimization criterion (see Section 5.4.2). In this simulation, it will simply be the probability of successfully performing the task (1 – probability of failure). In total, 57 task specialists are available to the SM for task allocation. For all TS $u$, $r_u$ takes its value $\in [0, 1]$; for 70% of the TS, the actual $r_u$ is hidden (assuming it is unknown to the SM) and its initial expected value, $\hat{r}_u$, is set, by default, to 0.3 (high probability of failure since the behavior of the TS has never been observed up to now), while actual $r_u$ value is available to the SM for the remaining 30% (assuming these TS are well known to the SM). Actual $r_u$ is randomly assigned from the interval $[0.5, 1.0]$ following a uniform probability distribution. It has been further assumed that $\hat{c}(t_i, a_u^{TS}) = -\ln(\hat{r}_u)$, meaning that it is the product of the $r_u$ along a path that is optimized (this is a standard
5.4. SM: Task Allocation Algorithm

![Figure 5.8: Average absolute difference between actual (r) and estimated (\( \hat{r} \)) criterion values in terms of run number, for three exploration methods (\( \epsilon \)-greedy, naive Boltzmann, RRL).](image)

measure of the reliability of a system). After all tasks are allocated, the selected TS execute their allocated tasks according to their actual \( r_u \) value (with failure \( 1 - r_u \)). The estimated TS criterion \( \hat{r}_u \) is then updated by exponential smoothing, according to Equation 5.4. In Equation 5.4, \( \hat{r}_u \) equals 1 if \( a^{TS}_u \) is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are of course updated in terms of the \( \hat{r}_u \) and each time a complete allocation occurs, the probability distributions of choosing a TS are updated according to Equation 5.3. 10,000 complete allocations were simulated for exploration rates 20\%, and 30\%.

5.4.5.2 Results

The RRL is compared to two other standard exploration methods, \( \epsilon \)-greedy and naive Boltzmann (see Chapter 2 for details), while tuning their parameters to ensure the same exploration level as for RRL. The success rate is defined as the proportion of services that are successfully completed (i.e., all tasks composing the service are allocated and executed successfully) and is displayed in Figures 5.6 and 5.7 in terms of the run number (one run corresponding to one complete
5.5 SR: Probabilistic Reputation Model

assignment of tasks, criterion estimation and probability distribution update). Figures 5.6 and 5.7 show the RRL behaves as expected. Its performances converge almost to the success rate of the RRL in which all actual $r$ are known from the outset (i.e., need not be estimated)—and indicate that exploration clearly helps by outperforming the allocation system without exploration (which has a constant 75% success rate). Figure 5.8 compares the three exploration methods by plotting the average absolute difference between actual $r_u$ and estimated $\hat{r}_u$ criterion values for a 30% exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new agents—i.e., exploration is useful for directing SM behavior in dynamic, changing, and open MAS, i.e., in the SCA.

5.5 SR: Probabilistic Reputation Model

The reputation algorithm introduced in Chapter 4 under the appellation of “simple probabilistic model of reputation” (sPMR5) is applied herein to service center, allowing to model and analyze the task specialists behaviors by computing their reputation scores. We extend the model proposed in Chapter 4 by adding a dynamic dimension, allowing the reputation scores to evolve over time.

It is assumed that TS usually supplies the task at the advertised QoS properties values according to its internal honesty value. After the execution of the service task, the client rates it according to its perception of respect to norms regulating the obligations of the TS agents. This rating is aggregated and scaled within a limited interval: $[-c, c]$, $c \in \mathbb{R}^+$. The client perception is a linear function of the achievement of the contract defined by norms between the client and the TS agents through the mediator. This perception is also subject to the bias of the service client in providing ratings as well as its reactivity towards changes in quality. The constancy of the service client to deliver a rating adjusted to the contracted quality level is quantified by the standard deviation of the normal law of ratings generation.

To regularize the estimate and to take the number of ratings into account when computing the reputation score of the TS agents, we use Bayesian priors. These priors are defined as a probability distribution on the reputation parameters and on the parameters characterizing the service client. The use of such priors is explained by the uncertainty about the reputation estimate. This reputation uncertainty is larger for a TS having received very few ratings than for a TS having a large number of ratings. Introducing a prior distribution on the reputa-
5.5. SR: Probabilistic Reputation Model

...tion scores allows to balance the a priori, subjective, opinion about the provider of interest and the evidence provided by the ratings.

The behavior of a TS also depends of the elapsed time, i.e., a TS not much honest in the past can change its behavior and become more honest. Evaluation are then time-dependent. To reflect this behavior, we use a damping factor that adapt the effect of an evaluation to the time elapsed in the calculation of the reputation. So, recent evaluations are more important than older ones in the reputation model.

5.5.1 Formulation of the Problem

Assume we have a set of \( n_x \) TS and \( n_y \) clients. Each TS \( a^{TS}_k \) (noted \( k \) in the sequel for simplicity) has a latent reputation value (i.e., its honesty) \( q_{a,tk} \) (noted \( q_k \) for simplicity). This value \( q_k \) is its ability to provide QoS advertised in the contracts set with service clients. This internal honesty value is hidden to the external world. Since, as for a one-way analysis of variance, the \( q_k \) are only defined up to an additive constant, we constrain the \( q_k \) parameters to sum to zero, \( \sum_{k=1}^{n_x} q_k = 0 \) [29]. The \( q_k \) are therefore normalized.

On the other hand, the service client \( l \) rates the task based on its conformance to norms of the contract. The client is characterized by three different features: (i) his reactivity with respect to the quality level provided, \( a_l \), (ii) his bias, \( b_l \), and (iii) his stability in providing constant ratings for a constant observed quality, \( \sigma^y_{yl} \).

A linear regression model taking all these features into account is assumed. The rating given by service client \( l \) for transaction (task execution) \( i \) with TS \( k \) is given by:

\[
y_{kli} = a_l q_k + b_l + \varepsilon^y_{li}
\]

where the random variable \( \varepsilon^y_{li} \) (the superscript \( y \) means that the noise model involves the consumer) is normal centered and \( \varepsilon^y_{li} \) is a realization of this random variable appearing in transaction \( i \): \( \varepsilon^y_{li} \sim N(0, \sigma^y_{yl}) \).

As stated before, ratings are often expressed on a limited scale. The model further assumes that the ratings are truncated in order to obtain a final rating \( z_{kli} \) given by the client in the interval \([-c, c]\). The ratings are then normalized in the interval \([-c, c]\). We only observe the truncated ratings \( z_{kli} \) for the \( N \) transactions. The \( y_{kli} \) are unobserved. Therefore: \( z_{kli} = trunc(y_{kli}, c) \) where \( trunc \) is the truncation operator defined as \( trunc(y, c) = \delta(y \geq 0) \min(y, c) + \delta(y < 0) \max(y, -c) \).

The function \( \delta(y \geq 0) \) is equal to 1 if the condition \( y \geq 0 \) is true and 0 otherwise. Thus, the truncation operator saturates the variable \( y \) by constraining its range to the interval \([-c, c]\). This model considers that we directly observe the truncated ratings for the \( N \) transactions, \( z_{kli} \), and the objective is to estimate...
the quality of the providers based on these ratings.

We state that, in this model, the importance given to an estimation is time dependent. Its importance decreases with the elapsed time. To include this hypothesis, we include an additional parameter, the damping factor: $\alpha_{kli}$. $\alpha_{kli}$ is situated within $[0, 1]$ and is the weight of the transaction $i$ occurring between the TS $k$ and the client $i$ occurring at time $t_{kli}$. As already mentioned, the parameters $\gamma_{kli}, \eta_k, \hat{a}_l, \hat{b}_l, \sigma^y_l$ are estimated based on the observed ratings by an Expectation Maximization like procedure optimizing the likelihood, see Chapter 4 for details.

We consider that the a priori reputation score is zero (a neutral rating), but this can be easily modified if some a priori information concerning the consumer is provided. In this case, the reputation score will be around zero at the beginning (no rating yet recorded) and will progressively deviate from zero when the number of ratings for this customer becomes more significant. The same approach is applied in order to regularize the parameters $a_l$ and $b_l$.

Concretely, we introduce a normal prior on $q_k$, $q_k \sim N(0, \sigma^q_0)$ where $\sigma^q_0$ is typically set in order to obtain a 0.99 probability of observing $q_k \in [-c, +c]$, in which case $\sigma^q_0 = c/2.57$. The normal distribution is the natural conjugate prior for the location parameter of a normal distribution [58]. This extension aims to maximize the a posteriori distribution of $q_k$. A detailed treatment, can be found in Chapter 4.

### 5.5.2 Computation of the TA Reputation Scores

The estimation of the reputation scores by the Service Reputation (SR) is divided into three steps. The first step is the initialization of the different parameters to estimate. The second step consists in the estimation of the ratings before truncation. The last step is the computation of the reputation score $\hat{q}_k$ and the other parameters (i.e., $\hat{a}_l, \hat{b}_l$ and $\hat{\sigma}^y_l$). Iterating the second and third steps until convergence provides an estimation of the reputation of the TS as well as of the other parameters.

#### 5.5.2.1 Initialisation phase:

Let us first consider the initialization of the parameters. At the initialization step (iteration $t = 0$), the quality levels are initialized to the average score and
5.5. SR: Probabilistic Reputation Model

normalized (\(\sum_{k=1}^{n_k} \hat{q}_k = 0\)): 

\[
\begin{align*}
\hat{q}_k &\leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} y_{kli}, \text{ and normalize the } \hat{q}_k \\
\sigma_q^0 &\leftarrow c/2.57 \\
\hat{a}_l &\leftarrow 1, \hat{b}_l \leftarrow 0, \hat{\sigma}_y^l \leftarrow 1
\end{align*}
\]  \hspace{1cm} (5.5)

5.5.2.2 Estimation of ratings before truncation phase:

First, we introduce some useful notations before stating the truncation formulas. The standard normal distribution (\(\varphi(x)\)) and the standard normal cumulative distribution function (\(\phi(x)\)) are denoted by:

\[\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}\] and \[\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du\]  \hspace{1cm} (5.6)

We now consider the three truncation cases: \(z_{kli} = -c, -c < z_{kli} < +c\) and \(z_{kli} = +c\).

**First case**: \(z_{kli} = -c\):

\[
\hat{y}_{kli} \leftarrow (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{\sigma}_y^l \lambda(\hat{\gamma}_{kl})
\]

with 

\[
\begin{align*}
\hat{\gamma}_{kl} &= -c - (\hat{a}_l \hat{q}_k + \hat{b}_l) \\
\lambda(\hat{\gamma}_{kl}) &= -\frac{\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
\end{align*}
\]  \hspace{1cm} (5.7)

**Second case**: \(-c < z_{kli} < +c\):

\[
\hat{y}_{kli} \leftarrow z_{kli}
\]  \hspace{1cm} (5.8)

**Third case**: \(z_{kli} = +c\):

\[
\hat{y}_{kli} \leftarrow (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{\sigma}_y^l \lambda(\hat{\gamma}_{kl})
\]

with 

\[
\begin{align*}
\hat{\gamma}_{kl} &= \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{\sigma}_y^l} \\
\lambda(\hat{\gamma}_{kl}) &= \frac{\varphi(\hat{\gamma}_{kl})}{1 - \phi(\hat{\gamma}_{kl})}
\end{align*}
\]  \hspace{1cm} (5.9)
5.5.2.3 Reputation computation phase

The reputation score associated to the TS agents are then updated at each iteration $t$:

$$\hat{q}_k \leftarrow \frac{\sum_{l=1}^{n_y} \sum_{i \in (k,l)} \alpha_{t_{ki}} \left[ \hat{a}_l \left( \hat{y}_{kli} - \hat{b}_l \right) \right]}{\left( \hat{\sigma}_{l} / \sigma_0^y \right)^2} + 1$$

(5.10)

For the parameters associated to the clients, we obtain:

$$\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \alpha_{t_{ki}} \left[ \hat{q}_k \left( \hat{y}_{kli} - \hat{b}_l \right) \right]}{\sum_{k=1}^{n_x} \left( \sum_{i \in (k,l)} \alpha_{t_{ki}} \hat{q}_k^2 \right) + \left( \hat{\sigma}_{l} / \sigma_0^a \right)^2}$$

(5.11)

$$\hat{b}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \alpha_{t_{ki}} \left( \hat{y}_{kli} - \hat{a}_l \hat{q}_k \right)}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \alpha_{t_{ki}} + \left( \hat{\sigma}_{l} / \sigma_0^b \right)^2}$$

(5.12)

$$\left( \hat{\sigma}_{l}^y \right)^2 \leftarrow \frac{\sum_{k=1}^{n_y} \sum_{i \in (k,l)} \left[ \alpha_{t_{ki}} \left( \hat{y}_{kli} - \left( \hat{a}_l \hat{q}_k + \hat{b}_l \right) \right)^2 \right]}{\sum_{k=1}^{n_y} \sum_{i \in (k,l)} \alpha_{t_{ki}}}$$

(5.13)

5.5.3 Illustrative Examples

5.5.3.1 Experimental setup

The reputation model is illustrated by a simple experiment. The experiment is performed on an artificial data set generated in order to simulate the interactions between $n_x$ consumers and $n_y$ TS. Here, a transaction is the execution of a requested task by a TS for a consumer. Each consumer-provider pair is connected through $n_i$ links; each link representing a transaction $i$ which is characterized by its quality $x_{ki}$ depending on TS $k$ and a rating $y_{kli}$ provided by client $l$. Thus, the reputation score of a TS for a transaction depends, on one hand, on the task quality and, on the other hand, on the behavior of the consumer. The investigated model estimates the reputation score $q_k$ and the stability $\sigma_k^y$ of each TS $k$, as well as the behavior $(a_l, b_l, \sigma_l^y)$ of each consumer $l$ from a data set containing...
transactions. The parameters are estimated on the available, observed, ratings only.

The idea is to generate some TS/clients as well as their features and test the ability of the algorithm to retrieve the values of the reputation scores based on the observation of the ratings only.

The first step aims to generate a set of consumers and TS having each their own parameters. The values for all parameters (related to TS \( k \) and consumer \( l \)) have been uniformly generated within an interval given by:

\[
\begin{align*}
  a_l & \in [-1, 2] \\
  b_l & \in [-1, 1] \\
  \sigma^y_l, \sigma^x_k & \in [0.05, 0.3] \\
  q_k & \in [-1, 1]
\end{align*}
\]

(5.14)

Notice that the values of \( a_l \) could be negative, corresponding to extreme conditions where the consumers are cheating about their ratings (i.e., allowing such behavior as providing a bad rating for a good item).

For this experiment, the data set contains 10 consumers, 10 TS, and 200 transactions for each couple of TS-consumer (i.e., \( 10 \times 10 \times 200 \) transactions). The model updates the parameters associated to the TS and the consumers, at each iteration, until convergence of \( \hat{q}_k \).

The second step consists of generating a set of TS-consumers transactions characterized by their quality \( x_{ki} \) and their rating \( y_{kli} \) for each triplet \((k, l, i)\). The quality and the ratings are generated from a normal distribution:

\[
\begin{align*}
  x_{ki} &= N(q_k, \sigma^x_k) \\
  y_{kli} &= N(a_l x_{ki} + b_l, \sigma^y_l)
\end{align*}
\]

(5.15)

The ratings are then truncated in order to belong to the \([-1, 1]\) interval.

The main objective of this experiment is to analyze the effect of various damping factors in a dynamic environment where TS have a quality of service changing over time. In other words, we study the performance and the robustness of the model, using different values of the damping factor, and its ability to estimate reputation scores of TS having a quality of task that fluctuates over time. To create this dynamic context, we change the real (i.e., the generated value) reputation score of two TS during the experiment. The real reputation score of the first TS is suddenly changed in the middle of the experiment (i.e., at the transaction number 100) from 0.5 to 0.9, while the real reputation score of the second TS is changed progressively by a step of 0.02 from -0.2 to 0.2.
during each transaction in order to obtain a triangular wave of its reputation as a graph. The real reputation score of the other TS remain constant during all the experiment. The values of the damping factor for this experiment are 0.50, 0.85, 0.99 and 1 (no damping factor).

Thus, the goal in this experimental context is of course to estimate at best the values of the parameters $\{\hat{a}_l, \hat{b}_l, \hat{\sigma}_l^y, \hat{\sigma}_k^x, \text{ and } \hat{q}_k\}$ from the ratings only. The estimated values are compared to the real generated values in order to evaluate the ability of the model to retrieve the real, generated, parameters’ value (which are hidden to the model). Two performance indicators within this context are reported: the average absolute error between real and predicted values and the linear correlation between real and predicted values.

The estimated parameters are initialized as follows:

$$\hat{q}_k \leftarrow 0, \hat{\sigma}_k^x \leftarrow 1, \hat{a}_l \leftarrow 1, \hat{b}_l \leftarrow 0, \hat{\sigma}_l^y \leftarrow 1$$  \hspace{1cm} (5.16)

### 5.5.3.2 Results

Figure 5.9 compares the real (generated, but hidden to the algorithm) and the predicted (estimated) reputation score of the first TS for various damping factors (i.e., 0.50, 0.85, 0.99 and 1). The first TS has a real reputation score changing suddenly in the middle of the experiment, i.e., at the transaction number 100. The closer the predicted values to the generated values $\hat{q} = q$, the better the prediction is (i.e., the model predicts well the actual value of the parameters). By analyzing this figure, the more the damping factor is small the more the model detects the change quickly. In fact, a small damping factor as 0.50 gives more weight to the last ratings and is more sensible to direct change than larger damping factors. But a large damping factor as 0.99 or 1 is more stable over the time providing a more stable reputation score when no change occurs (less fluctuations).

Figure 5.10 shows the real and predicted reputation score of the second TS for a damping factor equal to 0.50, 0.85, 0.99 and 1. The second TS has a real reputation score oscillating from $-0.2$ to 0.2 with a constant step of 0.02 for each transaction. This figure confirms the previous observations, small damping factor gives a predicted scores closer to the changed real scores than a larger damping factor.

The average absolute error as well as the linear correlation between the real and the predicted reputation scores ($q_k$) are provided in Table 5.5 for a damping factor equals to 0.50, 0.85, 0.99 and 1, confirming that a larger damping value provides better estimation than a smaller damping value in a more static con-
5.6 Logical Architecture

As presented in Figure 5.11, the logical architecture proposed in this section falls into four layers. The upper layer represents user client applications that interact with the middle layer and transmit client requirements.

The middle layer contains the task specialists. These TS are organized in service center (SC), which gather all the TS providing a specific task used to execute the global goal (represented by a service process model) of the SC. Thus, text (i.e., a context where the majority of the TS have a quality of service quite constant, except the first and the second TS).

Figure 5.9: Comparison of real and predicted reputation score for the first provider, on 200 transactions

![Figure 5.9](image)

Table 5.5: Comparison of four damping factors (i.e., 0.50, 0.85, 0.99 and 1) in terms of the average absolute error and the linear correlation.

<table>
<thead>
<tr>
<th>Damping factor</th>
<th>0.50</th>
<th>0.85</th>
<th>0.99</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average absolute error</td>
<td>0.2045</td>
<td>0.1801</td>
<td>0.1585</td>
<td>0.1692</td>
</tr>
<tr>
<td>Linear correlation</td>
<td>0.9705</td>
<td>0.9774</td>
<td>0.9821</td>
<td>0.9800</td>
</tr>
</tbody>
</table>
this layer illustrates the various SC connected by a communication layer. As remind, each SC has five special agents: (i) the service request manager that receives the request from the client layer, (ii) the service discoverer searching for new SC and connecting the SC in order to respond to complex service requests, (iii) the service process modeler which models the service process of its SC and is also able to build a more complex service process by merging various service process for complex service requests, (iv) service mediator which is the more important agent of a SC and is able to respond to simple or complex service request by using the service discoverer, the service process modeler and by allocating tasks to TS, and finally (v) the service reputation computing the reputation score of TS by using the client feedbacks.

The bottom layers are the technical/algorithm layer and the service process layer. Technical/algorithm layer contains all the algorithms and technical processes used and executed by the special agent of the service center, i.e., the service process modeler executes the process model algorithm to build the service process corresponding to the global goal of its service center, the service mediator uses the task allocation algorithm to allocate tasks to TS and the service reputation uses the reputation algorithm to compute the reputation score of the
5.6. Logical Architecture

Figure 5.11: Logical architecture and overview of the requirements-driven service-based architecture

TS. By keeping these algorithms and others technical processes in a lower level, we keep independent these technical features from the service center and these technical aspects can be easily changed and adapted without any changing to the higher level, i.e., the service center itself.

The service process layer contains business processes, i.e., compositions of tasks in simple services or compositions of simple services in a more complex service. These layers communicate between each other in order to answer clients' needs.
Chapter 6

Service-Oriented Framework for MAS Modeling

Contents

6.1 Problem Statement ........................................ 178
   6.1.1 Related Work .................................... 179
   6.1.2 The Lack of Scope Elements ...................... 180
   6.1.3 Contributions .................................. 180

6.2 A Unified-Framework for Agent-Oriented Requirements Engineering .......................... 181
   6.2.1 The service approach ............................ 181
   6.2.2 The Framework .................................. 182
   6.2.3 The FaMOs Framework: a Project Management Perspective .......................... 183

6.3 Service Driven Agent Modeling: a Formalization ........................................ 184
   6.3.1 Strategic Services Model ......................... 184
   6.3.2 Dynamic Service Model ......................... 186

6.4 Illustrative Example ...................................... 189
   6.4.1 The Strategic Services Diagram .................. 189
   6.4.2 Strategic Dependency Diagram ................. 190
   6.4.3 Dynamic Services Graph ......................... 190
   6.4.4 Lessons learned ................................ 192
Motivations Multi-Agent Systems (MAS) have become an important and promising research area in computer science in general and in software engineering in particular. Applying agent technology to huge enterprise information systems engineering is nevertheless seldom done. One may wonder where this report comes from. MAS development is maybe not mature enough to be applied in large industrial software development. Indeed, most of the research is focused on developing particular aspects of modeling, specifying, designing or implementing collaborating agents. Global frameworks for MAS model driven development and software project management are, until now, seldom approached in literature. Researchers need consequently to understand what dimensions are required to bring a technology to be used on a large scale and to fill the gap between the research field and its practical application.

This chapter is part of this effort: on the basis of the weaknesses identified in the i*/Tropos MAS development methodology, we propose a generic framework for organizational modeling and requirements engineering using higher abstraction level elements: the services. The importance of distinguishing the system behavior at such level will be characterized and a set of models allowing services representation through multiple complementary views will be introduced. This framework is composed of a static high level services view, of a static and a dynamic decomposed services views. Part of this work is a joint work together with Wautelet, Faulkner and Kolp.

The chapter is structured as follows. Section 6.1 presents the related work and points the limitations of the traditional Strategic Dependency and Strategic Rationale Diagrams for model driven software development. Section 6.2 points to the use of a framework based on services that can be extended for software project management. Section 6.3 formalizes the new diagrams included into the framework through the use of a meta-model. Finally, Section 6.4 uses FaMOs with an illustrative example.

6.1 Problem Statement

Some modern software development methodologies are said to be Model Driven. In such methods, the entire development process is deduced from the features modelled in high level diagrams. Object-oriented development methodologies such as the Unified Process [76, 77, 78, 101] are for example said to be use case driven. This means that the entire development process is driven by the use cases identified in the analysis phases. Moreover effort estimation techniques as Use-
6.1. Problem Statement

Case Points [157] directly use high level diagram elements - the use cases - as fundamentals for effort estimation.

Such model elements called *scope elements* are consequently useful not only to share a common high level vision with stakeholders, but also to estimate the project effort on a non-redundant basis, to evaluate related risks, to fix an acceptable level of quality, etc. Classical i*/Tropos developments using traditional Strategic Dependency (SD) and Strategic Rationale (SR) diagrams [194] for organisational modelling and requirements engineering do unfortunately not possess ideal elements to drive the development process, this will be shown in this section.

6.1.1 Related Work

As pointed out earlier, considerable work has been done in agent-oriented software development methodologies. The traditional i* modeling framework has however been poorly criticized and weaknesses as the lack of higher abstraction level elements has been only recently pointed out in literature [134]. Similarly, considerations on software project management using Tropos or other MAS methodologies are not easy to find in MAS literature. In the same way, even if some attempts such as AUML exist, no dynamic dimension implying i* concepts has been proposed and formalized. We will reference here three approaches, each one addressing one of the previously mentioned aspects. We will also briefly comment our view in regard to the aspects described in those papers.

- The process followed in this chapter has been strongly inspired by the concepts introduced in [53, 134]. Those papers define a series of features for evaluating the i* framework on the basis of case studies. The empirical study addresses the weaknesses of the approach and points to the use of higher level abstractions called business services into i* developments. Our conclusions are rather the same and we take their approach as a starting point for multiple view modeling framework.

- [48] proposes to model threats and vulnerabilities directly into SD and SR models and relates them directly with actors and goals. We also propose a framework where threats (and even opportunities) are incorporated into high level models but we relate them to services not in SD and SR diagrams. Furthermore operational solutions are incorporated to keep/improve quality of service.
6.1. Problem Statement

- [121] addresses i* security limitations and claims for the adoption of secure Tropos, an extension to the Tropos process incorporating security issues. Secure Tropos offers extensions for modeling requirements with security constraints. Moreover, in that process, security is an issue only considered at late requirement stage, the methodology partially identifies conflicts between security and other requirements. Our approach does not model security as constraint but identifies the potential threats to determine the most risky issues at the earliest stages of the project.

6.1.2 The Lack of Scope Elements

One of the main problems in using the Tropos software development methodology is to find an adequate scope element to drive the software process.

Actors from the SD and SR models are involved in dependencies of different intentional elements. The goal is the only candidate as scope element since it represents the most abstract functional issue. A complete description of these elements can be found in [194]. However, goals are not considered as the ideal scope element candidate since:

- The atomicity of the goal is not clearly defined so that a goal decomposition can involve atomic goals part of higher level goals consequently (i) a great amount of goals – too many for keeping an easily understandable vision of the software project – can be present for a huge software project; (ii) the goals are often at different levels of atomicity so that they can be redundant which is a major drawback to keep the simplest vision possible;

- The distinction between a task and a goal is not always trivial; different persons can model similar behavior by interchanging those different concepts.

Consequently, we need an element located at sufficiently high level of abstraction to represent a service the agents should provide in a non redundant way. This element should provide a clear vision of the project, allow to identify environmental threats for risk management, time management, etc. This element is called a service and is depicted in the next section.

6.1.3 Contributions

Compared to the approach followed into [53, 134], we provide in this chapter:
6.2. A Unified-Framework for Agent-Oriented Requirements Engineering

- A complete multiple view analysis framework for service-oriented modeling using i*. The framework also offers the fundamental elements to develop a complete model driven software project management framework. It can be considered as a startup for a global broader software engineering method;
- A dynamic model for representing services realization at defined quality of service level;
- Enhancements into the Strategic Services Model to model system threats;
- An illustrative example.

### 6.2 A Unified-Framework for Agent-Oriented Requirements Engineering

This section introduces the services approach and depicts the framework in an informal manner.

#### 6.2.1 The service approach

In [53, 134], the authors describe business services as "an abstract set of functionalities that are provided by a specific actor", they also specify that "an actor can be an organizational entity . . . that uses or offers services".

Services will be formalized in Section 6.3.2. In our approach, they present the following properties:

- they represent the highest abstraction level entities of the project, a service can encapsulate a large amount of atomic aggregate goals or tasks;
- they are independent but complementary to each other in the sense that they do not overlap, but represent the whole of the system functionalities;
- they involve at highest level two super actors that can be refined in multiple sub actors at lower levels;
- their realization induces of different quality of service (QoS) areas.

Moreover, we also propose to add concepts to model environmental elements addressing a threat – potentially lowering QoS – or an opportunity potentially
6.2. A Unified-Framework for Agent-Oriented Requirements Engineering

raising QoS. Once identified, we can also represent in a generic and in a more detailed manner, the measures that should be taken to fight against the threats (to maintain QoS) or to put the opportunities into practice (to raise QoS).

6.2.2 The Framework

To fix the weaknesses pointed out previously, a service-oriented framework for organizational modeling will be introduced. This framework is made of two complementary views represented by four different diagrams. Figure 6.1 depicts the framework diagrams hierarchy.

The first view called the static view allows to represent the services at two complementary abstraction levels:

- the highest level is materialized through the Strategic Services Diagram (SSD) described and formalized in section 6.3.1. This level provides a synthetic description of the services provided at defined QoS by the organization and that will be operationalized into the application. Environ-
mental factors influencing positively or negatively the provided services as well as means to operationalize them are represented here. This diagram can be used as a reference vision document for stakeholders as well as a basis for iteration planning, risk management, quality management, effort planning, etc. (see Section 6.2.3);

- the lowest level is materialized through the traditional Strategic Dependency and Strategic Rationale Diagrams as described in [194]. This level provides a more detailed description of the tasks, goals and resources involved in the achievement of all the services.

The second view called the dynamic view allows to represent the service achievement through the use of the Dynamic Service Graph (DSG) described and formalized in Section 6.3.2. The goals and tasks of the multiple agents involved are sequentialized in the graph to represent possible paths for service realization at given QoS.

Taken as a whole, these views allow to represent the multiple aspects of the services the system should offer.

6.2.3 The FaMOs Framework: a Project Management Perspective

As described previously, one of the main goals of the framework is to allow model driven software development. The main advantage of this type of approach is to furnish structured developments methodologies to software professionals. The framework was conceived to provide this kind of achievement but also furnish a broader range of possibilities.

Those services include:

- Risk management. By allowing modeling services threats and the operational solutions designed to maintain QoS when those occur, the framework includes some essential features for risk management. The framework includes different levels of detail (abstraction) for modeling those features.

- Quality management. Quality indicators are present through the use of a QoS-level so that quality benchmarks for each service can be fixed early into the project. Furthermore, through the “services opportunities”, improvement factors that, once operationalized, allow to higher QoS can be modelled at different levels. Finally, the framework constitutes an extension of traditional i*/Tropos modelling diagrams so that softgoal modeling
6.3. Service Driven Agent Modeling: a Formalization

...into the Strategic Dependency and Strategic Rationale Diagrams remains possible.

- Time management. Using the services as high level abstractions and the DSG for the service complexity evaluation, an approximate service effort can be computed (as for example in the Use Case Points methodology [12, 158]). On the basis of this evaluation, a project planning using waterfall or better an iterative development life cycle can be created. By including the number of human resources, their role, their wages, etc. a project cost evaluation can also be computed. The precision of those evaluations will successively increase from a project to another on the basis of the empirical statistics collected during the previous ones.

This chapter will only focus on the models formalization and the illustration of the framework on a small example (see Section 6.4). Chapter 9 introduces a case study to illustrate the use of the framework (in [188], we present also another case study). It only uses elements of risk and quality management but do not involve time and software process management. A complete presentation of the project management capabilities of the FaMOs-framework is currently under development.

6.3 Service Driven Agent Modeling: a Formalization

To drive the business and user services acquisition, we propose a meta-model which provides modeling elements relevant for specifying both strategic and operational aspects of the organizational context in which the future information system will be deployed. This meta-model is made of meta-concepts (e.g., “Actor”, “Dependencies”, “Services”, etc.), meta-relationships relating meta-concepts (e.g., “Performs”, “Operationalize”, “Act”, etc.), meta-attributes of meta-concepts or meta-relationships (e.g., “Probability_of_happening”, “Improvement_rate”, etc.), and meta-constraints on meta-concepts and meta-relationships (e.g., “An actor occupies a position if and only if that actor possesses all the capabilities required to occupy it”).

6.3.1 Strategic Services Model

The Strategic Services Model (or SSM) supports the acquisition, representation and reasoning about the services that should be provided by the agents of the application.
6.3. Service Driven Agent Modeling: a Formalization

Figure 6.1 depicts the SSM. *Actors* are intentional entities used to model people, physical devices or software systems that are processor for some actions. The inheritances from the *Actor* into *Position*, *Agent* and *Role* as well as the linking with the *Dependency* class have been inspired by the *Tropos* metamodel presented in [170]. A Position can cover 1 to n Roles, an Agent can play 0 to n Roles and can occupy 0 to n Positions. Actors achieve some *Services* which are functionalities they offer to others in order to fulfil a portion of their goals and tasks (for further formalization see section 6.3.2). The Service is always achieved through an Actor Dependency. The later relates two Actors: a *dependee* and a *dependum*, the *Service*.

Since the aim of the diagram is to offer a high level view of an under development enterprise information system, there are basically two types of services that need to be modelled: Business Services – business processes achieved into the company business domain – as well as User Services – services provided to the end user.

An *Environmental Factor* can be an external *Threat* for the adequate fulfilment of the service (both in terms of achievement and quality of service degradation) or an *opportunity* that can potentially raise the QoS. A service and its QoS are operationalized into a digraph as depicted in section 6.3.2. Moreover a *Probability of happening* is associated to each Threat as well as an *Improvement rate* to each Opportunity. Those features will be useful for the Threats and Opportunities prioritization into project planning.

An *Operational Solution* operationalizes an *Environmental Factor*. In other words Threats and Opportunities are external factors that can lower or raise QoS. To fight against a Threat or to benefit from an opportunity measures need to be taken. Those can be represented through the *Operational Solution*. The later is specialized into the *Risk Measure* and the *Improvement Solution* respectively aimed to model the operational solution putted into practice to fight against a threat and to take benefit of a potential opportunity. Note that once again an element (the operational solution) distinguished at SSD level induces a higher level of abstraction that will be refined further into the framework through SD/SR Models and DSG.

The elements of the SSM are instantiated to provide a *Strategic Service Diagram* (or SSD). An SSD presents on the highest aggregation level of the framework the services provided by each agent as well as the potential threats and opportunities they are facing. It also supports the identification of operational solutions that can be taken to fight against the Threats potentially lowering QoS and to take benefits of Opportunities potentially raising QoS. It shows the poten-
6.3. Service Driven Agent Modeling: a Formalization

Figure 6.2: Strategic Services Diagram: A Meta-Model.

...tial problems that will be encountered when attempting to achieve Services and, with the guidance of the methodology, helps operationalizing them. By essence, the SDD helps to understand the purpose of the software project in terms of Services, the problems (i.e. Threats) they can face, the potential improvements (i.e. Opportunities) and the social structure (i.e. the Dependencies) which govern Actor interactions.

6.3.2 Dynamic Service Model

In this section, we will bring the Dynamic Service Model (DSM) to further formalization. Figure 6.3 depicts the DSM. As the only properties of the agent relevant for this dynamic view are the capabilities in terms of tasks and goals it can execute along with its advertised quality values, an agent will be defined as follows.

A tuple $\langle \{(c_{p_1}^{a}, q_{c_{p_1}}^{a}), \ldots, (c_{p_{i+m}}^{a}, q_{c_{p_{i+m}}^{a}})\}, Ag^{a}\rangle$ is called an agent $a$, where $c_{p_i}$ is a capability. The agent advertises its capability to execute a task or goal...

---

*Most of these definitions have been already defined partially in Chapter 5. We repeat these definitions to improve the clarity and adapt them to a most general case.

†Note that since, in this view, the service realization is documented we refer to agent rather than to actors as in the SSD.
to the QoS level and cost \( q^a_{cp_i} \). The advertised level is a vector of QoS- and cost-property and value pairs following a QoS ontology. \( Ag^a \) is assumed to contain all additional properties of the agent irrelevant for the present discussion, yet necessary when building a MAS.

Format and content of \( Ag^a \) will depend on the agent programming language being used. A Capability is the generalization of Goal and Task. As it is unimportant to know that an agent can perform more than one capability, a specialist agent is defined over a single capability.

\[
\langle a, cp_i, q^a_{cp_i} \rangle
\]

associating a capability \( cp_i \) to a quality level \( q^a_{cp_i} \) advertised by the agent \( a \) is a specialist agent \( a_i^{SA} \). The agent must be capable of performing the capability: \( \forall a_i^{SA} = \langle a, cp_i, q^a_{cp_i} \rangle, (cp_i, q^a_{cp_i}) \in a \).

Any agent \( a \) that can accomplish \( m > 1 \) capabilities can also be seen as a set of specialist agents: \( \{ a_i^{SA}, \ldots, a_{i+m}^{SA} \} \). It is necessary to have a more precise idea of how capabilities and services are conceptualized.

A capability \( cp_i \) is \( \langle cp_i^{pre}, \tau_i, cp_i^{post} \rangle \), where \( cp_i^{pre} \) describes the capability precondition, \( \tau_i \) is a specification (in some language) of how the agent is to execute the capability, and \( cp_i^{post} \) describes the conditions true after the capability is executed. Capabilities belong to the set \( CP \).

\[
\langle s_j, s_j^N, s_j^E, servTransit_j, servState_j \rangle
\]

is a service \( s_j \), where \( s_j^t \) provides the details of the functional specification of the service, \( (s_j^N, s_j^E) \) defines a directed acyclic graph. Nodes represent states, and edges transitions between states. The two functions label nodes and edges with capability information: \( servTransit_j : \)
6.3. Service Driven Agent Modeling: a Formalization

$s^E_j \mapsto \mathbb{CP}$ is a partial function returning the capability for a given edge in the graph, while $\text{servState}_j : s^N_j \mapsto \{cp^\text{pre}_i\}_{cp_i \in \mathbb{CP}} \cup \{cp^\text{post}_i\}_{cp_i \in \mathbb{CP}}$ maps each edge to a condition from the set of all capability preconditions (i.e., $\{cp^\text{pre}_i\}_{cp_i \in \mathbb{CP}}$) and postconditions (i.e., $\{cp^\text{post}_i\}_{cp_i \in \mathbb{CP}}$). The capability specified on an edge must have the precondition and postcondition corresponding to conditions given, respectively, on its origin and its destination node.

Because quality of services, norms, sanctions, and incentives ensure that the system fulfils its purpose, they are derived from requirements placed on the system by users and other system stakeholders. In order to apply at design stage these requirements defined at analysis stage, we include the concept of Service Level Agreement.

Service Level Agreement (SLA) is a document which defines the quality of service requested between a provider and a client [95], and the system requirements described at analysis stage. SLA is therefore a contract in which they formalize her quality of the related service. In short, it represents an agreement between the providers at the level of services, preferences, incentives, sanctions, responsibilities and guarantees. SLA are used to meet system and client’s requirements, manage client’s expectations, regulate resources and control costs [154]. A SLA covers the functional side of the service (the provided service corresponds to the requested service in terms of input, output, pre and post-conditions) and concerns also the non-functional properties of the service [95].

A service can therefore be understood as a process, composed of a set of Tasks and Goals (i.e., these are specializations of capability) ordered over the graph representing the service. The functional specification of the service, i.e., $s^T_j$ is not of interest here, but involves in practice, e.g., a specification of interfaces, and other implementation considerations. Requesting a service requires the specification of expected QoS, in addition to a deadline for providing the service, minimal level of reputation for agents that are to participate in service execution, the maximal monetary cost, and explicit user preferences on agents to select (e.g., users may prefer globally the services of some providers over others, regardless of actual performance—this may occur with preferential threatment resulting from environment constraints such as, e.g., legal constrasts on cooperation between organizations and/or individuals).

By conceptualizing the service as suggested, the service is thus mapped onto a directed graph $G$ where each node is a step in service provision and an edge in $G$ corresponds to the execution of a capability $cp_k$ by a specialist agent $a^S_{k,u}$, where $u$ ranges over specialists that can execute $cp_k$ according to the criteria set in the service request. Each individual edge is labelled by a function of the cri-
6.4 Illustrative Example

In this section, the framework is applied to a simplified example: the travel planning service. Travel planning regroups all the activities provided by a service provider destined to organize a business or holiday trip. Those activities include the booking of facilities such as an hotel, a flight, a car, a taxi, etc. The different dimensions of a service description and fulfillment will be envisaged, their complementarities will be illustrated.

6.4.1 The Strategic Services Diagram

Figure 6.5 depicts the Travel Planner’s Strategic Services Diagram. In this service, the Customer addresses to a Travel Planner for its travel organization. The Travel Planner is in charge of the Commodities Booking to the...
6.4. Illustrative Example

Resource Provider (which can be an hotel resort, a flight company, a taxi company, etc. i.e. all the companies furnishing commodities to the Customer) for the destination of the Customer on the basis of its Budget, Proposed Date and Agreement.

![Figure 6.5: Strategic Services Diagram - Travel Planner.](image)

6.4.2 Strategic Dependency Diagram

Figure 6.6 illustrates a strategic dependency diagram of the Travel Planner system’s requirements. The system is intended to support a customer and a travel planner in travel related activities. Three actors are introduced into this diagram: Travel Planner is responsible for the organization of the travel and the necessary reservations. Customer gives these preferences concerning its travel which he wants to plan. Resource provider deals with providing the various resources helping to planning.

A strategic rationale diagram of the Travel Planner system’s requirements is depicted in Figure 6.7. This Figure details the various actors introduced into the strategic dependency diagram.

6.4.3 Dynamic Services Graph

Figure 6.8 represents fulfilment paths for the service Travel Planner. Each node is a step in service provision and each edge corresponds to the execution of a task $t_k$ by a specialist agent $a_{k,u}^{SA}$, where $u$ ranges over specialists that can execute $t_k$ according to the criteria set in the service request. A function of the
6.4. Illustrative Example

Figure 6.6: Strategic Dependency Diagram: Travel Planner.

criteria set, \( c(t_k, a^{SA}_{k,u}) \), labels each edge and represents the QoS advertised by the specialist \( a^{SA}_{k,u} \) for performing the task \( t_k \). Note that different paths offering different QoS are available. Indeed, path \( < t_1, t_2, t_3, t_4, t_6, t_7 > \) offers alternative ways of fulfilling the service. The following list gives a description of each task used in the graph.

- \( t_1 \): The customer proposes a range of dates fitting for his travel.
- \( t_2 \): The customer expresses the maximal budget he is able to allow for his travel.
- \( t_3 \): The Resource Provider gets a list of proposition for commodities on the basis of the customer constraints.
- \( t_4 \): The customer selects its preferred commodities.
- \( t_5 \): On the basis of the constraints defined, the Service provider organises a trip plan for the customer. Finally, the Service Provider books the commodities (i.e., flight, hotel, ...).
- \( t_6 \): The customer accepts a list (date, budget, flight and hotel) proposed.
6.4. Illustrative Example

Figure 6.7: Strategic Rationale Diagram: Travel Planner.

Figure 6.8: Dynamic Service Graph - Travel Planner.

- $t_7$: The Resource Provider provides the resource to the customer.

6.4.4 Lessons learned

This simplified example allowed illustrating:

- The use of different levels of knowledge:
  
  - A first level materialized through the SSD where stakeholders can have an aggregate view of the services the system offers as well as the threats and opportunities;
  
  - A second level materialized through the SDM, SDM and DSD to decompose the service in a static and dynamic manner. In this view analysts can see respectively the actors involved as well as the multiple realization paths.
6.4. Illustrative Example

- The alternative paths in the graph with different QoS.

To demonstrate the real contribution of the framework it should however be applied to a broader case study implying multiple services. This is done in Chapter 9.
Chapter 7

Process for Developing Adaptable and Open Service Systems

Contents

7.1 Open and Dynamic Architectures . . . . . . . . . . . . . 196
7.2 Process Structure . . . . . . . . . . . . . . . . . . . . . . . . 196
7.3 Process Life Cycle . . . . . . . . . . . . . . . . . . . . . . . . 197

Motivations

This chapter describes a process for developing adaptable and open service systems. At analysis level, the framework refines the traditional i* approach by including the FaMOS framework which introduces new models and refines existing ones. At design level, it defines a custom architecture as well as task allocation (thought reinforcement learning) and reputation algorithms for the development of the modelled problem converted in an architectural solution. The process is generic and flexible enough to be adapted to any MAS development methodology.

The remainder of the chapter starts with Section 7.1 is an introduction to open and dynamic architectures as well as the process context. Section 7.2 introduces the process ProDAOSS and specifies its components, i.e., the analysis and design stages. Section 7.3 describes the process life cycle.
7.2. Process Structure

7.1 Open and Dynamic Architectures

The process is aimed to develop open and dynamic multi-agent systems. Following [88], a multi-agent system (MAS) is open if it relies on heterogeneous agents designed, operated and maintained by, and distributed across, various providers. The number and kind of agents participating in the realization of the MAS is not known when developing the software but varies at runtime. Openness implies distributed architectures for entering and leaving heterogeneous resources bounded to be decentralized. Dynamic feature leads to architectures exhibiting the ability to self-organize and respond to changes in the environment without human involvement. Efficiency, flexibility, and adaptiveness that such architectures are expected to exhibit are valuable given the pressing complexity concerns.

The Process for Developing Adaptable and Open Service Systems (ProDAOSS) is a software development process for developing such multi-agent systems. It encompasses analysis models to represent software application requirements on different levels and through different views as well as design models to map operationally described requirements onto a multi-agent system.

7.2 Process Structure

Management sciences define several layers for decision making. Indeed, decisions do not have the same impact - from a marginal short term consequence to a major long term impact - so that their time horizon is variable. Traditionally, management sciences identify three levels of decision making to discriminate time horizons and resources that should be allocated:

- **The Strategic Level.** Strategic decisions are top level. Decisions taken here concern general direction, long term goals, philosophies and values;

- **The Tactical Level.** At the tactical level, more concrete decisions are taken which aim at implementing the strategy defined at the corporate level. The business units adapt this strategy in terms of policies under which the operations will happen;

- **The Operational Level.** These are every day decisions, used to support tactical ones. Their impact is immediate, short term, short range, and usually low cost. Operational decisions can be pre-programmed, pre-made, or set out clearly in policy manuals.
7.3. Process Life Cycle

Software development methodologies are traditionally divided into a set of disciplines. Among those disciplines the analysis and design stages are the most critical since the whole development is based on them. Moreover, the analysis phase is made of different complementary aspects to represent the problem to solve at different levels of abstraction and through different views. The different decision levels are incarnated by individuals having different visions and expectations on the software product in development. Indeed, while the software application is mainly used by operators (at operational level), requirements are often supervised by middle management (tactical level) while top management only has to perceive the main functionalities of the software package (strategic level).

PRoDAOSS defines three models at analysis stage to represent software project information through the three decision levels described above. Concretely, a top level view (strategic) offers an aggregate representation (the Strategic Services Diagram, see Chapter 6, Section 6.3.1) of the services the application has to offer, a middle layer view (tactical) further documents the elements (actors, goals, tasks and resources) that have to be taken into account when implementing the software application services using i* Strategic Dependency and Strategic Rationale Diagrams. Finally, the lowest level (operational) offers a dynamic view (the Dynamic Services Graph, see Chapter 6, Section 6.3.2) documenting the services realization paths and constraints (the Service Level Agreement, see Chapter 6, Section 6.3.2 for further formalization).

Finally, at design stage, the Service Center Architecture described in Chapter 5 aims to implement the graph documented at analysis operational level. The graph defines the series of tasks that have to be achieved as well as with respect to the constraints evoked.

7.3 Process Life Cycle

Since, in open architectures, the number and kind of agents participating in the realization of the MAS is not known when developing the system but varies at runtime, new information about the agents and their contextual specification can emerge. Indeed, agent contextual changes will appear into the open multi-agent systems and further documentation will appear. That is why we encourage iterative enhancements between the analysis operational level and design stage on the basis of the runtime experience.

Figure 7.1 represents the PRoDAOSS process analysis and design stages. It includes a feed back loop between the analysis and design stages to include in
7.3. Process Life Cycle

analysis models the iterative enhancements experienced at runtime.

Figure 7.1: ProDAOSS: Analysis and Design stages
Part V

Applications
Application to Web Services

Contents

8.1 Designing Service Center with FaMOS .......................... 203
  8.1.1 Strategic Services Model ................................. 203
  8.1.2 Strategic Dependency Model ............................... 204
  8.1.3 Strategic Dynamic Model ................................. 206
8.2 Service Center Architecture .................................. 206
8.3 Randomized Reinforcement Learning and Optimization
  Under Constraints .............................................. 208
  8.3.1 Reinforcement Learning for WS Composition ............. 210
  8.3.2 Generic Service Request Model ........................... 211
8.4 Randomized Reinforcement Learning Algorithm ............. 214
  8.4.1 RL Formulation of the Problem ........................... 216
  8.4.2 Dealing with Task Concurrency ........................... 217
  8.4.3 Computation of the Optimal Policy ........................ 219
8.5 Simulation Results ............................................ 222
  8.5.1 Experimental setup ....................................... 222
  8.5.2 Results .................................................. 224

Motivations This chapter focuses on the composition of services under the constraints of openness, resource distribution, and adaptability to changing web service availability w.r.t. multiple criteria and constraints. To enable such system characteristics, a fit between the system architecture and service composition
behavior is needed, that is: (1) To support openness, few assumptions can be made about the behavior of the web services that may participate in compositions. It thus seems reasonable to expect service composition responsibility not to be placed on any web service: the architecture ought to integrate a special set of web services, the mediators, that coordinate service composition. (2) To allow the distribution of web services, no explicit constraints should be placed on the origin of entering services. (3) To enable adaptability, mediator behavior should be specified along with the architecture. (4) Since there is no guarantee that web services will execute tasks at performance levels advertised by the providers, composition should be grounded in empirically observed service performance and such observation be executed by the mediators. (5) The variety of stakeholder expectations requires service composition to be driven by multiple criteria. (6) To ensure continuous adaptability at runtime, composition within the architecture should involve continual observation of service performance, the use of available information to account for service behavior, and exploration of new options to avoid excessive reliance on historical information.

A simplified version of the Service Center Architecture is proposed that corresponds to the requirements (1)–(6) above and where we focus only on the role of the mediator. The service process modeler, service reputation, service request manager and service center discoverer are not developed here. Thus, the architecture organizes web services into groups, called “service centers”. Each service center specializes in the provision of a composite (i.e., the composition of available web services) service. Within each center, a single “mediator web service” receives service requests. Upon reception, the mediator decides how to compose the available web services into the composite service to be delivered, and this depending on observed prior performance while accounting for anticipated performance of newly available services. As no constraints are placed beyond organizing service delivery through mediators, the architecture is adapted to open and distributed systems. Within such an architecture, each mediator plays a critical role: it organizes work by composing services, negotiates with individual services, and observes their performance in order to adjust compositions in the aim of continually optimizing quality criteria. Mediator behavior is guided by a multicriteria-driven (including QoS, deadline, reputation, cost, and user preferences) reinforcement learning (RL) algorithm, called Randomized Reinforcement Learning (RRL) (first introduced in Chapter 2 and specialized here), which integrates the exploitation of acquired knowledge with optimal, undirected, continual exploration. Instead of relying on experience only, exploration allows the mediator to continually explore the pool of agents avail-
able for task allocation, thus ensuring the responsiveness of the system to change. The reported experiments show the RRL outperforming two standard exploration methods, namely, $\epsilon$-greedy and naive Boltzmann.

The principal contributions of the present chapter are the architecture and the composition algorithm which is the adaptation of the algorithm suggested in Chapter 2 for use in WS composition. Compared to that work, the algorithm is adapted herein for the WS composition problem, extended for constrained search for optimal compositions, and accommodates concurrent task execution. No specific commitments are made on, e.g., task allocation or negotiation protocols, to ensure the results are generic. Such choices are left to the designer. Part of this work is a joint work together with Jureta and Faulkner, and has been published in [6].

The present chapter is structured as follows. Section 8.1 expresses with the FaMOS framework the Service Center. Section 8.2 presents the description and discussion of the Service Center Architecture simplified and instancied here for WS. Section 8.3 motivates the use of reinforcement learning in web composition and explained the role of the mediator. The generic model for describing SR is then presented in Section 8.3.2. The algorithm is introduced and discussed in Section 8.4 and simulation results presented in Section 8.5.

8.1 Designing Service Center with FaMOS

This section presents a description of the Service Center through the models included in the FaMOS framework. For this description, we illustrate a general description taking all the features of the Service Center (i.e., all the special agents) presented in Chapter 5. The simplification and instantiation of this description for WS is trivial and left to the interested reader.

8.1.1 Strategic Services Model

Figure 8.1 depicts the Strategic Service Diagram of the Service Center Architecture. In this diagram, a service consumer addresses to a Service Center (SC) for providing a (web) service. As remind, each SC can provide a specific service depending on its process model and the task agents available in this SC. Thus, a SC is in charge of the decomposition, the forwarding (i.e., if this SC can respond to the entire service request, it forwards the request to another SC or to the Support Center) and the execution of the service requested by the consumer on the basis of the parameters and the constraints giving in the consumer request.
8.1. Designing Service Center with FaMOs

We define also five opportunities in this diagram. The first opportunity is to use an agent reputation model based on the consumer feedbacks. This reputation model represents the behavior of the agents in executing a task of the service requested. This opportunity is related to a second opportunity, improve QoS. By using the reputation model, SC can execute a service by allocating the tasks to the agents that better suit to the QoS defined by the consumer. To improve the QoS, we have also included two opportunities: increase agent availability and load balancing. The task allocation algorithm has to allow a good load balancing in order to improve the QoS by increasing the availability of the agents.

8.1.2 Strategic Dependency Model

Figure 8.2 illustrates a Strategic Dependency Diagram of the Service Center requirements. The SC architecture is intended to support a consumer and Task agents organised in a Task Agent Pool in order to respond to a service request. Seven actors are introduced into this diagram: Service Consumer requests a service including the parameters and the constraints in term of QoS. Service Request Manager (SRM) is the public interface of SC. It receives the consumer request and forwards it to the private interface, Service Mediator (SM). SM is
8.1. Designing Service Center with FaMOs

Figure 8.2: Strategic Dependency Diagram - Service Center.

the most important agent in a SC. It handles the consumer request received from the SRM and decomposes it. The decomposed request is processed by this way: at first, the external sub-request (i.e., the sub-request that cannot be executed by the corresponding SC) is forwarded to the Service Center Discovery (SCD), and the internal sub-request is dynamically allocated by the SM to the Task Agents by following the process model and a task allocation algorithm. Afterwards, SM merges the external answers (i.e., answers received from another SC) with the internal answers (i.e., the answers resulting from the execution of the allocation list) and sends an integrated result to the SRM. Service Center Discovery handles external sub-requests by finding on the network SC able to respond to theses sub-requests. Service Process Modeler (SPM) creates the process model. Finally, the Service Reputation agent (SR) uses the consumer feedback to compute a reputation score of the task agents involved in the requested service.
8.2. Service Center Architecture

8.1.3 Strategic Dynamic Model

Figure 8.3: Dynamic Service Graph - Service Center.

Figure 8.3 represents the fulfillment paths for the operation of SC. Each node is a step in service provision and each edge corresponds to the execution of a task \( t_k \) by a specialist agent \( a_{k,u}^{SA} \), where \( u \) ranges over specialists that can execute \( t_k \) according to the criteria set in the service request. A function of the criteria set, \( c(t_k, a_{k,u}^{SA}) \), labels each edge and represents the QoS advertised by the specialist \( a_{k,u}^{SA} \) for performing the task \( t_k \). Note that different paths offering different QoS are available. Table 8.1 gives a description of each task used in the graph.

8.2 Service Center Architecture

The Service Center Architecture (SCA) groups services into Service Centers (SC). Each SC contains one Mediator Web Service (MWS) and all distinct Web Services (WS) needed to provide a Composite Web Service (CWS) corresponding to the Service Request (SReq; e.g., finding the itinerary between two physical addresses as common in map applications, booking a flight, searching for files, and so on) originating from the user (who can be an WS or an MWS). The MWS composes WS by observing past performance of individual WS, then subsequently using (and updating) this information through the RRL (see, Section 8.4). Combining the SCA and the RRL brings the following benefits: (a) Adaptability to changes in the availability and/or performance levels of WS is ensured, as the algorithm accounts for actual performance observed in the past and explores new compositions as new WS appear. (b) Continuous optimization...
Table 8.1: Description of the tasks included in the graph

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Destination state</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$t_1$</td>
<td>Generation of the process model on the basis of the SC global goal expressed in first-order logic.</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$t_2$</td>
<td>Handle the consumer service request by using the generated process model.</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_3$</td>
<td>$t_{2'}$</td>
<td>Handle the consumer service request by using a predefined (not generated from a global goal) process model.</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$t_3$</td>
<td>Decompose the consumer request into atomic sub-requests (or tasks).</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$t_4$</td>
<td>Send external sub-requests to another SC.</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$S_6$</td>
<td>$t_5$</td>
<td>Allocation of the tasks corresponding of the internal sub-request. This task generates a task allocation list.</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$t_6$</td>
<td>Receives external sub-answers from other SCs.</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$S_8$</td>
<td>$t_7$</td>
<td>Executes the task allocation list. This task creates internal sub-answers.</td>
</tr>
<tr>
<td>$S_8$</td>
<td>$S_9$</td>
<td>$t_8$</td>
<td>Merge external and internal sub-answers into an integrated result.</td>
</tr>
<tr>
<td>$S_9$</td>
<td>$S_{10}$</td>
<td>$t_9$</td>
<td>Send the integrated result.</td>
</tr>
<tr>
<td>$S_{10}$</td>
<td>$S_{11}$</td>
<td>$t_{10}$</td>
<td>Compute reputation score of the agents on the basis of implicit data’s.</td>
</tr>
<tr>
<td>$S_{11}$</td>
<td></td>
<td>$t_{11}$</td>
<td>Compute reputation score of the agents on the basis of user feedbacks (i.e., explicit data’s).</td>
</tr>
</tbody>
</table>

over various criteria in the algorithm allows different criteria to guide service composition, while exploration and exploitation ensure MWS continually revise composition choices. (c) By localizing composition decisions at each MWS, the architecture remains decentralized and permits distribution of resources. (d) The architecture and algorithm place no restrictions on the openness of the system or resource distribution in the system.

Because a number of CWS involve the execution of common services, the presence of generic services (i.e., those whose frequency of execution is above some externally fixed threshold) makes it possible to pool the information about idle WS executing the generic services into the same center, called the Support Service Center (SSC). The effects sought in doing so are (i) ensuring the availability of WS which execute frequently needed tasks; (ii) having an MWS dedicated to identifying and scheduling the work of WS which are most appropriate for generic services in various SC; and (iii) avoiding communication between various MWS regarding generic WS, but instead centralizing relevant informa-
A tuple \( \langle I, O, \tilde{s}^{QoS}, \tilde{s}^{cost}, s \rangle \) is called a Web Service (WS) \( w \). \( I \) and \( O \) specify, respectively, the inputs and the outputs of the service. The WS advertises its capability to provide the service to the QoS levels given by the vector \( \tilde{s}^{QoS} \) and cost \( \tilde{s}^{cost} \). The structure of \( \tilde{s}^{QoS} \) is determined by the employed QoS ontology. \( s \) is a specification of all additional properties of the service irrelevant for the present discussion, yet necessary when building a system.

Section 5.2 revisits the SCA with more precision. To apply SCA to web services, we use a simplified SCA including only a mediator as special agent (i.e., the service requester, service discoverer, service process modeler and service reputation are not included in this simplified SCA). As remind, the mediator is the more important agent in the SCA (we can’t use a SCA without a mediator). Presence or absence of the other special agents depends entirely on the choices of the MAS designer. Roughly speaking, these special agents can be optionally added to the architecture, depending to the designer in actual implementation.

### 8.3 Randomized Reinforcement Learning and Optimization Under Constraints

Thus, a service center incorporates service mediators which manage service requests from human users or other systems, then discovers, selects, and coordinates the execution of services so as to fulfill given service requests.

Fulfilling a service request in an SC often involves the execution of several tasks, requiring the participation of various WS. When facing such request, the service center proceeds to WS composition. Process-based composition is one relevant approach for the integration of distributed and open applications [17]. A process model defined in a service request gives a high-level description of tasks that the WS participating in the WS composition need to execute, along with the interactions between the participating WS. WS composition then consists of identifying the WS able to perform the required tasks, allocating the relevant WS to corresponding tasks, and coordinating WS execution according to the process model. This chapter focuses on the process-based composition of WS under the constraints of openness, resource distribution, and adaptability to changing web service availability w.r.t. multiple criteria and constraints. Our problem is therefore that of defining an appropriate task allocation procedure that guides service mediators in an SC and when composing WS for a service request. “Task allocation” designates the problem of allocating individual WS to tasks specified
8.3. Randomized Reinforcement Learning and Optimization Under Constraints

in the process model of a service request. “Procedure” denotes an algorithm (or some other form of behavior description) that governs how the mediator chooses a particular WS composition (i.e., a particular allocation of WS to tasks in a process model of a request) among the set of all potential WS compositions.

Defining a task allocation procedure for mediators is rendered difficult by the necessity to account for:

- The variation in the pool of WS to consider for a composition: the set of WS available at any time changes—new services may become available, while others become unavailable. An important consequence is that there is no guarantee that a WS composition proved relevant over some period remains such throughout runtime.

- The availability of many competing WS: Usually, many WS are able to execute a particular task within a SR, each satisfying to a different level some set of criteria, so that one WS among the competing ones needs to be selected.

- The non-determinism of WS executions: In actual application, there is no certainty that a WS will execute as expected and produce the exact anticipated output.

A task allocation procedure which acknowledges and is robust with regards to the above ought to:

1. Revise WS compositions as new WS appear and the WS participating in old compositions become unavailable. That is, select at runtime the WS for a composition rather than at design time. Otherwise, newly available WS are not taken into account so that compositions will fail to fulfill a request at all or to a desirable extent if a participating WS becomes unavailable.

2. Use multiple criteria (e.g., price, reliability, reputation, etc., including the preferences and constraints set by the requester) to choose one among competing WS. When competing WS are available, a rich set of comparison criteria is required to discriminate among competitors. Fine differences between competing WS may not be accounted for if a very restricted set of criteria is used.
3. Avoid selecting WS based on composition criteria values advertised by the service providers, basing instead the evaluation of a WS on its observed performance. Under competition, each provider has the interest to advertise better performance than the competitor, so that advertised values over composition criteria may not reflect properly the actual performance of WS.

4. Allow for uncertainty in the outcome of executions of individual WS selected to participate in a composition. Additional cost is otherwise incurred in developing monitoring mechanisms to reestablish functionality when unexpected outputs are obtained.

To enable 1–4, we advocate that WS compositions optimal w.r.t. a set of criteria need to be learned at runtime and revised as new WS appear and availability of old WS changes, whereby the learning should be based on observed WS performance, and not the performance advertised by the service providers. To enable such learning, an allocation procedure is needed which both exploits the observed past performance of WS, and explores new composition options to avoid excessive reliance on past data. To this aim, we suggest the Randomized Reinforcement Learning (RRL) approach to WS composition. RRL integrates two components:

- A generic service request model to describe the process to execute by the WS composition and the criteria and constraints to meet when executing it. The SR model indicates the kind of information that the algorithm requires from the service requester when allocating WS to tasks in the process.

- A reinforcement learning algorithm, called Randomized Reinforcement Algorithm (RRL) and presented in Chapter 2, to select the WS for performing tasks specified in the service request. The algorithm decides on the WS to select among competing WS based on multiple criteria (including various quality of service parameters, deadline, reputation, cost, and user preferences), while both exploiting available WS performance data and exploring new composition options.

8.3.1 Reinforcement Learning for WS Composition

One of the main goals of automating WS composition is to reduce as much as possible the intervention of the requestor (be it a human user or another system)
8.3. Randomized Reinforcement Learning and Optimization Under Constraints

in the fulfillment of a given SR. Ideally only high-level goals need to be specified, and the system would take care of all activities needed to fulfill the goal. If we abstract from the composition automation issues discussed elsewhere (such as discovering candidate WS—e.g., [18], and matching them—e.g., [116]), another important issue is the selection of WS that are to participate in performing the process described in a service request (SR). This problem is referred to as the task allocation problem in the remainder.

Using the terminology of Chapter 2, reinforcement learning can be said to refer to trial-and-error methods in which the mediator learns to make good allocations of WS to tasks through a sequence of “interactions”. In task allocation, an interaction consists of the following:

1. The mediator identifies the task to which a WS is to be allocated.
2. The mediator chooses the WS to allocate to the task.
3. The mediator receives a reward after the WS executes the task. Based on the reward, the mediator learns whether the allocation of the given WS to the task is appropriate or not.
4. The mediator moves to the next task to execute (i.e., the next interaction takes place).

8.3.2 Generic Service Request Model

The principal aim of the service request (SR) model presented in this section is to enable the description of the process that a WS composition is to execute, along with the definition of criteria and constraints that are to guide the mediator when allocating tasks to WS. No claims on originality are made regarding the model: it is purposefully generic, so as to facilitate the use of RRL with available WS-enabling technologies. We define an SR as follows.

A service request consists of the following:

- A process model which defines the sequence of tasks to execute in order to fulfill the service request.
- A vector of Quality of Service (QoS) dimensions and their required levels. Its definition follows an QoS ontology, such as, e.g., the FIPA QoS ontology specification [55]. Whatever the specific QoS ontology, expected qualities are likely to be specified as (at least) \( \{(p_1, d_1, v_1, u_1), \ldots, (p_r, d_r, v_r, u_r)\} \), where:
8.3. Randomized Reinforcement Learning and Optimization Under Constraints

- $p_k$ is the name of the QoS dimension (e.g., connection delay).
- $d_k$ gives the type of the dimension (e.g., nominal, ordinal, interval, ratio).
- $v_k$ is the set of desired values of the dimension, or a constraint $<, \leq, =, \geq, >$ on its value.
- $u_k$ is the unit of the dimension value.

- A deadline before which the request must be fulfilled.
- A value for reputation over QoS dimensions that any participating WS must fulfil. It is not necessary to specify reputation for all qualities over all WS, selective reputation expectations are admitted.
- A maximal monetary cost that the service requester is ready to pay to have the service request fulfilled.
- A set of assertions which constrain the pool of potential WS that can participate in the composition. For example, a requester may wish to use only some select providers’ WS.

8.3.2.1 Process Model

Similarly to Zeng and colleagues [196], our process model is defined as a statechart. Statecharts offer well defined syntax and semantics so that rigorous analysis can be performed with formal tools. Another advantage is that they incorporate flow constructs established in process modeling languages (i.e., sequence, concurrency, conditional branching, structured loops, and inter-thread synchronization). Consequently, standardized process modeling languages, such as, e.g., BPMN [31], can be used to specify the process model when using RRL.

While the statechart is a useful representation of a process that a WS composition needs to execute, the algorithm in RRL cannot process a statechart in its usual form. Instead, a statechart is mapped onto a directed acyclic graph, using definition in Section 8.3.2.1 and the technique for constructing Directed Acyclic Graphs, described below. Assume that, to allow concurrency, sets of concurrent transitions with a common origin state in a statechart are labeled AND.

(Adapted from [196]) An execution path of a statechart is a sequence of states $[t_1, t_2, \ldots, t_n]$, such that $t_1$ is the initial state, $t_n$ the final state, and for every state $t_i (1 < i < n)$, the following holds:
8.3. Randomized Reinforcement Learning and Optimization Under Constraints

- $t_i$ is a direct successor of one of the states in $[t_1, \ldots, t_{i-1}]$.
- $t_i$ is not a direct successor of any of the states in $[t_{i+1}, \ldots, t_n]$.
- There is no state $t_j$ in $[t_1, \ldots, t_{i-1}]$ such that $t_j$ and $t_i$ belong to two alternative branches of the statechart.
- All concurrent transitions are executed when an AND label on transitions is encountered.

It is apparent that an acyclic statechart has a finite number of execution paths. If the statechart is not acyclic, it must be “unfolded” [196]: logs of past executions need to be examined in order to determine the average number of times that each cycle is taken. The states between the start and end of a cycle are then duplicated as many times as the cycle is taken on average. Assuming for simplicity here that the statechart is acyclic, an execution path can be represented as a Directed Acyclic Graph.

Given a set of distinct execution paths $\{[t_{1,k}, \ldots, t_{n,k}]\}$ ($k$ is the index for execution paths), the Directed Acyclic Graph (DAG) is obtained as follows:

- DAG has an edge for every pair $(task, WS)$ which indicates the allocation of WS to the given task. DAG thus has as many edges as there are possible allocations of WS to tasks.
- DAG has a node for every state of the task allocation problem. Such a state exists between any two sequentially ordered tasks of the task allocation problem (i.e., a node connecting two sets of edges in the DAG, whereby the two tasks associated to the two sets of edges are to be executed in a sequence).
- Whenever there is a set of $n$ tasks to execute concurrently, each edge between the relevant nodes in the DAG is labeled with a tuple of pairs, e.g., $(\langle task_1, WS_1 \rangle, \ldots, \langle task_n, WS_n \rangle)$. Each edge between such nodes thus shows one possible allocation of a set of $n$ WS to the $n$ concurrent tasks.

Note that: (i) the DAG shows all alternative allocations and all alternative execution paths for a given statechart; (ii) conditional branchings in a statechart are represented with multiple execution paths. The representation of concurrent actions is revisited in Section 8.4.2.
8.3.2.2 Selection Criteria

The earlier service request definition pointed out that various criteria can be used in specifying a service request; namely, QoS dimensions along with deadline, reputation, monetary cost, and explicit requester preferences.

It is apparent that many criteria can be accounted for when selecting among alternative WS compositions. As decision making in presence of multiple criteria permits arguing for, and accepting various decision rules (which differ on, e.g., how criteria are aggregated), the algorithm is constructed to leave much freedom to the designer in actual implementation. Moreover, it does not require full specification of all possible criteria for each WS—in instead, it is up to the requester to choose what criteria to specify. The algorithm thus optimizes a single normalized (i.e., taking values in the interval \([0, 1]\)) variable, leading to three approaches to specifying this variable and the remaining hard constraints the algorithm takes as input when performing task allocation:

1. If the requester prefers to have one specific criterion in the SR optimized, expected values over the remaining criteria will be treated by the algorithm as hard constraints, whereby task allocations which violate hard constraints will be eliminated by the algorithm.

2. If the requester prefers to have several criteria optimized, it is necessary to provide an aggregation function for the relevant criteria, so that the result of the function is what the algorithm will optimize. Guidelines for aggregation functions can be found in, e.g., [184]. Non-aggregated criteria are treated as hard constraints.

3. A third option is to have the mediator suggest alternative allocations and the user chooses the one to apply. Presence or absence of this approach entirely depends on the choices of the system designer, as it does not affect the formulation of the algorithm—it is essentially the first option above, with the nuance that the user asks the mediator to provide a list of optimal allocations for each criteria, and then selects manually.

8.4 Randomized Reinforcement Learning Algorithm

If RL is applied to task allocation, the exploration/exploitation issue can be addressed by periodically readjusting the policy for choosing task allocations and re-exploring up-to-now suboptimal execution paths [119, 171]. Such a strategy
8.4. Randomized Reinforcement Learning Algorithm

is, however, suboptimal because it does not account for exploration. The Randomized Reinforcement Learning (RRL) algorithm introduced in Chapter 2 is adapted herein to task allocation in WS composition, allowing the assignment of tasks to WS while: (i) optimizing criteria, (ii) satisfying the hard constraints, (iii) learning about the performance of new agents so as to continually adjust task allocation, and (iv) exploring new options in task allocation. The exploration rate is quantified with the Shannon entropy associated to the probability distribution of allocating a task to a task specialist. This permits the continual measurement and control of exploration.

The task-allocation problem that the RRL resolves amounts to the mediator determining the WS to execute the tasks in a given process model. By conceptualizing the process in a SR as a DAG (see, Section 8.3.2.1), the task-allocation problem amounts to a deterministic shortest-path problem in a directed weighted graph. In the graph, each node is a step in WS composition problem and an edge corresponds to the allocation of a task \( t_k \) to a WS \( w_{k,u}^{WS} \), where \( u \) ranges over WS that can execute \( t_k \) according to the criteria set in the SR. Each individual allocation of a task to a WS incurs a cost \( c(t_k, w_{k,u}^{WS}) \), whereby this “cost” is a function of the criterion (or aggregated criteria, as discussed earlier Section 8.3.2.2) formulated so that the minimization of cost corresponds to the optimization of the criterion (i.e., minimization or maximization of criterion value). This criterion is the one the requester chooses to optimize, whereas other criteria are treated as hard constraints.

For illustration, consider the representation of a process model as a DAG in Figure 8.4 where nodes are labeled with states of the task-allocation problem, and edges with costs of alternative WS-task allocations (for clarity, only some labels are shown). Nodes are connected by several edges to indicate the presence of alternative allocations of the given task to WS. Any path from the starting node to the destination node in the graph is a potential complete allocation of tasks to WS (i.e., an execution path).

The task allocation problem is a global optimization problem: learn the optimal complete probabilistic allocation that minimizes the expected cumulated cost from the initial node to the destination node while maintaining a fixed degree of exploration, and under a given set of hard constraints (specified in the SR). At the initial node in the graph (in Figure 8.4, node \( s_1 \)), no tasks are allocated, whereas when reaching the destination node (\( s_{13} \) in the same figure), all tasks are allocated.

The remainder of this Section is organized as follows: Section 8.4.1 introduces the notations, the standard deterministic shortest-path problem, and the
1. **RL Formulation of the Problem**

At a state $k_i$ of the task allocation problem, choosing an allocation of $t_{k_i,l}$ (where $l$ ranges over tasks available in state $k_i$) to $w_{k_i}^W$ (i.e., moving from $k_i$ to another state) from a set of potential allocations $U(k_i)$ incurs a cost $c(t_{k_i,l}, w_{k_i}^W)$. Cost is an inverse function of the criterion the user wishes to optimize (see, Section 8.3.2.2), say $r$. The cost can be positive (penalty), negative (reward), and it is assumed that the service graph is acyclic [39]. Task allocation proceeds by comparing $W$ over estimated $\hat{r}$ values and the hard constraints to satisfy (see, Section 5.4.3). The allocation $(t_{k_i,l}, w_{k_i}^W)$ is chosen according to a Task Allocation policy (TA) $\Pi$ that maps every state $k_i$ to the set $U(k_i)$ of admissible allocations with a certain probability distribution $\pi_{k_i}(u)$, i.e., $U(k_i)$:

\[ \pi_{k_i}(u) \]

Figure 8.4: The service provision problem as a labeled Directed Acyclic Graph (DAG) where nodes $s_5$ and $s_6$ are AND nodes.
8.4. Randomized Reinforcement Learning Algorithm

\[ \Pi \equiv \{ \pi_{k_i}(u), i = 0, 1, 2, \ldots, n \} \]  

It is assumed that: (i) once the action (i.e., allocation of a given task to a WS) has been chosen, the state next to \(k_i\), denoted \(k_i'\), is known deterministically, \(k_i' = f_{k_i}(u)\) where \(f\) is a one-to-one mapping from states and actions to a resulting state; (ii) different actions lead to different states; and (iii) as in [21], there is a special cost-free destination state; once the mediator has reached that state, the task allocation process is complete. Although the current discussion focuses on the deterministic case, extension to the stochastic case is discussed elsewhere in Chapter 2 due to format constraints.

8.4.2 Dealing with Task Concurrency

AND/OR graphs can be used to represent a process model including concurrent tasks that a WS composition needs to execute. An AND/OR graph \(G\) can be viewed as a generalization of a directed graph with a special node \(k_0\), called the initial (or root) node, and a nonempty set of terminal nodes. The start node \(k_0\) represents the given problem to be solved, while the terminal nodes correspond to possible solutions. The nonterminal nodes of \(G\) are of two types: OR and AND. An OR node can be solved in any one of a number of alternate ways, but an AND node is solved only when every one of its immediate subproblems is solved.

Figure 8.5 shows two examples of AND/OR graphs. The admissible actions in state \(k_i\) are \(U(k_i) = \{ u_1 = 1, u_2 = 2, u_3 = 3 \}\), the distribution \(\pi_{k_i}\) specifies three probabilities \(\pi_{k_i}(u_1), \pi_{k_i}(u_2),\) and \(\pi_{k_i}(u_3)\). On the below in Figure 8.5, an AND node, \(k_i\), is represented by a double circle. The admissible actions still remain the same, but there is no probability distribution because all available actions need to be executed in \(k_i\) in order to move to the next state.

In the deterministic case and if all the concurrent actions lead to the same state, the expected cost of a AND state \(k_i\) is given by

\[
V(k_i) = c(k_i, \bar{u}) + V(k_i', \bar{u}), \text{ with } \bar{u} = U(k_i) \text{ and } k_i \neq k_d
\]  

(8.1)

where \(k_i', \bar{u} = f_{k_i}(\bar{u})\) is the following state and \(c(k_i, \bar{u})\) is the cost incurred when concurrent tasks of \(\bar{u}\) are executed in state \(k_i\).

Equation (8.1) represents the special case where all the concurrent actions lead to the same state, i.e., \(\forall u_1 \in U(k_i), \forall u_2 \in U(k_i) \ f_{k_i}(u_1) = f_{k_i}(u_2)\) for all AND node \(k_i\). To extend our model to the most general case (see Figure 8.5), i.e., \(\exists(u_1 \in U(k_i) \text{ and } u_2 \in U(k_i)), f_{k_i}(u_1) \neq f_{k_i}(u_2)\) where \(k_i\) is an AND node, we assume that the costs corresponding to the execution of the actions are independent (i.e., execution time, the use of independent resources, ...). In this
8.4. Randomized Reinforcement Learning Algorithm

In this chapter, we consider only independent costs (i.e., additive costs), then
\[ c(k_i, \bar{u}) = \sum_{u \in \bar{u}} c(k_i, u) \]. In case costs are not independent, Equation (8.1) will integrate the relation between the costs.

A specific issue to be addressed when dealing with concurrent tasks is establishing the termination schemes. A termination scheme [145, 146] determines

![Figure 8.5: A representation of state \( k \) as OR and AND nodes.](image)

The extension of Equation (8.1) to concurrent actions leading to different states is given by

\[
V(k_i) = \sum_{u \in U(k_i)} (c(k_i, u) + V(k_{i', u})), \text{ with } k_{i', u} = f_{k_i}(u) \text{ and } k_i \neq k_d \quad (8.2)
\]
8.4. Randomized Reinforcement Learning Algorithm

when the action that follows the set of concurrent actions is executed. Three termination schemes are proposed by Rohanimanesh and Mahadevan. For instance, the termination scheme $\tau_{\text{any}}$ means that the next action is executed after any of the concurrent actions finishes, while the non-terminated actions are interrupted. Alternatively, in $\tau_{\text{all}}$ the mediator waits until all concurrent actions finish before proceeding to the following action. Other termination schemes can be defined by combining $\tau_{\text{any}}$ and $\tau_{\text{all}}$.

8.4.3 Computation of the Optimal Policy

The mediator begins with task allocation from the initial state and chooses from state $k_i$ the allocation of a WS $u$ to a task $t_{k_i,j}$ with a probability distribution $\pi_{k_i}(u)$, which aims to exploration. When the mediator chooses to allocate a task $t_{k_i,j}$ in state $k_i$, two cases can arise: (i) the current state $k_i$ is either a OR state and $t_{k_i,j}$ is a single task, or (ii) $k_i$ is an AND state and $t_{k_i,j}$ is a set of concurrent tasks. In the case of a single task, the mediator performs the allocation of the task $t_{k_i,j}$ to a WS $u$ and the associated cost $c(t_{k_i,j}, w_{u}^{WS})$ is incurred and is denoted, for simplicity $c(k_i, u)$ (note that the cost may also vary over time in a dynamic environment); the mediator then moves to the new state, $k_{i'}$. In the case of concurrent tasks, all tasks of $t_{k_i,j}$ are allocated in parallel. Depending to the termination scheme (i.e, $\tau_{\text{any}}$, $\tau_{\text{all}}$ or $\tau_{\text{continue}}$), the mediator incurs the associated cost, $c(k_i, u)$, and moves to the new state, $k_{i'}$. This allows the mediator to update the estimates of the cost, of the policy, and of the average cost until destination; these estimates will be denoted by $\hat{c}(k_i, i)$, $\hat{\pi}_{k_i}(i)$ and $\hat{V}(k_i)$. The RRL for an acyclic graph, where the states are ordered in such a way that there is no edge going backward (i.e., there exists no edge linking a state $k_{i'}$ to a state $k_i$ where $k_{i'}$ is a successor state of $k_i$ ($k_{i'} > k_i$), is as follows (a detailed treatment can be found in Chapter 2):

1. Initialization phase:
   - Set $\hat{V}(k_d) = 0$, which is the expected cost at the destination state.

2. Computation of the TA policy and the expected cost under exploration constraints:
   
   For state $k_i = (k_d - 1)$ to the initial state $k_0$, compute:
   - Choose a task $t_{k_i,j}$ to allocate to a WS $u$ with current probability estimate $\hat{\pi}_{k_i}(u)$,
8.4. Randomized Reinforcement Learning Algorithm

- if $t_{k_i,j}$ is a single task: observe the current cost $c(k_i, u)$ for performing this action and update its estimate $\hat{c}(k_i, u)$

  $$\hat{c}(k_i, u) \leftarrow c(k_i, u)$$  \hspace{1cm} (8.3)

- if $t_{k_i,j}$ is a set of concurrent tasks: the mediator allocates in parallel all tasks of $t_{k_i,j}$. The next decision epoch occurs (i.e., the moment when mediator can jump to the next state $k_i'$) depending on the termination scheme (i.e., $\tau_{\text{any}}, \tau_{\text{all}}$ or $\tau_{\text{continue}}$) chosen. The $\hat{c}(k_i, u)$ is updated by:

  $$\hat{c}(k_i, u) \leftarrow \sum_{u_j \in u} c(k_i, u_j)$$  \hspace{1cm} (8.4)

- Update the probability distribution for state $k_i$ as:

  $$\hat{\pi}_{k_i}(u) = \exp\left[ -\hat{\theta}_{k_i} \left( \hat{c}(k_i, u) + \hat{V}(k_i') \right) \right] \sum_{u' \in U(k_i)} \exp\left[ -\hat{\theta}_{k_i} \left( \hat{c}(k_i, u') + \hat{V}(k_i'') \right) \right],$$  \hspace{1cm} (8.5)

  where $k_i' = f_{k_i}(u)$, $k_i'' = f_{k_i}(u')$ and $\hat{\theta}_{k_i}$ is set in order to respect the prescribed degree of entropy at each state (see Equation (2.4) which can be solved by a simple bisection search). One can show that this probability distribution law for task allocation minimizes the expected cost (see Equation (2.1)) from the starting node to the destination node for a fixed exploration rate.

- Update the expected cost of state $k_i$:

  - if $k_i$ is an OR state:

    $$\begin{array}{l}
    \hat{V}(k_i) = \sum_{u \in U(k_i)} \hat{\pi}_{k_i}(u) \left[ \hat{c}(k_i, u) + \hat{V}(k_i') \right], \\
    \hat{V}(k_d) \leftarrow 0, \text{ where } k_d \text{ is the destination state}
    \end{array}$$  \hspace{1cm} (8.6)

    with $k_i' = f_{k_i}(u)$ and $k_i \neq k_d$

  - if $k_i$ is an AND state:

    $$\begin{array}{l}
    \hat{V}(k_i) \leftarrow \sum_{u \in U(k_i)} \hat{c}(k_i, u) + \hat{V}(k_i'), \\
    \hat{V}(k_d) \leftarrow 0, \text{ where } k_d \text{ is the destination state}
    \end{array}$$  \hspace{1cm} (8.7)

    with $k_i' = f_{k_i}(u)$ and $k_i \neq k_d$
8.5. Simulation Results

**Figure 8.6:** Success rate in terms of run number, for an exploration rate of 20%, and for the five methods (no exploration, actual $r$ known, $\epsilon$-greedy, naive Boltzmann, RRL).

Various approaches can be applied to update the estimated criterion $\hat{r}_u$; e.g., exponential smoothing leads to:

$$\hat{r}_u \leftarrow \alpha \bar{r}_u + (1 - \alpha)\hat{r}_u \quad (8.8)$$

where $\bar{r}_u$ is the observed value of the criterion for $w^W_u$ and $\alpha \in ]0, 1[ \text{ is the smoothing parameter. Alternatively, various stochastic approximation updating rules could also be used. The mediator updates its estimates of the criterion each time a WS performs a task and the associated cost is updated accordingly.}
8.5. Simulation Results

8.5.1 Experimental setup

Task allocation for the service provision problem displayed in Figure 8.4 was performed. A total of three distinct WS were made available for each distinct task. Each $w_{k,u}$ is characterized by its actual $r_u$, which is an indicator of the WS's performance over the optimization criterion (see, Section 8.4.1). In this simulation, it will simply be the probability of successfully performing the task ($1 - \text{probability of failure}$). In total, 42 WS are available to the Composer for task allocation. For all WS $u$, $r_u$ takes its value $\in [0, 1]$; for 70% of the WS, the actual $r_u$ is hidden (assuming it is unknown to the Composer) and its initial expected value, $\hat{r}_u$, is set, by default, to 0.3 (high probability of failure since the behavior
8.5. Simulation Results

![Figure 8.8: Average absolute difference between actual ($r$) and estimated ($\hat{r}$) criterion values in terms of run number, for three exploration methods ($\epsilon$-greedy, naive Boltzmann, RRL).](image)

of the WS has never been observed up to now), while actual $r_u$ value is available to the Composer for the remaining 30% (assuming these WS are well known to the Composer). Actual $r_u$ is randomly assigned from the interval $[0.5, 1.0]$ following a uniform probability distribution. It has been further assumed that $\hat{c}(t_i, w_u) = -\ln(\hat{r}_u)$, meaning that it is the product of the $r_u$ along a path that is optimized (this is a standard measure of the reliability of a system). After all tasks are allocated, the selected WS execute their allocated tasks according to their actual $r_u$ value (with failure $1 - r_u$). The estimated WS criterion $\hat{r}_u$ is then updated by exponential smoothing, according to Equation (8.8). In Equation (8.8), $\bar{r}_u$ equals 1 if $w_u$ is successful at executing the task it has been allocated, 0 otherwise. Estimated costs are of course updated in terms of the $\hat{r}_u$ and each time a complete allocation occurs, the probability distributions of choosing a WS are updated according to Equation (8.6) and Equation (8.7). 10,000 complete
8.5. Simulation Results

allocations were simulated for exploration rates 20%, and 30%.

8.5.2 Results

The RRL is compared to two other standard exploration methods, \( \epsilon \)-greedy and naive Boltzmann (see Chapter 2 for details), while tuning their parameters to ensure the same exploration level as for RRL. The success rate is defined as the proportion of services that are successfully completed (i.e., all tasks composing the service are allocated and executed successfully) and is displayed in Figures 8.6 and 8.7 in terms of the run number (one run corresponding to one complete assignment of tasks, criterion estimation and probability distribution update). Figures 8.6 and 8.7 show the RRL behaves as expected. Its performance converges almost to the success rate of the RRL in which all actual \( r \) are known from the outset (i.e., need not be estimated)—and indicate that exploration clearly helps by outperforming the allocation system without exploration (which has a constant 75% success rate). Figure 8.8 compares the three exploration methods by plotting the average absolute difference between actual \( r_u \) and estimated \( \hat{r}_u \) criterion values for a 30% exploration rate. Exploration is therefore clearly helpful when the environment changes with the appearance of new agents—i.e., exploration is useful for directing Composer behavior in dynamic, changing, and open architectures, i.e., in the SCA.
Chapter 9

Application to Supply Chain Management

Contents

9.1 Research Approach .............................................. 226
   9.1.1 ProDAOSS: a Methodology for Developing Service-
          Oriented MAS ............................................. 227
   9.1.2 Actor Collaboration in Supply Chain Management . 228
   9.1.3 The Services Approach in Supply Chain Management 229
   9.1.4 MAS in Supply Chain Management: a Service-Center
          Approach .................................................. 229
9.2 Outbound Logistics ............................................. 230
9.3 The Outbound Logistics Software Development: Pro-
      DAOSS Approach ........................................... 231
   9.3.1 Application Analysis ................................... 232
   9.3.2 Application Design .................................... 234
9.4 Related Work .................................................. 236

Motivations Today’s enterprise information systems have to match with their
operational and organizational environment. Unfortunately, software project
management methodologies are traditionally inspired by programming concepts
rather than by organizational and enterprise ones. In order to reduce as much
9.1. Research Approach

this distance, Agent-Orientation is increasingly emerging and has been the ob-
ject of more and more research over the last 10 years. Its success comes from
the fact that it better meets the increasing complexity and flexibility required to
develop software applications built in open-networked environments and deeply
embedded into human activities.

The gap between the traditional software engineering approaches and multi-
agent software systems using artificial intelligent concepts remains nevertheless
important. A series of development methodologies for designing multi-agent
systems (MAS) have appeared over the years; those methods have their own
characteristics and use various models to analyse the environment, to design the
system and also various languages to implement the proposed solution. This
chapter is part of the effort to dispose of methodologies covering the whole de-
velopment life cycle: from the analysis of industrial (multi-actor) cases to the
development of MAS software applications through a Process for Developing
Adaptable and Open Service Systems, called ProDAOSS.

A supply chain is the set of all actors and relations between them which par-
ticipate in the process of delivering value to a customer as a product or service.
It includes all the processes from raw materials to delivery and is viewed as a
network of information, materials and financial flows. In this chapter we par-
ticularly focus on the application of our software development methodology on
outbound logistics - i.e. the process related to the movement and storage of prod-
ucts and goods from the supplier to the end user - with a strong focus on trans-
portation. The idea underlying the developed collaborative application package
is to offer the most advanced services to optimize the transportation chain on a
global basis. The chapter will present the analysis and design stages of the ap-
plication package development. Part of this work is a joint work together with
Wautelet and Kolp.

The chapter is structured as follows. Section 9.1 discusses the research ap-
proach and main contributions of the chapter. Section 9.2 briefly presents out-
bound logistics as envisaged in the study. Section 9.3 overviews the application
development: the analysis and design stages are depicted in detail. Finally, Sec-
tion 9.4 presents the related work.

9.1 Research Approach

This section describes our research approach and the contributions of the chap-
ter. Section 9.1.1 reviews the ProDAOSS process. Section 9.1.2 describes the
9.1. Research Approach

notion of actor collaboration in supply chain management. Finally, Section 9.1.3 justifies the interest of a services approach in supply chain management.

9.1.1 ProDAOSS: a Methodology for Developing Service-Oriented MAS

Service-oriented computing is becoming increasingly popular when developing large and flexible software solutions. Since our goal is to dispose of a methodology for developing large industrial software application packages this was adopted in our process called ProDAOSS (i.e., Process for Developing Adaptable and Open Service Systems). Indeed, services are used as fundamentals to drive the software process. They are distinguished and documented at analysis stages (through the FaMOs framework, described in Chapter 6) then converted into service centers at design stage for an agent oriented implementation. More precisely:

- The organisation is firstly described as a set of organisational services in the Strategic Services Diagram;
- Organizational services are split in a series of goals, tasks and resources depicted in the Strategic Dependency and Rationale Diagrams;
- Organisational services realization paths are documented by the Dynamic Service Graph;
- Organisational services are designed as service centers in the architectural design discipline;
- Services’ realization environment is open and adaptable through the use of a reinforcement learning algorithm and a probabilistic reputation model (detailed respectively in Parts 2 and 3).

The contribution of this chapter are the instantiation of the whole ProDAOSS process to a case study in outbound logistics *.

*Outbound logistics is part of supply chain (management), this will be overviewed later in the chapter.
9.1.2 Actor Collaboration in Supply Chain Management

By nature, supply chain management is an interesting area for the development of industrial multi-actor software systems since it involves a series of collaborating or competing companies with tens of roles played by hundreds of individuals. The benefits that can be taken from such systems can lead to avoid waste of resources for all the involved actors. Collaborative decision will tend to avoid local equilibriums (at actor level) and wastes in the global supply chain optimisation, giving opportunities to achieve the greatest value that the chain can deliver at lowest cost (see [129, 155]). Such a result can only be achieved through actors’ collaboration. As noticed by [122], the term collaboration is confusing since various interpretations take place in the context of supply chains, they distinguish 3 overlapping levels of collaboration in real supply chains:

- **Information centralization** is the most basic technique of information sharing. Applied to outbound logistics it can be the shipper announcing its transportation needs or the carrier sharing all its planned transportations. Moyeu et al. distinguish information sharing from centralisation by the fact that the later is "the multi-casting in real-time and instantaneously of the market consumption information" [122] while the former is the sharing of the demand and supply information between companies;

- **Vendor Managed Inventory (VMI) and Continuous Replenishment Program (CRP)** are collaboration techniques in which retailers do not place orders because wholesalers use information centralization to decide when to replenish them;

- **Collaborative Planning, Forecasting and Replenishment (CPFR)** enhances VMI and CRP by incorporating joint forecasting. CPFR includes only two levels of supply chain, retailers and wholesalers, it allows companies to electronically exchange a series of written comments and data which include past sales trends, scheduled promotions and forecasts. It shares more information than only demand information. This allows the participants to coordinate joint forecasts by focussing on the differences in forecasts.

Our interpretation of the term collaboration in the context of this chapter is at first information centralisation, indeed by announcing the demand and supply concerning transportation (in terms of transport services, logistic requests, transportations, etc.), the actors collaborate for answering to the requirements of the other actors at best conditions. Indeed, collaboration on the basis of information
sharing is not incompatible with competition. Since an actor role can be played by different - probably competing - companies, the demander can use the best possible offer or the supplier can make interesting offers to dispose of optimally filled transportations. At second by sharing this information benefits can be issued in terms of global optimization. The whole outbound logistic chain can be optimized in terms of load balancing, cross dockings, tours, time, etc. Moreover, real-time information about the development of the services allows internal follow up and (re)optimization.

9.1.3 The Services Approach in Supply Chain Management

In the context of the development of a multi-actor collaborative application package we used a service-oriented approach for developing the MAS. This has been done following the ProDAOSS methodology. The services approach in the context supply chains is of particular interest because:

- Supply chain software and data collection will, in the future, take the form of a utility accessible on demand;
- Service-oriented architecture (SOA) could have a profound effect on the way supply chain software is designed, sold and implemented in the future. Common workflow based developments progressively leave place to generic customizable systems. In that perspective, SOA is designed to support rather than dictate business processes. This trend is for example also present in ERP systems development;
- Software developments’ integration is from strategic importance in nowadays business context, SOA with its inherent flexibility ease these procedures.

9.1.4 MAS in Supply Chain Management: a Service-Center Approach

At first, a supply chain can naturally and easily be conceived as an organization of actors played by a series of companies. Each of them could be represented (i.e., instantiated) as one or many agent(s); agent-oriented modelling is thus particularly well indicated. Such modeling is achieved, in this chapter, through an extension of i* [194]: the FaMOs framework.
Moreover, the Service Center Architecture (SCA) presented in Chapter 5 and applied in the context of outbound logistics envisages the dynamic allocation of tasks to competing or collaborating agents. Indeed, at design stage the application package is conceived as a multi-agent system through the use of the SCA which works as follow: when a service is requested by a particular agent (instance of the responsible actor documented in the Strategic Services Diagram), the series of task realizing the service (documented in the Dynamic Service Graph) is communicated through the environment to the other agents so that task-specialist agents (documented in the Strategic Dependency and Strategic Rationale Diagrams) are assigned to the task(s) they are able to fulfil by a mediator agent evaluating the best possible candidate (following defined criteria such as reputation, cost, disposal of resources, etc.). This is documented in detail in Section 9.3.2.

Such an architecture is consequently of particular interest in outbound logistics, the aspect of supply chain management we choose to develop here and documented into the next section.

9.2 Outbound Logistics

Outbound logistics is the process related to the movement and storage of products from the supplier to the end user. In the context of this chapter we mostly focus on transportation decisions, which will additionally provide information for better internal storage decisions. The actors of the supply chain play different roles in the outbound logistic flow. The producer will be a logistic client in its relationship with the raw material supplier, which will be considered as the shipper. The carrier will receive transportation orders from the shipper and deliver goods to the client, while relying on the infrastructure holder and manager. In its relation with the intermediary wholesaler, the producer will then play the role of the shipper and the wholesaler will be the client.

Figure 9.1 summarizes the material flows between the actors of the outbound logistics chain. The responsibilities corresponding to the different roles are:

- **Shipper**: has received an order from a client, and does a logistic request to a carrier for the delivery of that order.

- **Carrier**:
  - *The strategic planner*: decides on the services that are offered on the long term, on the use of infrastructure, on the logistic resources to
9.3 The Outbound Logistics Software Development: ProDAOSS Approach

Figure 9.1: Material flows in the outbound logistics chain

- **The scheduler**: orders transports to be realized, according to the strategic network and constraints, coordinates with the infrastructure manager and assigns logistic requests to those transports such that the delivery requirements are met.

- **Infrastructure manager**: holds the logistic infrastructure and coordinates with the carrier’s scheduler to offer the network for the planned transports.

9.3 The Outbound Logistics Software Development: ProDAOSS Approach

This section introduces the application of the ProDAOSS process onto outbound logistics. The development of the outbound logistics application package is however too large to be fully developed here. That is why we will, in the Strategic Services Diagram, document the entire application package in terms of organizational services and focus on only one aspect, *Manage Transports*, in the rest of the presentation. So that, the Strategic Rationale Diagram, the Dynamic Service Graph - at analysis stage - and the Service Center Architecture - at design stage - will only document this service.
9.3. The Outbound Logistics Software Development: ProDAOSS Approach

9.3.1 Application Analysis

For the application analysis, the ProDAOSS process uses the models included in the FaMOs framework. Firstly, we describe all the services included in the application package through the Strategic Services Diagram. Afterwards, the Strategic Rationale Diagram and the Dynamic Service Graph focus on the service Manage Transport.

9.3.1.1 Strategic Services Diagram

The Strategic Services Diagram (SSD) of Figure 9.2 introduces all the services of the application package. This view allows all project stakeholders to share a common aggregate view of the services including their dependency relationships. As evoked previously, due to a lack of space, we will, in the context of this chapter, focus on the Manage Transports service. To fulfill this service, the Customer depends on the Scheduler.

Figure 9.2: Strategic Services Diagram for Outbound Logistics

9.3.1.2 Strategic Rationale Diagram

The Strategic Rationale Diagram (SRD) of Figure 9.3 depicts the actors, goals, tasks and resources involved into the Manage Transport service realization. This diagram offers a complete static view of the actors’ dependencies for resources, goals and tasks. This view is from primary importance for designing the MAS
9.3. The Outbound Logistics Software Development: ProDAOSS Approach

since the agents that will instantiate the involved actors will be the ones that will be assigned the tasks realization into the Manage Service service center at design stage (more information is given into the next section). Seven actors are involved in the process of managing transports with the scheduler as the central one. The scheduler main task is to *plan a new horizon*: this is achieved by *creating transports* and assigning the *logistic requests* transmitted by the *Order Representative* to those transports (for more information see the realization paths in the paragraph below).

![Figure 9.3: Strategic Rationale Diagram for Manage Transport](image)

### 9.3.1.3 Dynamic Service Graph

Finally, the *Dynamic Service Graph* offers a complementary view to the SRD presented above by showing the service realization paths. This diagram is also of primary importance in the context of the service center architecture at design stage (see next section).

Figure 9.4 represents fulfilment paths for the Manage Transport service. Each node is a step in service provision and each edge corresponds to the execution of a task $t_k$ by a specialist agent $a_{k,u}^{SA}$, where $u$ ranges over specialists that can execute $t_k$ according to the criteria set in the service request. A function of the criteria set $c(t_k, a_{k,u}^{SA})$, labels each edge and represents the QoS ad-
9.3. The Outbound Logistics Software Development: ProDAOSS Approach

Figure 9.4: Dynamic Service Graph for Manage Transport

<table>
<thead>
<tr>
<th>Init. state</th>
<th>Dest. state</th>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$t_0$</td>
<td>It modifies requests allocation</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$t_1$</td>
<td>It verifies the logistic requests or transportation services allocation.</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$t_2$</td>
<td>It modifies requests allocation.</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$S_4$</td>
<td>$t_3$</td>
<td>It creates transport.</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$S_5$</td>
<td>$t_4$</td>
<td>It links the logistic requests to a series of transports.</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$S_6$</td>
<td>$t_5$</td>
<td>It evaluates the schedule feasibility.</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$S_7$</td>
<td>$t_6$</td>
<td>It modifies transports.</td>
</tr>
<tr>
<td>$S_7$</td>
<td>$S_5$</td>
<td>$t_7$</td>
<td>It modifies logistic requests attribution to transports.</td>
</tr>
</tbody>
</table>

Table 9.1: Description of the tasks included in the graph

Vertised by the specialist $a_{k,u}^{SA}$ for performing the task $t_k$. Note that different paths offering different QoS are available. Indeed, as shown in Table 9.1 path $<t_0, t_1, t_2, t_3, t_4, t_6, t_7>$ offers alternative ways of fulfilling the service. The following list gives a description of each task used in the graph; tasks are here limited to a scheduler agent:

9.3.2 Application Design

The ProDAOSS process proposes, at architectural design stage, an open, distributed, self-organizing and service-oriented MAS architecture called Service Center Architecture (SCA) first defined in Chapter 5 and instanced here for an application to outbound logistics. SCA organizes agents into groups, called ser-
9.3. The Outbound Logistics Software Development: ProDAOSS Approach

Each service center specializes in the provision of a single service. Within each center, a task allocation and reputation algorithm are integrated in the proposed architecture.

As presented in Figure 9.5, the logical architecture proposed in this section falls four layers.

![Logical architecture and overview of the requirements-driven service-based architecture](image)

Figure 9.5: Logical architecture and overview of the requirements-driven service-based architecture

### 9.3.2.1 Upper Layer: User Client

The upper layer represents user clients (humans or applications) that interact with the middle layer and transmit client requests.

### 9.3.2.2 Middle Layer: Service Center

The middle layer contains the various service centers based on the Service Center Architecture connected by a communication layer. SCA groups services into service centers (SC). Each SC contains all distinct tasks, executed by Task-Specialist agents (TS), needed to provide a service corresponding to a service.
9.4. Related Work

request originating from the user. As represented in Figure 9.5, five special agents are also present in each SC: the Service Request Manager, the Service Mediator, the Service Process Modeler, the Service Reputation and the Service Center Discovery. Report to Section 5.2 for a detailed definition of the SCA and the special agents.

The SC representing the service Manage Transport and the corresponding Dynamic Service Graph illustrating its process model are shown in Figure 9.5. For each task (i.e., each edge composing the Dynamic Service Graph) of the service, we have a set of TS able to execute this task. For the Manage Transport service, we have seven tasks and thus seven set of TS. Each TS is characterized by a reputation score computed by the service reputation. The Mediator uses this reputation score and others quality constraints given by the users in their service requests in order to allocate tasks to fulfil the service.

9.3.2.3 Bottom Layer: Technical/Algorithm and Service Process

The bottom layers are the technical/algorithm layer and the service process layer. Technical/algorithm layer contains all the algorithms and technical processes used and executed by the special agent of the service center, i.e., the service process modeler executes a process model algorithm to build the service process corresponding to the global goal of its service center, the service mediator uses a task allocation algorithm to allocate tasks to TS and the service reputation uses the reputation algorithm to compute the reputation score of the TS. By keeping these algorithms and others technical processes in a lower level, we keep independent these technical features from the service center and these technical aspects can be easily changed and adapted without any changing to the higher level, i.e., the service center itself.

The service process layer contains business processes, i.e., compositions of tasks in simple services or compositions of simple services in a more complex service. For the service Manage Transport, this layer contains its process model represented by the Dynamic Service Graph as shown in Figure 9.5. These layers communicate between each other in order to answer clients’ needs.

9.4 Related Work

The application of MAS technology onto supply chain management has already driven plenty of literature. Indeed, the subject covers several aspects that is why
9.4. Related Work

we will focus here on the application of MAS development processes onto supply chain cases.

[137, 136] focuses on the socially-driven approach of the Tropos software development methodology in order to build up an agent-based information system prototype in supply chain management. Their approach however remains limited since their supply chain model is (too) simplified and mostly focuses on the application of architectural styles and design patterns at architectural and detailed design levels rather than on the whole development life cycle.

[64] proposes a generic process-centered methodological framework called Multi-Agent Supply Chain Framework (MASCF), to simplify MAS development for supply chains applications. Their idea is to map elements from the Supply Chain Operations Reference (SCOR, see [156]) model to the Gaia development methodology [191] for multi-agent supply chain system development. Mapping the SCOR and the Gaia concepts is interesting for analysts but it supposes modelling supply chain processes using this formalism while it would be strain forward to dispose of a fully integrated generic and flexible methodology for MAS development in supply chain management. That is why we choose to extend a current methodology with elements and axioms adapted for such flexible industrial developments rather than to map existing supply chain modelling elements to software development elements.

In [62], the idea that of an electronic marketplace (eMarketplace) as an architectural model to develop collaborative supply chain management and integration platform is defended. In their architecture, the eMarketplace exists as a collection of economically motivated software agents of service-oriented cooperative distributed systems. The authors discuss how coordination approaches as auctions and multi-issues negotiation can be developed in the context of this eMarketplace. The objective here is to enable business entities to obtain efficient resource allocation while preserving long-term relationships. This idea is also present in our software system but little developed here since the chapter mostly focus on the methodology and its application on industrial cases. Our software system presentation at the light of their ideas would nevertheless constitute an interesting perspective.
Part VI

Conclusion
This thesis presents a process for developing adaptable and open service systems called ProDAOSS that offers a broader perspective of the traditional Tropos methodology for designing adaptable and open service systems (AOSS). The process is composed of four complementary core disciplines: Organizational Modeling, Requirements Engineering, Architectural Design and Detailed Design. These disciplines repeated iteratively provide continuous evolving dimensions to completely fulfill the development of AOSS involving users in an intensive manner. For each discipline, a generic description has been given, the fundamental concepts and notions were introduced. This conclusion presents a summary of the dissertation and its further work.
10.1 Thesis Summary

The main contribution of this dissertation is the process itself. It integrates the agent concepts extracted and generalized from agent-oriented modeling language i* and uses service elements as scope elements to drive the development process. The process is generic and flexible enough to be adapted to any MAS development methodology.

10.1.1 Analysis Level

At organizational modeling and requirements engineering stages, the process refines the traditional i* approach by including the FaMOs framework which introduces new models and refines existing ones. The FaMOs framework furnishes multiple views for model-driven software development. This framework includes a high level service view allowing to identify in a non redundant way the services that must be fulfilled by the system as well as elements raising or lowering quality of service (QoS). Such framework is much broader than only allowing organisational modeling and requirements engineering. Indeed the framework can be extended for risk, quality or time management.

10.1.2 Design Level

At architectural design stage, ProDAOSS defines the Service Center Architecture. Service Center Architecture involves service centers composed by five special agents: the service mediator, the service center discoverer, the service process modeler, the service request manager, and the service reputation. The principal agent is the service mediator which dynamically allocates tasks composing the requested service to task specialist agents capable of executing the given tasks according to specific criteria. The service mediators use a reinforcement learning algorithm combining exploitation with exploration to ensure both the use of acquired knowledge about the actual performance of task specialists and the adaptability to changing environment conditions. Task allocation is thus dynamic, and is driven by multiple criteria, including quality of service, deadline, reputation, cost, and additional explicit user preferences. The other special agents treated in this chapter are the service process modeler and service reputation. The first agent is responsible of the creation and the updating of the process model. Indeed, it generates a graph representing the process model from the service-center goal expressed in first-order logic and the data’s on the tasks and the task specialists. The second agent allows to model and analyze the task
specialists behaviors by computing their reputation scores. It uses an extension of the probabilistic reputation model proposed in a previous chapter by adding a dynamic dimension, through a damping factor, allowing the reputation scores to evolve over time.

At detailed design stage, ProDAOSS includes a task allocation algorithm as well as a reputation algorithm for the development of the modeled problem converted in an architectural solution. In this dissertation, we present a unified reinforcement learning framework integrating exploitation and exploration for continual exploration. This framework is used by the mediator as task allocation algorithm. As reputation algorithm, we describe a general procedure allowing to estimate reputation scores characterizing a provider’s quality of service, based on transactions between consumers (raters) and providers. As a by-product, some essential features of the raters, such as their reactivity and their bias, are also estimated.

10.1.3 Thesis Organization

This dissertation is organized as follows:

- Chapter 1 formulates the thesis by motivating the advantages of service-oriented computing and MAS, and introducing the principal contributions, the validation, and limitations of this thesis.

- Chapter 2 presents a RL framework allowing to tune continual exploration in an optimal way by integrating exploration and exploitation in a common framework.

- Chapter 3 describes two extensions of the basic RL framework introduced in Chapter 2. The first extension introduces a global entropy at the level of the entire network rather than a local entropy at the level of the nodes as made in the basic framework. The second extension introduces a hierarchy in the learning process.

- Chapter 4 introduces probabilistic models of reputation based on a simple consumer-provider interaction model.

- Chapter 5 proposes a MAS architecture called Service Center Architecture, consisting of as many service centers as there are requests of distinct services.
10.2. Further Work

- Chapter 6 introduces an analysis framework called FaMOs providing a multiple view process for conducting a software project through a model driven approach.
- Chapter 7 describes the ProDAOSS process, its structure and its life cycle.
- Chapter 8 illustrates an application of the ProDAOSS framework in the context of web services.
- Chapter 9 proposes to apply ProDAOSS to a second context: an industrial case study in outbound logistics.
- Chapter 10 is the conclusion.

10.2 Further Work

This section introduces the future developments that can be made on the basis of this thesis. Future work is separated on four levels, the first level introduces the work that can be made to improve the process itself, the second, third and fourth introduce respectively the possible evolutions of the FaMOs framework, the Service Center Architecture and finally, the RL framework and reputation model.

10.2.1 Process ProDAOSS

The process description contributes to the definition of service-oriented MAS development. This dissertation focuses on the process description by conceptualizing a framework allowing to build MAS using a services approach. It models and introspects the process along different complementary stages. Nevertheless, some important improvements can be identified.

Firstly, we only consider the design of cooperative MAS. Indeed, MAS may be either cooperative or competitive. In a cooperative MAS, the agents cooperate together in order to achieve common goals. Inversely, a competitive MAS is composed of agents that pursue personal goals and defend their own interests. The design of competitive MAS can be a future development.

The process needs to gain experience with its use. In this dissertation, we have applied it on two case studies. By doing so, we have tailored the process to particular projects and shown how our framework can help the design of MAS. However, it should be tested on more case studies and knowledge base should
10.2. Further Work

evolve from project to project. For example, it can be applied to the context of Enterprise Ressource Planning which is an interesting growing field in service-oriented computing.

The methodology is mostly designed for industrial software projects involving lots of agents. It is especially suited for companies information system development rather than for stand alone developments. Another further work can be an adaptation of the process to handle this issue.

Finally, a complete CASE tool including the multiple stages of the process ProDAOSS can be implemented.

10.2.2 FaMOs Framework

The FaMOs framework as presented is much broader than only allowing organisational modeling and requirements engineering. Indeed the framework can be extended for risk, quality or time management. The aim is consequently to create a complete software project management dimension for Tropos based on the models of the framework; this work is in progress; a CASE Tool is also under development including only the FaMOs framework.

Future research directions will include systematic forward engineering rules based on the models in the framework. This work has partially been done in [47] but needs to be extended to include the benefits of dynamic models as the Strategic Services Graph. Such systematic approach will provide guidelines to software designers to facilitate the framework adoption.

10.2.3 Service Center Architecture

We define the Service Center Architecture as an open, distributed, service-oriented MAS architecture capable of dynamic adaptation in response to the changing environment. Service Center Architecture proposes the corresponding requirements: (i) It is a simple architecture minimizing the variety of interactions between the participating agents. (ii) Internal functions such as task allocation, reputation computation, etc; ought to remain outside the responsibility of the entering and leaving agents to avoid bias. The architecture integrates therefore a special class of agents that manage these internal functions and that do not leave the system. (iii) Since there is no guarantee that agents will execute tasks at performance levels advertised by the task specialists, reputation calculation and task allocation should be grounded in empirically observed agent performance and such observation be executed by internal agents. (iv) Varying QoS
requests and change in the availability of agents require task allocation to be au-
tomated and driven by QoS, reputation scores and other relevant considerations
(e.g., deadlines). (v) To ensure continuous optimization of system operation,
task allocation within the architecture involves continuous observation of agent
performance, the use of available information to account for agent reputation,
and exploration of new options to avoid excessive reliance on prior information.

Further work will investigate the other specialist agents of the architecture,
i.e., service center discoverer, service process modeler and service request man-
ger. Indeed, this dissertation describes in details the behavior of the service
mediator and the service reputation agent. It proposes also a simple algorithm
used by service process modeler to build the process model from the global goal
of the service. This algorithm can be improved in order to take into account
tasks concurrency, macro and primitive action, etc. The behavior of the service
discoverer and the request manager is not described in this dissertation. We can
also investigate the inclusion in the architecture of a standard protocol (e.g., Con-
tractNet [167]) used by the task specialists and the specialist agents to interact.

Another important further work is the implementation of a generic platform
based on the service center architecture. This platform can include the chosen
specialist agents, and the implementation of the task allocation and the reputation
algorithms.

10.2.4 RL Framework and Reputation Model
10.2.4.1 RL Framework
In term of RL, we have presented a model integrating continual exploration and
exploitation in a common framework and shown that the Boltzmann strategy is
optimal within this framework. The exploration rate is controlled by the entropy
of the choice probability distribution defined on the states of the system (i.e.,
local entropy). We have also investigate (in a first extension of the basic frame-
work) the setting of a global entropy defined on the whole network, instead of as-
signing the entropy at the state (or node) level. When no exploration is performed
(zero entropy on each node), the model reduces to the common value-iteration
algorithm computing the minimum cost policy. On the other hand, when full
exploration is performed (maximum entropy on each node), the model reduces
to a “blind” exploration, without considering the costs. The main drawback of
the approach is that it is computationally demanding since it relies on iterative
procedures such as the value-iteration algorithm and a line search algorithm is
performed at each node for controlling the entropy. To handle this drawback,
the second extension of the basic framework introduces the hierarchical reinforcement learning. This extension permits to decompose a large problem into multiple small (sub)problems.

Further work could investigate alternative cost formulations, such as the “average cost per step”. Since the optimal expected cost is the minimum among all the potential policies, it defines a dissimilarity measure between the states of the graph. Further work could study the properties of this dissimilarity measure, which generalizes the Euclidean commute-time distance between nodes of a graph, as investigated in [57, 153]. We also plan to investigate mixed, randomized, strategies based on the randomized shortest-paths (RSP) instead of the usual minimax in game playing. Indeed, in a two-person game, it is unrealistic to assume that the opponent is completely rational, as minimax does. Therefore, it could be interesting to model the opponent’s behavior by a RSP strategy instead of a pure minimax one, which leads to mixed minimax strategies. Finally, computing dissimilarities between strings or sequences based on a RSP strategy, which will account for all the alignments between the two sequences instead of the single shortest one, could also be studied.

10.2.4.2 Reputation Model

We have proposed a general procedure allowing to estimate reputation scores characterizing a provider’s quality of service, based on transactions between consumers (raters) and providers. As a by-product, some essential features of the raters, such as their reactivity and their bias, are also estimated.

The procedure is based on a probabilistic model of consumer-provider interactions whose parameters are estimated by a variant of the expectation-maximization algorithm. Computer simulations show that the model is able to retrieve the correct parameters, much more accurately than simply taking, as a measure of reputation, the average of the ratings for each provider.

Further work could investigate the adaptation and the application of the proposed models to the computation of importance scores in social networks or web pages. Indeed, the probabilistic models of reputation (PMR models) could be adapted in order to provide importance scores instead of reputation scores. One key difference between these two features is the crucial impact of the number of ratings on the importance score. A node is considered as important if both the received ratings and the number of ratings are high. We will therefore have to integrate the influence of the number of ratings in our PMR models.
Part VII

Appendix
Appendix A

Determination of the optimal policy under local entropy constraints

Consider an agent trying to reach a destination state $d$ by choosing actions according to a policy $\Pi$. The expected cost $V_{\pi}(k)$ for reaching $d$ from any state $k$ is provided by Equation (2.2), rewritten here:

\[
\begin{align*}
V_{\pi}(k) &= \sum_{i \in U(k)} \pi_k(i) \left[ c(k, i) + V_{\pi}(k') \right], \text{ with } k' = f_k(i) \text{ and } k \neq d \\
V_{\pi}(d) &= 0, \text{ where } d \text{ is the destination state}
\end{align*}
\]

(A1)

The goal is to determine the policy that minimizes this expected cost, when starting from state $k_0$ and under entropy constraints (2.4). We therefore introduce
the following Lagrange function, taking all the constraints into account:

\[ L = V(k_0) + \sum_{k \neq d} \lambda_k \left( V(k) - \sum_{i \in U(k)} \pi_k(i) [c(k, i) + V(k')] \right) \]

\[ + \lambda_d(V(d) - 0) + \sum_{k \neq d} \mu_k \left( \sum_{i \in U(k)} \pi_k(i) - 1 \right) \]

\[ + \sum_{k \neq d} \eta_k \left( \sum_{i \in U(k)} \pi_k(i) \log \pi_k(i) + E_k \right) \]  

(A2)

with \( k'_i = f_k(i) \). Differentiating this Lagrange function with respect to the choice probabilities, \( \partial L/\partial \pi_l(j) \), and setting the result equal to 0 provides

\[ -\lambda_l(c(l, j) + V(k'_j)) + \mu_l + \eta_l(\log \pi_l(j) + 1) = 0 \]  

(A3)

Multiplying Equation (A3) by \( \pi_l(j) \) and summing over \( j \in U(l) \) yields

\[ -\lambda_l V(l) + \mu_l + \eta_l(-E_l + 1) = 0 \]  

(A4)

from which we deduce \( \mu_l = \lambda_l V(l) + \eta_l(E_l - 1) \). Replacing \( \mu_l \) by its expression in (A3) provides

\[ -\lambda_l(c(l, j) + V(k'_j) - V(l)) + \eta_l(\log \pi_l(j) + E_l) = 0 \]  

(A5)

Defining \( \theta_l = -\lambda_l/\eta_l \) and extracting \( \log \pi_l(j) \) from (A5) yields

\[ \log \pi_l(j) = -\theta_l(c(l, j) + V(k'_j) - V(l)) - E_l \]  

(A6)

or, equivalently,

\[ \pi_l(j) = \exp \left[ -\theta_l(c(l, j) + V(k'_j)) \right] \exp [\theta_l V(l) - E_l] \]  

(A7)

Summing this last equation over \( j \in U(l) \) and remembering that \( k'_j \) depends on \( j \) \( (k'_j = f_l(j)) \) allows us to compute the second factor of the right-hand side of Equation (A7):

\[ \exp [\theta_l V(l) - E_l] = \left( \sum_{j \in U(l)} \exp \left[ -\theta_l(c(l, j) + V(k'_j)) \right] \right)^{-1} \]  

(A8)
Replacing (A8) in Equation (A7) yields

\[
\pi_l(i) = \frac{\exp \left[ -\theta_l(c(l, i) + V(k'_i)) \right]}{\sum_{j \in U(l)} \exp \left[ -\theta_l(c(l, j) + V(k'_j)) \right]}
\] (A9)

Finally, expressing that each probability distribution has a given entropy,

\[-\sum_{i \in U(l)} \pi_l(i) \log \pi_l(i) = E_l,\]

allows us to compute the value of \( \theta_l \) in terms of \( E_l \). This defines the optimal policy; it can be shown that it is indeed a minimum.

Differentiating the Lagrange function with respect to the \( V(k) \) provides dual equations allowing to compute the Lagrange multipliers.
Appendix B

Convergence of the iterative updating rule

Before proving the convergence of the iterative updating rule (2.11)-(2.12), let us prove a preliminary lemma that will be useful later.

Indeed, we will first show that minimizing the linear form \( \sum_i p_i x_i \) in terms of \( p_i \), subject to entropy and “sum-to-one” constraints leads to the following functional form:

\[
p_i = \frac{\exp[-\theta x_i]}{\sum_j \exp[-\theta x_j]} \quad \text{(B1)}
\]

where \( \theta \) is chosen in order to satisfy \( -\sum_i p_i \log p_i = E \) (entropy constraint). This probability distribution has exactly the same form as (2.11), in the proposed algorithm. This lemma shows that using the distribution (B1) automatically minimizes the linear form \( \sum_i p_i x_i \).

To this end, let us introduce the Lagrange function

\[
L = \sum_i p_i x_i + \lambda \left( \sum_i p_i \log p_i + E \right) + \mu \left( \sum_i p_i - 1 \right) \quad \text{(B2)}
\]

and compute the differential of \( L \) with respect to \( p_j \), \( \partial L / \partial p_j = 0 \),

\[
x_j + \lambda \log p_j + \lambda + \mu = 0 \quad \text{(B3)}
\]

Now, by defining \( \theta = \lambda^{-1} \), Equation (B3) can be rewritten as

\[
p_i = \exp[-(1 + \theta \mu)] \exp[-\theta x_i] \quad \text{(B4)}
\]
Summing Equation (B4) over $i$ and using the fact that $\sum_j p_j = 1$ allows us to compute the value of the first factor of the right-hand side of Equation (B4) and finally obtain the expected result from (B4):

$$p_i = \frac{\exp[-\theta x_i]}{\sum_j \exp[-\theta x_j]} \quad (B5)$$

where $\theta$ is chosen in order to satisfy $-\sum_i p_i \log p_i = E$ (entropy constraint). This proves our preliminary lemma.

Now, we are ready to prove the convergence of (2.11)-(2.12) by induction. First, we show that if the $\hat{V}(k)$ have been decreasing (less than or equal to) up to the current time step, then the next update will also decrease $\hat{V}(k)$. Then, since the first update necessarily decreases $\hat{V}(k)$, the following updates will also decrease the value of $\hat{V}(k)$. Now, since $\hat{V}(k)$ is decreasing at each update and cannot be negative, this sequence must converge.

Thus, we first show that if all the $\hat{V}(k)$ have been decreasing up to the current time step $t$, then the next update will also decrease $\hat{V}(k)$. At each time step $t$, one state $k$ is updated in turn, asynchronously. Now, suppose that we decide to update state $k$ at time step $t$. If $\hat{V}^t(k)$ and $\hat{\pi}^t_k(i)$ denote the estimate of respectively the expected cost and the transition probability for state $k$ at time step $t$, the update of $\hat{V}(k)$, $\hat{\pi}^t_k(i)$ is given by

$$\begin{align*}
\hat{\pi}^t_l(j) &= \hat{\pi}^{t-1}_l(j), \text{ for all } l \neq k \text{ and } j \in U(l) \\
\hat{V}^t(l) &= \hat{V}^{t-1}(l), \text{ for all } l \neq k \\
\hat{\pi}^t_k(i) &= \frac{\exp\left[-\theta_k \left(c(k, i) + \hat{V}^t(k'_i)\right)\right]}{\sum_{j \in U(k)} \exp\left[-\theta_k \left(c(k, j) + \hat{V}^t(k'_j)\right)\right]}, \text{ for all } i \in U(k) \\
\hat{V}^t(k) &= \sum_{i \in U(k)} \hat{\pi}^t_k(i) [c(k, i) + \hat{V}^t(k'_i)], \text{ with } k'_i = f_k(i) \text{ and } k \neq d
\end{align*} \quad (B6)$$

Notice that all the $\hat{\pi}^t_k$ verify the entropy as well as sum-to-one constraints at any time $t$. Now, denote by $\tau < t$ the time of the latest update of state $k$ before
APPENDIX B. CONVERGENCE OF THE ITERATIVE UPDATING RULE

time $t$. We easily find

$$\hat{V}^t(k) = \sum_{i \in U(k)} \hat{\pi}^t_k(i) [c(k, i) + \hat{V}^t(k')] \quad (B7)$$

$$\leq \sum_{i \in U(k)} \hat{\pi}^t_k(i) [c(k, i) + \hat{V}^t(k')] \quad (B8)$$

$$\leq \sum_{i \in U(k)} \hat{\pi}^t_k(i) [c(k, i) + \hat{V}^\tau(k')] \quad (B9)$$

$$= \hat{V}^\tau(k) \quad (B10)$$

The passage from Equation (B7) to (B8) follows from the preliminary lemma, while the passage from (B8) to (B9) results from the assumption that the $\hat{V}(k)$ have been decreasing up to the current time step $t$. This proves the first part.

Now, initially, an arbitrary initial policy, $\hat{\pi}_k(i)$ for all $i$ and $k$, verifying the exploration rate constraints (2.8) is chosen, and the corresponding expected costs until destination $\hat{V}^0(k)$ are computed by solving the set of linear equations (2.2). Therefore, initially ($t = 0$),

$$\hat{V}^0(k) = \sum_{i \in U(k)} \hat{\pi}^0_k(i) [c(k, i) + \hat{V}^0(k')]$$

Thus, at the beginning of the updating process ($t = 1$), we first update, say, state $k_0$ and adjust $\hat{\pi}_{k_0}(i), \hat{V}(k_0)$ according to

$$\hat{\pi}^1_{k_0}(i) = \frac{\exp \left[ -\theta_{k_0} \left( c(k_0, i) + \hat{V}^1(k') \right) \right]}{\sum_{j \in U(k_0)} \exp \left[ -\theta_{k_0} \left( c(k_0, j) + \hat{V}^1(k') \right) \right]}, \text{ for all } i \in U(k_0) \quad (B11)$$

and

$$\hat{V}^1(k_0) = \sum_{i \in U(k_0)} \hat{\pi}^1_{k_0}(i) [c(k_0, i) + \hat{V}^1(k')] \quad (B12)$$

$$\leq \sum_{i \in U(k_0)} \hat{\pi}^0_{k_0}(i) [c(k_0, i) + \hat{V}^0(k')] \quad (B13)$$

since for every node $k' \neq k_0, \hat{V}^1(k') = \hat{V}^0(k')$ (no update has yet occurred for the nodes $\neq k_0$). The assumption is thus clearly verified: all the $\hat{V}(k)$ are
decreasing up to the current time step $t = 1$. Consequently, the next update will certainly decrease $\hat{V}(k)$, as well as all the subsequent updates.

Thus, if we define the decrease of $\hat{V}(k)$ as $\Delta \hat{V}^t(k) = \hat{V}^{t-1}(k) - \hat{V}^t(k) \geq 0$, since $\hat{V}(k)$ cannot become less than the cost of the shortest path $C$, we have $\hat{V}^\infty(k) = \hat{V}^0(k) - \sum_{t=1}^{\infty} \Delta \hat{V}^t(k) \leq C$ so that $\Delta \hat{V}^t(k) \to 0$ as $t \to \infty$. Therefore, the updating process converges.
Appendix C

Computation of the expected cost-to-go in terms of the fundamental matrix

Let us start from Equation (3.3) which is restated here,
\[ \mathbf{v} = \text{diag}(\mathbf{PC}^\top) + \mathbf{Pv} \]

Thus, we have
\[ \begin{bmatrix} \tilde{\mathbf{v}}_0 \\ 0 \end{bmatrix} = \text{diag} \left( \begin{bmatrix} \mathbf{Q} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \mathbf{D}^\top \\ \mathbf{s}^\top \\ 0 \end{bmatrix} \right) + \begin{bmatrix} \mathbf{Q} \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{v}}_0 \\ 0 \end{bmatrix}, \] (D1)
from which we easily deduce
\[ \tilde{\mathbf{v}} = \text{diag}(\mathbf{QD}^\top + \mathbf{rs}^\top) + \mathbf{Qv} \] (D2)

Isolating \( \tilde{\mathbf{v}} \) provides us with the required result:
\[ \begin{align*}
\tilde{\mathbf{v}} &= (\mathbf{I} - \mathbf{Q})^{-1} \text{diag}(\mathbf{QD}^\top + \mathbf{rs}^\top) \\
&= \mathbf{N} \text{diag}(\mathbf{QD}^\top + \mathbf{rs}^\top) \quad (D3)
\end{align*} \]

This equation expresses the expected cost-to-go in terms of the fundamental matrix, the transition probabilities and the cost matrix.
Appendix D

Determination of an optimal policy

The goal here is to determine the set of transition probabilities $\pi \equiv \{p_{kk'}; k = 1, 2, \ldots, (n-1); k' = 2, \ldots, n\}$ that minimizes the expected cost, when starting from state 1 and under entropy constraints (3.10). We therefore introduce the following Lagrange function, taking all the constraints into account,

$$L = V(1) + \sum_{k \neq n} \lambda_k \left[ V(k) - \sum_{k' \in S(k)} p_{kk'} \left( c(k, k') + V(k') \right) \right] + \lambda_n (V(n) - 0) + \sum_{k \neq n} \mu_k \left[ \sum_{k' \in S(k)} p_{kk'} - 1 \right] + \eta \left[ \sum_{k \neq n} u_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'} + H \right]$$  \hfill (E1)

Differentiating this Lagrange function in terms of the transition probabilities, $\partial L / \partial p_{ll'}$, and equating to zero gives

$$-\lambda_l (c(l, l') + V(l')) + \mu_l + \eta u_l (\log p_{ll'} + 1) - \eta \sum_{k \neq n} (\partial u_k / \partial p_{ll'}) h_k = 0$$  \hfill (E2)

where $h_k = -\sum_{k' \in S(k)} p_{kk'} \log p_{kk'}$. Extracting $\log p_{ll'}$ from this equation provides

$$\eta u_l \log p_{ll'} = -\mu_l - \eta u_l + \lambda_l (c(l, l') + V(l')) + \eta \sum_{k \neq n} (\partial u_k / \partial p_{ll'}) h_k$$  \hfill (E3)
Then, exponentiating this last equation provides

$$p_{ll'} = \exp \left[ \frac{-\mu_l - \eta u_l}{\eta u_l} \right] \exp \left[ \frac{\lambda_l(c(l, l') + V(l')) + \eta \sum_{k \neq n} (\partial u_k / \partial p_{ll'}) h_k}{\eta u_l} \right]$$

(E4)

Summing the equation over $l' \in S(l)$ allows us to compute the first factor of the right-hand side:

$$\exp \left[ \frac{-\mu_l - \eta u_l}{\eta u_l} \right] = \sum_{l' \in S(l)} \exp \left[ \frac{\lambda_l(c(l, l') + V(l')) + \eta \sum_{k \neq n} (\partial u_k / \partial p_{ll'}) h_k}{\eta u_l} \right]^{-1}$$

(E5)

By replacing (E5) in Equation (E4) and defining $\theta_k = -\lambda_k$, we finally obtain

$$p_{kk'} = \sum_{l' \in S(k)} \exp \left[ -\frac{\theta_k}{\eta u_k} (c(k, k') + V(k')) + \frac{1}{u_k} \sum_{l \neq n} (\partial u_l / \partial p_{kl'}) h_l \right]$$

$$\sum_{l' \in S(k)} \exp \left[ -\frac{\theta_k}{\eta u_k} (c(k, l') + V(l')) + \frac{1}{u_k} \sum_{l \neq n} (\partial u_l / \partial p_{kl'}) h_l \right]$$

(E6)

Now, differentiating $L$ in terms of the expected costs, $\partial L / \partial V(l), l \neq 1$, and equating to zero allows us to compute the Lagrange multipliers $\lambda_l$,

$$\lambda_l = \sum_{k \in P(l)} p_{kl} \lambda_k, \text{ for } l \neq 1,$$

(E7)

$P(l)$ being the set of nodes from which node $l$ is accessible in one step (the predecessors of $l$).

For the initial state, we obtain (recall that we assumed in Section 3.1.3 that the initial state has no predecessor)

$$\lambda_1 = -1$$

(E8)

Defining $\theta_k = -\lambda_k$ provides

$$\left\{ \begin{array}{l} \theta_l = \sum_{k \in P(l)} p_{kl} \theta_k, \text{ for } l \neq 1 \\ \theta_1 = 1, \text{ for initial state } 1 \end{array} \right.$$
Rewriting this last equation (E9) in matrix form gives $\theta = P^T\theta + e_1$; therefore when considering only the transient states, $\tilde{\theta} = Q^T\bar{\theta} + \bar{e}_1$. In other words, we have

$$\tilde{\theta} = (I - Q^T)^{-1}\bar{e}_1 \quad (E10)$$

By comparing Equation (E10) with (3.5), we easily observe that the Lagrange parameters vector $\tilde{\theta}$ simply corresponds to the expected number of visits to each transient state before reaching the absorbing state, $n$. In other words, $\tilde{\theta} = n$ in Equation (E6).

Finally, expressing the fact that each probability distribution has a given entropy,

$$-\sum_{k \neq n} u_k \sum_{k' \in S(k)} p_{kk'} \log p_{kk'} = H, \quad (E11)$$

allows us to compute the value of $\eta$ in terms of $H$. 

---

261
Appendix E

Computation of
\[ \sum_{l \neq n} \left( \frac{\partial n_l}{\partial p_{kk'}} \right) h_l \]

Let us first compute \( \sum_{l \neq n} \left( \frac{\partial n_l}{\partial p_{kk'}} \right) h_l \) for \( k, k' \neq n \). We have

\[
\sum_{l \neq n} \left( \frac{\partial n_l}{\partial p_{kk'}} \right) h_l = \frac{\partial n^T}{\partial p_{kk'}} h_l = \frac{\partial (n^T e_1)^T}{\partial p_{kk'}} h_l = e_1^T \frac{\partial N}{\partial p_{kk'}} h
\]

where we used \( n = N^T e_1 \) (see Equation (3.5)).

But since \( N = (I - Q)^{-1} \),

\[
\frac{\partial N}{\partial p_{kk'}} = -(I - Q)^{-1} \frac{\partial (I - Q)}{\partial p_{kk'}} (I - Q)^{-1}
\]

\[
= N \frac{\partial Q}{\partial q_{kk'}} N
\]

\[
= N e_k e_k^T N
\]

(F1)

\(262\)
APPENDIX E. COMPUTATION OF $\sum_{l \neq N} (\partial N_L / \partial P_{K'K'}) H_L$

where we used the standard formula for matrix differentiation, $dM^{-1} = -M^{-1}(dM)M^{-1}$.

Replacing (F2) in (F1) provides

$$
\sum_{l \neq n} (\partial n_l / \partial p_{kk'}) h_l = \tilde{e}_k^T N \tilde{e}_k' \tilde{e}_k^T N h
$$

$$
= n_k \tilde{e}_k' \tilde{e}_k^T N h
$$

$$
= n_k \sum_{l \neq n} n_{k'l} h_l
$$

which is the required result.

Now, when $k' = n$ (absorbing node), since the $n_l$ do not depend explicitly on the $p_{kn}$ ($n_l$ only depends on $Q$), $\partial n_l / \partial p_{kn} = 0$ and thus $\sum_{l \neq n} (\partial n_l / \partial p_{kn}) h_l = 0$. 

263
Updating rules for the basic model

PMR1

In this appendix, the EM algorithms allowing to estimate the various parameters of the basic model are detailed.

F.1 A study of the likelihood function

Let us remember the complete log-likelihood of the data,

\[ l = \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \log(P(x_{ki})) + \log(P(y_{kli}|x_{ki})) \right\} \quad (G1) \]

\[ = -\sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \frac{1}{2} \left[ \frac{x_{ki} - q_k}{\sigma_k^x} \right]^2 + \frac{1}{2} \left[ \frac{y_{kli} - (a_l x_{ki} + b_l)}{\sigma_l^y} \right]^2 + \log(\sigma_k^x \sigma_l^y) + \log(2\pi) \right\} \quad (G2) \]

It can easily be shown that the joint probability of \([x_k, y_{kl}]\) is normal, \([x_k, y_{kl}] \sim N(m_{kl}, \Sigma_{kl})\), with mean vector and variance-covariance matrix

\[ m_{kl} = \begin{bmatrix} q_k \\ a_l q_k + b_l \end{bmatrix} \quad \text{and} \quad \Sigma_{kl} = \begin{bmatrix} (\sigma_k^x)^2 & a_l (\sigma_k^x)^2 \\ a_l (\sigma_k^x)^2 & a_l^2 (\sigma_k^x)^2 + (\sigma_l^y)^2 \end{bmatrix} \quad (G3) \]
The inverse of this variance-covariance matrix is
\[
\Sigma^{-1}_{kl} = \begin{bmatrix} 
\frac{a_l^2 (\sigma_k^x)^2 + (\sigma_l^y)^2}{(\sigma_k^x)^2 (\sigma_l^y)^2} & -\frac{a_l}{(\sigma_l^y)^2} \\
-\frac{a_l}{(\sigma_k^x)^2} & \frac{1}{(\sigma_l^y)^2}
\end{bmatrix}
\]  \hfill (G4)

Notice that the marginal distributions are
\[
P(x_k) = \frac{1}{\sqrt{2\pi \sigma_k^x}} \exp\left\{ -\frac{(x_k - q_k)^2}{2(\sigma_k^x)^2} \right\} \hfill (G5)
\]
\[
P(y_{kl}) = \frac{1}{\sqrt{2\pi (a_l^2 (\sigma_k^x)^2 + (\sigma_l^y)^2)}} \exp\left\{ -\frac{(y_{kl} - (a_l q_k + b_l))^2}{2(a_l^2 (\sigma_k^x)^2 + (\sigma_l^y)^2)} \right\} \hfill (G6)
\]

The complete log-likelihood of the data (Equation (G2)) can be rewritten as
\[
l = -\sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in \{k,l\}} \left\{ \frac{1}{2} \begin{bmatrix} x_{ki} - q_k \\
y_{kl} - (a_l q_k + b_l) \end{bmatrix}^T \Sigma^{-1}_{kl} \begin{bmatrix} x_{ki} - q_k \\
y_{kl} - (a_l q_k + b_l) \end{bmatrix} + \log(\sigma_k^x \sigma_l^y) + \log(2\pi) \right\} \hfill (G7)
\]

The first step (the expectation step) in the EM algorithm aims to compute the expectation of the unobserved variables \(x_k\) given the observed variables, \(y_{kl}\), and the current estimate of the parameters, \(\hat{\Theta}\). For the sake of simplicity, the sets of variables \(\{x_k\}, \{y_{kl}\}\) are denoted by \(x\) and \(y\). Moreover, the set of parameters \(\{q_k, \sigma_k^x, a_l, b_l, \sigma_l^y\}\) is denoted by \(\Theta\).

Let us first evaluate \(E_{x|y} [x_k | y, \Theta]\) and \(V_{x|y} [x_k | y, \Theta]\) which will be used in order to compute the EM updates. In order to compute the conditional expectation given \(y\), we need some standard results from applied statistics [115]. Consider a random vector \(w\) which is partitioned into two subvectors such that \(w = [w_1 \ w_2]^T\) and the mean, together with the variance-covariance matrix, are partitioned correspondingly as
\[
\begin{align*}
\mathbf{m} &= \begin{bmatrix} m_1 \\
m_2 \end{bmatrix} \\
\mathbf{\Sigma} &= \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\
\Sigma_{21} & \Sigma_{22} \end{bmatrix}
\end{align*}
\]  \hfill (G8)
F.2. The expectation step

It is well-known [115] that if \( w \) is normal, i.e., \( w \sim \mathcal{N}(\mu, \Sigma) \), then the conditional distribution of \( w_1 \) given \( w_2 \) is also normal \( w_1 | w_2 \sim \mathcal{N}(\mu_1 | w_2, \Sigma_{12} \Sigma_{22}^{-1} (w_2 - \mu_2)) \) with mean vector and variance-covariance matrix

\[
\begin{align*}
\mu_1 | w_2 &= \mu_1 + \Sigma_{12} \Sigma_{22}^{-1} (w_2 - \mu_2) \\
\Sigma_{12} &= \Sigma_{11} - \Sigma_{12} \Sigma_{22}^{-1} \Sigma_{21}
\end{align*}
\] (G9)

This result is used in the next section when computing the expectation step.

F.2 The expectation step

If we apply this result to our problem and use the current estimate of the parameter vector \( \hat{\Theta} \) in order to perform the expectation, we obtain, from Equations (G3) and (G9)

\[
\begin{align*}
\hat{x}_{kl} &= \mathbb{E}_{x|y} \left[ x_k | y_{kl}, \hat{\Theta} \right] = \hat{q}_k + \frac{\hat{a}_l (\hat{\sigma}_k^x)^2}{\hat{a}_l^2 (\hat{\sigma}_k^y)^2 + (\hat{\sigma}_l^y)^2} [y_{kl} - (\hat{a}_l \hat{x}_{kl} + \hat{b}_l)] \\
(\hat{\sigma}_{kl}^x)^2 &= \mathbb{V}_{x|y} \left[ x_k | y_{kl}, \hat{\Theta} \right] = \frac{(\hat{\sigma}_k^x)^2 (\hat{\sigma}_l^y)^2}{\hat{a}_l^2 (\hat{\sigma}_k^y)^2 + (\hat{\sigma}_l^y)^2}
\end{align*}
\] (G11)

where we defined the estimates of the conditional expectation of \( x_k \) and of the conditional variance as \( \hat{x}_{kl} \) and \( (\hat{\sigma}_{kl}^x)^2 \) respectively. These equations provide the expectation and the variance of the unobserved variables given the observed variables.

By dropping the constant term (i.e., \( \log(2\pi) \)) in Equation (G7), taking the expectation of the log-likelihood \( l \), using Equations (G10) and (G11), and going through a little calculus*, we obtain

\[
\begin{align*}
\mathbb{E}_{x|y} \left[ l | y, \hat{\Theta} \right] &= -\frac{1}{2} \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \left[ \frac{y_{kil} - (a_l \hat{x}_{kl} + b_l)}{\sigma_l^y} \right]^2 + \left[ \frac{\hat{x}_{kl} - q_k}{\sigma_k^x} \right]^2 \right. \\
&\left. + \frac{(\hat{\sigma}_{kl}^x)^2 [a_l^2 (\hat{\sigma}_k^x)^2 + (\sigma_l^y)^2]}{(\hat{\sigma}_k^x)^2 (\sigma_l^y)^2} + \log(\sigma_k^x \sigma_l^y)^2 \right\}
\end{align*}
\] (G12)

This expected likelihood has to be maximized with respect to the parameters.

*The computations have been performed with the help of the Mathematica system from Wolfram Research.
F.3 The maximization step

The maximization step consists in maximizing the expectation of the complete likelihood given the observed ratings, \( E_{x|y}[l|y, \Theta] \), in terms of the parameters of the model. For this purpose, we take the derivative of \( E_{x|y}[l|y, \hat{\Theta}] \) (Equation (G12)) and solve the resulting system of equations with respect to the parameters. Furthermore, the \( q_k \) are constrained to sum to zero (normalization, \( \sum_k q_k = 0 \)). Thus, taking the derivative with respect to \( q_k \), \( \sigma_k \), and isolating the parameter yields

\[
q_k = \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } q_k \quad \text{(G13)}
\]

\[
\sigma_k^2 = \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left[ (q_k - \hat{x}_{kli})^2 + (\hat{\sigma}_{kl}^2)^2 \right] \quad \text{(G14)}
\]

For \( a_t \), \( b_t \) and \( \sigma_t^2 \), we obtain

\[
a_t = \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \hat{x}_{kli} (y_{kli} - b_t)}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{x}_{kli}^2 + (\hat{\sigma}_{kl}^2)^2)} \quad \text{(G15)}
\]

\[
b_t = \frac{1}{n_{kl}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (y_{kli} - a_t \hat{x}_{kli}) \quad \text{(G16)}
\]

\[
(\sigma_t^2)^2 = \frac{1}{n_{kl}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (y_{kli} - (a_t \hat{x}_{kli} + b_t))^2 + a_t^2 (\hat{\sigma}_{kl}^2)^2 \right] \quad \text{(G17)}
\]

Normally, this set of equations should be solved with respect to the parameters but, in our case, this is not an easy task because of a strong coupling between the different equations. Therefore, instead of using a standard EM algorithm, we rely on the so-called “one-step-later” algorithm introduced by Green [65, 66]; see also [115]. Instead of solving the system of equations, Green proposed to replace the parameters on the right-side of the equation by their current estimate, that is, by their current estimate rather than the new estimate. This procedure, quite similar to coordinate descent [108], provides the following set of updating
F.3. The maximization step

rules

\[ \hat{q}_k \leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } \hat{q}_k \quad (G18) \]

\[ (\hat{\sigma}_k^x)^2 \leftarrow \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left[ (\hat{q}_k - \hat{x}_{kli})^2 + (\hat{\sigma}_k^x)^2 \right] \quad (G19) \]

\[ \hat{\alpha}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \hat{x}_{kli} (y_{kli} - \hat{b}_l)}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{x}_{kli}^2 + (\hat{\sigma}_k^x)^2)} \quad (G20) \]

\[ \hat{\beta}_l \leftarrow \frac{1}{n_\ast l} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (y_{kli} - \hat{\alpha}_l \hat{x}_{kli}) \quad (G21) \]

\[ (\hat{\sigma}_l^y)^2 \leftarrow \frac{1}{n_\ast l} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (y_{kli} - (\hat{\alpha}_l \hat{x}_{kli} + \hat{b}_l))^2 + \hat{\alpha}_l^2 (\hat{\sigma}_k^x)^2 \right] \quad (G22) \]

This algorithm is much easier to compute. What is lost, however, in comparison with the true EM algorithm, is the guarantee that the method converges, and in particular that the iteration always increase the likelihood. Green [65, 66] did not observe any convergence problem nor did we in our experiments.
Updating rules for the model PMR2 involving truncation

G.1 The expectation step

We consider now a model involving a truncation of the ratings, which therefore lie in the interval \([-c, +c]\). The first step in the EM algorithm involves the expectation of the likelihood, which is a function of the unobserved variables, \(x_k, y_{kl}\), given the observed variables, \(z_{kl}\), and the current estimate of the parameters, \(\hat{\Theta}\). As before, the sets of variables \(\{x_k\}, \{y_{kl}\}, \{z_{kl}\}\) are denoted by \(x, y\) and \(z\); moreover, the set of parameters \(\{q_k, \sigma^2_y, a_l, b_l, \sigma^2_l\}\) is denoted by \(\Theta\).

First, observe that

\[
E_{xy|z} [l | z, \hat{\Theta}] = E_{y|x} \left[ E_{x|yz} [l | y, z, \hat{\Theta}] \bigg| z, \hat{\Theta} \right] \quad (H1)
\]

because \(l\) is independent of \(z\) given \(y\). The computation of \(\bar{l}_y = E_{x|y}[l|y, \hat{\Theta}]\) is exactly the same as for the basic model PRM1 and is not repeated here (see Equation (G12)).

The second expectation, \(E_{y|z}[\bar{l}_y|z, \hat{\Theta}]\), involves the conditional expectation of a set of independent normal-distributed variables \(y\) given the observed truncated variables, \(z\). Truncated variables are common in econometrics and have therefore been widely investigated. For instance, a regression model for which the dependent variable is truncated is called a tobit model (see, e.g., [67]). By
G.1. The expectation step

examining the form of the likelihood $\tilde{L}_y$ in Equation (G12), we immediately see that we have to compute the expectations of the following functions, depending on the variable $y_{kli}$: 

$$[(y_{kli} - (a_l \hat{x}_{kli} + b_l))/\sigma_y]^2 \text{ and } [((\hat{x}_{kli} - q_k)/\sigma_{\hat{x}})^2 \text{. Notice that the } \hat{x}_{kli} \text{ depend on the } y_{kli} \text{ (see Equation (G10)).}

In order to compute these expectations, we make use of a well-known decomposition related to the bias-variance decomposition; see [11] for a similar computation for estimating the parameters of a tobit model with the EM algorithm:

$$E[(\alpha y + \beta)^2] = (\alpha E[y] + \beta)^2 + \alpha^2 E[(y - E[y])^2] \quad (H3)$$

$$= (\alpha E[y] + \beta)^2 + \alpha^2 V[y] \quad (H4)$$

where $V[y]$ is the variance of the random variable $y$. The first term consists in replacing the random variable $y$ by its expectation $E[y]$. This amounts, when computing the expectation of the likelihood in Equation (G12), to simply replace the unobserved values of $y_{kli}$ by their conditional expectations. The second term involves the variance of the random variable, $V[y]$.

Thus, we have to evaluate both $E_{Y|Z} [y_{kli}|z, \Theta]$ and $V_{Y|Z} [y_{kli}|z, \Theta]$; that is, the conditional expectation and variance given the truncated values $z$. Since $y_{kli}$ only depends on $z_{kli}$, the expectations reduce to $E[y_{kli}|z_{kli}, \hat{\Theta}]$ and $V[y_{kli}|z_{kli}, \hat{\Theta}]$.

Now, it is well-known [67, 71] that if a normal random variable $u$ is $u \sim N(\mu, \sigma)$, the expectation and the variance can easily be computed for the two truncation cases. If the truncation is from the left, we have

$$E[u | u \leq -c] = \mu + \sigma \lambda(\gamma)$$

$$V[u | u \leq -c] = \sigma^2 [1 + \gamma \lambda(\gamma) - \lambda^2(\gamma)]$$

with

$$\begin{cases} 
\gamma = \frac{-c - \mu}{\sigma} \\
\lambda(\gamma) = -\frac{\varphi(\gamma)}{\phi(\gamma)} 
\end{cases} \quad (H5)$$

where $\varphi$ and $\phi$ are the normal distribution and the normal cumulative function, as defined in Equation (5.6).
G.1. The expectation step

Now, if the truncation is from the right,

\[
\begin{align*}
E[u | u \geq c] &= \mu + \sigma \lambda(\gamma) \\
V[u | u \geq c] &= \sigma^2 \left[ 1 + \gamma(\lambda(\gamma) - \lambda^2(\gamma)) \right]
\end{align*}
\]

with

\[
\begin{align*}
\gamma &= \frac{c - \mu}{\sigma} \\
\lambda(\gamma) &= \frac{\varphi(\gamma)}{1 - \phi(\gamma)}
\end{align*}
\]  \hspace{1cm} (H6)

In our case, from Equation (G6), \(y_{kl} \sim N(a_l q_k + b_l, s_{kl})\) with \(s_{kl} = (a_l^2 (\sigma^x)^2 + (\sigma^y)^2)^{1/2}\). Thus, for computing the \(E[y]\) term in Equation (H4), we define a new variable, \(\hat{y}_{kli} = E[y_{kli} | z, \hat{\Theta}]\) whose value depends on the three cases:

- **First case**: \(z_{kli} = -c\) (in other words, \(y_{kli} \leq -c\), \(-c < z_{kli} < +c\) (in other words, \(y_{kli} = z_{kli}\)) and \(z_{kli} = +c\) (in other words, \(y_{kli} \geq c\)). Equations (H5)-(H6) yield:

  \[
  \hat{y}_{kli} = \begin{cases} 
  \hat{a}_l \hat{q}_k + \hat{b}_l + \hat{s}_{kl} \lambda(\hat{\gamma}_{kl}) \\
  \left\{ \begin{array}{l} 
  \hat{\gamma}_{kl} = -\frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{s}_{kl}} \\
  \lambda(\hat{\gamma}_{kl}) = -\frac{\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
  \end{array} \right.
  \end{cases}
  \]  \hspace{1cm} (H7)

  First case: \(z_{kli} = -c\):

  \[
  \hat{y}_{kli} = \begin{cases} 
  -c \\
  z_{kli}
  \end{cases}
  \]  \hspace{1cm} (H8)

  **Second case**: \(-c < z_{kli} < +c\):

  \[
  \hat{y}_{kli} = \begin{cases} 
  (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{s}_{kl} \lambda(\hat{\gamma}_{kl}) \\
  \left\{ \begin{array}{l} 
  \hat{\gamma}_{kl} = \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{s}_{kl}} \\
  \lambda(\hat{\gamma}_{kl}) = \frac{\varphi(\hat{\gamma}_{kl})}{1 - \phi(\hat{\gamma}_{kl})}
  \end{array} \right.
  \end{cases}
  \]  \hspace{1cm} (H9)

  **Third case**: \(z_{kli} = +c\):

  \[
  \hat{y}_{kli} = \begin{cases} 
  (\hat{a}_l \hat{q}_k + \hat{b}_l) + \hat{s}_{kl} \lambda(\hat{\gamma}_{kl}) \\
  \left\{ \begin{array}{l} 
  \hat{\gamma}_{kl} = \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{\hat{s}_{kl}} \\
  \lambda(\hat{\gamma}_{kl}) = \frac{\varphi(\hat{\gamma}_{kl})}{1 - \phi(\hat{\gamma}_{kl})}
  \end{array} \right.
  \end{cases}
  \]  \hspace{1cm} (H9)

In the same way, we easily obtain for the variance (i.e., the \(V[y]\) term in Equation (H4)):
G.1. The expectation step

**First case:** \( z_{kli} = -c \):

\[
\hat{V}_{kl}^- = V[y_{kli} | z_{kli} = -c, \hat{\Theta}] = s_{kl}^2 \left[ 1 + \hat{\gamma}_{kl} \lambda(\hat{\gamma}_{kl}) - \lambda^2(\hat{\gamma}_{kl}) \right]
\]

with

\[
\begin{align*}
\hat{\gamma}_{kl} &= -c - (\hat{a}_l \hat{q}_k + \hat{b}_l) \\
\lambda(\hat{\gamma}_{kl}) &= -\frac{\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
\end{align*}
\]

(H10)

**Second case:** \(-c < z_{kli} < +c\): \( V[y_{kli} | -c < z_{kli} < +c, \hat{\Theta}] = 0 \)

**Third case:** \( z_{kli} = +c \):

\[
\hat{V}_{kl}^+ = V[y_{kli} | z_{kli} = +c, \hat{\Theta}] = s_{kl}^2 \left[ 1 + \hat{\gamma}_{kl} \lambda(\hat{\gamma}_{kl}) - \lambda^2(\hat{\gamma}_{kl}) \right]
\]

with

\[
\begin{align*}
\hat{\gamma}_{kl} &= \frac{c - (\hat{a}_l \hat{q}_k + \hat{b}_l)}{s_{kl}} \\
\lambda(\hat{\gamma}_{kl}) &= -\frac{\varphi(\hat{\gamma}_{kl})}{\phi(\hat{\gamma}_{kl})}
\end{align*}
\]

(H11)

Now that the \( E[y] \) and \( V[y] \) terms in Equation (H4) are found, the last term that remains to be computed is \( \alpha \). From Equation (G12), replacing the \( \hat{x}_{kli} \) by their value (Equation (G10)) in the \( \left[ (y_{kli} - (a_l \hat{x}_{kli} + b_l))/\sigma^2 \right]^2 \) term and identifying \( \alpha \) yields

\[
\alpha = \frac{s_{kl}^2 - \hat{a}_l(\hat{\sigma}_k^2 a_l)}{s_{kl}^2 \sigma^2_l}
\]

(H12)

while the \( \left[ (\hat{x}_{kli} - q_k)/\sigma^2_l \right]^2 \) term provides

\[
\alpha = \frac{\hat{a}_l(\hat{\sigma}_k^2)^2}{s_{kl}^2 \sigma^2_l}
\]

(H13)

Therefore, by replacing the values of \( y_{kli} \) by their expected values \( E[y] = \hat{y}_{kli} \) and by adding the extra variance term, \( \alpha^2 V[y] \) in the likelihood function
G.2. The maximization step

(G12), the expected likelihood can be rewritten as

\[
\begin{align*}
\mathbb{E}_{xy \mid z, \Theta} \left[ l \mid z, \hat{\Theta} \right] &= -\frac{1}{2} \sum_{k=1}^{n_x} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left\{ \hat{y}_{kli} - (a_l \hat{x}_{kli} + b_l) \right\}^2 + \left[ \hat{x}_{kli} - q_k \right]^2 \\
&+ \frac{\left( \hat{\sigma}^x_{kl} \right)^2}{\left( \sigma_k^x \right)^2} \left[ a_l^2 \left( \sigma_k^x \right)^2 + (\sigma_l^y)^2 \right] + \log(\sigma_k^x \sigma_l^y)^2 \\
&+ \delta (z_{kli} = -c) \frac{\hat{V}_{kl}^-}{\hat{s}_{kl}^2} \left[ \frac{\hat{a}_l (\hat{\sigma}_k^x)^2}{\sigma_k^x} + \left( \frac{\hat{s}_{kl}^2 - \hat{a}_l (\hat{\sigma}_k^x)^2 a_l}{\sigma_l^y} \right)^2 \right] \\
&+ \delta (z_{kli} = +c) \frac{\hat{V}_{kl}^+}{\hat{s}_{kl}^2} \left[ \frac{\hat{a}_l (\hat{\sigma}_k^x)^2}{\sigma_k^x} + \left( \frac{\hat{s}_{kl}^2 - \hat{a}_l (\hat{\sigma}_k^x)^2 a_l}{\sigma_l^y} \right)^2 \right]
\end{align*}
\]

where \( \hat{x}_{kli} \) and \( \hat{\sigma}^x_{kl} \) were defined in Equations (G10)-(G11):

\[
\begin{align*}
\hat{x}_{kli} &= \hat{q}_k + \frac{\hat{a}_l (\hat{\sigma}_k^x)^2}{\hat{s}_{kl}^2} \left[ \hat{y}_{kli} - (\hat{a}_l \hat{q}_k + \hat{b}_l) \right] \\
\hat{s}_{kl}^2 &= \frac{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2}{\hat{s}_{kl}^2} \\
(\hat{\sigma}^x_{kl})^2 &= \frac{\hat{a}_l^2 (\hat{\sigma}_k^x)^2 + (\hat{\sigma}_l^y)^2}{\hat{s}_{kl}^2} + (\hat{\sigma}_l^y)^2
\end{align*}
\]

The next step aims to maximize the expected likelihood with respect to the parameters.

**G.2 The maximization step**

Taking the derivative with respect to \( q_k \), \( \sigma_k^x \), defining \( \hat{V}_{kl}^- = \delta (z_{kli} = -c) \hat{V}_{kl}^- \) and \( \delta (z_{kli} = +c) \hat{V}_{kl}^+ \), and isolating the parameter yields

\[
\begin{align*}
q_k &= \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } q_k \\
(\sigma_k^x)^2 &= \frac{1}{n_k} \sum_{l=1}^{n_y} \sum_{i \in (k,l)} \left[ (q_k - \hat{x}_{kli})^2 + (\hat{\sigma}^x_{kl})^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_k^x)^4 \hat{V}_{kl}^-}{\hat{s}_{kl}^4} \right]
\end{align*}
\]
G.2. The maximization step

For $a_l$, $b_l$ and $\sigma^y_l$, we obtain

$$
a_l = \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli} (y_{kli} - b_l) + \frac{\hat{a}_l (\hat{\sigma}_k) \hat{\tilde{V}}_{kli}}{s_{kl}^2} \right]}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli}^2 + (\hat{\sigma}_{k|y})^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_k) \hat{\tilde{V}}_{kli}}{s_{kl}^4} \right]} \quad (H20)
$$

$$
b_l = \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - a_l \hat{x}_{kli}) \quad (H21)
$$

$$
(\sigma^y_l)^2 = \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (\hat{y}_{kli} - \left( a_l \hat{x}_{kli} + b_l \right))^2 + a_l^2 (\hat{\sigma}_{k|y})^2 + \hat{\tilde{V}}_{kli} \right] \quad (H22)
$$

As before, we rely on the so-called “one-step-later” algorithm [65, 66], which yields

$$
\hat{q}_k \leftarrow \frac{1}{n_{\bullet k}} \sum_{i=1}^{n_y} \sum_{k \in (k,l)} \hat{x}_{kli}, \text{ and normalize the } \hat{q}_k \quad (H23)
$$

$$
(\hat{\sigma}_k^x)^2 \leftarrow \frac{1}{n_{\bullet k}} \sum_{i=1}^{n_y} \sum_{k \in (k,l)} \left[ (\hat{q}_k - \hat{x}_{kli})^2 + (\hat{\sigma}_{k|y})^2 + \frac{\hat{a}_l (\hat{\sigma}_k) \hat{\tilde{V}}_{kli}}{s_{kl}^4} \right] \quad (H24)
$$

$$
\hat{a}_l \leftarrow \frac{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli} (\hat{y}_{kli} - b_l) + \frac{\hat{a}_l (\hat{\sigma}_k) \hat{\tilde{V}}_{kli}}{s_{kl}^2} \right]}{\sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ \hat{x}_{kli}^2 + (\hat{\sigma}_{k|y})^2 + \frac{\hat{a}_l^2 (\hat{\sigma}_k) \hat{\tilde{V}}_{kli}}{s_{kl}^4} \right]} \quad (H25)
$$

$$
\hat{b}_l \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} (\hat{y}_{kli} - \hat{a}_l \hat{x}_{kli}) \quad (H26)
$$

$$
(\hat{\sigma}_l^y)^2 \leftarrow \frac{1}{n_{\bullet l}} \sum_{k=1}^{n_x} \sum_{i \in (k,l)} \left[ (\hat{y}_{kli} - \left( \hat{a}_l \hat{x}_{kli} + \hat{b}_l \right))^2 + \hat{a}_l^2 (\hat{\sigma}_{k|y})^2 + \frac{(\hat{\sigma}_l^y)^4 \hat{\tilde{V}}_{kli}}{s_{kl}^4} \right] \quad (H27)
$$

274
G.2. The maximization step

Of course, the remarks about the convergence of the method made for model PMR1 can be repeated.
Bibliography


BIBLIOGRAPHY


[33] C. Cappiello, M. Comuzzi, and P. Plebani. On automated generation of

[34] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dy-
namic networks. Proceedings of the International Conference on Software


[37] F. Casati, M.-C. Shan, and D. Georgakopoulos(Eds.). VLDB Journal,

[38] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven in-
formation systems engineering: The tropos project. In Information Sys-
tems (27), 2002.


[40] T. M. Cover and J. A. Thomas. Elements of information theory. John

[41] DAML Services Coalition. Daml-s: Web service description for the se-

278, 1993.

[43] D. de Farias and B. Van Roy. On the existence of fixed points for ap-
proximate value iteration and temporal-difference learning. Journal of

for publication, 2005.

[45] R. Dial. A probabilistic multipath assignment model that obviates path


BIBLIOGRAPHY


BIBLIOGRAPHY


BIBLIOGRAPHY


